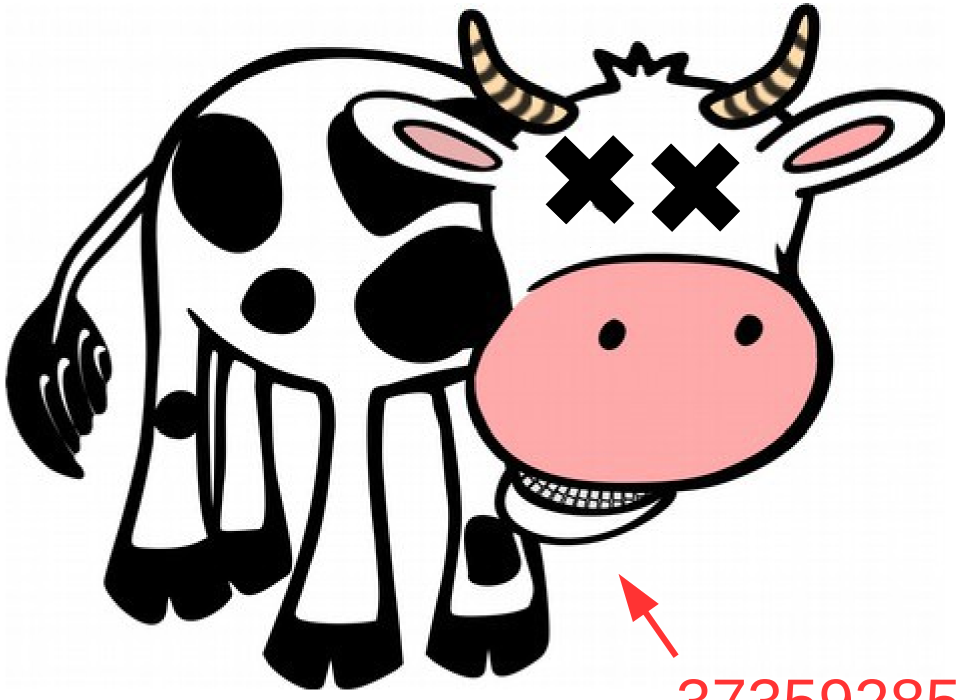
# CS 261
# Fall 2016

Mike Lam, Professor




# Binary Information

# Binary information

- Topics
  - Base conversions (bin/dec/hex)
  - Data sizes
  - Byte ordering
  - Bitwise operations

# Why binary?

- Computers store information in binary encodings
  - 1 bit is the simplest form of information (on / off)
  - Minimizes storage and transmission errors
- To store more complicated information, use more bits
  - However, we need context to understand them
  - Data encodings provide context
  - For the next two weeks, we will study encodings
  - First, let's become comfortable working with binary

# Base conversions

- Binary: bit i represents the value $2^i$
  - Bits typically written from most to least significant (i.e., $2^3\ 2^2\ 2^1\ 2^0$)

$1 = 1 = 0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = [0001]$ 1-**1**=0

$5 = 4 + 1 = 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = [0101]$ 5-**4**=1 1-**1**=0

$11 = 8 + 2 + 1 = 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = [1011]$ 11-**8**=3 3-**2**=1 1-**1**=0

$15 = 8 + 4 + 2 + 1 = 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = [1111]$ 15-**8**=7 7-**4**=3 3-**2**=1 1-**1**=0

**Binary to decimal:**

Add up all the powers of two (memorize powers of two to make this go faster!)

**Decimal to binary:**

Find highest power of two and subtract to find the remainder
Repeat above until the remainder is zero
Every power of two become 1; all other bits are 0

# Base conversions

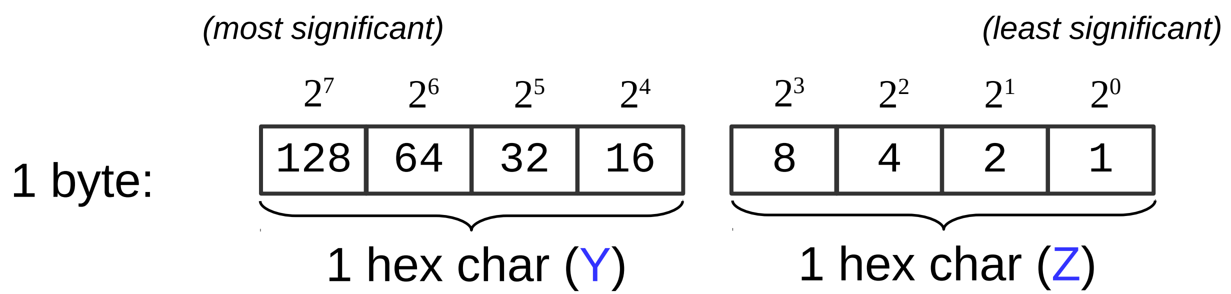
- Hexadecimal: each char represents 4 bits
  - You will/should memorize these eventually

