Overview of Topic:
Processors, I/O/S and components of a computer

- The structure and function of the CPU
- Registers
- The FDE cycle
- The performance of a CPU
- Different types of processor
- CISC, RISC, multicore and parallel systems
- Input, output and storage

Key Terms

Technical Terms

Hardware
Software
Peripherals
Input
Output
CPU
Main memory (RAM)
Registers

Specific Vocab

Load
Store
Instructions
Data
Programs
Processing
Throughput
Architecture
Contents

What is a computer?

A computer is made up of 'hardware' and 'software'.

These work together to make up a complete computer system.

This computer system can then process instructions and data using the brain of the computer, known as the Central Processing Unit (CPU).

Hardware, software and peripherals

Hardware	**Software**
is anything that is physical, that you can touch Examples: keyboard, a web cam, a stick of RAM, a CPU chip or a pen drive.	are the programs (or sets of instructions) that make the hardware do useful things. Examples: word processing program operating system web browser Virus checker

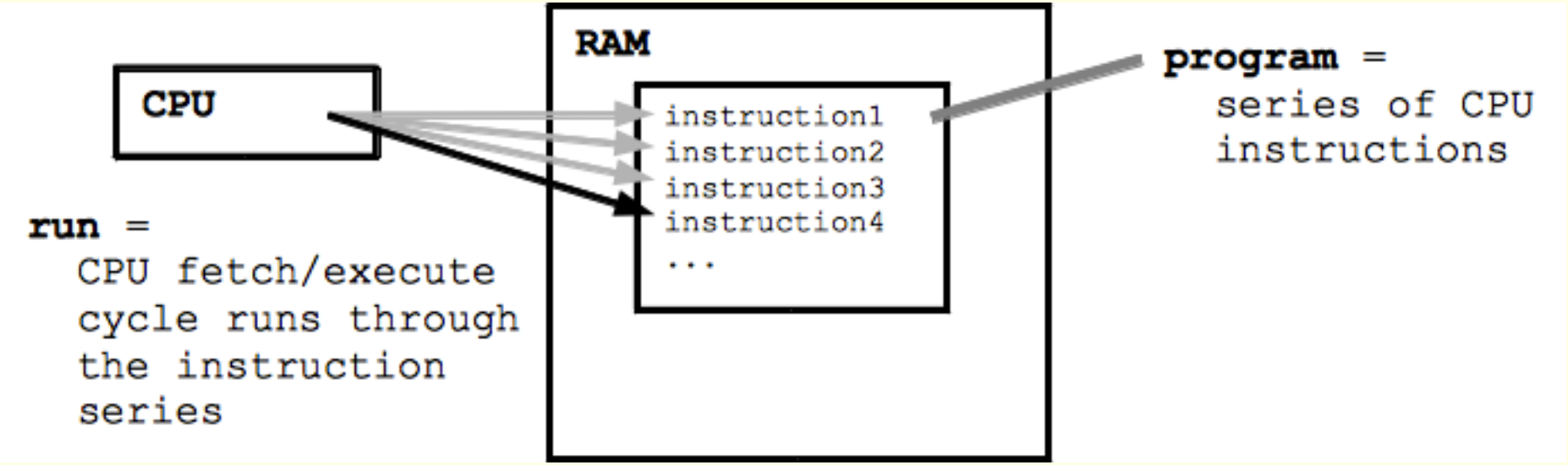
What is a peripheral?

Simply any piece of hardware that you can connect to the computer. Normally, you would do this by plugging it into somewhere on the outside of your computer system, such as a USB port or by using a wireless connection such such as Bluetooth.

When you plug a peripheral into your computer, you are making a direct connection from the peripheral to the CPU, so it can be used.

Role of CPU

- The CPU processes data according to sets of instructions, or programs, written by programmers.
- All the programs that have been **loaded** onto a computer are **stored** on the **hard disk**.
- When one needs to be used, a copy of that program is made and put into RAM.
- The CPU then works with this program in RAM.
- The CPU cannot directly work with any programs or data held on a storage medium such as a hard drive or pen drive.
- If it wants to use them, it first has to move them into RAM.



Stored programs

- In 1945, the mathematician and physicist John von Neumann, working at the University of Princeton USA, published a paper about the computer he and his team had designed and built.

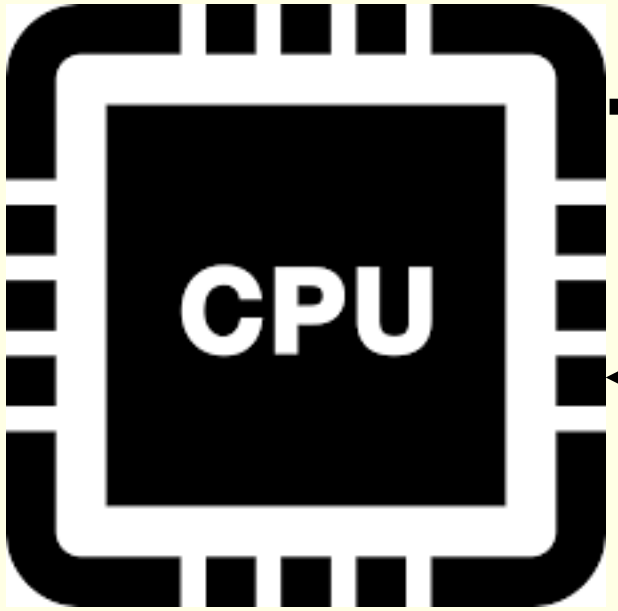
- It was the first computer that used the basic component architecture we recognise in modern computers.

- **He identified that data and programs could be stored in the same memory known as the stored program concept**

Stored Program Concept

Storing instructions and data in main memory
Instructions are then fetched from memory, decoded and executed by the processor.

Storing Instructions and data



Memory Address	
0001	LOAD 0101
0010	ADD 0110
0011	STORE 0111
0100	STOP
0101	23
0110	12
0111	

Main Memory

Purpose of Main memory

Also described as:

- Random Access Memory (RAM)
- primary memory
- Immediate Access Store (IAS)

The place where **programs** and the **data** that is needed by programs are held, ready to be fetched then decoded and executed by the CPU.

The operating system, the programs you are **currently** using and any data you are currently using are worked on by the CPU from the main memory, not from storage.

The CPU does not work directly with storage devices, only the main memory.

Main memory is made up of lots of individual **'memory' locations**, each capable of storing a **byte of data**.

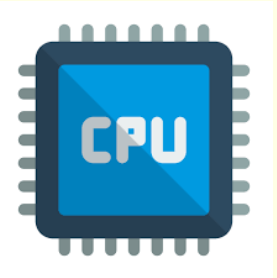
Example

Open an application or a file

moved from storage on your hard disk

main memory

CPU



save work

moved from the main memory

storage device

CPU

The Central Processing Unit, or CPU, is the central part of a computer, the brain that does all of the computations. Because the CPU performs many different functions, you need to divide it up into its main parts to understand it. Only then can you successfully describe what it does.

There are four component parts that need a mention.

These are:

Arithmetic Logic Unit (known as the ALU).
Control Unit
Registers
Main memory

ALU

The Arithmetic Logic Unit is that part of the CPU that does all the calculations. It has electronic circuits that can manipulate data in various ways.

It has three main functions.

Perform **arithmetic calculations** on data. For example, it can add and subtract two numbers together or multiply and divide numbers (in binary, of course).	Perform **logical operations** on data. These are computations that involve, for example, the use of AND, OR and NOT.	**Hold data** it has already worked on, ready to be sent out. **Hold data** that has been fetched, ready to be processed. Acts as the pathway for all input and output operations, acting as the **gateway between the processor and external devices.**

Control Unit

This part of the CPU is responsible for managing how instructions are executed.

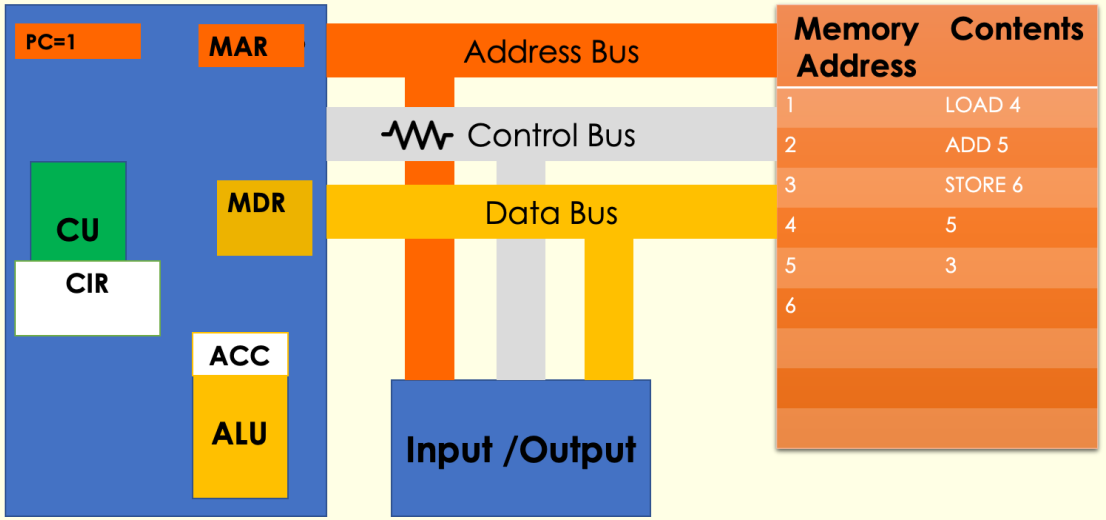
1. **Decode** an instruction
2. The **Control Unit** directs the flow of data and instructions within the CPU.
3. It send control signals to coordinate the operations of the **ALU**, **memory**, **and input/output devices** so that instructions are carried out in the correct sequence.

What can you remember?

Clock:
Regular electrical pulse which synchronises all the components.

Clock speed:
number of instructions that can be performed in any given moment of time.

Address bus
Carries memory addresses from the processor to other components such as RAM and input/output devices.



Registers
are small amounts of high speed *memory* contained within the CPU.

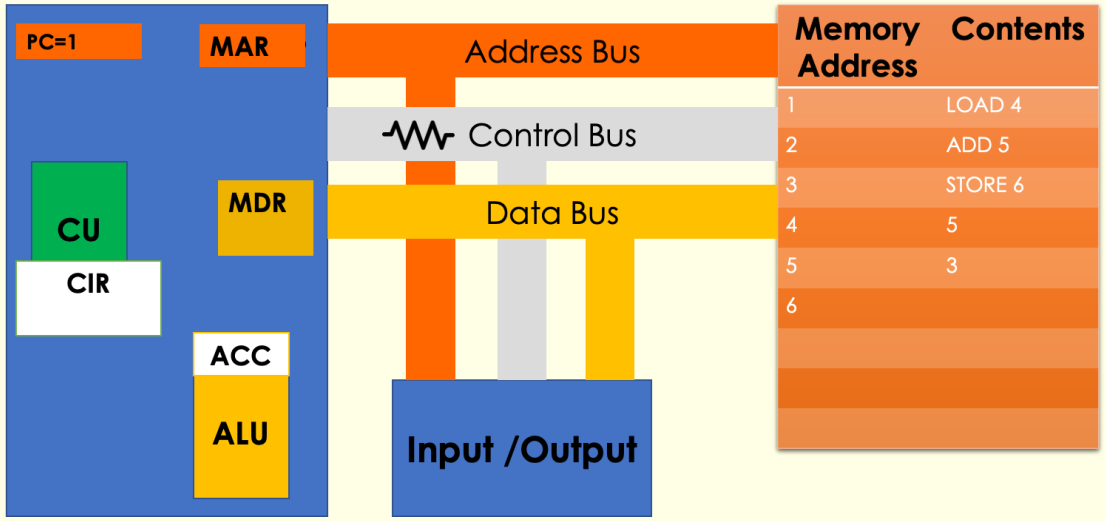
Data bus
carries the actual data between the processor and other components.

Control bus

carries control signals from the processor to other components.

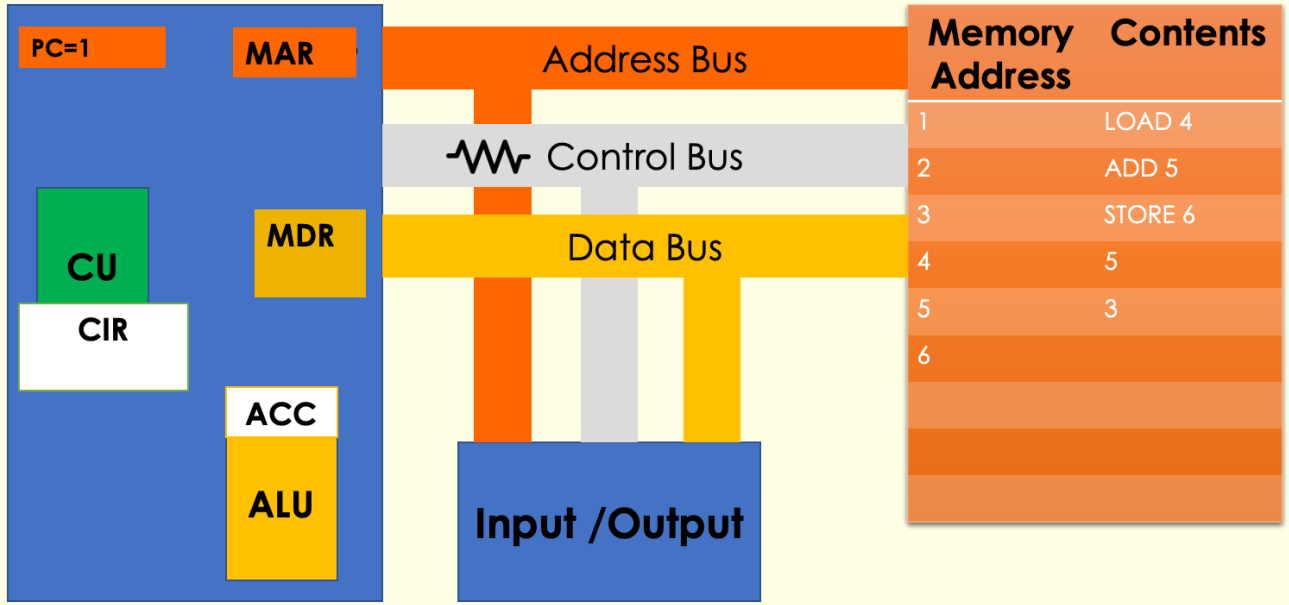
Registers

- Type of memory that can be accessed very quickly compared to other types of memory.
- The contents hold are needed by the CPU to run each program instruction during a 'fetch-decode-execute cycle' or they can be used to hold values that are generated as part of the ALU working on data.
- There are a number of very special registers that do very specific jobs.



CPU Registers - Registers are small areas of memory in the CPU

- **Program Counter** – keeps track of where the CPU is in the program. Points to the next instruction in the cycle.
- **Memory Address Register (MAR)** – holds the address of the instruction to be fetched.
- **Memory Data Register (MDR)** – stores the instruction about to be executed.
- **Accumulator** – Temporary storage for data being processed during calculations by the ALU
- **Current instruction register (CIR)** - instruction to be executed



ALU

Computer architectures use registers including the accumulator.

Describe the purpose of the accumulator. [2]

Temporary storage for data being processed during calculations

I/O in processor used as a buffer

A **buffer** is a temporary storage area used to hold data while it is being transferred from one place to another.

Input/Output (I/O) buffering is a process that uses a buffer to store data temporarily while it's being transferred between the processor and other devices.

