

COMP110: Principles of Computing

# **4: Logic and memory**

# Learning outcomes

- ▶ **Distinguish** the basic types of logic gate
- ▶ **Use** logic gates to build simple circuits
- ▶ **Explain** how computer memory works

# Logic gates

# Boolean logic

- ▶ Works with two values: TRUE and FALSE
- ▶ Foundation of the **digital computer**: represented in circuits as **on** and **off**
- ▶ Representing as 1 and 0 leads to **binary notation**
- ▶ One boolean value = one **bit** of information
- ▶ Programmers use boolean logic for conditions in **if** and **while** statements

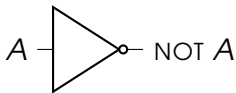
# Simulating logic circuits

`http://logic.ly/demo/`

# Not

NOT  $A$  is TRUE  
if and only if  
 $A$  is FALSE

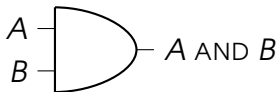
$A$	NOT $A$
FALSE	TRUE
TRUE	FALSE



# And

$A$  AND  $B$  is TRUE  
if and only if  
**both**  $A$  **and**  $B$  are TRUE

$A$	$B$	$A$ AND $B$
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE



# Or

$A$  OR  $B$  is TRUE  
if and only if  
**either**  $A$  **or**  $B$ , **or both**, are TRUE

$A$	$B$	$A$ AND $B$
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE





# Socratic FALCOMPED

What is the value of

$A \text{ AND } (B \text{ OR } C)$

when

$A = \text{TRUE}$

$B = \text{FALSE}$

$C = \text{TRUE}$

?

# Socratic FALCOMPED

What is the value of

$(\text{NOT } A) \text{ AND } (B \text{ OR } C)$

when

$A = \text{TRUE}$

$B = \text{FALSE}$

$C = \text{TRUE}$

?

# Socratic FALCOMPED

For what values of  $A, B, C, D$  is

$$A \text{ AND NOT } B \text{ AND NOT } (C \text{ OR } D) = \text{TRUE}$$

?

# Socratic FALCOMPED

What is the value of

$A \text{ OR NOT } A$

?

# Socratic FALCOMPED

What is the value of

$A \text{ AND NOT } A$

?

# Socratic FALCOMPED

What is the value of

$A \text{ OR } A$

?

# Socratic FALCOMPED

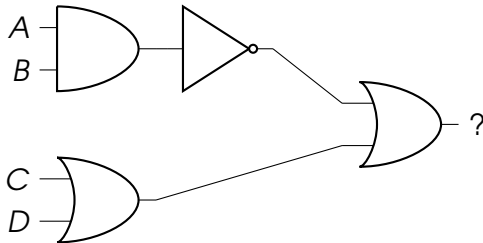
What is the value of

$A \text{ AND } A$

?

# Socratic FALCOMPED

What expression is equivalent to this circuit?





# Writing logical operations

Operation	Python	C family	Mathematics
NOT $A$	<b>not</b> $a$	$!a$	$\neg A$ or $\overline{A}$
$A$ AND $B$	$a$ <b>and</b> $b$	$a \ \&\& \ b$	$A \wedge B$
$A$ OR $B$	$a$ <b>or</b> $b$	$a \    \ b$	$A \vee B$

Other operators can be expressed by combining these

# De Morgan's Laws

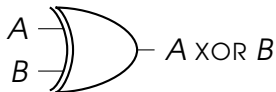
$$\text{NOT } (A \text{ OR } B) = (\text{NOT } A) \text{ AND } (\text{NOT } B)$$

$$\text{NOT } (A \text{ AND } B) = (\text{NOT } A) \text{ OR } (\text{NOT } B)$$

# Exclusive Or

$A \text{ XOR } B$  is TRUE  
if and only if  
**either**  $A$  **or**  $B$ , **but not both**, are TRUE

$A$	$B$	$A \text{ AND } B$
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE



# Socratic FALCOMPED

How can  $A \text{ XOR } B$  be written using the operations  
AND , OR , NOT ?

