Bits, Bytes, Ints

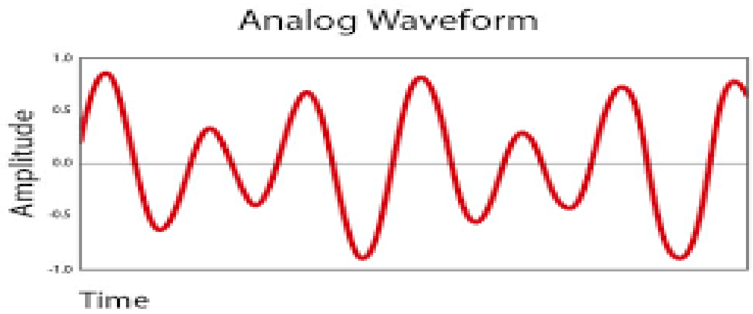
Jinyang Li

Some slides are due to Tiger Wang

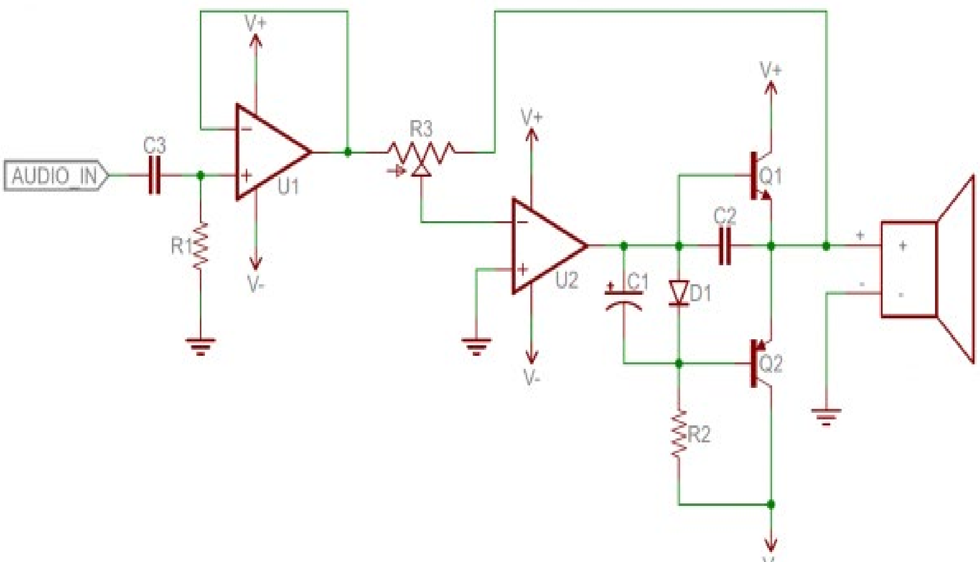
Lesson plan

- How computers represent integers in binary formats
  - Bit, Byte
- How humans represent binary formats
  - Hex notation
- How computers add/subtract integers
- Unsigned vs. signed integer representation

