








## Syllabus Content:

### 1.4.1 CPU architecture

-  show understanding of the basic Von Neumann model for a computer system and the stored program concept
-  show understanding of the roles carried out by registers, including the difference between general purpose and special purpose registers including:
  - Program Counter
  - Memory Data Register
  - Memory Address Register
  - Index Register
  - Current Instruction Register
  - Status Register
-  show understanding of the roles carried out by the Arithmetic and Logic Unit (ALU), Control Unit and system clock, Immediate Access Store(IAS)
-  show understanding of how data are transferred between various components of the computer system using the address bus, data bus and control bus
-  show understanding of how the factors contribute to the performance of the computer system including:
  - processor types and cores
  - bus width
  - clock speed
  - cache memory
-  show understanding of the need for ports, including
  - Universal Serial Bus (USB), to provide the connection to peripheral devices
  - High Definition Multimedia Interface (HDMI)
  - Video Graphics Array (VGA)
-  **The fetch-execute cycle**
  - describe the stages of the fetch-execute cycles how understanding of 'register transfer' notation
  - possible causes and applications of interrupts are handled

## Von Neumann Model:

The earliest computers were not “programmable”. They were designed to do specific tasks only. **Reprogramming** when it was possible at all was a tedious process, starting with flowcharts and paper notes, followed by detailed engineering designs, and then the often process of physically re-wiring and re-building the machine.

It could take three weeks to set up a program on ENIAC (a computer of 1940s) and get it working. **ENIAC (Electronic Numerical Integrator and Computer) was the first electronic general-purpose computer. It was Turing-complete, digital, and capable of being reprogrammed to solve "a large class of numerical problems**

The **von Neumann architecture**, also known as the **von Neumann model** and **Princeton Architecture**, is based on John **von Neumann's** (mathematician and physicist) research paper in **1945** and others in the First Draft of a Report on the **EDVAC**.

**EDVAC** (Electronic Discrete Variable Automatic Computer) was one of the earliest electronic computers.

This described design architecture for an electronic digital computer with parts consisting

- 📌 **Central Processing Unit containing:**
  - 📌 **Control Unit**
  - 📌 **Arithmetic/Logic unit**
  - 📌 **Processor registers,**
  - 📌 **Memory to store data & instructions**
- 📌 **Input / Output Mechanism**
- 📌 **External Storage**

This describes **design architecture for an electronic digital computer with subdivisions of a central arithmetic part, a central control part, a memory to store both data and instructions, external storage, and input and output mechanisms.**

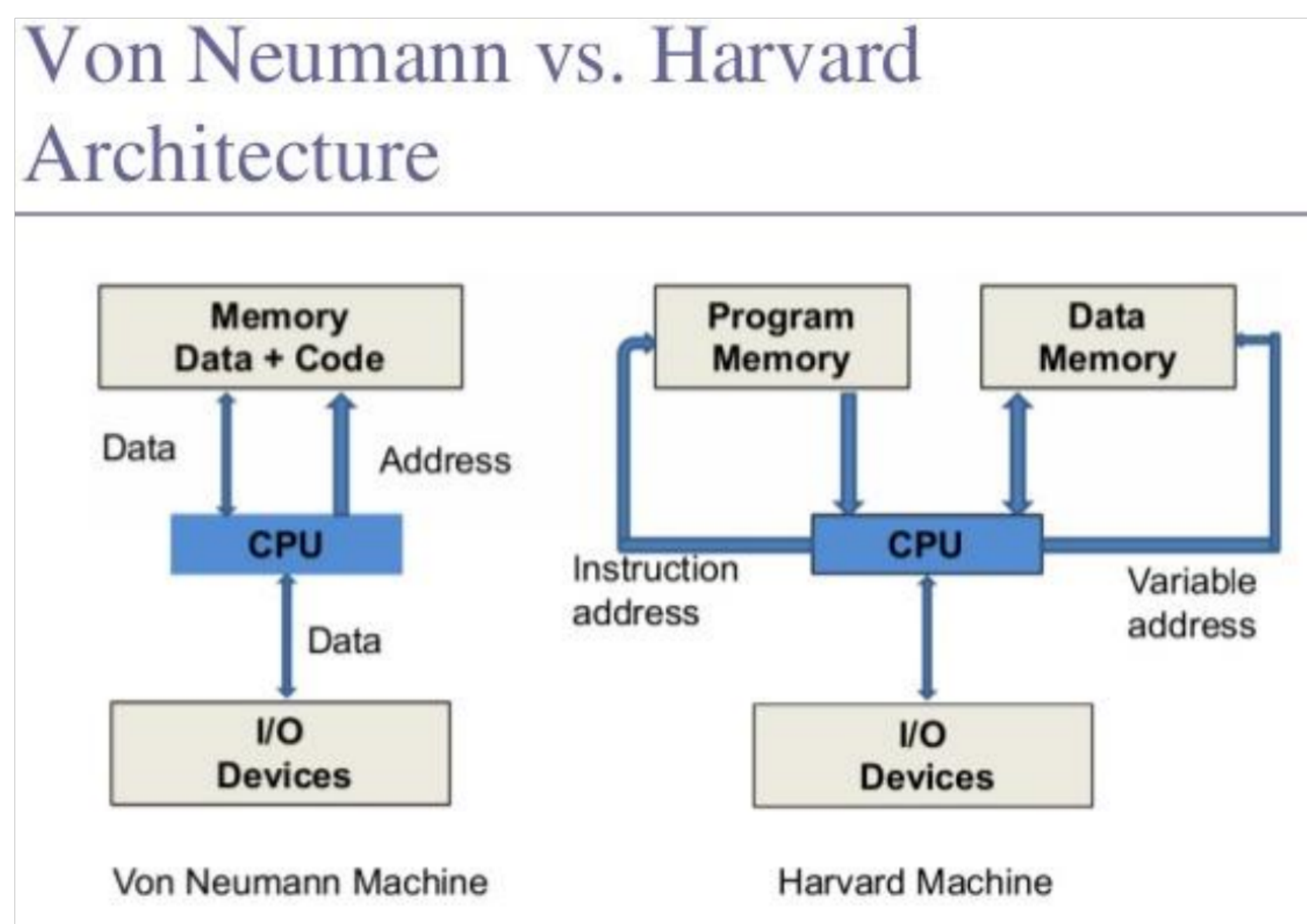
The meaning of the phrase has evolved to mean a stored-program computer. A stored-program digital computer is one that keeps its programmed instructions, as well as its data, in read-write, random-access memory (**RAM**)

So **John Von Neumann** introduced the idea of the **stored program**.

