Components in CPUs

Component Name	Description
Memory/RAM	Random Access Memory which computers use to store running systems software and applications software.
ALU (Arithmetic Logic Unit)	Executes instructions by performing mathematical and logical operations.
Bus	A bidirectional or unidirectional channel of communication between components.
Register	Stores a tiny amount of data next to the CPU.
Clock Speed	Measure of CPU clock cycles (instructions per second) in MHz or GHz. If a clock speed is too high, instructions won't complete before the next starts. Clock speed can be increased in a process known as *overclocking*, however the CPU could get too hot.
Core	A logical unit of processing with its own ALU and registers.
Cache	A type of storage which is very close to the CPU and stores a small amount of data.

Registers

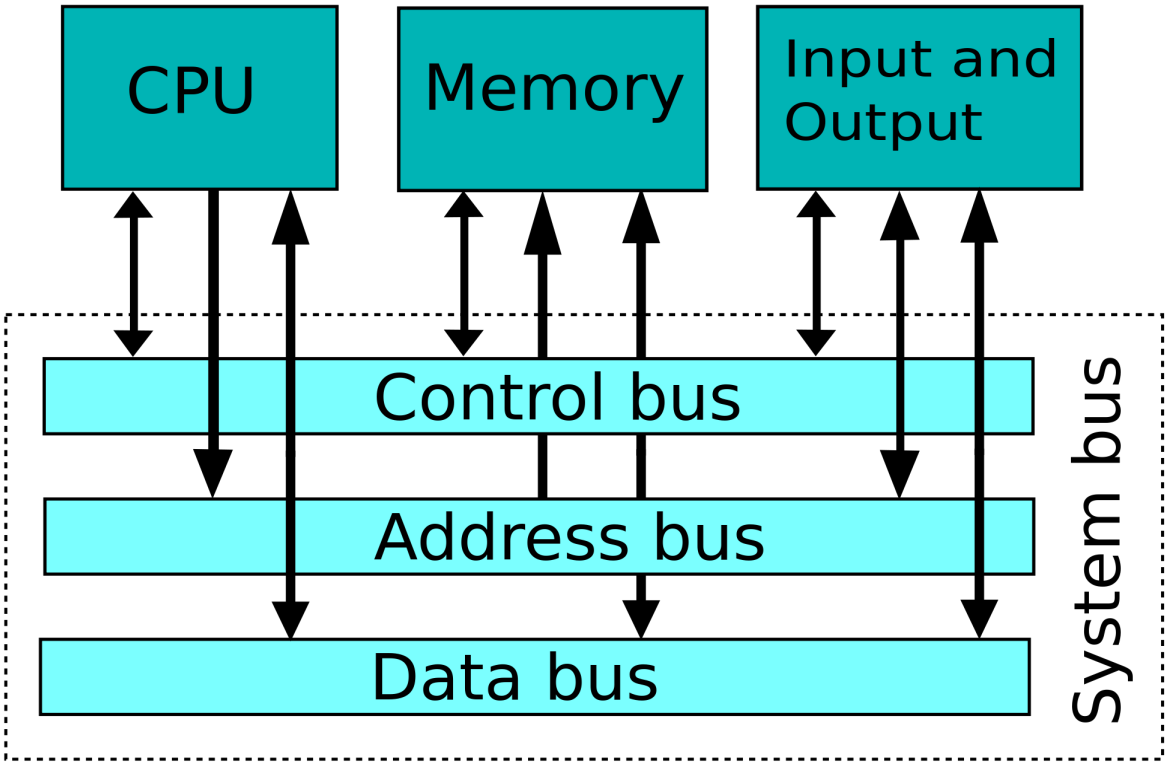
Store a tiny amount of data next to the CPU. Unlike RAM or cache, registers aren't designed to store large blocks of memory instead, only used for keeping data for CPU operations.

Register Name	Description
PC (Program Counter)	Contains the address in memory of the current instruction in the running program at the time. Increments after an instruction has finished executing.
ACC (Accumulator)	Register where results from arithmetic and logical calculations are stored after they are calculated.
MAR (Memory Address Register)	CPU register that stores the memory address from which data will be fetched from RAM to the CPU.
MDR (Memory Data Register)	Register that stores data that's being moved into or just moved out of RAM.
CIR (Current Instruction Register)	Stores the current instruction which is executing or being executed. When the instruction is finished, the program counter

	increments and the next instruction is loaded from RAM over the data bus.

