

# SYSTEM ARCHITECTURE



**AHMAD  
JALLOH**

# STARTER

Open Student Portal

Click on **Fafl** 

Click on **Year 10**

Click on **Add Note**

**Answer the following question;**

- 1. Describe the purpose of the CPU [2 marks]**
- 2. Describe four function of the CPU [4 marks]**
- 3. Describe the function of Control Unit (CU) [2 marks]**
- 4. Describe the function of ALU [2 marks]**
- 5. Describe the function of Cache [2 marks]**

# STARTER

- 1. Describe the purpose of the CPU [4 marks]**
  - The purpose of the central processing unit to perform process data and perform calculation
  - The central processing unit also perform the fetch, decode and execute cycle
- 2. Describe four function of the CPU [4 marks]**
  - Fetch instruction or data from RAM
  - Decode the instruction – This could be convert instruction to binary
  - Execute the instruction – carry out the instruction
  - Repeat the fetch, decode and execute cycle
- 3. Describe the function of Control Unit (CU) [2 marks]**
  - It manages and monitors hardware on the computer to ensure the correct data goes to the correct hardware.
  - It manages the input and output signals ensuring these are dealt with correctly.
  - It manages the Fetch-Decode-Execute cycle.

# STARTER

## **Describe the function of ALU [2 marks]**

- **The Arithmetic Logic Unit is where all the calculation and comparison takes place**
- **Arithmetic part**, which performs calculations on the data, e.g.  $3 + 2 = 5$
- **Logic part** – which deals with logical operations such as is True / False / Equal to / Greater than etc.

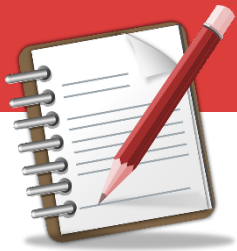
## **Describe the function of Cache [2 marks]**

- The cache memory stores the instruction/data that frequently processed
- The instruction/data in cache can be access faster than accessing them from RAM
- The larger the cache the faster the faster the computer system



# STARTER

- Student Resources
- Computing
- Year 10
- GCSE Computer Science 9 - 1
- Unit 1 –Computer System
- Copy **System Architecture** folder
- Paste it your **Unit 1 - Computer System** folder

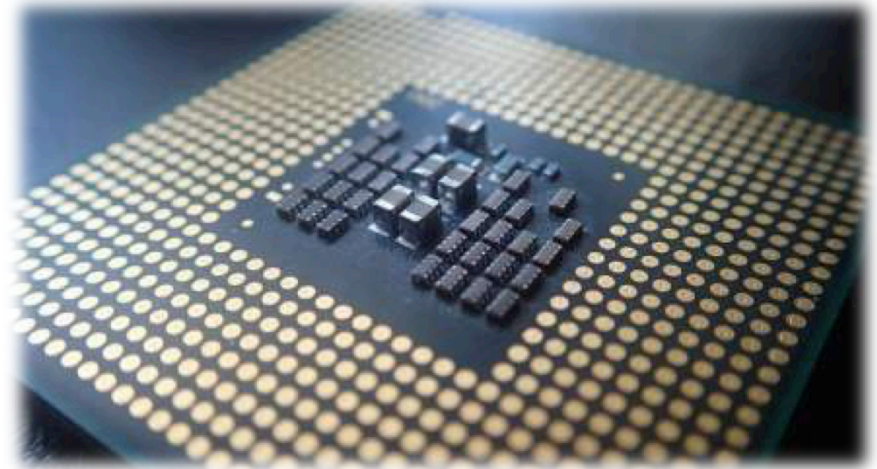


# ACTIVITY



**5 minutes to discuss and feedback on the following:**

- 1. What is this?**
- 2. What is it for?**

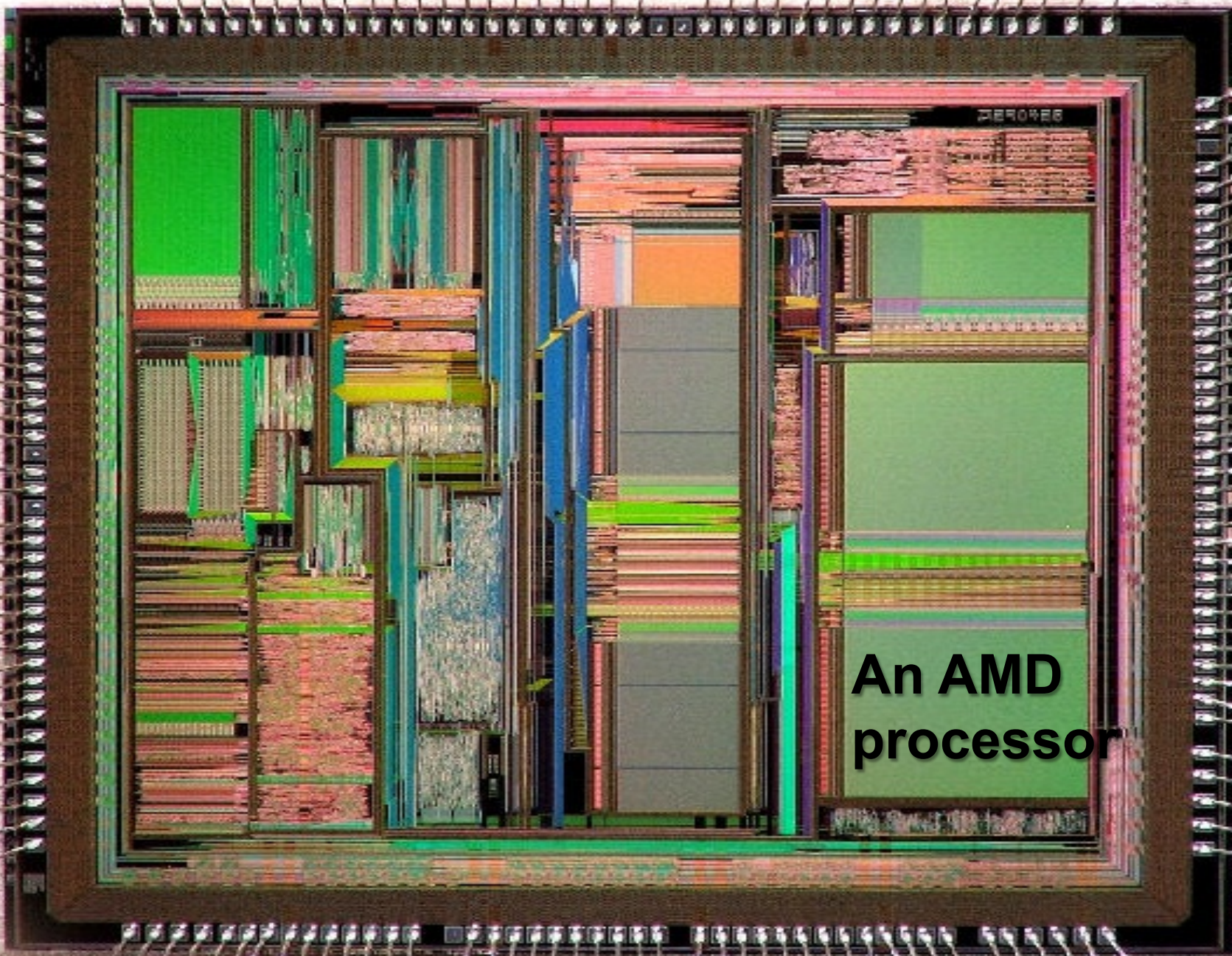


**What do you notice?**

**What questions do you have?**

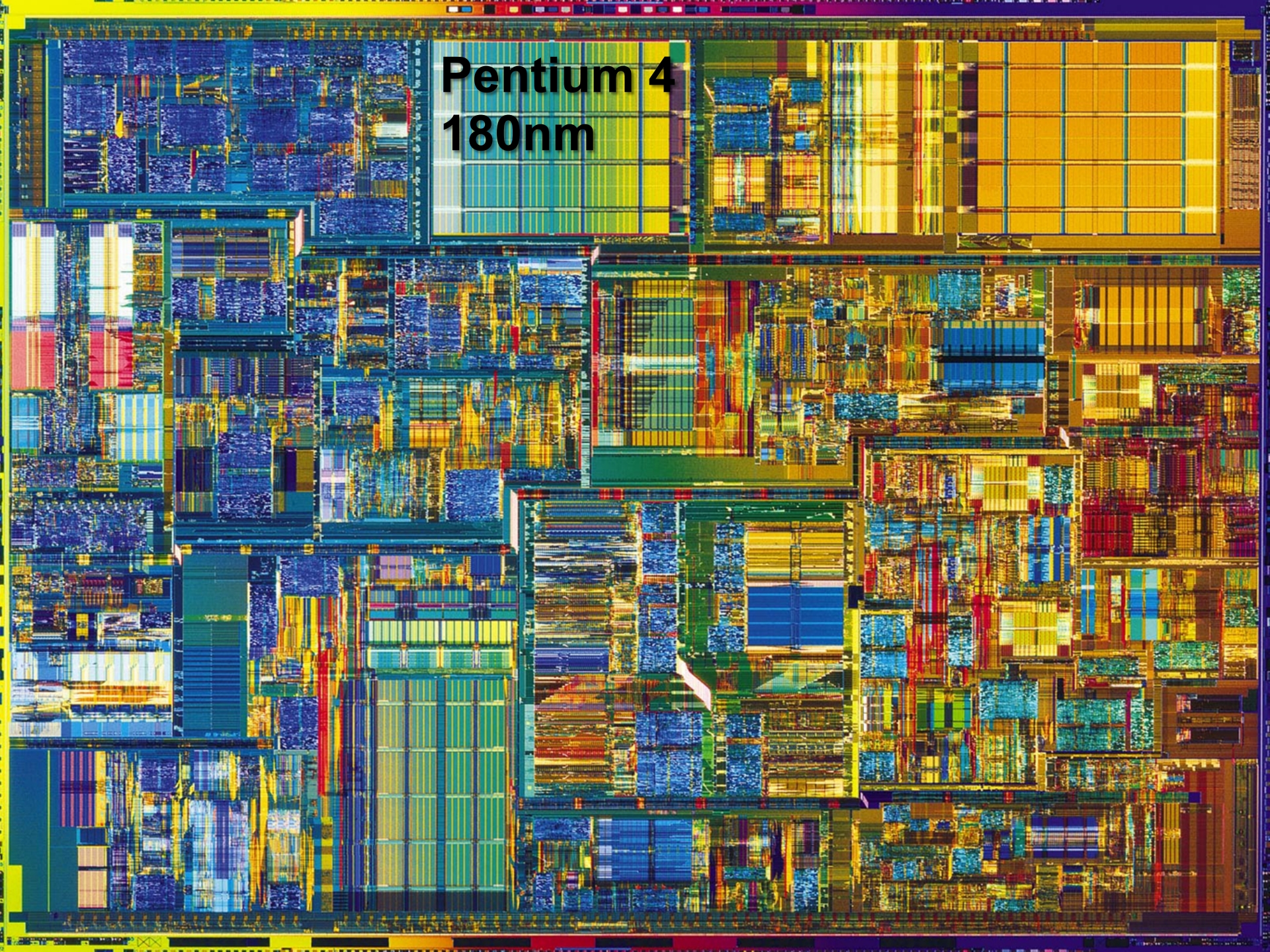
**What do you think? (hypothesis)**





**An AMD  
processor**





**Pentium 4  
180nm**



# PURPOSE AND FUNCTIONS OF THE CPU

**The main purpose of the CPU are**

- To **processing** data such as **searching, sorting, loading and saving data**
- To perform **Calculating** and **decision making**
- Perform **Fetch – Decode – Execute Cycle**

**The main function of the CPU are**

- Fetch data and instruction from main memory (RAM or Cache)
- Decode the instruction (Convert to Binary)
- Execute instruction
- Repeat Fetch – Decode – Execute Cycle
- Copy the location of instruction and data into the Memory Address Register

# BEFORE THE VON NEUMANN ARCHITECTURE

## History

Before 1945, computers were essentially preprogrammed machines.

This meant that a computer would typically be set up, with wires and switches to perform a specific task.

In this way, a computer program was considered to be part of the machine.

The only thing it would be given is data to be processed.

# THE VON NEUMANN ARCHITECTURE

Von Neumann



But in 1945, a mathematician from the USA called **John Von Neumann**, had an idea.

He wondered if it would be possible to create a computer where the program (and its data) could be stored together, independent of 'the machine'.

This meant that the same computer could work, no matter what program it was given.

No more hours setting up machines, instead, the time would be spent on creating the program instructions!

# FEATURES OF THE CPU

The CPU is the core of every computer system and has four main components:

1. The **control unit** which manages the **Fetch-Decode-Execute** cycle and control flow of data.
2. The **ALU – Arithmetic and Logic Unit** – carries out all of the arithmetic and logical operations, including:
  - **Addition;**
  - **Subtraction;**
  - **Comparisons** (for example, equal to, less than, greater than).
3. **Register** - used to temporarily hold bits of data needed by the CPU.
4. **Cache Memory** - Immediate Access Store (IAS)

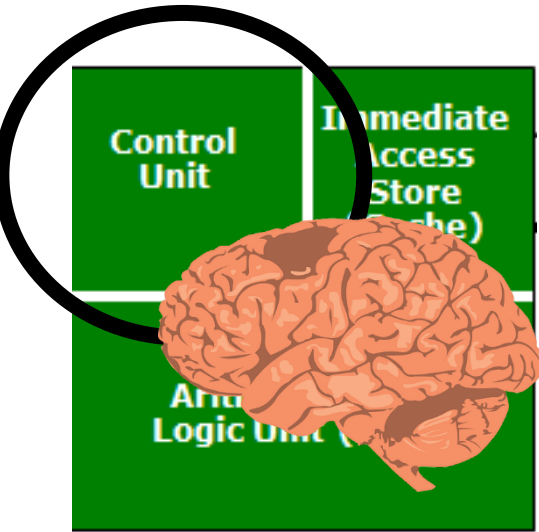


# THE CPU – CENTRAL PROCESSING UNIT

## The Control Unit

There are three main jobs of the Control Unit:

1. It manages and monitors hardware on the computer to ensure the correct data goes to the correct hardware.
2. It manages the input and output signals ensuring these are dealt with correctly.
3. It manages the Fetch-Decode-Execute cycle.