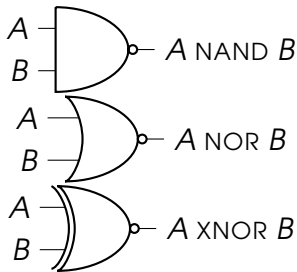
# Negative gates

NAND , NOR , XNOR  
are the **negations** of  
AND , OR , XOR

$$A \text{ NAND } B = \text{NOT } (A \text{ AND } B)$$

$$A \text{ NOR } B = \text{NOT } (A \text{ OR } B)$$

$$A \text{ XNOR } B = \text{NOT } (A \text{ XOR } B)$$



**Binary notation**

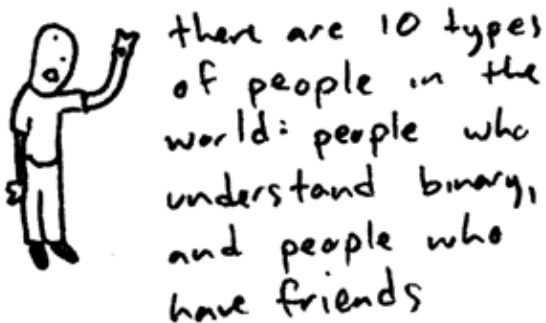


Image credit: <http://www.toothpastefordinner.com>

# How we write numbers

- ▶ We write numbers in **base 10**
- ▶ We have 10 **digits**: 0, 1, 2, ..., 8, 9
- ▶ When we write 6397, we mean:
  - ▶ Six thousand, three hundred and ninety seven
  - ▶ (Six thousands) and (three hundreds) and (nine tens) and (seven)
  - ▶  $(6 \times 1000) + (3 \times 100) + (9 \times 10) + (7)$
  - ▶  $(6 \times 10^3) + (3 \times 10^2) + (9 \times 10^1) + (7 \times 10^0)$



# Binary

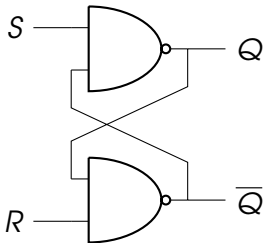
- ▶ Binary notation works the same, but is **base 2** instead of **base 10**
- ▶ We have 2 **digits**: 0, 1
- ▶ When we write 10001011 in binary, we mean:
$$\begin{aligned}& (1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) \\& + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\& = 2^7 + 2^3 + 2^1 + 2^0 \\& = 128 + 8 + 2 + 1 \text{ (base 10)} \\& = 139 \text{ (base 10)}\end{aligned}$$

# Bits, bytes and words

- ▶ A **bit** is a binary digit
  - ▶ Can store a 0 or 1 (i.e. a boolean value)
- ▶ A **byte** is 8 **bits**
  - ▶ Can store a number between 0 and 255 in binary
- ▶ A **word** is the number of bits that the CPU works with at once
  - ▶ 32-bit CPU: 32 bits = 1 word
  - ▶ 64-bit CPU: 64 bits = 1 word
- ▶ An  $n$ -bit word can store a number between 0 and  $2^n - 1$ 
  - ▶  $2^{16} - 1 = 65,535$
  - ▶  $2^{32} - 1 = 4,294,967,295$
  - ▶  $2^{64} - 1 = 18,446,744,073,709,551,615$

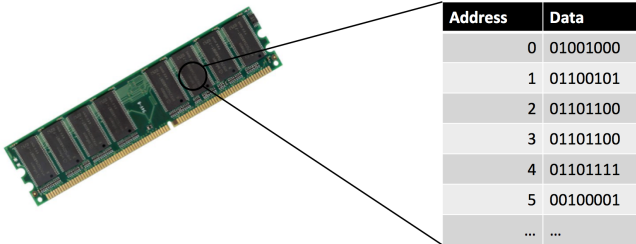
# Computer memory

# What does this circuit do?



- ▶ This is called a **NAND latch**
- ▶ It “remembers” a single boolean value
- ▶ Put a few billion of these together (along with some control circuitry) and you’ve got **memory**!

# Memory



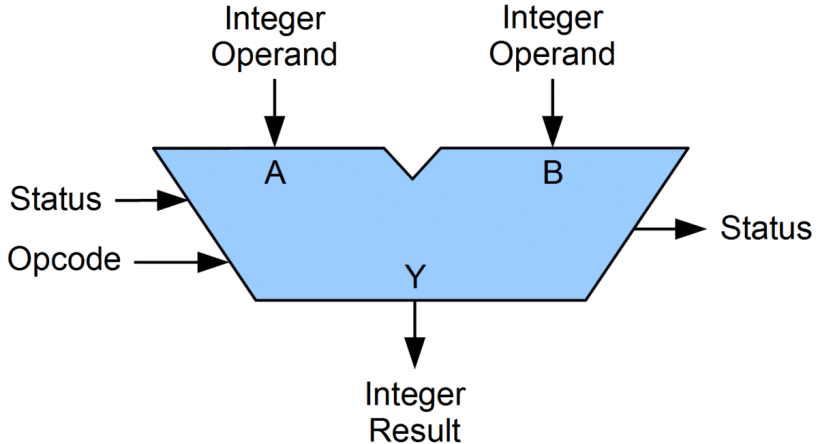
- ▶ Memory works like a set of **boxes**
- ▶ Each box has a number, its **address**
- ▶ Each box contains a **byte** (8 bits)

