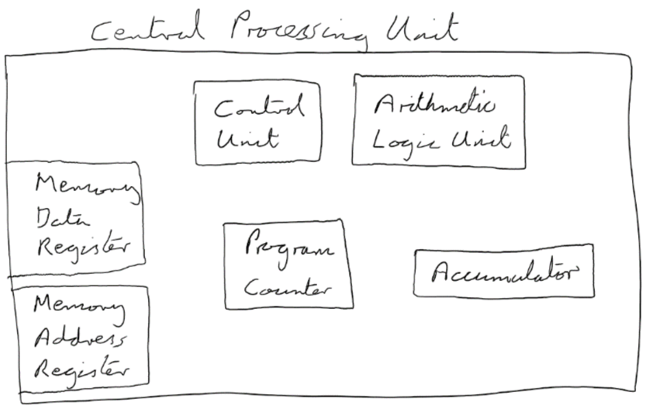
Year 9 Computer Science

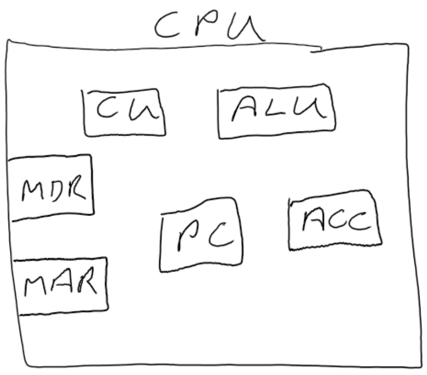
All the topics below can be found in your One Note Computer Science notebook.

If you have any problem accessing your One Note Computer Science note, email me (dmay@okehamptoncollege.devon.sch.uk). In case you do not have access to your online notebook, the notes are also reproduced on the following pages.

1. CPU Components

The **Central Processing Unit (CPU)** is what makes a computer a computer! It is the hardware that reads and executes instructions to perform every type of task, from running the operating system to letting the user play a game.





We usually reduce the names of the components to their abbreviations.

This is what the components do (you need to learn these descriptions):

Control Unit (CU)	The **CU** is responsible for: • Fetching, decoding and executing instructions; • telling the computer's memory, **ALU** and I/O devices how to respond to the instructions that have been sent to the processor.
Arithmetic Logic Unit (ALU)	The **ALU** is that part of the **CPU** that handles all the calculations the CPU may need. It is used by the **CU** when a logical operation needs to be executed.
Accumulator (ACC)	The **ACC** stores temporary data during the execution of instructions by the **ALU**. Although it is often shown as a single register, it is really a series of registers. It's like the 'memory' buttons on a calculator.
Program Counter (PC)	The **PC** stores the *address* of the *next* instruction to be fetched from RAM. N.B. It never stores DATA; it only ever stores an ADDRESS.
Memory Address Register (MAR)	The **MAR** stores the *address* in RAM where instructions and/or data is to be read from or written to. N.B. It never stores DATA; it only ever stores an ADDRESS.
Memory Data Register (MDR)	The **MDR** stores instructions and/or data that has been read from or written to RAM.

Interesting extra information...

The ACC, PC, MAR and MDR are all types of register. You sometimes hear a CPU being described as 64-bit or 128-bit, etc. This is referring to the number of bits in each register. In other words, it's the number of bits that can be processed by the ALU *at the same time*.

2. CPU Architecture

A bit of (interesting) history...

75 years ago a scientist called John Von Neumann came up with an idea that would change the future of computers. Up to that point programs were hard-wired into computers using electronic switches - in other words, a computer could only run the one program it was designed for. If you wanted another program, you had to redesign the computer by changing all the switches!

Von Neuman thought it would be more useful if programs could somehow be stored along with the data, so that a computer could run different programs at different times. He came up with his own computer design which he called the **stored program concept**. He said a computer must have:

1. A single area of memory (the **RAM**) for programs *and* data;
2. A **bus** to transfer programs and data between the RAM and the CPU;
3. **Registers** inside the CPU to hold the programs and data as they are being processed;
4. A **control unit (CU)** inside the CPU to decode the program instructions;
5. An **arithmetic logic unit (ALU)** inside the CPU to execute the logical operations.

A computer must also have **external storage** to store programs and data when the computer is switched off, as well as a way of controlling all the other parts of the computer using **inputs** and **outputs**.

Von Neumann's idea was adopted by all the computer manufacturers and today almost every computer in the world uses the **stored program concept**, whether it's your smartphone, your microwave or your laptop. It made John Von Neumann so famous it became widely known as **Von Neumann architecture**.

Part of Von Neumann's design was the **fetch-decode-execute cycle**. Now that you have learnt the component parts of the CPU, we can describe the fetch-decode-execute cycle.

This is the level of detail you need to be able to remember for your GCSE.

FETCH	The memory address of the *next* instruction to be fetched is copied from the **program counter (PC)** to the **memory address register (MAR)**; The instruction (or data) is FETCHED from **RAM**, using the memory address in the **MAR**, and is stored in the **memory data register (MDR)**; The **PC** is changed (incremented) to the address of the *next* instruction;
DECODE	The instruction in the **MDR** is DECODED by the **control unit (CU)**.
EXECUTE	The decoded instruction is EXECUTED by the **CU** using the **arithmetic logic unit (ALU)** for any logical operations; During the EXECUTE stage of the cycle the **accumulator (ACC)** may be used to store temporary data.

Interesting extra information...

Primary memory, **primary storage** and **main memory** are all the same thing!

