# CS 31: Intro to Systems
# Binary Representation


Kevin Webb

Swarthmore College

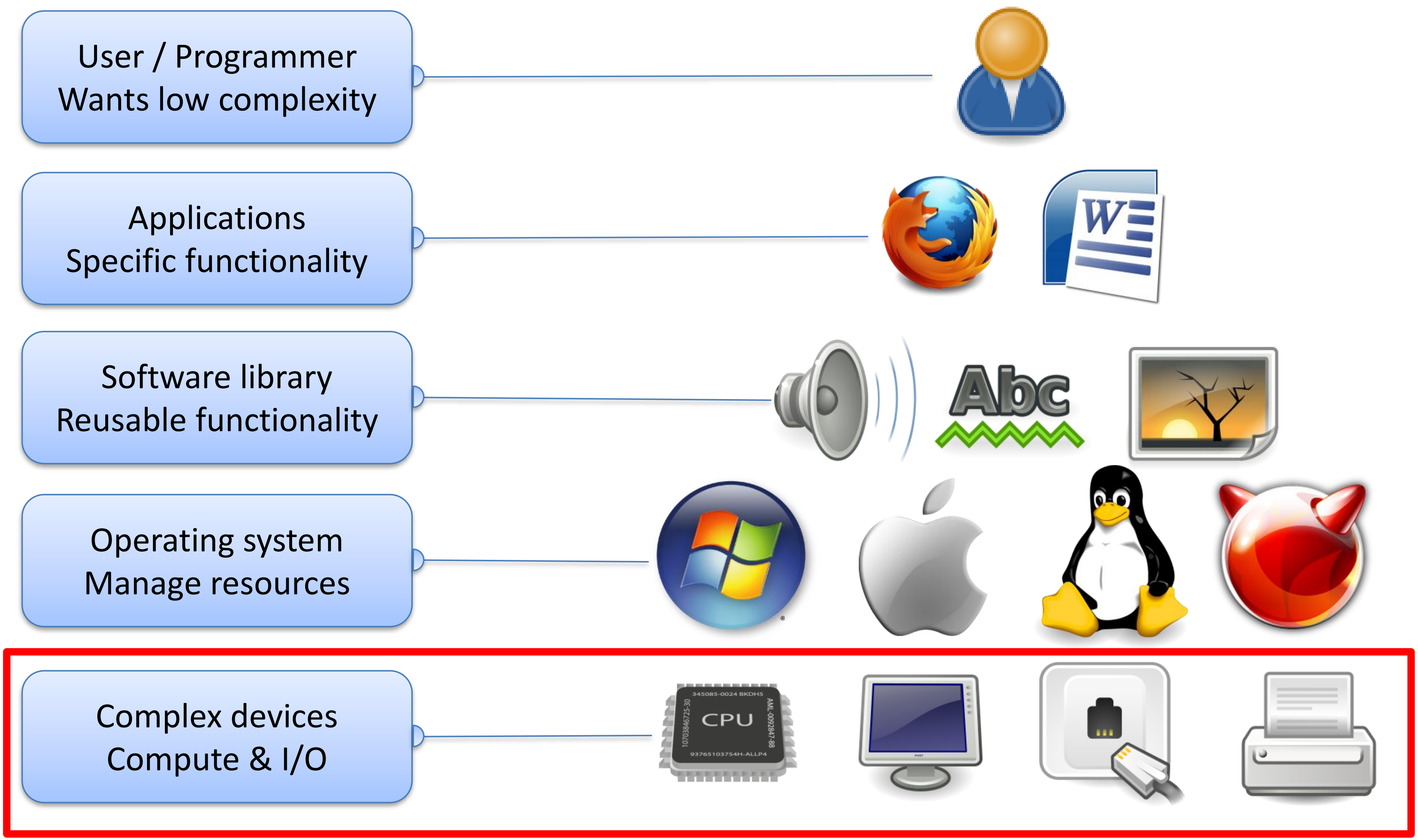

September 6, 2018

# Reading Quiz

# Announcements

- Sign up for Piazza!
- Let me know about exam conflicts!
- Register your clicker (clarification on Piazza)!