The language of technology has evolved from analog signals…



Analog signals: smooth and continuous

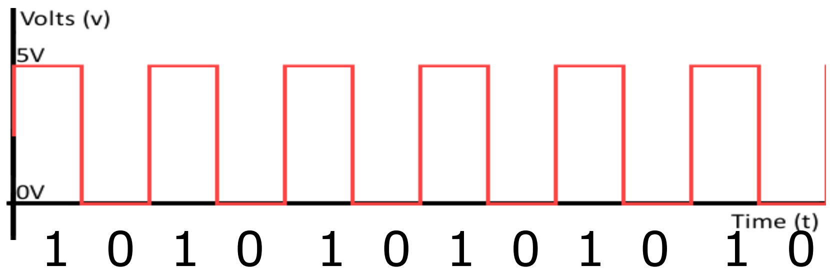


Hard

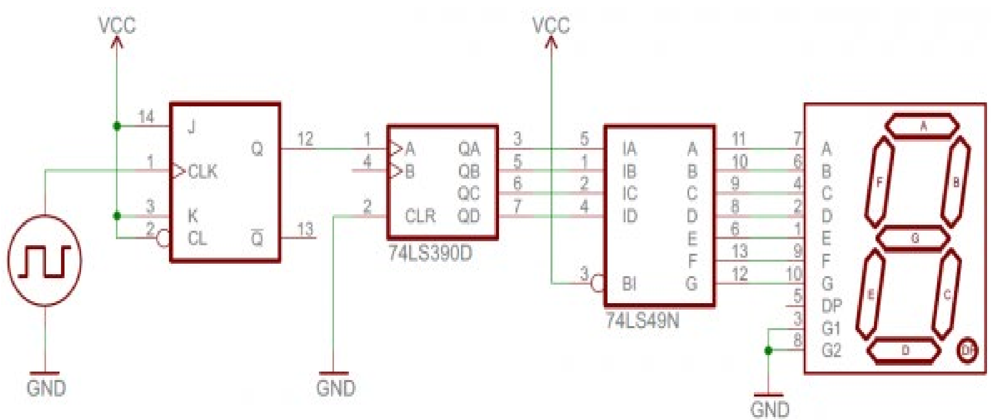
1. **Difficult to design**
2. **Susceptible to noise**

Analog components: resistors, capacitors, inductors, diodes, etc.

… to digital



Digital signals: discrete (0 or 1)



Easier

1. **Easier to design**
2. **Robust to noise**

Digital components: transistors, logic gates …

Using bits to represent everything

Bit = Binary digit, 0 or 1

- A bit is too small to be useful
  - A bit has 2 values; the English alphabet has 26 values (characters)
- Idea: use a group of bits
  - different bit patterns represent different “values”

Question

- How many bit patterns can a group of 2 bits have?

Can be
either 0 or 1

$b_1\ b_0$

All patterns of 2-bits: 00, 01, 10, 11

- How many bit patterns does a group of n bits have?

$\underbrace{b_{n-1}b_{n-2}\ldots b_1b_0}_{\text{n bits}}$

\# of patterns of n-bits: 2^n

Digression: Any self-respecting CS person must memorize powers of 2

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

2^5

2^8

Approximations of powers of 2

$2^{10} = 1024 \approx 10^3$ (Kilo)

$2^{20} \approx 10^{3*2} = 10^6$ (Mega)

$2^{30} \approx 10^{3*3} = 10^9$ (Giga)

$2^{40} \approx 10^{3*4} = 10^{12}$ (Tera)

$2^{50} \approx 10^{3*5} = 10^{15}$ (Peta)

verizon

200 Mbps Speed $\approx 2^{??}$

Stream and download movies, shows and photos.

$39.99[6]

Per Month. With Auto Pay. Plus taxes and equipment charges. 200/200 Mbps

Represent ~~non-negative~~ unsigned integer

bits: b_1b_0

Goal: map each bit pattern to an integer

4 patterns

00 → 0

01 → 1

10 → 2

11 → 3

Range: [0, 3]

Represent unsigned integer

Bit pattern: $b_{n-1}b_{n-2}...b_2b_1b_0$

Range: $[0, 2^n - 1]$

Base-2 representation:

$$b_{n-1}b_{n-2}...b_2b_1b_0 = \sum_{i=0}^{n-1} b_i * 2^i$$

b_i is bit at i-th position (from right to left, starting at i=0)

Examples

Bit pattern: 00000110

Value: $0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2^1+0*2^0 = 6$

Bit pattern: 10000110

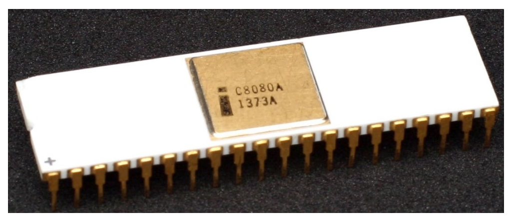
Value:  ?

Byte

- Byte: a fixed size group of bits
  - The term is coined by Werner Buchholz (IBM).
  - Long long ago, different vendors use different byte sizes
- Now: Byte is 8-bit

IBM System/360, 1964

Introduced 8-bit byte



Intel 8080, 1974

Widely adopted

Modern processors

Standardized

Byte

00011010
00001110
11001010
10101110
......

Memory

00111010

CPU

Byte is the smallest unit of information storage, computation and transfer

Integers are represented by 1,2,4, or 8 bytes.

Range of 1-byte non-negative integers: [0, ??]

Bit-pattern of the largest integer?

Range of 4-byte non-negative integers: [0, ??]

Bit-pattern of the largest integer?