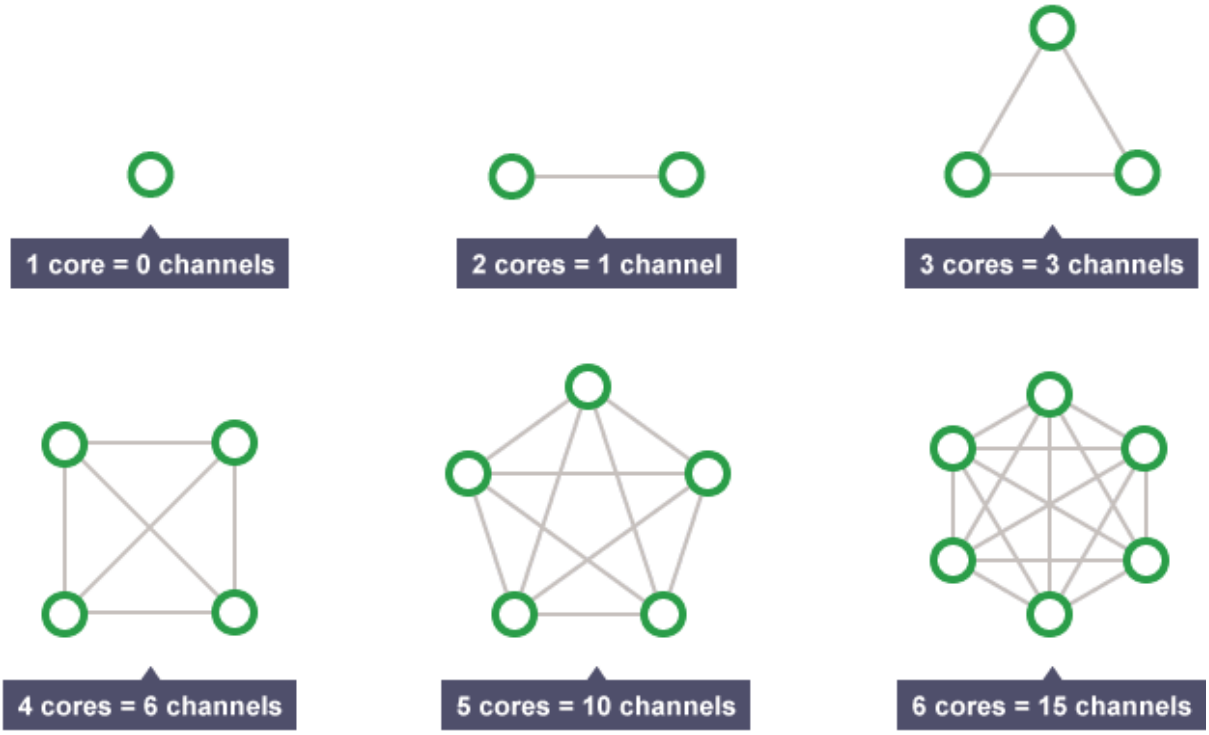
Communicating Between Components in a Computer

A *bus* is a channel of communication between components such as the CPU, GPU or RAM. This can be for the CPU to retrieve data from RAM, or could be the CPU controlling the GPU or a hard drive.



Bus Name	Description
Data Bus	The bus where data flows between RAM and CPU.
Address Bus	Sends addresses from the CPU to RAM.
Control Bus	Sends instructions and data to and from devices allowing them to communicate and be controlled.

Channels are a simple way of communicating between CPU cores. This allows each core to communicate and work together. In a multi-core CPU, each core has a channel to every other CPU, so in a dual core CPU there is 1 channel, but in a quad core CPU, there are 5 channels.



CISC vs RISC

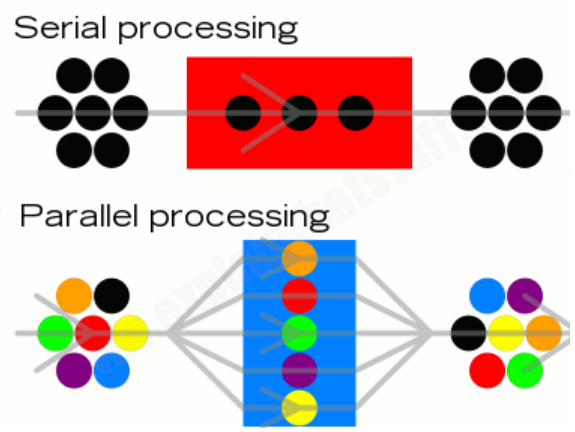
CPUs use instruction sets to allow programs to control them, do computational operations and manipulate data in RAM.