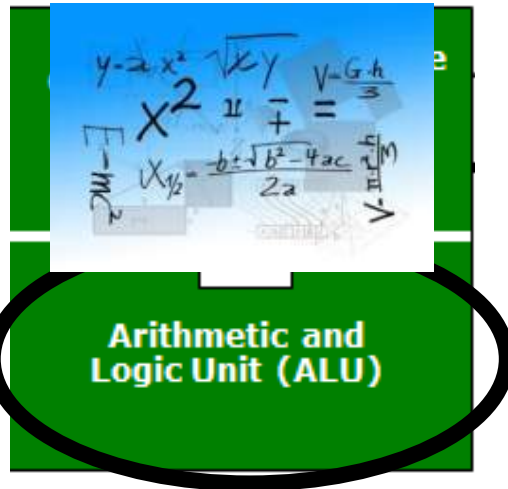
# THE CPU – CENTRAL PROCESSING UNIT

## Arithmetic and Logic Unit (ALU)

This is where the CPU actually carries out the maths and logic on the data (processes it).

It has two parts:

- **Arithmetic part**, which performs calculations on the data, e.g.  $3 + 2 = 5$
- **Logic part** – which deals with logical operations such as is True / False / Equal to / Greater than etc.

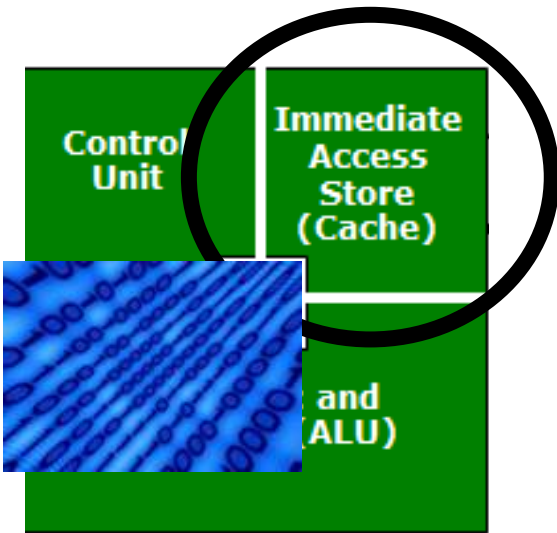


# Registers

- The CPU has various registers which are used to temporarily hold bits of data needed by the CPU.
- They are very quick to read or write to, much quicker than any other form of memory.
- The main registers you need to know about are:
  - Memory Address Register (**MAR**)
  - Memory Data Register (**MDR**)
  - The program counter (**PC**)
  - Accumulator

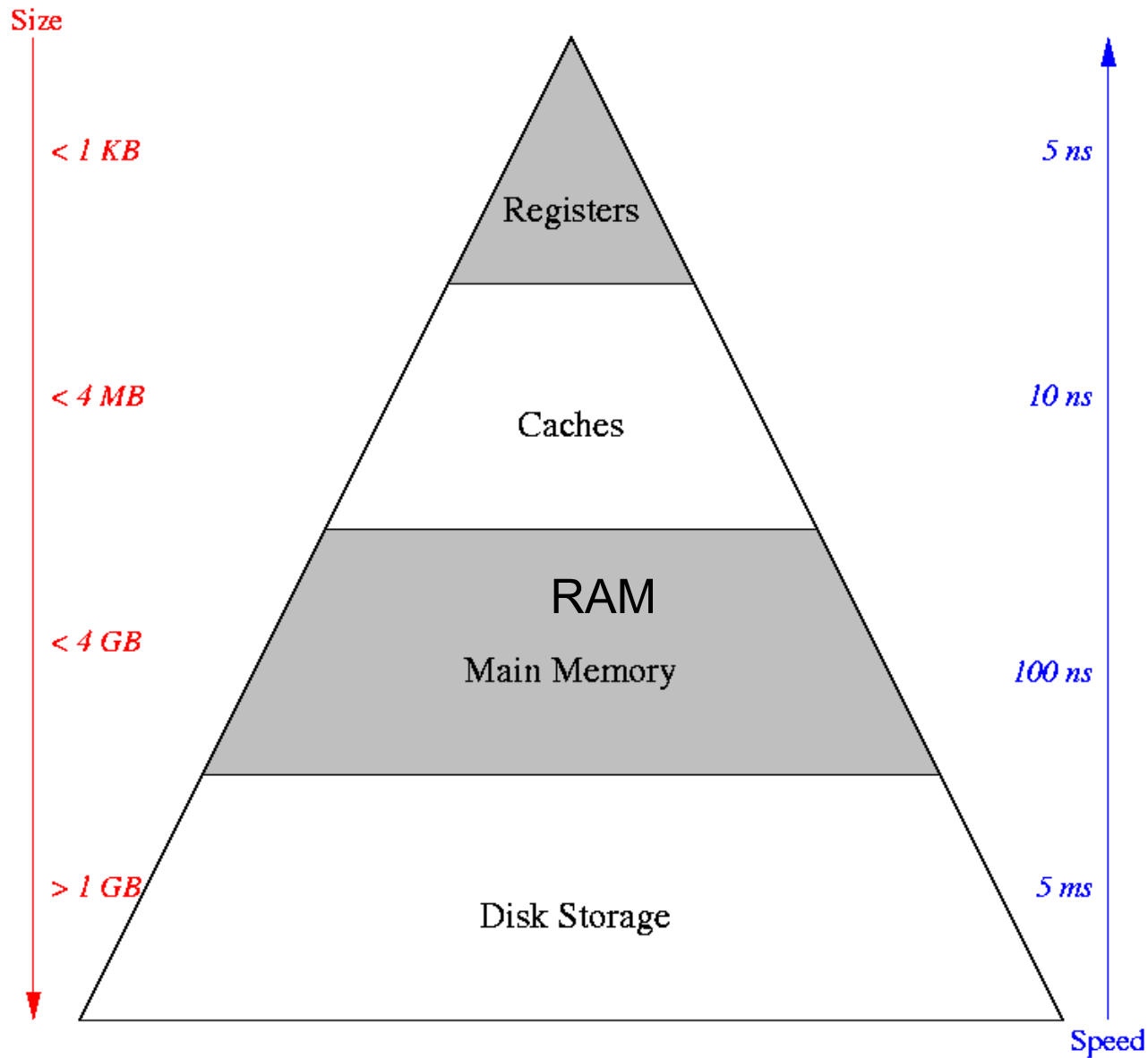
# THE CPU – CENTRAL PROCESSING UNIT

## Immediate Access Store (Cache)



- This part stores the data which is to be immediately processed.
- The CPU takes a chunk of data / instructions from the RAM and keeps it close so that it always has a constant supply of data to process.
- If data and instructions were downloaded from RAM one item at a time, the CPU would work far slower because the CPU cycles much faster than the RAM can deliver data.
- So instead, chunks are downloaded and stored on the CPU so the CPU doesn't spend wasted time waiting for a deliver of data.

# Memory Hierarchy

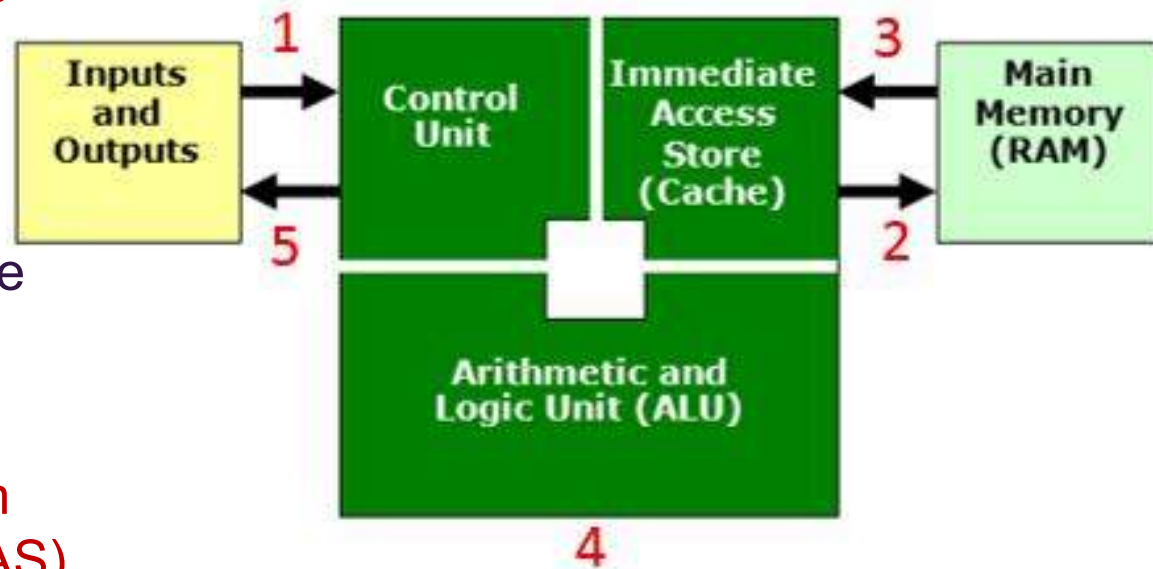


# Registers

CPU REGISTER	FUNCTION
<b>Memory Address Register (MAR)</b>	Memory Address Register - This holds a memory address of instruction or data to be accessed by the CPU.
<b>Memory Data Register (MDR)</b>	This holds the actual data/instruction that has been retrieved from memory (in case of read) or to be stored into memory (in case of write).
<b>Program Counter</b>	Program Counter stores the address or location of the next instruction to be run This automatically ticks to the next memory address when an instruction is loaded. It increment by 1 after each cycle
<b>Accumulator</b>	This stores data that is being used in calculations. Accumulator stores the result of the ALU
<b>Instruction Register (CIR)</b>	This holds the instruction currently being executed or decoded. It also known as the Current Instruction Register (CIR)

# SUMMARY OF THE CPU

1. An input device (e.g. keyboard) sends data to the CPU. The Control Unit receives this data.
2. The Control Unit sends this data into main memory to be used later.
3. When the time is right, the data will be transferred from main memory into cache (IAS)
4. The data will then be sent to the ALU for processing
5. The control unit will send the processed data back (for example to an output device such as a screen or monitor).



# Activity – Von Neumann

- Open the “**System Architecture Booklet**”
- Complete all actives in the **Von Neumann Architecture** section



# STARTER

Open Student Portal

Click on **Fafl** 

Click on **Year 10**

Click on **Add Note**

**Answer the following question;**

- 1. Describe the purpose of MAR [2 marks]**
- 2. Describe the purpose of MDR [2 marks]**
- 3. Describe the purpose of Program Counter [2 marks]**
- 4. Describe the purpose of Accumulator [2 marks]**

# STARTER

## 1. Describe the purpose of MAR [2 marks]

- Memory Address Register - This holds a memory address of instruction or data to be accessed by the CPU.

## 1. Describe the purpose of MDR [2 marks]

- Memory Data Register - This holds the actual data/instruction that has been retrieved from memory (in case of read) or to be stored into memory (in case of write).

## 1. Describe the purpose of Program Counter [2 marks]

- Program Counter stores the address or location of the next instruction to be run. It increments by 1 after each cycle

## 1. Describe the purpose of Accumulator [2 marks]

- This stores data that is being used in calculations. Accumulator stores the result of the ALU

# STARTER

Open Student Portal

Click on **Fafl** 

Click on **Year 10**

**Read the feedback from your Teacher;**

1. Click on **Add Note**
2. **Response to the feedback**
3. **Answer the questions on the Target section**
  1. **All question must be answered in exam condition**
  2. **Answer all question as detail as you can.**

# CPU INSTRUCTIONS SET

Every CPU has a set of instructions to perform its functions, called “Instructions Set”

Some of these instructions includes;

- **OUTPUT** – this output to a device such as monitor
- **INPUT** – this input from a devices such as a keyboard
- **LOAD** – this load number from RAM into the C PU
- **ADD** – this add two number together
- **STORE** – this store number from the CPU back out of RAM
- **COMPARE** – this compare one number to another
- **Branch if Zero** – This jump to another address in RAM if the value in accumulator is zero
- **Branch** – this jump to another address in RAM

# INSTRUCTIONS SET

**Move (Number of Step ) Forward**

**Turn (Number of Degree) Right**

**Turn (Number of Degree) Left**

# CPU: THE BOOT PROGRAM

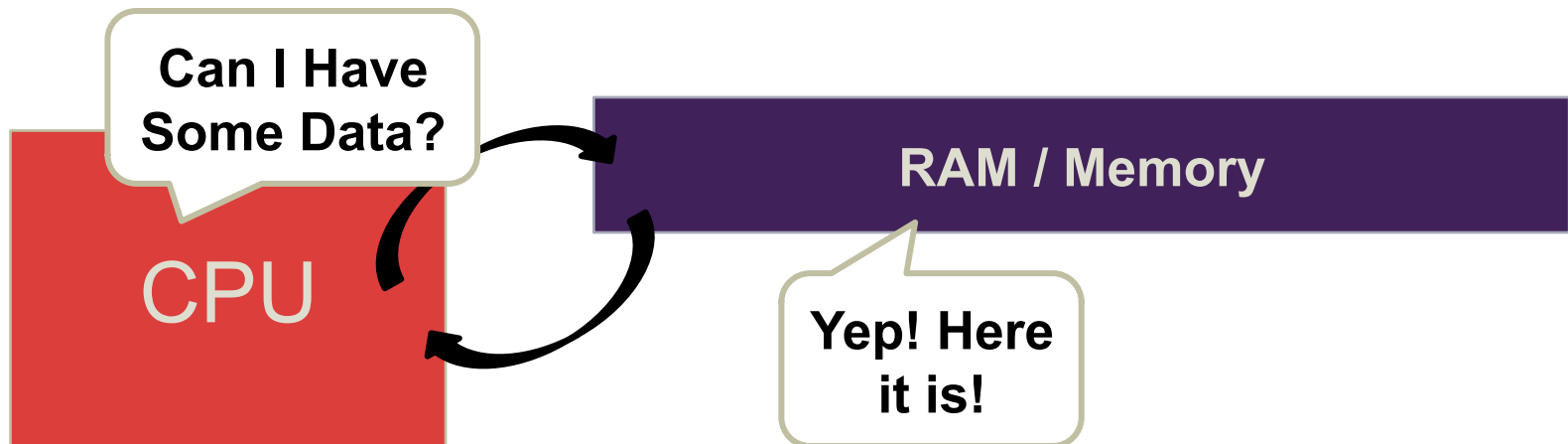
## The Boot Program

- Immediately after being switched on, the CPU looks in a specific location in **read only memory (ROM)** for the first program to load and execute.
- This is the boot sequence.
- The boot process gets the computer up and running and the operating system started.
- After this initial boot process is complete, control is handed to the operating system to provide the programs for the CPU to run.

