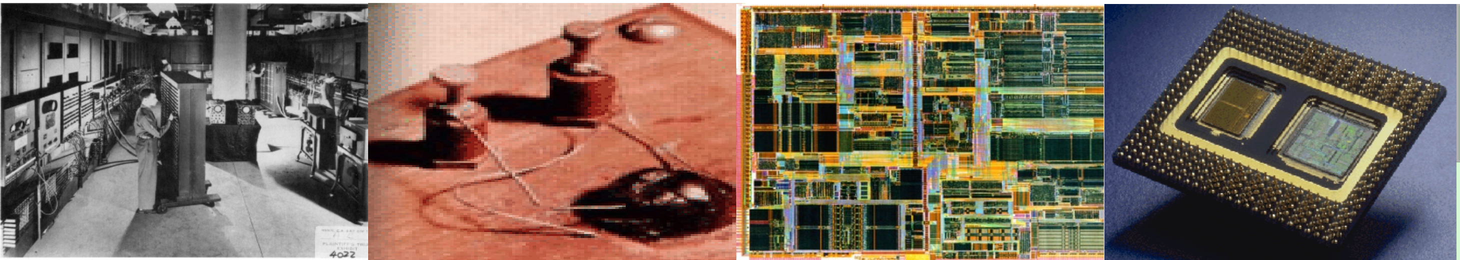
CPE 487: Digital System Design

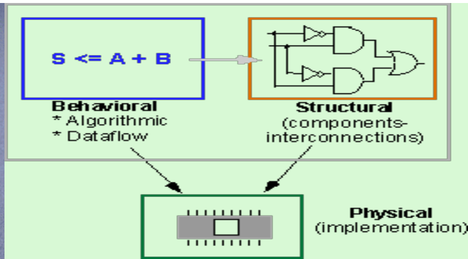
Spring 2018

Lecture 2

Digital Logic Basics

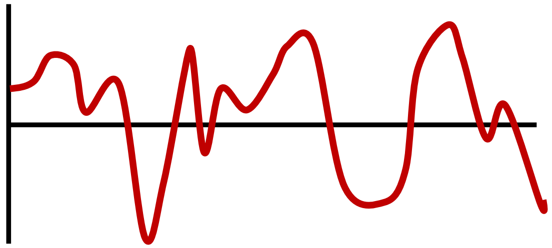
Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030





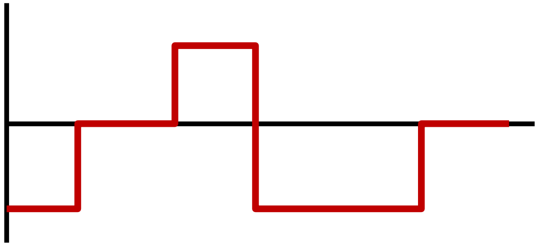
Digital Abstraction

- Most physical variables are **continuous**
  - voltage on a wire
  - frequency of an oscillation
  - position of a mass



- Computation on continuous variables subject to noise and distortion
  - any computation will have finite error
  - errors will accumulate

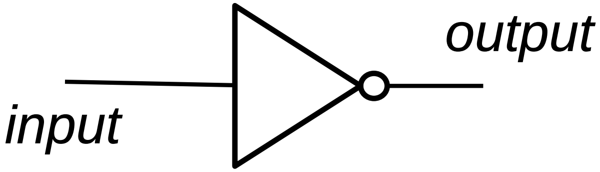
- Digital abstraction considers **discrete subset** of values
  - output can be “restored” to correct value
  - error free (with very high probability)

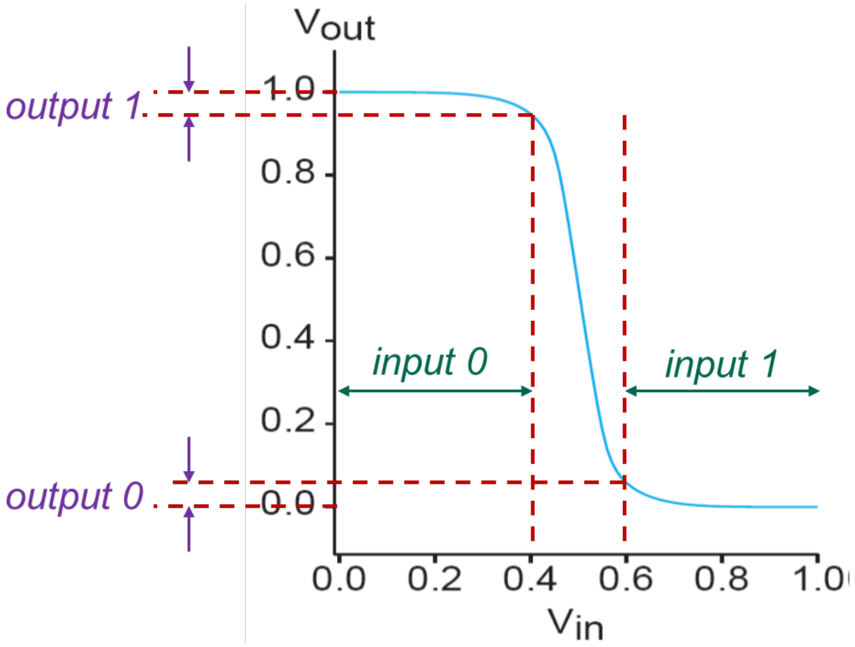


Digital Discipline: Binary Values

- Early computing engines used multi-value digital variables
  - Babbage engine used gears with 10 different positions
  - Simplified base$_{10}$ arithmetic
- Very difficult to build electronic circuits that restore to multiple (>2) discrete values
- Very easy to build circuits that restore to two values
- Use two discrete (binary) values: **0** and **1**