Complex Instruction Set Computing	**Reduced Instruction Set Computing**
Usually used in desktop computer and high-end machines (however modern CPUs convert CISC into RISC at runtime, as they are RISC underneath).	Used in laptops, computers, mobile phones and portable devices.
More complex set of instructions.	Smaller (reduced) set of instructions and thus simpler.
More complex hardware.	Less complex hardware.
Instructions can take multiple cycles to finish execution.	Instructions take only one cycle to execute.
Physically larger in size and requires more silicon, thus more expensive.	Smaller in size as less complex and cheaper.
Can use pipelining to optimise instructions.	No pipelining.
More intensive tasks will do better with CISC.	Runs at a lower clock speed, but can perform simpler tasks more quickly.
Uses a lot more power.	Uses less power (ideal for mobile). Can go

	into a power-saving mode.
Higher clock speed.	Lower clock speed, however can run much lower if needing to save power.
Smaller binaries.	Larger binaries.
Examples: x86 and variants	Examples: MIPS, PowerPC, ARM

Cores, Threading and Parallel Processing

By processing data in parallel, large operations can be finished much faster. For example when data mining Twitter, each core can be used to process four separate streams of tweets. Four tweets can be processed at the same time rather than just one which means efficiency is increased.



Threads are an abstraction on top of cores. Programs can simply create threads and the operating system can schedule hundreds of threads onto separate CPUs. It can also balance usage so each core is used to it's maximum potential.

Some Intel CPUs use hyper-threading which means that each core has two threads and the CPU simply balances these out.

Programs have to be specifically developed to specifically use each core. Because of the complexities of writing code to work in a multithreaded environment (such as locking, globals and cross-core communication), only programs that require high performance, such as Adobe Photoshop or games use multiple threads.

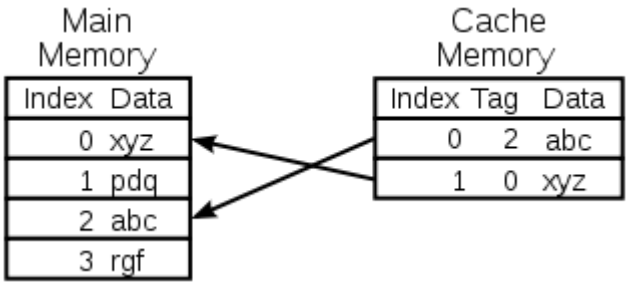
I/O such as writing to disk or network operations, so it's common to create another thread specifically for accessing data or downloading an image and then a program can do other operations while waiting for the operation to complete. This is known as a callback in software engineering.

How Data Flows in a CPU

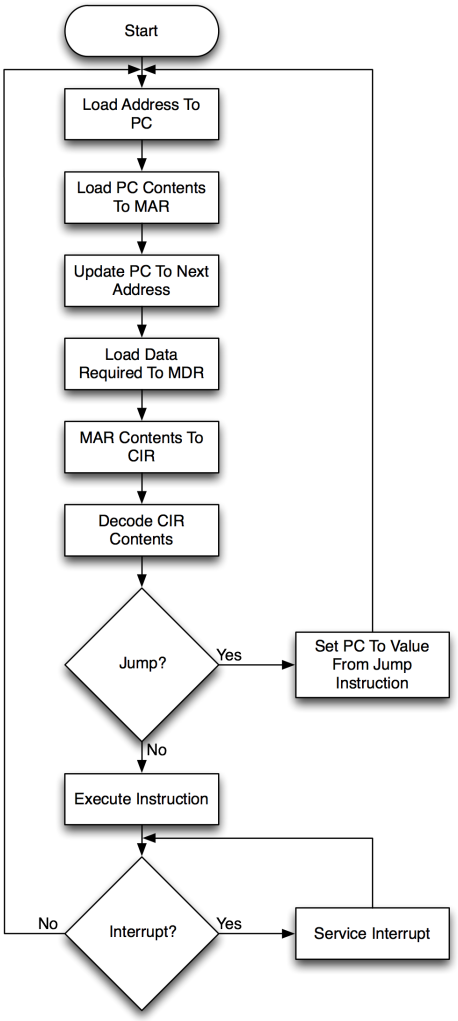
Fetch, decode execute is the process that happens on every CPU cycle. An instruction is fetched from RAM and then a component decodes the code into a logical operation which can be executed by the ALU.