Previously data and programs were stored in separate memories. Von Neumann realized that data and programs are indistinguishable and can, therefore, use the same memory

The Von Neumann architecture uses a single processor which follows a linear sequence of fetch-decode-execute.

The Picture below shows difference between Von Neumann architecture and Harvard architecture (earliest computers)



## Features of a Von Neumann architecture

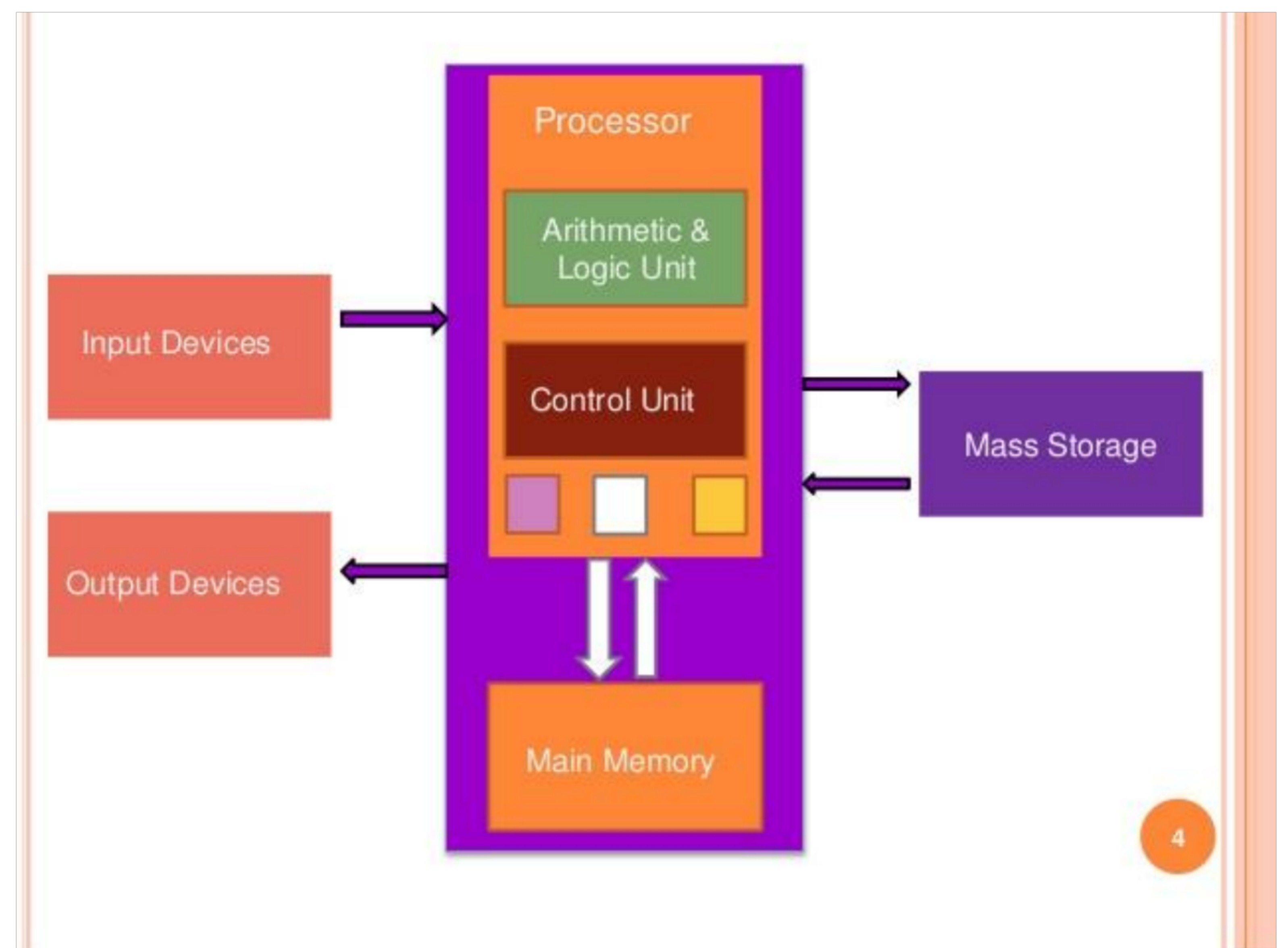
The illustration shows the essential features of the Von Neumann or stored-program architecture.

### Memory

The computer will have memory that can hold both data and also the program processing that data. In modern computers this memory is RAM.

### Control Unit

The control unit will manage the process of moving data and program into and out of memory and also deal with carrying out (executing) program instructions - one at a time. This includes the idea of a 'register' to hold intermediate values. In the illustration above, the 'accumulator' is one such register.



