
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

# Boolean logic

- Works with two values: TRUE and FALSE

# Boolean logic

- Works with two values: TRUE and FALSE
- Foundation of the **digital computer**: represented in circuits as **on** and **off**

# 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**

# 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

# 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

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

# Not

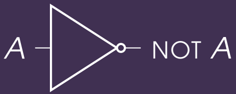
NOT *A* is TRUE
if and only if
*A* is FALSE

| *A* | NOT *A* |
|---|---|
| FALSE | TRUE |
| TRUE | FALSE |

# Not

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

| $A$ | NOT $A$ |
|---|---|
| FALSE | TRUE |
| TRUE | FALSE |



# And

# And

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

# 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 |

# 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

# Or

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

# 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 |

# Socrative `FALCOMPED`

What is the value of

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

when

$$A = \text{TRUE}$$
$$B = \text{FALSE}$$
$$C = \text{TRUE}$$

?

# Socrative FALCOMPED

What is the value of

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

when

$$A = \text{TRUE}$$
$$B = \text{FALSE}$$
$$C = \text{TRUE}$$

?

# Socrative `FALCOMPED`

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

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

?

# Socrative `FALCOMPED`

What is the value of

$$A \text{ OR NOT } A$$

?

# Socrative FALCOMPED

What is the value of

$$A \text{ AND NOT } A$$

?

# Socrative `FALCOMPED`

What is the value of

$$A \text{ OR } A$$

?

# Socrative `FALCOMPED`

What is the value of

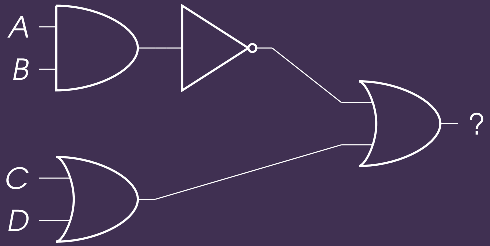
$$A \text{ AND } A$$

?

# Socrative `FALCOMPED`

## (not (A and B)) or (C or D)

What expression is equivalent to this circuit?



# Writing logical operations

# Writing logical operations

| Operation | Python | C family | Mathematics |
|---|---|---|---|
| NOT $A$ | `not a` | `!a` | $\neg A$ or $\overline{A}$ |

# Writing logical operations

| Operation | Python | C family | Mathematics |
|---|---|---|---|
| NOT $A$ | `not a` | `!a` | $\neg A$ or $\overline{A}$ |
| $A$ AND $B$ | `a and b` | `a && b` | $A \wedge B$ |

# Writing logical operations

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

# Writing logical operations

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

Other operators can be expressed by combining these

# De Morgan's Laws

# De Morgan's Laws

$$\textsc{not}\ (A\ \textsc{or}\ B) = (\textsc{not}\ A)\ \textsc{and}\ (\textsc{not}\ B)$$

# De Morgan's Laws

$$\textsc{not}\ (A\ \textsc{or}\ B) = (\textsc{not}\ A)\ \textsc{and}\ (\textsc{not}\ B)$$

$$\textsc{not}\ (A\ \textsc{and}\ B) = (\textsc{not}\ A)\ \textsc{or}\ (\textsc{not}\ B)$$

# Exclusive Or

# Exclusive Or

*A* XOR *B* is TRUE
if and only if
**either** *A* **or** *B*, **but not both,** are TRUE

# Exclusive Or

*A* XOR *B* is TRUE
if and only if
**either** *A* **or** *B*, **but not both,** are TRUE

| *A* | *B* | *A* AND *B* |
|---|---|---|
| FALSE | FALSE | FALSE |
| FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| TRUE | TRUE | FALSE |

# Exclusive Or

*A* XOR *B* is TRUE
if and only if
**either** *A* **or** *B*, **but not both,** are TRUE

| *A* | *B* | *A* AND *B* |
|---|---|---|
| FALSE | FALSE | FALSE |
| FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| TRUE | TRUE | FALSE |

# Socrative `FALCOMPED`

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

# Negative gates

# Negative gates

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

# Negative gates

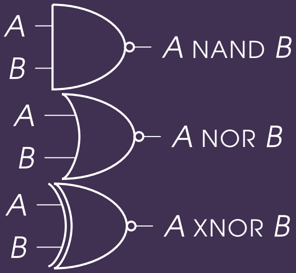
NAND , NOR , XNOR
are the **negations** of
AND , OR , XOR

$$A \text{ NAND } B = \text{NOT } (A \text{ AND } B)$$
$$A \text{ NOR } B = \text{NOT } (A \text{ OR } B)$$
$$A \text{ XNOR } B = \text{NOT } (A \text{ XOR } B)$$

# 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

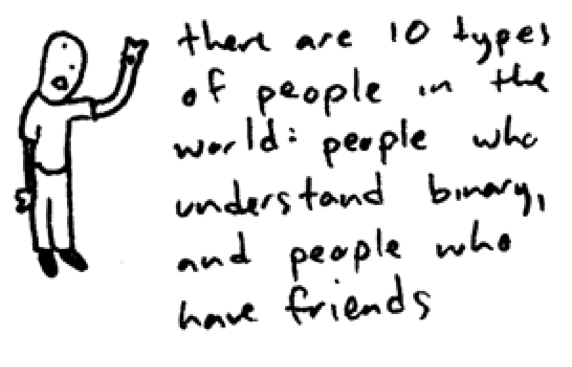



Image credit: http://www.toothpastefordinner.com

# How we write numbers

# How we write numbers

- We write numbers in **base 10**

# How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$

# How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$
- When we write $6397$, we mean:

# How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$
- When we write $6397$, we mean:
  - Six thousand, three hundred and ninety seven

