Introduction

Digital logic and information representation review
Reference: Russell chapter 1

Nuno Alves (nalves@wne.edu), College of Engineering

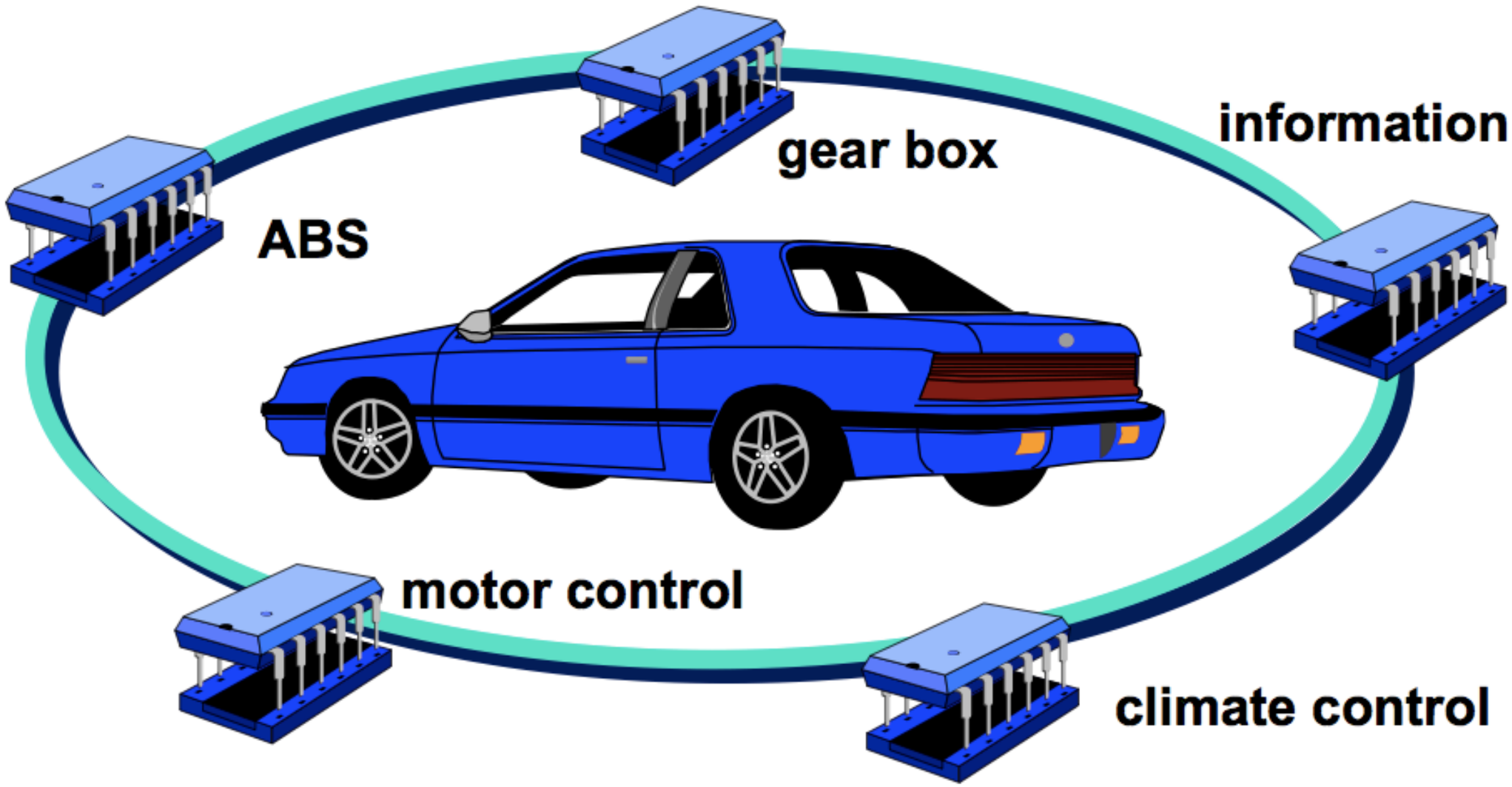
What in an embedded system?

- An embedded system is an electronic system that contains at least one controlling device, i.e. "the brain", but in such a way that it is hidden from the end user.

- That is, the controller is embedded so far in the system that usually users don't realize its presence.

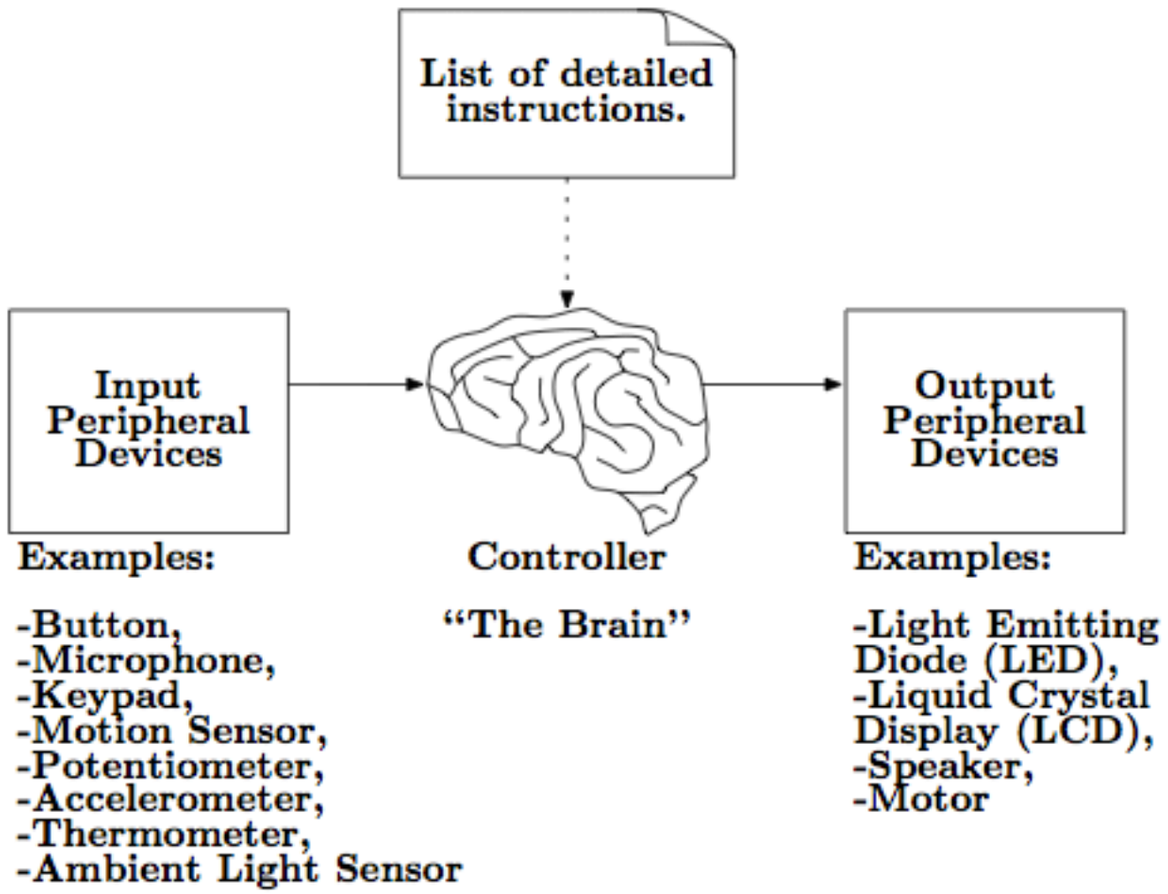
- Examples of devices containing embedded systems:

Embedded systems are everywhere



We trust them too much!
What happens if they fail???

Conceptual view of an embedded system



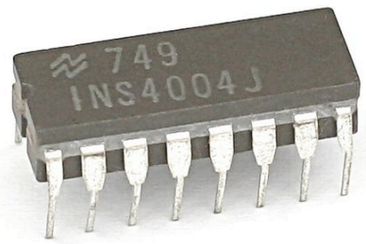
- All embedded systems have some common features.
- Every system contains some input and output elements in order to interact with the environment
- There is something that controls the behavior of the entire system.

Intel 1969

- Transistor and lithography methods are understood
- Intel is just a another custom chip start-up
- Every implementation required a new layout

Genesis: Intel 4004

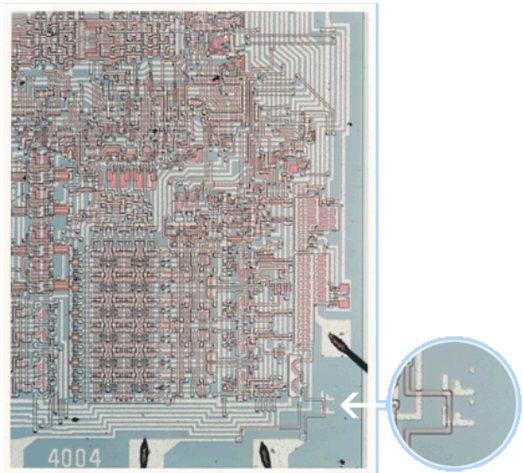
- A Japanese company wanted a custom chip for a calculator
- Design could be re-programmed for other products
- Intel 4004
- 1st complete CPU (on a chip)
- 1st mainstream µProcessor
- 740 kHz, 2000 Transistors



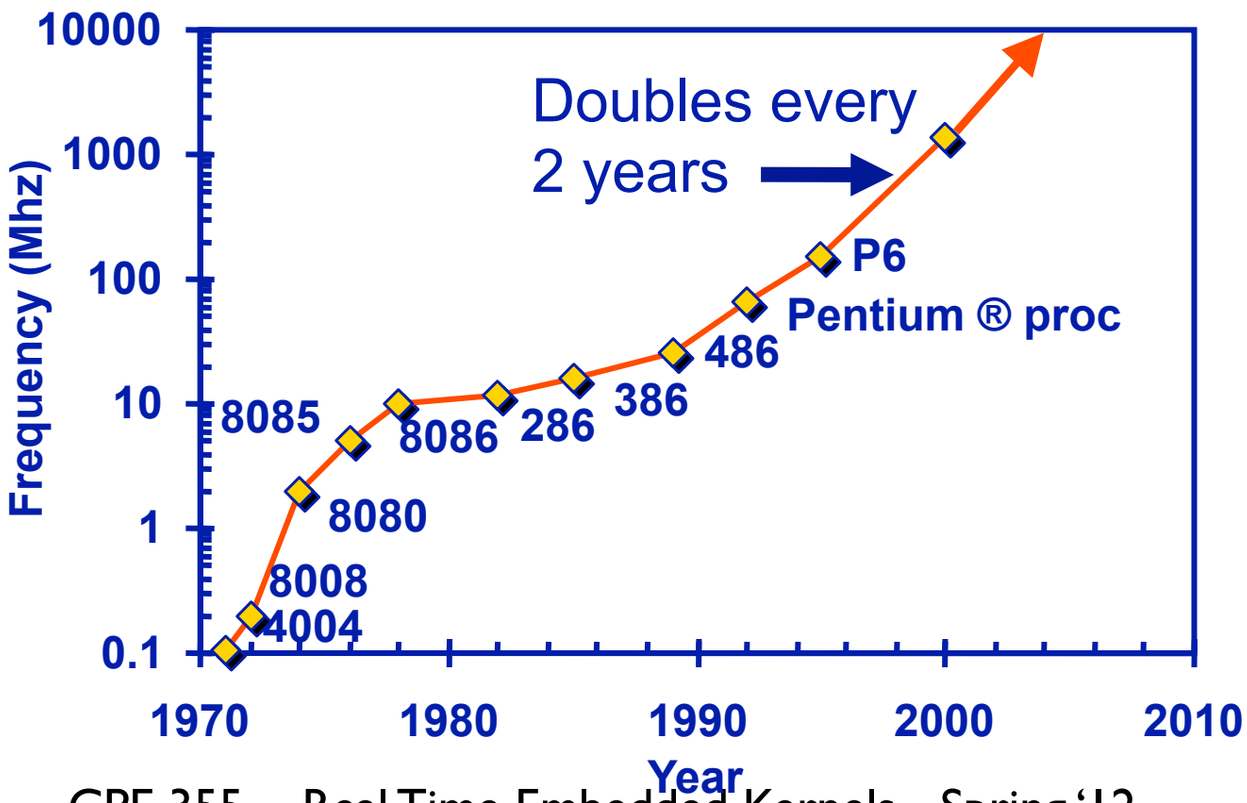
EBay ~$400-$800!!