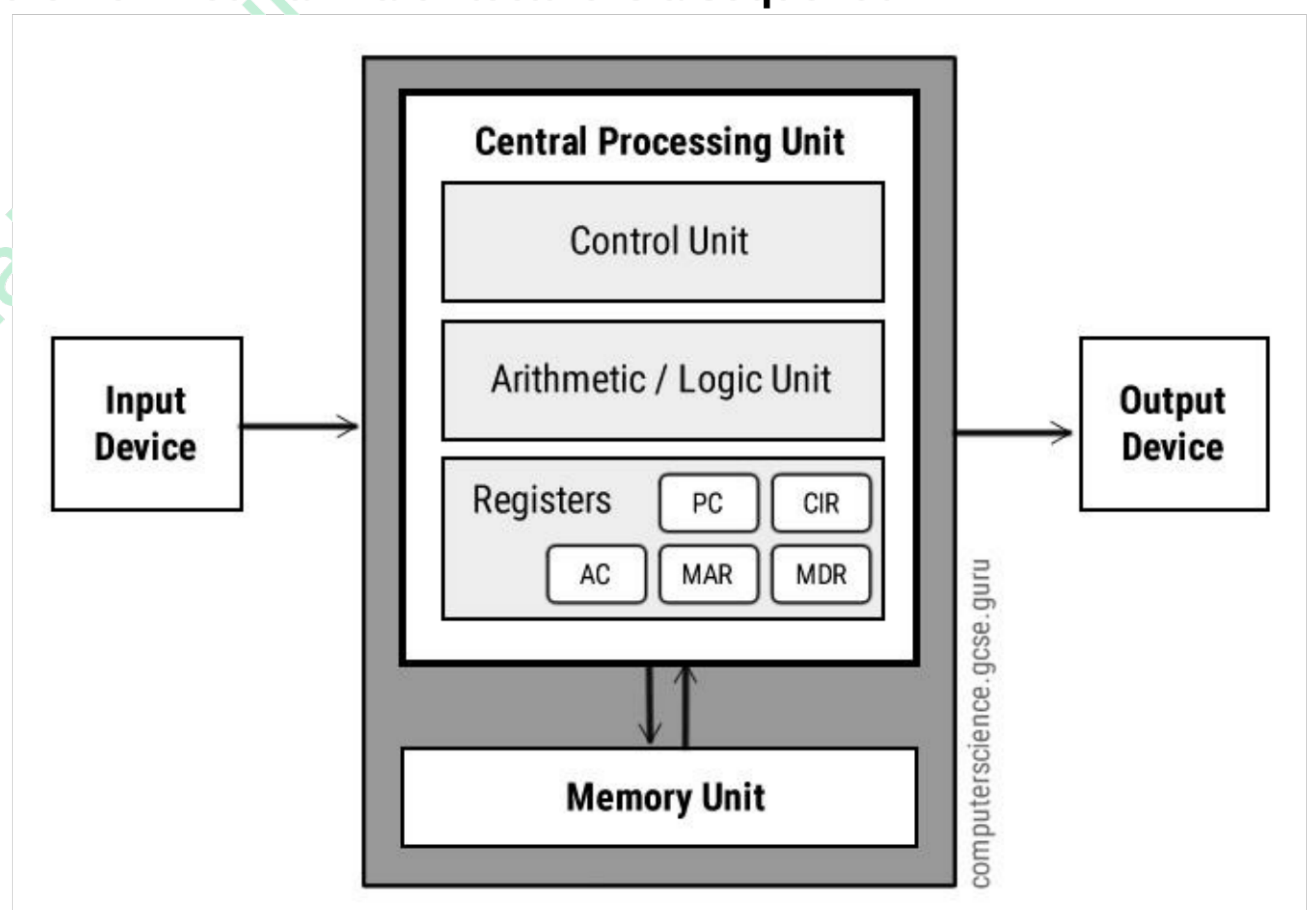
The 'one-at-a-time' phrase means that the Von Neumann architecture is a **sequential processing machine**.

### Input - Output

This architecture allows for the idea that a person needs to interact with the machine. Whatever values that is passed to and forth is stored once again in some internal registers.

### Arithmetic Logic Unit

This part of the architecture is solely involved with carrying out calculations upon the data. All the usual Add, Multiply, Divide and Subtract calculations **will be available but also data comparisons such as 'Greater Than', 'Less Than', 'Equal To' will be available**.



**Registers:** The Von Neumann architecture uses a single processor which follows a linear sequence of **fetch-decode-execute**. In order to do this, the processor has to use

some **special registers**, which are discrete memory locations with special purposes attached.

Register	Name/ Function
PC	<b>Program Counter</b> - keeps track of where to find the next instruction so that a copy of the instruction can be placed in the current instruction register
MAR	<b>Memory Address Register</b> - to hold the memory address that contains either the next piece of data or an instruction that is to be used.
MDR	<b>Memory Data Register</b> : acts like a buffer and holds anything that is copied from the memory ready for the processor to use it
CIR	<b>Current Instruction Register</b> : The current instruction register holds the instruction that is to be executed
IR/ IX	<b>Index Register</b> : is a register used for modifying operand addresses during the run of a program, typically for doing vector/array operations. Index registers are used for a special kind of indirect addressing.
Accumulator	<b>Accumulator</b> : This is simply the special register where data is worked on. Again, you can think of it as a box. If I wanted to add 4 to 7, for example, I would fetch 4 from RAM and put 4 in the Accumulator. I would then get 7 from RAM and add that to whatever was in the Accumulator. I would then store the result briefly in the Accumulator before moving it back to somewhere in RAM to be used later. All calculations of any description are done using the Accumulator. In fact, CPUs often have a few of these important registers, to help them process data quickly.
Processor Status Register	The Processor Status Register (abbreviated as P) is a <a href="#">hardware register</a> which records the condition of the <a href="#">CPU</a> as a result of arithmetic, logical or command operations. The purpose of the Processor Status Register is to hold information about the most recently performed <a href="#">ALU operation</a> , control the enabling and disabling of <a href="#">interrupts</a> and set the CPU operating mode.

## Special Purpose Register

A Special Function Register (or Special Purpose Register, or simply Special Register) is a register within a microprocessor, which controls or monitors various aspects of the microprocessor's function.

## General Purpose Registers

General purpose registers are available to store any transient data required by the program.