Transfer function of a simple CMOS inverter:





Digital Discipline: Binary Values

- Binary signals can be used to represent logical values:
  - **0** = FALSE **1** = TRUE
- Binary signals can be used to represent numerical values:
  - using base_2 representation
  - each binary signal represents one binary digit (**bit**)
- Binary signals can be used to represent any other variable that can only take on one of two different values
  - e.g. black/white, on/off, up/down
- In digital electronic circuits:
  - **0** is usually low voltage (ground, VSS, 0 volts)
  - **1** is usually high voltage (power supply, VDD, 3.3 volts)
- Beauty of (binary) digital abstraction is that the designer does not need to know the (physical) implementation details
  - can just focus on **0**'s and **1**'s

Formal (Philosopher’s) Logic

A: All dogs are warm blooded

B: Molly is a dog

C: Molly is warm blooded

A	**B**	**C**
T	T	**T**
T	F	**?**
F	T	**?**
F	F	**?**

If (A is true) ***and*** *if (B is true),* ***then*** *(C is true)*

What if B is not true. Does that make C false?

- e.g. What if Molly is a cat?

Formal logic does not address cases not explicitly covered in the logic statement

Digital (Boolean) Logic

In digital logic, there is always an implied *else* clause

If (A is true) ***and*** *if (B is true),*

then *(C is true);* ***else*** *(C is false)*

A: If you have come to a complete stop

B: There is no traffic coming

C: You may proceed

If (A is false) ***or*** *if (B is false),*

then (C is false); else (C is true)

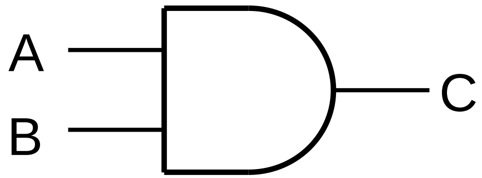
In digital logic, usually use ‘1’ for true, ‘0’ for false

A	**B**	**C**
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

A	**B**	**C**
1	1	**1**
1	0	**0**
0	1	**0**
0	0	**0**

AND gate

logic symbol



truth table

A	**B**	**C**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Boolean equation

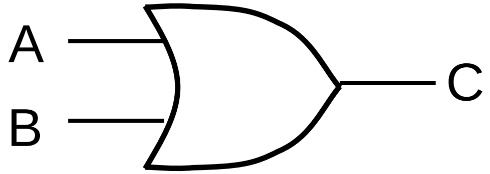
C = A*B

or C = A.B

When more than two inputs, the output equals ‘1’ *only* when all inputs are equal to ‘1’

OR gate

logic symbol



Boolean equation

C = A+B

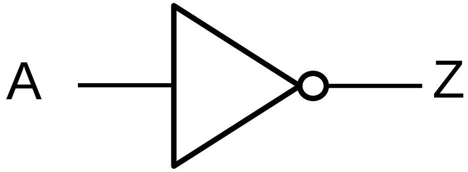
truth table

A	B	C
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

When more than two inputs, the output equals ‘1’ *when any* input is equal to ‘1’

Inverter or NOT gate

logic symbol



truth table

A	Z
0	**1**
1	**0**

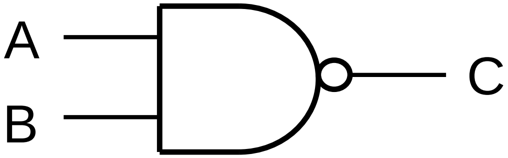
Boolean equation

$$Z = \overline{A}$$

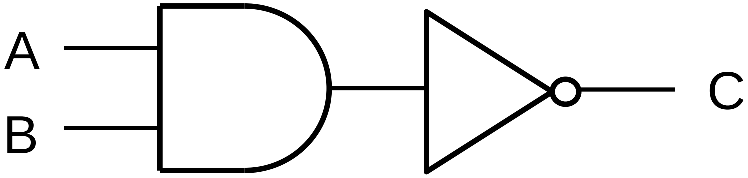
or $$Z = A'$$

NAND gate

logic symbol



equivalent to:



Boolean equation

$$C = \overline{A*B}$$

or $$C = \overline{A.B}$$

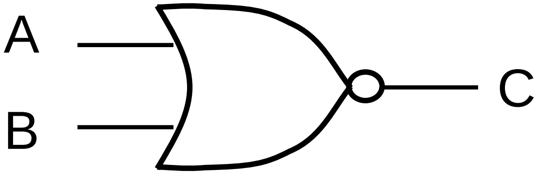
truth table

A	**B**	**C**
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

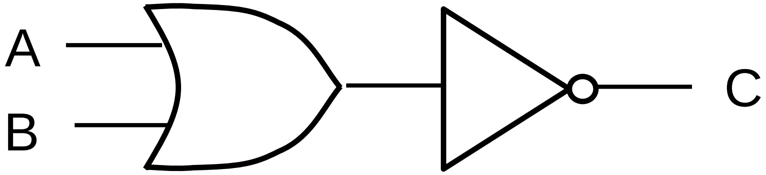
When more than two inputs, the output equals ‘0’ *only* when all inputs are equal to ‘1’

NOR gate

logic symbol



equivalent to:



Boolean equation

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

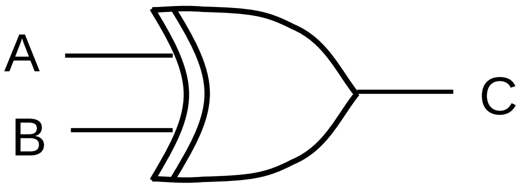
truth table

A	B	C
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

When more than two inputs, the output equals ‘0’ *when any* input is equal to ‘1’

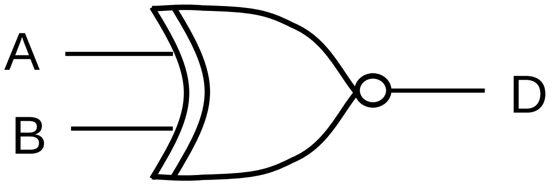
XOR and XNOR gate

XOR symbol



$C = A \oplus B$

XNOR symbol



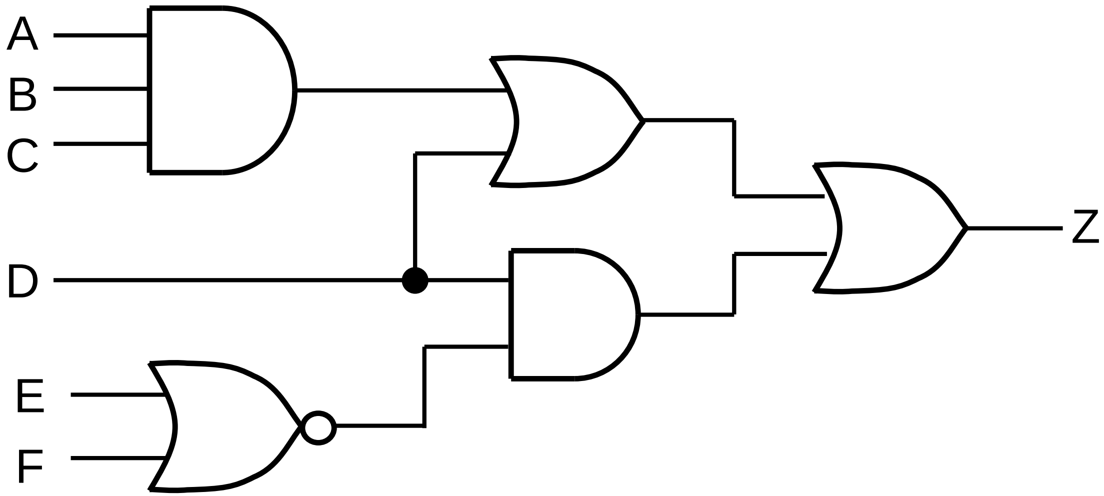
$D = \overline{A \oplus B}$

XOR/XNOR truth table

A	B	C	D
0	0	**0**	**1**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

When more than two inputs, the output of XOR equals ‘1’ *only when an odd number of* inputs are equal to ‘1’

Creating More Complex Logic Functions



$$Z=[(A.B.C) + D] + [D.\overline{(E+F)}]$$

Some Useful Formulae

$A + \text{‘0’} =$

$A + \text{‘1’} =$

$A + A =$

$A + \overline{A} =$

$A \oplus \text{‘0’} =$

$A \oplus \text{‘1’} =$

$A \oplus A =$

$A \oplus \overline{A} =$

$A \bullet \text{‘0’} =$

$A \bullet \text{‘1’} =$

$A \bullet A =$

$A \bullet \overline{A} =$

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

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

Some Useful Formulae

$A + \text{‘0’} = A$

$A + \text{‘1’} = \text{‘1’}$

$A + A = A$

$A + \overline{A} = \text{‘1‘}$

$A \oplus \text{‘0’} = A$

$A \oplus \text{‘1’} = \overline{A}$

$A \oplus A = \text{‘0’}$

$A \oplus \overline{A} = \text{‘1‘}$

$A \bullet \text{‘0’} = \text{‘0’}$

$A \bullet \text{‘1’} = A$

$A \bullet A = A$

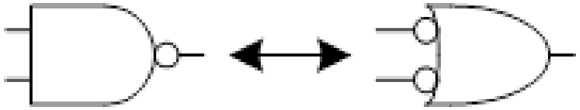
$A \bullet \overline{A} = \text{‘0‘}$

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

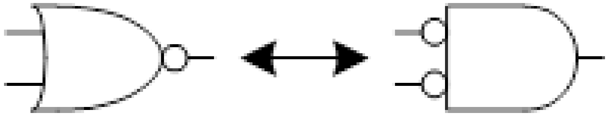
$$\overline{A \oplus B} = (A \bullet B) + (\overline{A} \bullet \overline{B})$$

DeMorgan’s Theorem

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



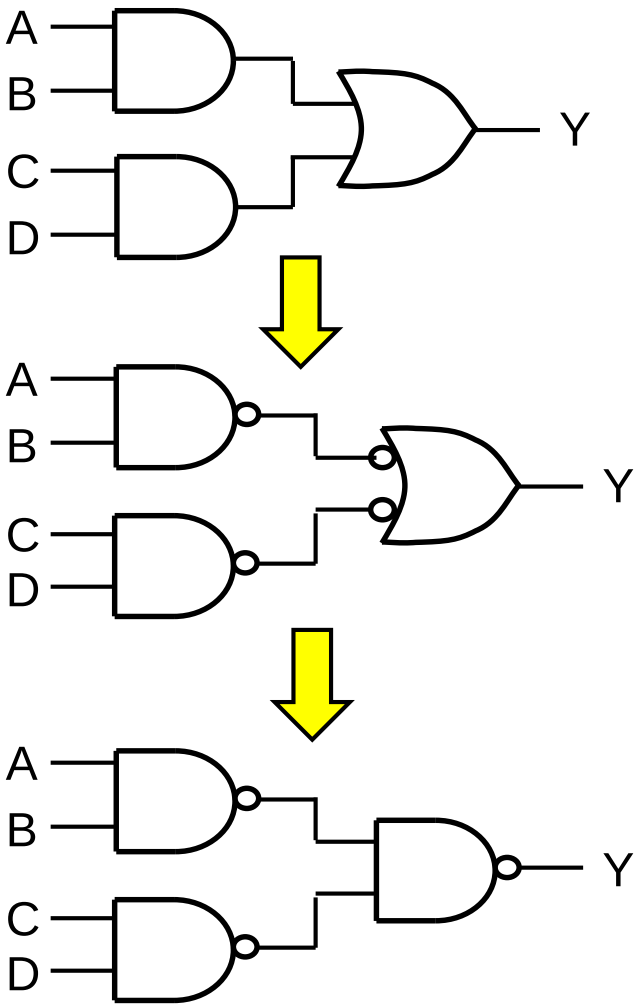
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$



1. Change AND to OR (OR to AND)
2. Invert all inputs
3. Invert output

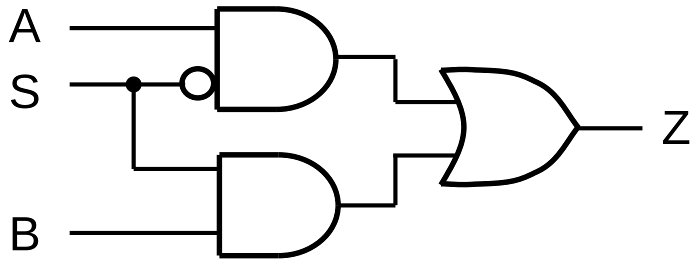
Example: AOI22

- $Y = A.B + C.D$

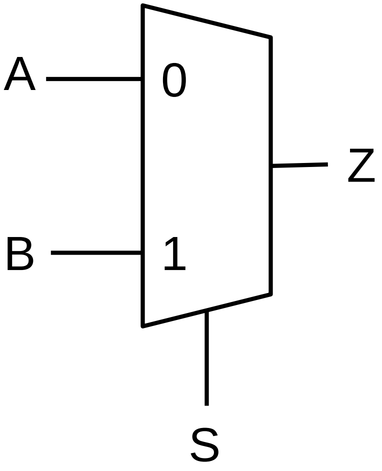


Multiplexer

- $Z = \overline{S}.A + S.B$



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

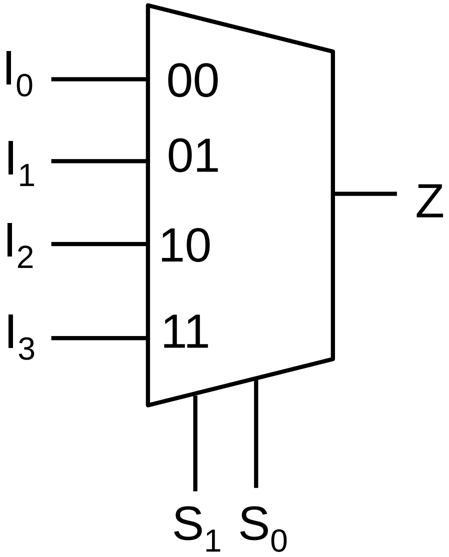


S	**A**	**B**	**Z**
0	0	-	**0**
0	1	-	**1**
1	-	0	**0**
1	-	1	**1**

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

4-input Multiplexer

- $Z = \overline{S_0}.\overline{S_1}.I_0 + S_0.\overline{S_1}.I_1 + \overline{S_0}.S_1.I_2 + S_0.S_1.I_3$



S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

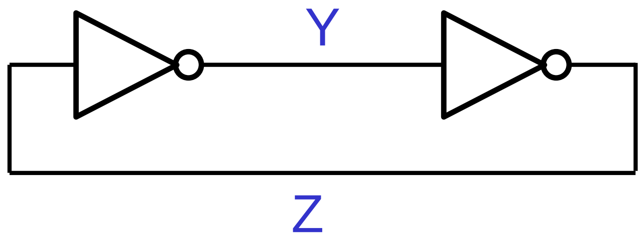
- Typically, an 2^N-*way* multiplexor will use N *select* signals to choose between one of 2^N inputs

Combinational vs. Sequential Logic

- A combinational circuit (logic) is one in which the output depends only on the current value of the inputs
  - All of the logic gates we have described so far (AND, NOR, XOR, multiplexer etc.) are combinational
  - If you know the inputs you know the outputs
- A sequential circuit (logic) is one in which the output depends on the current value and previous values of the inputs
  - Output depends on the sequence of applied inputs
  - Sequential circuits include some form of memory of previous inputs that modify output values
  - We often call these remembered values the state of the circuit or system.
  - All sequential circuits include some form of feedback loop to feed the remembered state back into the inputs of the circuit.

Memory – the cross coupled inverter

- Almost all form of digital memory are built around the idea of having two inverters (NOT gates) connected in a feedback loop.

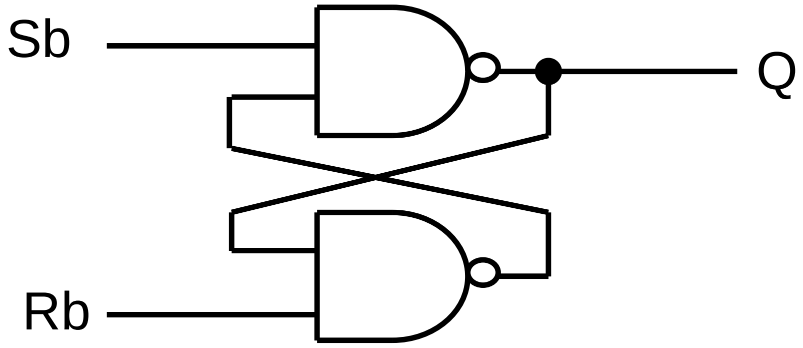


- Positive feedback drives circuit into one of two stable states
- Either: (Y=1, Z=0) *OR* (Y=0, Z=1)
  - Circuit will hold state indefinitely

- How do we change the state?

RS Latch

- Simple “writable” storage element



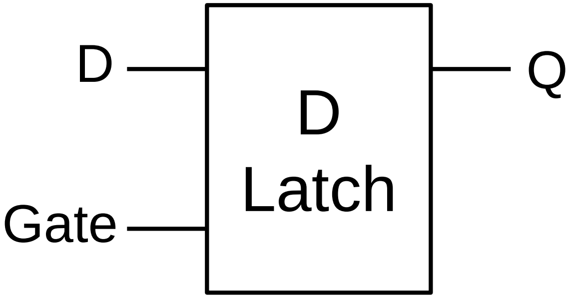
Rb	**Sb**	**Q**
0	1	0
1	0	1
1	1	*no change*
0	0	*illegal*

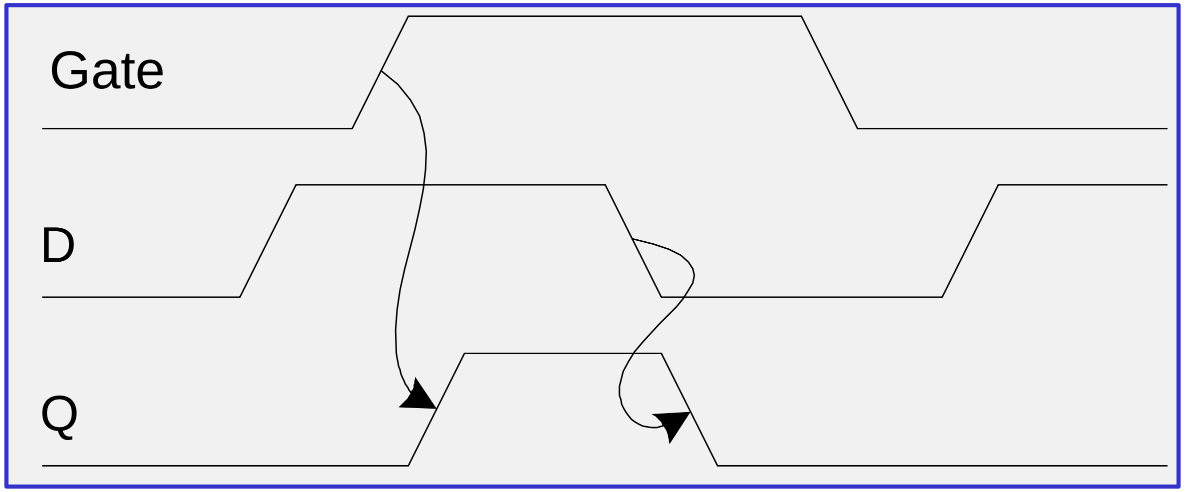
- Normally, Sb and Rb are both 1
- When Sb=0, Q is set to 1
- When Rb=0, Q is reset to 0

D Latch

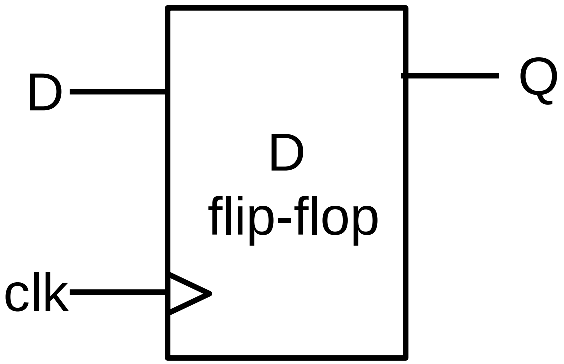
- When Gate = 1, latch is transparent
- D flows through to Q like a buffer
- When gate = 0, the latch is opaque
- Q holds its old value independent of D
- a.k.a. transparent latch or level-sensitive latch

D	Gate	Q
0	1	0
1	1	1
0	0	*no change*
1	0	*no change*



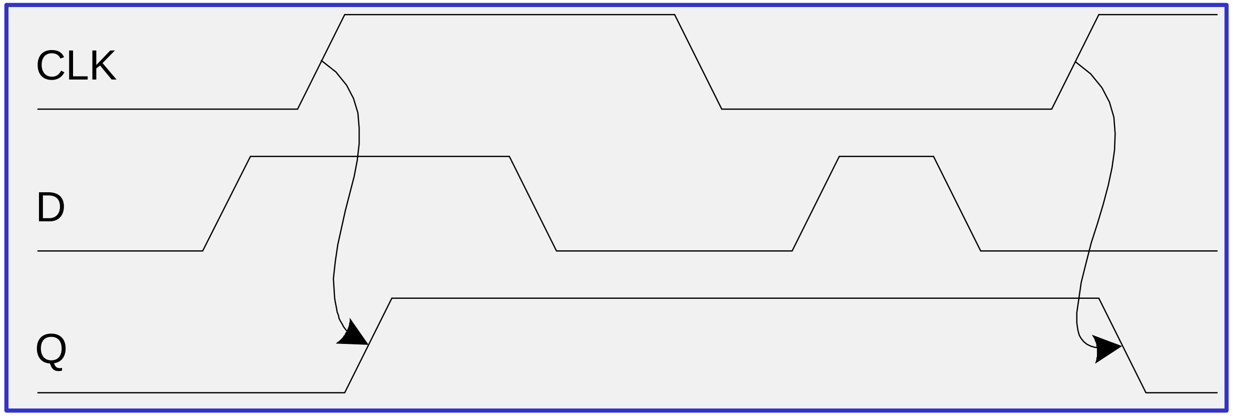


D Flip-flop



clk	D	Q
0	X	no change
1	X	no change
↑	1	1
↑	0	0

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



Number Systems

Decimal ($\textit{base}_{10}$)

$$A = \sum_{i=0}^{n-1} a_i . 10^i$$

10^2	10^1	10^0
1	**5**	**7**

= (1x100)+(5x10)+(7x1)

Binary ($\textit{base}_{2}$)

$$A = \sum_{i=0}^{n-1} a_i . 2^i$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**0**	**0**	**1**	**1**	**1**	**0**	**1**

$=128+ 0+ 0 +16 +8 + 4 + 0 + 1 =157_{10}$

Powers of 2

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$
- $2^{16} = 65536$
- handy to memorize up to 2^{10}

Range of Binary Numbers

- *N*-digit decimal number
  - How many values?
  - Range?
  - Example: 3-digit decimal number:
- *N*-bit binary number
  - How many values?
  - Range:
  - Example: 3-bit binary number:

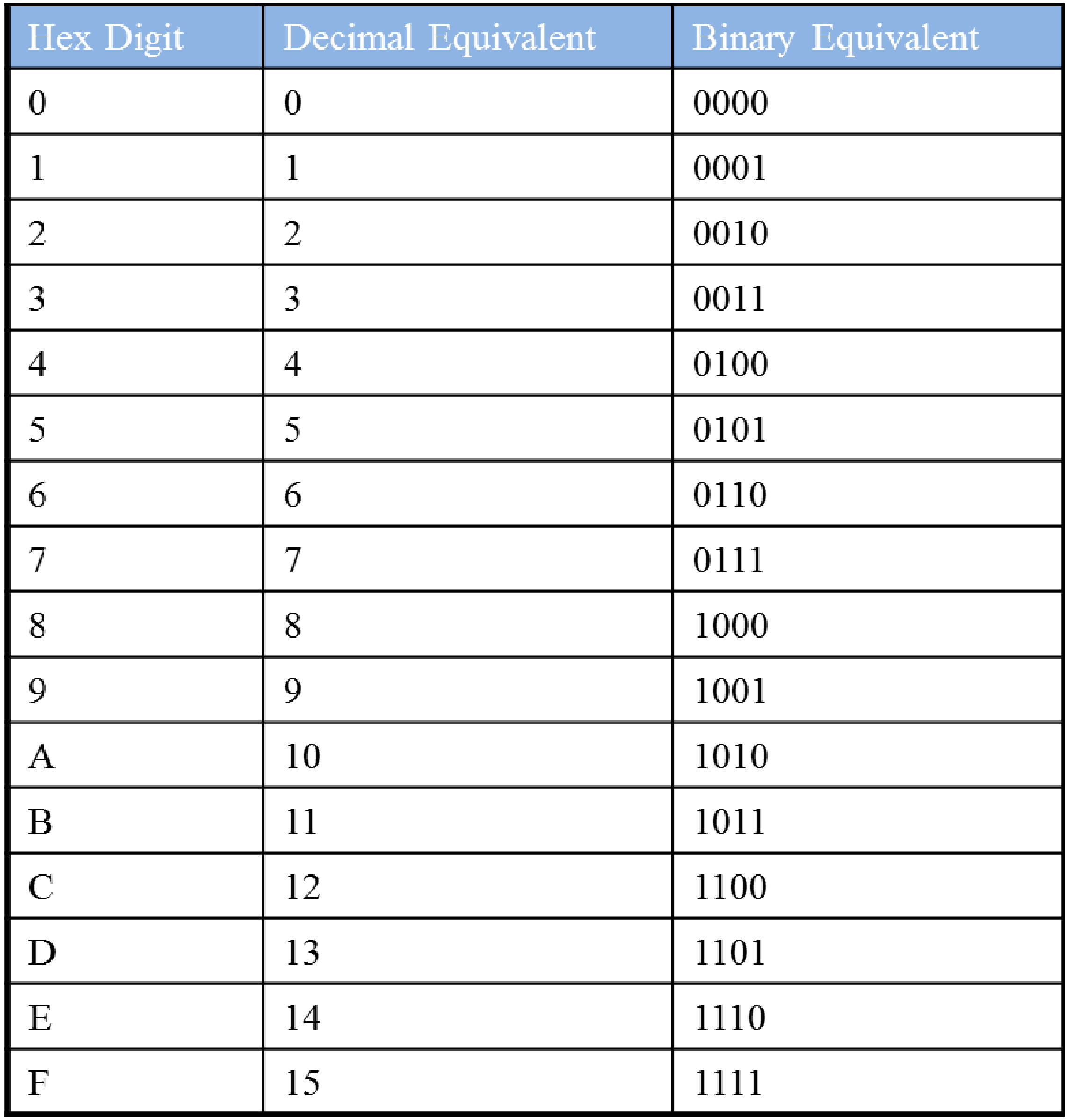
Hexadecimal Numbers

- For humans, its clumsy to always work in binary
  - just too many bits!
- Divide a binary number into 4-bit groupings and represent each 4-bits by a single hexadecimal (base$_{16}$) digit.

Binary:	0010	1001	0101	0111
Hex:	2	9	5	7

- But, in hexadecimal, each digit can have a value of $0 – 15_{10}$!!
- We need new symbols to represent the values $10_{10} – 15_{10}$
- Use symbols A, B, C, D, E and F

Hexadecimal Numbers



Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

- For example:

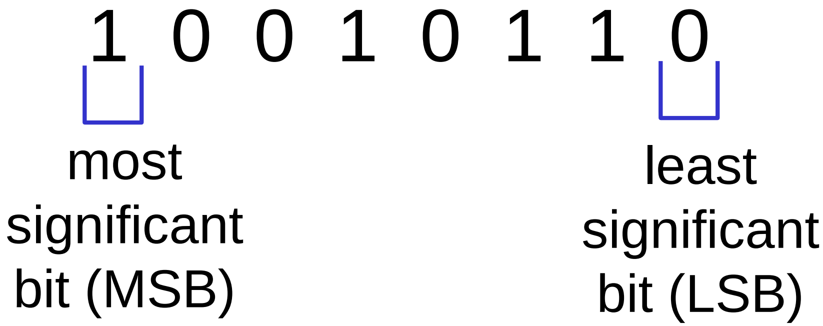
$4AF_{16}$ =

$0100\ 1010\ 1111_2$

$$\begin{aligned} = \quad & (4 \text{ x } 256) \\ & + (10 \text{ X } 16) \\ & + (15 \text{ x } 1) \\ = \quad & 1199_{10} \end{aligned}$$

Bits, Bytes and Nibbles…

- Bits:
  (8-bit binary)

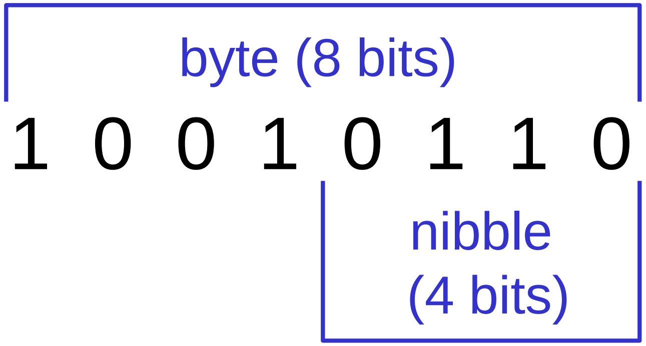


1 0 0 1 0 1 1 0

most significant bit (MSB)

least significant bit (LSB)

- Bytes & Nibbles:
  (8-bit binary)

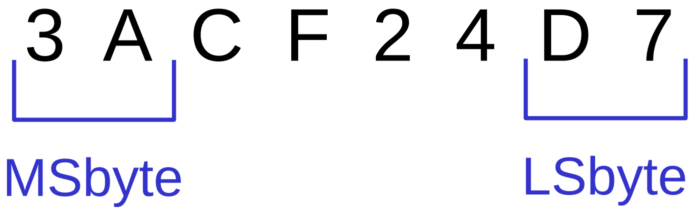


byte (8 bits)

1 0 0 1 0 1 1 0

nibble (4 bits)

- Bytes:
  (32-bit hex)



3 A C F 2 4 D 7

MSbyte

LSbyte

Addition

- Decimal:

$$\begin{array}{r} 3\ 7\ 3\ 4 \\ +\ \underline{5\ 1\ 6\ 8} \end{array}$$

- Binary:

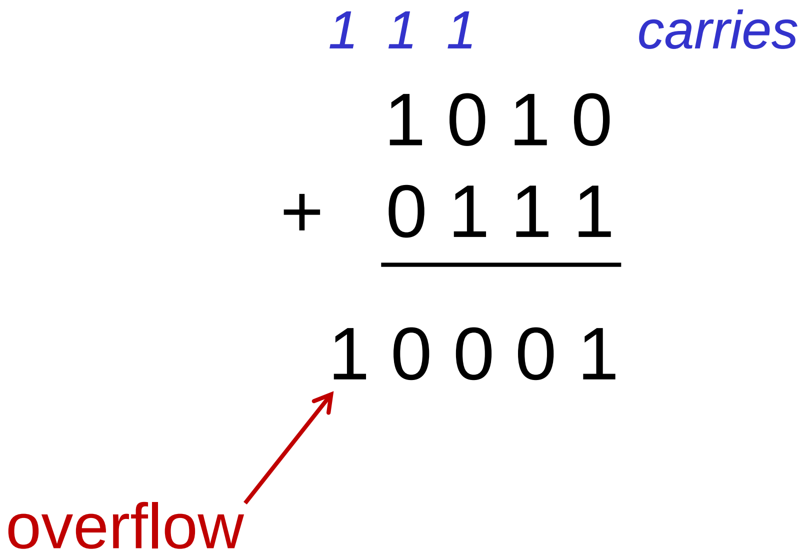
$$\begin{array}{r} 1\ 0\ 1\ 1 \\ +\ \underline{0\ 0\ 1\ 1} \end{array}$$

- Hex:

$$\begin{array}{r} 1\ A\ 3\ 7 \\ +\ \underline{0\ 9\ F\ 6} \end{array}$$

Overflow

- Note that if we add two n-bit numbers, we will (in general) get an (n+1) bit result:



1 1 1 *carries*

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

overflow

Signed Binary Representation

How do we deal with negative numbers?

Two common approaches:

- Sign-magnitude representation
- Two’s complement representation

Sign-Magnitude Representation

- One sign bit plus n-1 magnitude bits
- MSBit is the sign bit:
  - MSB=0 means positive number
  - MSB=1 means negative number

$$A = (-1)^{a^{n-1}} \times \sum_{i=0}^{n-2} a_i . 2^i$$

- *for example, for n=8:*

0 0 0 1 0 1 1 1

= +1 x (0 + 0 +16 +0 + 4 + 2 + 1) = 23

1 0 0 1 0 1 1 1

= –1 x (0 + 0 +16 +0 + 4 + 2 + 1) = – 23

- n-bit sign-magnitude number can take on values $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

Problems with Sign-Magnitude

1. Addition doesn't work
    - for example, 4-bit addition of $(-5)\ and\ (+2)$

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ +\ 0\ 0\ 1\ 0 \\ \hline 1\ 1\ 1\ 1 \end{array} \quad = -7_{10}\ (incorrect)$$

2. Two representations of zero (± 0):

0 0 0 0

1 0 0 0

Two’s Complement Representation

- MSBit has value (-2^{n-1}) :

$$A = -(a_{n-1}.\,2^{n-1}) + \sum_{i=0}^{n-2} a_i.\,2^i$$

- *for example, n=8:*

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	**0**	**0**	**1**	**0**	**1**	**1**	**1**

= 0 + 0 + 0 +16 +0 + 4 + 2 + 1 = 23

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**

= -128 +64+32 +0 +8 + 0 + 0 + 1 = – 23

- n-bit two’s complement number can take on values (-2^{n-1}) to $(2^{n-1}-1)$

Two’s Complement

- To form two’s complement (i.e. flip the sign) of number A, *either*

- Working from LSB to MSB, complement (invert) all bits after (to the left of) first ‘1’:
  - e.g. A = 0101 (= 5)

  complementing all bits to left of first ‘1’ (occurs at bit 0):

  $-A = 1011$ (= -5)

  OR

- Invert all bits in A and add 1:

$$-A = \overline{A} + 1 = 1010 + 1 = 1011 \quad (= -5)$$

Convenience of Two's Complement

1. MSB still indicates sign

2. Addition *does* work

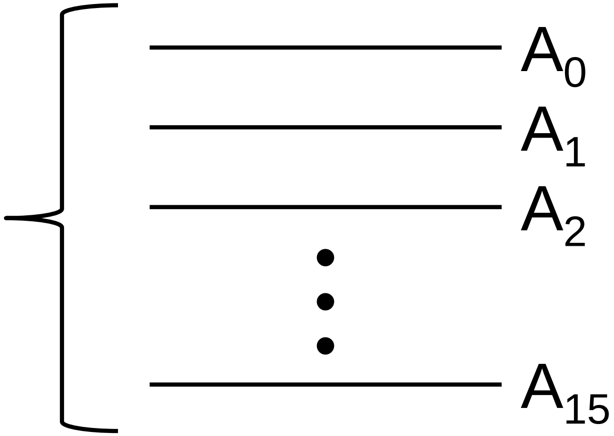
$$\begin{array}{r} 1\ \ 0\ \ 1\ \ 1 \\ +\ 0\ \ 0\ \ 1\ \ 0 \\ \hline 1\ \ 1\ \ 0\ \ 1 \end{array} \qquad \begin{array}{l} -\,5_{10} \\ +\,2_{10} \\ \hline -3_{10}\ (\mathit{correct}!) \end{array} \qquad\qquad \begin{array}{r} 1\ \ 0\ \ 1\ \ 1 \\ +\ 0\ \ 1\ \ 1\ \ 1 \\ \hline 1\ \ 0\ \ 0\ \ 1\ \ 0 \end{array} \qquad \begin{array}{l} -\,5_{10} \\ +\,7_{10} \\ \hline +2_{10}\ (\mathit{correct}!) \end{array}$$

note: throw away
the “overflow” bit

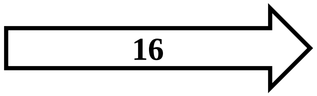
3. Only one representation of zero: 0 0 0 0

Busses

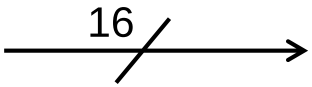
- Frequently useful to group a number of signals into a group as a bus:
  - e.g. A is a 16-bit bus:



  - represented as



or



- Bus may carry a binary value (with a LSB and a MSB)
- Or just a collection of non-numerically related bits
  - e.g. binary instruction

Registers

- When we want to “remember” an N-bit value…
  - may be numerical value, instruction, code, address etc.
- We often group N D-flip-flops together to capture and store the value on the rising edge of a common clock
- We call this an N-bit register
  - e.g. 16-bit register