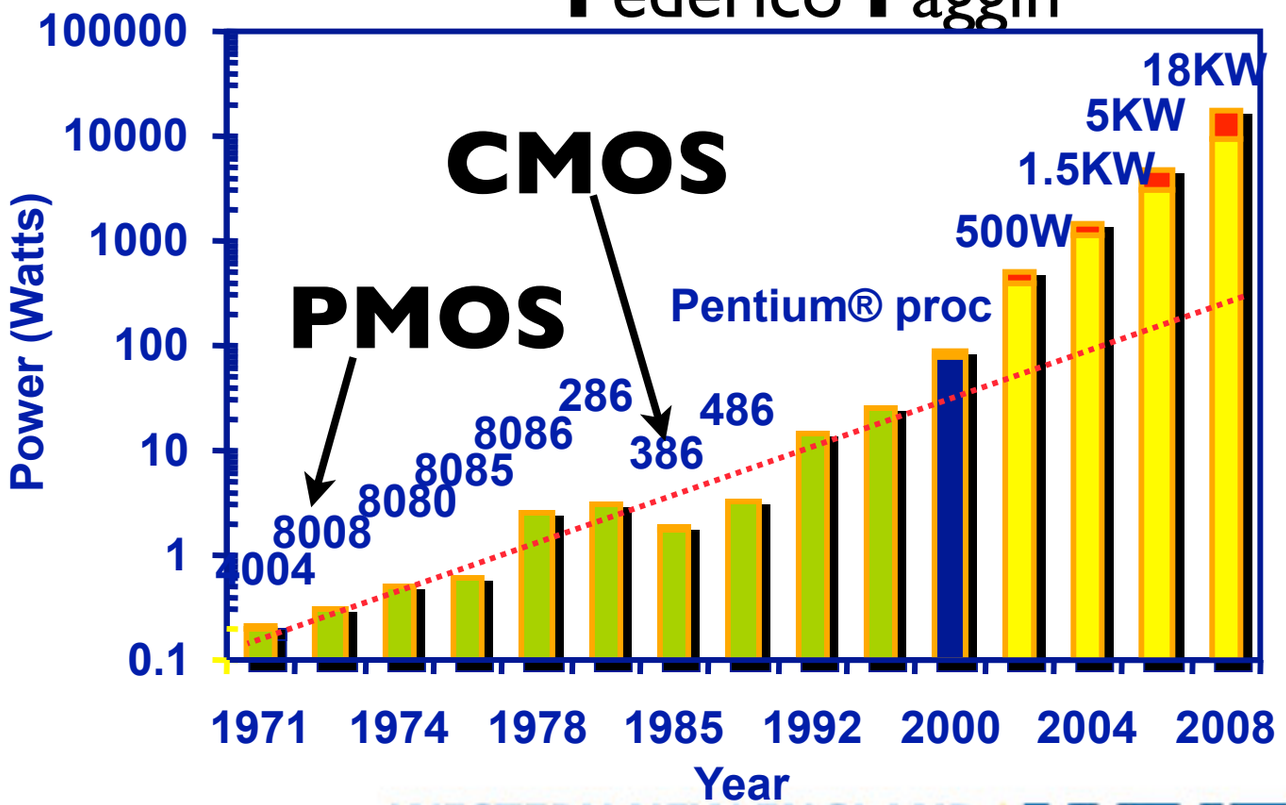
The rest is history

- Busicom sold 100k units
- Intel bought back the rights to the chip
- 4004 schematics and manual are online!

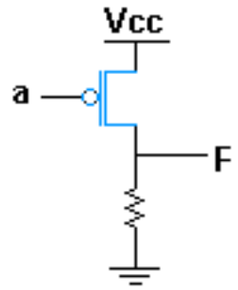


Federico Faggin

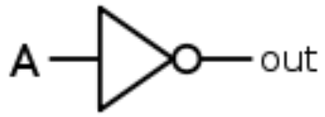


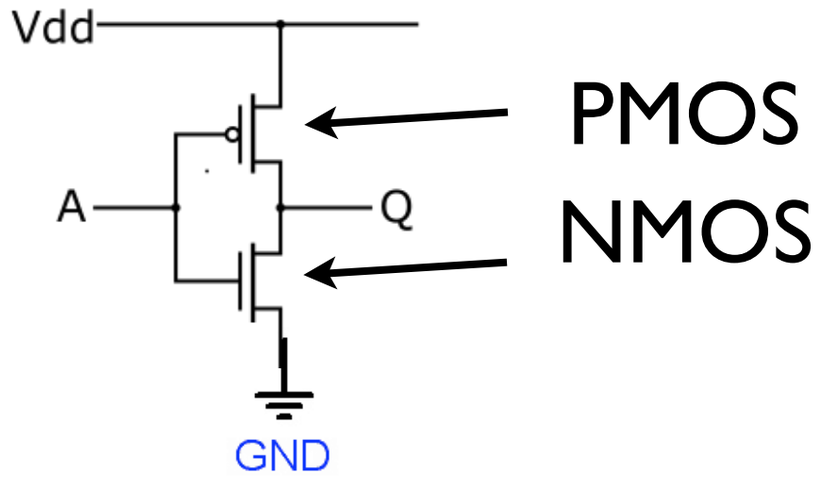


Brief deviation into CMOS and PMOS



PMOS
NOT gate





PMOS Technology

If **a** is LOW, the PMOS transistor is open, therefore current will go into both F and also to GND, which means a lot of power will be "wasted" and dissipated into heat.

CMOS Technology

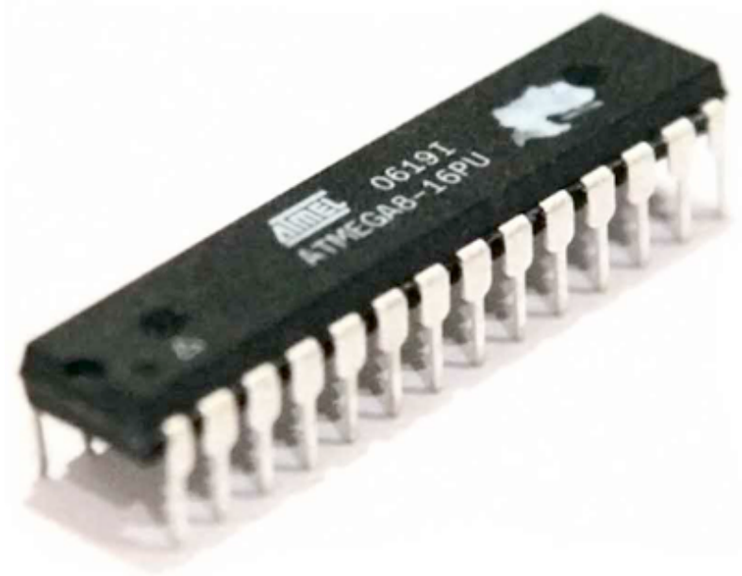
If **a** is LOW, the PMOS transistor is open, but the NMOS transistor is closed, which means most of the current will go into F.

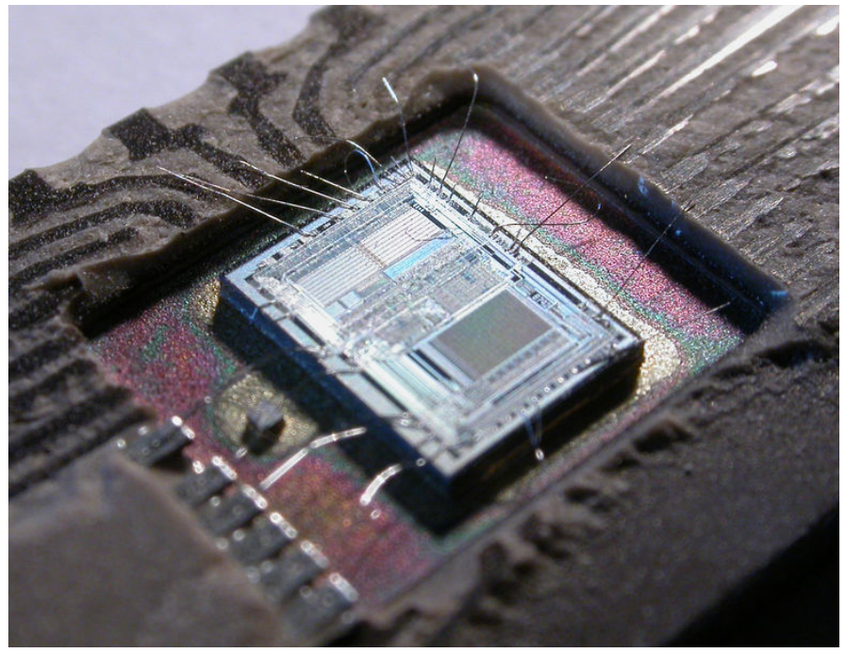
Micro-controllers

- Certain applications do not need big powerful chips.
- Micro-processor vs micro-controller: micro-controller is used in embedded devices and is normally:
  - Is cheaper
  - Is smaller
  - Has internal programable memory
  - Has lower clock frequencies
  - Consumes less power

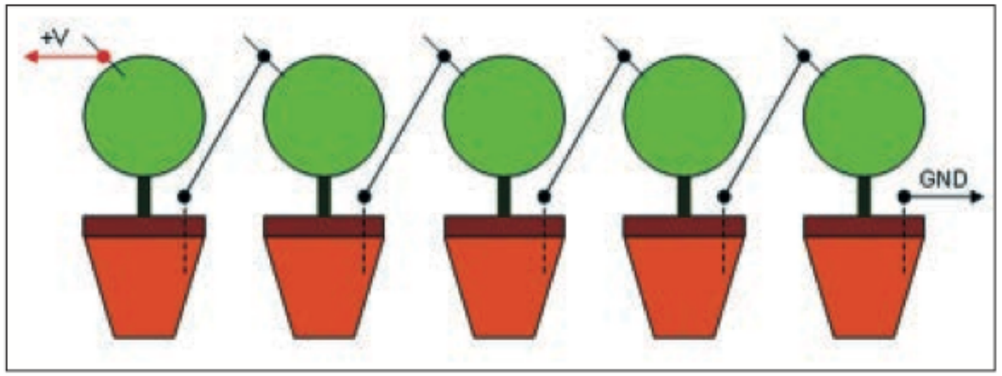
The controller of an embedded system

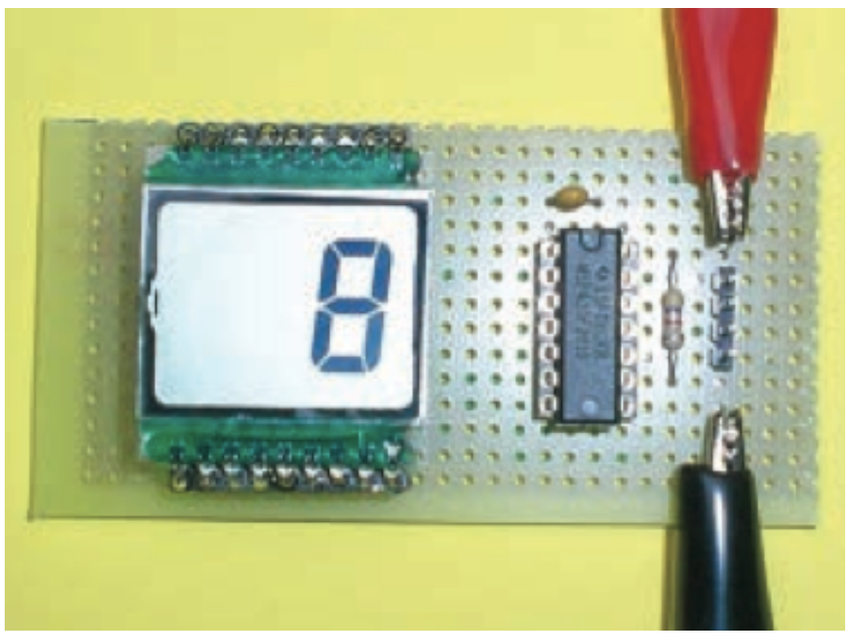
- The controller of an embedded system can be a micro-controller or a micro-processor.
- These two terms greatly overlap these days...
- Micro-controller: single integrated circuit containing a processor core, memory, and programmable input/ output peripherals. Program memory in the form of NOR flash is also often included on chip, as well as a typically small amount of RAM.





Low power trend

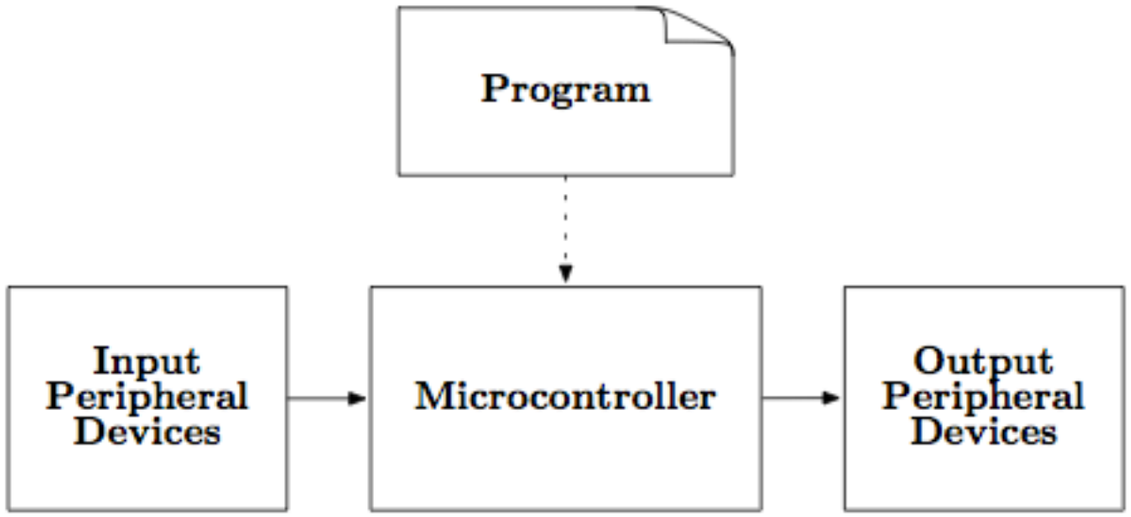




- There is a voltage difference between a branch of a plant and the soil.
- A single plant can supply 0.8 microwatts, at 400 mV.
- Five plants are enough to power an ultra-low power MSP430F2013 from TI

"Flower Power" - Nuts and Volts Magazine, September 2010.

