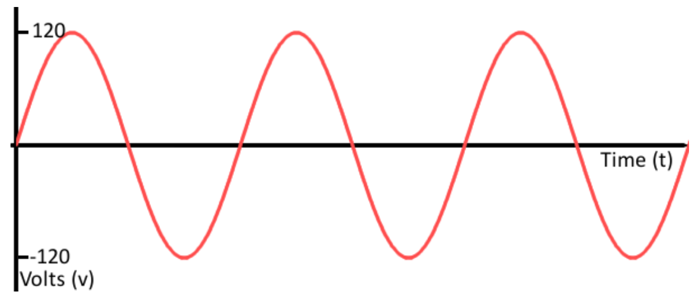
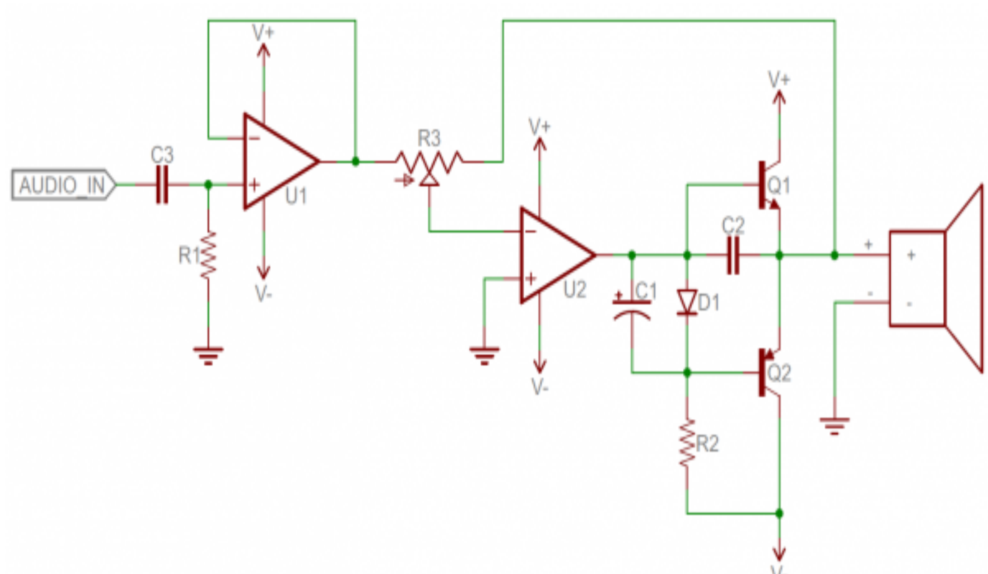
Bits, Bytes, Ints

Jinyang Li

The world has moved away from Analog ...

Analog signals: smooth and continuous

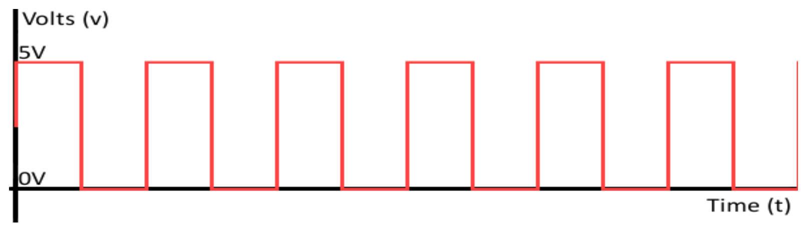


- difficult to design
- susceptible to noise

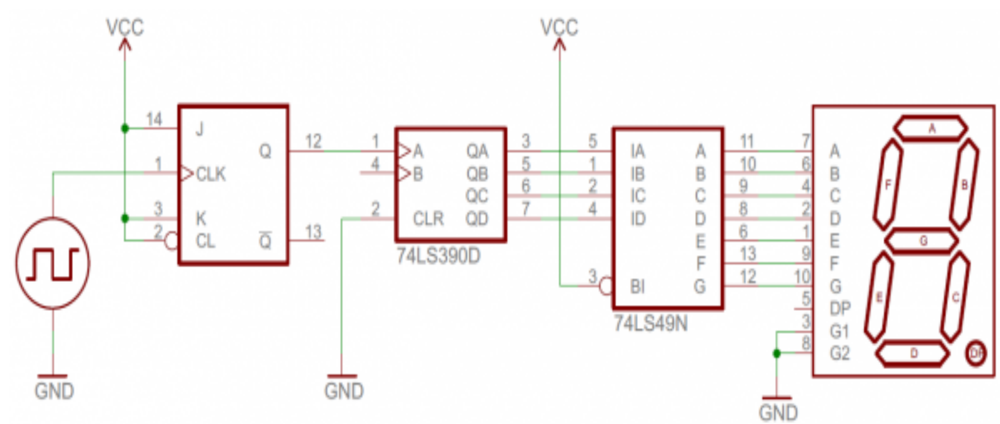
Analog components: resistors, capacitors etc.