Most and least significant bit

MSB: bit position with the largest positional value
LSB: bit position with the smallest positional value

1-byte unsigned int:

10011010

4-byte unsigned int:

01110011 10001101 01010011 11011010

Most significant byte

Least significant byte

Describing bit patterns in a human-readable way

`1-byte int: 10101110`

C program:

```
unsigned int a = 0b10101110;
```

If I ask you to type a 4-byte int, …

Describing a bit pattern: hex notation

`10101110`

Use one (hex) symbol to represent a group of 4 bits

How many hex symbols are needed?

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binary	Hex
1000	8
1001	9
1010	a
1011	b
1100	c
1101	d
1110	e
1111	f

C program:

```
unsigned int a = 0xae;
```

Breakout time!

Breakout exercises

- What's the least significant bit of any even number? `0`
- Write the hex notation of this 2-byte bit pattern: `1111000011000011` `0xf0c2`
- What's the decimal value of unsigned int: `0xff`

`255`

Lesson plan

- How computers represent integers
  - Bit, Byte
- Hex notation
- How computers add/subtract integers
- Unsigned vs. signed integer representation

Unsigned int addition

```
  0 0 0 0 1 0 1 1
+ 0 0 0 0 1 0 1 0
-----------------
  0 0 0 1 0 1 0 1
```

```
  1 0 0 0 1 0 1 1
+ 1 0 0 0 1 0 1 0
-----------------
  0 0 0 1 0 1 0 1
```

Overflow!

Unsigned int subtraction

```
    0 0 0 0 1 1 1 0
-   0 0 0 0 1 0 1 1
-------------------
    0 0 0 0 0 0 1 1
```

```
    0 0 0 0 1 0 1 0
-   0 0 0 0 1 0 1 1
-------------------
    1 1 1 1 1 1 1 1
```

Overflow!

How to represent negative numbers?

Strawman

Most significant bit (MSB) represents the sign

$0\;\underbrace{0\;0\;0\;0\;0\;0\;1}_{\text{magnitude}}{}_2 \longrightarrow 1$

sign

$1\;0\;0\;0\;0\;0\;0\;1_2 \longrightarrow -1$

$$\begin{array}{rr} & 0\;0\;0\;0\;0\;0\;0\;1 \\ + & 1\;0\;0\;0\;0\;0\;0\;1 \\ \hline & 1\;0\;0\;0\;0\;0\;1\;0 \end{array}$$

-2 ???