Embedded system block diagram using a micro-controller



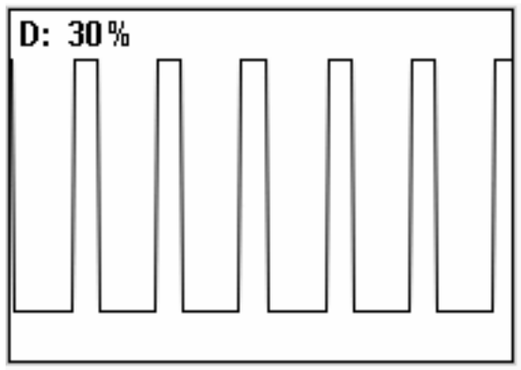
- The program is a list of instructions that will analyze the inputs and write an output.
- The program exists in the system in some form of memory device. Then, the processor has the ability to access the program, execute individual instructions from the list, pick the next instruction to execute, and continue through the entire list until it is finished.

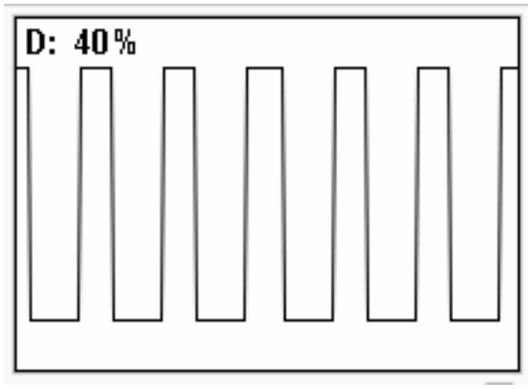
Some words you must know

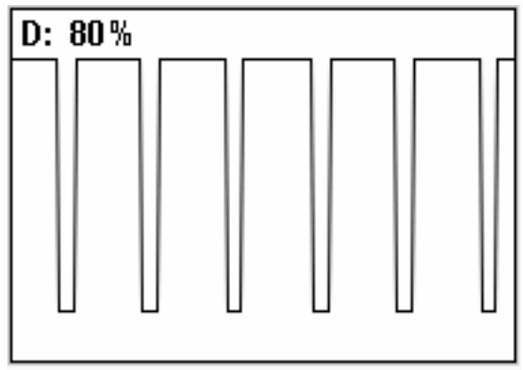
- A processor is just a **synchronous** digital circuit.
- It is an Application Specific Integrated Circuit (**ASIC**), a very large digital circuit entirely fabricated on a single chip.
- Our processor needs DC power (**VDD**), ground (**GND**) and a clock signal (**CLK**).
- A clock is a periodic square-wave signal, usually with a 50% **duty cycle**, i.e., logic '1' 50% of the time.
- The **signal transitions** of the clock, i.e., the rising and falling edges, are used as signal events so that all system components will perform actions at the same time.

Duty cycle

- Is just the percentage of time a signal is in the logic one state (HIGH).







$$D = \frac{\tau}{T}$$

D is the duty cycle.
τ is the duration that the function is active.
T is the period of the function.

Role of CPU

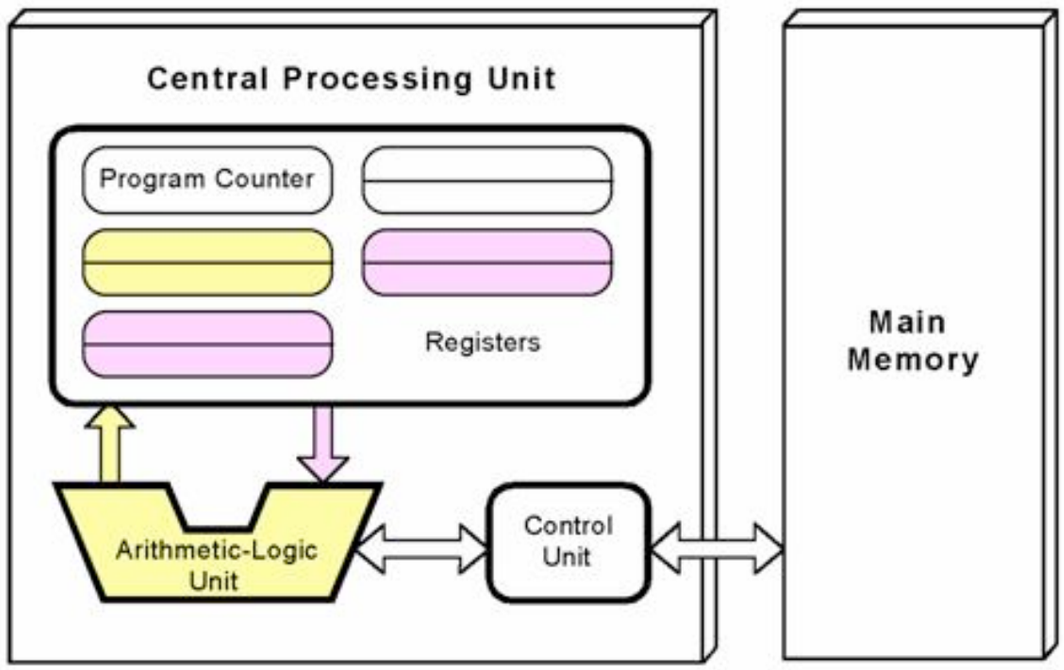
- A processor contains a **custom instruction decoder** sub-circuit that takes as input some single "instruction" and outputs a signal or set of signals that go to other circuit sub-systems within the micro-controller, telling them to perform some function.

- This is the heart of the processor, and it is the bulk of the Central Processing Unit (**CPU**).

- Because this is just a custom decoder circuit, instructions valid for processor A are likely completely different for processor B. That is, the machine language is unique for every processor.

Role of ALU

- Arithmetic/Logic Unit (**ALU**) is responsible for performing all of the standard operations such as addition, subtraction, multiplication, etc.
- Each operation takes in its operands from values stored in CPU registers, which are small groups of memory used by the ALU and CPU.



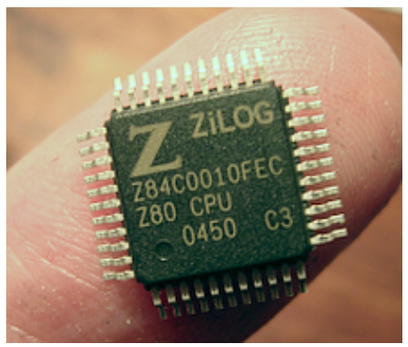
Role of memory

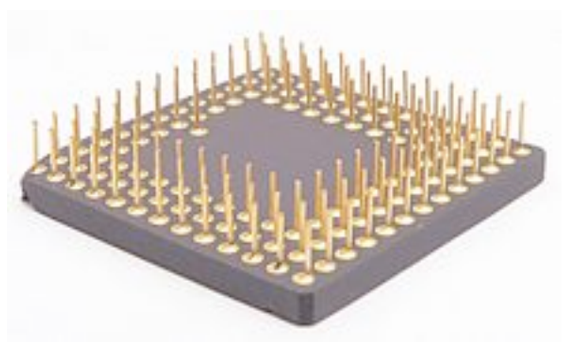
- Finally, every processor has access to two conceptual banks of memory.
- **Non-volatile** memory holds the program, kind of like a textbook
- **Volatile** memory provides the CPU the freedom to run the program, kind of like scratch paper.

Microprocessor characteristics

- Each micro-processor has particular characteristics
- What are **power supply requirements**, **maximum clock frequency** which determines how fast a single instruction can be executed, the **machine language**, the capabilities of the ALU, the number and size of the CPU registers, how instructions interact with the registers, how much memory is available, etc.
- All relevant processor information is found in a document provided by the manufacturer called a data sheet.
- It also includes physical characteristics including the package dimensions and operating properties such as **temperature effects**.

There are several packaging types





Type	Dual Inline Package DIP (70's)	Quad Flat Package QFP (80's)	Pin Grid Array PGA (90's)	Ball Grid Array BGA (00's)
+	- Easy to solder, handle and replace - Extremely mature technology (cheap)	- More available I/O pins than DIP	- More available I/O pins than QFP - Often mounted with through hole methods	- High density - Good heat conduction - Low inductance
-	- Low pin density - Signal propagates "slowly" through pins	- No socketing or hole mounting (only soldering)	- Long leads means loss of signal integrity	- Expensive testing equipment - Unreliable test sockets

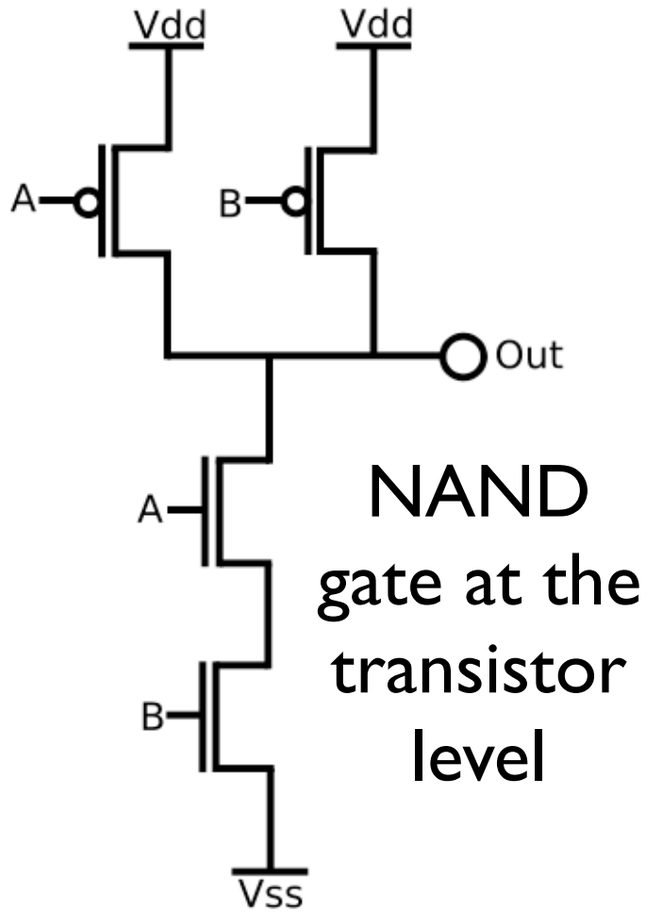
Logic gates

Basic Gates

1) These are the basic gates we will be dealing with:

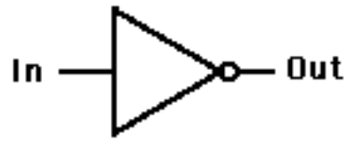
- AND, NAND
- BUF, NOT
- OR, NOR
- XOR, XNOR

2) While you should know how each logic gate is implemented at the transistor level, in this course we are mainly working at the register-transfer level (RTL).



NAND gate at the transistor level

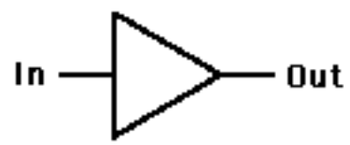
AND, OR, NOT and BUF



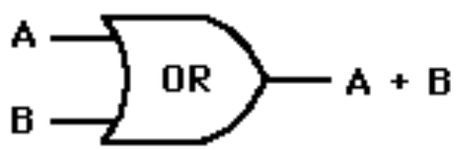
In	Out
0	1
1	0

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

The AND operation will be signified by AB or A·B. Other common mathematical notations for it are $A \wedge B$ and $A \cap B$, called the intersection of A and B.