... to digital

010011110001010101



Digital signals: discrete (encode sequence of 0s and 1s)



- easier to design
- robust to noise

Digital components: transistors, logic gates, microcontrollers

How to turn everything into 0s and 1s

- The decimal scheme (base 10):

1 2 3

$$x = \sum x_i * 10^i$$

100 10 1

10^2 10^1 10^0

the digit [0,1,...,9] at i-th position

From base-10 to base-b

- Base-b representation of x

$$x = \sum x_i * 10^i$$

the digit [0,1,...,9]
at i-th position

$$x = \sum x_i * b^i$$

the digit [0,1,...,b-1]
at i-th position

12 in base 10 is : 12

12 in base 8 is : 14

base-2 (binary), base-8 (oct), base-16(hex)

- 12 in base 2 : 1100_2
- 12 in base 8 : 14_8
- 12 in base 16: c_{16}
  - hex digits: 0-9, a, b, c, d, e, f

11011100 11011100

3 3 4 d c

Computer scientists' trivia

- The first ten powers of 2 numbers

  1 2 4 8 16 32 64 128 256 512 1024

- 2^10 = kilio $\approx 10^3$
- 2^20 = mega $\approx 10^6$
- 2 ^30 =giga $\approx 10^9$
- 2 ^40 = tera $\approx 10^{12}$

Computer Scientists' trivia

- What's the number 1111 1111 ?
- 1111 1111 = 2^7 + 2^6 + 2^5 + 2^4 + … + 2^1 + 1 = 2^8 – 1 = 255
- What's 1111 1111 in Hex?
- 0xff
- What's the number 1000 0000?
- 2^7=128
- What's the largest 8-bit number?
- 255

How do computers do arithmetic?

- CPU has circuitry to compute on numbers of serveral fixed sizes:
  - 1, 2, 4, 8 bytes
- Byte is the smallest addressable unit
  - 1 byte = 8 bits

Unsigned addition

$$
\begin{array}{r}
0\,0\,0\,0\,0\,0\,1\,1 \\
+\quad 0\,0\,0\,0\,0\,1\,1\,0 \\
\hline
0\,0\,0\,0\,1\,0\,0\,1
\end{array}
$$

Unsigned addition (overflow)

$$
\begin{array}{r}
1\,0\,0\,0\,0\,0\,0\,0 \\
+ \quad 1\,0\,0\,0\,0\,0\,0\,0 \\
\hline
\boxed{1}\;0\,0\,0\,0\,0\,0\,0\,0
\end{array}
$$