# Data representation

- ▶ Memory stores **sequences of numbers**
- ▶ Therefore, any data stored by a computer must be represented as a sequence of numbers
  - ▶ Text: sequence of ASCII (or Unicode etc) character codes
  - ▶ Image: sequence of pixel colour values
  - ▶ 3D model: sequence of vertex coordinates
  - ▶ Audio: sequence of displacements
  - ▶ Executable: sequence of machine code operations

# **Arithmetic Logic Unit**

# Arithmetic Logic Unit





# Arithmetic Logic Unit

- ▶ Important part of the CPU
- ▶ Inputs:
  - ▶ **Operand** words  $A, B$
  - ▶ **Opcode**
  - ▶ **Status** bits
- ▶ Outputs:
  - ▶ **Result** word  $Y$
  - ▶ **Status** bits
- ▶ Opcode specifies how  $Y$  is calculated based on  $A$  and  $B$

# ALU operations

Typically include:

- ▶ Add with carry
- ▶ Subtract with borrow
- ▶ Negate (2's complement)
- ▶ Increment, decrement
- ▶ Bitwise AND, OR, NOT, ...
- ▶ Bit shifts

# Addition with carry

In base 10:

$$\begin{array}{rcccc} & 1 & 2 & 3 & 4 \\ + & 5 & 6_1 & 7_1 & 8 \\ \hline & 6 & 9 & 1 & 2 \end{array}$$

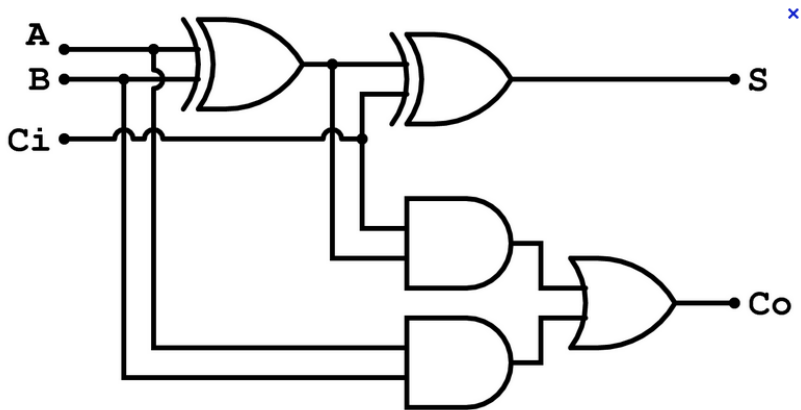
# Addition with carry

In base 2:

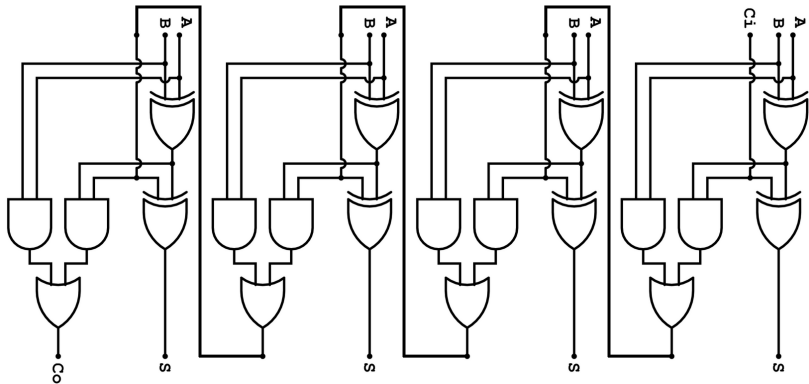
$1 + 1 = 10$	$1 + 1 + 1 = 11$
--------------	------------------

	0	1	1	0	1	1	1	0
+	0 <sub>1</sub>	0 <sub>1</sub>	1	0 <sub>1</sub>	0 <sub>1</sub>	1 <sub>1</sub>	1	1
<hr/>								
	1	0	0	1	0	1	0	1

# 1-bit adder



# $n$ -bit adder



# **Worksheet B**