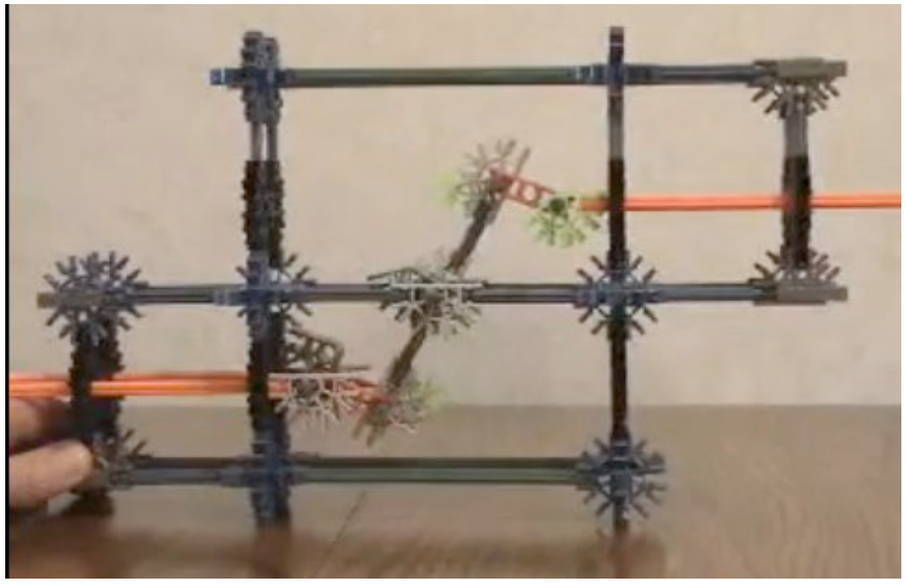
In	Out
0	0
1	1

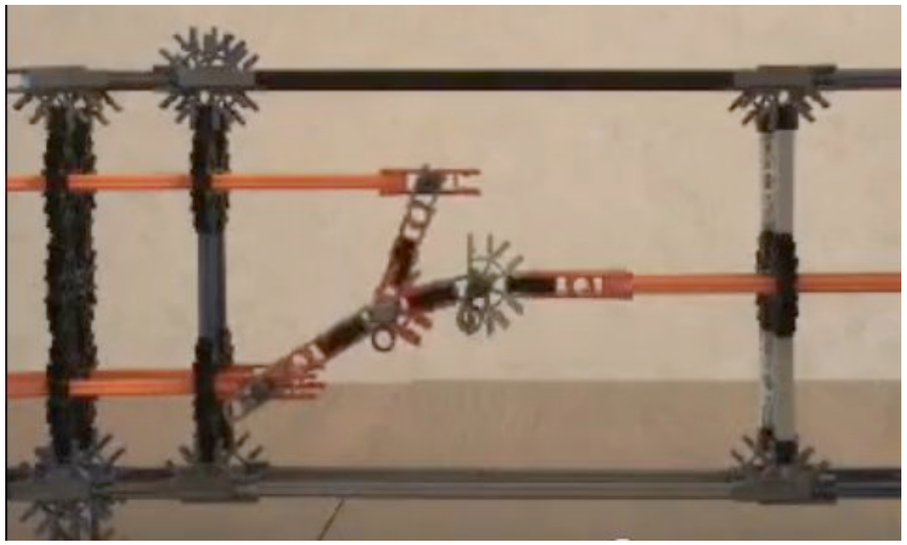


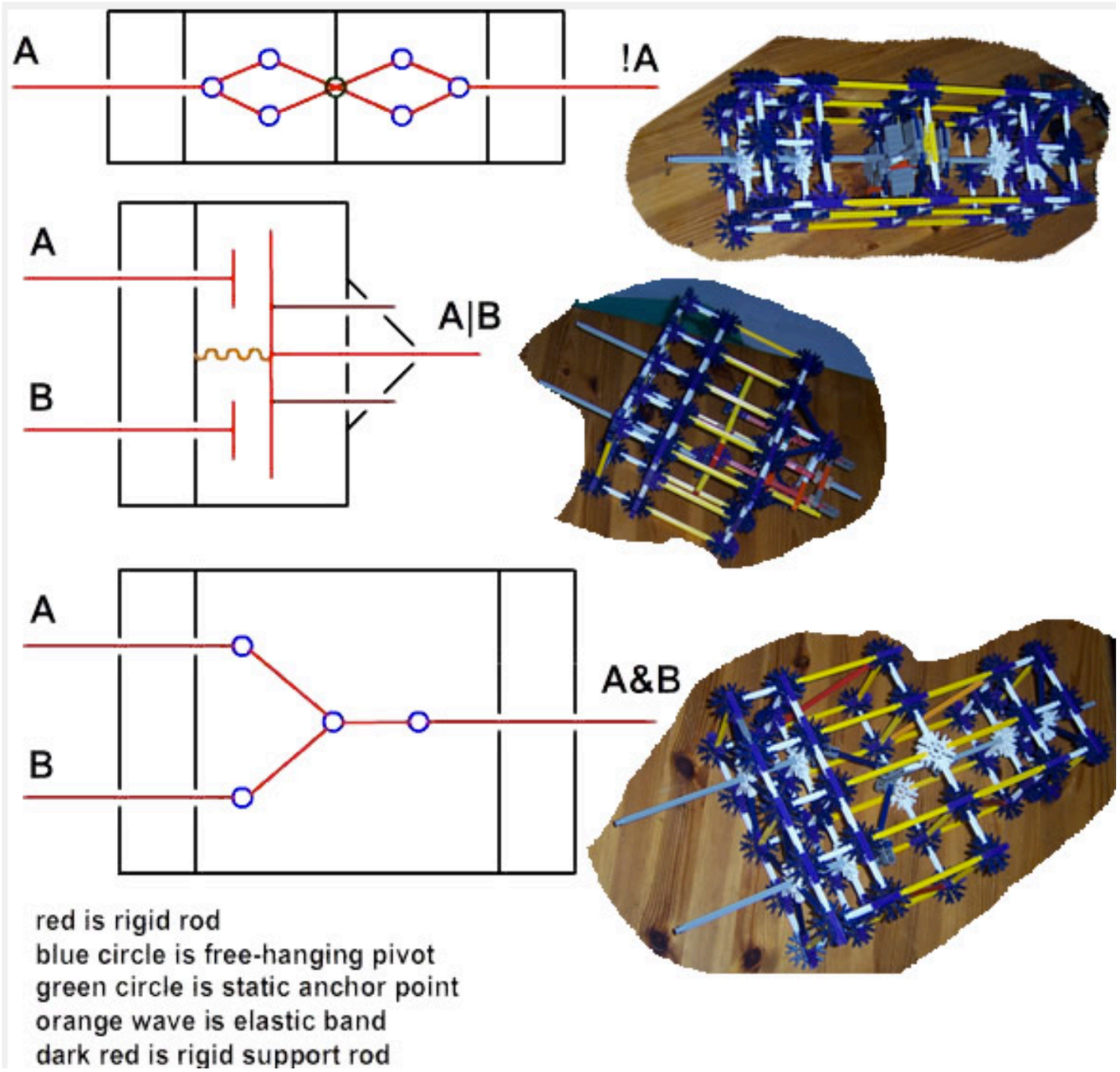
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

The OR operation will be signified by A + B. Other common mathematical notations for it are $A \vee B$ and $A \cup B$, called the union of A and B.

K'nex implementations of logic gates

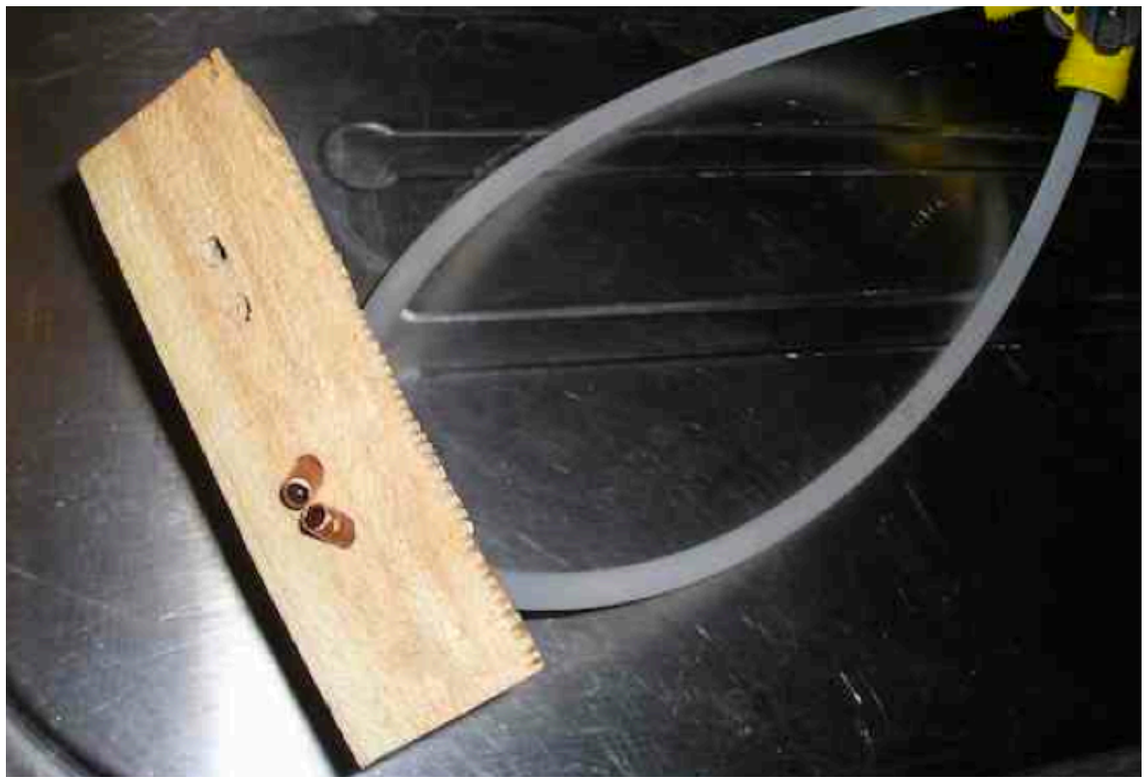






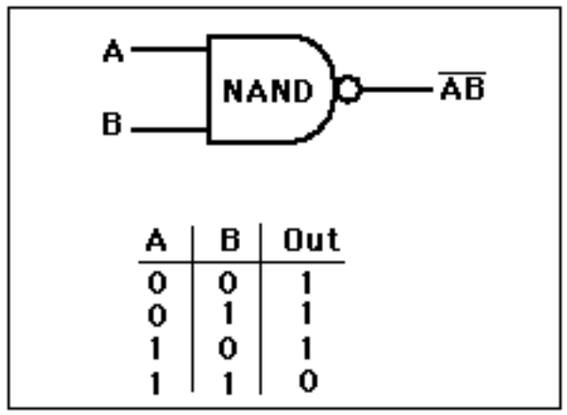
red is rigid rod
blue circle is free-hanging pivot
green circle is static anchor point
orange wave is elastic band
dark red is rigid support rod

Water gates

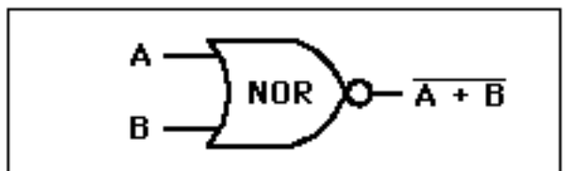


AND gate

NAND and NOR gates



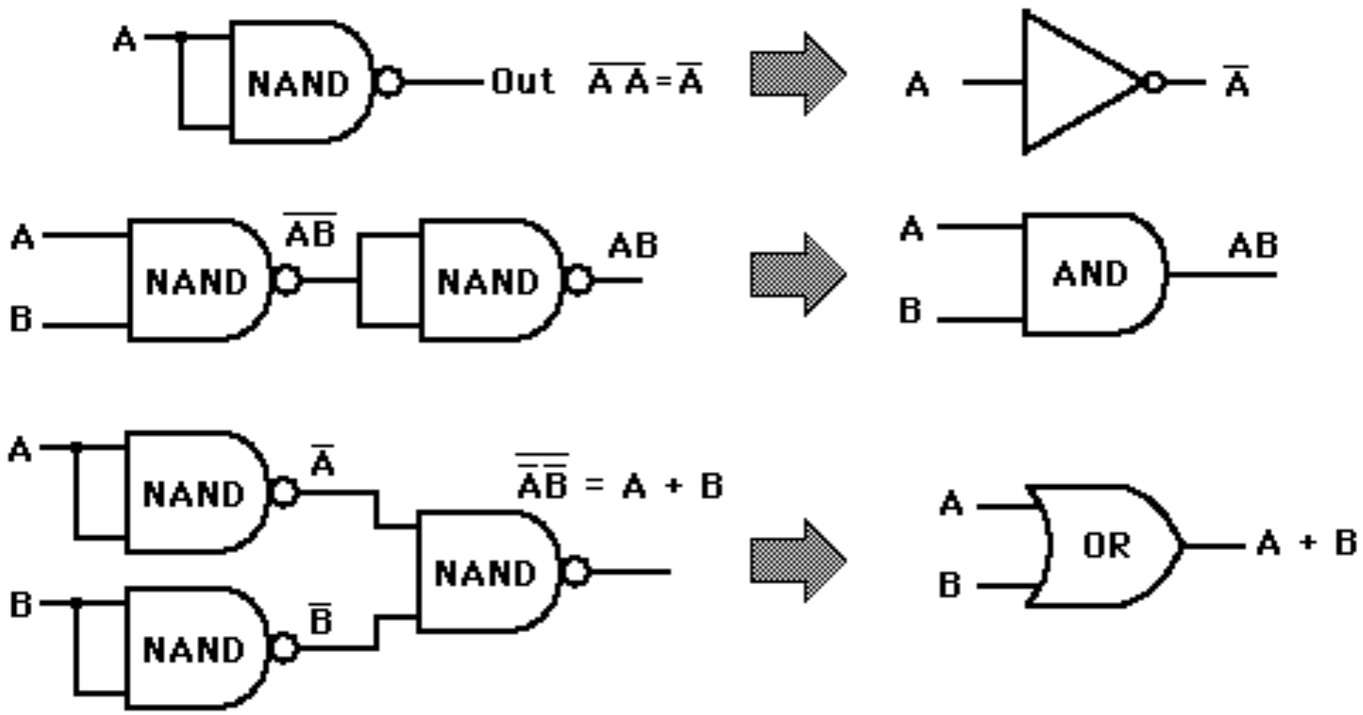
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



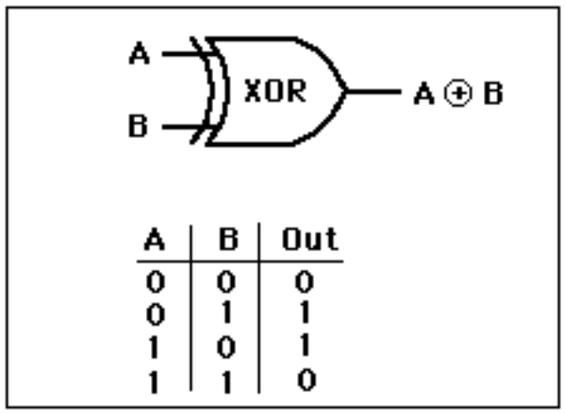
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

- With just NAND and NOR gates we can come up with all other basic gates.

- They are called **universal gates**!