Data, address and control busses

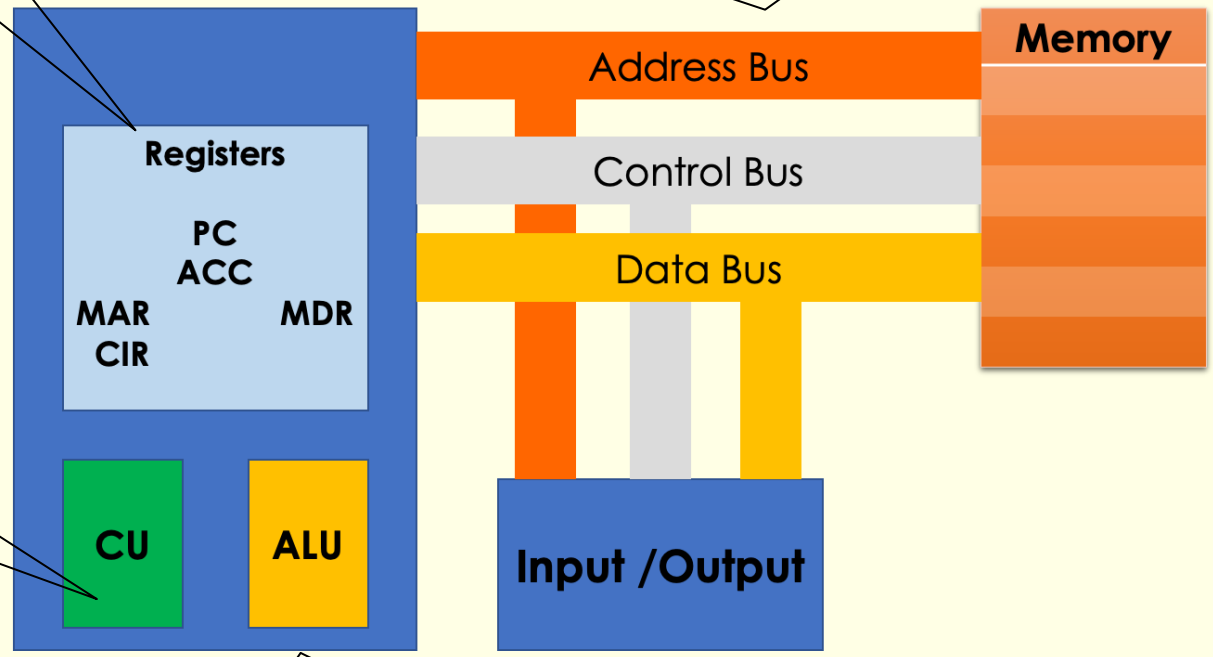
A bus is a set of parallel wires / channels connecting two or more independent components of a computer system in order to pass signals between them.

Control Bus	Data Bus	Address Bus
■ Control signals are **sent** along the control bus ■ E.g. Memory Read, Memory Write ■ This instructs which data will be travelling to/from memory.	■ **Carries** Data/Instructions from Main Memory to the Processor (or from other secondary storage devices) to the processor. ■ Bi-Directional (two way) ■ Data can be read/written	■ **Carries** addresses from the Processor to main memory ■ It is one direction (Uni-Directional) ■ The processor generates an address

Registers
Registers: provide fast, temporary storage, e.g. registers for designated functions, such as holding the results of operations, holding instructions, and holding other data

Buses
- Address: holds the address of the memory location involved in the instruction
- Data: holds the value to be read from or written to memory
- Control: carries signals from the control unit to other components to coordinate operations

Main Memory
a collection of storage locations



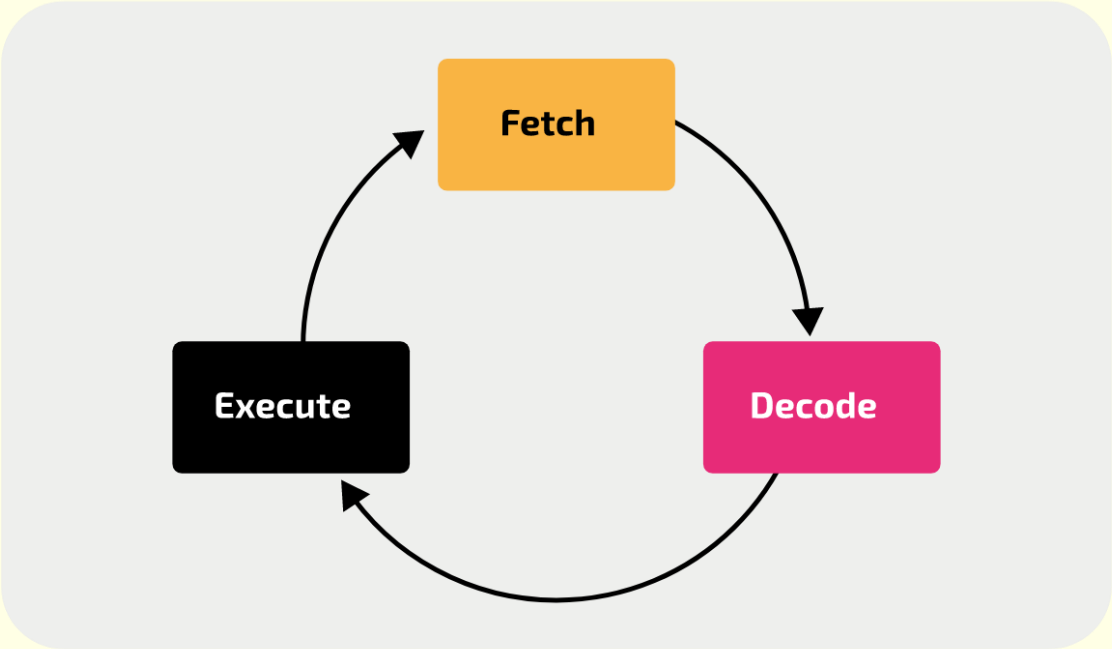
Control Unit
Control unit: decodes instructions received from main memory and provides a signal to coordinate operations

ALU
performs arithmetic and logical operations

- **Program Counter** – keeps track of where the CPU is in the program. Points to the next instruction in the cycle.
- **Memory Address Register (MAR)** – holds the address of the instruction to be fetched.
- **Memory Data Register (MDR)** – stores the instruction about to be executed.
- **Accumulator** – stores the most recent result of processing.
- **Current instruction register (CIR)** - The instruction to be decoded is held here.

Learning Aims

- Be able to describe the **FDE cycle**



Fetch next instruction from memory
Decode instruction
Execute the instruction

Key Terms

Fetch
Decode
Execute
Data
Instructions
Memory Address

MAR
MDR
CIR
Program counter
Accumulator

Buses (Address, data and control bus)
Contents
Copies
Read/Write
Carries data

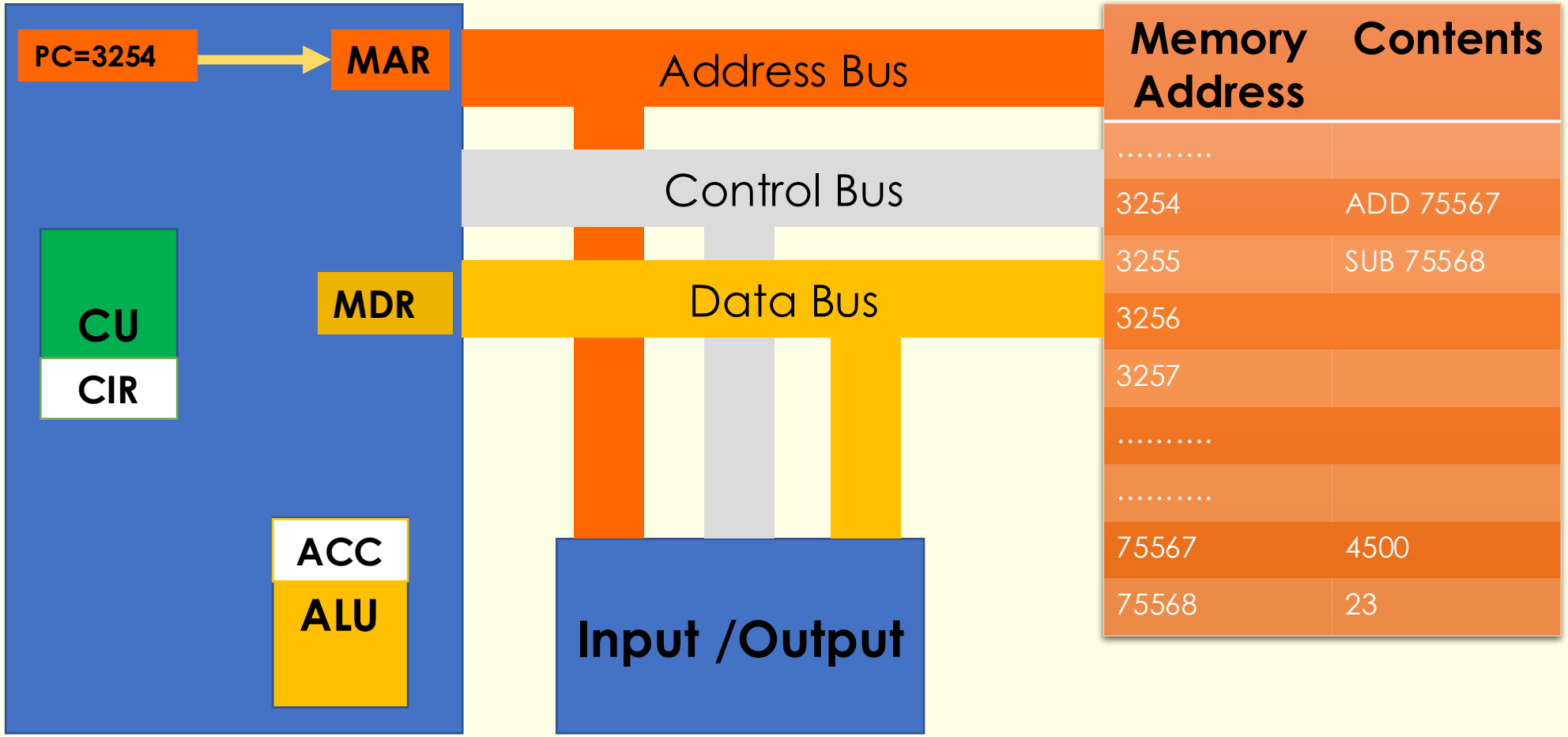
Racap: Registers and the Fetch-Decode-Execute cycle

The registers you should know about include:

- **Program Counter (PC)** - this holds the address of the next instruction to be fetched and executed.
- **Current Instruction Register (CIR)** - this holds the current instruction being executed.
- **Memory Address Register (MAR)** - this holds the RAM address you want to read to or write from.
- **Memory Data Register (MDR)** - this holds the data you have read from RAM or want to write to RAM.
- **Accumulator** - hold the data being worked on and the results of arithmetic and logical operations.
- **Interrupt Register** - this holds details about whether an interrupt has happened.

Example F-D-E

Using registers to execute an instruction in a program.
Consider the following situation:



Note in the above that we have not used binary either for the RAM address or the contents, to make things easier to understand! In reality everything in RAM would be stored as 0's and 1's

Before we begin let's understand what instructions look like in memory

Memory Address	Contents
.....	
3254	ADD 75567
3255	SUB 75568
3256	
3257	
.....	
.....	
75567	4500
75568	23

Processors have a specific **instruction set**.

Here are some examples of the operations a CPU can perform.

Mnemonic	Action
LDA	Loads a value from a memory address
STA	Stores a value in a memory address
ADD	Adds the value held in a memory address to the value held in the accumulator
SUB	Subtracts from the accumulator the value held in a memory address
MOV	Moves the contents of one memory address to another

Address bus and word sizes

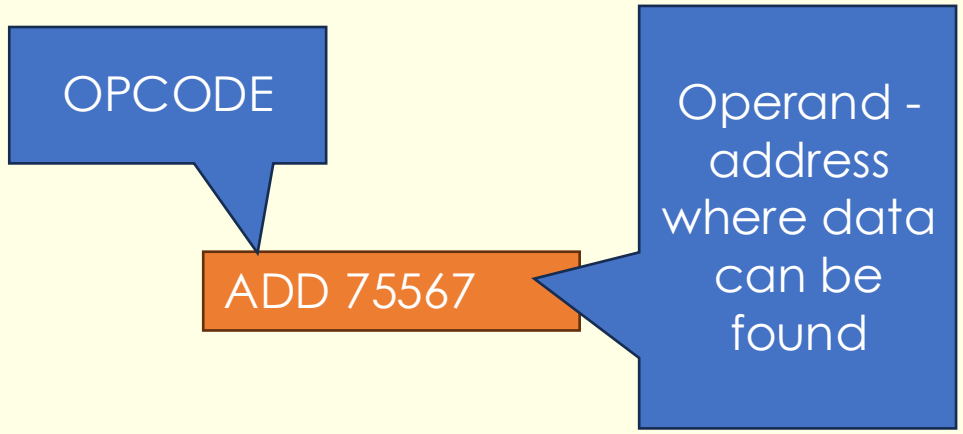
Memory is divided up internally into units called **words**.

A word is a fixed size group of digits, typically 16, 32 or 64 bits, which is handled as a unit by the processor, and different types of processor have different **word sizes**.

Each word in memory has its own specific address.

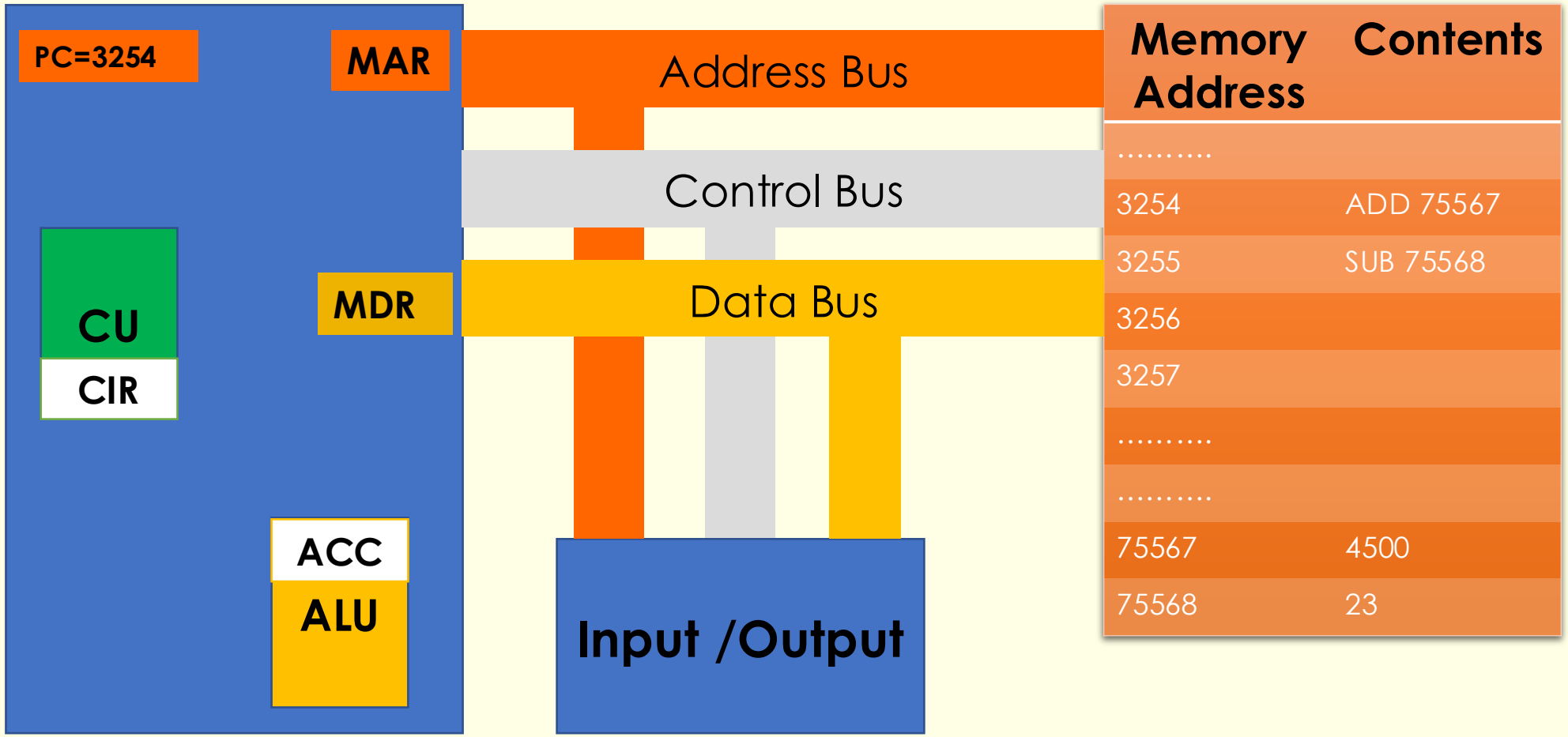
The **address bus** transmits the memory addresses of words that are used as **operands** in program instructions, so that the data can be retrieved and sent back to the processor.

When an instruction has been performed and the result is to be stored at a particular memory location, it is transmitted via the data bus.