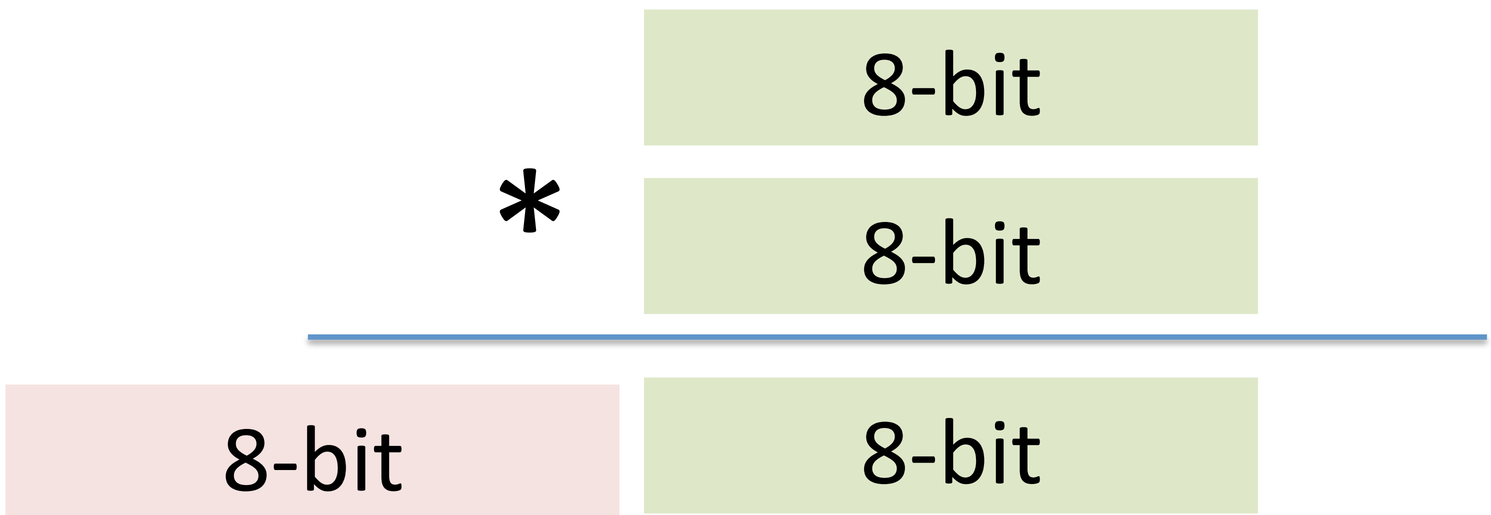
Unsigned subtraction

$$
\begin{array}{rr}
 & 1\,0\,0\,0\,0\,0\,0\,0 \\
- & 0\,0\,0\,0\,0\,0\,0\,1 \\
\hline
 & 0\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

Unsigned subtraction (underflow)

	0 0 0 0 0 0 0 1
-	1 0 0 0 0 0 0 0
-1	1 0 0 0 0 0 0 1

Unsigned multiplication



How many bits required to represent x*y?

How to represent negative numbers?: the naive approach

- Let the most significant bit (MSB) represent the sign
  - 0000 0001 → 1
  - 1000 0001 → -1
- Why not naive representation?

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

☹ CPU circuitry has to be different for signed/unsigned arithematics

Negative numbers: 2's complement

$$x = \sum x_i * 2^i$$

If x_i is MSB and is 1, multiply component by -1

- 0000 1000
- ... + 1*2^3 + ... = 8
- What is 1000 0001 ?
- -1* 2^7 + ... + 1*2^0 = -127

The numerical range

- Range of 1-byte unsigned?

  [0, 2^8-1]

  1111 1111

- Range of 1-byte signed?

  [-2^7, 2^7-1]

  1000 0000

  0111 1111

- Range of w-bit signed?

  [-2^(w-1), 2^(w-1)-1]

A trick to find 2's complement quickly

- Given x = 0000 0011, what's –x in 2's complement?

0000 0011 → 1111 1100 → 1111 1101

Flip all bits

Add 1

Why does the trick work?

$$x + \overbrace{\cancel{\text{x\_with\_bits\_flipped}\ +1}}^{-x} = \overbrace{\cancel{-1\ +1}}^{0}\text{uence of 1’s}$$

What does a sequence of 1s represent in 2’s complement?

2's complement arithmetic

- Same hardware circuitry as unsigned numbers!

	0 0 0 0 0 0 1 1	3	3
+	1 1 1 1 1 0 1 1	251	-5
	1 1 1 1 1 1 1 0	254	-2

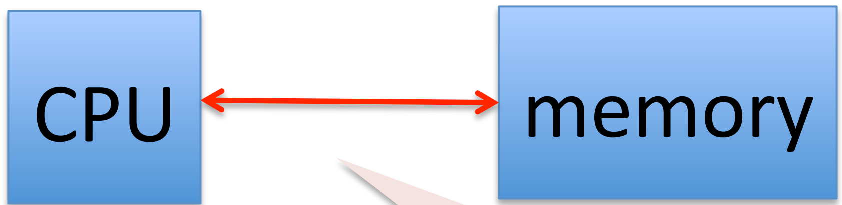
2’s complement (overflow)

	0 1 0 0 0 0 0 0	64
+	0 1 0 0 0 0 0 0	64
	1 0 0 0 0 0 0 0	-128

2’s complement (overflow)