XOR gates

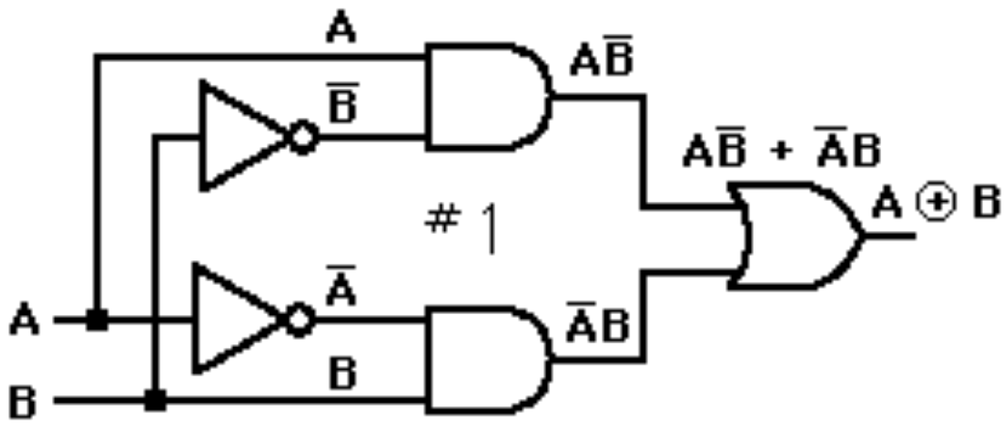


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

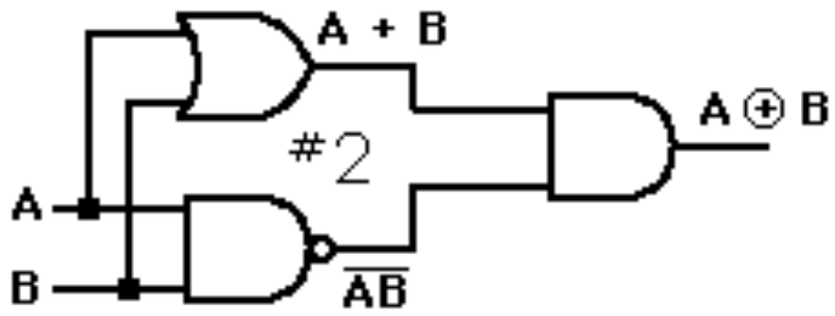
The output is high when either of inputs A or B is high, but not if both A and B are high.

Logically, the XOR operation can be seen as either of the following operations:

$A \oplus B = A\overline{B} + \overline{A}B$

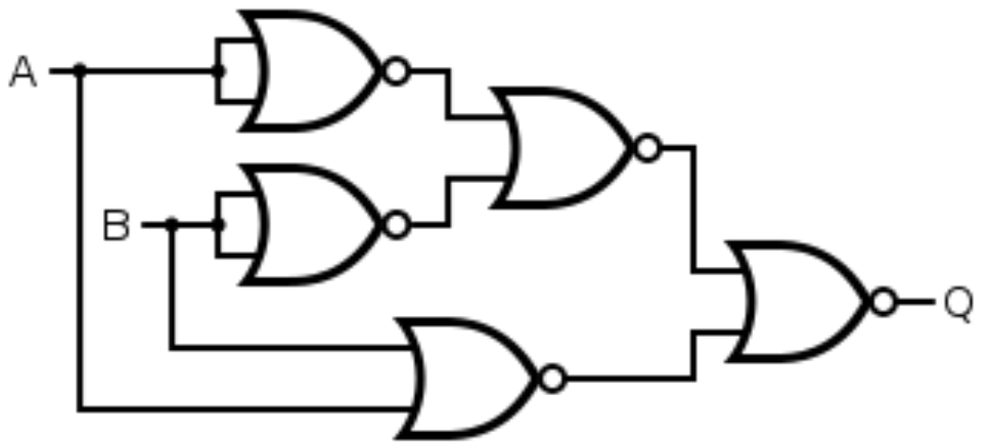


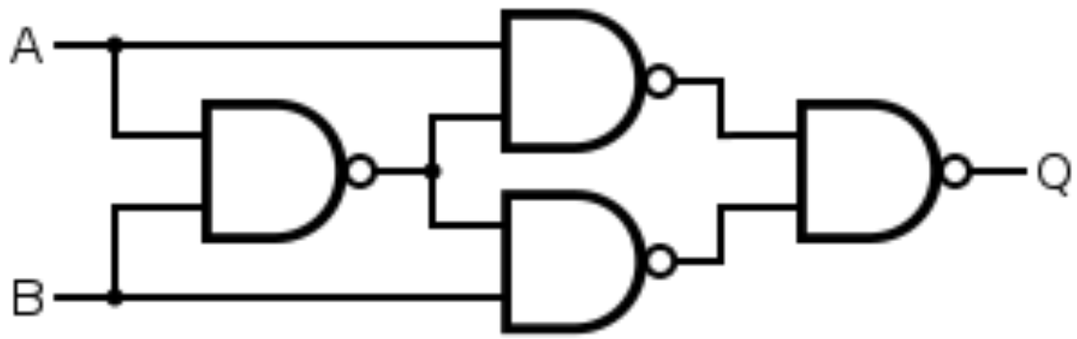
$A \oplus B = (A + B)(\overline{AB})$



XOR gate implementations

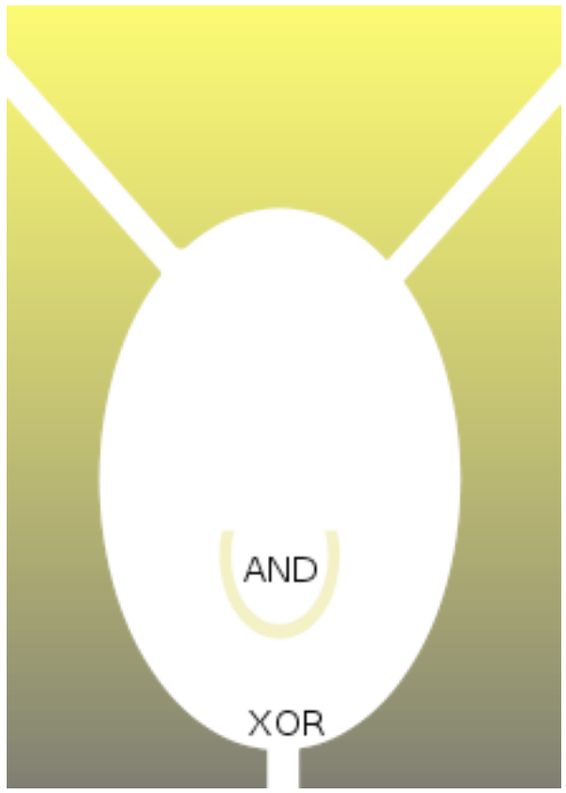
Implementing XOR gates with NOR and NAND universal gates...





Fluidics

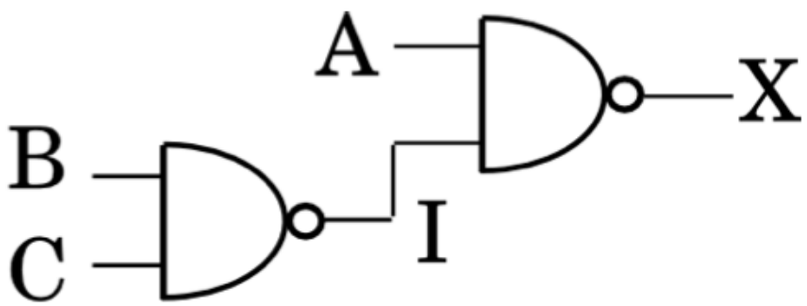
Fluidics is the use of a fluid to perform analog or digital operations similar to those performed with electronics.



The two gates AND and XOR in one module. The bucket in the center collects the AND output, and the output at the bottom is A XOR B.

http://www.blikstein.com/paulo/projects/project_water.html

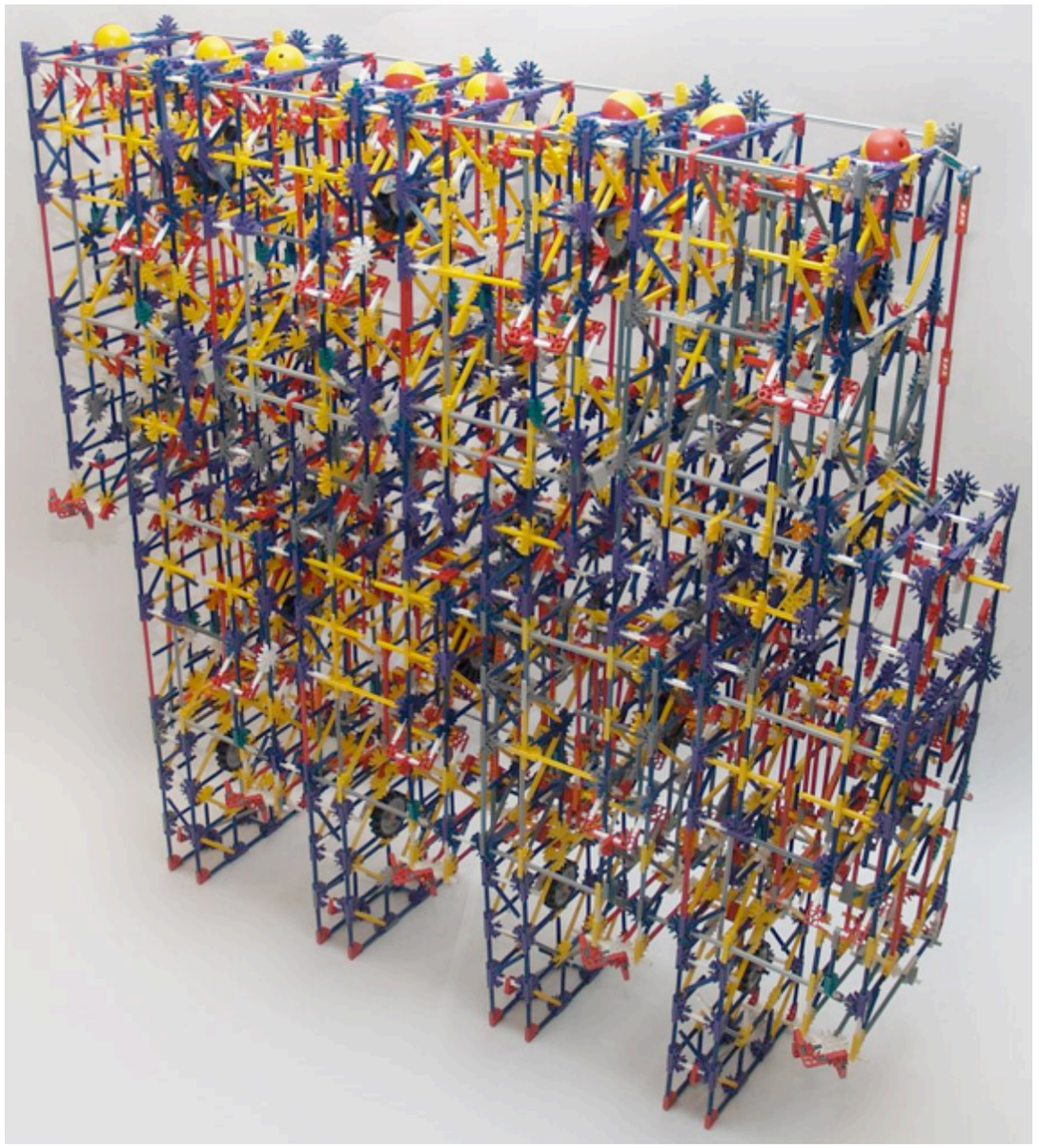
Truth tables



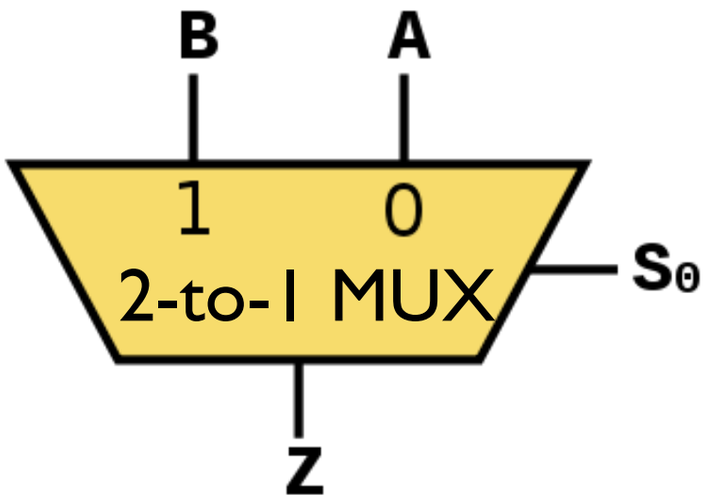
A	B	C	I = $(B \cdot C)'$	X = $(A \cdot I)'$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Physical implementation of a 4-Bit adder

http://knexcomputer.blogspot.com/2006_12_01_archive.html



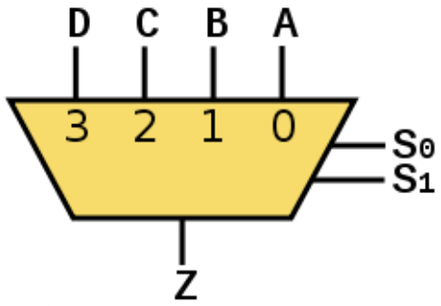
Multiplexer (MUX)



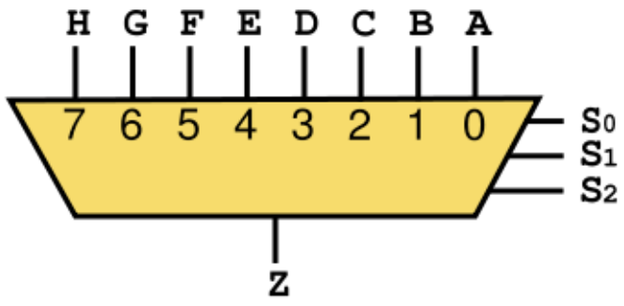
$$Z = (A \cdot \overline{S}) + (B \cdot S)$$

S	A	B	Z
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

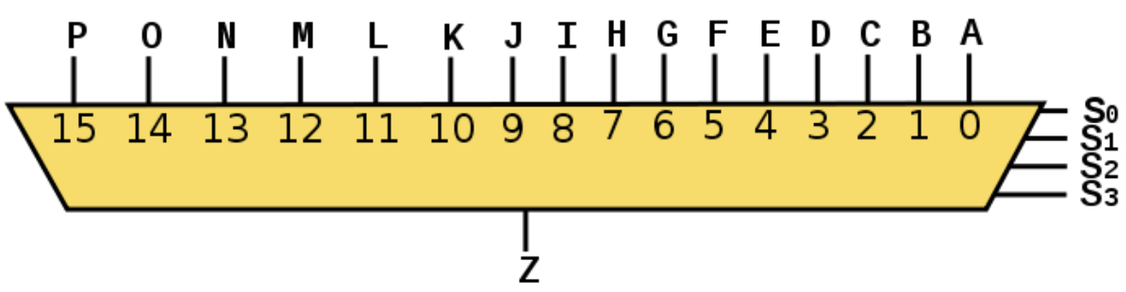
You have n input signals (A and B in this example). With the select bit(s), you determine which input will propagate to the single output.