They are referring to RAM and ROM which is the fast memory inside a computer used by the CPU.

The CPU can only read data and instructions from RAM and ROM: if you want to run a game or open a document, it *must* be loaded into RAM first.

3. Cache Memory

There is a problem with the fetch-decode-execute cycle. The problem is that if the same instruction is used over and over again, it has to be fetched every time from RAM. Sometimes the RAM can't give the CPU the instructions and data fast enough, so the CPU has to wait until it has the data. Slowing down the CPU like this is called a **bottleneck**. This never used to be a problem with older computers because the CPUs were slower than the RAM.

To get around the bottleneck, modern CPUs contain **cache memory**, which is very small, super-fast memory inside the CPU which holds instructions that are frequently used. So when one of these instructions is called, the fetch-decode-execute cycle only has to go to the cache instead of getting what it needs from RAM.

To make things a little more complicated, there are actually three levels of cache in a computer system:

- **Level 1 (L1)** is inside the CPU, next to the CU and ALU
  - Very, very fast
  - Very, very small (between 8 KB and 64 KB)
  - Very, very expensive to make

- **Level 2 (L2)** is on a separate chip to the CPU
  - Very fast
  - Larger than L1 (256 KB to 8 MB)
  - If the CPU doesn't find the instruction in L1 it looks in L2.

- **Level 3 (L3)** is on the motherboard
  - Not as fast as L1 and L2 (but still much faster than RAM)
  - Larger than L1 and L2 (4 MB to 50 MB)
  - If the CPU doesn't find the instruction in L1 or L2 it looks in L3.

Interesting extra information...

It is easy to get confused between RAM and cache memory. The thing to remember is that a computer can work without any cache memory; it *cannot* work without RAM. The cache memory simply helps frequently-used data get to the CPU faster.

4. Understanding binary

To convert the 8-bit number 1010 1000 from binary to denary, complete the following steps:

a. Put the decimal values 128, 64, 32, 16, 8, 4, 2, 1 into a table...

Denary	128	64	32	16	8	4	2	1

b. Next, copy the binary number into the row below...

Denary	**128**	64	**32**	16	**8**	4	2	1
Binary	**1**	0	**1**	0	**1**	0	0	0

c. Add all the denary values that have a corresponding binary value of 1 to get your result.

128 + 32 + 8 = 168

d. So, the binary number 1010 1000 is 168 in denary.

Converting denary numbers to binary: Convert **73** to binary.

1. Draw the table.

128	64	32	16	8	4	2	1
0	1	0	0	1	0	0	1

2. Find the **biggest** number in the table which is the same as, or smaller than, 73.
3. Take 64 away from 73.
4. Repeat.
    1. Find the **biggest** number in the table which is the same as, or smaller than, 9.
    2. Take 8 away from 9.
    3. Repeat.
        1. Find the **biggest** number in the table which is the same as, or smaller than, 1.
        2. Take 1 away from 1.
        3. Nothing left to do!
5. Fill in the gaps with zeros.

Adding two binary numbers together.

- Four rules to learn:

$$\begin{array}{r} 0\ + \\ 0 \\ \hline 0 \\ \hline \end{array} \qquad \begin{array}{r} 1\ + \\ 0 \\ \hline 1 \\ \hline \hline \end{array} \qquad \begin{array}{r} 1\ + \\ 1 \\ \hline 0 \\ \hline \hline \text{carry } 1 \end{array} \qquad \begin{array}{r} 1\ + \\ 1 \\ \hline 1 \\ \hline \hline \text{carry } 1 \quad 1 \end{array}$$

carried

5. Understanding hexadecimal

Hexadecimal

- Hexadecimal (or hex) is a number system which uses base 16
- As we only have 10 digits, hex uses 0-9 and then letters A to F

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**

Hex to decimal conversion

- You will only need to translate one- or two-digit hexadecimal numbers

16s		Units	
2		**A**	
2x16	+	10	= 42 in decimal

- Multiply the left-hand digit by 16, then add the units

Binary to hex conversion

- Take a binary word of 8 bits

1 1 1 0 0 1 0 1

- Divide into two nibbles of 4 bits

1 1 1 0 **0 1 0 1**

- Convert each nibble into its hex value and rejoin

1 1 1 0 = 14 = E in Hex + **0 1 0 1 = 5** in Hex

So **1 1 1 0 0 1 0 1 = E5** in Hex

Decimal to hex conversion

- Divide the decimal number by 16 to get the number of 16s (the left-hand hex digit)
- The remainder gives you the units

Decimal **18** becomes:

18 / 16 = **1** r **2** so the hex value for **18** is **12**

(Spoken, 'One Two', not 'Twelve')

Hex to binary conversion

- What is **3B** in hex?

Split the two hex characters

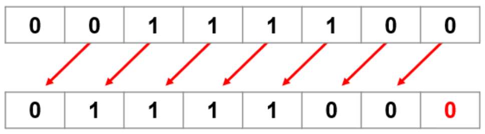
3 = 0011 in binary and **B = 1011**

So **3B = 0011 1011** in binary

6. Binary shift

Logical binary shift operations

- A binary shift left of one bit moves all the bits one place to the left
    - New spaces are filled with zeros



- Left binary shift moves all bits one place to the left and DOUBLES the number.
- Right binary shift moves all bits one place to the right and HALVES the number.

N.B. If you need a way to remember which binary shift does what, think of the alphabet...

J
K
LEFT BINARY SHIFT
MULTIPLIES BY 2