Steps in the fetch process

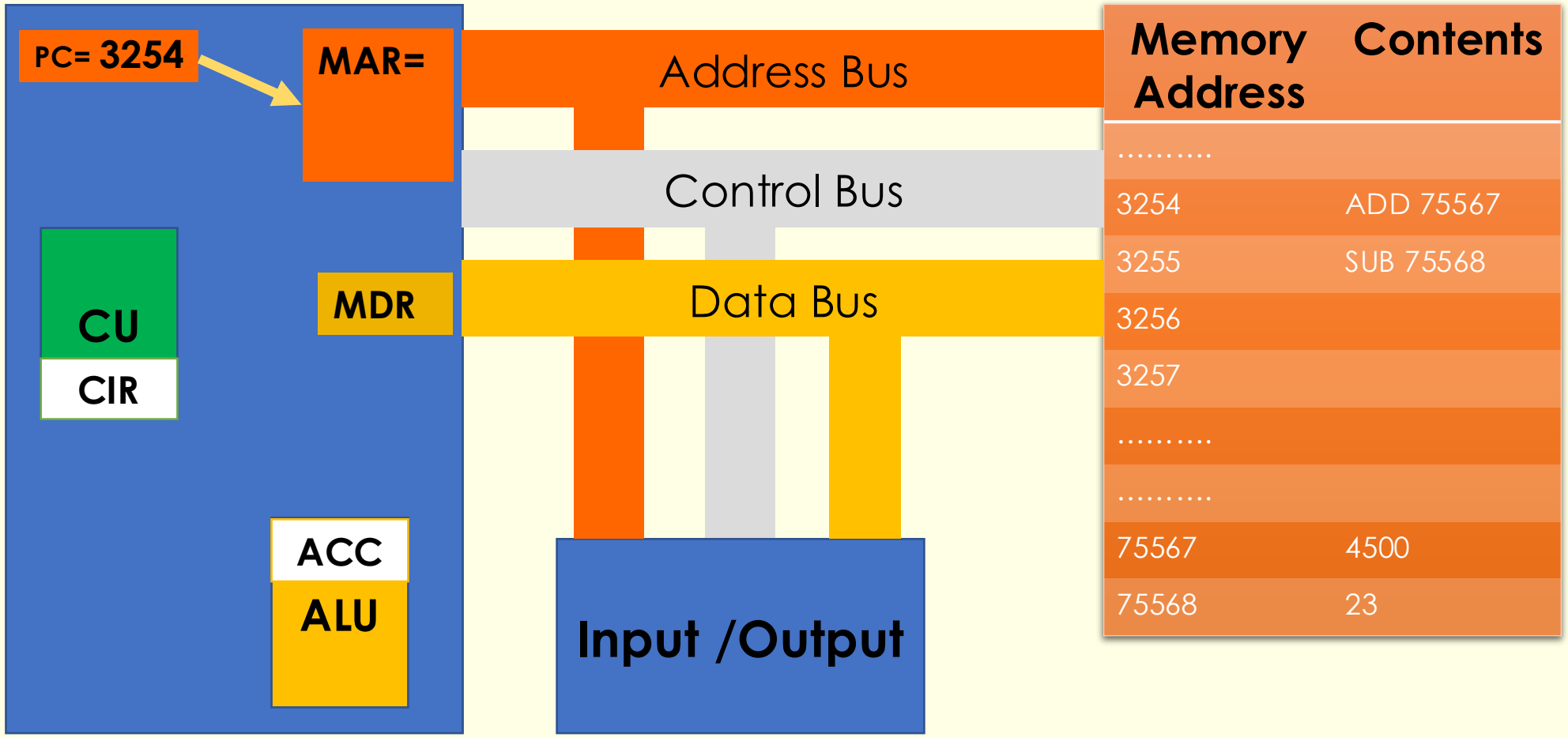
1. The CPU reads the contents of the Program Counter to find the address of the next instruction to be fetched, decoded and executed.



Steps in the fetch process

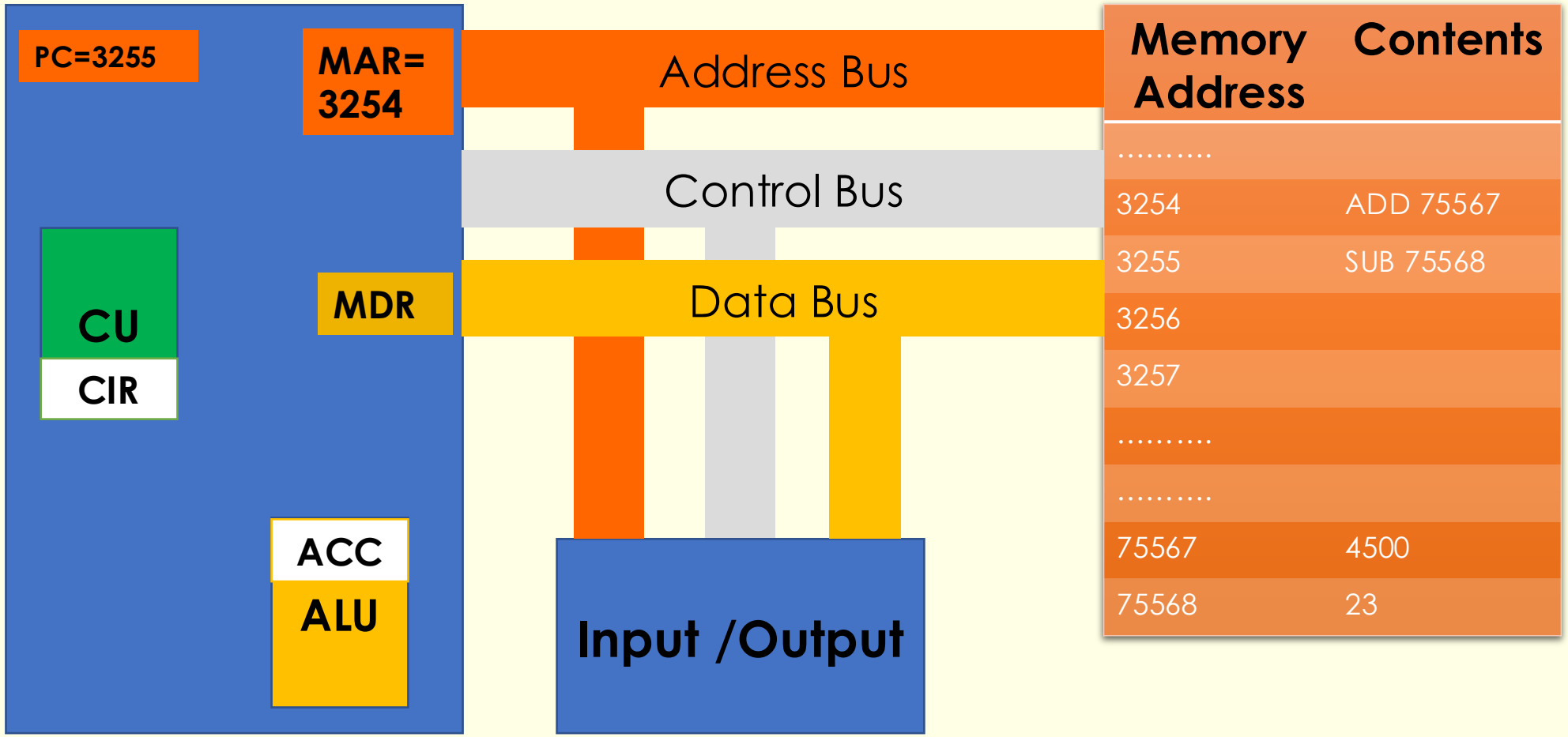
The CPU reads the contents of the Program Counter to find the address of the next instruction to be fetched, decoded and executed.

2. The contents from the PC are then **copied** to the MAR.



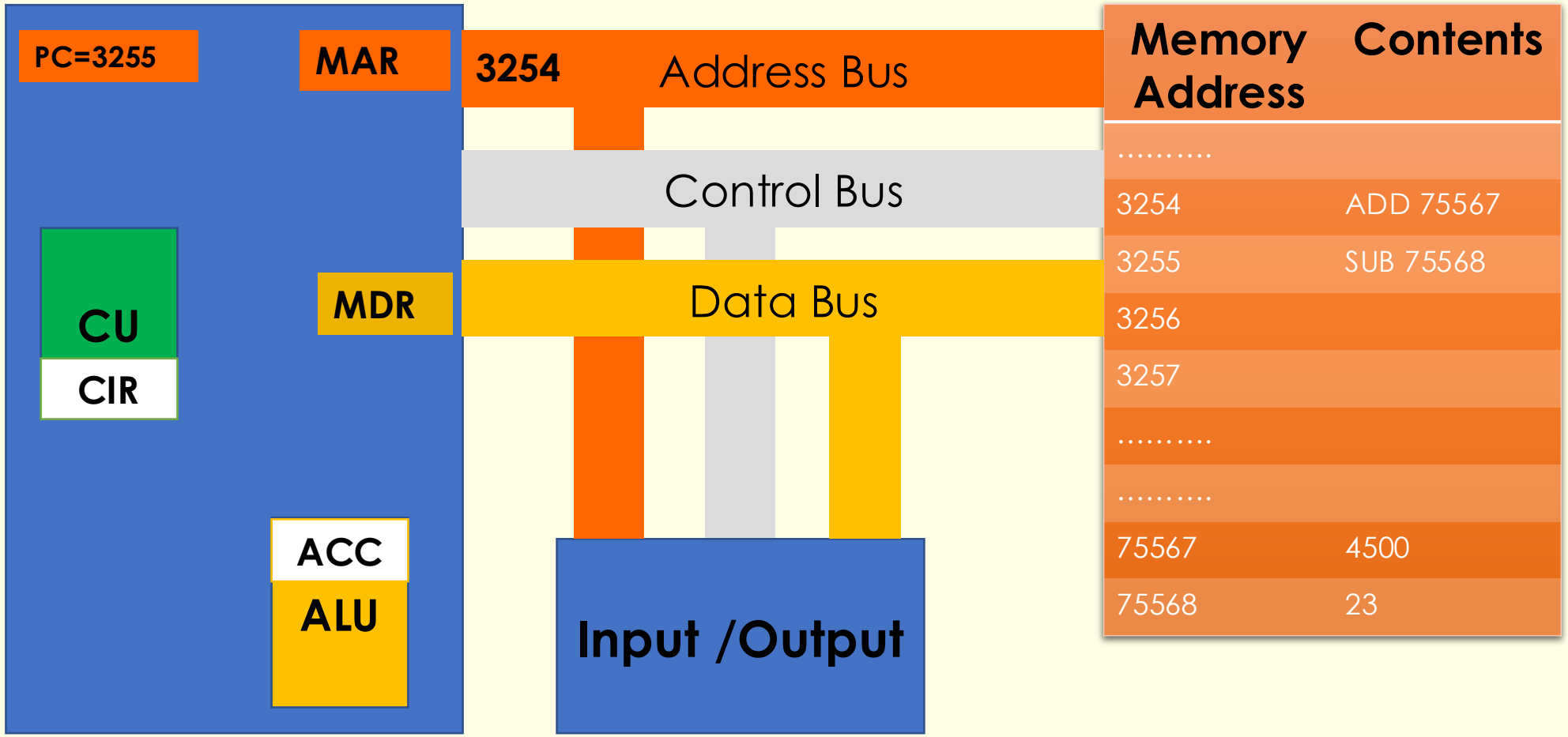
Steps in the fetch process

3. As soon as it is read, the PC increments. PC = PC + 1, or 3255



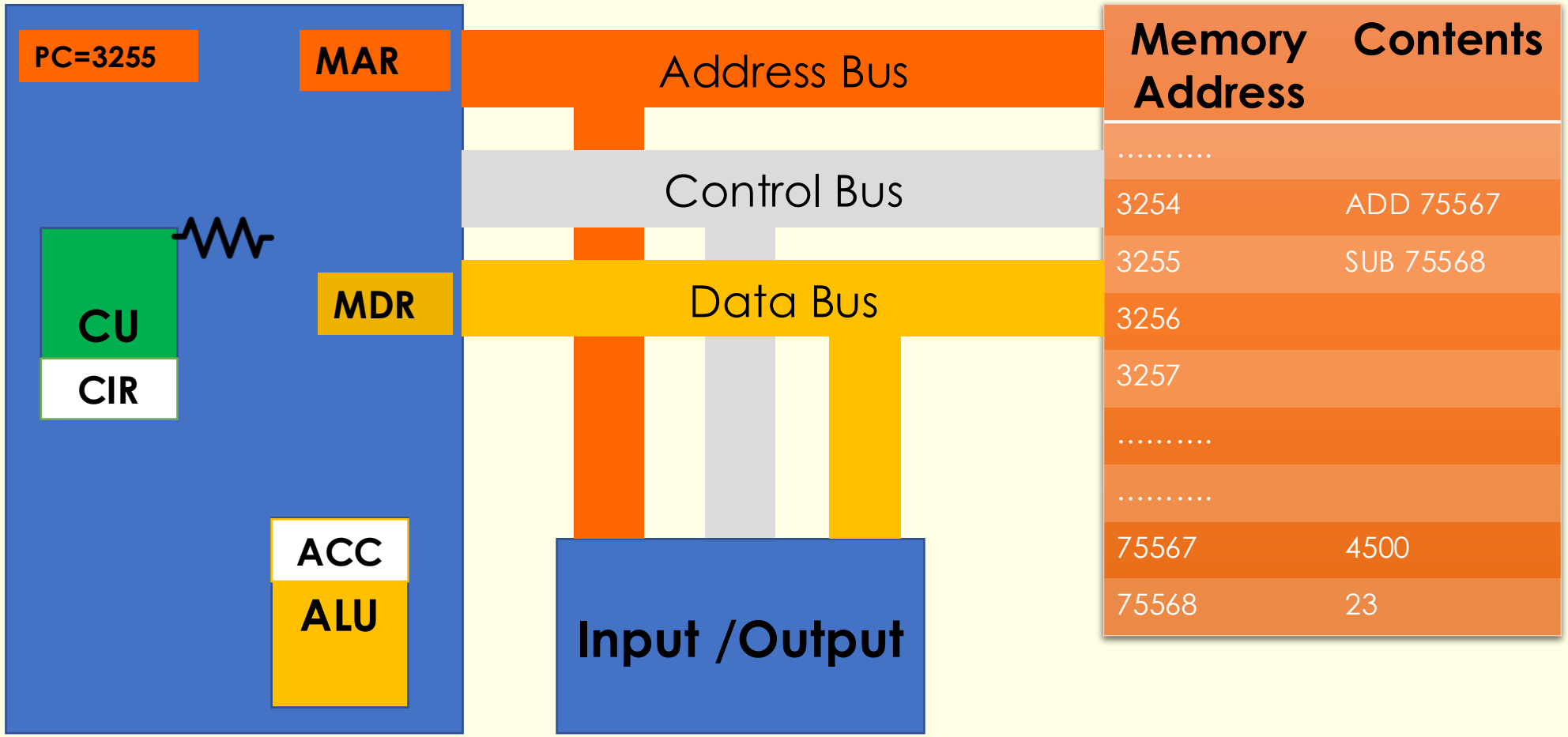
Steps in the fetch process

4. The **address** in the MAR is **sent/carries** across the **address bus** to locate the contents in RAM



Steps in the fetch process

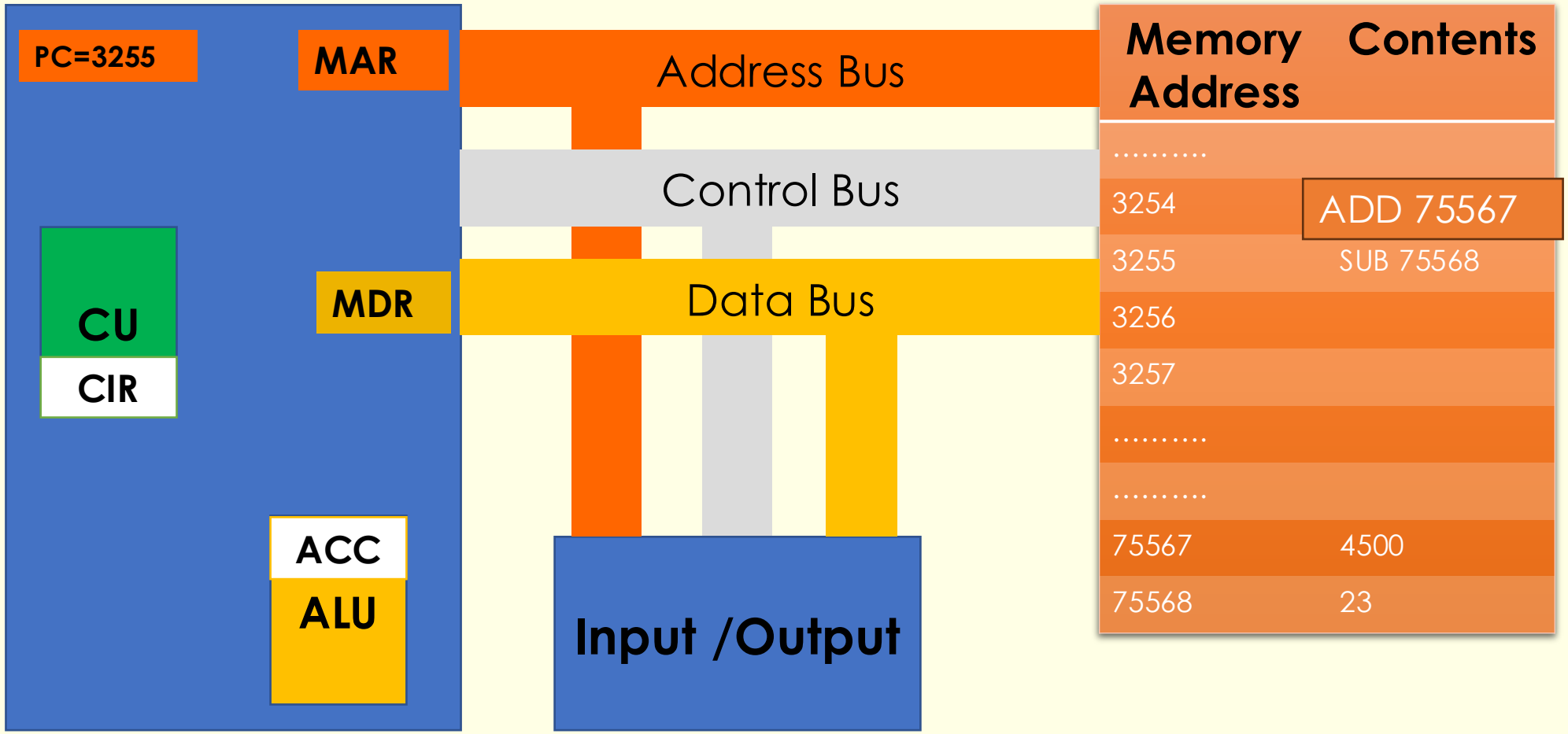
5. The control unit sends a **read signal** across the control bus to **read** from the memory address