4-to-1 MUX

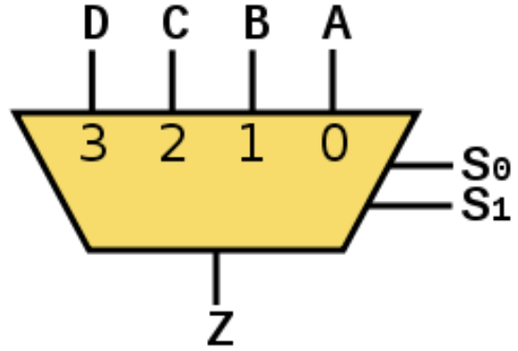


8-to-1 MUX



16-to-1 MUX

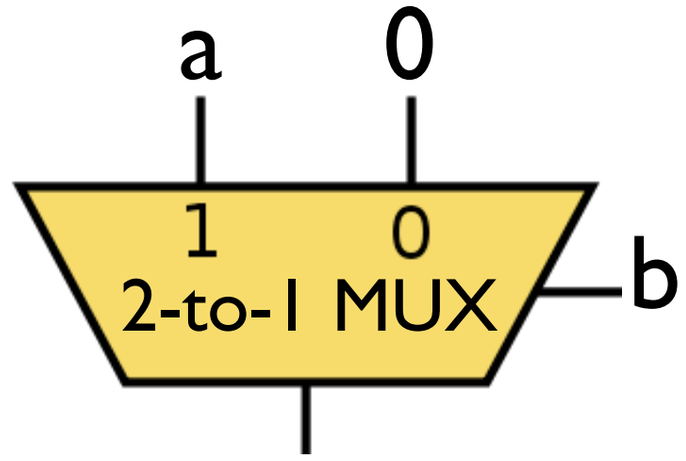
4-to-1 MUX



$$F = (A \cdot \overline{S_0} \cdot \overline{S_1}) + (B \cdot S_0 \cdot \overline{S_1}) + (C \cdot \overline{S_0} \cdot S_1) + (D \cdot S_0 \cdot S_1)$$

Lets play some MUX games

With any number of 2-to-1 MUX gates, implement an AND gate with the function OUT = A . B



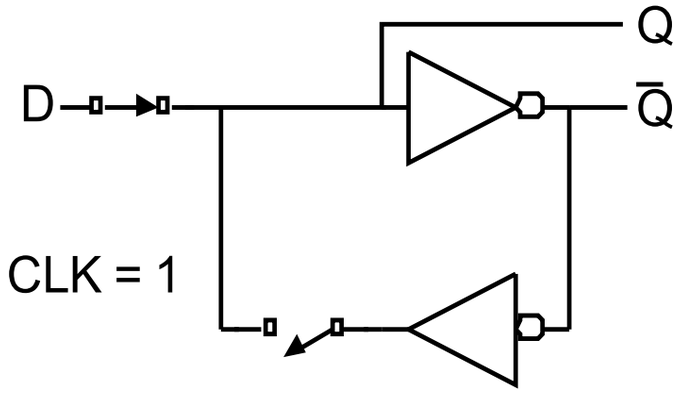
$$OUT = a.b + 0.\overline{b}$$

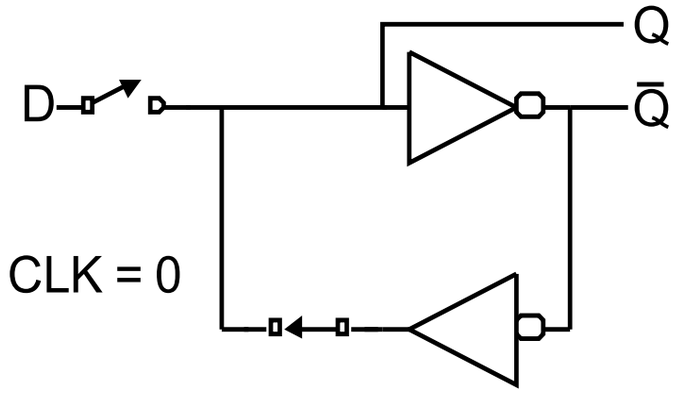
What is sequential logic

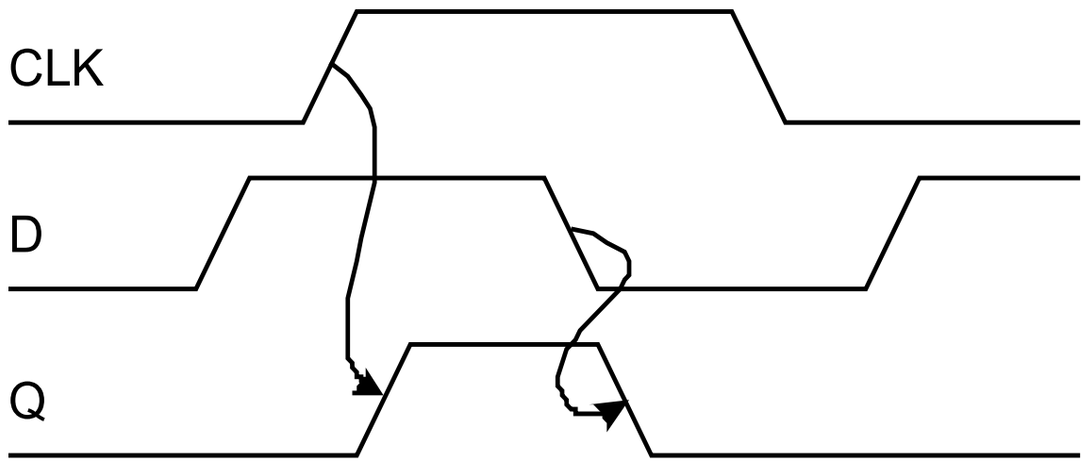
- Sequential logic is a type of logic circuit... whose output depends not only on the present input but also on the history of the input.
- In combinational logic, the output is a function of, and only of, the present input.
- In other words, sequential logic has state (memory) while combinational logic does not.
- Sequential logic is used to construct some types of computer memory storage elements, and finite state machines.

D-Latch operation

Inputs		Outputs		
Clock	D	Q	~Q	Comment
0	X	Qprev	~Qprev	No change
1	0	0	1	Reset
1	1	1	0	Set

