Computer Systems

System basics

ILO’s

- Define the terms hardware and software and understand the relationship between them.
- Explain what’s meant by systems software and application software and be able to give examples of them.
- Understand the need for and functions of the OS and utility programs.

Hardware vs software

- **Hardware** is the electrical/mechanical components of a computer
- **Software** runs on hardware and tells it what to do to perform a task

Types of software

- Two types
  - System
  - Application

Types of software

- Two types
  - System
  - Application
- System software performs tasks related to the management of computer systems
  - Examples: Operating systems, USB drivers, antivirus

Types of software

- Two types
  - System
  - Application
- System software performs tasks related to the management of computer systems
  - Examples: Operating systems, USB drivers, antivirus
- Application software completes user-oriented tasks that the user would need to do with or without a computer
  - Examples: Microsoft word, web browsers

Operating systems

- Manages software and hardware

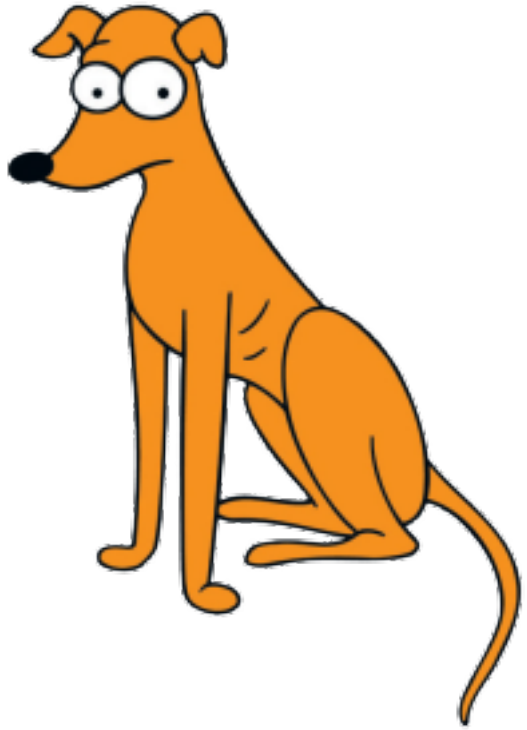
Operating systems

- Manages software and hardware
- Processors
- Memory
- I/O interfaces

- Which operating systems do you know?

Utilities

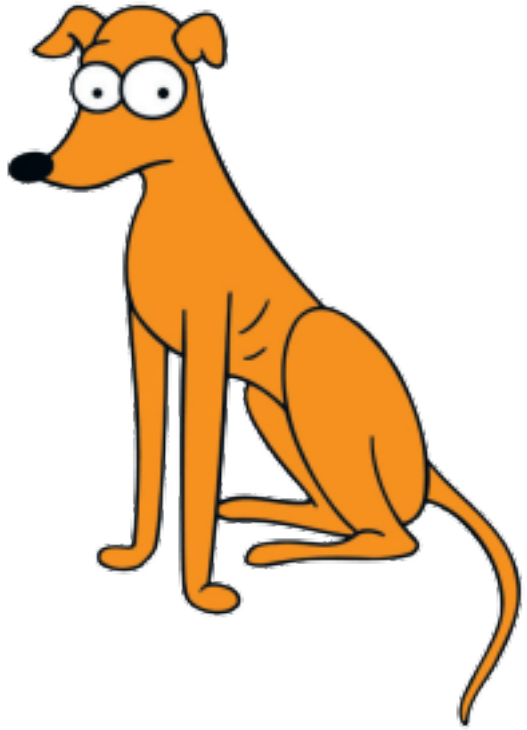
- Helps manages computer but not essential



Utilities

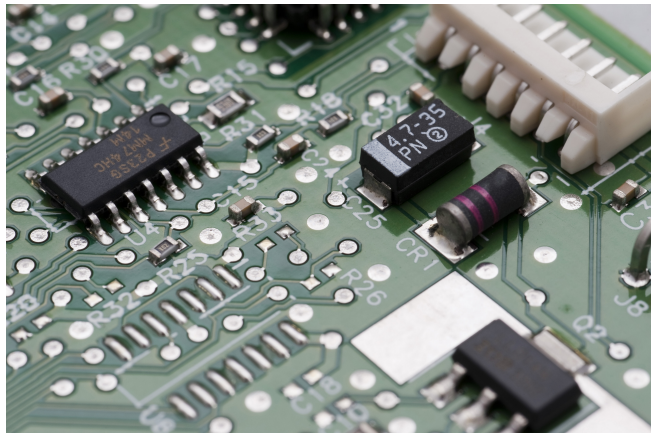
- Helps manages computer but not essential
- Compression program
- Antivirus
- Software updates
- Disk defragmentation

- Any others?



ILO’s

- Define the terms hardware and software and understand the relationship between them.
- Explain what’s meant by systems software and application software and be able to give examples of them.
- Understand the need for and functions of the OS and utility programs.



Truth tables and logic circuits

ILO’s

- Be able to construct truth tables for gates and circuits.
- Be able to draw logic circuits to represent a simple logic problem.
- Be able to create simple Boolean expressions and convert between Boolean expressions and logic circuits.

Logic gates

https://www.youtube.com/watch?v=3xYXUeSmb-Y

Logic gates

- AND
- OR
- NOT

} Obvious ones we have seen already

Logic gates

- AND
- OR
- NOT

} Obvious ones we have seen already

- XOR - Either, or but not both

Logic gates

- AND
- OR
- NOT

} Obvious ones we have seen already

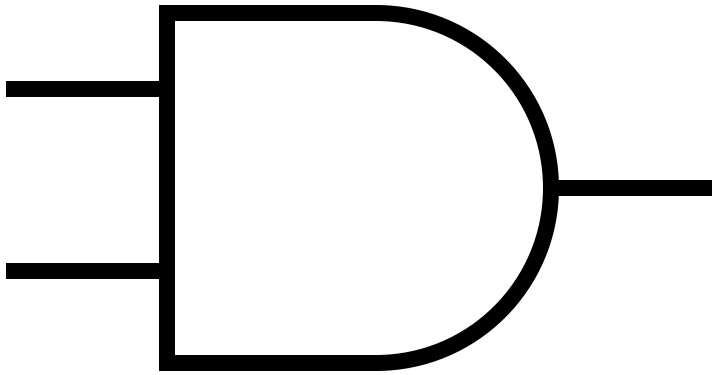
- XOR - Either, or but not both

- We can convert these logic gates into symbols so that we can design program circuits
- This enables us to design programs and share these with others
- Similar to flowcharts

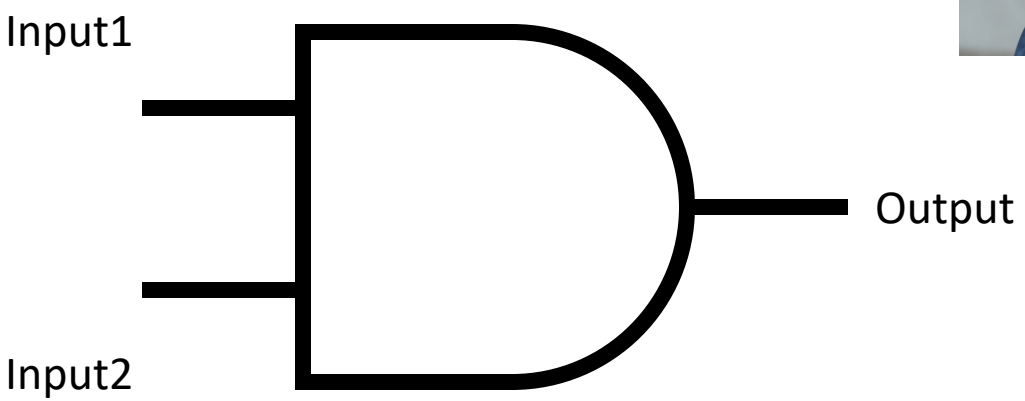
0 = False

1 = True

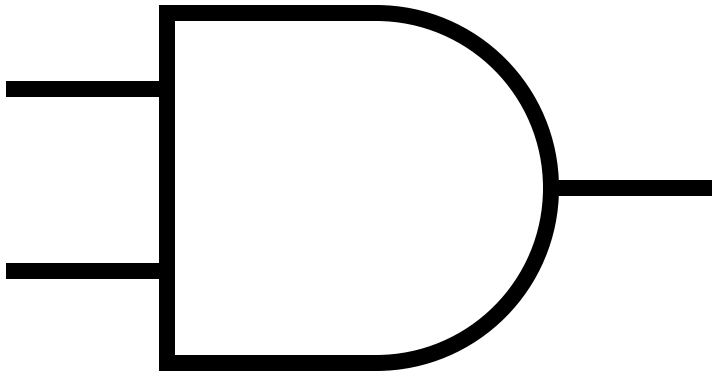
AND



AND



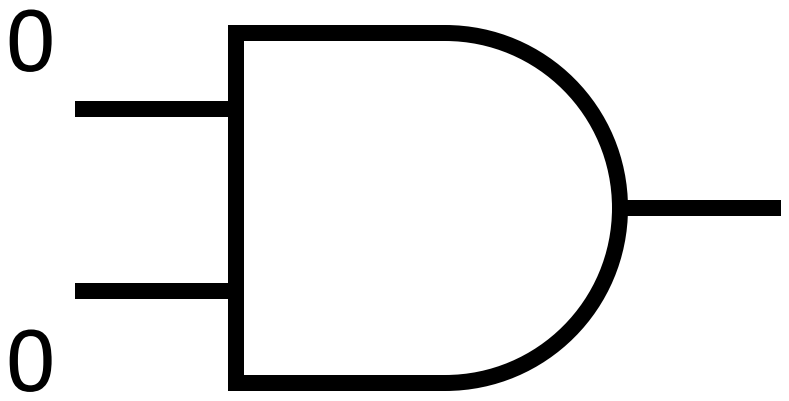
AND



Truth table:

Input1	Input2	Output

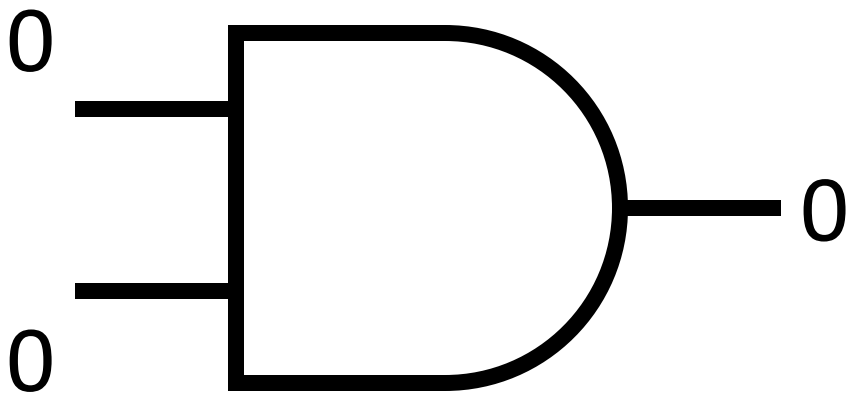
AND



Truth table:

Input1	Input2	Output
0	0	

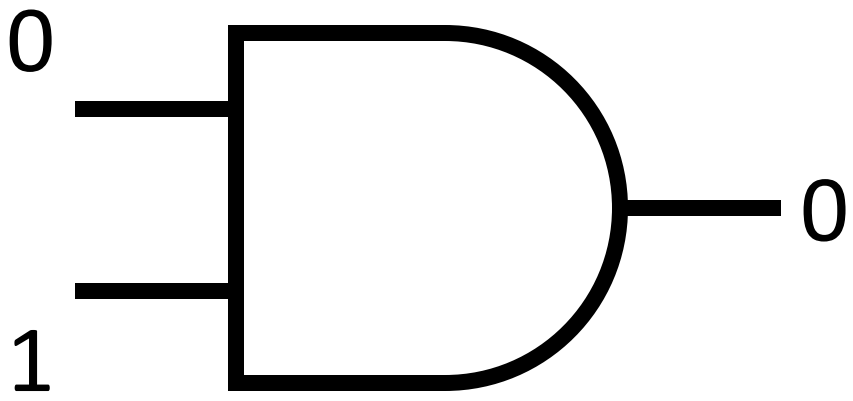
AND



Truth table:

Input1	Input2	Output
0	0	0
0	1	0
1	0	0
1	1	1

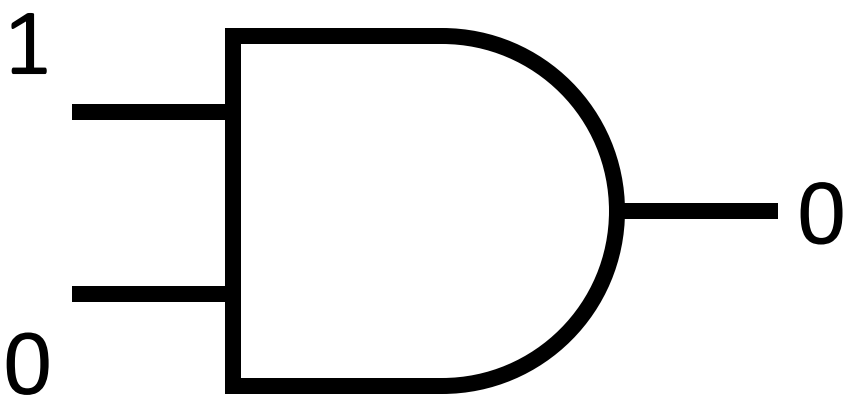
AND



Truth table:

Input1	Input2	Output
0	0	0
0	1	0

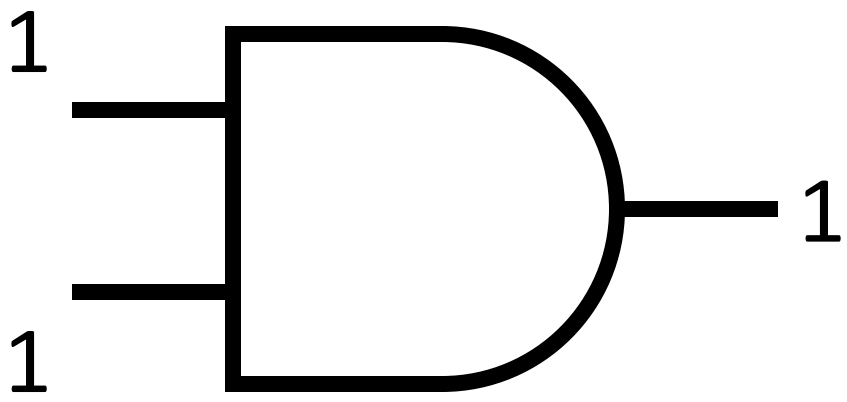
AND



Truth table:

Input1	Input2	Output
0	0	0
0	1	0
1	0	0

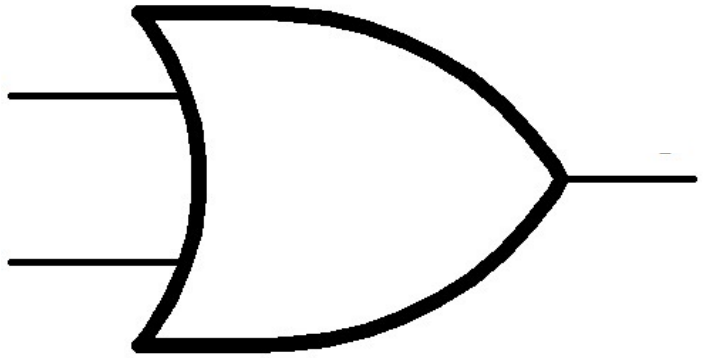
AND



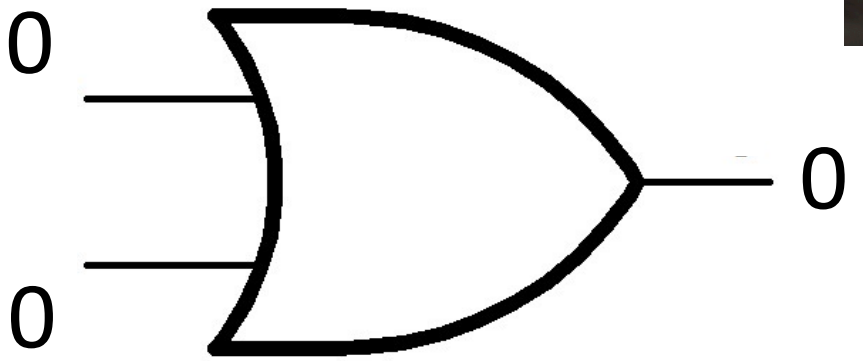
Truth table:

Input1	Input2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR



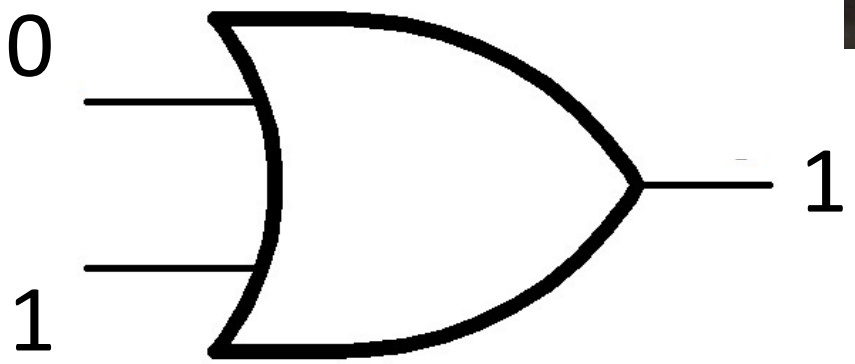
OR



Truth table:

Input1	Input2	Output
0	0	0

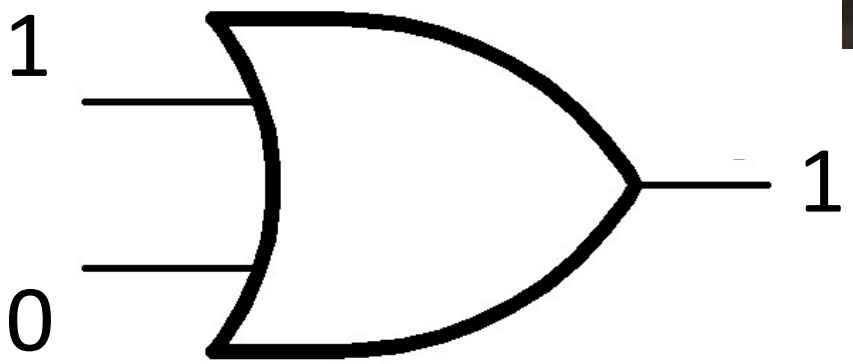
OR



Truth table:

Input1	Input2	Output
0	0	0
0	1	1

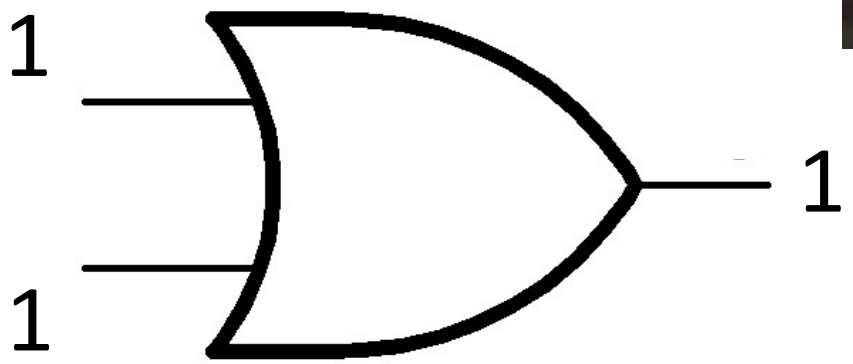
OR



Truth table:

Input1	Input2	Output
0	0	0
0	1	1
1	0	1

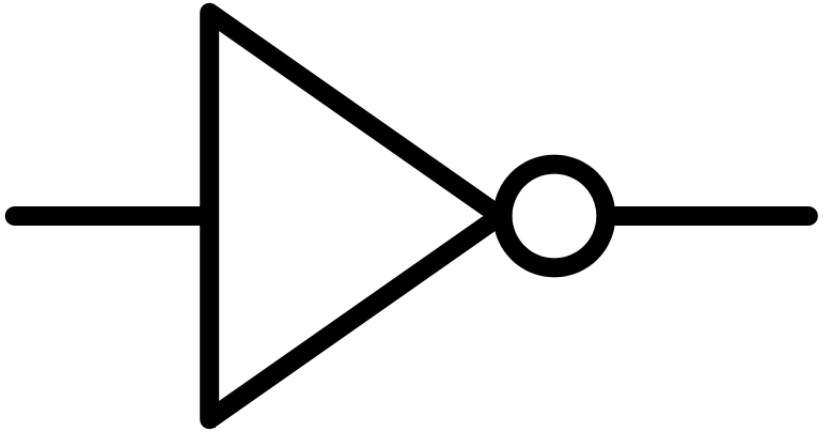
OR



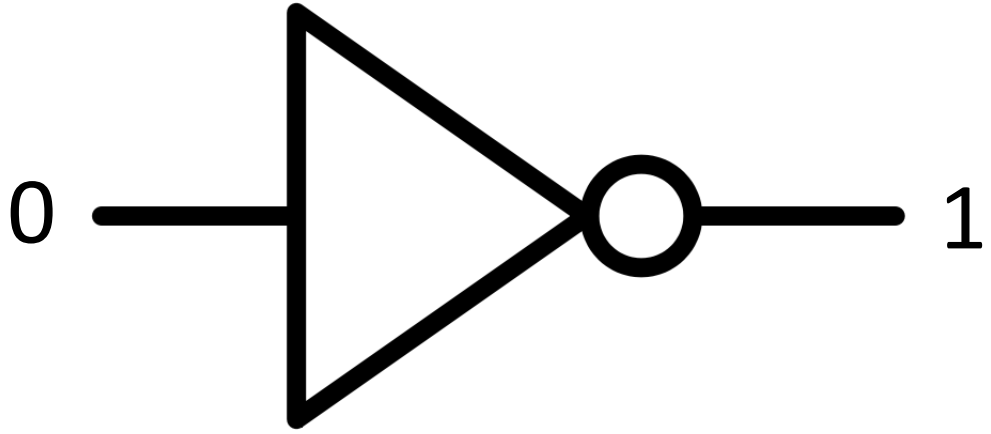
Truth table:

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT



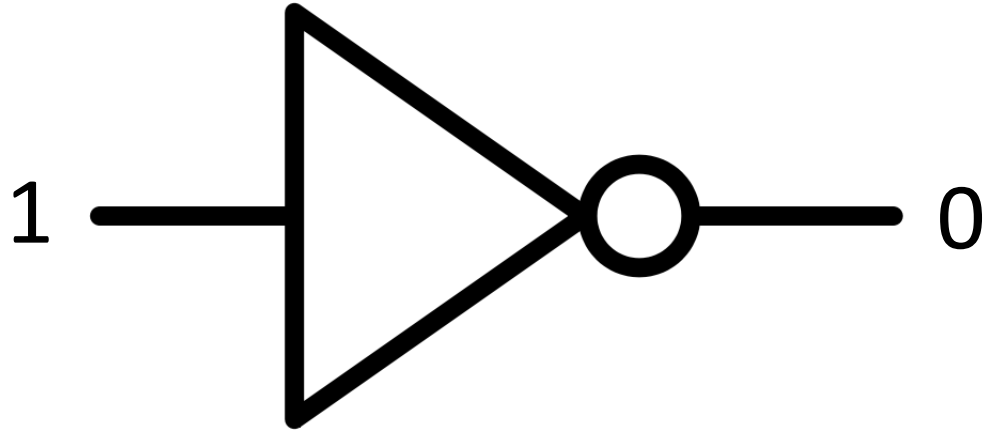
NOT



Truth table:

Input1	Output
0	1

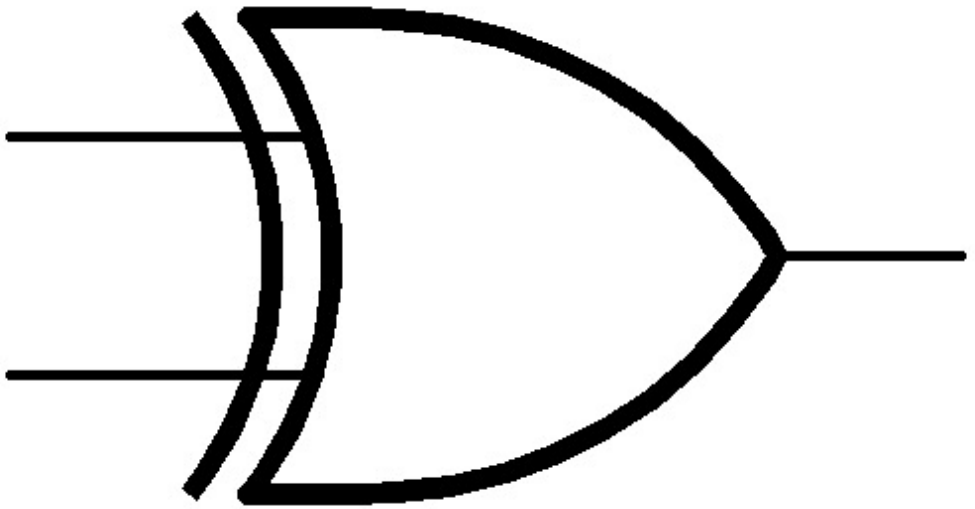
NOT



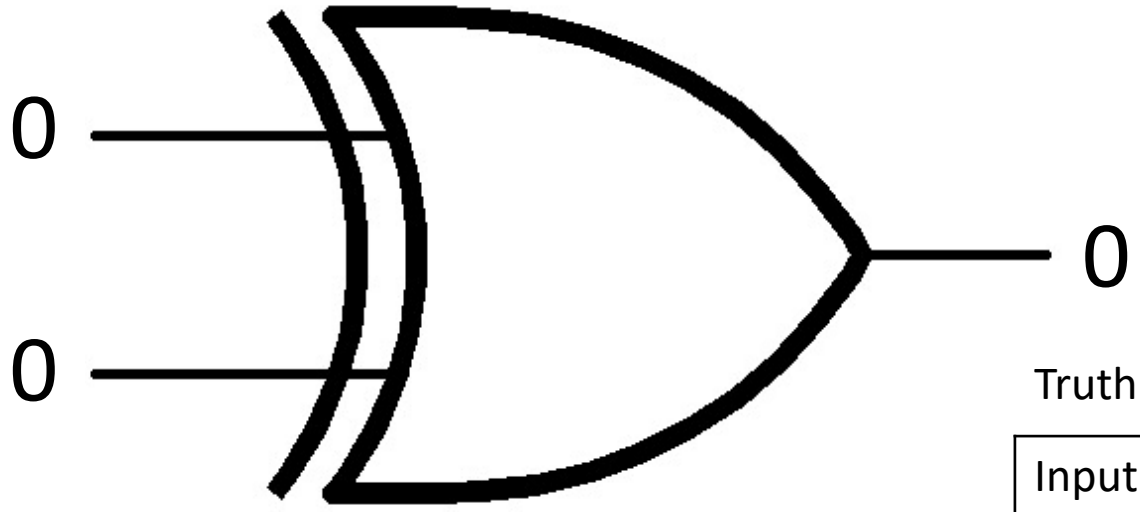
Truth table:

Input1	Output
0	1
1	0

XOR



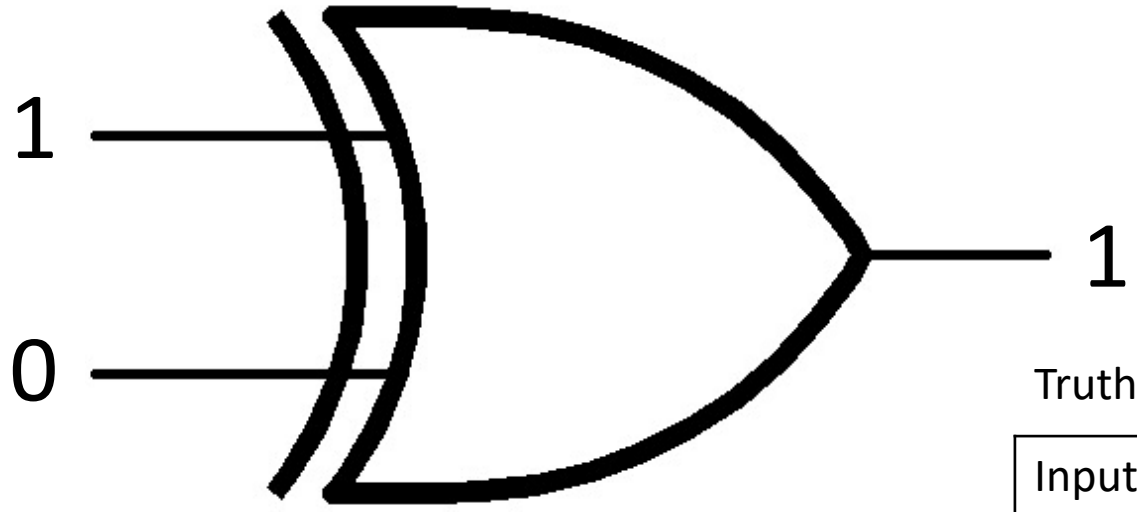
XOR



Truth table:

Input1	Input2	Output
0	0	0

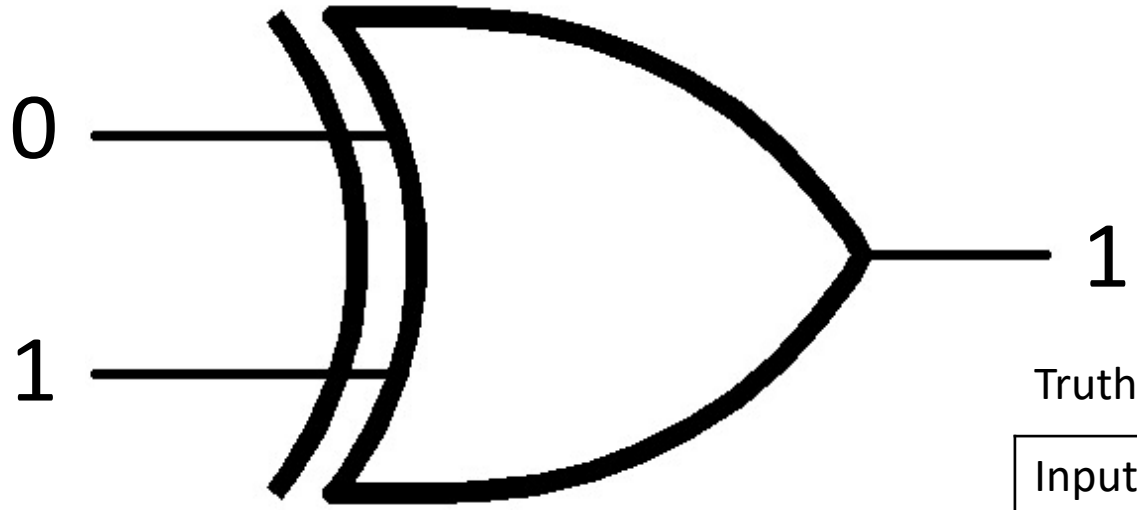
XOR



Truth table:

Input1	Input2	Output
0	0	0
1	0	1

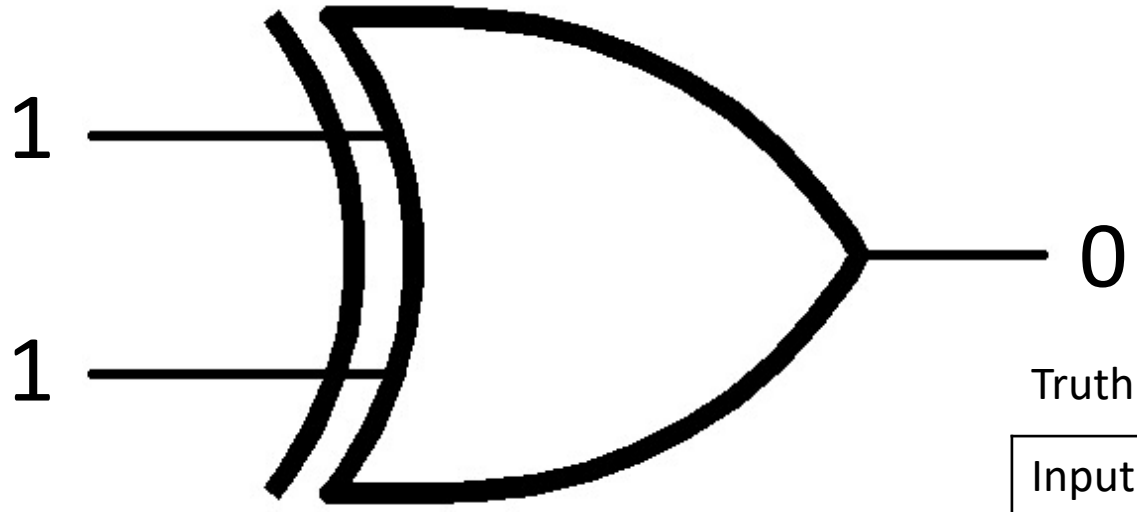
XOR



Truth table:

Input1	Input2	Output
0	0	0
1	0	1
0	1	1

XOR



Truth table:

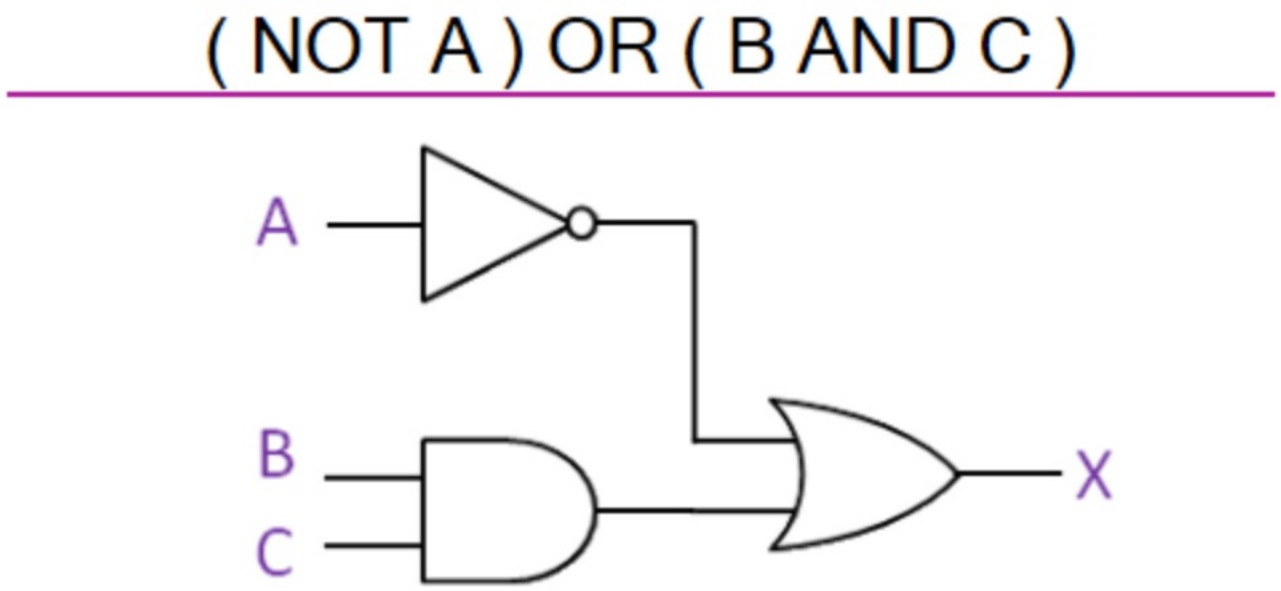
Input1	Input2	Output
0	0	0
1	0	1
0	1	1
1	1	0

Logic circuits

- We can combine logic gates to produce logic circuits

Logic circuits

- We can combine logic gates to produce logic circuits



Logic circuits

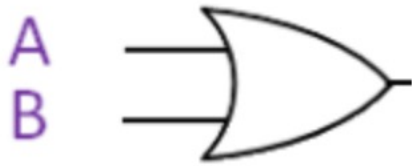
- How would we represent the following:

(A OR B) AND (NOT C)

Logic circuits

- How would we represent the following:

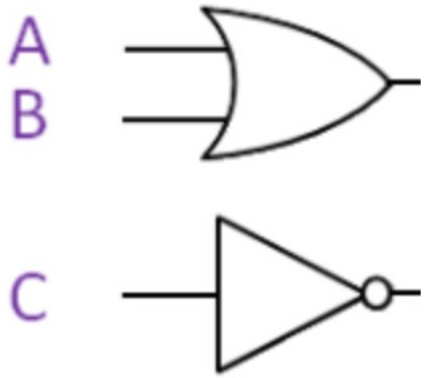
(A OR B) AND (NOT C)



Logic circuits

- How would we represent the following:

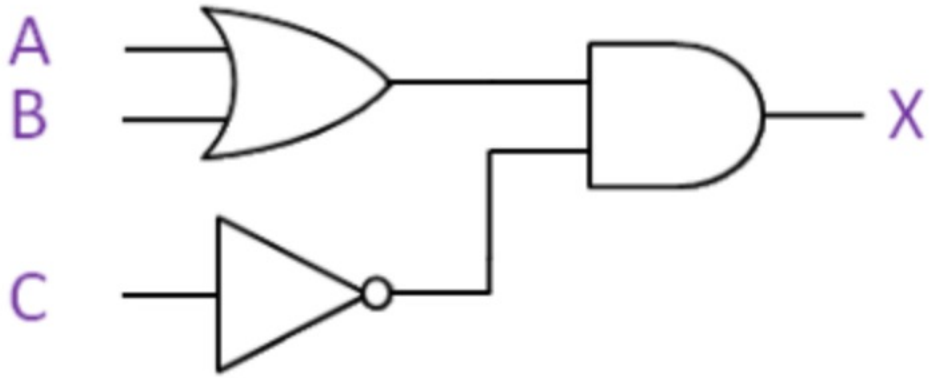
(A OR B) AND (NOT C)



Logic circuits

- How would we represent the following:

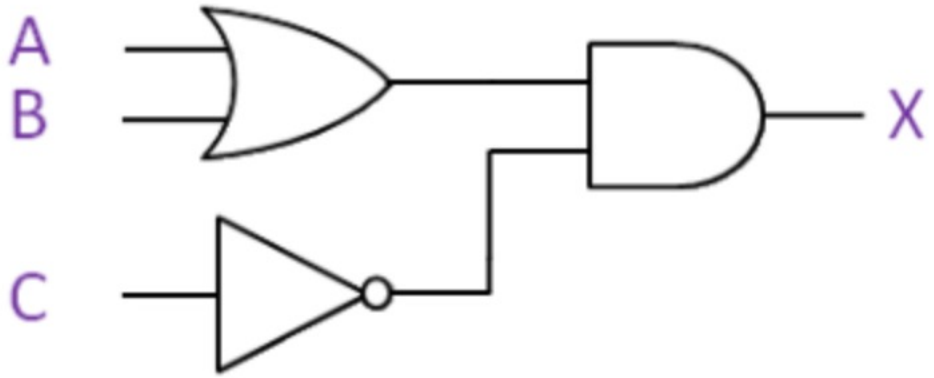
(A OR B) AND (NOT C)



Logic circuits

- How would we represent the following:

(A OR B) AND (NOT C)



Exercises – part 1

- Generate logic circuits and truth tables for the following:

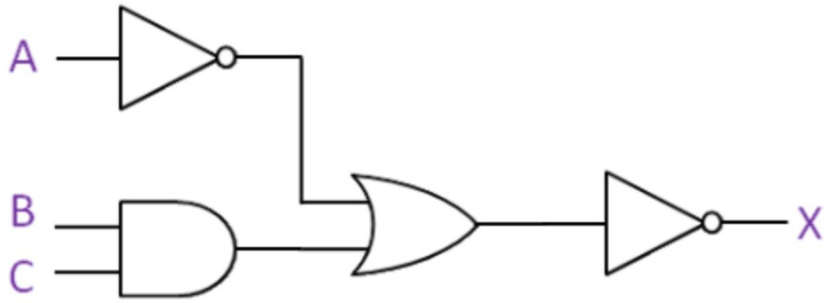
NOT ((NOT A) OR (B AND C))

(NOT (A AND B)) OR C

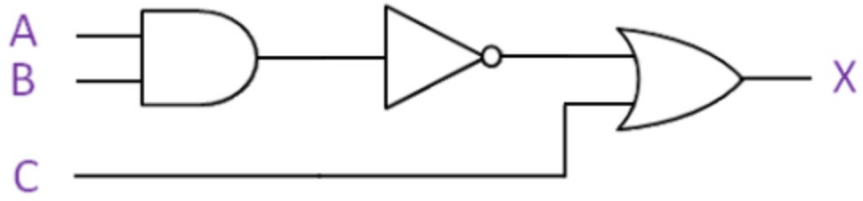
Answers – part 1

- Generate logic circuits and truth tables for the following:

NOT ((NOT A) OR (B AND C))



(NOT (A AND B)) OR C



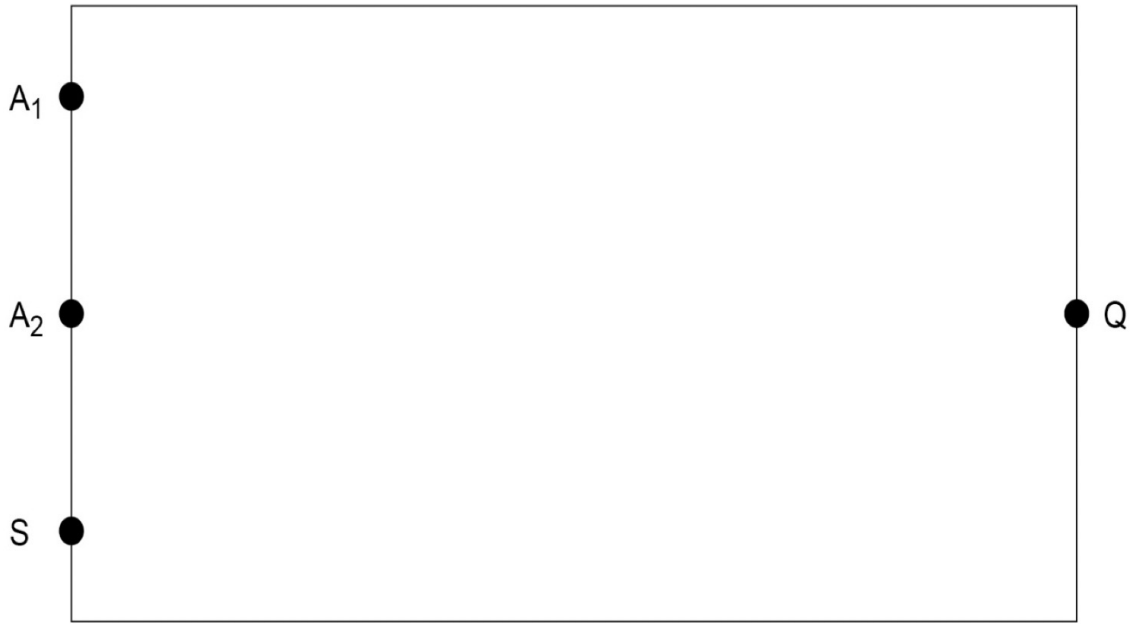
Exercises –part 2

A logic circuit is being developed for an audio advert in a shop that plays automatically if a customer is detected nearby.

- The system has two sensors, A_1 and A_2, that detect if a customer is near. The audio plays if either of these sensors is activated.
- The system should only play if another audio system, S, is not playing.
- The output from the circuit, for whether the advert should play or not, is Q.

Complete the logic circuit for this system.

[3 marks]

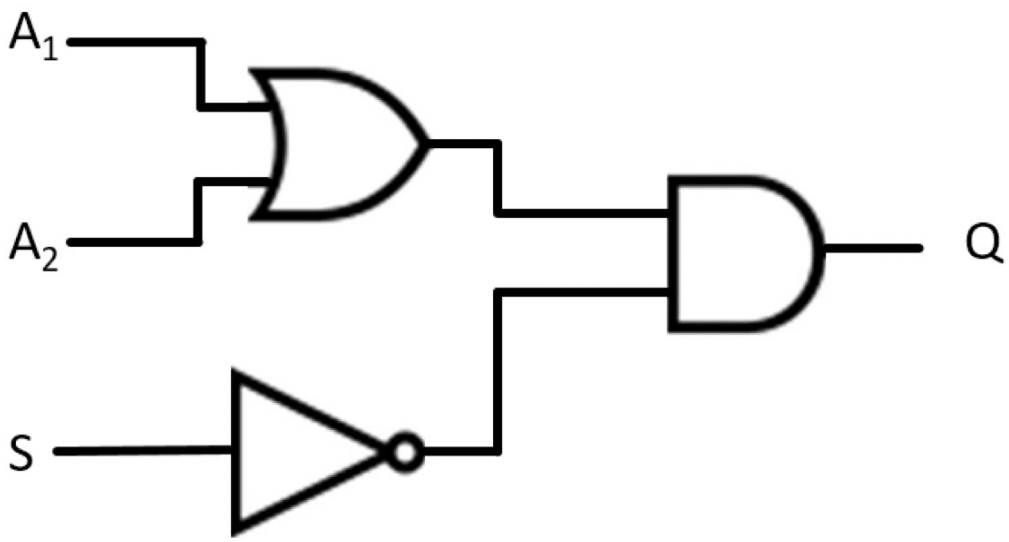


Answers – part 2

Max 2 marks if not fully correct (the fully correct answer is given in example 1).

Mark A if A_1 and A_2 are the inputs to an OR gate;
Mark B if S is the input to a NOT gate;
Mark C if the output from an AND gate is Q;

Example 1 (Fully correct answer)



ILO’s

- Be able to construct truth tables for gates and circuits.
- Be able to draw logic circuits to represent a simple logic problem.
- Be able to create simple Boolean expressions and convert between Boolean expressions and logic circuits.

Computer language

ILO’s

- Be able to describe and discuss low and high-level languages, their advantages and their uses.
- Be able describe and discuss interpreters, compilers and assemblers and when they could be used.

Low vs high (level languages)

- What is low level language?

Low vs high (level languages)

- What is low level language?
  - Written directly for computers

Low vs high (level languages)

- What is low level language?
  - Written directly for computers
- What types of low level language are there?

Low vs high (level languages)

- What is low level language?
  - Written directly for computers
- What types of low level language are there?
  - Machine code
  - Assembly language

Low vs high (level languages)

- What is low level language?
  - Written directly for computers
- What types of low level language are there?
  - Machine code
  - Assembly language

Low vs high (level languages)

- What is low level language?
  - Written directly for computers
- What types of low level language are there?
  - Machine code
  - Assembly language

  - Machine code: Executed directly by the CPU and uses binary values

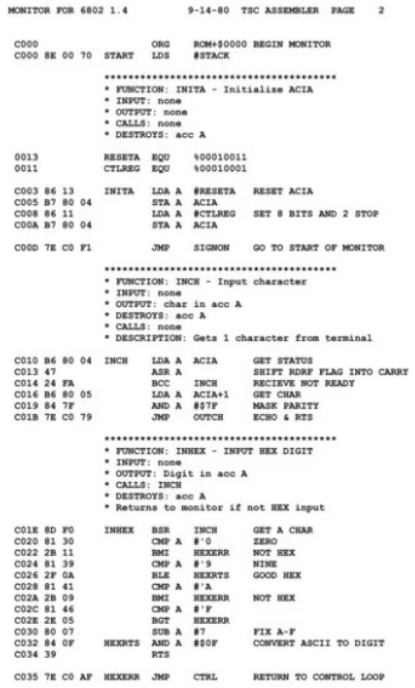
Low vs high (level languages)

- What is low level language?
  - Written directly for computers
- What types of low level language are there?
  - Machine code
  - Assembly language

- Machine code: Executed directly by the CPU and uses binary values
- Assembly language: Intermediate between high level-languages uses English syntax but is more difficult to interpret

Low vs high (level languages)

```
00000000: 01001101 01011010 10010000 00000000 00000011 00000000  MZ····
00000006: 00000000 00000000 00000100 00000000 00000000 00000000  ......
0000000c: 11111111 11111111 00000000 00000000 10111000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 01000000 00000000 00000000 00000000 00000000 00000000  @.....
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000024: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000002a: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000030: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000036: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000003c: 10000000 00000000 00000000 00000000 00001110 00011111  ......
00000042: 10111010 00001110 00000000 10110100 00001001 11001101  ......
00000048: 00100001 10111000 00000001 01001100 11001101 00100001  !··L·!
0000004e: 01010100 01101000 01101001 01110011 00100000 01110000  This p
00000054: 01110010 01101111 01100111 01110010 01100001 01101101  rogram
0000005a: 00100000 01100011 01100001 01101110 01101110 01101111   canno
00000060: 01110100 00100000 01100010 01100101 00100000 01110010  t be r
00000066: 01110101 01101110 00100000 01101001 01101110 00100000  un in
0000006c: 01000100 01001111 01010011 00100000 01101101 01101111  DOS mo
00000072: 01100100 01100101 00101110 00001101 00001101 00001010  de.····
00000078: 00100100 00000000 00000000 00000000 00000000 00000000  $·····
0000007e: 00000000 00000000 01010000 01000101 00000000 00000000  ··PE··
```

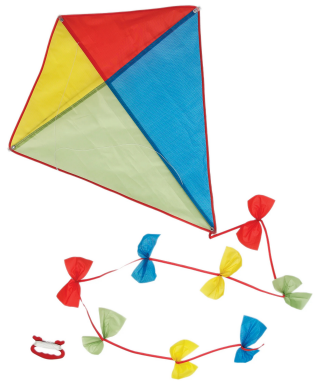


```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

C000                    ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70   START   LDS     #STACK

                        ************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA   RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON    GO TO START OF MONITOR

                        ************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A   ACIA      GET STATUS
C013 47                 ASR A             SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC     INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A   ACIA+1    GET CHAR
C019 84 7F              AND A   #$7F      MASK PARITY
C01B 7E C0 79           JMP     OUTCH     ECHO & RTS

                        ************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR     INCH      GET A CHAR
C020 81 30              CMP A   #'0       ZERO
C022 2B 11              BMI     HEXERR    NOT HEX
C024 81 39              CMP A   #'9       NINE
C026 2F 0A              BLE     HEXRTS    GOOD HEX
C028 81 41              CMP A   #'A
C02A 2B 09              BMI     HEXERR    NOT HEX
C02C 81 46              CMP A   #'F
C02E 2E 05              BGT     HEXERR
C030 80 07              SUB A   #7        FIX A-F
C032 84 0F      HEXRTS  AND A   #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF   HEXERR  JMP     CTRL      RETURN TO CONTROL LOOP
```

- Machine code: Executed directly by the CPU and uses binary values
- Assembly language: Intermediate between high level-languages uses English syntax but is more difficult to interpret

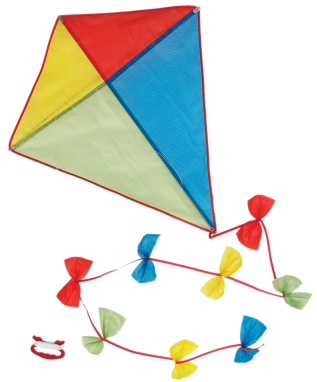
Low vs high (level languages)

- What is high level language?



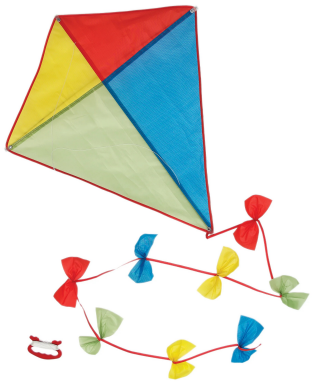
Low vs high (level languages)

- What is high level language?
  - Easier to understand but requires an interpreter/compiler to convert into machine code



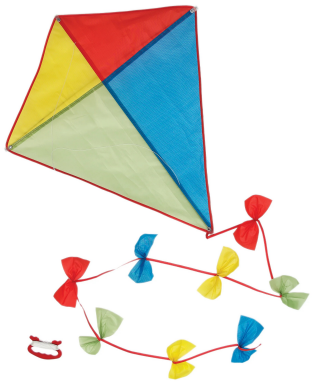
Low vs high (level languages)

- What is high level language?
  - Easier to understand but requires an interpreter/compiler to convert into machine code
- What types of high level language are there?



Low vs high (level languages)

- What is high level language?
  - Easier to understand but requires an interpreter/compiler to convert into machine code
- What types of high level language are there?
  - C++, Python, Java



Low vs high (level languages)

BASIS	HIGH LEVEL LANGUAGE	LOW LEVEL LANGUAGE
Understandable	A user or programmer can easily understand it as it is more closer to English. Therefore, it is user or programmer friendly	A machine or computer can easily understand it. Therefore, it is machine friendly
Dependency	Completely machine independent	Totally machine dependent
Translator	It requires a compiler or an interpreter for an effective translation	No translator required, but for Assembly language, assembler is required
Program Speed	The code written in High level language must be compiled. Therefore, a translator is required to translate it to low level language. Therefore, the program execution is slower than low-level language	program execution is faster than high-level language

Low vs high (level languages)

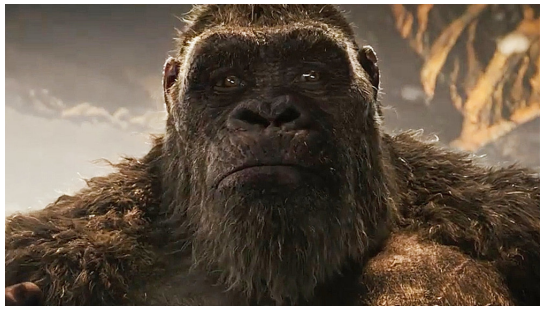
BASIS	HIGH LEVEL LANGUAGE	LOW LEVEL LANGUAGE
Memory efficiency	Least	High
Structure	Well structured	Not well structured
Debug	Easy to debug	Difficult to debug
Portability	Portable	Not portable
Manipulation in Code	Easy	Difficult
Hardware Knowledge	Not needed	Required
Example	C++,Python, Java	Machine language and Assembly Language
Application	Websites, Mobile and Desktop Application development	Operating Systems development