# HOW THE CPU COMMUNICATES

## 1 - The Fetch step:

- In this step the CPU fetches some data and instructions from main memory (RAM) and then store them in its own temporary memory called 'registers'.
- The memory address of the instruction is stored in a register called the Program Counter (PC) so the CPU can keep track of which instruction is next.
- After an instruction is fetched, the PC is updated so the CPU knows the address of the next instruction it has to fetch.



# THE CPU – CENTRAL PROCESSING UNIT

## The Fetch Stage - Continued

- For this to happen, the CPU uses a piece of hardware path called the 'address bus'.
- The address of the next item that the CPU wants is put onto the 'address bus'.



- Data from this area then travels from the RAM to the CPU on another piece of hardware called the 'Data Bus'

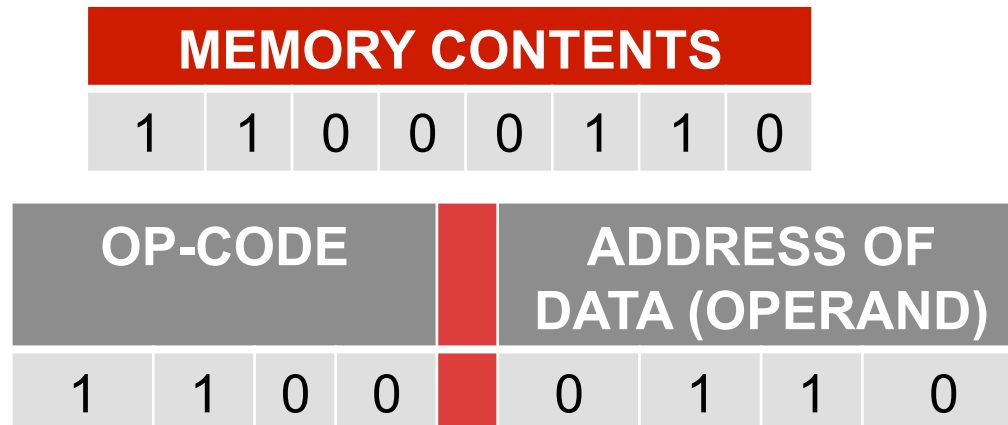




# HOW THE CPU COMMUNICATES

## 2 - The Decode step:

- The decode step is where the CPU understands or works out what the instruction it has just fetched actually means.
- This means the micro-processor will look the binary code up in a table.
- The CPU 'decodes' the instruction and gets things ready for the next step.

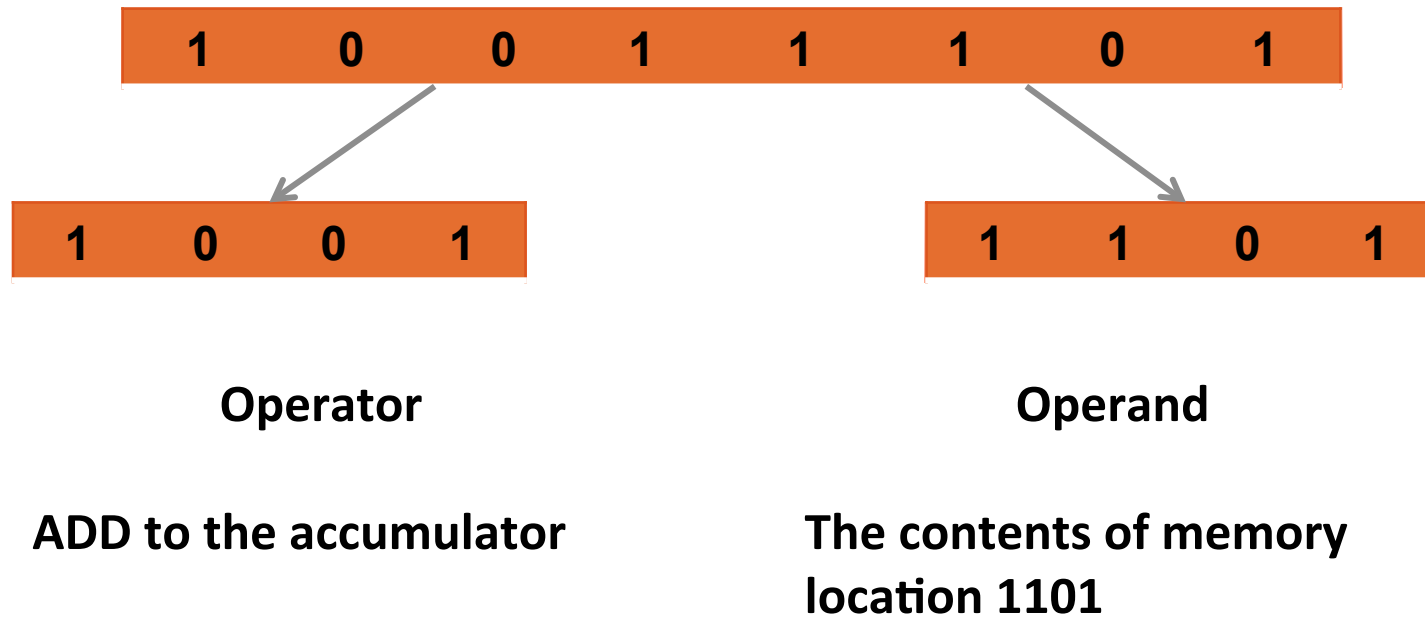


# PROGRAM INSTRUCTIONS

A program instruction has two parts –

1. **Opcode** – This is the instruction part that tells the processor what to do such as **ADD** or move a byte of data ,
2. **Operand** – This the data part.

The operand might be an actual number, or more commonly it will be an address where the required data can be found or where it must be sent.



# HOW THE CPU COMMUNICATES

## 3 - The Execute Step:

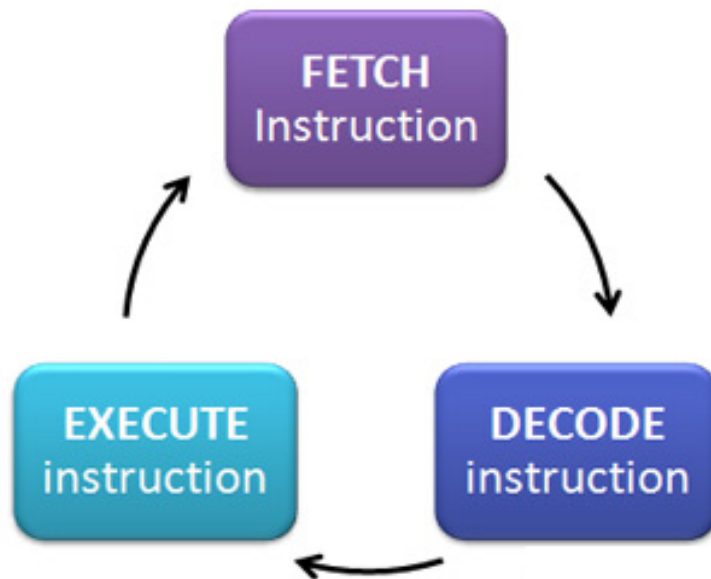
- In this step the **control unit** links together the parts of the CPU that are needed to **execute the instruction**.
- The Execute stage is where data processing happens.
- Instructions are carried out on the data.
- Once a cycle has completed, another begins.

# INSTRUCTION

- When a computer is instructed to run a program it is directed to the start address for these data and instructions.
- The **CPU fetches** the first instruction from this start location and decodes it to find out what to do next.

# ENCODING INSTRUCTIONS

- When an instruction is passed to a micro-processor the first thing which happens is the opcode is decoded.
- This means the micro-processor will look the binary code up in a table.
- Once the instruction is found it will execute it.
- This is called fetch execute cycle



# HOW THE CPU COMMUNICATES

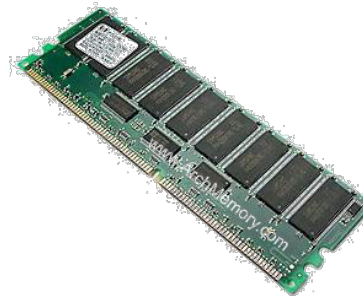


Software

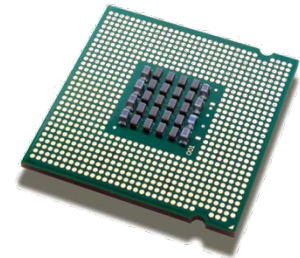
1. Software, like office, are stored on the hard drive. To run the software we must “load” it.



Hard drive



Memory



CPU

# HOW THE CPU COMMUNICATES



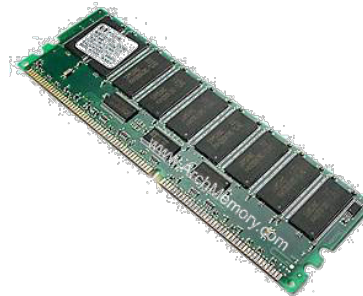
Software

2. The operating system (like windows) will be running on the CPU.

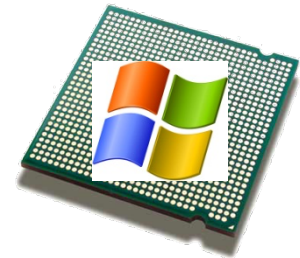
When the user selects to load office up, it is the OS which starts it off.



Hard drive



Memory



CPU

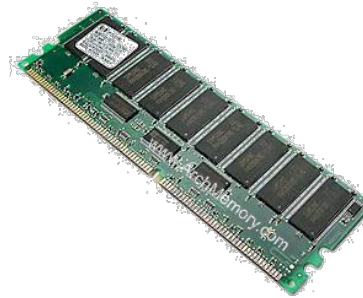
# HOW THE CPU COMMUNICATES

3. Software is loaded from the hard drive and stored in main memory

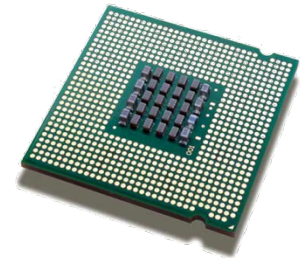
Software



Hard drive



Memory



CPU



# HOW THE CPU COMMUNICATES

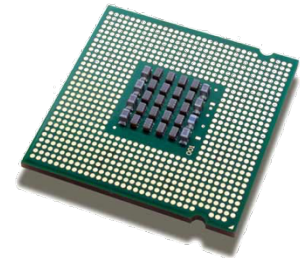
4. The CPU then transfers instructions from memory into the CPU in order to run them. It does this ONE at a time!



Hard drive



Memory



CPU

# Activity – Instructions Set

- Open the “**System Architecture Booklet**”
- Complete all actives in the **Instruction Set** section

# What is the Little Man Computer?

- **The Little Man Computer (LMC) is a simulator which copies the basic features of a modern computer.**
- **It is based on the **Von Neumann** architecture featuring a central processing unit consisting of**
  - An Arithmetic Logic unit
  - Registers,
  - A control unit containing an instruction register and program counter,
  - Input and Output mechanisms and
  - Memory (RAM) to store both data and instructions.

# What is the Little Man Computer?

- The LMC is based on the concept of a little man (shut in a small room or inside a computer) acting like the control unit of a CPU i.e.
  - **Fetching** instructions from memory,
  - **Decoding** and
  - **Executing** the instructions, and managing the input and output.
- The LMC can be programmed using either "Machine Code" or "Assembly Language".

# Activity – CPU FETCH DECODE EXECUTE

- Get yourself in a group of 3
- Decide who will play each of the following roles
  - **FETCHER:** Goes to Memory (Teacher) asking for instructions at specific memory addresses
  - **DECODER:** uses a cypher to decode the instructions fetched from memory
  - **EXECUTER:** executes the instructions, which are all about PLOTTING on a grid to make an image.
- The teacher (in this case Me) acts as MEMORY, holding all the instructions in numbered memory locations.
- Each team starts executing instructions at Memory location 1.

# COMPUTATIONAL THINKING CONCEPTS

## Your instructions:

- **Fetcher** - Collect one instruction at a time from the teacher (RAM).
- **Decoder** - Decode the red and blue numbers.
- **Executer** - Fill in the square:
  - Blue is the X position (left and right)
  - Red is the Y position (up and down)

## Examples:

0011

Blue			
8	4	2	1
0	0	1	1

$$2 + 1 = 3$$

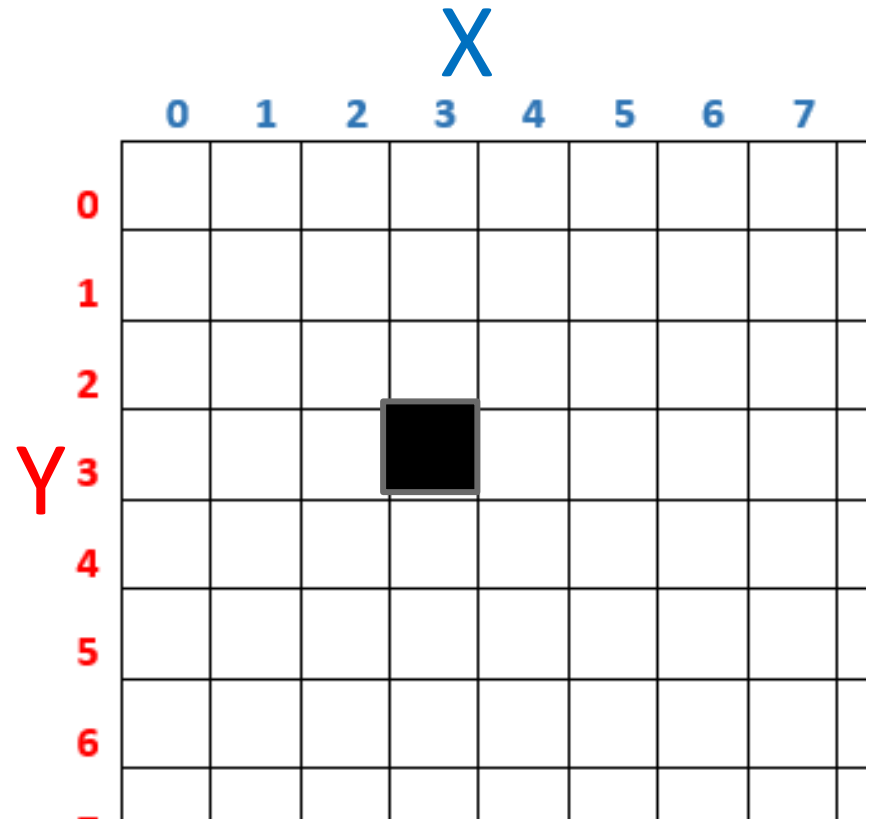
$$X = 3$$

0011

Red			
8	4	2	1
0	0	1	1

$$2 + 1 = 3$$

$$Y = 3$$



# FETCH, DECODE AND EXECUTE

## DECODING SYSTEM

A	B	C	D	E	F	G	H	I	J	K	L	M
X	Y	Z	A	B	C	D	E	F	G	H	I	J

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
K	L	M	N	O	P	Q	R	S	T	U	V	W

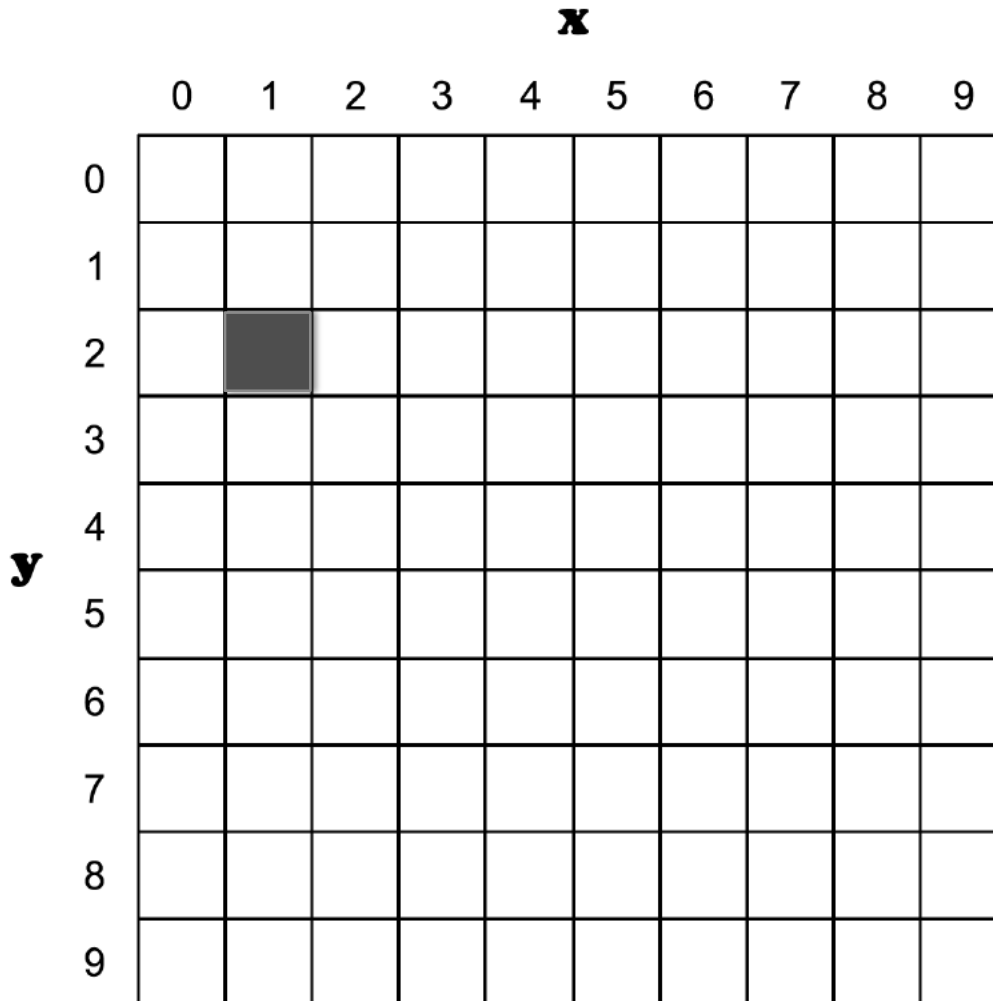
- **Example**

- RN → **OK**
- KHOOR → **HELLO**

# PLOTTING ACTIVITY (EXECUTER)

Fill in the cell as indicated by the instructions

PLOT (n1, n2)  $\rightarrow$  X = n1, Y = n2





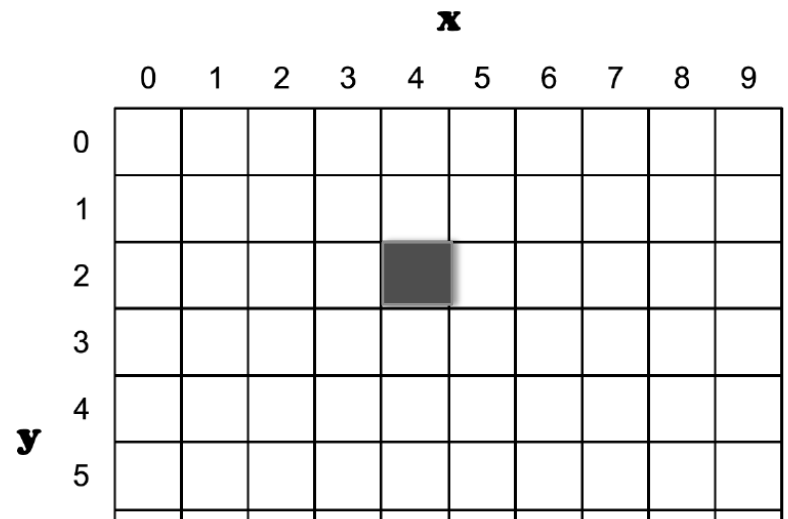
# INSTRUCTION TO DECODE

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>
X	Y	Z	A	B	C	D	E	F	G	H	I	J

<b>N</b>	<b>O</b>	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>
K	L	M	N	O	P	Q	R	S	T	U	V	W

INSTRUCTION AT MEMORY LOCATION	INSTRUCTION TO BE DECODED AND EXECUTED	THE NEXT INSTRUCTION IS AT MEMORY LOCATION... ↓
<b>1</b>	<b>SORW (IRXU , WZR)</b>	<b>6</b>

**PLOT (Four, Two)**



# Activity – Instructions Set

- Open the “**System Architecture Booklet**”
- Complete the activity in the **Fetch Decode Execute** section

# Which LMC?

**Assembly Language Code**

```
00 INP
01 STA 99
02 INP
03 ADD 99
04 OUT
05 HLT
// Output the sum of two
numbers
```

**CPU**

00 PROGRAM COUNTER

INSTRUCTION REGISTER

ADDRESS REGISTER

ACCUMULATOR 000

ARITH-METIC UNIT

**RAM**

0	1	2	3	4	5	6	7	8	9
901	399	901	199	902	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

**INPUT**

01

RUN/STEP your program, SELECT, LOAD or edit program

ASSEMBLE INTO RAM RUN STEP

RESET LOAD HELP add

OPTIONS ©GCSEcomputing.org.uk and Peter Higginson

•There are many implementations of the Little Man Computer (LMC).

•We will use the excellent web based one that can be found here:

<http://peterhigginson.co.uk/LMC/>

# Parts of the LMC

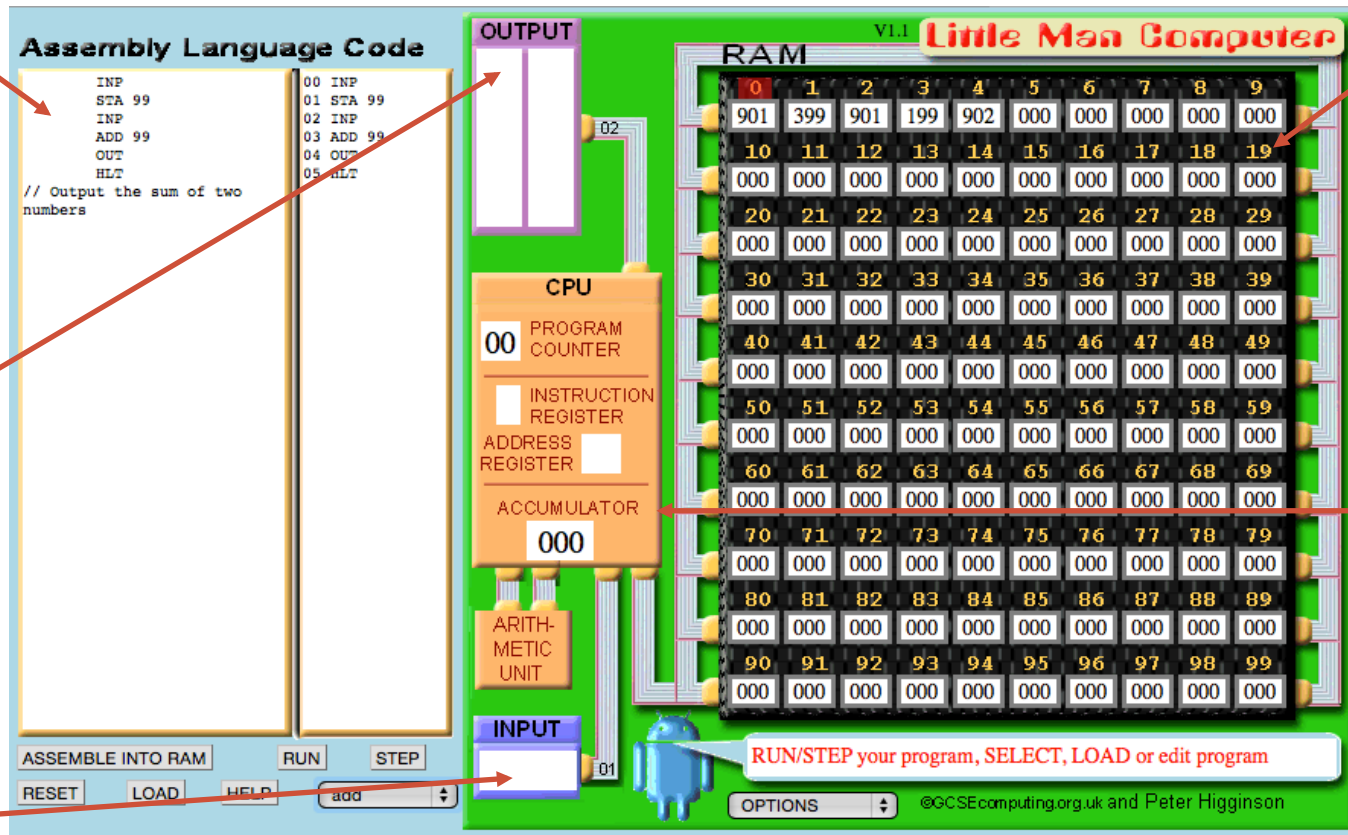
Assembly instructions

Output

Input

RAM with 100 memory locations

CPU with 4 registers



RUN/STEP your program, SELECT, LOAD or edit program

©GCSEcomputing.org.uk and Peter Higginson

# Parts of the CPU

CPU REGISTER	FUNCTION
<b>Accumulator</b>	This stores data that is being used in calculations. It can perform simple addition and subtraction.
<b>Program Counter</b>	This contains the memory address of the next instruction to be loaded. This automatically ticks to the next memory address when an instruction is loaded. It can be altered during the running of the program depending on the state of the accumulator.
<b>Instruction Register</b>	An Instruction Register to hold the top digit of the instruction read from memory. It holds the actual instruction being executed
<b>Memory Address Register (MAR)</b>	An Address Register hold the bottom two digits of the instruction read from memory. This holds a memory address of data to be accessed.
<b>Memory Data Register (MDR)</b>	This holds the data/instruction that has been retrieved from memory (in case of read) or to be stored into memory (in case of write).
<b>Input</b>	This registers allows the user to input numerical data to the LMC.
<b>Output</b>	This shows the data to output to the user.

# LMC Instruction set

MNEMONIC CODE	INSTRUCTION	NUMERIC CODE	DESCRIPTION
<b>ADD</b>	<b>ADD</b>	<b>1xx</b>	Add the value stored in mailbox xx to whatever value is currently on the accumulator (calculator). Note: the contents of the mailbox are not changed, and the actions of the accumulator (calculator) are not defined for add instructions that cause sums larger than 3 digits.
<b>SUB</b>	<b>SUBTRACT</b>	<b>2xx</b>	Subtract the value stored in mailbox xx from whatever value is currently on the accumulator (calculator). Note: the contents of the mailbox are not changed, and the actions of the accumulator are not defined for subtract instructions that cause negative results - however, a negative flag will be set so that 8xx (BRP) can be used properly.
<b>STA</b>	<b>STORE</b>	<b>3xx</b>	Store the contents of the accumulator in mailbox xx (destructive). Note: the contents of the accumulator (calculator) are not changed (non-destructive), but contents of mailbox are replaced regardless of what was in there (destructive)
<b>LDA</b>	<b>LOAD</b>	<b>5xx</b>	Load the value from mailbox xx (non-destructive) and enter it in the accumulator (destructive).
<b>INP</b>	<b>INPUT</b>	<b>901</b>	Go to the INBOX, fetch the value from the user, and put it in the accumulator (calculator) Note: this will overwrite whatever value was in the accumulator (destructive)
<b>OUT</b>	<b>OUTPUT</b>	<b>902</b>	Copy the value from the accumulator (calculator) to the OUTBOX. Note: the contents of the accumulator are not changed (non-destructive).

# LMC Instruction set

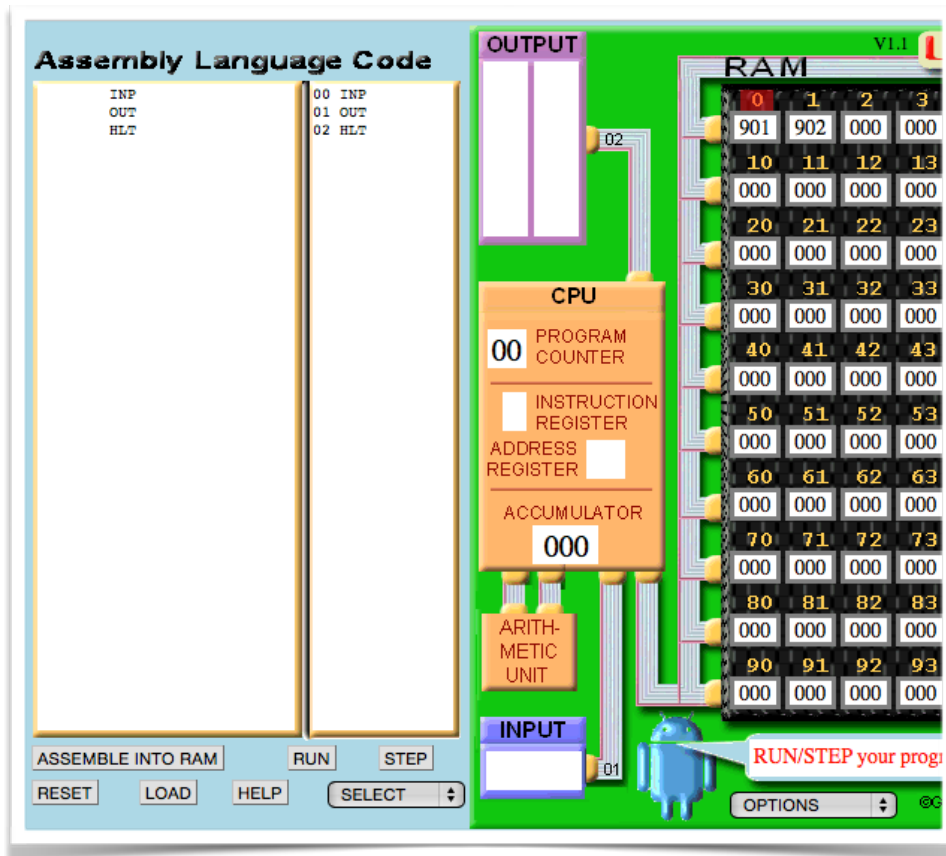
MNEMONIC CODE	INSTRUCTION	NUMERIC CODE	DESCRIPTION
<b>BRA</b>	<b>BRANCH (unconditional)</b>	<b>6xx</b>	Set the program counter to the given address (value xx). That is, value xx will be the next instruction executed.
<b>BRZ</b>	<b>BRANCH IF ZERO (conditional)</b>	<b>7xx</b>	If the accumulator (calculator) contains the value 000, set the program counter to the value xx. Otherwise, do nothing. Note: since the program is stored in memory, data and program instructions all have the same address/location format.
<b>BRP</b>	<b>BRANCH IF POSITIVE (conditional)</b>	<b>8xx</b>	If the accumulator (calculator) is 0 or positive, set the program counter to the value xx. Otherwise, do nothing.
<b>HLT</b>	<b>HALT</b>	<b>0</b>	Stop working.
<b>DAT</b>	<b>DATA</b>		This is an assembler instruction which simply loads the value into the next available mailbox. DAT can also be used in conjunction with labels to declare variables. For example, DAT 984 will store the value 984 into a mailbox at the address of the DAT instruction.

# Activity – Instructions Set

- Open the “**System Architecture Booklet**”
- Complete the activity in the **LMC Instruction Set** section



# Examples - input & output



• This program simply asks the user for an input and then outputs what was input.

- INP
- OUT
- HLT

• The program has been assembled into RAM and you can see the numeric codes for the instructions in the first three memory locations.

• Try the example above and save a copy on Notepad as "Input and Output"

# Activity – Little Man Computer

- Open the “**System Architecture Booklet**”
- Complete the activity in the **Little Man Computer Activities** section

# Using memory (Variable)

```
    INP
    STA FIRST
    INP
    STA SECOND
    LDA FIRST
    OUT
    LDA SECOND
    OUT
    HLT
FIRST DAT 0
SECOND DAT 0
```

- Try the example on the left and save a copy on Notepad as *“Load Variable”*
- This program asks the user to input a number.
- This is stored in a memory location defined by the DAT label.
- A second number is asked for and stored.
- These numbers are then loaded and output in order.

# Adding

```
    INP
        STA FIRST
        INP
    STA SECOND
    LDA FIRST
        ADD SECOND
        STA ANSWER
    LDA ANSWER
        OUT
        HLT
FIRST    DAT 0
SECOND  DAT 0
ANSWER  DAT 0
```

- Try the example on the left and save a copy on Notepad as “*Adding two Variables*”
- This program asks the user to input a number.
- This is stored in a memory location defined by the DAT label.
- A second number is asked for and stored.
- These numbers are then add together
- These numbers are then add together and display the result

# LMC Activity Using Memory

- **Guess what this program does**
- Try the example on the left and save a copy on Notepad as ***“Adding and Subtracting Variable”***

```
INP
STA FIRST
INP
ADD FIRST
STA SECOND
SUB ONE
OUT
HLT
```

```
FIRST DAT 0
SECOND DAT 0
ONE DAT 1
```

- This program asks the user to input a number.
- This is stored in a memory location defined by the DAT label.
- A second number is asked for and stored.
- These numbers are then add together
- These numbers are then add together
- After the addition one is subtracted and display the result
- NOTE: the value 1 in a variable called ONE

# Activity – Little Man Computer

- Open the “**System Architecture Booklet**”
- Complete the activity in the **Little Man Computer Activities** section

# BRANCHES AND LOOPS IN LMC

## BRA – Branch Always

This is the same as a forever loop in Scratch. It is an unconditional loop.



```
LOOPTOP LDA A
        OUT
        BRA LOOPTOP
```

```
A      DAT 5
```

- Try the example on the left and save a copy on Notepad as “*Branch Always*”.
- Add comments explaining what the code does
- This example will keep outputting the value which is stored in A forever.
- The program will not stop.



# BRANCHES AND LOOPS IN LMC

## BRZ – Branch if Zero

- This allows us to branch our program if the value in the accumulator is zero.
- This can be used to create counting loops similar to the *repeat* loop in Scratch or a *for* loop in Python.



```
LOOPTOP LDA A
        SUB ONE
        STA A
        OUT
        BRZ END
        BRA LOOPTOP
END     HLT
```

```
A      DAT 5
ONE    DAT 1
```

- Try the example on the left and save a copy on Notepad “*Branch if Zero*”.
- Add a comment describing what it does
- This example uses a **BRA** to create the loop and
- A **BRZ** to break the loop.

# BRANCHES AND LOOPS IN LMC

## BRP – Branch if Positive

- This also allows us to create a branch in our program, depending on whether the contents of the accumulator are positive or not.
- These can also be used to create loops.

```
LOOPTOP  LDAA  
         SUB ONE  
         STAA  
         OUT  
         BRP LOOPTOP  
         HLT  
A        DAT 5  
ONE     DAT 1
```

- Try the example on the left and save a copy on Notepad as “*Branch if Positive*”.
- Add a comment describing what it does
- This example uses a **BRP** to both create and terminate the loop.

# Activity – Little Man Computer

- Open the “**System Architecture Booklet**”
- Complete the activity in the **Little Man Computer Activities** section

# Bigger

```
                INP
                STA FIRST
                INP
                STA SECOND
                SUB FIRST
                BRP SECONDBIG
                LDA FIRST
                OUT
                HLT
SECONDBIG      BRZ SAME
                LDA SECOND
                OUT
                HLT
SAME           LDA ZERO
                OUT
                HLT
FIRST          DAT 0
SECOND        DAT 0
ZERO          DAT 0
```





- This program uses two branch commands to alter the path of the program.
- There is no greater than or less than command so we simply subtract the second number from the first.
- If it is positive then the first number must have been bigger so we branch if positive.
- The two numbers could be the same however so we need to check to see if the result is zero. We branch if it is.
- The biggest number is output or zero if they are both the same.

# Writing a Little Man Computer Program

- Writing an LMC program can be quite a challenge.
- As the instruction set is very limited we often need to perform what seems to us to be a very simple task in an even simpler way.
- Using a Flow chart to help write the program is very helpful.
- When the flow chart is created we can simply look at each shape on the chart and think what instructions would we need to have for that shape.
- These will often be no more than a couple of lines of LMC code.

# How to Write a Little man Computer program

-In flow charts there are 4 symbols that we commonly use.

SYMBOL	MEANING	LMC INSTUCTIONS
	<b>Start / Stop</b>	Start has no instruction but Stop is <b>HLT</b> .
	<b>Input &amp; Output</b>	Any inputs will that need to be saved will be <b>INP</b> followed by an <b>STA</b> command to store the value. <b>OUT</b> is the output command. It may need to be
	<b>Process</b>	This could be a <b>DAT</b> command where we see variables initialised (e.g. counter = 0). addition and subtraction commands fit into this. A process such as $X = X + Y$ would need to be done in the correct order. So we would Load X, Add Y and then store the result as X. This would be <b>LDA X, ADD Y, STA X</b>
	<b>Decision</b>	There are only two instructions that can have two alternatives. <b>Branch if Positive</b> and <b>Branch if Zero</b> . If the test is true then the program can branch to another part of the program. If not the program carries on.

# Using memory (Variable)

```
    INP
    STA FIRST
    INP
    STA SECOND
    LDA FIRST
    OUT
    LDA SECOND
    OUT
    HLT
FIRST DAT 0
SECOND DAT 0
```

- This program asks the user to input a number.
- This is stored in a memory location defined by the DAT label.
- A second number is asked for and stored.
- These numbers are then loaded and output in order.

# Using Memory (Variable)

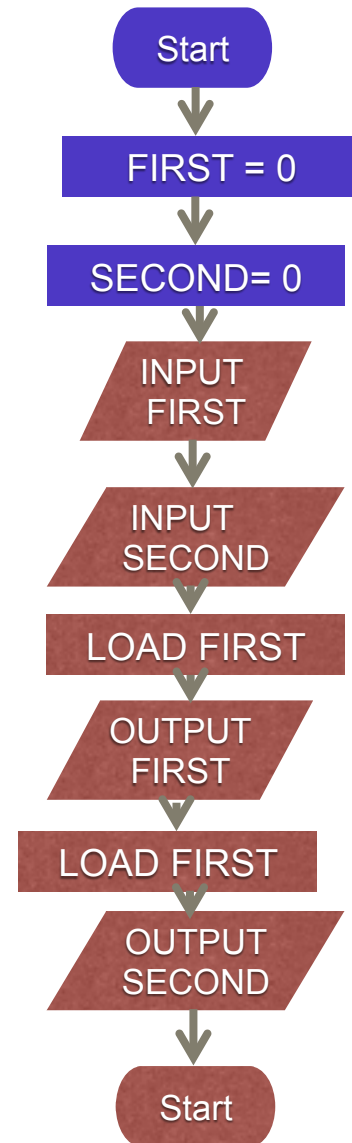
- First thing - create a flow chart to show what needs to be done.
- Be as detailed as you can be.
- Note any values you need to remember.
  - These will be the variables.
  - In LMC code they will become the DAT commands.
  - Note if they have a start value.

```
INP
STA FIRST
INP
STA SECOND
LDA FIRST
OUT
LDA SECOND
OUT
HLT
FIRST DAT 0
SECOND DAT 0
```



# Using Memory (Variable)

- We start
- We have 2 variables
- **FIRST DAT 0**
- **SECOND DAT 0**
  
- **NOTE:** The variable is at the top of the Program in the Flowchart.

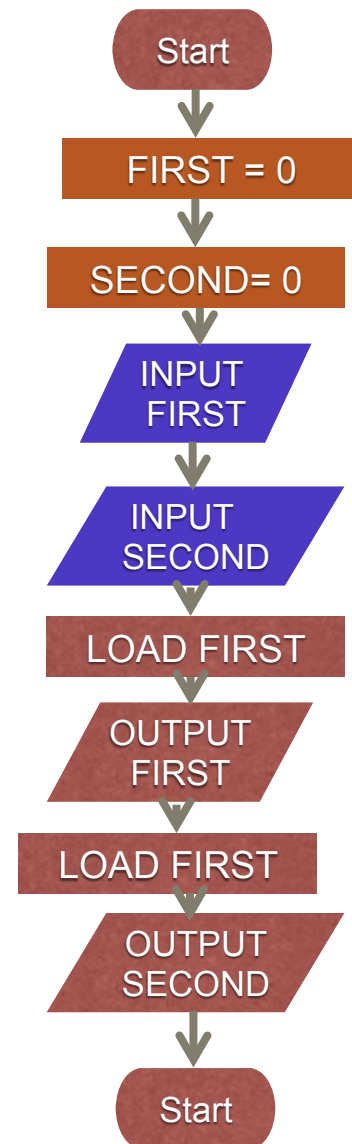


# Using Memory (Variable)

- Now start at the top and write down the commands for the instructions for the flow chart.
- The LMC command for INPUT is **INP**
- If we need to store that we need to follow this with a store command, **STA** and save it to memory using the DAT label we created

**INP**  
**STA FIRST**

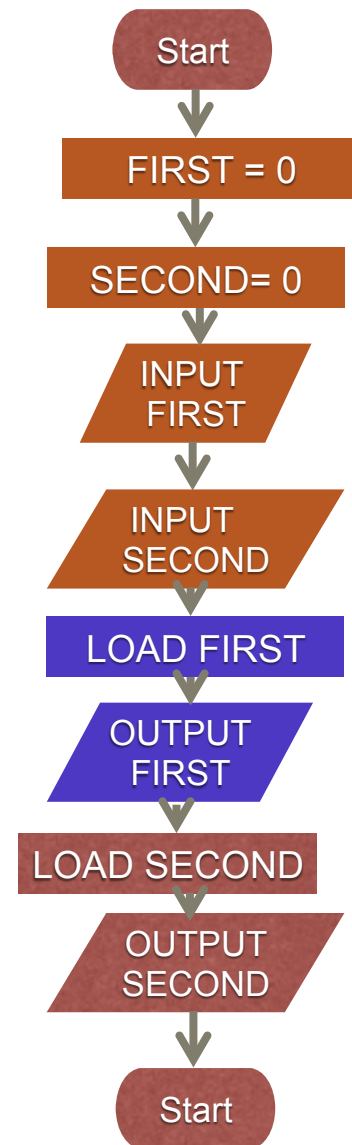
**INP**  
**STA SECOND**



# Using Memory (Variable)

- Now we use the **LDA** command to load the value stored in FIRST Variable.
- We use the **OUT** command to output the value stored in FIRST Variable.

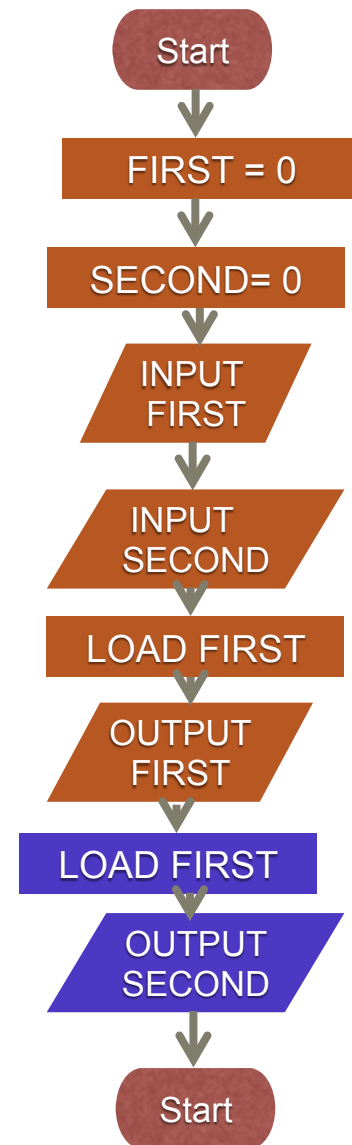
LDA FIRST  
OUT



# Using Memory (Variable)

- We do the same for the Second value
- We use the **LDA** command to load the value stored in SECOND Variable.
- We use the **OUT** command to output the value stored in SECOND Variable.

**LDA SECOND**  
**OUT**



# Using Memory (Variable)

Once we Output both FIRST and SECOND Value,

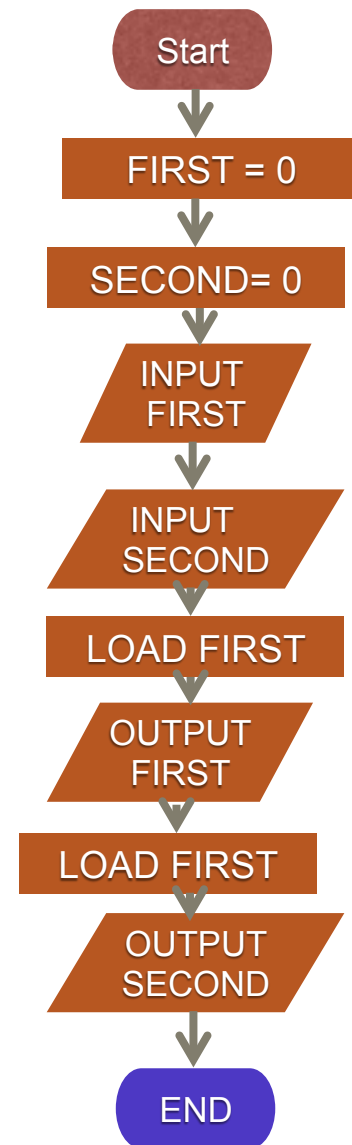
We use **HLT** to end the program

```
INP  
STA FIRST
```

```
INP  
STA SECOND
```

```
LDA FIRST  
OUT
```

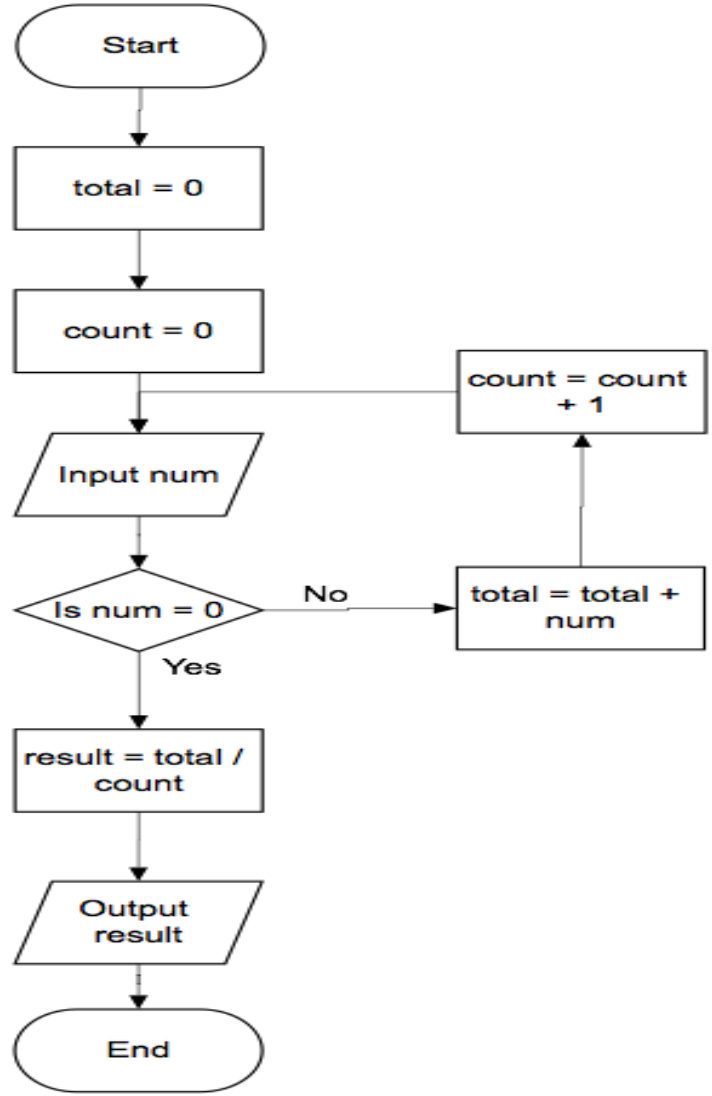
```
LDA SECOND  
OUT
```



**Task**

- Try this code on LMC
- Re-create the flowchart
- Describe what you think the program does.

```
LOOPTOP INP
STA num
BRZ CALCULATE
LDA total
ADD num
STA total
LDA count
ADD one
STA count
BRA LOOPTOP
CALCULATE LDA total
SUB count
STA total
BRP DIVIDE
LDA RESULT
OUT
HLT
DIVIDE LDA result
ADD one
STA result
BRA CALCULATE
total DAT 0
count DAT 0
num DAT
result DAT 0
one DAT 1
```



# Activity – Little Man Computer

- Open the “**System Architecture Booklet**”
- Complete the activity in the **Little Man Computer Challenges** section

# STARTER – 5 MINUTES

Describe how does the following characteristics of the CPU affect performance of the computer system

- 1. Clock Speed**
- 2. Number of Cores**
- 3. Cache**
- 4. Bus**



# **CHARACTERISTIC OF CPU VS PERFORMANCE**

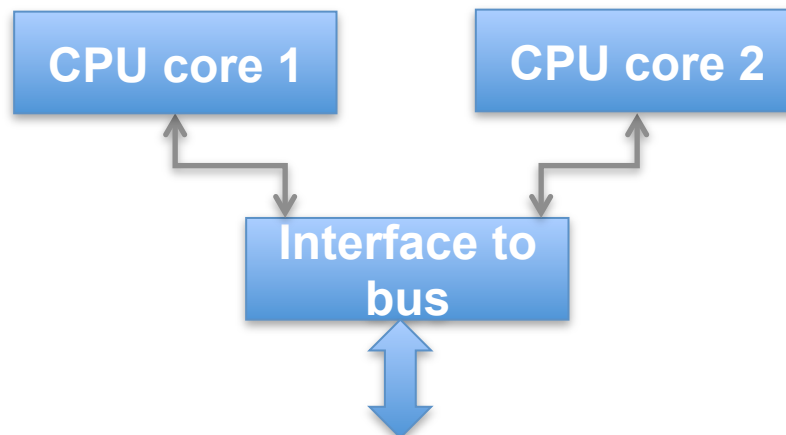
# CPU: CLOCK SPEED

## What characteristics affect the performance of the CPU?

- The **clock speed** – how quickly the CPU processes instructions per second:
  - The clock chip uses a vibrating crystal that maintains a constant rate and all processes are synchronised to this clock signal.
  - The clock speed is measured in **Hertz(Hz)** or cycles per second. A clock speed of 600 Hz would be 600 cycles per second.
  - **MHz** - means million of cycles per second, so a clock speed of **3MHz** would mean **3 million** cycles per second
  - **GHz** - means billions of cycles per second, so a clock speed of **3GHz** would mean **3 billion** cycles per second.

# CPU: NUMBER OF CORES

- The number of cores – how many instructions it can process at a time:
- A multiple core processor has more than one CPU
  - In a dual core CPU, two CPUs work together
  - In a quad core CPU, four CPUs works together
- Since each core can fetch, decode and execute instructions at the same time, the computer is able to process two or instructions at once.



# CPU: BUS

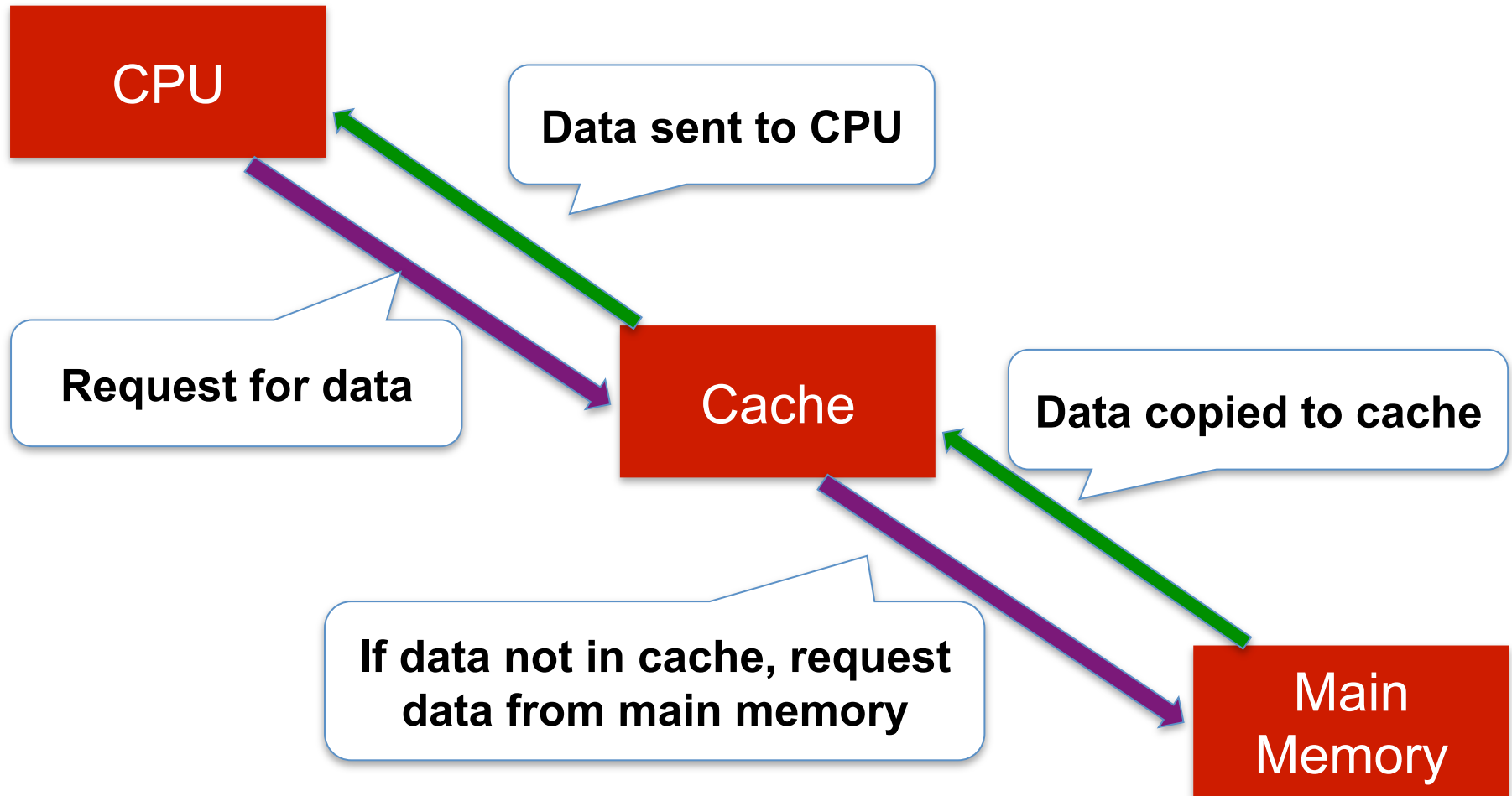
- **Data is moved around the computer on buses.**
  - Bus speed affects how quickly the computer can move data, and therefore has an effect on the speed of the computer.
- A bus is simply a circuit that connects one part of the motherboard to another.
- The speed of the bus, measured in **MHz** (millions of cycles per second), refers to how much data can move across the bus at the same time.