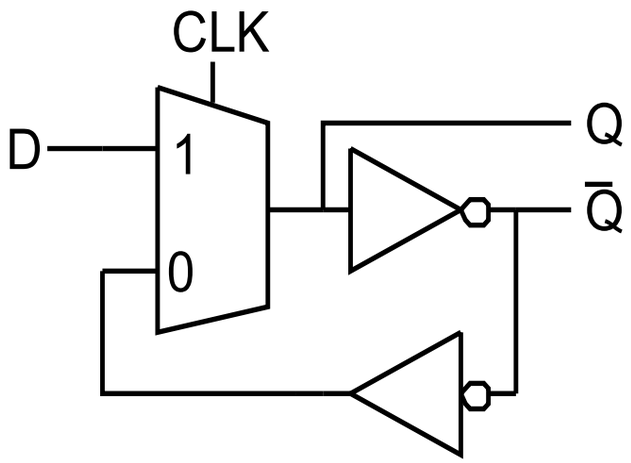




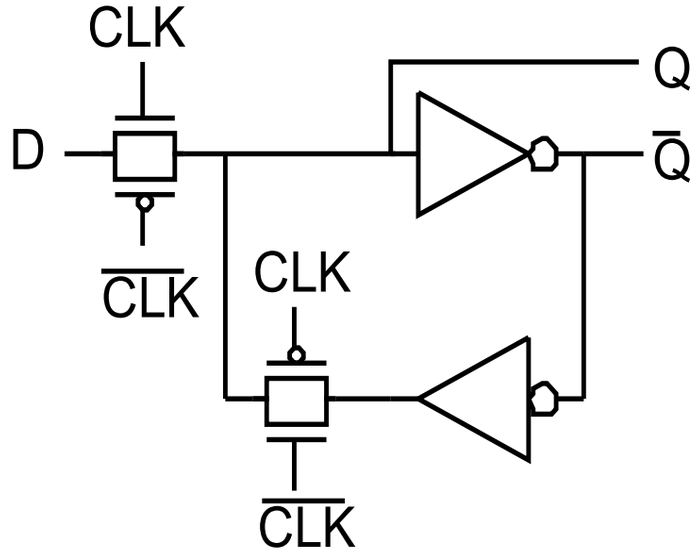


Two D-Latch designs

Multiplexer chooses D or old Q



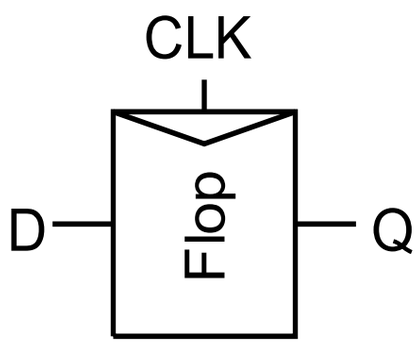
...With a MUX

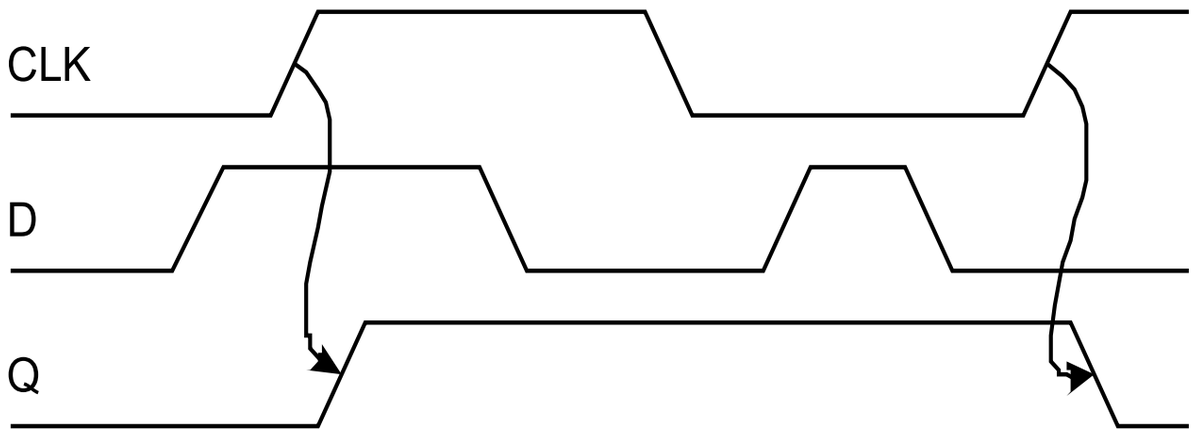


...With transmission gates

D-Flip Flop

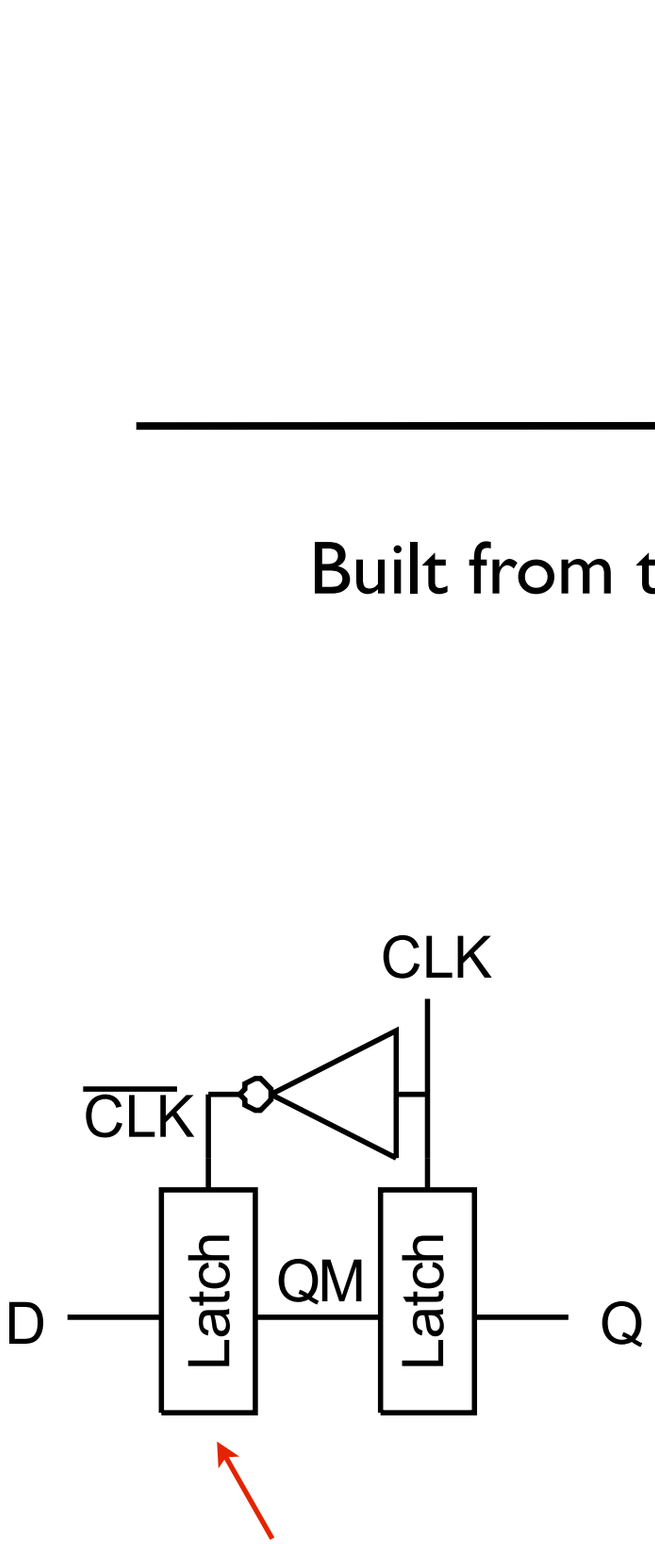
- **Only when CLK rises**, D is copied to Q
- At all other times, Q holds its value
- a.k.a. positive edge-triggered flip-flop, master-slave flip-flop



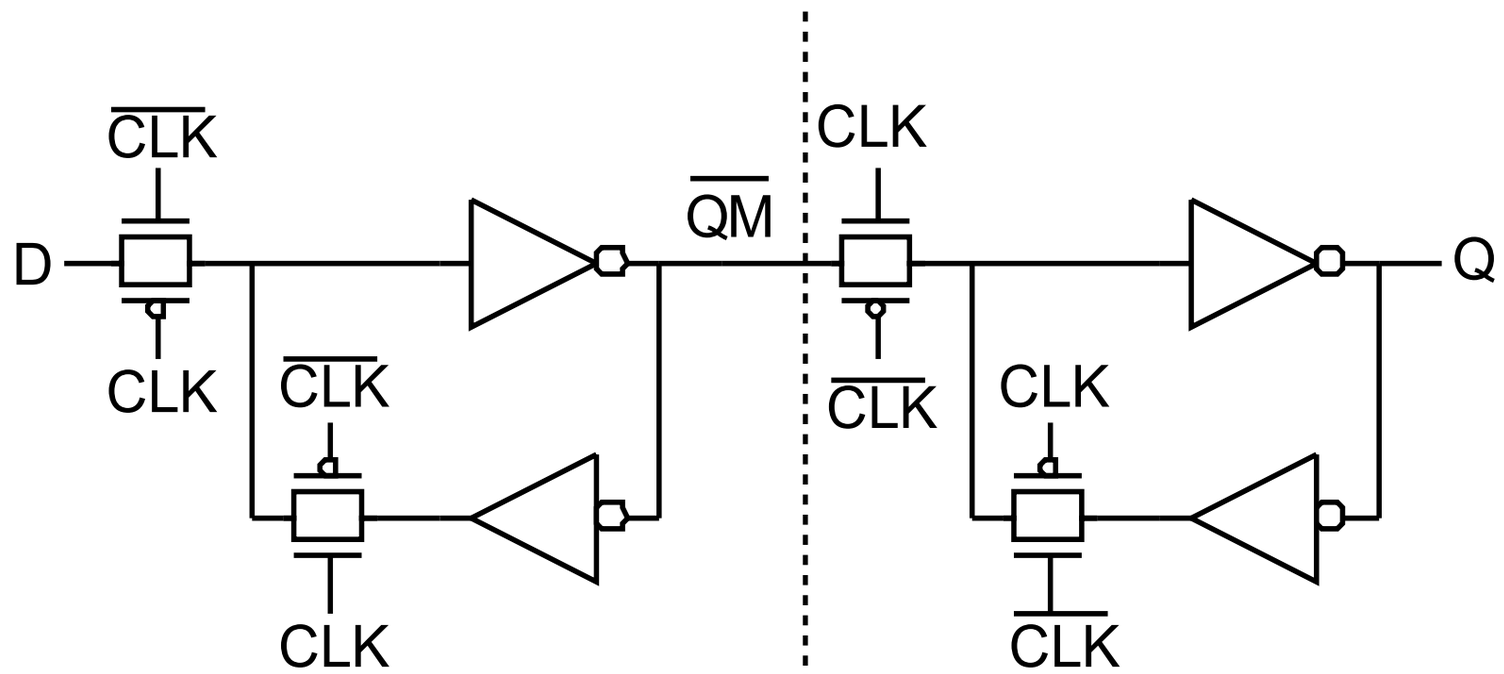


D-Flip Flop design

Built from two D latches (one is master, other is the slave)



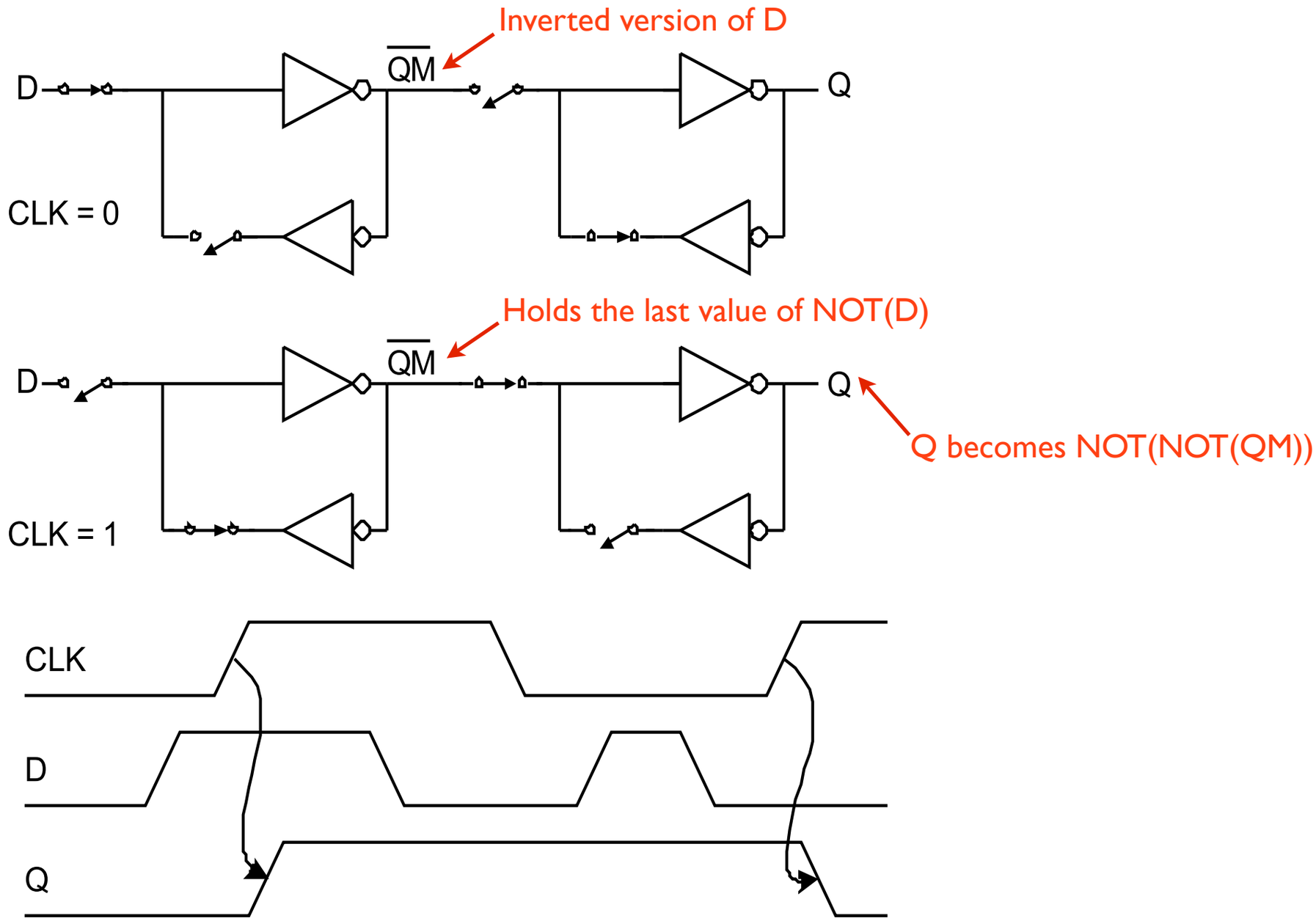
Note: This latch is ON when CLK is LOW!



"Negative level-sensitive" latch

"Positive level-sensitive" latch

D-Flip Flop operation



Logic arithmetic

Nuno Alves (nalves@wne.edu), College of Engineering

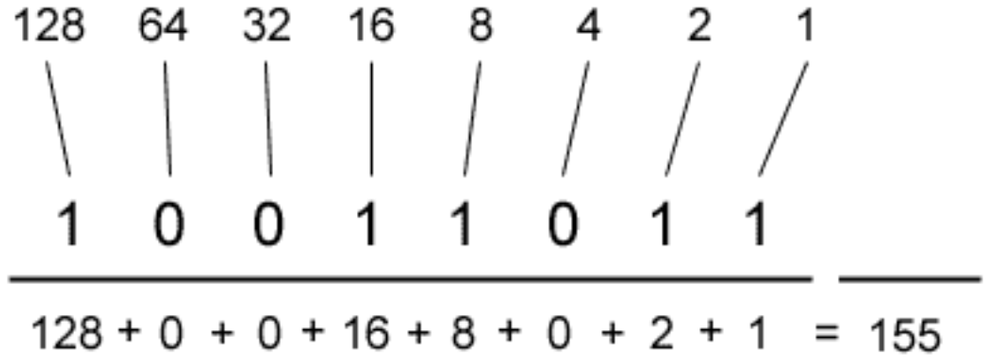
Digital vectors

- We need many bits in order to do anything meaningful.
- So, we group them together into n-bits.
- A group of bits is called a vector.
- 8 bits is a byte.
- 4 bits is a nibble.

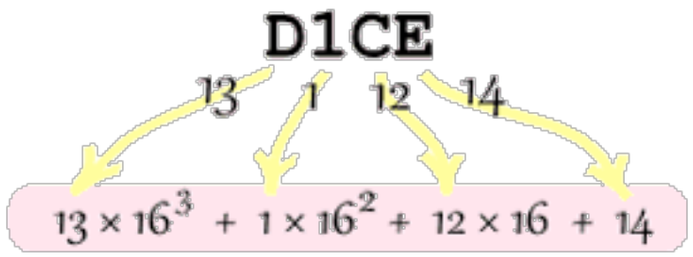
Decimal	Hexadecimal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversions

- You must know how to convert between base 2 (binary), 10 (decimal), and 16 (hexadecimal).