Steps in the fetch process

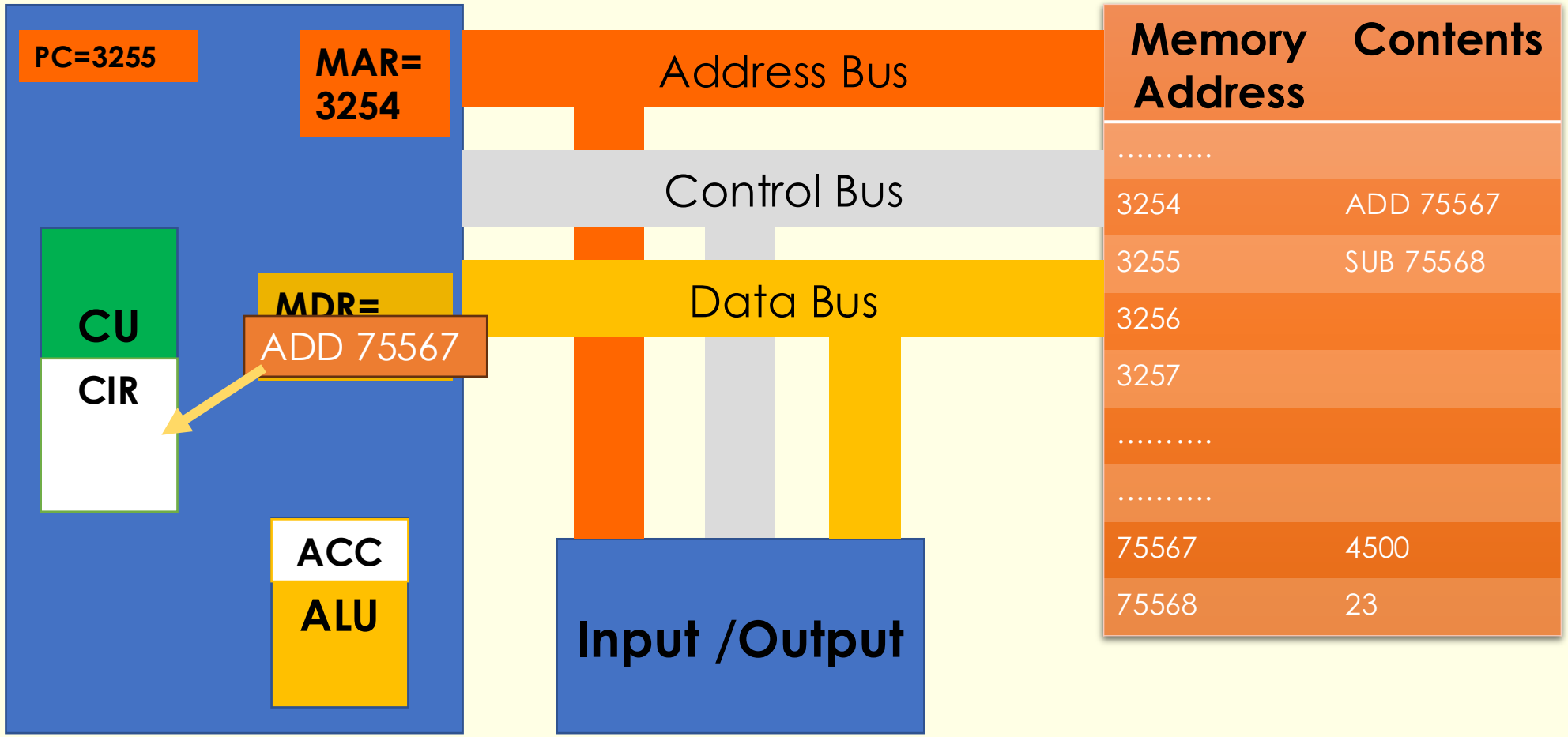
6. The **contents/ instructions and data** of this address are **sent/ carried** across the data bus to the MDR

7. The MDR now holds the instruction that must be executed.



Steps in the fetch process

8. The instruction in the MDR is then copied to the CIR, as we will often need to use the MDR again to complete the execution of an instruction.



Summary

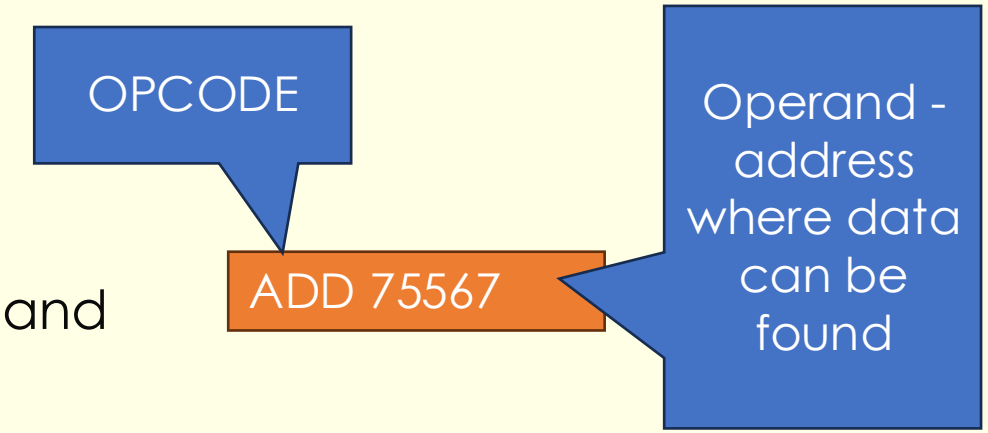
Steps in the fetch process

1. The CPU reads the contents of the Program Counter to find the address of the next instruction to be fetched, decoded and executed.
2. As soon as it is read, the PC increments.
3. The contents from the PC are then copied into the MAR.
4. The address in the MAR is sent across the address bus to locate the contents in RAM
5. The control unit sends a read signal across the control bus to read from the memory address
6. The contents of this address are copied to the MDR via the data bus
7. The MDR now holds the instruction that must be executed.
8. The instruction in the MDR is then copied to the CIR, as we will often need to use the MDR again to complete the execution of an instruction.

Key words:
Contents
Copied
SEND/carries /TRANSFERRED

DECODE Stage

- The control unit decodes the instruction
- The contents of the CIR are divided.
- Part of the instruction might be an operation (like ADD) and part of the instruction might be data,
- The ADD part is known as the **OPCODE** and the data part is known as the **OPERAND**.
- The operator (ADD) is decoded by the Control Unit in the CPU, so it knows what it has to do (ADD in our case).
- The operand 75567 is put back on the MAR.
- The contents of 75567 is then found in RAM and put on the MDR.



Control bus

The control bus is a **bi-directional bus**, meaning that signals can be carried in both directions.

The control bus is used to send control signals that manage and coordinate the activities of the system's components.

Control signals include:

- **Memory Write:** causes data on the data bus to be written into the addressed location
- **Memory Read:** causes data from the addressed location to be placed on the data bus
- **Interrupt request:** indicates that a device is requesting access to the CPU
- **Clock:** used to synchronise operations

EXECUTE

The instruction can now be executed.

This could be:

- Load data from memory
- Write data to memory
- Do a calculation or logic operation using the ALU
- Change the address in the PC
- Halt the program

For instance:

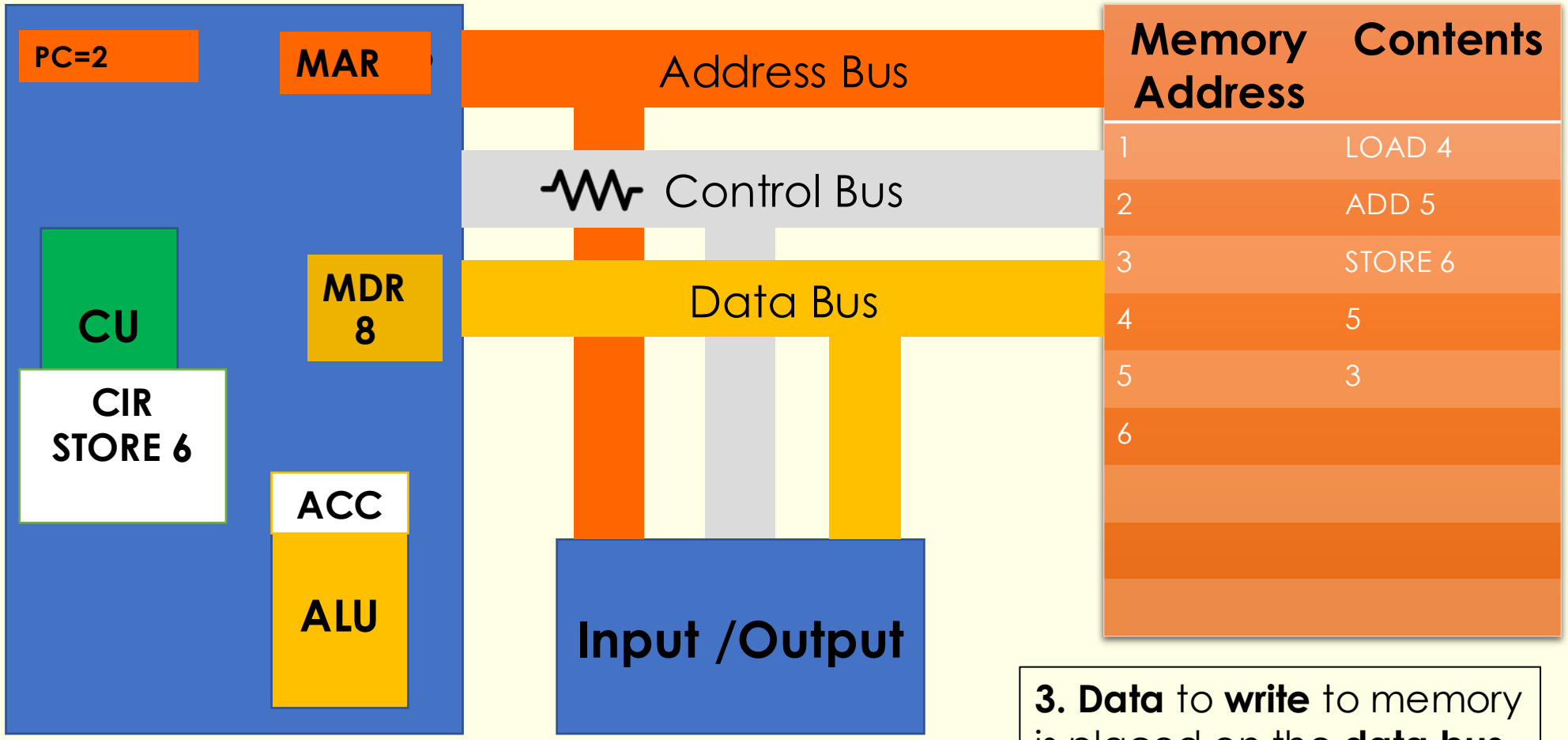
- if the instruction is for a memory location to be read from or written to (that is, LDA or STA), then the address stored within the instruction will be loaded into the MAR.
- In the case of STA, the data stored in the ACC is sent to memory.
- In the case of LDA, the data is loaded from memory into the ACC.
- If the instruction is to carry out a calculation (that is, ADD or SUB) then the contents of the MDR and ACC are sent to the ALU and the result sent back to the ACC.

Executing **STORE 5** instruction

STORE 5 is an instruction used to write data to memory.

1. Control unit sends a signal on the **control bus** to start a **write** operation

2. Address of memory (to store data) is placed on the **address bus**



3. Data to **write** to memory is placed on the **data bus**

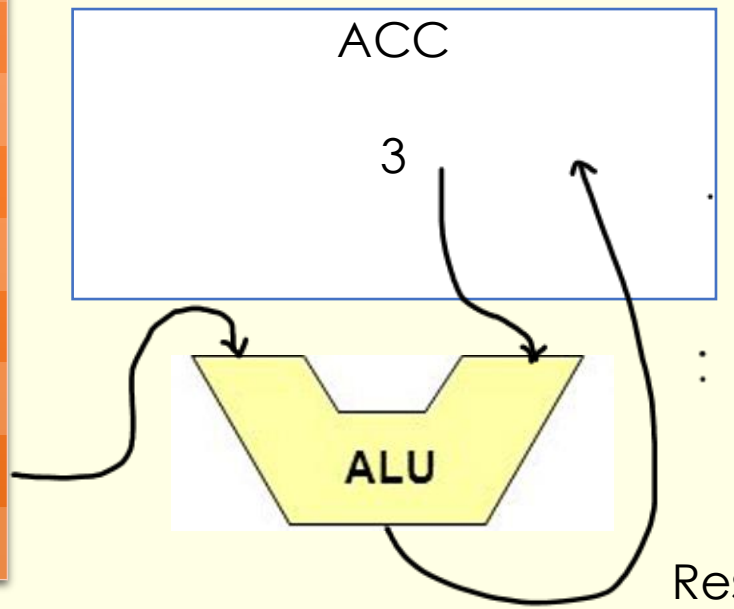
Example: ADD instruction

Arithmetic and logical instructions are carried out using the Accumulator(s) in a CPU.

Signals are sent out to different parts of the CPU to execute the instruction ADD.

This will result in adding 4500 to whatever is in the Accumulator, and then over-writing the contents of the Accumulator with the result of the addition.

Memory Address	Contents
.....	
3254	ADD 75567
3255	SUB 75568
3256	
3257	
.....	
.....	
75567	4500
75568	23



1. Assume a value of 3 has already been loaded into the accumulator
2. Now **ADD 75567** is read, this will get the value 4500 from memory.
3. The value 4500 is loaded into the ALU and the value 3 is loaded into the ALU from the ACC
4. ALU performs the ADD (using an adder circuit)
5. Writes the result back into the ACC
6. Then it can be output or stored in memory for later use.

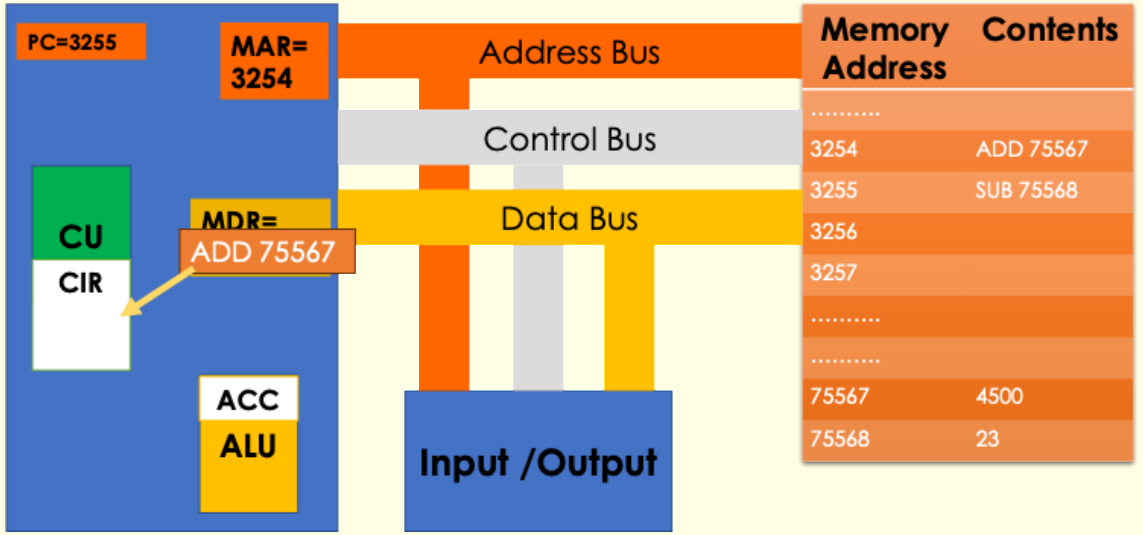
FDE cycle - Long answer question

A processor uses a number of different registers. One of these is the **Current Instruction Register.**

Describe the use of special **registers** and their functions during the **fetch-decode-execute cycle**, including **jump** instructions, **reading** from and **writing** to memory.