☹ Need different h/w for signed vs. unsigned computation

Two's complement

Unsigned int

`00010110` $= 0*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

`10010110` $= 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

Signed int

`00010110` $= 0*(-2^7) + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

`10010110` $= 1*(-2^7) + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

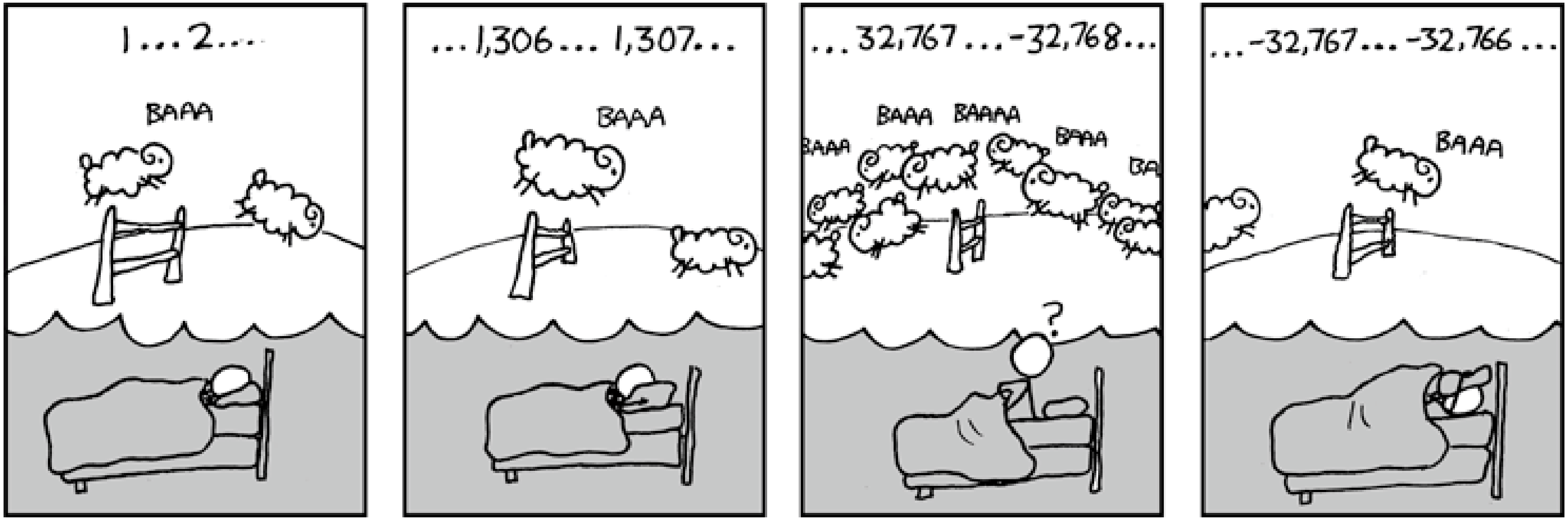
Two's complement

- 1-byte bit pattern → signed int

Bit pattern	value
00000000	0
00000001	1
…	…
01111111	2^7-1 = 127
10000000	-2^7 = -128
10000001	-2^7 +1= -127
…	
11111111	-2^7+(2^7-1) = -1

Two’s complement

- 1-byte bit pattern → signed int



Source: xkcd.com

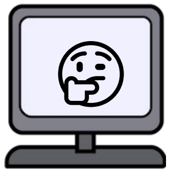
Basic facts of 2's complement

Signed int

Size (bytes)	Bit pattern of smallest	Bit pattern of largest	Range
1	`0x80`	`0x7f`	$[-2^7, 2^7-1]$
2	`0x8000`	`0x7fff`	$[-2^{15}, 2^{15}-1]$
4	`0x80000000`	`0x7fffffff`	$[-2^{31}, 2^{31}-1]$
8	`0x8000000000000000`	`0x7fffffffffffffff`	$[-2^{63}, 2^{63}-1]$

🤔 **Home exercise: make a similar table for unsigned int**

- Negative numbers ←→ MSB=1
- A sequence of 1's (e.g. `0xff`, `0xffffffff`) ←→ -1



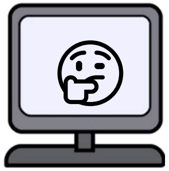
Signed addition on hardware

```
  0 0 0 0 0 0 0 1  (1)10          0 0 0 0 0 0 0 1  (1)10
+ 0 0 0 0 0 0 1 1  (3)10        + 1 0 0 0 0 0 0 1  (-127)10
-------------------------       -------------------------
  0 0 0 0 0 1 0 0  (4)10          1 0 0 0 0 0 1 0  (-126)10
```

This is what 2's complement is designed to accomplish!

```
  0 1 0 0 0 0 0 1  (65)10         1 0 0 0 0 0 0 1  (-127)10
+ 0 1 0 0 0 0 0 0  (64)10       + 1 1 1 1 1 1 1 0  (-2)10
-------------------------       -------------------------
  1 0 0 0 0 0 0 1  (-127)10       0 1 1 1 1 1 1 1  (127)10
```

Overflow!



Signed subtraction on hardware

0 0 0 0 0 0 0 1 $(1)_{10}$	0 0 0 0 0 0 0 1 $(1)_{10}$
- 0 0 0 0 0 0 1 1 $(3)_{10}$	- 1 1 1 1 1 1 1 1 $(-1)_{10}$
1 1 1 1 1 1 1 0 $(-2)_{10}$	0 0 0 0 0 0 1 0 $(2)_{10}$

This is what 2's complement is designed to accomplish!

0 1 1 1 1 1 1 1 $(127)_{10}$	1 0 0 0 0 0 0 0 $(-128)_{10}$
- 1 1 1 1 1 1 1 0 $(-2)_{10}$	- 0 0 0 0 0 0 0 1 $(1)_{10}$
1 0 0 0 0 0 0 1 $(-127)_{10}$	0 1 1 1 1 1 1 1 $(127)_{10}$

Overflow!

Summary

- Integer representation
  - Unsigned (base-2)
  - Signed (2's complement)
- Hex notation
- Operations (e.g. add,subtract) on fixed-width integers can cause overflow