	1 0 0 0 0 0 0 1	-127
+	1 0 0 0 0 0 0 1	-127
1	0 0 0 0 0 0 1 0	2

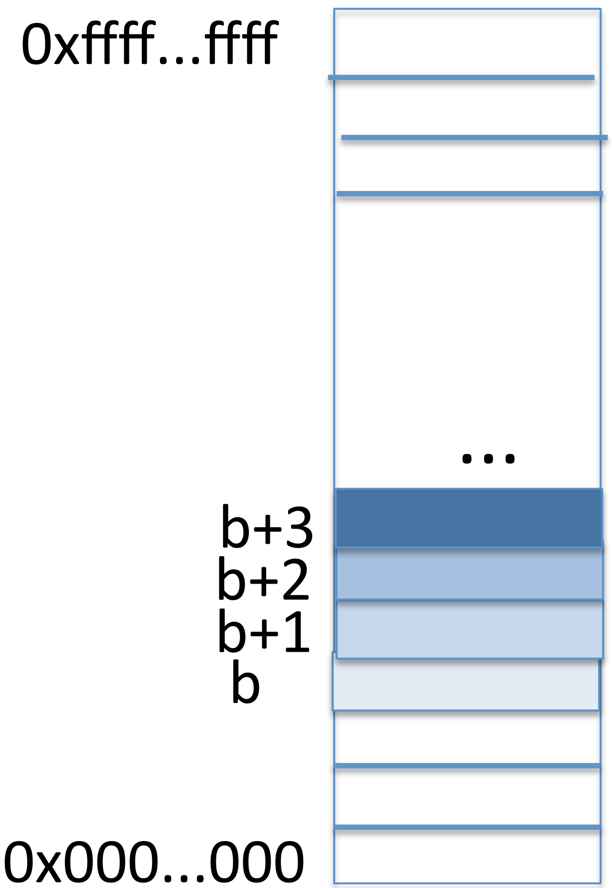
Byte-ordering



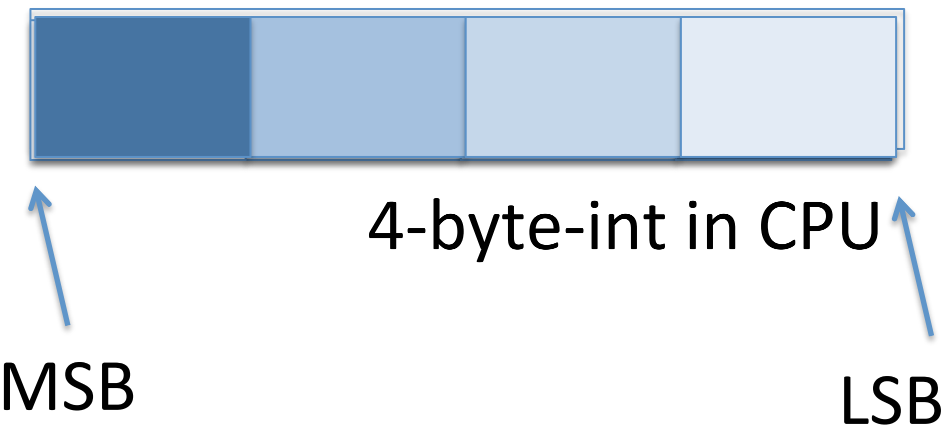
CPU must load and store 1, 2, 4, or 8-bytes from memory as integers

Byte-ordering

- Conceptually, memory is a big array, addressable at each byte



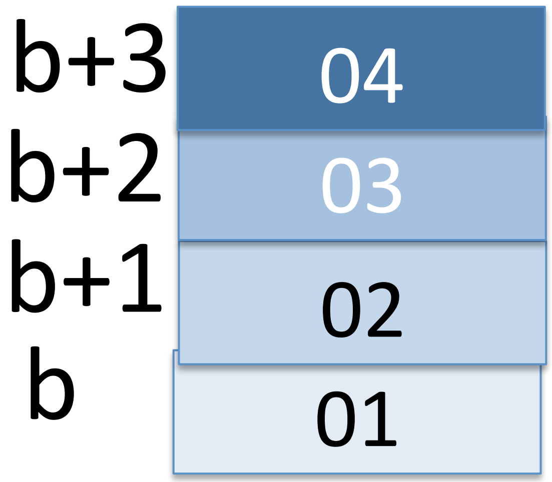
Load a 4-byte integer at address b



Big Endian:
most significant byte in smallest addr

Little Endian:
least significant byte in smallest addr

Big Endian / Small Endian



Big Endian: 0x01020304

Little Endian: 0x04030201

- Intel architecture (laptops, server machines) adheres to Little Endian
- ARM architecture >v3 (cellphones, ipads) are big endian