The quality of written communication will be assessed in your answer to this question. [8]

jump instructions – required for **loops, conditionals, subroutine calls, and branching**.



Explorer task

1. Find out about the jump instruction and the role of the CIR
2. Find out about the interaction between the CIR and MAR when performing Memory Access Instructions (Load/Store).

Jump instruction

A **jump instruction** alters the sequence of execution by moving to a new memory location instead of continuing with the next sequential instruction.

When a jump instruction is encountered in the CIR, it sends the address of the next instruction to be executed via the MAR and Address Bus

Memory Access Instructions (Load/Store):

If the instruction in the CIR involves memory access (like loading or storing data), the CIR plays a similar role in influencing the address sent to the MAR:

- For a load instruction (e.g., LOAD 0x2000), the instruction in the CIR specifies the memory address (0x2000) where data needs to be fetched from.
- The CPU decodes the instruction and sends the address (0x2000) to the MAR.
- The MAR accesses memory at address 0x2000 and retrieves the data, which is then loaded into a data register (like the Accumulator).

Similarly, for a store instruction, the address where data needs to be written is sent to the MAR after being decoded from the CIR.

Summary of CIR and MAR Interaction:

- The CIR holds the current instruction and helps the CPU determine what operation to perform.

- For jump instructions or memory-related instructions, the decoded instruction in the CIR influences the address that is sent to the MAR.

- In the case of a **jump**, the address is first updated in the **Program Counter (PC)**, and the PC sends it to the MAR to fetch the next instruction from the new address.

- In memory access instructions (like load/store), the address to access is derived from the instruction in the CIR and sent to the MAR directly for memory access.

Learning Aims

- Describe the factors affecting the performance of the CPU:
  - clock speed
  - number of cores
  - cache
- Understand the use of **pipelining** in a processor to improve **efficiency**
- Understand how address and data bus size relates to assembly language programs

Key words

Key term	Definition
Clock	The CPU's internal timer that sends out regular pulses to synchronise operations. Its speed (clock speed, measured in GHz) determines how many instructions can be processed per second.
Cache	A very fast type of memory inside or close to the CPU that stores frequently used instructions and data, reducing the time needed to fetch them from main memory.
Cores	Independent processing units inside a CPU. Each core can fetch, decode, and execute its own instructions, allowing true parallel processing.
Pipelining	A technique where the CPU overlaps the steps of different instructions (fetch, decode, execute), so several instructions are in progress at once.
Increases throughput	means that the processor is able to execute more instructions per second.
Idle components	Pipelining means all major CPU components (e.g., fetch unit, decode unit, ALU) are working at the same time on different instructions. It reduces wasted time and prevents components from being idle.
Word size	The number of bits the CPU can process or transfer at once (e.g., 32-bit or 64-bit). A larger word size means more data can be handled in a single operation.
Buffer	a temporary storage area used to hold data while it is being transferred from the CPU to other components.

Factors affecting processor performance

The main factors affecting processor performance are:

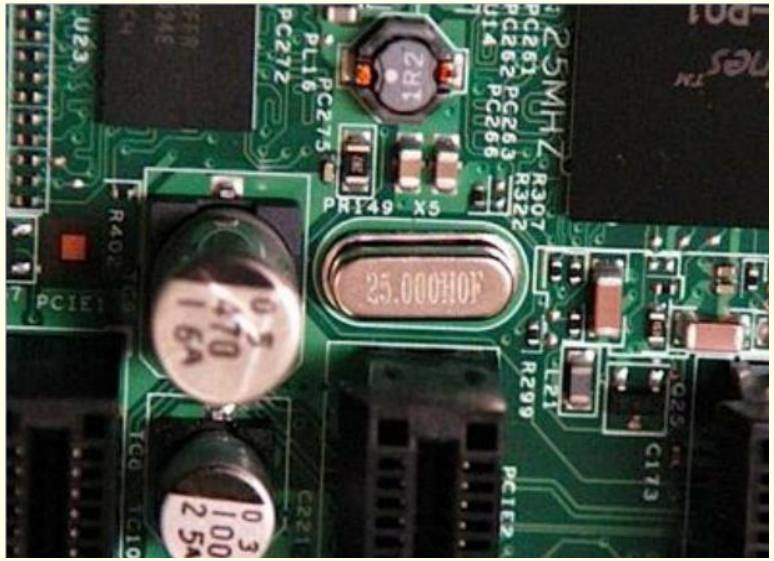
- **Clock speed**
- The number of **cores**, or duplicate processors, linked together on a single chip
- The amount and type of **cache memory**

System Clock

- Using a quartz crystal, the clock in a computer breathes life into the microprocessor by feeding it a constant flow of pulses.

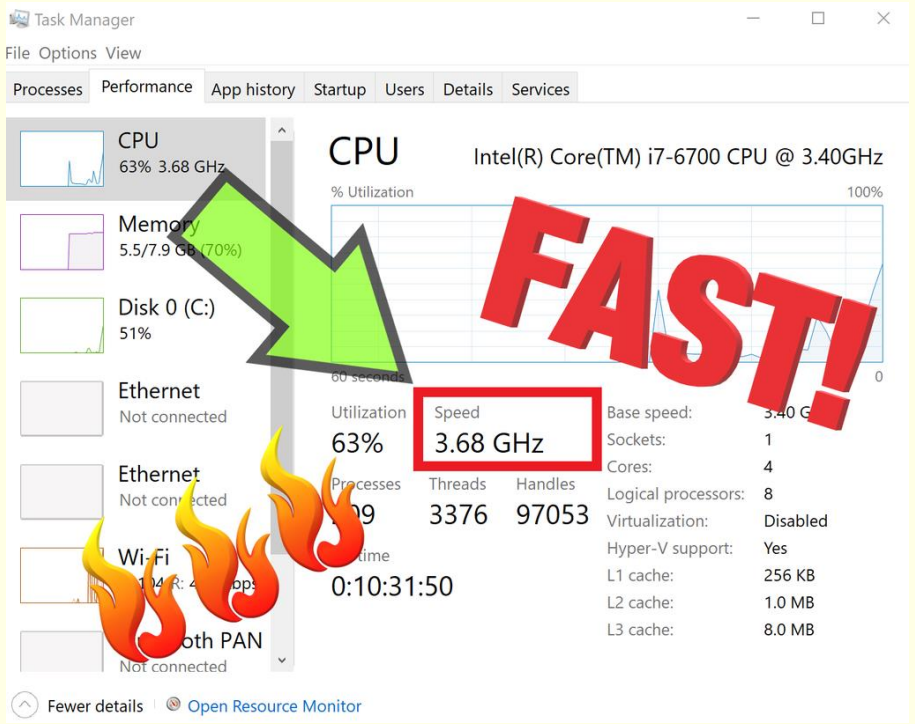
- The clock rate of a CPU is normally determined by the frequency of an oscillator crystal.

- In other words, the **frequency of electronic signals per second**



System Clock

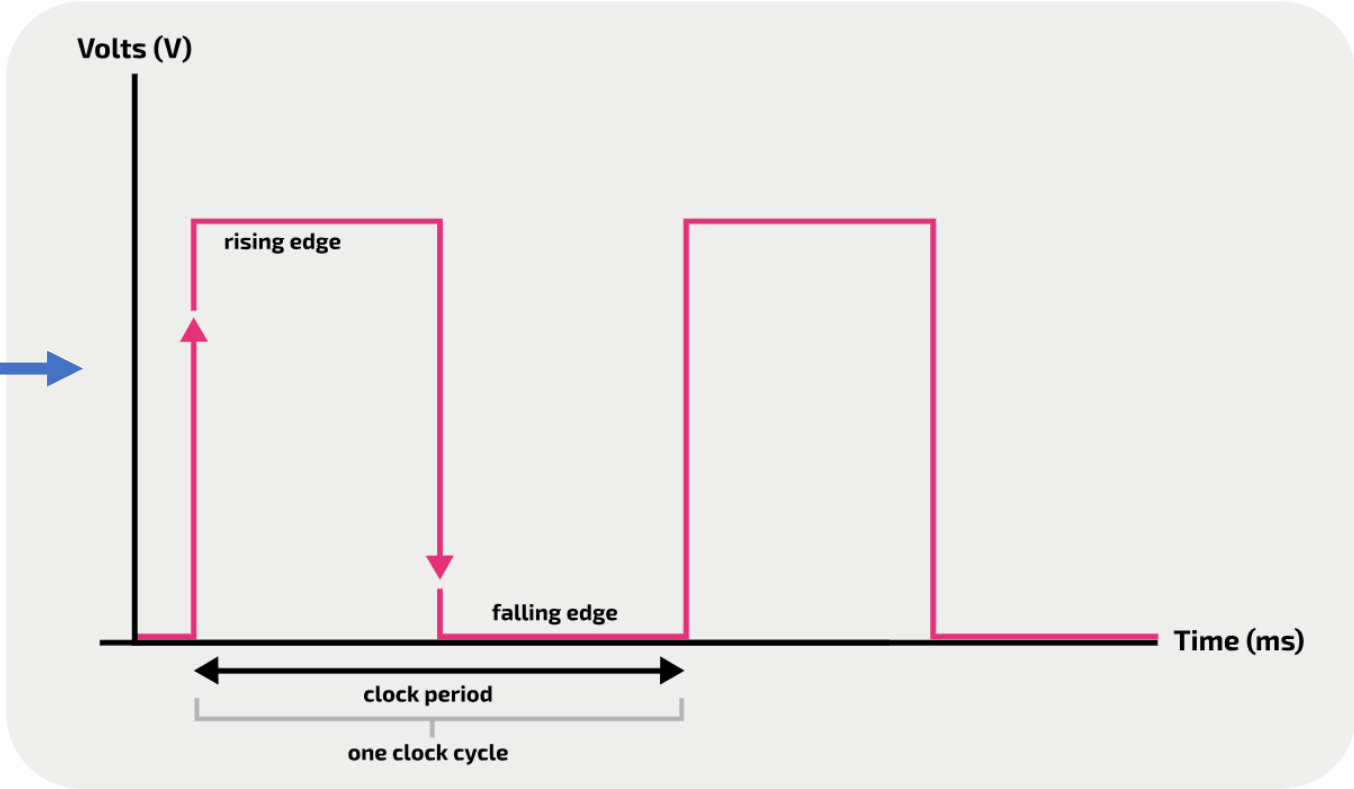
- A clock determines the speed of the CPU.
- Regular electrical pulse which synchronises all the components.
- **The faster the clock the more instructions can be executed per second.**
- The speed of the clock is measured in **Hertz** (Hz), which is the **amount of cycles per second**.
- **A clock speed of 500Hz means 500 cycles per second.**
- Current computers have clock speeds of **3GHz**, which means **3-billion cycles per second.**
- Each 'tick' means that one part of the fetch-decode-execute cycle can be carried out.



System Clock

- The **system clock** — also simply referred to as the **clock** — generates regular clock pulses by emitting a signal that continuously switches between a low (or '0') and a high (or '1') state.

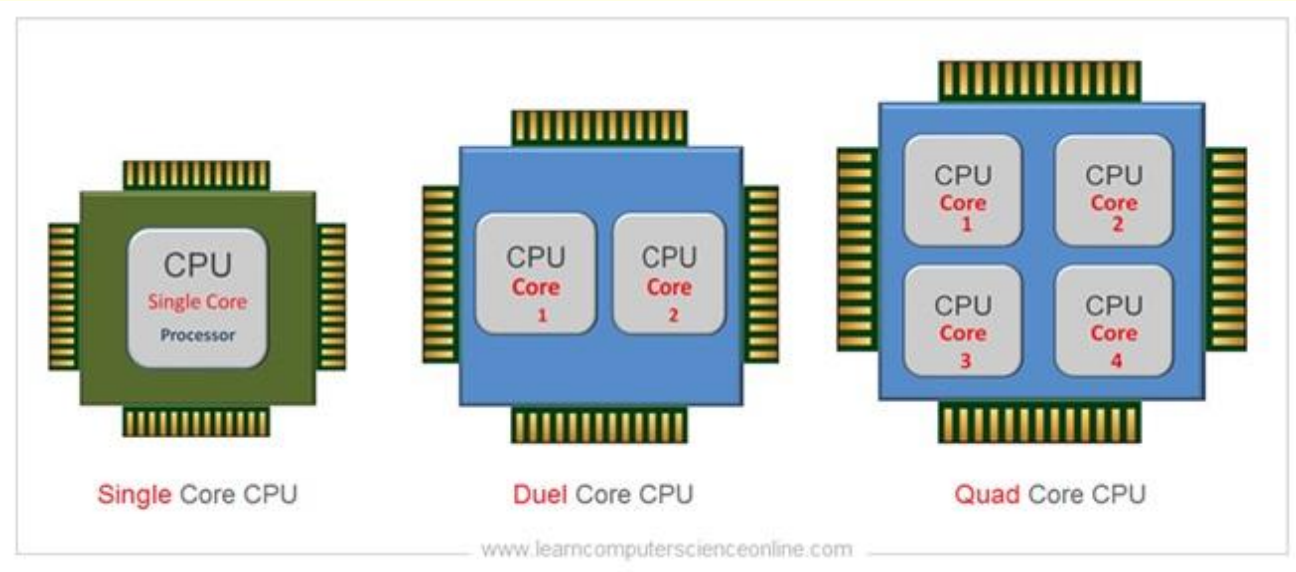
On the rising edge a FDE task is carried out



The time taken between two sequential rising edges is called a **clock cycle** or a **clock period**. The **clock speed** is measured by the number of clock cycles in one second — 1 clock cycle per second is 1Hz.

Cores

- A CPU traditionally had one 'core' but processors these days might be dual-core or quad-core
- For example, a core is actually a processor with its own cache.
- So a dual-core CPU has not one but two processors.
- A quad core CPU has four processors.
- Each core is theoretically able to process a different instruction at the same time with its own fetch-execute cycle, making the processor two or even four times faster with a quad-core chip.
- However, although a dual-core processor has twice the power, it does not always perform twice as fast, because the **software may not always be able to take full advantage of both processors.**

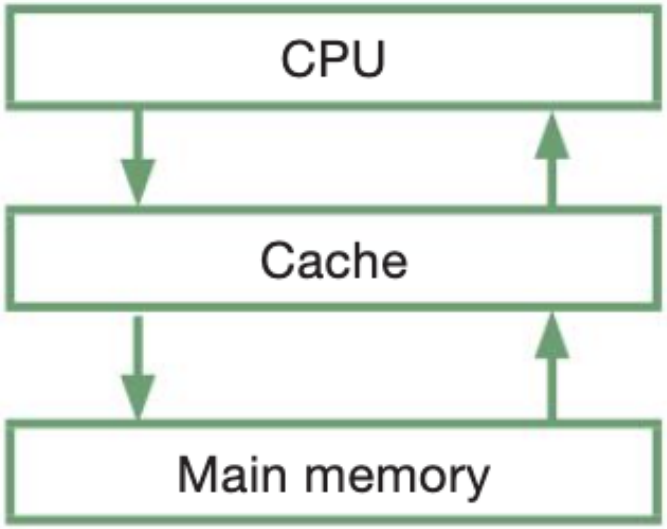


Amount and type of cache memory

Cache is a small amount of **expensive, very fast memory inside the CPU**.

When an instruction is fetched from main memory it is copied into the cache so if it is needed again soon after, it can be fetched from cache, which is much quicker than going back to main memory.

As cache fills up, unused instructions or data still being held are replaced with more recent ones.



There are different "levels" of cache:

- Level 1 cache is extremely fast but small (between 2-64KB)
- Level 2 cache is fairly fast and medium-sized (256KB-2MB)
- Some CPUs also have Level 3 cache

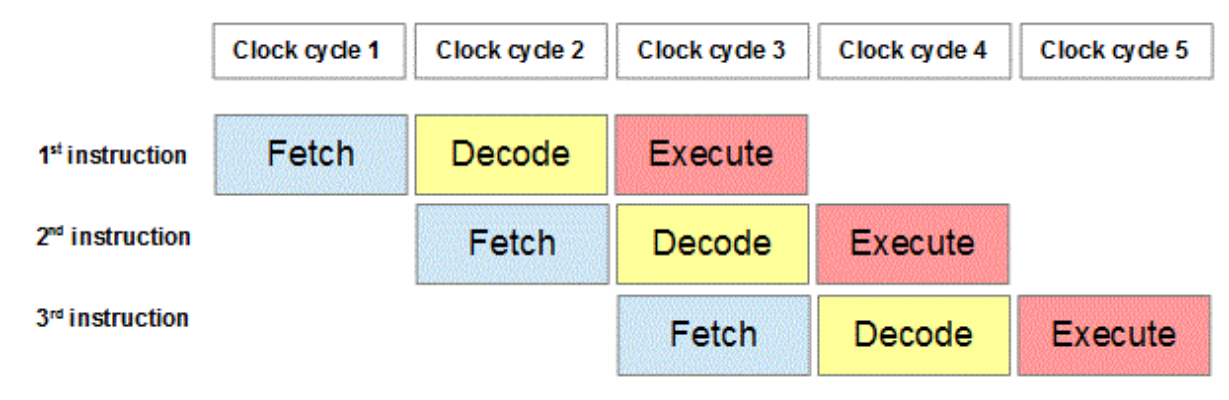
Pipelining

Pipelining is a technique used by some processors to improve performance.

Without pipelining, the steps in the Fetch-Execute cycle take place one after the other.

- Pipelining allows the next instruction to be fetched **whilst** the previous one is being decoded/executed
- It allows the overlapping of different parts of the FDE
- It increases **throughput** // increases the number of instructions processed in a set period of time
- It prevents the CPU having to wait and prevents **idle components**

Pipelining is now common in microprocessors used in personal computers. Intel's Pentium chip uses pipelining to execute as many as six instructions simultaneously.



Words and word size

Address bus word size

- Each word, or group of bytes, in memory has its own specific address.
- When the processor wishes to read a word of data from memory, it first puts the address of the desired word on the address bus.
- The width of the address bus determines the **maximum possible memory capacity of the system.**
- For example, if the address bus consisted of only **8 lines**, then the maximum address it could transmit would be (in binary) 11111111 or 255, giving a maximum memory capacity of **256** (including address 0).
- A system with a 32-bit address bus can address 2^{32} (4,294,967,296) memory locations giving an addressable memory space of 4GiB.

Data bus word size

- The data bus transmits the data held in a word of memory, between processor components and memory.
- The largest operand (which is either an address or an actual value) that can be held in a word is therefore related to the size of the data bus.
- If the data bus is 16 bits wide, a word cannot hold an integer greater than 2^{16} at a time, or allow more bits per instruction.

How this relates to assembly language

The basic structure of a machine code instruction in a computer with a 16-bit word may take the format shown below:

Operation code								Operand(s)							
Basic machine operation						Addressing mode									
0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	1

In assembly language, the operation code (opcode) will be expressed as a mnemonic such as ADD, SUB, LDA (load into the accumulator) etc.

With only six bits for the opcode, there cannot be more than 2^6 Operand(s) -1, or more than two characters.

A wider data bus can transmit larger values, or more characters different instructions.

The operand has to be held in only 8 bits.

This would clearly not be practical in a **general purpose computer** which is more likely to have a word size of **32, 64 or 128 bits.**

Summary

Clock Speed:	The frequency at which a CPU executes instructions, measured in cycles per second (Hertz). Higher clock speeds generally result in better performance, but other factors also play a significant role.
Instructions per Cycle (IPC):	The number of instructions a CPU can execute per clock cycle. Increasing IPC improves performance without requiring an increase in clock speed.
Cache Size and Efficiency:	The CPU cache stores frequently accessed data and instructions, reducing the need to fetch them from main memory. Larger and more efficient caches enhance performance.
Number of Cores:	Multi-core CPUs contain multiple processing units, enabling them to execute multiple tasks simultaneously. Software must be optimized to take advantage of multiple cores to see performance benefits.
Instruction Set Architecture (ISA):	The design of the CPU's instruction set affects its performance. Complex instruction set computers (CISC) have fewer instructions that perform more complex operations, while reduced instruction set computers (RISC) have a larger number of simpler instructions.
Pipelining:	Pipelining breaks down the execution of instructions into multiple stages, allowing different stages of different instructions to be executed concurrently. A typical pipeline includes stages such as instruction fetch, instruction decode, execute, memory access, and write back. Advantages: Pipelining increases CPU throughput by overlapping the execution of instructions. It can improve performance by reducing the time taken to complete a single instruction. Hazards: Data hazards, control hazards, and structural hazards can occur in pipelined architectures, potentially reducing performance. Techniques like forwarding and branch prediction are used to mitigate these hazards.

Learning aims

- Describe von Neumann, Harvard and contemporary processor architecture

Data and instructions
Memory
CPU architectures

Von Neumann
Stored program concept
Serially
bit pattern
Von Neumann bottleneck

Harvard
Special-purpose
Real-time

Cache - a small, fast type of memory in the CPU that stores frequently used data to speed up processing.

General Terms

Key term	Definition
Data and instructions	Data = the values processed (numbers, text, etc.). Instructions = the commands telling the CPU what to do. Both must be stored in memory.
Memory	Stores data and instructions. Can be RAM (temporary) or other storage.
CPU architecture	The design of how the CPU, memory, and buses are organised and interact.

Von Neumann Architecture

Key term	Definition
Von Neumann	A CPU design where data and instructions are stored in the same memory and travel along the same buses.
Stored program concept	Instructions for a program are stored in memory alongside the data, so the CPU can fetch them as needed.
Serially	Instructions are fetched and executed one after another in order.
Bit pattern	Binary digits (0s and 1s) used to represent instructions or data.
Von Neumann bottleneck	The limitation caused because data and instructions share the same bus, meaning only one can be transferred at a time, slowing performance.

Harvard Architecture

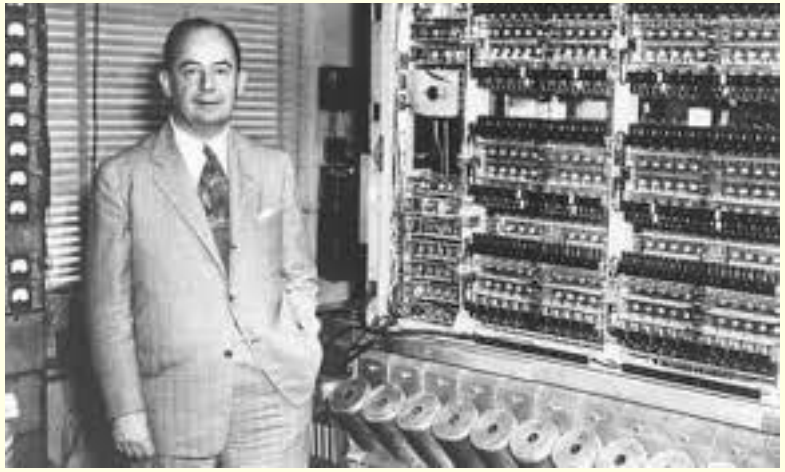
Key term	Definition
Harvard	A CPU design where data and instructions are stored in separate memory and use separate buses. This allows simultaneous access and can be faster.

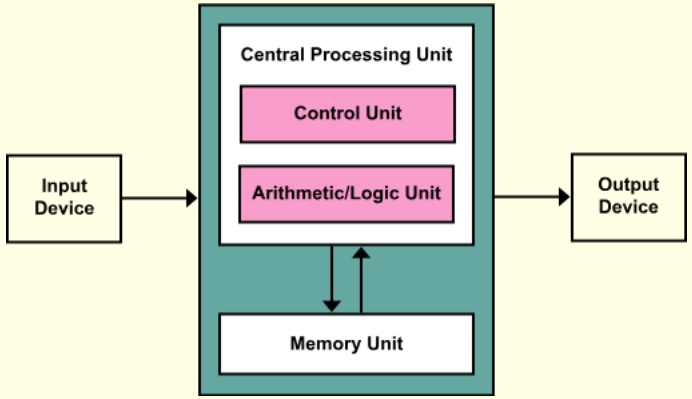
Special-purpose Systems

Key term	Definition
Special-purpose	A computer designed for a specific task (e.g., a washing machine controller, sat-nav).
Real-time	A system that responds immediately to inputs and events, often used in safety-critical systems (e.g., car airbags, medical equipment).

CPU Architectures - Memory

Von Neumann

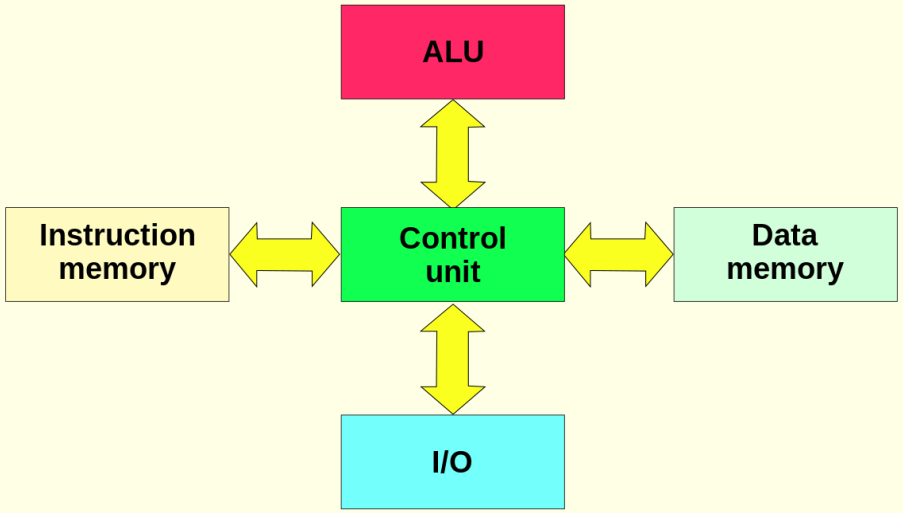




Stored program concept

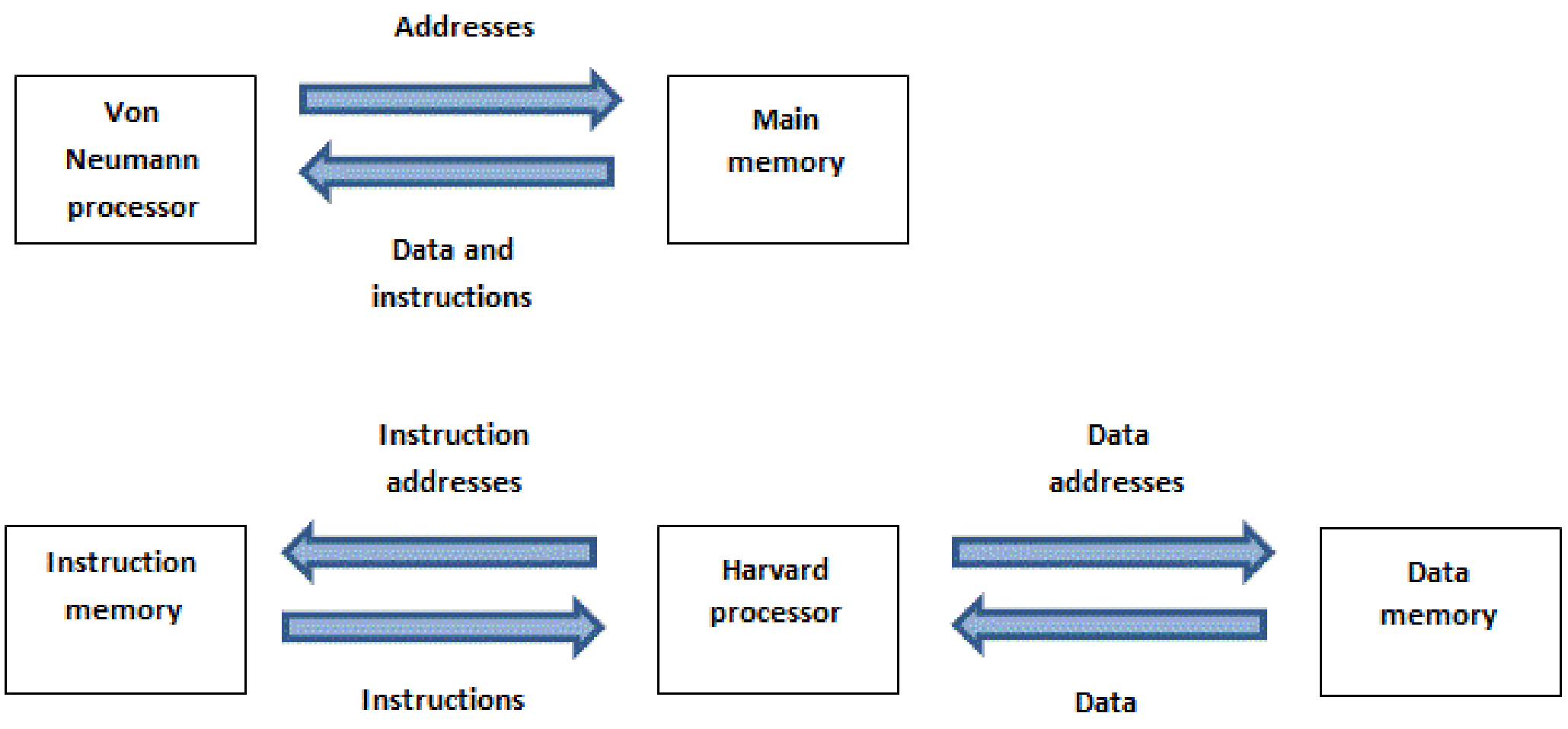
Instructions and data are stored in the same memory.

Harvard architectures



Physically **separate** memories for instructions and data

Comparison


Addresses
Von Neumann processor
Main memory
Data and instructions
Instruction addresses
Data addresses
Instruction memory
Harvard processor
Data memory
Instructions
Data

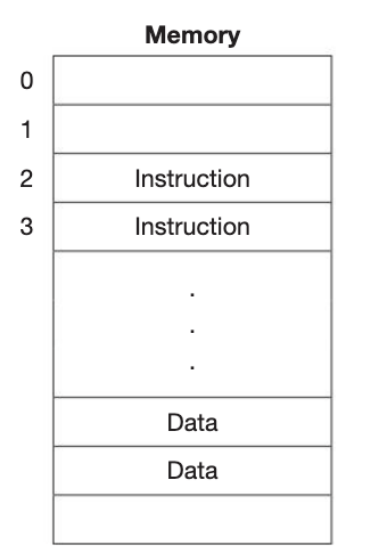
The Von Neumann machine

von Neumann

Same data bus is used to transfer both data and instructions.

Same word length is used for all memory, whether it holds data or instructions.

Can only fetch either data or instructions at one time/follows FDE



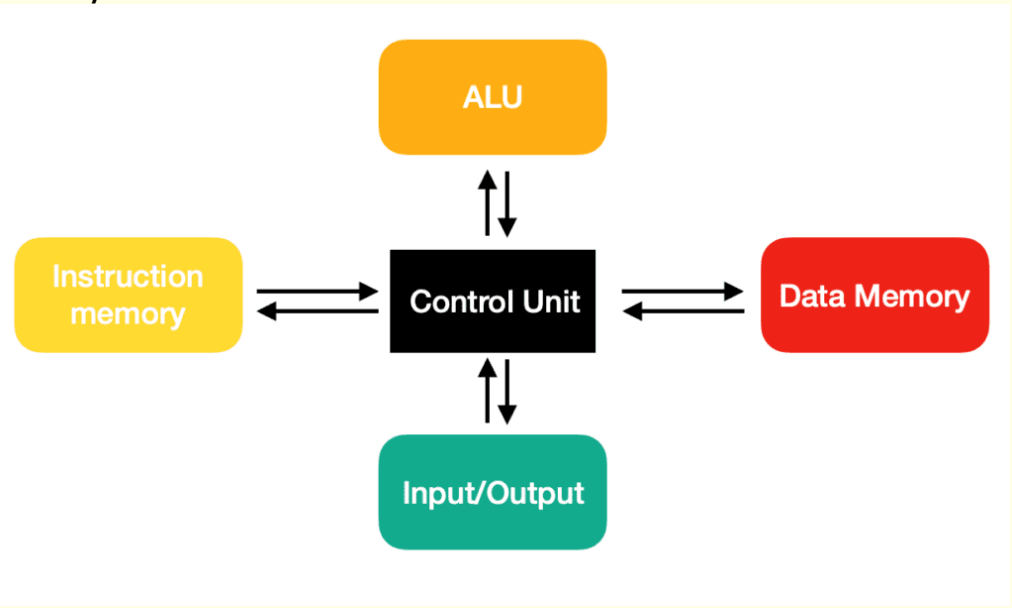
The stored program concept

Harvard

Different (sets of) buses. one for instructions & one for data

Can use **different word lengths** for data and instructions, (optimising memory use)

Instructions and data can be accessed concurrently/ parallel/at same time.



Von Neumann bottleneck

- Whatever you do to improve performance, you cannot get away from the fact that instructions can only be done one at a time and can only be carried out sequentially.
- Both of these factors hold back the efficiency of the CPU.
- This is commonly referred to as the '**Von Neumann bottleneck**'.
- The von Neumann bottleneck is a computing limitation that occurs when data transfer between the central processing unit (CPU) and memory is slow.
- This causes the CPU to wait for data to be accessed.

Used for …

Architecture	Typical Uses	Examples
Von Neumann Architecture	- Used in **general-purpose computers** where tasks vary (e.g. PCs, laptops, tablets). - Suitable for systems that **run many types of programs** stored in the same memory (instructions + data).	Desktop computers, laptops, smartphones, servers.
Harvard Architecture	- Used in **embedded systems** and **microcontrollers**, where the program and data are fixed or predictable. - Ideal for **digital signal processing (DSP)** or **real-time systems** needing fast, simultaneous access to data and instructions.	Washing machines, robots, digital cameras, automotive control units, DSP processors.

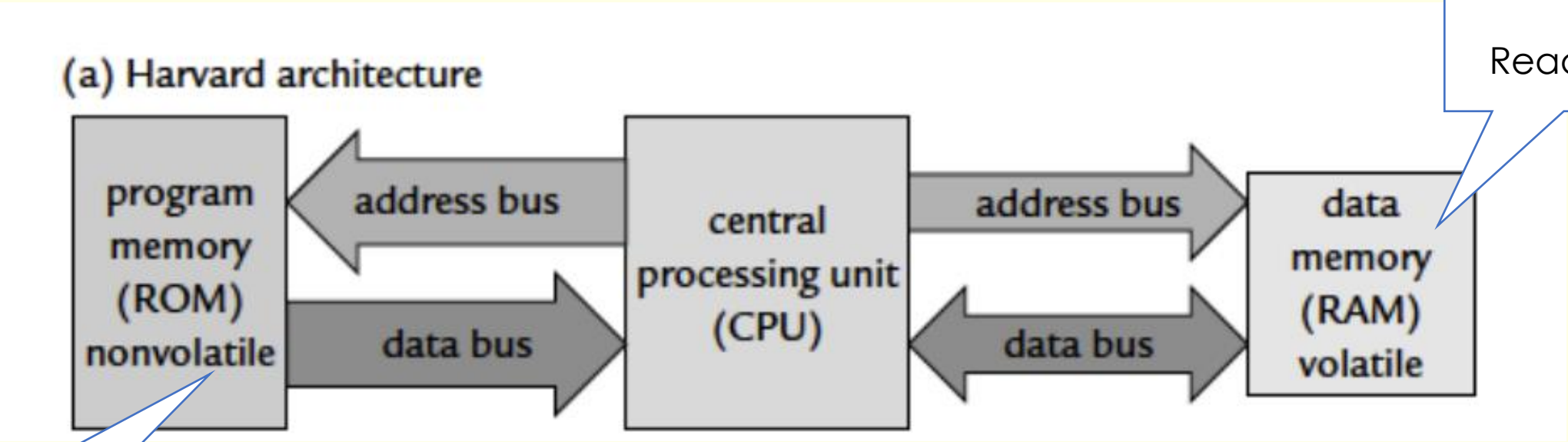
Harvard and Real-time Embedded systems

Embedded systems include **special-purpose computers** built into devices often operating in **real time** such as those used in:

- navigation systems
- traffic lights
- aircraft flight control systems and simulators

Harvard architecture can be **faster** than von Neumann architecture because **data and instructions** can be fetched in **parallel** instead of competing for the same bus.

Embedded Systems



RAM is used for data (Flash storage)

Read/write

ROM is used for instructions/ firmware

Read-only

Summary

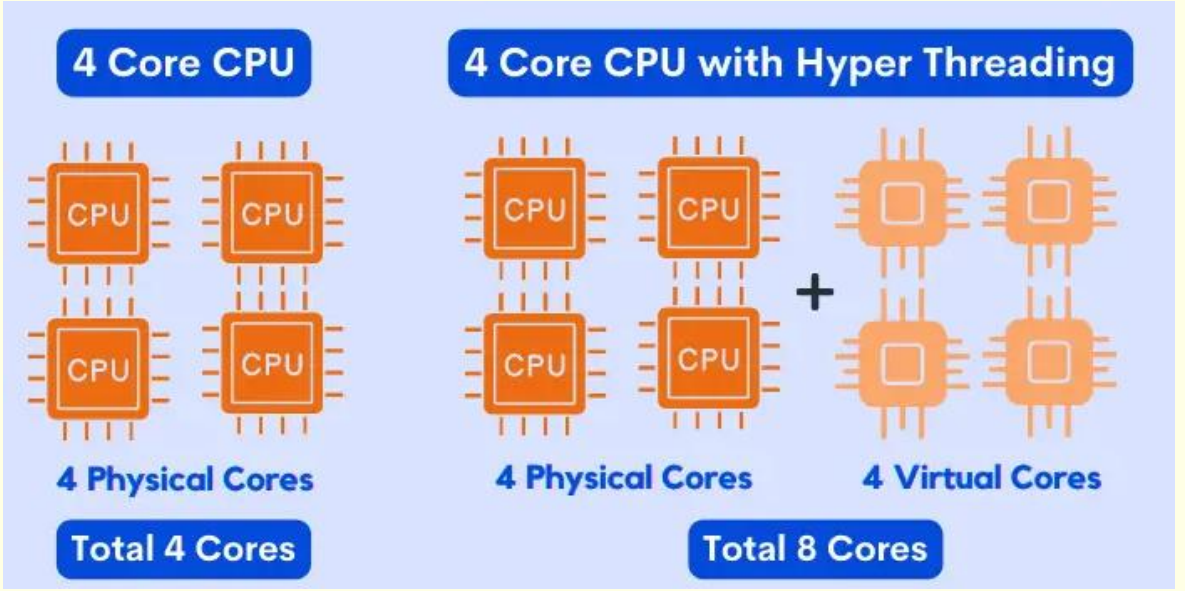
Von Neumann Architecture	Harvard architecture
• **Instructions and Data stored in same area of memory** • **Von Neumann fetches data and instructions sequentially/follows FDE cycle** • **Von Neumann uses a single bus for both data and instructions** • Single control bus • Slower due to the Von Neumann bottleneck (using the same bus) • Used in general-purpose computers (e.g. PCs, laptops). • Can be optimised by using pipelining • Same word length for both data and instructions	• **Harvard stores data and instructions in separate memory units** • **Harvard can fetch data and instructions at the same time (fetching the next instruction while reading/writing data.)** • Separate control buses • Harvard uses different buses for data and instructions • Faster because data and instructions are accessed in parallel. • Common in embedded systems and microcontrollers. Digital Signal Processing (DSP) systems that require fast access to data and instructions • Can use different word lengths for data and instructions, (optimising memory use)

Features of contemporary processors

- **Two separate areas of memory**…one for instructions & one for data./instructions and data can be accessed concurrently.
- **Different (sets of) buses**…one for instructions & one for data./instructions and data can be accessed concurrently.
- **Pipelining**…whilst an instruction is being executed the next can be decoded and the subsequent one fetched.
- **Use of cache**…A small amount of high performance memory is (next to the CPU) / which stores frequently used data/instructions
- **Virtual cores/Hyper-threading** …Treating a physical core as two virtual cores.
- **Multiple Cores**…Each core acts as a separate processing unit.
- **Onboard Graphics**…Built in circuitry for graphics processing.

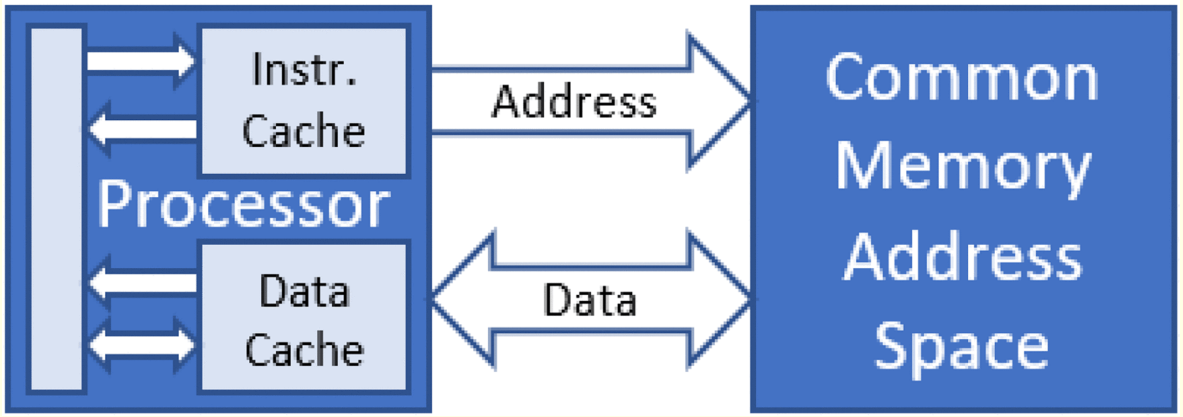
Virtual cores/Hyper-threading

- Hyperthreading refers to the technology invented by Intel, with which a physical microprocessor behaves like two logical, virtual cores.
- **Virtual cores** are created by hyper-threading.
- **Hyper-threading** allows each physical core to handle two tasks (threads) at once, creating virtual cores.
- This helps the CPU handle more tasks at the same time, improving performance, especially in multi-threaded applications.
- For example, a CPU with 4 physical cores and hyper-threading can appear to the operating system as 8 virtual (or logical) cores.



Contemporary processor architectures

- Modern high-performance CPU chips incorporate aspects of both von Neumann and Harvard architecture.
- In one design, there is one main memory for holding both data and instructions, but CPU cache memory is divided into an instruction cache and a data cache.
- Harvard architecture is used as the CPU accesses the **cache**.



- Some digital signal processors such as Texas Instruments TMS320 C55x have multiple parallel data buses (two write, three read) and one instruction bus.

The Intel Core i9-14900KS is the world's fastest desktop processor, with a maximum turbo frequency of 6.2 gigahertz (GHz)

From mid-2023 onwards, we started to hear increasing reports from game devs and gamers that high-end Intel 13th and 14th Gen CPUs (primarily the Core i9 13900K and Core i9 14900K) were crashing in Unreal Engine games.

£666.99

Intel has recently announced that it has found the cause of the instability issues affecting its 13th and 14th generation processors:

Cause

The issue is caused by "elevated operating voltage". This is due to a microcode algorithm that sends incorrect voltage requests to the processor.

Learning aims

- Describe GPUs and their uses
- Describe multicore and parallel systems

Key Terms

Multi-core & parallel systems
Co-processor
Workload
Supercomputers
Graphics processing unit (GPU)
Concurrent
Processing Speed:
Data
Performance
Core

Throughput - how much data a system can process in a given time.

Latency is the delay before a computer starts processing a task. CPUs aim to have low latency to respond quickly to commands.

Key term	Definition
Multi-core & parallel systems	CPUs with more than one core that can work on tasks at the same time (parallel processing), improving performance for multitasking and large workloads.
Co-processor	An extra processor designed to carry out specific tasks alongside the main CPU (e.g., GPU for graphics, FPU for maths).
Workload	The total amount of processing tasks a computer system has to handle.
Supercomputers	Extremely powerful computers with thousands of processors, designed to perform massive calculations quickly (e.g., climate modelling, scientific simulations).
Graphics Processing Unit (GPU)	A specialised processor with many cores designed for handling graphics and parallel tasks like AI or scientific computing.
Concurrent	When multiple tasks are in progress at the same time (not necessarily finished at the same time).
Processing speed	How quickly a computer can complete instructions, often measured in GHz (clock speed) or in instructions per second.
Data	The raw values (numbers, text, images) that the computer processes.
Performance	How efficiently a computer system completes tasks, measured using factors like speed, throughput, and responsiveness.
Core	An individual processing unit inside a CPU. Each core can fetch, decode, and execute instructions independently.
Throughput	The amount of data a system can process in a given time (higher throughput = more work done).
Latency	The delay before a system starts processing a task. Low latency means faster response times.

Co-processor systems

- A **co-processor** is an **additional processor** used for **specialised tasks**.
- In early computers, a single **CPU** handled all operations.
- As demands grew, co-processors were developed to work **alongside the CPU** to **boost performance**.

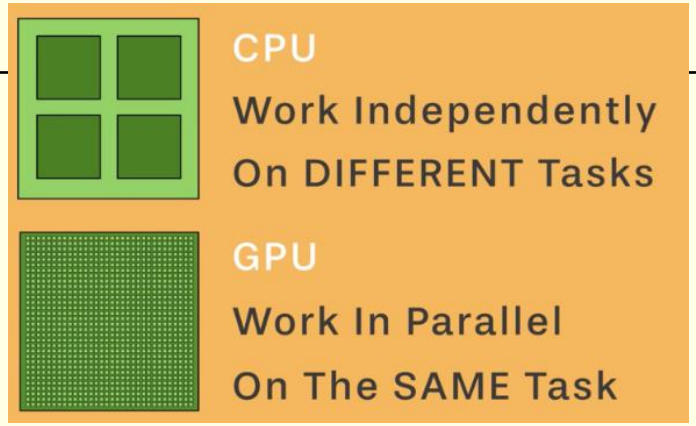
The Most Common Co-Processor:

GPU = Graphical Processing Unit.

- Originally used just for **rendering graphics**.
- Now used for **parallel data processing** across many cores.
- Tasks not limited to graphics anymore (e.g. machine learning, scientific simulations).

Differences Between CPUs and GPUs

CPU	GPU
• Fewer cores, higher clock speed • Great for complex, sequential tasks • General-purpose, flexible • Good for **small, complex tasks**.	• Thousands of slower cores • Great for simple repetitive tasks • Specialised for parallel data tasks • Good for **large, repetitive calculations** on **matrices, vectors, pixels**.

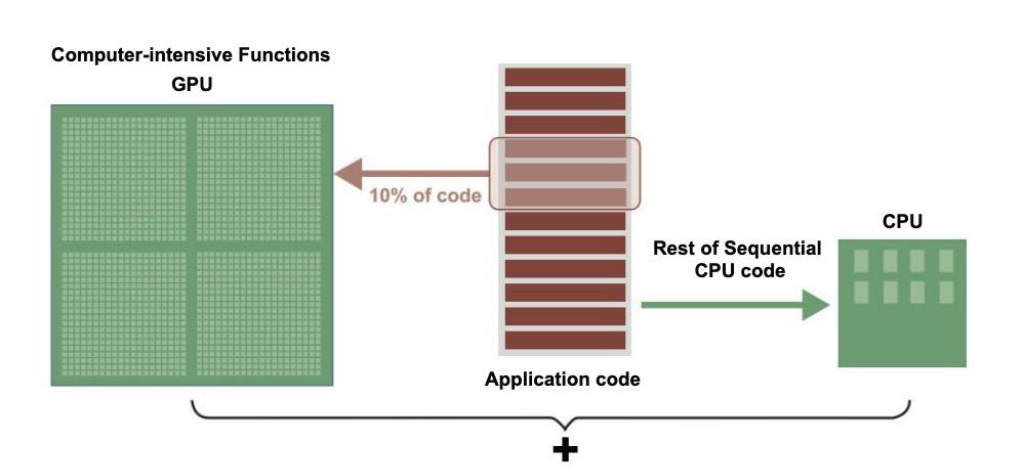


Why are GPUs Great for Graphics?

- Break graphics into:
  - **Vertices** with **XYZ coordinates**
  - **Textures**, **lighting**, and **camera angles**
  - All this requires **3D arithmetic, RGB color, and pixel-level** calculation.
  - GPUs handle this in **parallel**, speeding up rendering massively.

What is a GPU?

- In A Level Computer Science, a GPU is responsible for **processing graphics** within the computer to **reduce the load on the CPU**
- **CPUs are general purpose processors** whereas **GPUs are designed specifically for graphics**
- GPUs are likely to have built in circuitry or instructions for common graphics operations
- GPUs can perform an instruction on **multiple pieces of data at one time**
- This is useful when processing graphics (e.g. transforming points in a polygon or shading pixels) which means it can perform transformations to on screen graphics quicker than a CPU
- The GPU can either be **part of the graphics card or embedded in the CPU**
- Modern GPUs typically contain hundreds or even thousands of smaller processing cores, allowing them to perform many operations in **parallel**



What can a GPU be used for besides graphics

Besides graphics processing, a GPU can also be used for:

- **3D modelling**
  - The GPU can be used to **render** lighting effects, textures and shadows
- **Data modelling**
  - As GPUs can handle many calculations simultaneously, they can handle large datasets and complex operations like sorting and filtering data
- **Financial modelling**
  - GPUs are used to simulate different scenarios in risk modelling, option pricing and other financial modelling types
  - Lots of simulations can be run in parallel
- **Data Mining**
  - Data mining is the process of **analysing large amounts of data to find patterns**
  - The main computational tasks are sorting, searching, pattern recognition, statistical analysis and graph algorithms
- **Performing Complex Numerical Calculations**
  - Matrix multiplication and inversion can be done in parallel
  - Numerical Simulations - Physics and engineering simulations often involve solving complex maths models, which can be done in parallel
  - Solving Differential equations
  - Solving differential equations involves computations which can be performed in parallel
- **Machine learning**
  - This involves **training a computer on a massive amount of data** which can be done in parallel. There are lots of matrix multiplications and other computations which can be performed
  - After the training, GPUs can be used to speed up the process of **making predictions** on new data
- **Calculations on multiple data at the same time**
  - There are a number of scenarios where **calculations will be needed to be carried out on multiple data at the same time** e.g. insurance pricing, modelling risk, calculating bills
  - This is done by GPUs rather than CPUs due to being set up **for parallel processing**

What types of task are GPUs suited for?

- GPUs are suited to certain tasks that utilise:
  - **Specialist instructions**
    - GPUs are designed to execute specialist instructions which are common in 3D graphics rendering such as operations on matrices, vectors and geometric transformations
    - These capabilities have been expanded over time and have been generalised which makes GPUs suitable for a wide range of complex calculations besides graphics processing
  - **Multiple cores**
    - Although a CPU can have multiple cores, these are optimised for **serial** processing
    - GPUs have smaller cores but these are optimised for parallel processing
    - GPUs can perform many calculations simultaneously - ideal for tasks that can be broken down into smaller parts
    - This is useful in machine learning and situations where large amounts of data need to be processed
  - **SIMD processing**
    - Single Instruction Multiple Data (SIMD) processing is computers that have multiple processing elements which perform the same operation on multiple data points simultaneously
    - GPUs support SIMD processing as they were originally designed to perform the same operations on multiple pixels or vertices simultaneously - this is a common requirement in image processing, simulations and machine learning

Example

Convert a colour image to grayscale

1. **Split the image** - The GPU divides the image into thousands of small blocks or pixels so each one can be worked on separately.
2. **Send tasks to many cores** -Each GPU core (tiny processor) gets a small piece of the image — like one pixel or a group of pixels — to process.
3. **Process pixels in parallel** -All cores run the same instructions (like adjusting colour, brightness, or applying a filter) at the same time on different pixels.
4. **Combine the results** -Once all cores finish, the GPU combines all processed pixels back into a single, complete image.
5. **Display or save the image** -The final image is sent to the screen or stored in memory — ready for viewing or further processing.

What are the benefits of using a GPU?

- There are a number of benefits to using a GPU as well as a CPU (it isn't possible to only use a GPU as the CPU assigns tasks to the GPU)
  - **Parallel processing**
    - GPUs can handle many tasks simultaneously as they are multicore processors
  - **Speed**
    - As GPUs can use parallel processing, this speeds up tasks, particularly those involving large amounts of data or complex computations
  - **Efficiency**
    - GPUs can perform more calculations per unit of power consumed in comparison to CPUs making them more energy efficient when it comes to parallel tasks

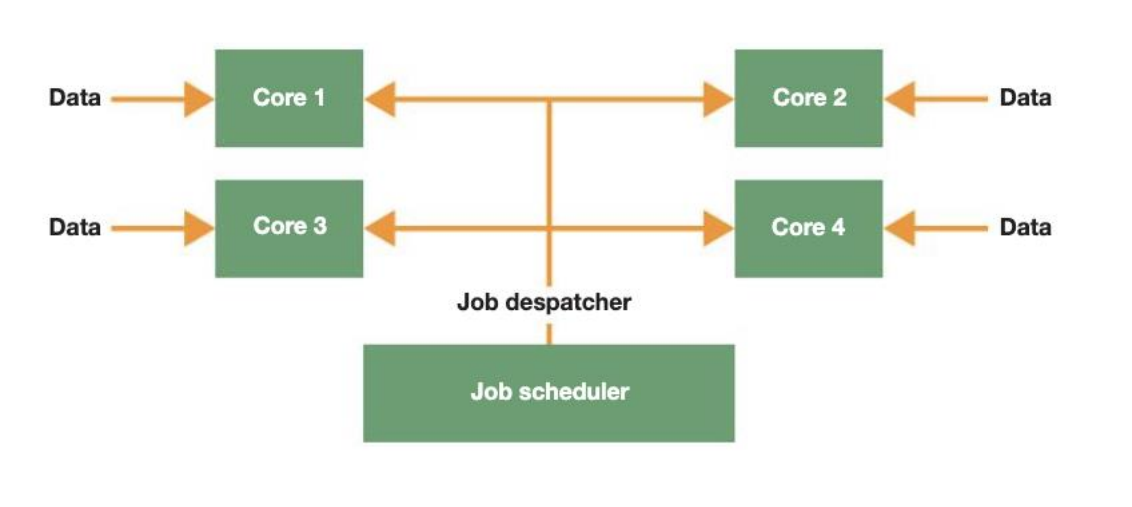
Multicore & Parallel Processors

What is parallel processing?

- In A Level Computer Science, parallel processing is when a computer has **multiple cores**
- Each core can work on the **same task**, to complete it **more quickly**, or each core can work on **separate tasks** at the **same time**

What is multicore processing?

- A multicore system has **more than one processing unit** in a single processor which can **independently process instructions at the same time**
- Parallel processing can also be achieved by utilising more than one processor (a CPU and a GPU)



Benefits and Limitations of Multicore Processors

Benefits	Limitations
Speed: If a task can be divided into subtasks that can be executed simultaneously, the total execution time can be reduced	**Limit on maximum speed**: Even with an infinite number of processors, there is a limit to the maximum speed improvement that can be made using parallel processing if a part of the program can't be parallelised
Improved performance: Simultaneous computation can take place on different data subsets (this would be used in machine learning, data mining and scientific computing)	**Complex programming**: It is harder to write code for parallel processing than serial processing. Tasks have to be synchronised and data shared correctly
Better resource utilisation: Parallel processing allows for better use of computer resources as multi-core or multiple processors can be used more effectively	**Debugging difficulty**: It is more difficult to debug a parallel program than a serial program due to the precise timing of specific events
Problem solving: Problems which are large and complex (which lend themselves to parallel processing) can be solved more easily	**Communication between processors**: Communication between processors can take significant time and resources, potentially outweighing the benefits of using parallel processing
Real-Time applications: Real-time applications including graphics rendering are more feasible and will benefit significantly	**Limited applicability**: Not all tasks can be run in parallel and must be executed serially

What are the benefits of using multicore processors?

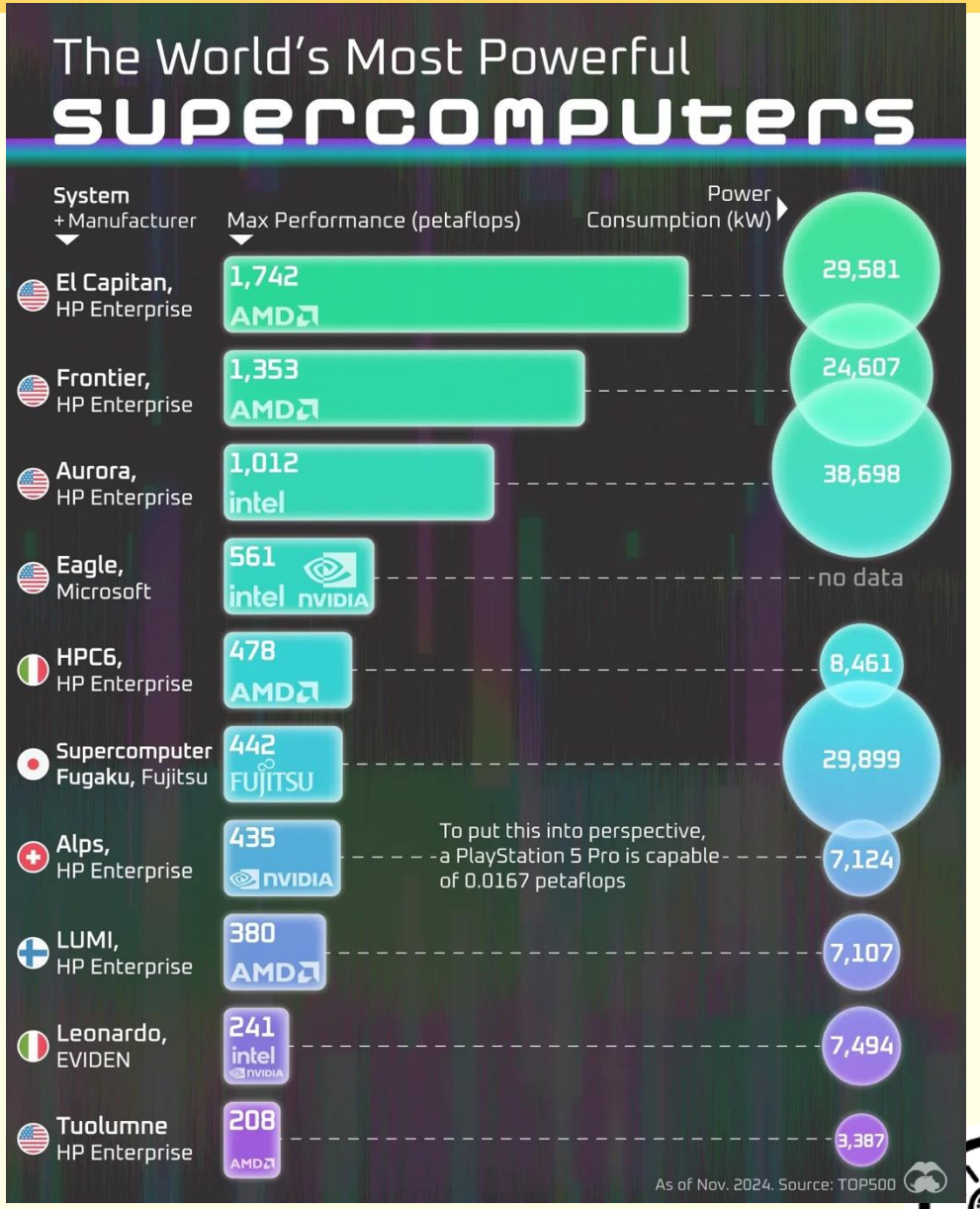
- **Multitasking**
  - Each core can work on a different task - this is particularly effective when the user has multiple applications open at the same time
- **Background tasks**
  - When using a single core processor, a background task like anti-malware scans can slow down the user's other task. A multi-core processor can assign the background task to one core, to reduce the impact on the other task
- **Improved responsiveness**
  - If a program becomes unresponsive, it won't slow the user's computer down as much if they're using multi-core as other cores will continue running their task

Supercomputers

Supercomputers are used on problems such as weather forecasting, running climate change models, processing Big Data or sequencing DNA.

The world's largest and most powerful supercomputer is El Capitan, located at the Lawrence Livermore National Laboratory in California and dedicated in January 2025.

It utilises AMD processors and has **11,039,616 cores**, making it the most powerful computer in the world



Case study

Case study: DeepMind AlphaGo program

Go is a Chinese game of far greater complexity than chess.

In March 2016 world champion Go player Lee Se-dol from South Korea was defeated by Google's DeepMind AlphaGo program. This was the first time a computer had been able to beat a human player at the game. DeepMind started by taking a huge database of professional Go matches and training a program to try to predict what move would come next in any given situation.

AlphaGo runs on Google's cloud computer network, using 1,920 processors and a further 280 GPUs.

A simpler version of the program that uses only 48 processors and 8 GPUs has been built to run on one machine.

Learning aims

- Describe the differences between, and uses of, CISC and RISC processors

Keyword	Simplified definition
RISC (Reduced Instruction Set Computer)	CPU design with a **small set of simple instructions**. Each instruction is designed to execute very quickly (often one clock cycle). Relies on efficient use of registers.
CISC (Complex Instruction Set Computer)	CPU design with a **large set of complex instructions**, some of which can execute multi-step operations. Each instruction may take multiple cycles but can reduce the number of instructions per program.
Machine code	The **binary code (0s and 1s)** that a CPU directly understands and executes.
Assembly language	A low-level programming language that uses **mnemonics** instead of binary machine code, e.g. ADD, MOV.
Mnemonics	Short, human-readable codes used in assembly to represent machine instructions, e.g. SUB = subtract.
Instruction Set Architecture (ISA)	The complete set of instructions that a CPU can execute. Different for RISC and CISC designs.
Opcode	The part of a machine code instruction that specifies the operation (e.g., ADD, LOAD).
Operand	The part of a machine code instruction that specifies the data or memory location to be used.
Instruction length	RISC instructions are usually fixed length (simpler to decode); CISC instructions vary in length.
Clock Cycles	RISC aims for 1 instruction per cycle, CISC often takes multiple cycles per instruction.
Pipelining	Easier to implement in RISC because instructions are simple and uniform in size.
Compiler optimisation	RISC relies heavily on the compiler to break complex tasks into simple instructions efficiently.

A history lesson …

The first ever programs were written solely in **machine code** (0s and 1s).

Each instruction was entered by hand before being **executed**.

Operation code (**opcode**) tables were used to help with this, but it was still very time consuming.

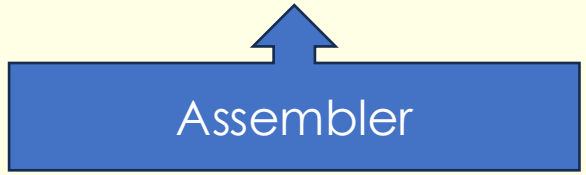
To speed things up, programmers developed an **assembly language**.

This made the **opcodes** easier to read by using a **mnemonic** and an **operand** to form an instruction.

Each line of **assembly language** was equivalent to one line of **machine code**.

Assemblers were created to automatically translate the assembly language into machine code.

```
10001011010101011111111100
10001011010001011111111000
111010000
10001001010001011111110100
```




```
LDA R1, A
LDA R2, B
MULT R1, R2
STO R1
A
```

Low-level languages

Assembly language and machine code are **low-level languages**.

Each line of assembly language is equivalent to one line of machine code.

The code is **specific** to the **CPU** that it is written for.

Assembly Language

LDA R1, A

Machine Code

```
10001011010101011111100
```

RISC

RISC, like what we have seen in Little man computer is a way to program a RISC architecture.

Using simple instructions, each taking one clock cycle, can be executed.

Thus the multiplication instruction **a = a * b.** might be written:

LDA R1, A
LDA R2, B
MULT R1, R2
STO R1
A

Mnemonic	Action
LDA	Loads a value from a memory address
STA	Stores a value in a memory address
ADD	Adds the value held in a memory address to the value held in the accumulator
SUB	Subtracts from the accumulator the value held in a memory address
MOV	Moves the contents of one memory address to another

Reduced Instruction Set Computers (RISC)

The opposite approach is adopted in the more modern RISC architecture.

Only simple, smaller instructions, each taking one clock cycle, can be executed.

Thus the multiplication instruction **a = a * b.** might be written:

LDA R1, A
LDA R2, B
MULT R1, R2
STO R1
A

Complex Instruction Set Computers (CISC)

Intel processors, for example use a CISC architecture to use with modern computers.

CISC uses a **large** instruction set, used to accomplish tasks in as **few lines** of assembly language as possible.

The processor hardware is capable of understanding and executing the **series of sub-tasks** that make up a **single instruction.**

Complex instructions are built into the **machine's hardware.**

CISC Example

To multiply two values held in different memory locations A and B, storing the result in A, a processor using several general purpose registers would load each of the values into a separate register, carry out the multiplication and then store the result back in A.

The assembly language instruction for a CISC processor might be written something like

MULT A, B

A CISC processor has in its instruction set a single instruction that will do the loading, multiplication and storing of the result.

The instruction is equivalent to the high level instruction:

a = a * b

Comparison

Example:
Multiply value in memory location "X" by value in memory location "Y"; store result back into location "X". Registers "A" and "B" are available.

CISC Assembly:	**RISC Assembly:**
IMUL X, Y	LOAD A, X LOAD B, Y PROD A, B STORE X, A

Comparison

RISC	**CISC**
Smaller instruction set	Larger instruction set
Requires less complex hardware/ requires little cooling, minimising manufacture cost/ less transistors	Requires more complex hardware/ requires cooling, more expensive to manufacture/ more transistors
One clock cycle to execute an instruction	Multiple clock cycles to execute an instruction
Tends to use less energy	Tends to use more energy
Uses more RAM	Uses less RAM
Easier to pipeline	Difficult to pipeline
But compiler has to do more work to translate the code into machine code	Compiler has to do less work to translate the code into machine code
Fewer addressing modes (drawback)	More addressing modes
Applications requiring **high-speed processing** and **efficiency**, such as embedded systems and mobile devices, due to its **faster, simpler** instructions.	Preferred in general-purpose computers where **ease of programming** are important, as complex instructions can make the software simpler.

RISC back in fashion!

Mobile devices and Apple ARM processors are using RISC architectures instead of CISC

Why?

Easier to design (less transistors)
Smaller in Design
More power efficient
Cheaper to produce

But compiler has to do more work to translate the code into machine code