# How we write numbers

- We write numbers in **base 10**
- We have 10 **digits**: $0, 1, 2, \ldots, 8, 9$
- When we write $6397$, we mean:
  - Six thousand, three hundred and ninety seven
  - (Six thousands) and (three hundreds) and (nine tens) and (seven)

# How we write numbers

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

# How we write numbers

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

# Binary

# Binary

- Binary notation works the same, but is **base 2** instead of **base 10**

# Binary

- Binary notation works the same, but is **base 2** instead of **base 10**
- We have 2 **digits**: 0, 1

# 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:

# 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:
  $$(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

# 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:

$$
\begin{aligned}
&(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) \\
&+ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
&= 2^7 + 2^3 + 2^1 + 2^0
\end{aligned}
$$

# 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:
  $$(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4)$$
  $$+ (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$
  $$= 2^7 + 2^3 + 2^1 + 2^0$$
  $$= 128 + 8 + 2 + 1 \text{ (base 10)}$$

# 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

# Bits, bytes and words

- A **bit** is a binary digit

# Bits, bytes and words

- A **bit** is a binary digit
  - Can store a 0 or 1 (i.e. a boolean value)

# 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**

# 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

# 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

# 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

# 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

# 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 $2^n - 1$

# 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 $2^n - 1$
  - $2^{16} - 1 = 65,535$

# 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 $2^n - 1$
  - $2^{16} - 1 = 65,535$
  - $2^{32} - 1 = 4,294,967,295$

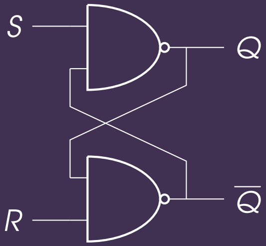
# Bits, bytes and words

- 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 $2^n - 1$
  - $2^{16} - 1 = 65,535$
  - $2^{32} - 1 = 4,294,967,295$
  - $2^{64} - 1 = 18,446,744,073,709,551,615$

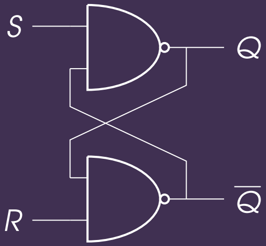
FALMOUTH UNIVERSITY

# Computer memory

# What does this circuit do?

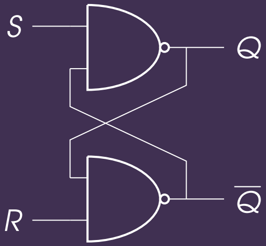


# What does this circuit do?



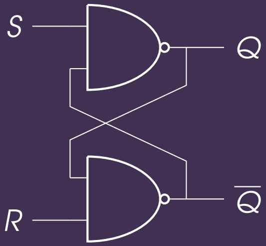

- This is called a **NAND latch**

# What does this circuit do?




- This is called a **NAND latch**
- It "remembers" a single boolean value

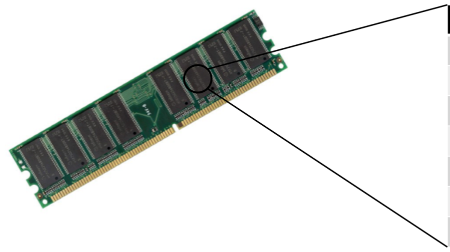
# What does this circuit do?




- 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

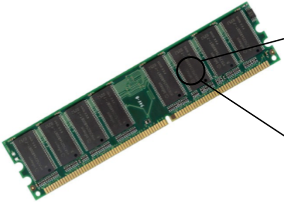
# Memory



| Address | Data |
|---:|---|
| 0 | 01001000 |
| 1 | 01100101 |
| 2 | 01101100 |
| 3 | 01101100 |
| 4 | 01101111 |
| 5 | 00100001 |
| ... | ... |

- Memory works like a set of **boxes**

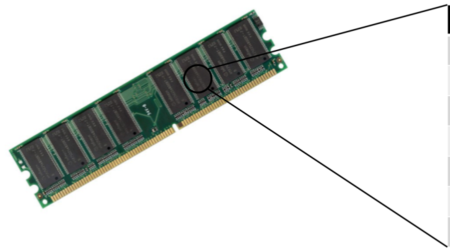
# Memory



| Address | Data |
|---|---|
| 0 | 01001000 |
| 1 | 01100101 |
| 2 | 01101100 |
| 3 | 01101100 |
| 4 | 01101111 |
| 5 | 00100001 |
| ... | ... |

- Memory works like a set of **boxes**
- Each box has a number, its **address**

# Memory



| Address | Data |
|---:|---|
| 0 | 01001000 |
| 1 | 01100101 |
| 2 | 01101100 |
| 3 | 01101100 |
| 4 | 01101111 |
| 5 | 00100001 |
| ... | ... |

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

# Data representation

# Data representation

- Memory stores **sequences of numbers**

# Data representation

- Memory stores **sequences of numbers**
- Therefore, any data stored by a computer must be represented as a sequence of numbers

# 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

# 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

# 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

# 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

# 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

FALMOUTH UNIVERSITY

# Worksheet B