# Abstraction



# Today

- Number systems and conversion

- Data types and storage:
  - Sizes
  - Representation
  - Signedness

# Data Storage

- Lots of technologies out there:
  - Magnetic (hard drive, floppy disk)
  - Optical (CD / DVD / Blu-Ray)
  - Electronic (RAM, registers, …)
- Focus on electronic for now
  - We'll see (and build) digital circuits soon
- Relatively easy to differentiate two states
  - Voltage present
  - Voltage absent

# Bits and Bytes

- Bit: a 0 or 1 value (binary)
  - HW represents as two different voltages
    - 1: the presence of voltage (high voltage)
    - 0: the absence of voltage (low voltage)

- Byte: 8 bits, the smallest addressable unit

| | | | | |
|---|---|---|---|---|
| Memory: | 01010101 | 10101010 | 00001111 | … |
| (address) | [0] | [1] | [2] | … |

- Other names:
  - 4 bits: Nibble
  - “Word”: Depends on system, often 4 bytes

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

# Discussion question

- Green border

- Recall the sequence
  - Answer individually (room quiet)
  - Discuss in your group (room loud)
  - Answer as a group
  - Class-wide discussion

# How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

# How many values?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 bit: | 0 | | | | 1 | | | |
| 2 bits: | 0 0 | | 0 1 | | 1 0 | | 1 1 | |
| 3 bits: | 0 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1 0 0 | 1 0 1 | 1 1 0 | 1 1 1 |

| | | | | | |
|---|---|---|---|---|---|
| 4 bits: | 0 0 0 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 16 values |
| | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 | |
| | 1 0 0 0 | 1 0 0 1 | 1 0 1 0 | 1 0 1 1 | |
| | 1 1 0 0 | 1 1 0 1 | 1 1 1 0 | 1 1 1 1 | |

N bits: $2^N$ values

# C types and their (typical!) sizes

- `1 byte:  char, unsigned char`
- `2 bytes: short, unsigned short`
- `4 bytes: int, unsigned int, float`
- `8 bytes: long long, unsigned long long, double`
- `4 or 8 bytes: long, unsigned long`

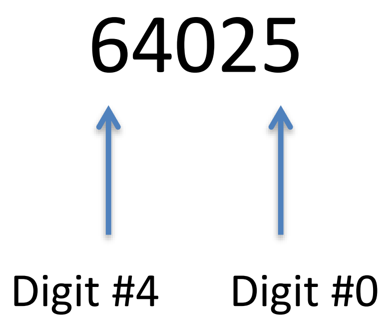
```
unsigned long v1;
short s1;
long long ll;

printf("%lu %lu %lu\n", sizeof(v1), sizeof(s1),
        sizeof(ll));  // prints out number of bytes
```

How do we use this storage space (bits) to represent a value?

# Let’s start with what we know…

- Decimal number system (Base 10)
- Sequence of digits in range [0, 9]



# What is the significance of the $N^{th}$ digit number in this number system? What does it contribute to the overall value?

64025

Digit #4 $d_4$

Digit #0 $d_0$

A. $d_N * 1$
B. $d_N * 10$
C. $d_N * 10^N$
D. $d_N * N^{10}$
E. $d_N * 10^{d_N}$

Consider the meaning of $d_3$ (the value 4) above. What is it contributing to the total value?

# Decimal: Base 10

- Favored by humans…
- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7,8,9}, represents the value:

$$[d_n * 10^n] + [d_{n-1} * 10^{n-1}] + \ldots + [d_2 * 10^2] + [d_1 * 10^1] + [d_0 * 10^0]$$

64025 =

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $6 * 10^4$ | + | $4 * 10^3$ | + | $0 * 10^2$ | + | $2 * 10^1$ | + | $5 * 10^0$ |
| 60000 | + | 4000 | + | 0 | + | 20 | + | 5 |

# Generalizing

- The meaning of a digit depends on its position in a number.
- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ in base b represents the value:

$$[d_n * b^n] + [d_{n-1} * b^{n-1}] + \ldots + [d_2 * b^2] + [d_1 * b^1] + [d_0 * b^0]$$

# Binary: Base 2

- Used by computers to store digital values.
- Indicated by prefixing number with **0b**
- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1}, represents the value:

$$[d_n * 2^n] + [d_{n-1} * 2^{n-1}] + \ldots + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

# What is the value of 0b110101 in decimal?

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1}, represents the value:

$$[d_n * 2^n] + [d_{n-1} * 2^{n-1}] + \ldots + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

A. 26
B. 53
C. 61
D. 106
E. 128

# Other (common) number systems.

- Base 10: decimal
- Base 2: binary

- Base 16: hexadecimal
- Base 8: octal
- Base 64

# Hexadecimal: Base 16

- Indicated by prefixing number with **0x**
- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where d is in {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, represents the value:

$$[d_n * 16^n] + [d_{n-1} * 16^{n-1}] + \ldots + [d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$$

# What is the value of 0x1B7 in decimal?

A. 397
B. 409
C. 419
D. 437
E. 439

$16^2 = 256$

# Important Point…

- You can represent the same value in a variety of number systems / bases.

- It’s **all** stored as binary in the computer.
  - Presence/absence of voltage.