# CPU: CACHE MEMORY

- **Cache memory** is very fast memory that can work at speed similar to the CPU, but it is very expensive and only used to store data waiting to be processed by the CPU.
- Cache memory is usually provided in much smaller sizes than the main memory, **megabytes, MB as opposed to gigabytes, GB** (1GB = 1024 MB)
- Cache memory is located very close to the CPU to minimise access times the CPU fetch instruction from RAM.
- A mid-range laptop might have 8GB of RAM but only 2 or 3 MB of cache memory.
- ***The larger the cache, the faster the computer system***

# CPU: ROLE OF THE CACHE

The role of cache in data transfer:



# CPU: CORE I5

A typical Intel CORE i5 processor

- Intel i5-3210M
  - 2.5 GHz clock speed (2.5 billion operations per second)
  - 2 cores
  - 5MB cache memory



# EXAM PRACTICE

**In a short while you will carry out some exam practice.**

**But first...**

**...Some exam technique tips.**



# COMMAND WORDS

What do each of these words mean?

- State/Identify/Give/Name
- Describe
- *Describe/Explain/Discuss* using examples
- Explain
- Discuss
  
- ...you need to know because they appear in all exams.
- If (for example) the examiner asks you to **EXPLAIN** and you instead just **DESCRIBE**, you will not get any marks.

# COMMAND WORDS

## State/Identify/Give/Name

Simply label a diagram, fill out a table or write a few words

## Describe

Describing is 'saying what you see'

*E.G.: A computer will have a CPU, Primary and Secondary storage etc*

## Explain

Explaining is 'saying why something is like that'

*E.G.: A computer will have a CPU so that it can process all of the data the computer needs to perform a range of tasks. Primary and Secondary storage is needed because...*

# COMMAND WORDS

## Discuss

Discussing is 'looking at two sides of an issue, weighing up the two views and giving a conclusion'. Often these require a mini essay answer.

*E.G.: New technology could be seen as being bad for the environment because..., but on the other hand, new technology has lead to... In conclusion I believe that...*

*Describe/Explain/Discuss using examples*

*Finally, if you are asked to give examples in any of these types of questions – **YOU MUST GIVE EXAMPLES!***

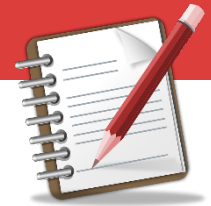
# REMEMBER

- Whenever you answer exam questions you must CaM it!
- Look at the...
  - Command words AND Marks
- Do what the question asks and make sure your answers have enough points or explanations to get the marks available.



5 minutes

# ACTIVITY 2



7

(a) The table below contains statements about the functions of the CPU.

Tick **one** box in each row to show whether the statement is true or false.

	TRUE	FALSE
It performs arithmetic operations on data.		
It fetches and executes instructions		
Input and output devices are plugged into it		
It moves data to and from memory locations.		

[4]

(b) Some CPUs have cache memory.

(i) Describe what is meant by cache memory.

.....

.....

.....

.....

(ii) Explain why cache memory is needed.

.....

.....

.....

.....

[4]

**Complete the  
timed exam  
practice  
questions.**

# EXTENSION

Explain in as much detail as you can –

What effects the speed of a CPU?

Focus on the following ONLY

Cache

Bus speed

Cores

Clock speed

These are the main ones you need for the exam. The others I have included for completeness.

Use the **“The CPU-Student Note”** to help you

# **EMBEDDED SYSTEM**

# LEARNING INTENTIONS

- **Know:** What makes up a computer system
- **Understand:** What is embedded system
- **Be able to:** Define a embedded system and give examples

---

## Keywords

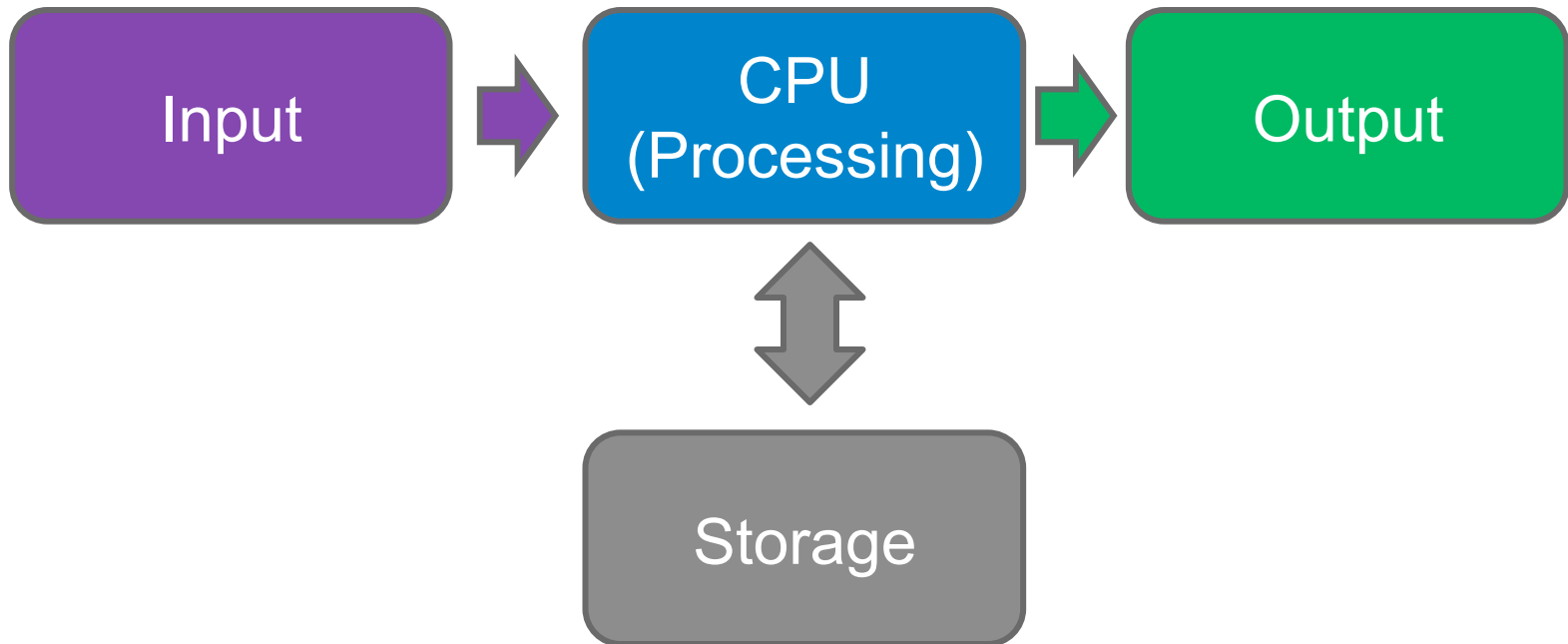
**Embedded**                      **Input**                      **Output**

**Computer System**                      **CPU**                      **Storage**

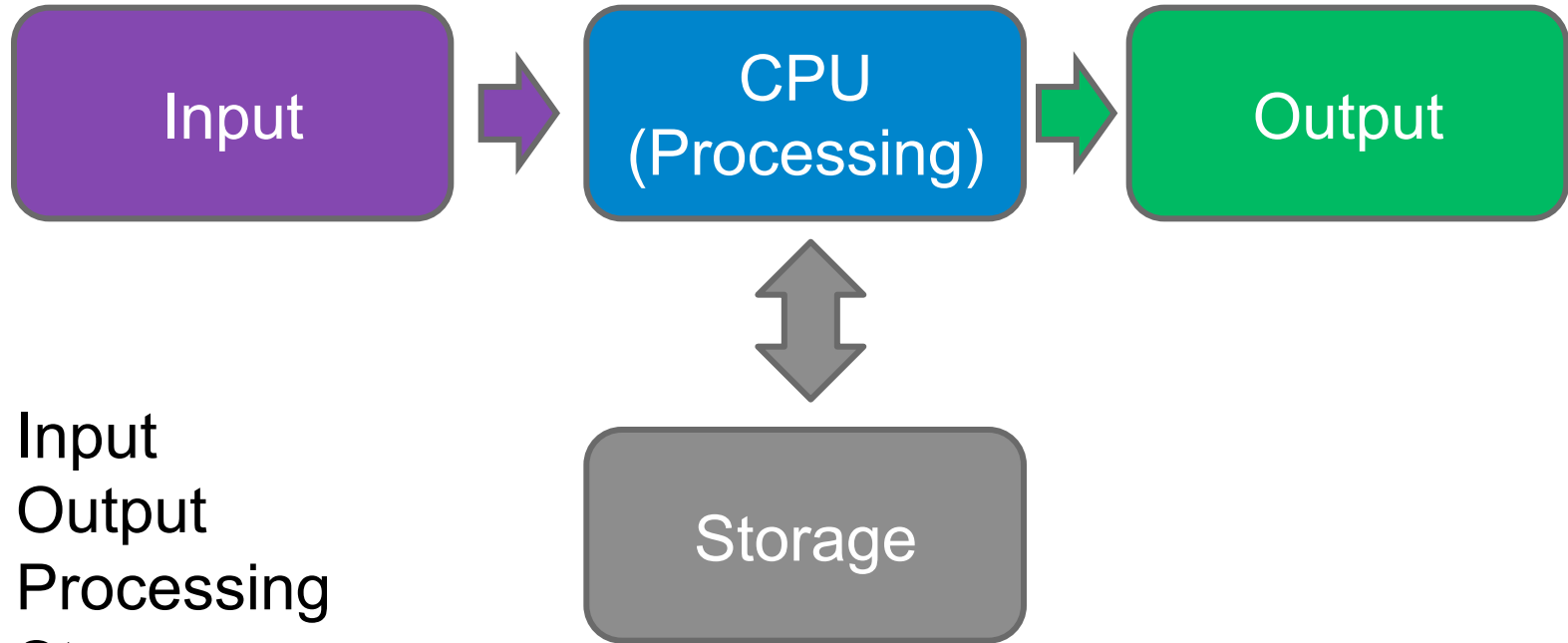


# RE-CAP

- A computer system can be broken down into four sections: **Input**, **Processing**, **Output** and **Storage**.
- A computer needs **devices** to input, process, output and store data.



# Activity



- Input
  - Output
  - Processing
  - Storage
  - Communication
- 
- Task – List three hardware for each of the devices in a PowerPoint

# TYPES OF INPUT DEVICES - MANUAL

- Keyboard
- Mouse
- Touchpad
- Joystick
- Touch screen
- Concept keyboard
- Scanner
- Graphics tablet
- Microphone
- Digital camera

# TYPES OF INPUT DEVICES - AUTOMATIC

- Barcode readers
- OMR (Optical Mark Reader)
- Magnetic Ink Character Recognition (MICR)
- Optical Character Reader (OCR)
- Magnetic stripe reader
- Sensors
- Biometric devices

# TYPES OF OUTPUT DEVICES

- Monitor
- Printer
- Plotter
- Projector
- Speaker
- Headphones
- Light/LED

# PROCESSING DEVICES

**CPU – Central Processing Unit** - Watch 30 second clip



# STORAGE DEVICES

- Hard disks
- DVDs
- CDs
- Magnetic tape
- Flash memory (USB)
- Solid State Drive (SSD)
- Micro SD Card

# COMMUNICATION DEVICES

- NIC (Network Interface Card)
- Wi-Fi cards
- Router
- Modem



# GENERAL SYSTEMS VS SPECIFIC

## **General computer systems –**

- These are systems which we are used to.
  - play games,
  - do work,
  - chat with friends or
  - surf the web!

## **Specific (or dedicated) computer systems –**

- These are systems which will only perform a single task.
- For example a airplane computer system or a shops till.

# COMPUTER SYSTEMS

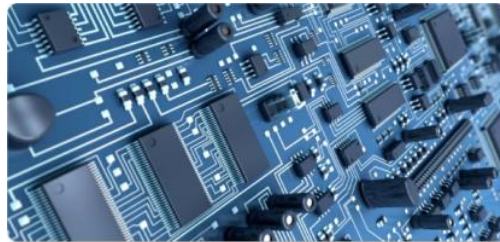
- Computer systems are based on processing data and producing information.
- They are fast, and the important thing about them is that they are programmable.
- Computer systems are found in most electronic gadgets.
  - For example, washing machines, cameras, burglar alarms and telephones
  - These are embedded system

# EMBEDDED SYSTEMS

- For your exam, you will also need to understand what an embedded system is.
- When we think of a computer, you usually think of a PC and as many of you know, a PC is made up of various components including a motherboard, CPU, RAM, input devices etc.
- But of course a computer is any programmable machine...or any electronic device which takes in data, processes it and then outputs the result.
- Can you therefore think of any other examples of computers?

# EMBEDDED SYSTEMS

- So when you consider devices like cameras and watches, as these are programmable machines, they can also be called computers.
- The main difference is that these computers run specific tasks – they are not general purpose.
- Because of this, they do not need to have separate components as these devices won't need updating when new software / hardware is released.
- These systems instead have all of their components arranged together on a single circuit board.



Source: <http://neuronelab.unisa.it>

- As a result they are known as embedded systems as all of their hardware is embedded together as one.

# EMBEDDED SYSTEMS

**Computer systems used in electronic gadgets have all of the basic functionality that drives a desktop PC.**

- There are input and output devices, storage, a processor and, most importantly, software this is an embedded system.