Interpreter vs compiler vs assembler

- https://www.youtube.com/watch?v=RiV6voYxldQ

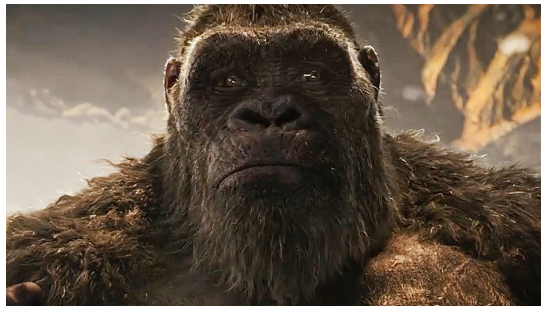
Interpreter vs compiler vs assembler

- **Interpreter**:



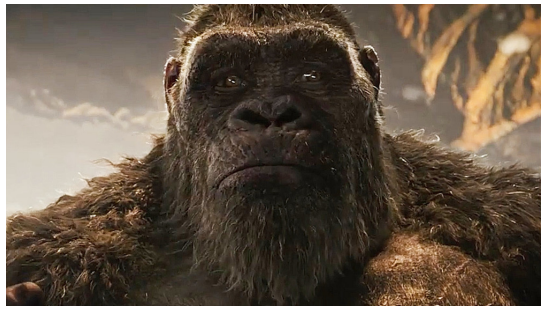
Interpreter vs compiler vs assembler

- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.



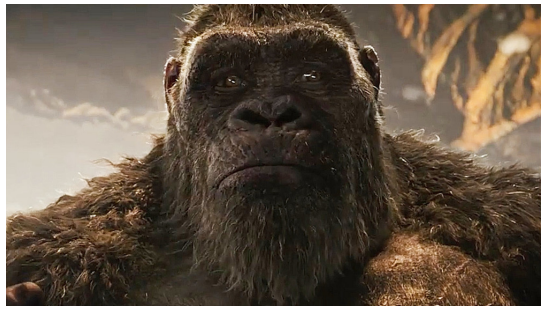
Interpreter vs compiler vs assembler

- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.
- **Compiler:**



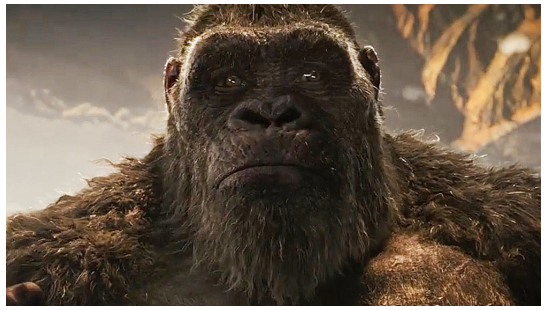
Interpreter vs compiler vs assembler

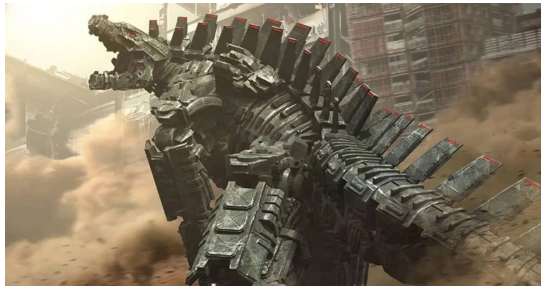
- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.
- **Compiler:** A compiler takes the source code as a whole and translates it into object code all in one go. Once converted, the object code can be run at any time. This process is called compilation.



Interpreter vs compiler vs assembler

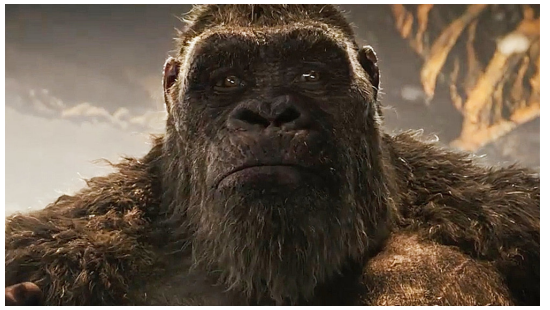
- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.
- **Compiler:** A compiler takes the source code as a whole and translates it into object code all in one go. Once converted, the object code can be run at any time. This process is called compilation.
- **Assemblers:**

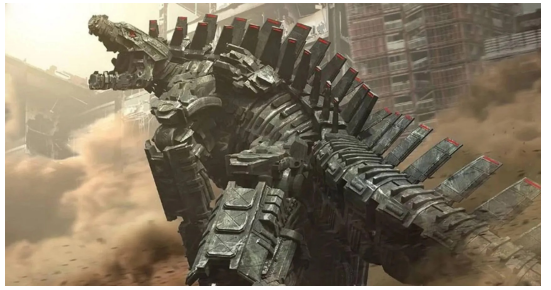




Interpreter vs compiler vs assembler

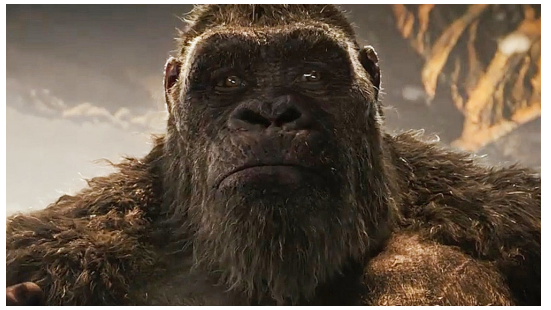
- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.
- **Compiler:** A compiler takes the source code as a whole and translates it into object code all in one go. Once converted, the object code can be run at any time. This process is called compilation.
- **Assemblers:** An assembler translates assembly language into object code. Whereas compilers and interpreters generate many machine code instructions for each high-level instruction, assemblers create one machine code instruction for each assembly instruction.

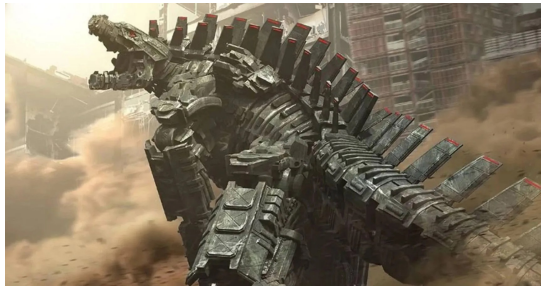




Interpreter vs compiler vs assembler

- **Interpreter**: An interpreter translates source code into object code one instruction at a time. It is similar to a human translator translating what a person says into another language, sentence by sentence. The resulting object code is then executed immediately. The process is called interpretation.
- **Compiler:** A compiler takes the source code as a whole and translates it into object code all in one go. Once converted, the object code can be run at any time. This process is called compilation.
- **Assemblers:** An assembler translates assembly language into object code. Whereas compilers and interpreters generate many machine code instructions for each high-level instruction, assemblers create one machine code instruction for each assembly instruction.





Exercises

- Make flash cards summarising what we have learnt today
- Test yourself until you get all of them correct

ILO’s

- Be able to describe and discuss low and high-level languages, their advantages and their uses.
- Be able describe and discuss interpreters, compilers and assemblers and when they could be used.

Central Processing Unit (CPU)

ILO’s

- Explain role of main memory, components of CPU, buses and the impact of certain changes.
- Understand and explain the fetch-execute cycle.

Components of the CPU

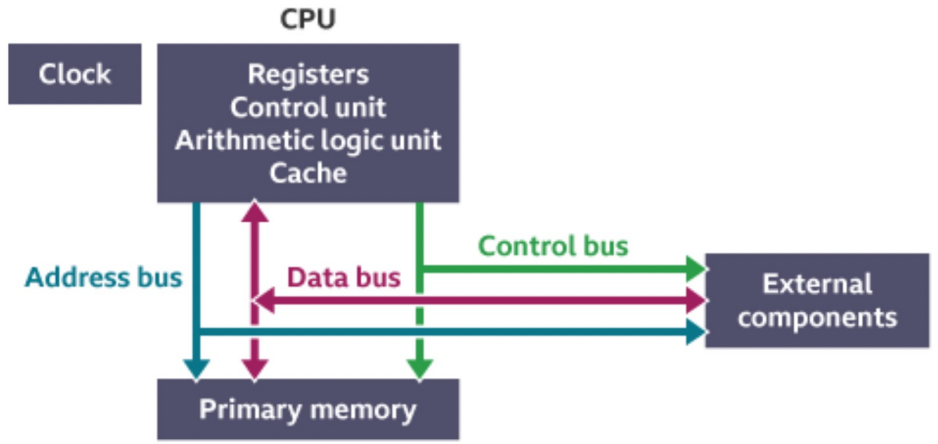
- Six main components:

Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - Registers
  - Cache
  - Buses
  - Clock

Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - Registers
  - Cache
  - Buses
  - Clock



Components of the CPU

- Six main components:
  - **Control unit (CU)**
  - Arithmetic logic unit (ALU)
  - Registers
  - Cache
  - Buses
  - Clock

Control unit:

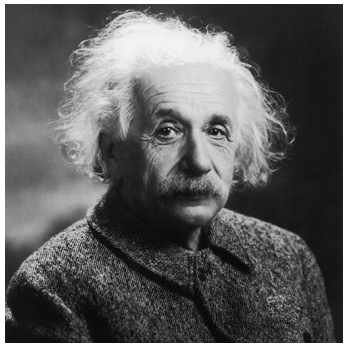
- It fetches, decodes and executes instructions
- It issues control signals that control hardware
- It moves data around the system

Components of the CPU

- Six main components:
  - Control unit (CU)
  - **Arithmetic logic unit (ALU)**
  - Registers
  - Cache
  - Buses
  - Clock

Algorithmic logic unit:

- It performs arithmetic and logical operations (decisions). The ALU is where calculations are done and where decisions are made.
- It acts as a gateway between primary memory and secondary storage . Data transferred between them passes through the ALU.



Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - **Registers**
  - Cache
  - Buses
  - Clock

Registers:

- Registers are small amounts of high-speed memory contained within the CPU. They are used by the processor to store small amounts of data that are needed during processing, such as:
  - The address of the next instruction to be executed
  - The current instruction being decoded
  - The results of calculations

Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - Registers
  - **Cache**
  - Buses
  - Clock

Cache:

- Cache is a small amount of high-speed random access memory (RAM) built directly within the processor. It is used to temporarily hold data and instructions that the processor is likely to reuse. This allows for faster processing as the processor does not have to wait for the data and instructions to be fetched from the RAM.

Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - Registers
  - Cache
  - **Buses**
  - Clock

Buses:

- A bus is a high-speed internal connection. Buses are used to send control signals and data between the processor and other components.
- Three types of bus are used:
  - Address bus - carries memory addresses from the processor to other components such as primary memory and input/output devices.
  - Data bus - carries the actual data between the processor and other components.
  - Control bus - carries control signals from the processor to other components. The control bus also carries the clock's pulses

Components of the CPU

- Six main components:
  - Control unit (CU)
  - Arithmetic logic unit (ALU)
  - Registers
  - Cache
  - Buses
  - **Clock**

Clock:

- The CPU contains a clock which is used to coordinate all of the computer's components. The clock sends out a regular electrical pulse which synchronises (keeps in time) all the components.
- The frequency of the pulses is known as the clock speed. Clock speed is measured in hertz. The higher the frequency, the more instructions can be performed in any given moment of time.

Factors affecting performance

- https://youtu.be/X2WH8mHJnhM

Factors affecting performance

- Three main factors:
  - Clock speed
  - Cache size
  - Number of cores

Factors affecting performance

- Three main factors:
  - **Clock speed**
  - Cache size
  - Number of cores

Clock speed:

- Clock speed is the number of pulses the central processing unit's (CPU) clock generates per second. It is measured in hertz.
- CPU clocks can sometimes be sped up slightly by the user. This process is known as overclocking. The more pulses per second, the more fetch-decode-execute cycles that can be performed and the more instructions that are processed in a given space of time. Overclocking can cause long term damage to the CPU as it is working harder and producing more heat.

Factors affecting performance

- Three main factors:
  - Clock speed
  - **Cache size**
  - Number of cores

Cache size:

- Cache is a small amount of high-speed random access memory (RAM) built directly within the processor. It is used to temporarily hold data and instructions that the processor is likely to reuse.
- The bigger its cache, the less time a processor has to wait for instructions to be fetched.

Factors affecting performance

- Three main factors:
  - Clock speed
  - Cache size
  - **Number of cores**

Number of cores:

- A processing unit within a CPU is known as a core. Each core is capable of fetching, decoding and executing its own instructions.
- The more cores a CPU has, the greater the number of instructions it can process in a given space of time. Many modern CPUs are dual (two) or quad (four) core processors. This provides vastly superior processing power compared to CPUs with a single core.

Von Neumann Architecture

- CPU receives data and instructions from input and memory
- Von Neumann architecture outlines rules which process must abide by:

Von Neumann Architecture

- CPU receives data and instructions from input and memory
- Von Neumann architecture outlines rules which process must abide by:
  - Data and instructions are both stored as binary.
  - Data and instructions are both stored in main memory.
  - Instructions are fetched from memory one at a time and in order - serially.
  - The processor decodes and executes an instruction, before cycling around to fetch the next instruction.
  - The cycle continues until no more instructions are available.

Von Neumann Architecture

- CPU receives data and instructions from input and memory
- Von Neumann architecture outlines rules which process must abide by:
  - Data and instructions are both stored as binary.
  - Data and instructions are both stored in main memory.
  - Instructions are fetched from memory one at a time and in order - serially.
  - The processor decodes and executes an instruction, before cycling around to fetch the next instruction.
  - The cycle continues until no more instructions are available.
  - **https://www.bbc.co.uk/bitesize/guides/z7qqmsg/revision/7**

Exercises

- Make flash cards summarising what we have learnt today
- Test yourself until you get all of them correct

ILO’s

- Explain role of main memory, components of CPU, buses and the impact of certain changes.
- Understand and explain the fetch-execute cycle.

Storage

ILO’s

- Understand difference between main memory and secondary storage and between RAM and ROM. Be able to explain volatile and non-volatile.
- Explain the operation of solid state, optical and magnetic storage.
- Discuss their relative advantages.
- Explain what cloud storage is and compare it to local storage

Main memory vs secondary storage

- Main memory is the temporary memory that computers use for storage

Main memory vs secondary storage

- Main memory is the temporary memory that computers use for storage
  - This comprises RAM and ROM

Main memory vs secondary storage

- Main memory is the temporary memory that computers use for storage
  - This comprises RAM and ROM
- Secondary storage is permanent external memory

Main memory vs secondary storage

- Main memory is the temporary memory that computers use for storage
  - This comprises RAM and ROM
- Secondary storage is permanent external memory
  - Such as floppy disks, CD’s and hard drives

RAM vs ROM

- RAM = Random Access Memory

RAM vs ROM

- RAM = Random Access Memory
  - Volatile memory which is lost after power supply is removed
  - High speed memory

RAM vs ROM

- RAM = Random Access Memory
  - Volatile memory which is lost after power supply is removed
  - High speed memory
- ROM = Read-Only Memory

RAM vs ROM

- RAM = Random Access Memory
  - Volatile memory which is lost after power supply is removed
  - High speed memory
- ROM = Read-Only Memory
  - Pre-recorded data used to store firmware
  - Non-volatile: Remains even after the power supply is removed

RAM vs ROM

- RAM = Random Access Memory
  - Volatile memory which is lost after power supply is removed
  - High speed memory
- ROM = Read-Only Memory
  - Pre-recorded data used to store firmware
  - Non-volatile: Remains even after the power supply is removed

Secondary storage

- Magnetic
- Solid-state
- Optical

Secondary storage

- **Magnetic**
- Solid-state
- Optical

Magnetic devices

- Use magnetic fields to magnetise tiny individual sections of a metal spinning disk.
- Each tiny section represents one bit.
- A magnetised section represents a binary ‘1’
- A demagnetised section represents a binary '0’.
- Data is accessed directly on a disk drive by knowing the location of the data
- Low cost, high in capacity and durable but subject to damage

Secondary storage

- Magnetic
- **Solid-state**
- Optical

Solid-state

- Technology used in SSD and USB
- Low power
- Fast access speed

Secondary storage

- Magnetic
- Solid-state
- **Optical**

Optical

- Uses a laser to scan the surface of a spinning disc made from metal or plastic
- Disc surface is split into tracks with many flat surfaces (lands) and pits (hollows)
- An optical scanner measure the ability of light to reflect along a track with reflection (from a land) = 1 and no reflection (from a pit) = 0
- Writing a disc requires the use of a laser to burn pits into a disc
- Storage for multimedia files

Cloud storage

Cloud storage

- Save data and files off-site
- Accessed via the internet
- Cost-effective, scalable and accessible

Cloud storage

- Save data and files off-site
- Accessed via the internet
- Cost-effective, scalable and accessible

Cloud storage

- Save data and files off-site
- Accessed via the internet
- Cost-effective, scalable and accessible

What types of cloud storage do you know?

Exercises

- Make flash cards summarising what we have learnt today
- Test yourself until you get all of them correct

ILO’s

- Understand difference between main memory and secondary storage and between RAM and ROM. Be able to explain volatile and non-volatile.
- Explain the operation of solid state, optical and magnetic storage.
- Discuss their relative advantages.
- Explain what cloud storage is and compare it to local storage

Embedded Systems

ILO’s

- Explain what an embedded system is and how an embedded system differs from a non-embedded system.
- Give examples of embedded systems.


BOSCH

Embedded system

Non-embedded system

Designed for a specific function

Can be used for many purposes

What are the differences in processor speed, amount and type of main memory, secondary storage, input and output devices and upgradeability?

What types of embedded/non-embedded systems do you know?

Exercises

- Make flash cards summarising what we have learnt today
- Test yourself until you get all of them correct

ILO’s

- Explain what an embedded system is and how an embedded system differs from a non-embedded system.
- Give examples of embedded systems.