# Other (common) number systems.

- Base 2: How data is stored in hardware.
- Base 8: Used to represent file permissions.
- Base 10: Preferred by people.
- Base 16: Convenient for representing memory addresses.
- Base 64: Commonly used on the Internet, (e.g. email attachments).

# Hexadecimal: Base 16

- Fewer digits to represent same value
  - Same amount of information!
- Like binary, base is power of 2
- Each digit is a “nibble”, or half a byte.

# Each hex digit is a “nibble”

- One hex digit: 16 possible values (0-9, A-F)

- $16 = 2^4$, so each hex digit has exactly four bits worth of information.

- We can map each hex digit to a four-bit binary value. (helps for converting between bases)

# Each hex digit is a “nibble”

Example value: 0x1B7

Four-bit value: 1

Four-bit value: B (decimal 11)

Four-bit value: 7

| | | | |
|---|---|---|---|
| In binary: | 0001 | 1011 | 0111 |
| | 1 | B | 7 |

# Converting Decimal -> Binary

- Two methods:
  - division by two remainder
  - powers of two and subtraction

Method 1: decimal value D, binary result b ($b_i$ is ith digit):

```
i = 0
while (D > 0)
  if D is odd
      set b_i to 1
  if D is even
      set b_i to 0
  i++
  D = D/2
```

Example: Converting 105

```
idea:   D = b          example: D = 105        a0 = 1
```

Method 1: decimal value D, binary result b ($b_i$ is ith digit):

```
i = 0
while (D > 0)
  if D is odd
      set b_i to 1
  if D is even
      set b_i to 0
  i++
  D = D/2
```

Example: Converting 105

```
idea:    D = b          example: D = 105        a0 = 1
         D/2 = b/2               D =  52        a1 = 0
```

Method 1: decimal value D, binary result b ($b_i$ is ith digit):

```
i = 0
while (D > 0)
  if D is odd
      set b_i to 1
  if D is even
      set b_i to 0
  i++
  D = D/2
```

Example: Converting 105

```
idea:     D  = b          example: D = 105        a0 = 1
          D/2 = b/2                D =  52        a1 = 0
          D/2 = b/2                D =  26        a2 = 0
          D/2 = b/2                D =  13        a3 = 1
          D/2 = b/2                D =   6        a4 = 0
          D/2 = b/2                D =   3        a5 = 1
            0 =  0                 D =   1        a6 = 1
                                   D =   0        a7 = 0

                                          105   =  01101001
```

# Method 2

- $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$
- To convert <u>105</u>:
  - Find largest power of two that's less than 105 (64)
  - Subtract 64 (105 – 64 = <u>41</u>), put a 1 in $d_6$
  - Subtract 32 (41 – 32 = <u>9</u>), put a 1 in $d_5$
  - Skip 16, it's larger than 9, put a 0 in $d_4$
  - Subtract 8 (9 – 8 = <u>1</u>), put a 1 in $d_3$
  - Skip 4 and 2, put a 0 in $d_2$ and $d_1$
  - Subtract 1 (1 – 1 = <u>0</u>), put a 1 in $d_0$ (Done)

| __ | __ | __ | __ | __ | __ | __ |
|---|---|---|---|---|---|---|
| $d_6$ | $d_5$ | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ |

# What is the value of 357 in binary?

A. 101100011
B. 101100101
C. 101101001
D. 101110101
E. 110100101

$2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$,
$2^5 = 32$, $2^6 = 64$, $2^7 = 128$, $2^8 = 256$

# So far: Unsigned Integers

- With N bits, can represent values: 0 to $2^n$-1

- We can always add 0's to the front of a number without changing it:

10110= <u>0</u>10110 = <u>00</u>010110 = <u>0000</u>010110

- `1 byte:  char,` <u>`unsigned char`</u>
- `2 bytes: short,` <u>`unsigned short`</u>
- `4 bytes: int,` <u>`unsigned int`</u>`, float`
- `8 bytes: long long,` <u>`unsigned long long`</u>`, double`
- `4 or 8 bytes: long,` <u>`unsigned long`</u>

# Representing Signed Values

- One option (used for floats, NOT integers)
  - Let the first bit represent the sign
  - 0 means positive
  - 1 means negative

- For example:
  - 0101 -> 5
  - 1101 -> -5

- Problem with this scheme?

# Floating Point Representation

1 bit for sign
8 bits for exponent
23 bits for precision

sign | exponent | fraction |

$$value = (-1)^{sign} * 1.fraction * 2^{(exponent-127)}$$

let's just plug in some values and try it out

```
0x40ac49ba: 0 10000001   01011000100100110111010
      sign = 0 exp = 129   fraction = 2902458

            = 1*1.2902458*2^2 = 5.16098
```

I don't  expect you to memorize this

# Up Next: Binary Arithmetic