# EMBEDDED SYSTEMS (WASHING MACHINE)

## INPUT

- This washing machine has various input devices: Buttons, Sensors including weight, temperature, water-level & Door sensor

## PROCESS

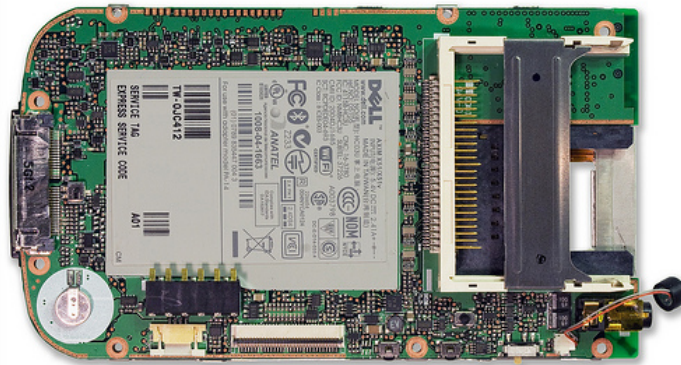
- It uses the data from these inputs to calculate the water temperature, steps to follow, time to complete the program etc.

## OUTPUT

- And it outputs information on a display and by using 'beeps' played through a speaker

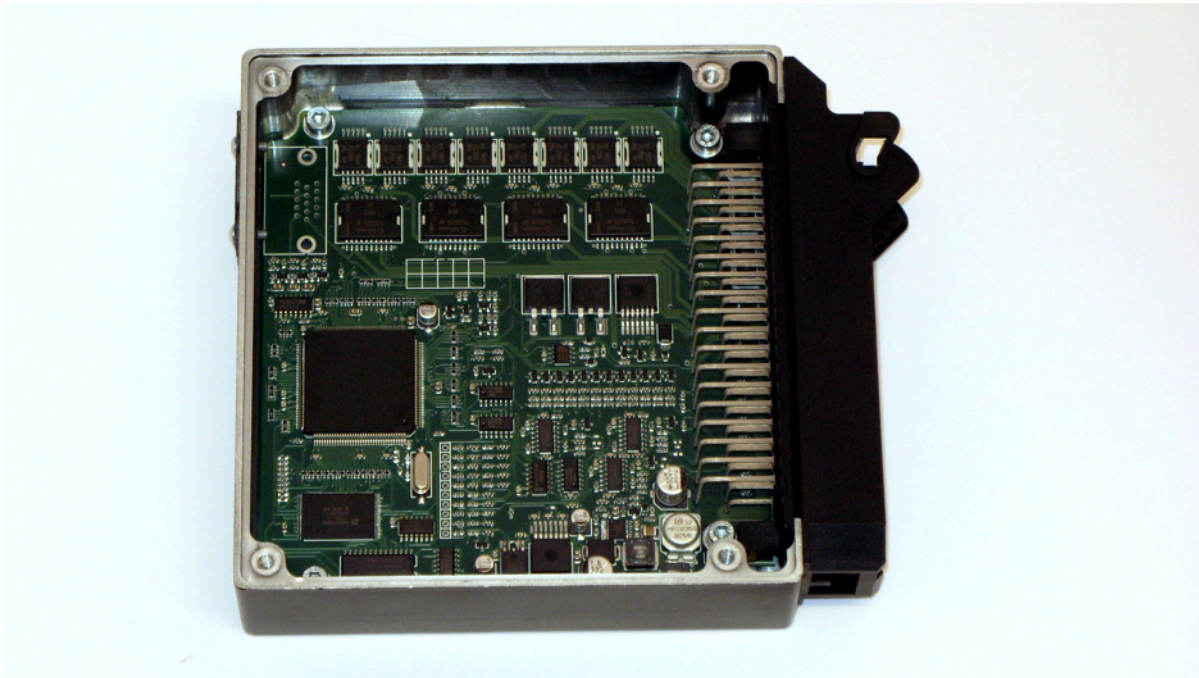
# EMBEDDED SYSTEMS

- Software that is programmed to carry out a number of dedicated functions.
- For example, the software to run a washing machine is stored on a computer chip and embedded into the system.



# EMBEDDED SYSTEMS

- Control systems can be quite complex, for example an engine management system.





# EMBEDDED SYSTEMS

- A typical engine management system in a car has over 50 processors, which explains why car engine faults can be difficult to trace without the right equipment.
- The embedded systems in a car look after various safety features.
- It is easy to see why these must be reliable and thoroughly tested,
  - both as individual items and as integrated systems.

**Use The “Embedded System” PDF file to answer the following questions;**

- 1.** What is Embedded System?
- 2.** List down 4 different types of Embedded System
- 3.** List and Describe 4 Characteristics of Embedded System
- 4.** List down the Skills Needed for Embedded Applications

# CONTENT

There are several Generations of Programming Languages

## –Low Level

- 1<sup>st</sup> Generation (Machine Code e.g. 1010101)
- 2<sup>nd</sup> Generation (Assembly Code e.g. INP OUT HLT)

## –High Level

- 3<sup>rd</sup> Generation (e.g. Python, Java)
- 4<sup>th</sup> Generation (e.g. MySQL)

# 1<sup>ST</sup> GENERATION

Machine Code (Example Binary - 101010101011)

Directly Executable by the processor

The Generation that “computers understand”

Difficult to program in, hard to understand, hard to find errors (hard to debug)

# 2<sup>ND</sup> GENERATION

- **Assembly Code**
- **Uses mnemonics**
  - Example LMC codes (INP, OUT ADD, BRP, BRZ etc)
- **Easier to program in than 2<sup>nd</sup> Generation but still difficult.**
- **One Assembly Language instruction translates to one Machine Code Instruction (1-1 relationship)**

# 2<sup>ND</sup> GENERATION

- **Needs to be translated into Machine Code for the computer to be able to execute it**
- **Uses an Assembler**
- **Assembler – Assembles Assembly Language**

# 2<sup>ND</sup> GENERATION

- **Despite 2<sup>nd</sup> Generation being difficult to Debug it still has its uses**
  - **Debug** means to go through the code to search for where/why an error has occurred
- **It is most commonly used to program Device Drivers**
  - **Device Drivers** are loaded into memory by the Operating System and used to control the operation of a Hardware Device e.g. Graphics Card Drivers, Printer Drivers.

# 3<sup>RD</sup> GENERATION

- Easier to understand (programmer)
- Easier to find errors, easier to de-bug
- Uses English-Like Keywords
  - Example, `print ("Hello World")`
- One instruction translates into many machine code instructions
- E.g.'s Python, Java, Basic, Pascal, C+, Visual Basic



# 2<sup>ND</sup> VS 3<sup>RD</sup> GENERATION

## 2<sup>nd</sup> Generation

```
x  INP
    STA x
    OUT
    HLT
    DAT
```

## 3<sup>rd</sup> Generation

```
x = int(input("Enter a number: "))
print(x)
```

- Both code allow the user to enter a number and display the number
- **3<sup>rd</sup> generation** has **2 lines** while
- **2<sup>nd</sup> generation** translate to **5 lines**

# 2<sup>ND</sup> VS 3<sup>RD</sup> GENERATION

## 2<sup>nd</sup> Generation

```
    INP
    STA FIRST
    INP
    STA SECOND
    LDA FIRST
    ADD SECOND
    STA ANSWER
    LDA ANSWER
    OUT
    HLT
FIRST  DAT 0
SECOND DAT 0
ANSWER DAT 0
```

## 3<sup>rd</sup> Generation

```
first = int(input("Please enter a number"))
second = int(input("Please enter a number"))
answer = first + second
print (answer)
```

- Both code allow the user to enter two numbers
- These numbers are then add together and display the result
- **3<sup>rd</sup> generation** has **4 lines** while
- **2<sup>nd</sup> generation** translate to **13 lines**

# 3<sup>RD</sup> GENERATION

## Translated using:

- Interpreter
- Compiler

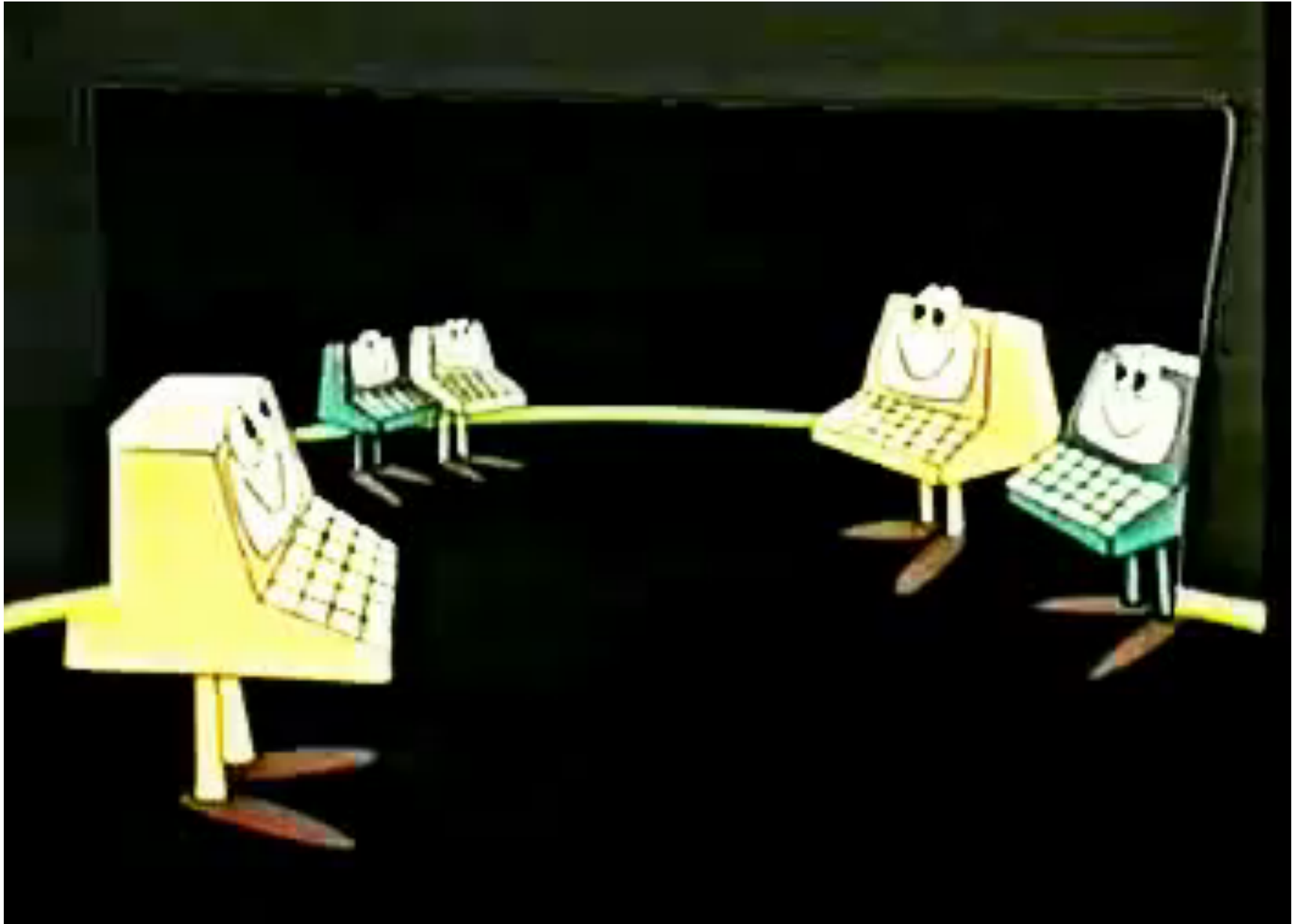
### Interpreters

- Translate and execute source code
- Line by Line, Statement by Statement
- Source code is checked for syntax – if correct, code is executed. If incorrect interpreting is stopped.
- Used for development (aide debugging)

### Compilers

- Translate entire source code all in one go into Machine Code
- Optimise code
- Used at the end of development (ready for shipping)
- Error Reports created along with Object Code

# COMPILER VS INTERPRETER



# 4<sup>TH</sup> GENERATION

- **Known as a Declarative Language**
- **Facts and Rules are stated**
- **It describes what computation should be performed and not how to perform it**
- **Examples include:**
  - SQL
  - Expert Systems
  - Artificial Intelligence

# ACTIVITY – IDENTIFY THE GENERATION

```
Dim Num1, Num2, Tot as Integer
Num1 = Console.ReadLine()
Num2 = Console.ReadLine()
Tot = Num1 + Num2
Console.WriteLine("Total is: " & Tot)
```

**High Level (3<sup>rd</sup> Generation)**

```
0101010101010010101010010101
0101010011111001000100001010
100101
```

**Low Level (2<sup>nd</sup> Generation)**

```
LOAD r1, c
LOAD r2, d
ADD r1, r2
DIV r1, #2
```

**Low Level  
(1<sup>st</sup> Generation – Machine Code)**

# ACTIVITY - ANSWER

```
Dim Num1, Num2, Tot as Integer  
Num1 = Console.ReadLine()  
Num2 = Console.ReadLine()  
Tot = Num1 + Num2  
Console.WriteLine("Total is: " & Tot)
```

High Level (3<sup>rd</sup> Generation)

```
010101010101001010101001  
010101010100111110010001  
00001010100101
```

Low Level (2<sup>nd</sup> Generation)

```
LOAD r1, c  
LOAD r2, d  
ADD r1, r2  
DIV r1, #2
```

Low Level  
(1<sup>st</sup> Generation – Machine Code)

# Why Use Low-Level Language?

- They allow direct access to the processors registers.
- This means very fast processing for big data sets.
- Hardware drivers and embedded systems (eg the coding that goes into a washing machine) are written to directly access the hardware for those devices.
- Does not mean that a compiler is required which makes for more much faster and efficient use of processor time.



# Assembly Language Vs Machine Code

## Assembly Language

- A language related closely to the computer being programmed/low level language/machine specific
- Uses mnemonics for instructions
- Uses descriptive names for data stores (symbolic addressing)
- Translated by an assembler into machine code
- Easier to write than machine code but more difficult than high level language

## Machine code

- Written in Binary (or hexadecimal)
- Actual binary code run by the machine
- Set of all instructions available to the architecture which depend on the hardware design of the processor
- No translation needed
- Very difficult to write

# Points to note

- The instruction set is very limited so you often need to come up with a different way to perform things like multiplication or comparing two numbers.
- The LMC does not store decimals.
- The LMC does not have a loop structure but you can use a Branch Always command to redirect the code to an earlier command.
- Start with simple program
- Create a flowchart for the program before going into LMC
- Enjoy!