For example, when a program is interrupted its state, ie: the value of the registers such as the program counter, instruction register or memory address register - may be saved into the general purpose registers, ready for recall when the program is ready to start again. In general

the more registers a CPU has available, the faster it can work. **Accumulator** is a General Purpose Register.

## Bus

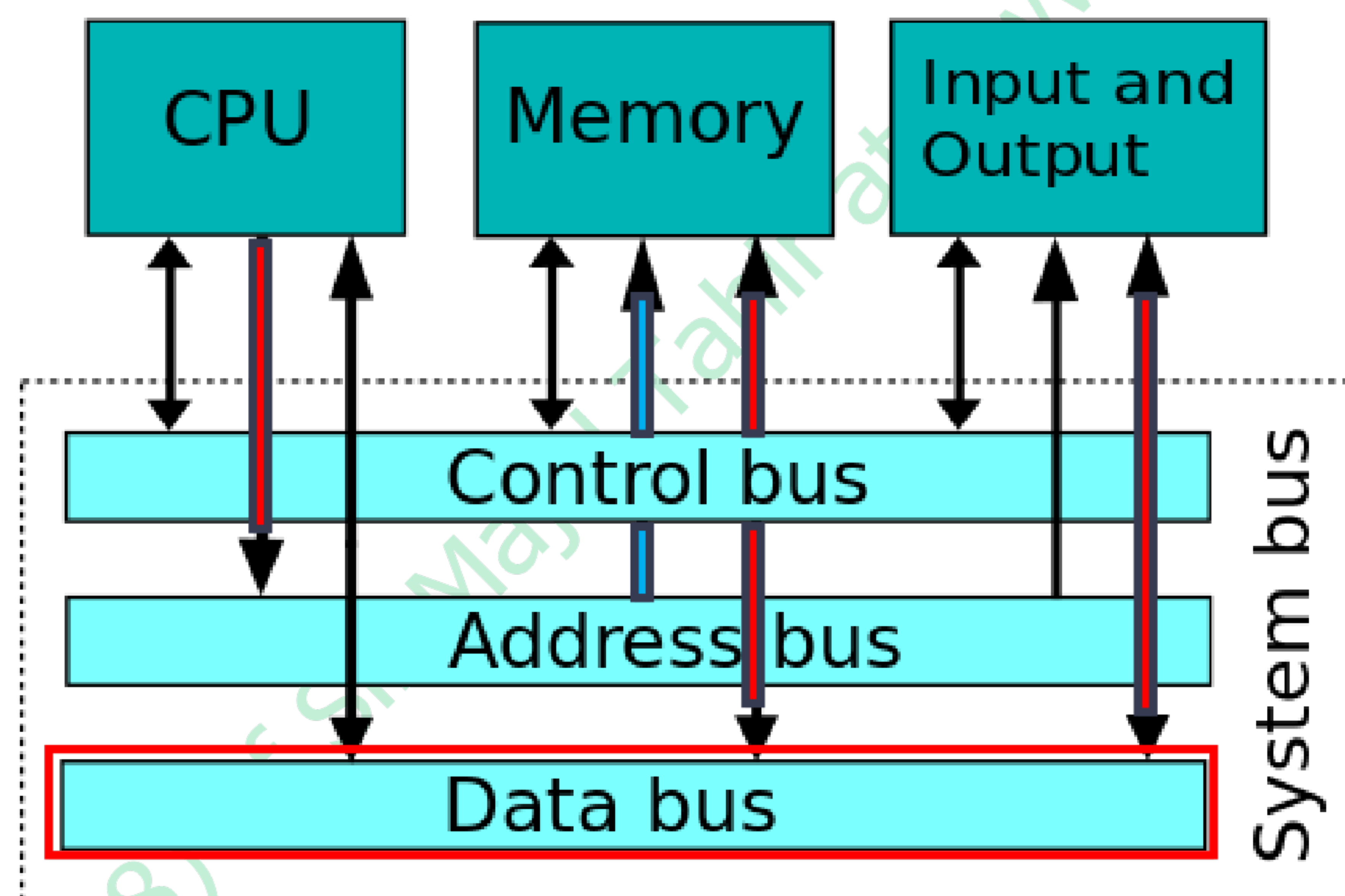
Notice the arrows between components? This implies that information should flow between various parts of the computer. In a modern computer built to the Von Neumann architecture, information passes back and forth along a 'bus'. There are buses to identify locations in memory - an 'address bus'

**Address Bus:** Address bus is unidirectional (single directional) bus that carries signals relating to memory addresses between processor and memory.

**Data Bus:** This bi-directional (two way traffic) bus is used to exchange data between processor, memory and input/output devices

### Control bus:

This bus that carries signals used to coordinate the computer's activities



## Bus Width

A bus is a channel over which information flows. The wider the bus, the more information can flow over the channel, much as a wider highway can carry more cars than a narrow one. The original ISA bus on the IBM PC was 8 bits wide; the universal ISA bus used now is 16 bits. The other I/O buses (including VLB and PCI) are 32 bits wide. The memory and processor buses on Pentium and higher PCs are 64 bits wide.

The address bus width can be specified independently of the data bus width. The width of the address bus dictates how many different memory locations that bus can transfer information to or from.

### System Clock:

Every computer contains an internal clock that regulates the rate at which instructions are executed and synchronizes all the various computer components. The CPU requires a fixed number of clock ticks (or **clock cycles**) to execute each instruction. The faster the clock, the more instructions the CPU can execute per second.

In order to synchronize all of a computer's operations, a system clock—a small quartz crystal located on the motherboard—is used. The system clock sends out a signal on a regular basis to all other computer components.

One full period is also called a clock cycle. On most modern systems, the system clock switches between zero and one at rates exceeding several million times per second.

The clock frequency is simply the number of clock cycles which occur each second. A typical 80486 chip runs at speeds of 66million cycles per second.

### Clock rate:

The speed at which a micro-processor executes instructions. One clock cycle is expressed in herdz(Hz). Clock speeds are expressed in megahertz (MHz) or gigahertz ((GHz).

### Clock Speed:

In a computer, clock speed refers to the number of pulses per second generated by an oscillator that sets the tempo for the processor. Clock speed is usually measured in MHz (megahertz, or millions of pulses per second) or GHz (gigahertz, or billions of pulses per second).

The operating speed of a computer, or its microprocessor, expressed in cycles per second (Megahertz or Gigahertz).

1 Megahertz is exactly **one** million **Hertz**.  $1 \text{ MHz} = 1 \times 10^6 \text{ Hz}$ .  $1 \text{ MHz} = 1000000\text{Hz}$ . **1 Cycle** per Second: A **period** of **1** second is equal to **1 Hertz** frequency.

### Port:

In computer hardware, a port serves as an interface between the computer and other computers or peripheral devices. In computer terms, a port generally refers to the female part of connection. Computer ports have many uses, to connect a **monitor**, **webcam**, **speakers**, or other **peripheral devices**.

On the **physical layer**, a computer port is a specialized outlet on a piece of equipment to which a **plug** or **cable** connects.

Electronically, hardware ports can almost always be divided into two groups based on the signal transfer:

After ports are connected, they typically require **handshaking**, where **transfer type**, **transfer rate**, and other necessary information is shared before data are sent.

## Universal Serial Bus (USB)

The **UNIVERSAL SERIAL BUS (USB)** is an asynchronous serial data transmission method

- It has quickly become the standard method for transferring data between a computer and a number of devices. Essentially the USB cable consists of:
  - a four-wire shielded cable
  - two of the wires are used for power and the earth
  - two of the wires are used in the data transmission.

When a device is plugged into a computer using one of the USB ports:

- the computer automatically detects that a device is present (this is due to a small change in the voltage level on the data signal wires in the cable)
- the device is automatically recognized, and the appropriate **DEVICE DRIVER** is loaded up so that computer and device can communicate effectively
- if a new device is detected, the computer will look for the device driver which matches the device; if this is not available, the user is prompted to download the appropriate software.

✓	X
Devices plugged into the computer are automatically detected; device drivers are automatically uploaded	–
The connectors can only fit one way; this prevents incorrect connections being made	The maximum cable length is presently about 5 metres
This has become the industry standard; this means that considerable support is available to users	–
Several different data transmission rates are supported	The present transmission rate is limited to less than 500 megabits per second
Newer USB standards are backward compatible with older USB standards	The older USB standard (e.g. 1.1) may not be supported in the near future

## Specialised multimedia ports

Despite the widespread use of USB ports there are some peripheral devices that require a different port, one that is specialised for the type of device. Although computer systems come packaged with a monitor for screen display there is sometimes a requirement for a second screen to be used.

The connection of the second screen can be through a **Video Graphics Array (VGA)** port. This provides high-resolution screen display which is suitable for most display requirements. However, if the screen is needed to display a video, the VGA port is not suitable because it does not transmit the audio component.

## High Definition Multimedia Interface (HDMI)

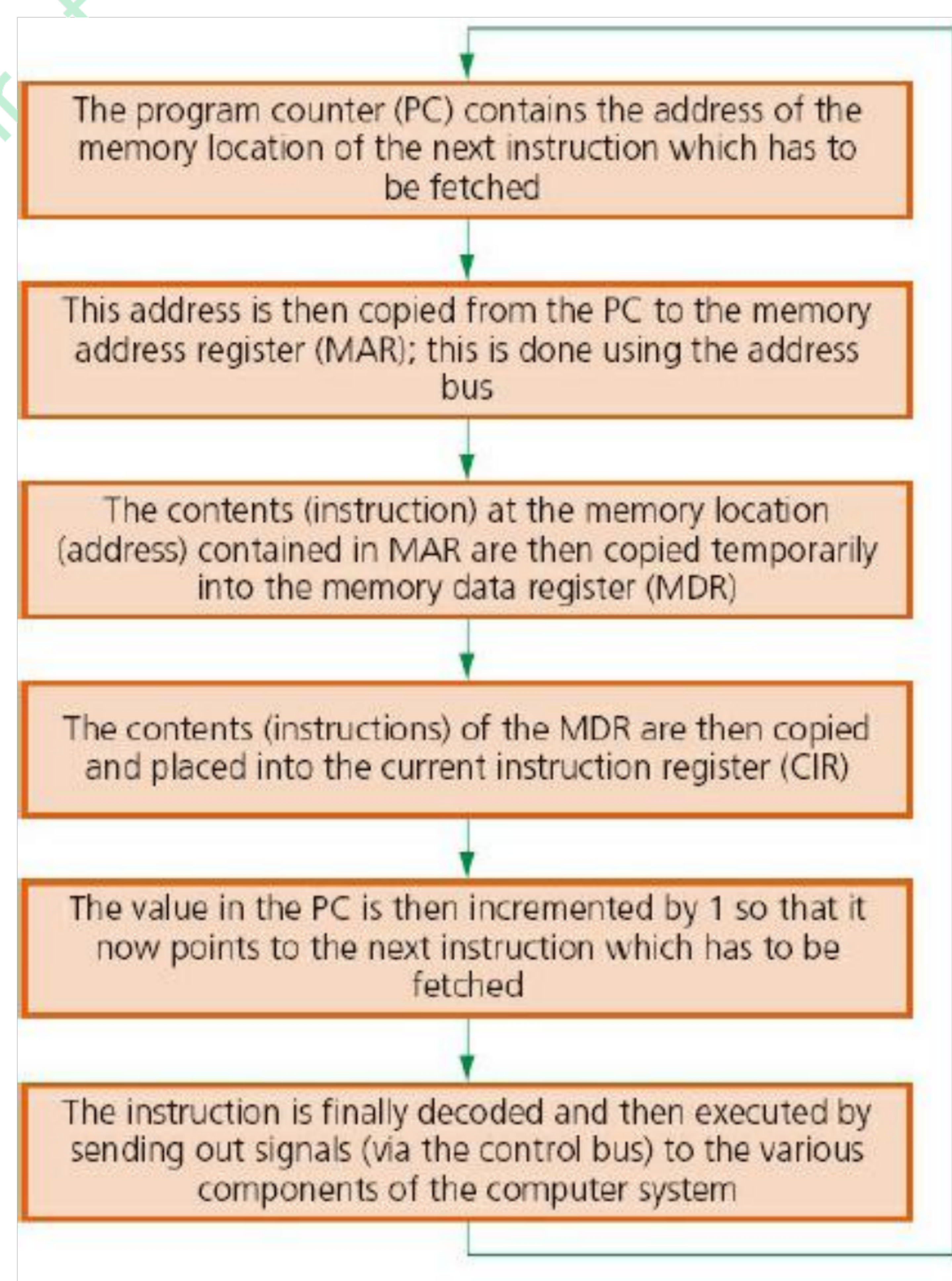
A High Definition Multimedia Interface (HDMI) port will provide a connection to a screen and allow the transmission of high-quality video including the audio component.

# CPU and Fetch-Execute Cycle

## Fetch-Decode-Execute-Reset Cycle:

The following is an algorithm that shows the steps in the cycle. At the end the cycle is reset and the algorithm repeated.

1. Load the address that is in the program counter (PC) into the memory address register (MAR).
2. Load the instruction that is in the memory address given by the MAR into the memory data register (MDR).
3. Load the instruction that is now in the MDR into the current instruction register (CIR).
4. Increment the PC by 1.
5. Decode the instruction that is in the CIR.
6. If the instruction is a jump instruction then
  - a. Load the address part of the instruction into the PC
  - b. Reset by going to step 1.
7. Execute the instruction.
8. Reset by going to step 1



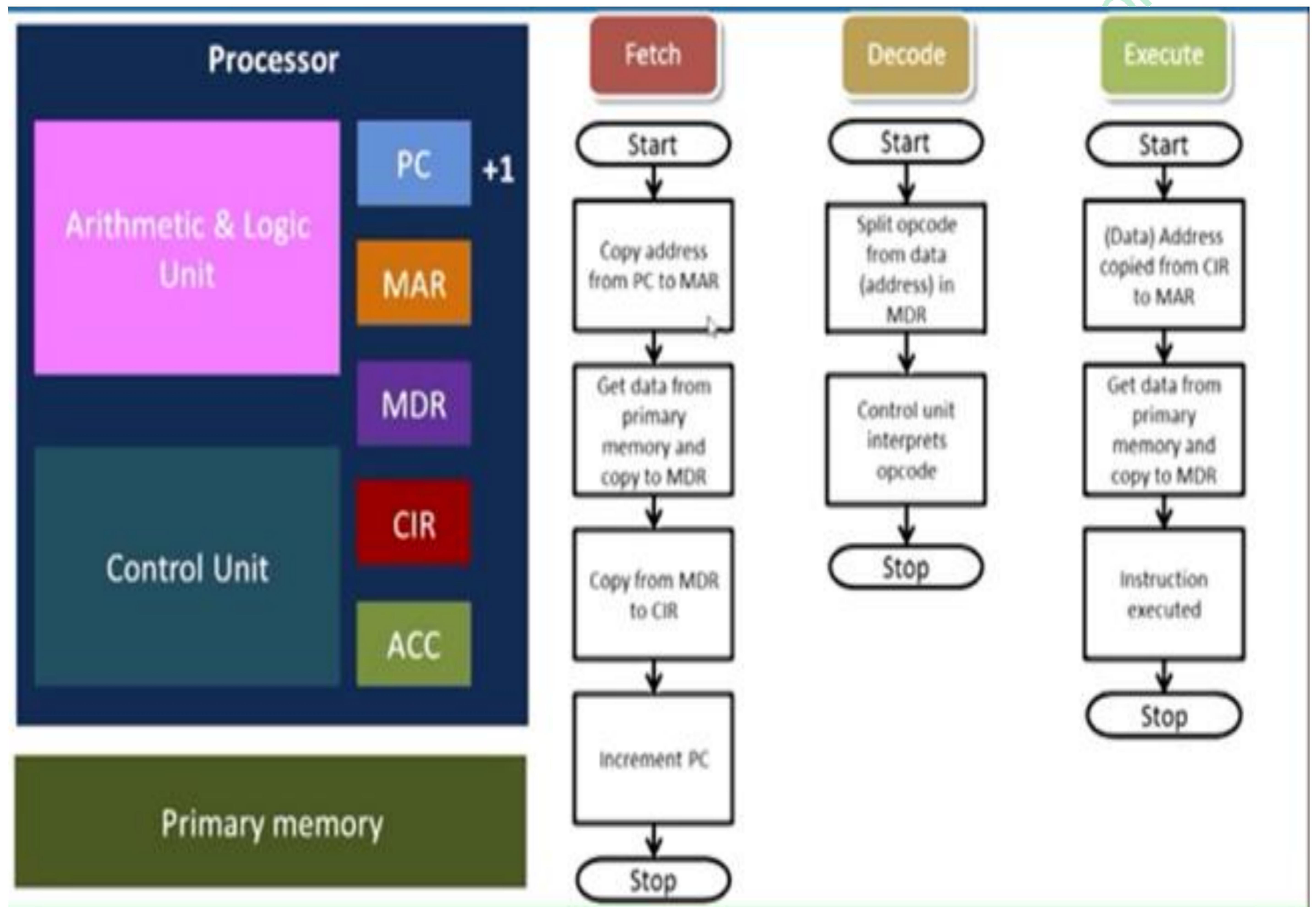
Steps 1 to 4 are the *fetch* part of the cycle.



Steps 5, 6a and 7 are the execute part of the cycle and steps 6b and 8 are the reset part.

**Step 1** simply places the address of the next instruction into the memory address register so that the control unit can fetch the instruction from the right part of the memory.

The **program counter** is then incremented by 1 so that it contains the address of the next instruction, assuming that the instructions are in consecutive locations.



The **MDR memory data register** is used whenever anything is to go from the central processing unit to main memory, or vice versa. Thus the next instruction is copied from memory into the MDR and is then copied into the current instruction register.

Now that the instruction has been **fetch**ed the control unit can **decode** it and decide what has to be done. This is the execute part of the cycle.

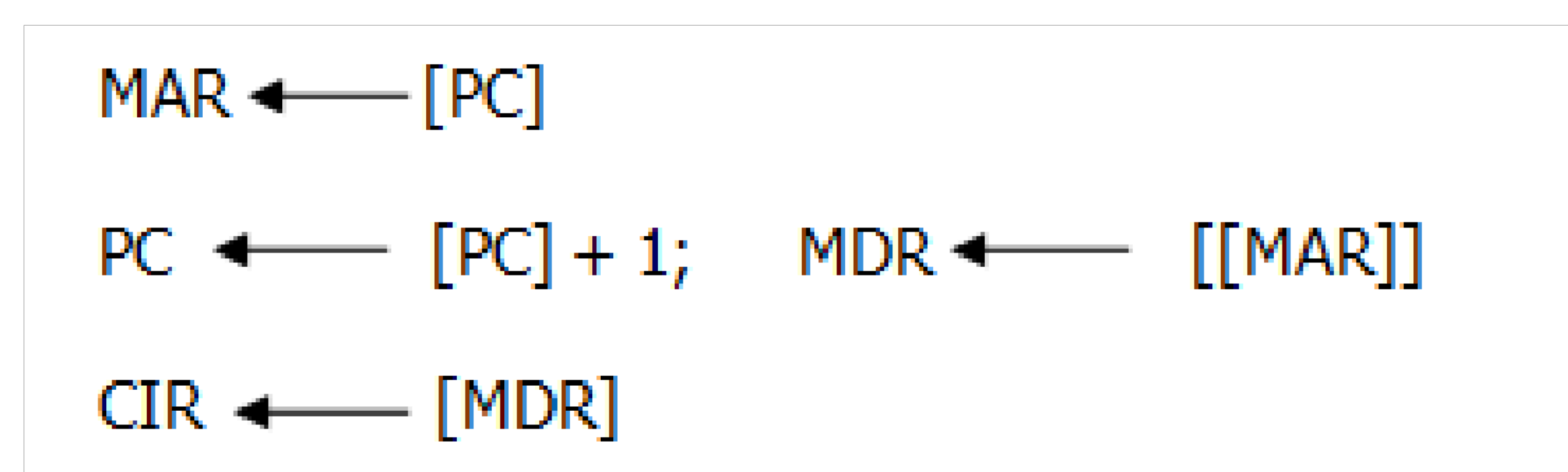
If it is an arithmetic instruction, this can be executed and the cycle restarted as the PC contains the address of the next instruction in order. However, if the instruction involves jumping to an instruction that is not the next one in order, the PC has to be loaded with

the address of the instruction that is to be executed next. This address is in the address part of the current instruction, hence the address part is loaded into the PC before the cycle is reset and starts all over again.

### Register Transfer Notation:

**Register Transfer Notation** (or RTN) is a way of specifying the behavior of a digital synchronous circuit.

Operations involving registers can be described by register transfer notation. The simplest form of this can be illustrated by the following representation of the fetch stage of the fetch execute cycle:



The basic format for an individual data transfer is similar to that for variable assignment.

The first item is the **destination of the data**. Here the appropriate abbreviation is used to identify the particular register.

To the right of the arrow showing the **transmission of data** is the definition of this data.

In this definition, the **square brackets** around a register abbreviation show that the content of the register is **being moved possibly with some arithmetic operation** being applied.



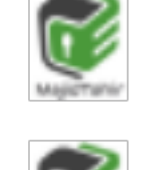


When two data operations are placed on the same line separated by a **semi-colon** this means that the **two transfers take place simultaneously**.

The double pair of brackets around MAR on the second line needs careful interpretation.

The content of the MAR is an address; it is the content of that address which is being **transferred to the MDR**.

## Interrupt handling

There are many different reasons for an interrupt to be generated. Some examples are:

-  fatal error in a program
-  hardware fault
-  need for I/O processing to begin
-  user interaction
-  a timer signal.

There are a number of different approaches possible for the detailed mechanisms used to handle interrupts but the overriding principles are clearly defined.

Each different interrupt needs to be handled appropriately and different interrupts might possibly have different priorities. Therefore, the processor must have a means of identifying the type of interrupt.

One way is to have an interrupt register in the CPU that works like the status register, with each individual bit operating as a flag for a specific type of interrupt.

