Speeding up With Binary Operations

Thomas Schwarz, SJ

Binary Representations

- All data in a modern computer is stored in binary
  - Binary operations can speed up many operations
    - Binary representation of positive numbers uses base two
    - Binary representation of negative numbers uses ones or two compliment
- Simplest way to think about it uses base 16

Binary Representations

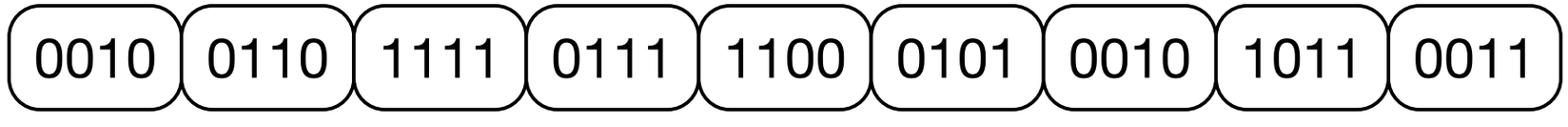
- Example:
  - $3^{21} = 10460353203$

```
>>> b = 3**21
```

  - In hexadecimal:
    - $26f7c52b3_{16}$

```
>>> hex(b)
'0x26f7c52b3'
```

- In binary:
  - 100110111101111100010100101010110011
- Group binary digits in groups of four



Binary Operations

- Shifts:
  - `x << n` : shift x to the left by *n* bits
    - Same effect as multiplying with 2^n
  - `x >> n` : shift x to the right by *n* bits
    - Same effect as floor (= integer) division by 2^n
- But much faster

Binary Operations

- `x | y` : bitwise or of *x* and *y*
- `x & y` : bitwise and of *x* and *y*
- `x ^ y` : bitwise exclusive-or of *x* and *y*

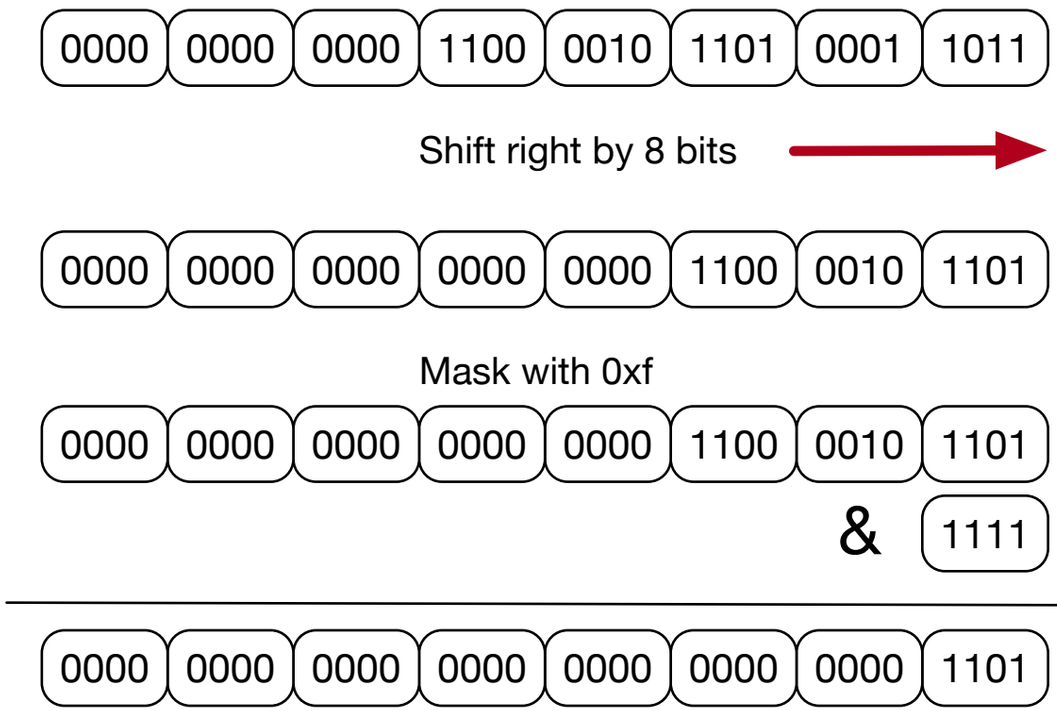
- Bitwise and in conjunction with shifts allow you to isolate areas of bits.
  -

```
(x>>8) & 0xf
```

Binary Operations

- Example: Calculate the 3d hexadecimal from the right

```
>>> c = 0xc2d1b
>>> bin(c)
'0b110000101101000011011'
>>> hex(c>>8)
'0xc2d'
>>> hex(c>>8&0xf)
'0xd'
```



Binary Operations

- We can combine table look-up, masking, and shifting to create more complicated functions
  - Example: Let's (re-)implement the bit-wise inversion of 16b numbers
    - First, we create a table of 4b digits
      - 0000 -> 1111; 0001 -> 1110; 0010 -> 1101; 0011 -> 1100
      - 0100 -> 1011; 0101 -> 1010; 0110 -> 1001; 0111 -> 1000
      - 1000 -> 0111; 1001 -> 0110; 1010 -> 0101; 1011 -> 0100
      - 1100 -> 0011; 1101 -> 0010; 1110 -> 0001; 1111 -> 0000
- Create a list in lieu of a dictionary:
  - translation = [0xf, 0xe, 0xd, 0xc, 0xb, 0xa, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Binary Operations

- We can check that the list works:

```
>>> translation = [0xf, 0xe, 0xd, 0xc, 0xb, 0xa, 9, 8, 7, 6,
5, 4, 3, 2, 1, 0]
>>> for x in range(16):
    print('{:04b}  {:04b}'.format(x, translation[x]))
0000  1111
0001  1110
0010  1101
0011  1100
0100  1011
0101  1010
0110  1001
0111  1000
1000  0111
1001  0110
1010  0101
1011  0100
```

Binary Operations

- Create a loop that moves a half-byte to the right, use the table, and then move back

```
translation = [0xf, 0xe, 0xd, 0xc, 0xb, 0xa,
                9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

def flip(x):
    result = 0
    for i in range(0,4):
        d = translation[(x >> 4*i)&0xf]
        result = result | (d << 4*i)
    return result
```