| Dec | Bin | Hex |
|---|---|---|
| 0 | 0000 | 0x0 |
| 1 | 0001 | 0x1 |
| 2 | 0010 | 0x2 |
| 3 | 0011 | 0x3 |
| 4 | 0100 | 0x4 |
| 5 | 0101 | 0x5 |
| 6 | 0110 | 0x6 |
| 7 | 0111 | 0x7 |

| Dec | Bin | Hex |
|---|---|---|
| 8 | 1000 | 0x8 |
| 9 | 1001 | 0x9 |
| 10 | 1010 | 0xA |
| 11 | 1011 | 0xB |
| 12 | 1100 | 0xC |
| 13 | 1101 | 0xD |
| 14 | 1110 | 0xE |
| 15 | 1111 | 0xF |

# Data sizes

- 1 byte = 2 hex chars (= *2 nibbles!*) = 8 bits




Value of byte `0xYZ` is 16Y + Z

- Machine word = size of an address (w)
  - (i.e., the size of a pointer in C)
  - Early computers used 16-bit addresses
    - Could address $2^{16}$ bytes = 64 KB
  - Now 32-bit (4 bytes) or 64-bit (8 bytes)
    - Can address 4GB or 16 EB

| Prefix | Bin | Dec |
|---|---|---|
| Kilo | $2^{10}$ | $\sim 10^3$ |
| Mega | $2^{20}$ | $\sim 10^6$ |
| Giga | $2^{30}$ | $\sim 10^9$ |
| Tera | $2^{40}$ | $\sim 10^{12}$ |
| Peta | $2^{50}$ | $\sim 10^{15}$ |
| Exa | $2^{60}$ | $\sim 10^{18}$ |

# Byte ordering

- Big endian: most significant byte (MSB) first (MSB to LSB)
  - Standard way to write binary/hex (implied with “0x” prefix)
- Little endian: least significant byte first (LSB to MSB)
  - Default byte ordering on most Intel-based machines

```
0x11223344 in big endian:    11 22 33 44
0x11223344 in little endian: 44 33 22 11

Decimal: 1
16-bit big endian:     00000000 00000001  (hex: 00 01)
16-bit little endian:  00000001 00000000  (hex: 01 00)

Decimal: 19 (16+3)
16-bit big endian:     00000000 00010011  (hex: 00 13)
16-bit little endian:  00010011 00000000  (hex: 13 00)

Decimal: 256
16-bit big endian:     00000001 00000000  (hex: 01 00)
16-bit little endian:  00000000 00000001  (hex: 00 01)
```

# Bitwise operations

- Basic bitwise operations
  - & (and)   | (or)   ^ (xor)
- Not boolean algebra!
  - && (and)   || (or)   ! (not)
  - 0 (false)   non-zero (true)
- Important properties:
  - `x & 0 = 0`
  - `x & 1 = x`
  - `x | 0 = x`
  - `x | 1 = 1`
  - `x ^ 0 = x`
  - `x ^ x = 0`
- Commutative:

```
x & y = y & x
x | y = y | x
x ^ y = y ^ x
```

- Associative:

```
(x & y) & z = x & (y & z)
(x | y) | z = x | (y | z)
(x ^ y) ^ z = x ^ (y ^ z)
```

- Distributive:

```
x & (y | z) = (x & y) | (x & z)
x | (y & z) = (x | y) & (x | z)
```

| & | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

AND

| \| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

OR

| ^ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

XOR

# Bitwise operations

- Bitwise complement (~) - “flip the bits”
  - ~0000 = 1111 (~0 = 1) ~1010 = 0101 (~0xA = 0x5)
  - Also called ones' complement (useful on Friday)
- Left shift (<<) and right shift (>>)
  - Left shift: 0110 << 1 = 1100 1 << 3 = binary 1000 = $2^3$ = 8
  - Logical right shift (fill zeroes): 1100 >> 2 = 0011
  - Arithmetic right shift (fill most sig. bit): 1100 >> 2 = 1111

    0100 >> 2 = 0001

**On stu:**

```
 int: 0f00 >> 8 = 000f   (arithmetic)
 int: ff00 >> 8 = ffff
uint: 0f00 >> 8 = 000f   (logical)
uint: ff00 >> 8 = 00ff
```