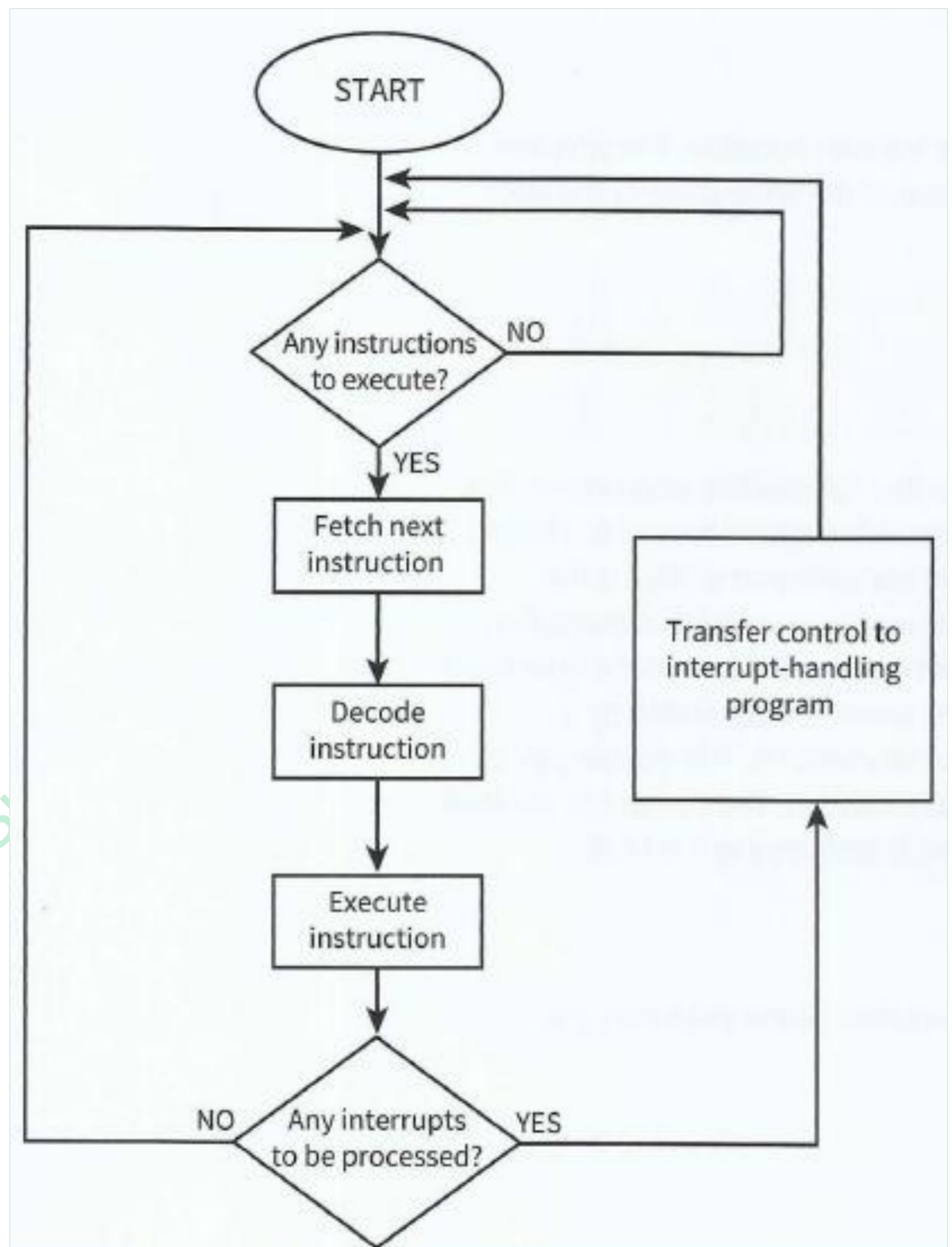
As the flowchart in Figure shows, the existence of an interrupt is only detected at the end of a fetch-execute cycle.

This allows the current program to be interrupted and left in a defined state which can be returned to later.

The first step in handling the interrupt is to store the contents of the program counter and any other registers somewhere safe in memory.

Following this, the appropriate interrupt handler or interrupt service routine (ISR) program is initiated by loading its start address into the program counter.

When the ISR program has been executed there needs to be an immediate check to see if further interrupts need handling. If there are none, the safely stored contents of the registers are restored to the CPU and the originally running program is resumed.



## Interrupt handling

There are many different reasons for an interrupt to be generated. Some examples are:

- a fatal error in a program
- a hardware fault
- a need for I/O processing to begin
- user interaction
- a timer signal.

Interrupts are handled by a number of different mechanisms, but there are some clear overriding principles. Each different interrupt needs to be handled appropriately. Different interrupts might have different priorities. Therefore, the processor must have a means of identifying the type of interrupt. One way is to have an interrupt register in the CPU that works like the status register, with each individual bit operating as a flag for a specific type of interrupt.

As the flowchart in Figure above shows, the existence of an interrupt is only detected at the end of a fetch–execute cycle. This allows the current program to be interrupted and left in a defined state which can be returned to later. An interrupt is handled by the following steps.

- The contents of the program counter and any other registers are stored somewhere safe in memory.
- The appropriate interrupt handler or Interrupt Service Routine (ISR) program is initiated by loading its start address into the program counter.
- When the ISR program has been executed there is an immediate check to see if further interrupts need handling.
- Further interrupts are dealt with by repeated execution of the ISR program.
- If there are no further interrupts, the safely stored contents of the registers are restored to the CPU and the originally running program is resumed.

### Past paper Question June 2015

#### 7 (a)

Three buses and three descriptions are shown below.

Draw a line to connect each bus to its correct description.

Bus	Description
address bus	this bus carries signals used to coordinate the computer's activities
control bus	this bi-directional bus is used to exchange data between processor, memory and input/output devices
data bus	this uni-directional bus carries signals relating to memory addresses between processor and memory

[2]

(b) The seven stages in a von Neumann fetch-execute cycle are shown in the table below. Put each stage in the correct sequence by writing the numbers 1 to 7 in the right hand column.

The first one has been done for you.

Stage	Sequence number
the instruction is then copied from the memory location contained in the MAR (memory address register) and is placed in the MDR (memory data register)	
the instruction is finally decoded and is then executed	
the PC (program counter) contains the address of the next instruction to be fetched	<b>1</b>
the entire instruction is then copied from the MDR (memory data register) and placed in the CIR (current instruction register)	
the address contained in the PC (program counter) is copied to the MAR (memory address register) via the address bus	
the address part of the instruction, if any, is placed in the MAR (memory address register)	
the value in the PC (program counter) is then incremented so that it points to the next instruction to be fetched	

References:

Computer Science book by Sylvia Langfield & Dave Duddell, [www.wikipedia.com](http://www.wikipedia.com)  
IGCSE Computer Science by Hodder Education, VCN – ICT Department 2013 Prepared by Davis Rwatooro T and PAST PAPERS

[6]

Notes (9618) of Sir Majid