Converting binary to decimal



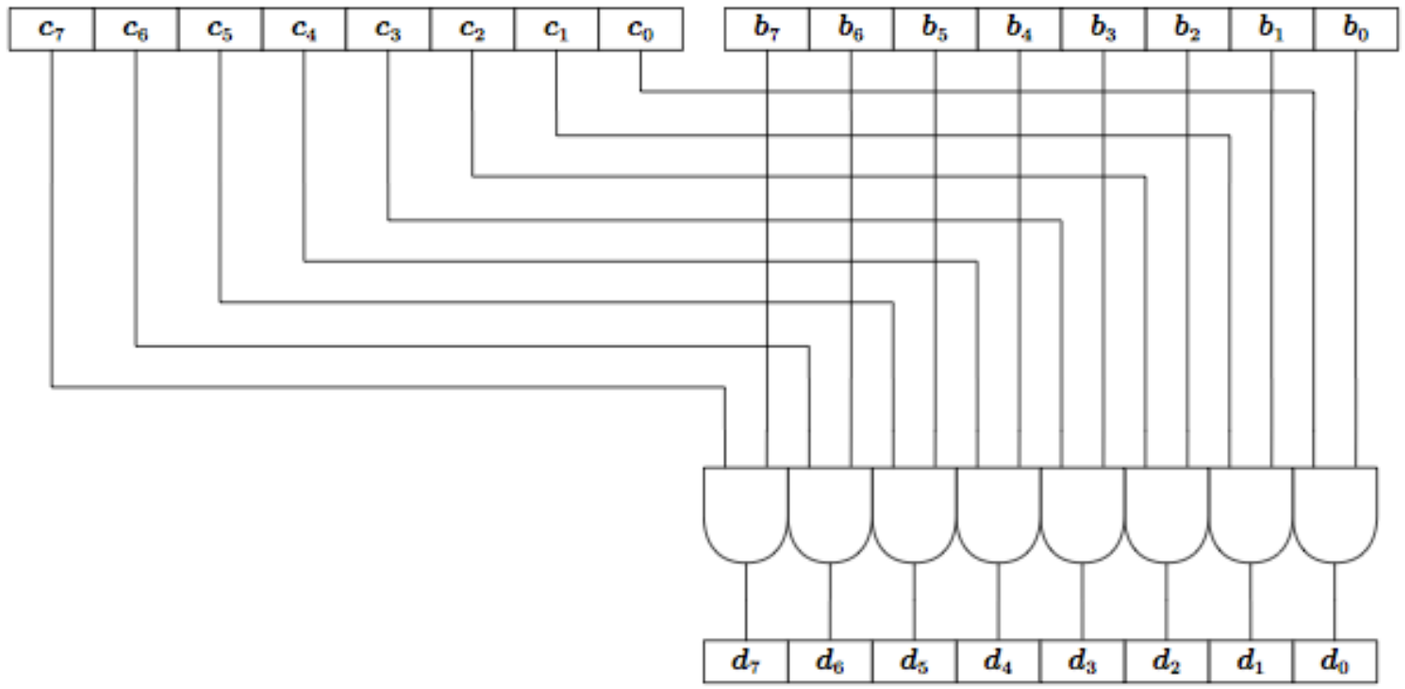
Converting hexadecimal to decimal

Some examples

- $01001101_2 = 4D_{16}$
- $53_{16} = 01010011_2$
- $01001101_2 = 77_{10}$
- You can generate as many examples as you want in wolfram alpha (www.wolframalpha.com)

AND of two inputs

- Suppose the CPU is required to logically AND two inputs together.
- It would have to perform the logical AND operation bit-by-bit on two 8-bit vectors, probably sitting in CPU registers.
- This is required since the CPU generally can't access individual bits – the smallest unit in most CPUs is an 8-bit byte.



Endians

- **Big-endian** bit format: the most significant bit (MSB) is the first element of the bit vector and the least significant bit (LSB) is the last element of the vector.
- However, some manufacturers will reverse this order, which is called **little-endian**.
- ATMEL ATmega328P (the micro-controller inside the Arduino) follows the big-endian format.
- Suppose we have 00100110_2 but we are not told the order. This number could be either 32_{10} (big endian) or 100_{10} (little-endian). It is important to know the bit-endianness.

Signed Integers

- There is a sign bit, usually the most-significant bit (MSB).
- If MSB = 0, then number is positive, otherwise the number is negative

Let $n = 8$. The representation for 77 and -77 are

		sign	magnitude
77	=	0	1001101
-77	=	1	1001101

A signed byte has a range of values from -127_{10} to 127_{10}.

Problems with signed integers:

- Two versions of zero, ($10_2 = 0_{10}$) and ($00_2 = 0_{10}$) which are both mathematically equal.
- The hardware required to perform arithmetic in the ALU is complex, large, slow, etc.

Two's Complement

- When performing ALU operations with integers, two's complement is the only signed integer code that is ever used in practice.
- The two's complement code allows for addition and subtraction treatment of bit strings as though they were unsigned integers.
- The code was designed to allow for efficient hardware in an ALU, i.e., the code does not care if human engineering students are unable to decipher the meaning of a bit vectors for a homework assignment.

How to write a positive number in 2's complement?

- Suppose we're working with 8 bit quantities
- We want to find 27_{10} in two's complement.

1. We just write out 27_{10} in binary form : $0001\ 1011_2$
2. ...and thats it!

Use wolfram alpha to verify your answers! →

How to write a negative number in 2's complement?

- Suppose we're working with 8 bit quantities
- We want to find -28_{10} in two's complement.

1. First we write out 28_{10} in binary form : 00011100_2
2. Then we invert each digits : 11100011_2
3. Then we add 1 : 11100100_2

Be careful with wolfram alpha here, since we can't specify the output bit length.

2‘s Complement conversion examples

Q: In 2‘s complement, what is 00110111_2 in base 10?

A: The MSB bit is 0, so its a positive number. So it is 55_{10}.

Q: In 2‘s complement, what is 11100110_2 in base 10?

A: The MSB bit is a ‘1’, so we know it is a negative number. We need to invert the bits before we add the powers-of-2. Also, we need to add 1 before applying the minus sign.

So,

$(11100110) \Rightarrow -[(00011001) + 1] \Rightarrow -26_{10}$

Addition and subtraction with 2's complement

- Addition of any two numbers in two's complement is as follows:

- 1. Perform binary addition of all bits including the sign bit.
  2. If a '1' is carried out, discard it.

- Subtraction of any two numbers in two's complement is handled by negating one of the numbers and performing addition.

Example of addition in 2's complement

- Let's add 0100_2 with 1101_2 in 2's complement
- 0100_2 is 4 while 1101_2 is -3 (do this by hand if you don't believe me)

```
   0100
 + 1101
 ------
  10001   -> We get ignore the carry out
```

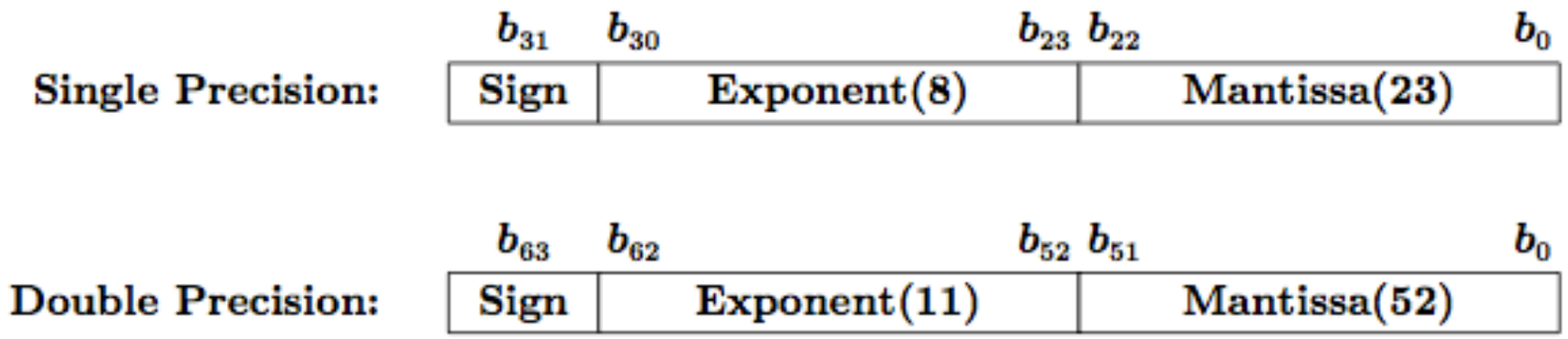
- The sum is 0001_2 which, as expected, is 1_{10}.

Floating point in hardware

- Unlike integers, in which hardware is pretty much all the same across all architectures, floating-point codes are not standard.

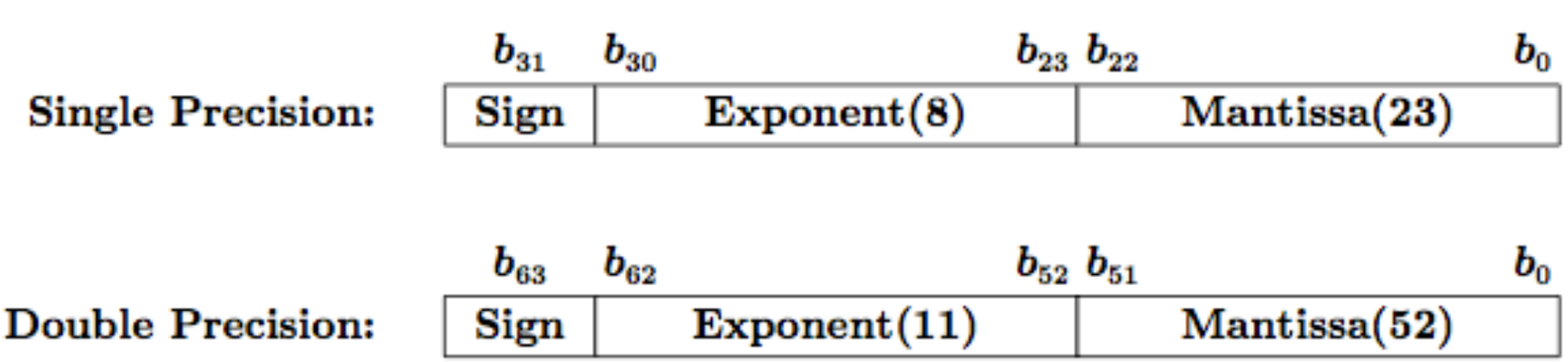
- However, the same basic concepts are generally present. In particular, out of an **n**-bit number, one bit is required for the sign of the number, **m** bits are required for the mantissa, and the remaining **e** bits are used to represent the exponent.

- Here are the IEEE floating point formats:



Representing floating points

- Floating point arithmetic is always very cycle expensive unless the processor has a floating point unit built into the ALU, which is hardware expensive.



Converting a binary number into a decimal number

$$x = (-1)^s \times \left(2^{e-127}\right) \times 1.m$$ Single precision

$$x = (-1)^s \times \left(2^{e-1023}\right) \times 1.m$$ Double precision

What's that binary dot?

Remember that

- $2^{-1} = 1/2^1 = 0.5$
- $2^{-2} = 1/2^2 = 0.25$
- $2^{-3} = 1/2^3 = 0.125$

$$1111. = 2^3 + 2^2 + 2^1 + 2^0 = 15$$
$$111.1 = 2^2 + 2^1 + 2^0 + 2^{-1} = 7.5$$
$$11.11 = 2^1 + 2^0 + 2^{-1} + 2^{-2} = 3.75$$
$$1.111 = 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 1.875$$
$$.1111 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 0.9375$$

Base-10 Number	Sign	Exponent	Mantissa	Value
1.0	0	01111111	00000000000000000000000	1.0
0.5	0	01111110	00000000000000000000000	0.5
3.5	0	10000000	11000000000000000000000	3.5
largest	0	11111111	11111111111111111111111	6.80564e+38
smallest	0	00000000	00000000000000000000001	5.87747e-39
0.0	0	00000000	00000000000000000000000	0.0
2.44	0	10000000	00111000010100011110101	2.43999
-12.16	1	10000010	10000101000111101011100	-12.15999
-9.72	1	10000010	00110111000010100011110	-9.71999

Single precision

$$x = (-1)^s \times \left(2^{e-127}\right) \times 1.m$$

Double precision

$$x = (-1)^s \times \left(2^{e-1023}\right) \times 1.m$$

Example of converting a base 2 single precision float into base 10

Base-10 Number	Sign	Exponent	Mantissa	Value
3.5	0	10000000	11000000000000000000000	3.5

Single precision

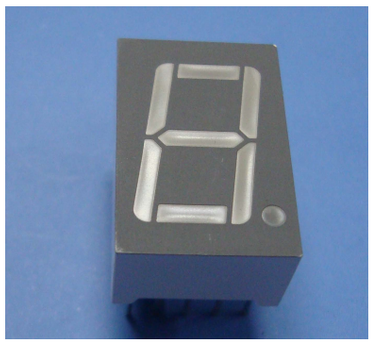
$$x = (-1)^s \times \left(2^{e-127}\right) \times 1.m$$

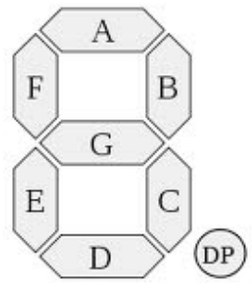
- Sign = 0. So, **$(-1)^s$** is $(-1)^0 = 1_{10}$
- The 8 bit exponent is 10000000 or 128. So, **2^{e-127}** is $2^{128-127} = 2^1 = 2_{10}$
- The 23 bit mantissa is 1100(...) = 0.5+0.25 = 0.75. So, **1.m** is 1.75_{10}
- $x = 1 * 2 * 1.75 = 3.5_{10}$

Binary Coded Decimal

- Binary Coded Decimal (BCD) is a code used to allow direct conversion between a nibble of bits and a positive decimal value.
- BCD is a code that allows for a quick conversion between base-2 and base-10. However, it is not very compact because each nibble is only able to code for ten digits instead of 16.
- Generally computer architectures are not designed to manage BCD values directly. BCD is still useful for embedded applications when interacting with seven-segment displays.

Base-2	BCD	Base-2	BCD	Base-2	BCD	Base-2	BCD
0000	0	0100	4	1000	8	1100	-
0001	1	0101	5	1001	9	1101	-
0010	2	0110	6	1010	-	1110	-
0011	3	0111	7	1011	-	1111	-





ASCII

- The American Standard Code for Information Interchange (ASCII) is a binary code in which 7-bit vectors are used to represent many alphabetic, numeric and symbolic characters.

- The standard was originally designed to include control characters meant to manage peripherals such as printers, and so some of the symbols do not see a lot of use any more.

- Given the typical 8-bit byte, the bit vector 01010011_2 represents the printable character 'S' based on the ASCII code.

Base-16	ASCII	Base-16	ASCII	Base-16	ASCII	Base-16	ASCII
00	NULL	20	Space	40	@	60	‘
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	’	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL