01 Introduction to Digital Systems

- What are digital systems?
- Digital functions
- Truth tables and logic symbols
- Analysis of simple logic networks
- Timing diagram

What are digital systems?

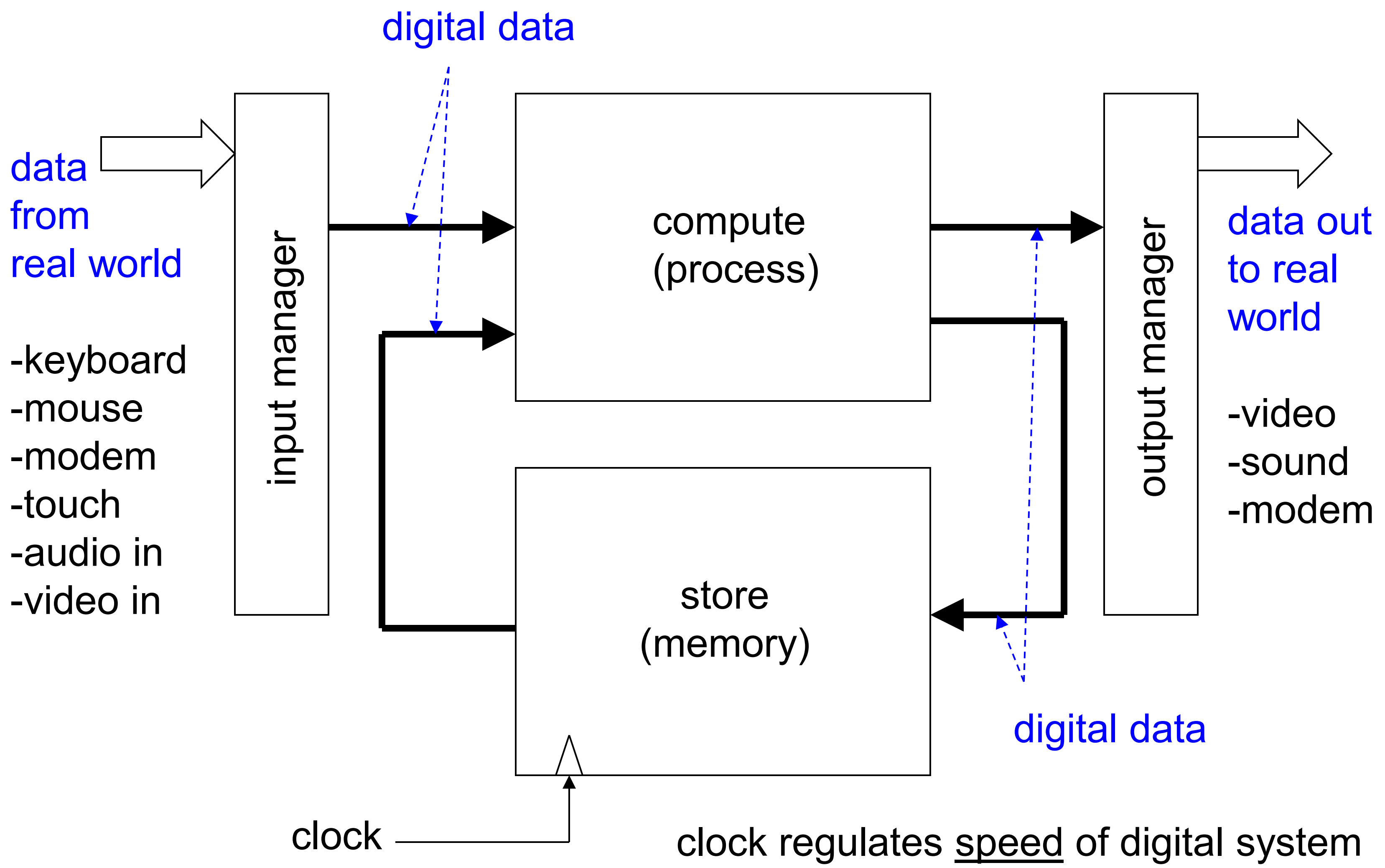
- Systems that store and process digital data.
- Digital data : two possible states/values

  0 = FALSE (F) = Low voltage (L)

  1 = TRUE (T) = High voltage (H)

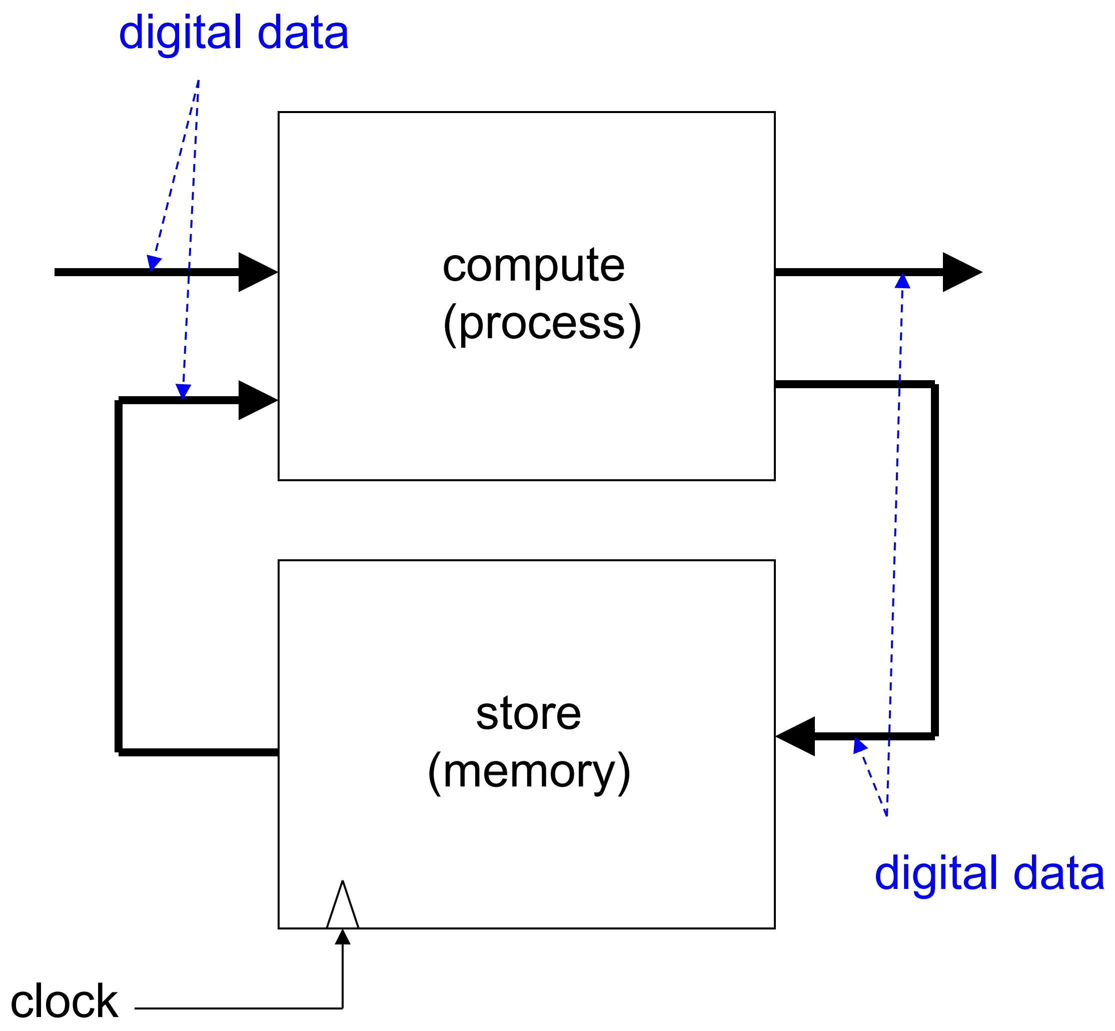
  also called binary data.
- Modern implementation use voltages to indicate logic 0 (Ground) or logic 1 (+V).

Organization of a digital system:

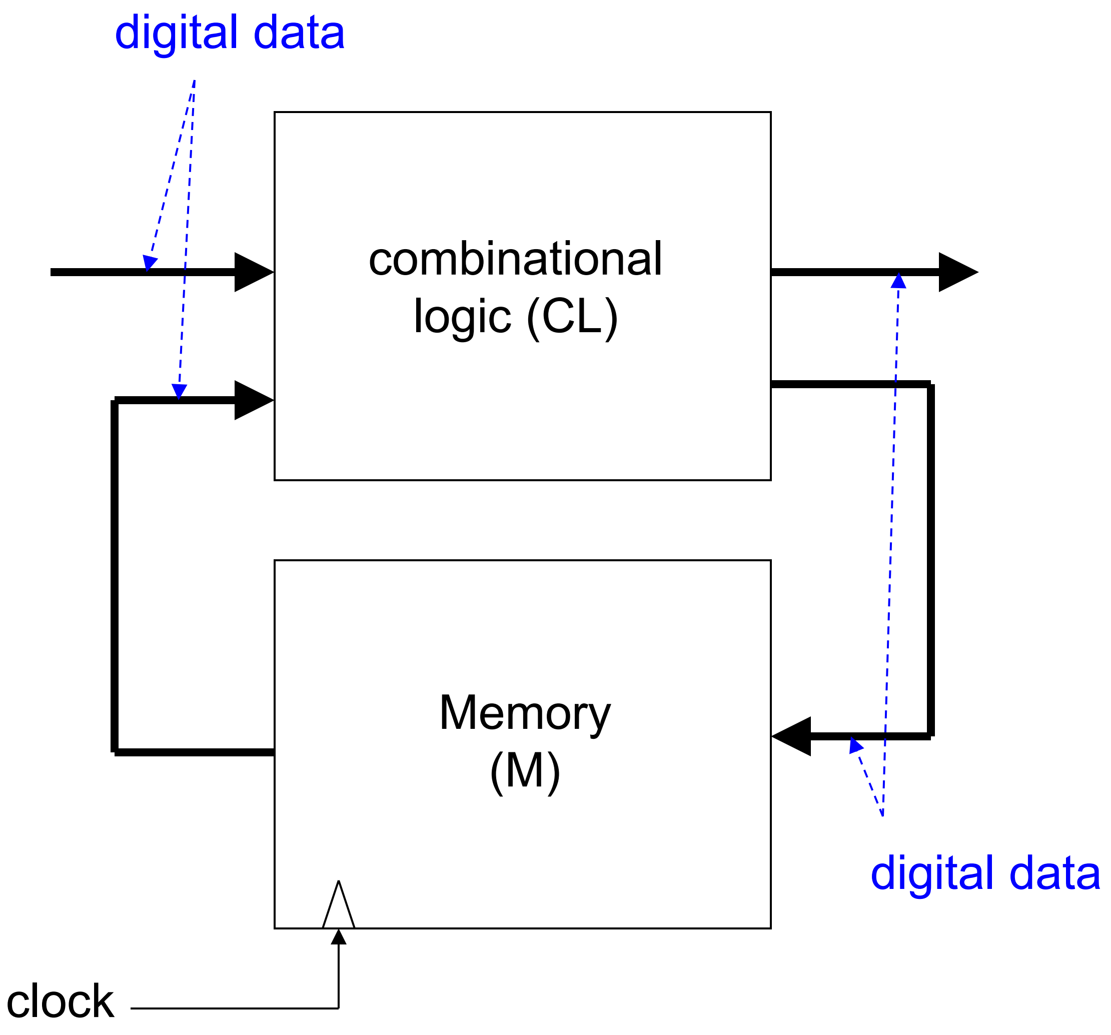


clock regulates speed of digital system

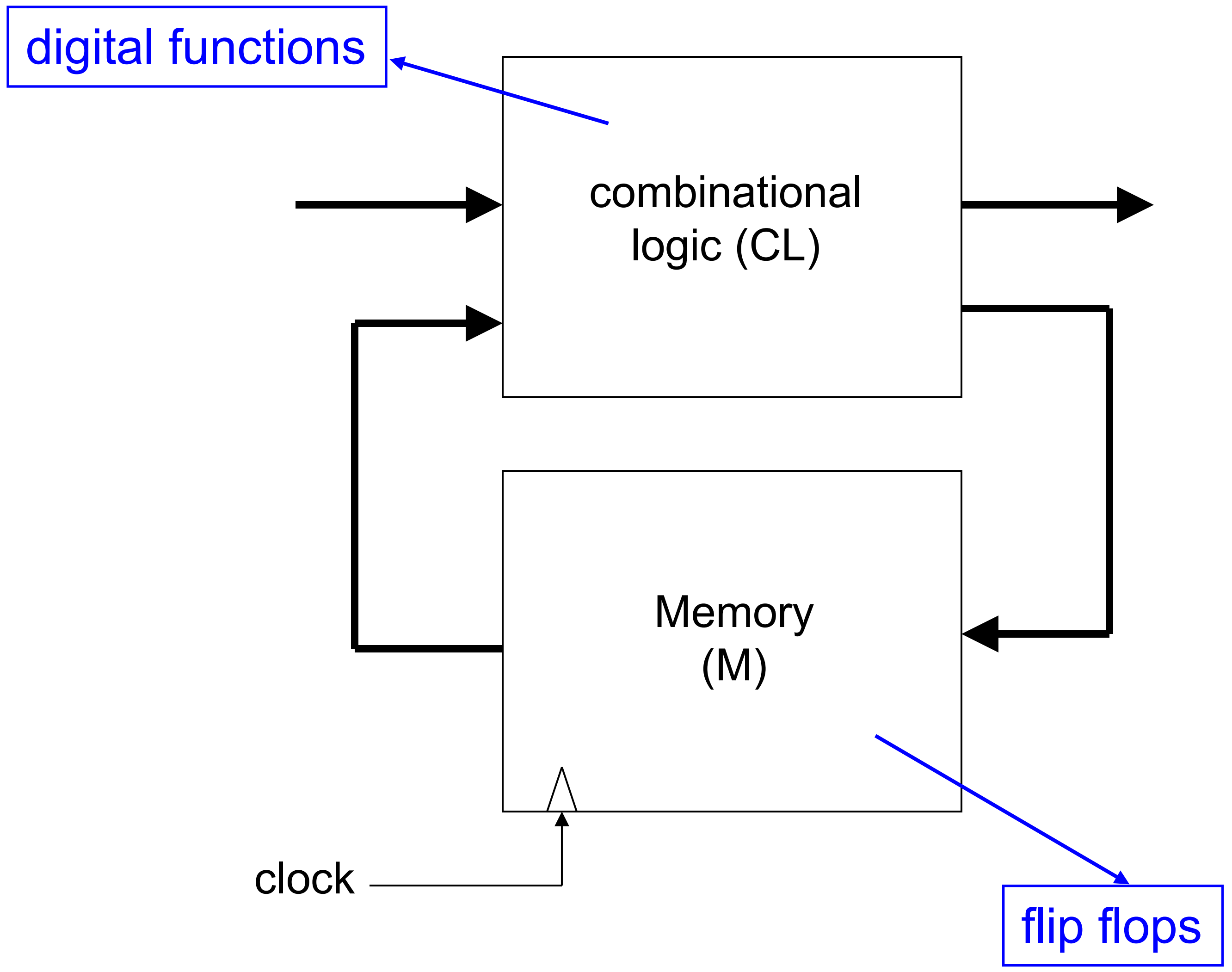
Organization of a digital system:



Organization of a digital system:



Organization of a digital system:



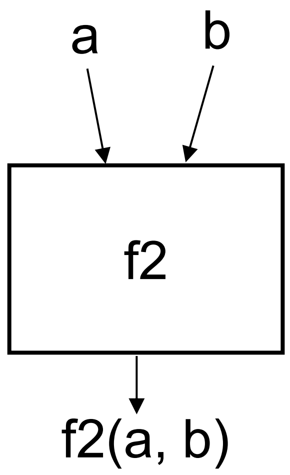
Digital functions (binary functions):

Binary numbers and binary function:

$$f2(a, b) = ab$$

a, b are variables and f2 a function.

f2 transforms binary numbers to binary num.

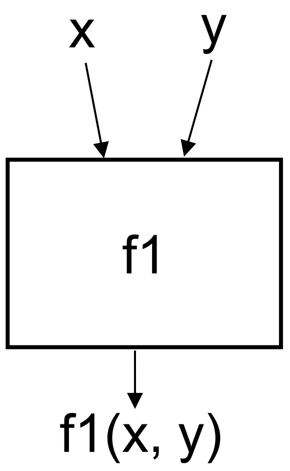


DIAGRAM

$$f2(a, b) = ab$$

EQUATION

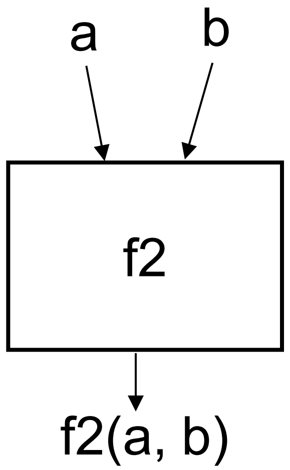
Digital functions (binary functions):



real-valued function

$f1(x, y) = x^2 + y^2 + xy + 2$

infinitely many (x, y) possible

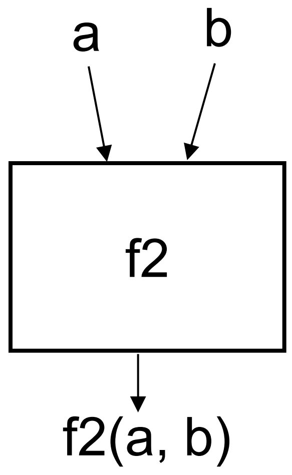


binary function

$f2(a, b) = ab$

only 4 possibilities
ab = 00, 01, 10, 11

Digital functions (binary functions):



binary function

f2(a, b) = ab

only 4 possibilities
ab = 00, 01, 10, 11

a	b	f2
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table
A table that lists all possible inputs to a function and the corresponding output

Truth table completely describes the function

Some famous functions:

AND

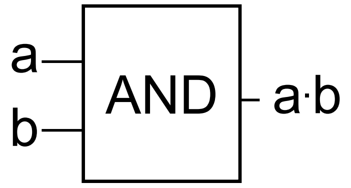
a	b	f = a·b
0	0	0
0	1	0
1	0	0
1	1	1

TRUTH TABLE

use dot (.) or nothing for AND

$f = a \cdot b$

EQUATION



DIAGRAM

OR

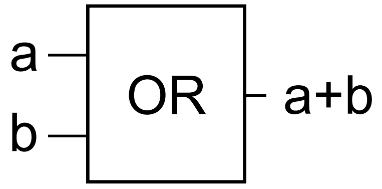
a	b	f = a+b
0	0	0
0	1	1
1	0	1
1	1	1

TRUTH TABLE

use plus (+) for OR

$f = a+b$

EQUATION



DIAGRAM

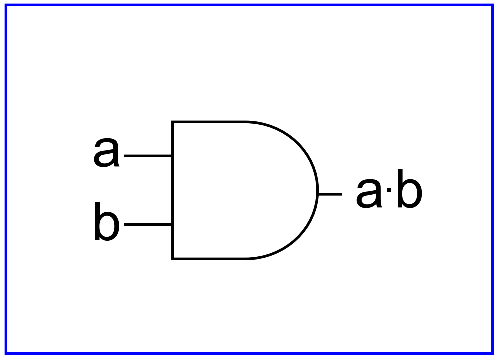
Some famous functions:

AND

a	b	f = a·b
0	0	0
0	1	0
1	0	0
1	1	1

$f = a \cdot b$

AND gate symbol

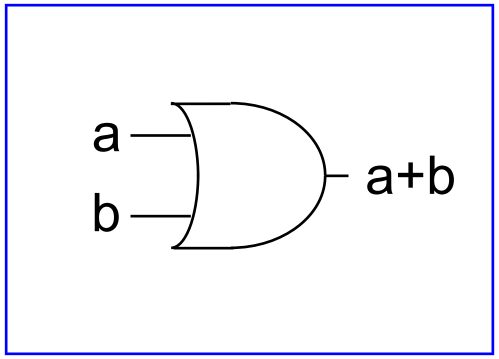


OR

a	b	f = a+b
0	0	0
0	1	1
1	0	1
1	1	1

$f = a+b$

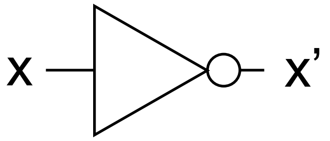
OR gate symbol



NOT (invert) function:

input x can be either 0 or 1

x	NOT x
0	1
1	0



SO MANY equation symbols for NOT:

- **We’ll use them all interchangeably**

Symbols for NOT x:

$\overline{x}$ (x bar)
x’ (x prime)

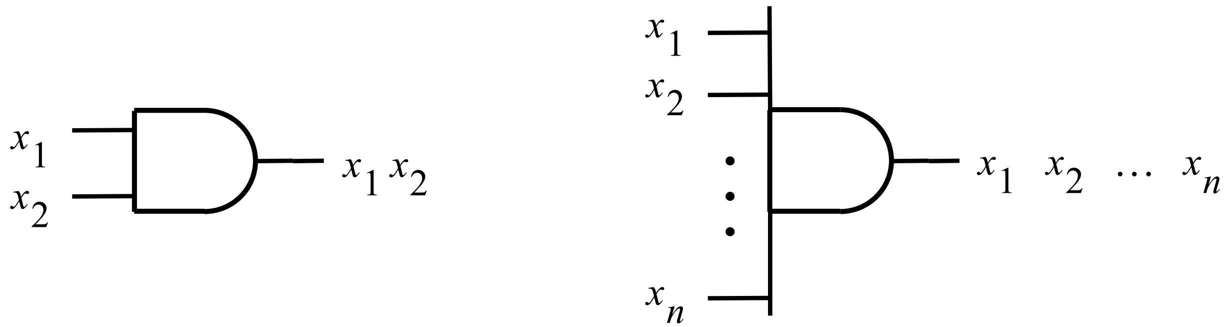
textbooks
lectures
homeworks

~x (tilde x)
!x (not x)
/x (slash x)

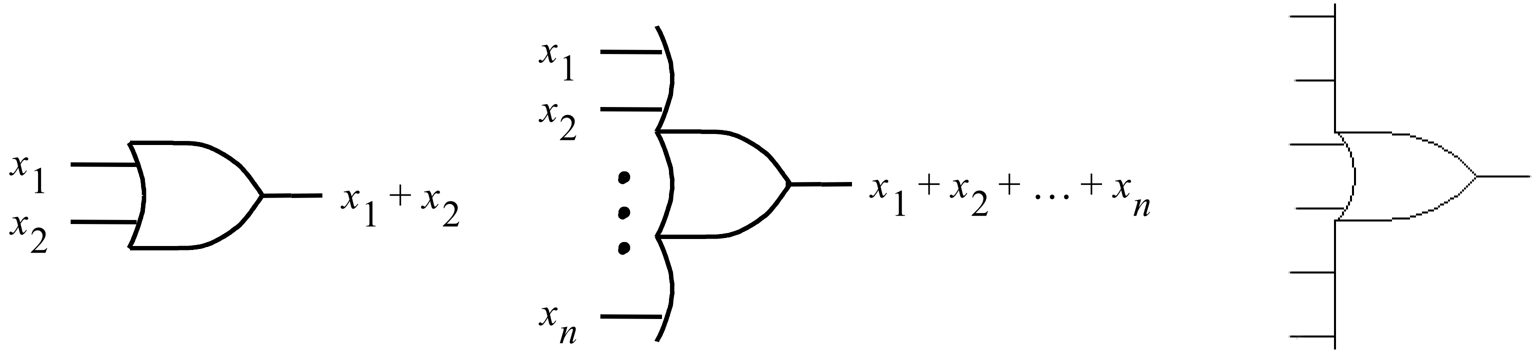
computer
programs

Gate symbols: **gate** = computing element

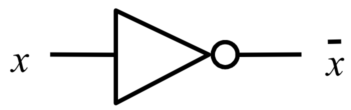
[BV]



------------------------------- AND gates ----------------------------



----------------------------- OR gates --------------------------------------



------------- NOT gate (also called an INVERTER) -----------------

Even more gates:

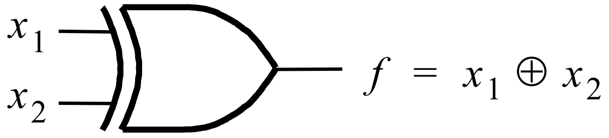
XOR: (exclusive-or) → why the name?

symbol for xor is circled plus or ^

x_1	x_2	$f = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

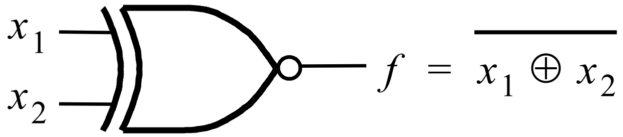
Truth table

[BV]



Graphical symbol

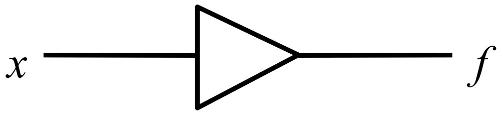
XNOR: XNOR is XOR followed by a NOT



Even more gates 2:

Buffer or Follower (BUF): output = input

[BV]



Graphical symbol

Truth table

x	f
0	0
1	1

Tri-state buffers:

- Sometimes we want to momentarily disconnect or disable output of a gate.

- When gate is disabled, it no longer gives out voltage, hence no longer supplying logic level. We call this state float, high-impedance, in-tri-state, high-Z, Z.

- Called tri-state because output can be 0, 1, or Z. (3 possible values)

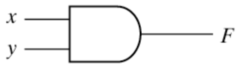
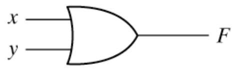
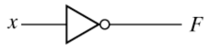
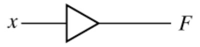
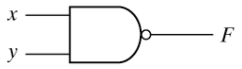
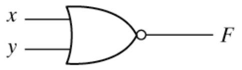
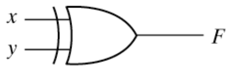
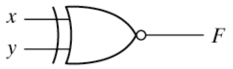
Tri-state buffers:

- They are buffers which have enable input
- When enable is on, they act like buffers.
- When enable is off, their outputs are disconnected from the buffer output

enable	input	output
0	0	Z
0	1	Z
1	0	0
1	1	1

⇦ tri-state buffer truth table

Gate symbol list:

Name	Graphic symbol	Algebraic function	x	y	F
AND		$F = xy$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$F = x + y$	0	0	0
			0	1	1
			1	0	1
			1	1	1
Inverter		$F = x'$	0		1
			1		0
Buffer		$F = x$	0		0
			1		1
NAND		$F = (xy)'$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$F = (x + y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	1

[Mano]

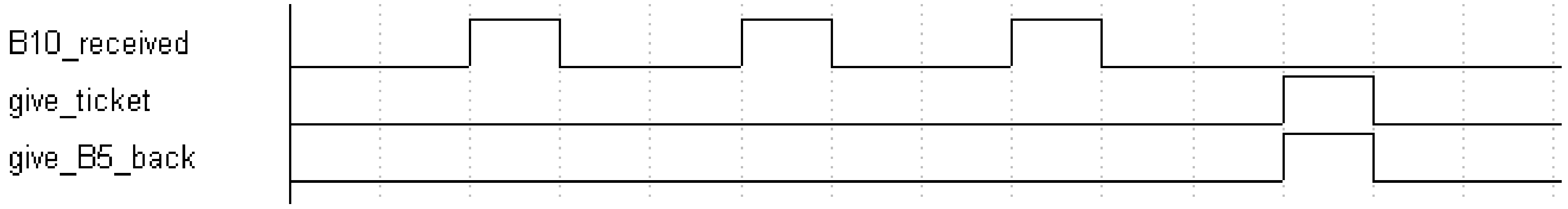
Fig. 2-5 Digital logic gates

Viewing Digital Data

Digital Signals (waveforms / timing diagram)

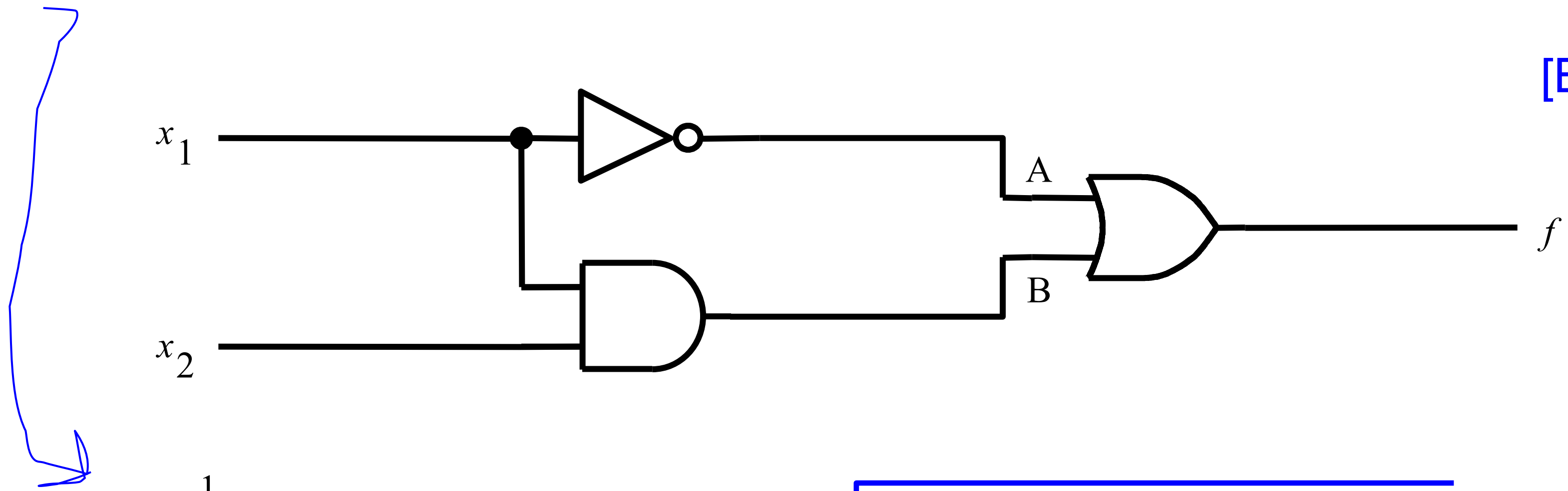
- signal: a function of time
- digital signal: value of function is 0 or 1
  0 = low, 1 = high

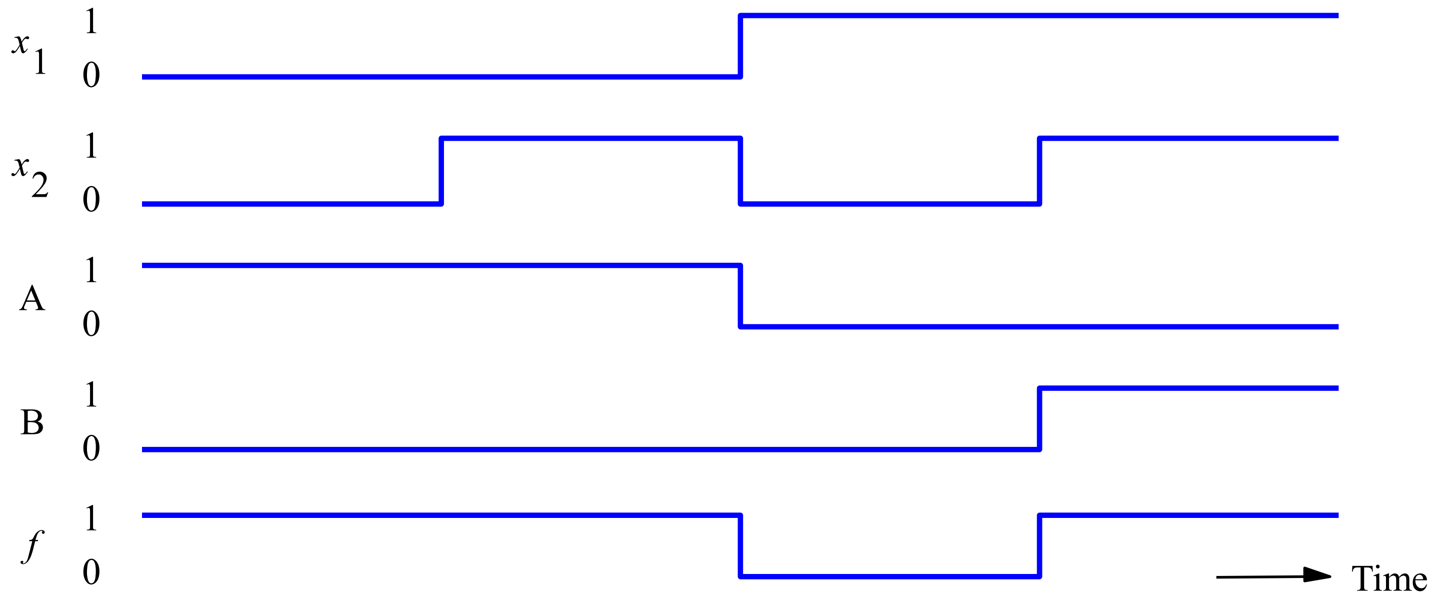
Example: Buying a 25-Baht BTS Ticket



Timing diagram / waveforms / signals:

[BV]

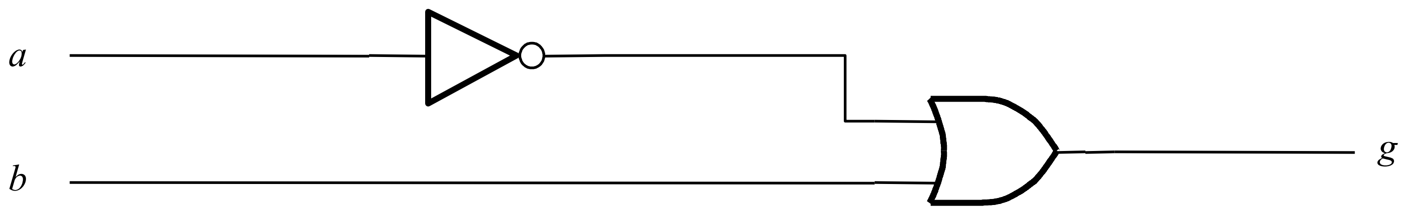




Timing diagram

What does the following implement?

[BV]



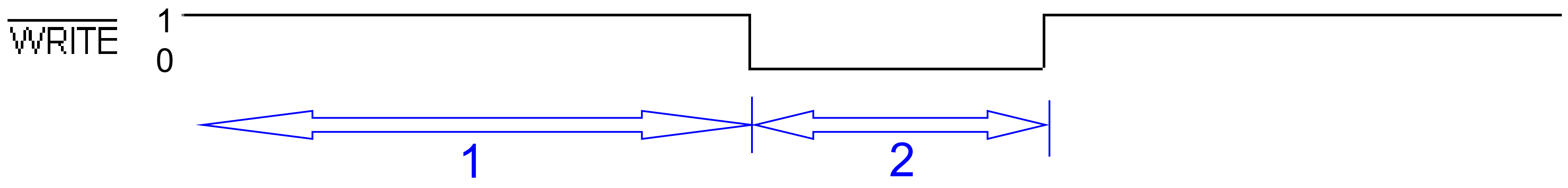
by following gates: g = a' + b

by listing all combinations:

a	b	g
0	0	1
0	1	1
1	0	0
1	1	1

How to read “NOT” or inverted signals:

Example: $\overline{\text{WRITE}}$ signal is generated



Bad interpretation:

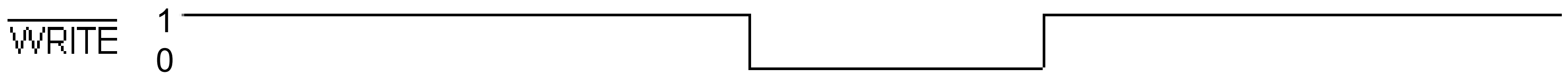
1. NOT WRITE is 1, so it is TRUE, so it is not writing.
2. NOT WRITE is 0, so it is FALSE, so it is not not writing, so it is not-not write, so it is writing, yes? no? maybe?

Good interpretation:

$\overline{\text{WRITE}}$ signal is asserted (is true) when the logic is 0.
This is called an *asserted low* or *active low* signal.
So, $\overline{\text{WRITE}}$ means writing occurs when $\overline{\text{WRITE}}$ is low (0).

Names for inverted signals:

Example: $\overline{\text{WRITE}}$ signal is generated



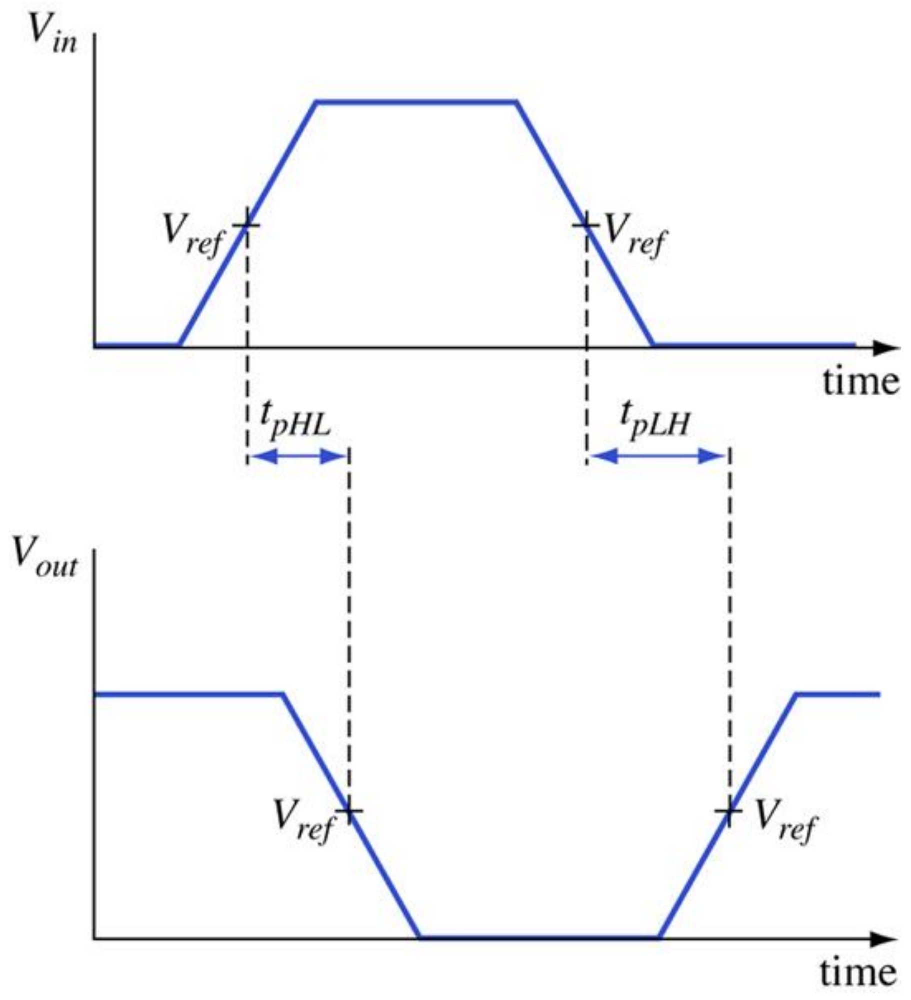
What to call $\overline{\text{WRITE}}$ when it is asserted (0):

- $\overline{\text{WRITE}}$ is *asserted*
- $\overline{\text{WRITE}}$ is *active*

What to call WRITE when it is not asserted (1):

- $\overline{\text{WRITE}}$ is *de-asserted* or *deasserted*
- $\overline{\text{WRITE}}$ is *inactive*
- $\overline{\text{WRITE}}$ is *negated*

Real gates have propagation delay:



[Source: Givone (G)]

Input and output signals at an inverter (NOT gate):
→ When $V_{in} = 1$, $V_{out} = 0$
→ When $V_{in} = 0$, $V_{out} = 1$
but…
V_{out} does not respond to change in V_{in} immediately

The time it takes to respond to change in input is called propagation delay.

t_{pHL} = high→low prop. delay
t_{pLH} = low→high prop. delay

HL or LH? → Look at output!

<u>Logic-only design</u>:

When we first design or analyze the circuit in this class, we will assume that propagation delay is zero. <u>Focus on logic</u>.

We will worry about propagation delay later.

More on how to get real propagation delay information later.

Propagation Delay (cont):

- Real circuits have propagation delay

→ Real circuits takes time to compute something. Changes don’t happen instantaneously.

More complex computation = longer time

- t_{pLH} may not be equal to t_{pHL}

Good designs reduce the larger one, even at the expense of smaller one (why later)

Noise and noise margins:

→Real circuits have noise.

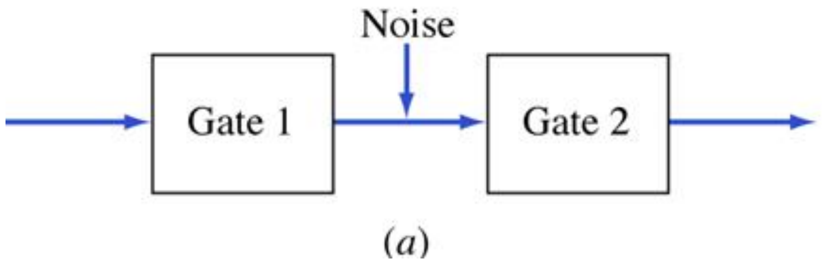
→Real gates have noise margins.

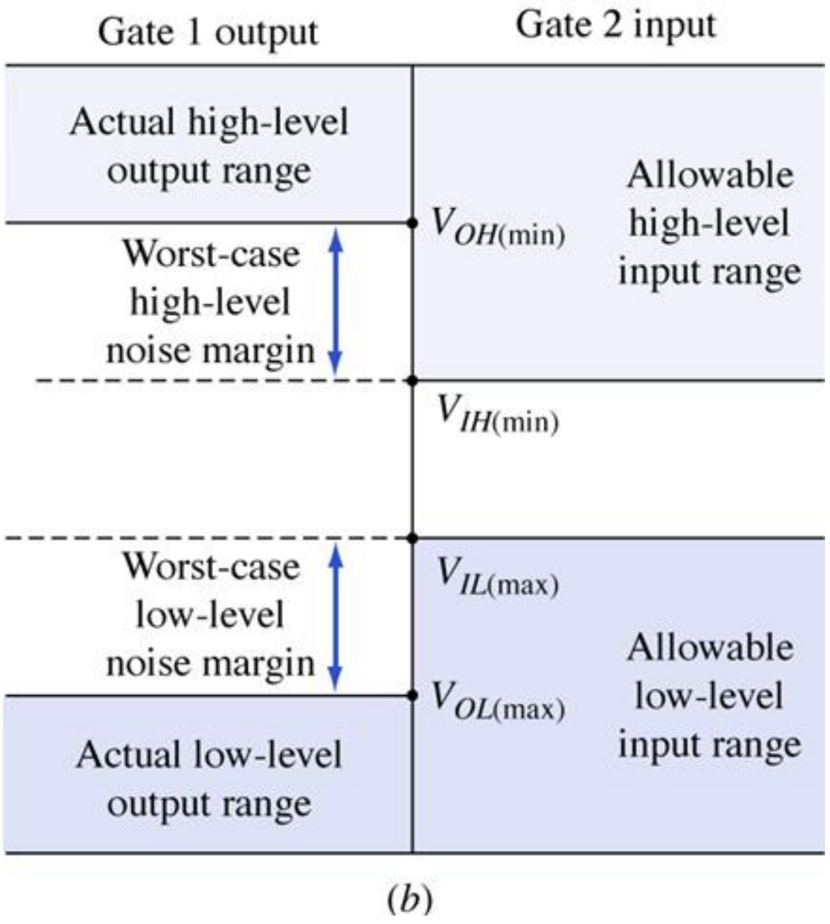
[G]

Sources of noise:

1. EMI from other gates
2. Noisy power supply
3. Cosmic ray (really!)

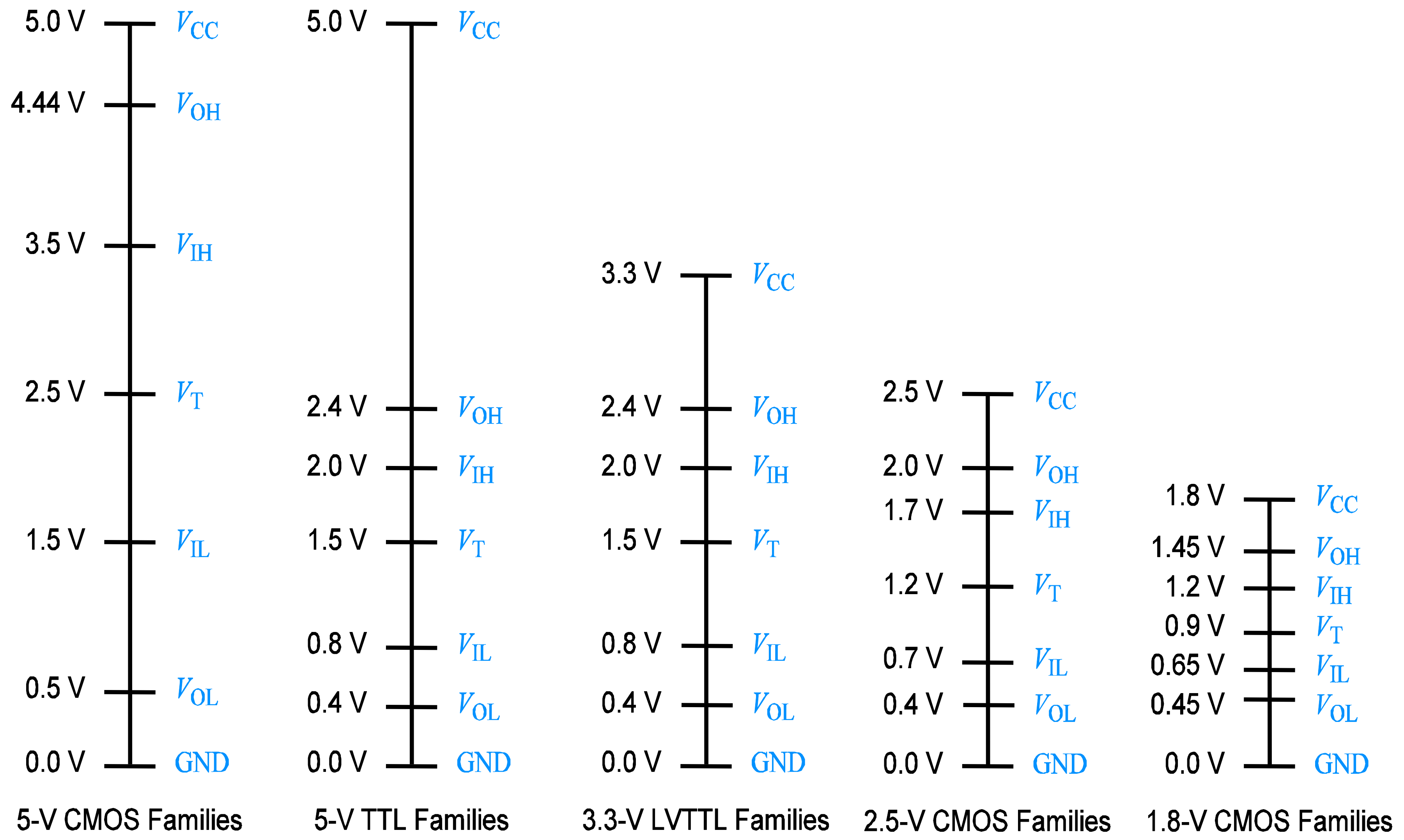
etc.




Gate 1 output
Gate 2 input
Actual high-level output range
Worst-case high-level noise margin
$V_{OH(\min)}$
Allowable high-level input range
$V_{IH(\min)}$
Worst-case low-level noise margin
$V_{IL(\max)}$
$V_{OL(\max)}$
Allowable low-level input range
Actual low-level output range
(b)

Real values of V_{IL} V_{OL} V_{IH} V_{OH}:

[W]



Exercise: Calculate low/high noise margins for some technologies

Logic-only design:

We will not cover the topic of noise and interference in this class.

It is a very big topic - enough for several more courses. -- topic is called signal integrity.