When data is retrieved from memory, the CPU first checks if the data is contained in L1 cache and continues onto L2 and L3 cache on cache miss. If data isn't stored in cache, the

CPU will retrieve it directly from RAM. Frequently accessed or ‘hot’ data will be stored in CPU cache with colder data staying in RAM.



When a CPU begins executing a program, the compiled program blob is loaded into memory. The address of the first byte of the program is then loaded into the PC over the data bus.



Fetch, Decode, Execute Cycle

On each cycle the address of the next byte of the program is loaded into the PC. The PC’s contents is loaded into the MAR which is sent to RAM over the address bus. The RAM then returns the next instruction to the MDR over the data bus. The MAR’s contents are then loaded into the CIR where the instructions are decoded into a logical or mathematical operation. During this whole process the control unit orchestrates the moving of data around the various registers and busses.

At the start of the execution phase, if the instruction is to jump to another line, this is immediately loaded into the PC and the cycle starts again. Otherwise the instruction is executed and it’s contents are loaded into the ACC or sent back to RAM via the MDR, MAR and busses.

Programming with Assembly

The lowest level representation of a program. ASM is a direct one-to-one representation of machine code, but in human readable form. Higher level languages such as C or C++ compile down to assembly. ASM programs are architecture specific such as CISC and RISC and can only run on CPUs with the instruction set they were programmed for.

```
100
101
102
103
104
105
106
107 00000030 B9FFFFFFFF
108
109
110 00000035 41
111 00000036 803C0800
112
113
114 0000003A 75F9
115
116
117
118
119
120
121
122 0000003C C3
```

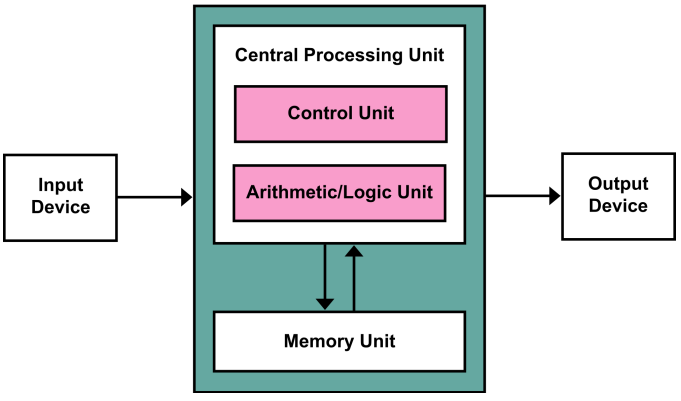
```
;-------------------------------------------------------
; zstr_count:
; Counts a zero-terminated ASCII string to determine its size
; in:   eax = start address of the zero terminated string
; out:  ecx = count = the length of the string

zstr_count:                 ; Entry point
    mov  ecx, -1            ; Init the loop counter, pre-decrement
                            ;  to compensate for the increment
.loop:
    inc  ecx                ; Add 1 to the loop counter
    cmp  byte [eax + ecx], 0  ; Compare the value at the string's
                            ;  [starting memory address Plus the
                            ;  loop offset], to zero
    jne  .loop              ; If the memory value is not zero,
                            ;  then jump to the label called '.loop',
                            ;  otherwise continue to the next line
.done:
                            ; We don't do a final increment,
                            ;  because even though the count is base 1,
                            ;  we do not include the zero terminator in the
                            ;  string's length
    ret                     ; Return to the calling program
```

Although you can write programs in Assembly, it's extremely rare outside embedded systems like microcontrollers where extreme performance is required. Programs written in some higher level languages, such as C (which is still low-level) are compiled. When compiling a program the output binary blob is architecture specific and has to be compiled separately for each platform, such as x86 and ARM.

Platforms such as Java compile programs to bytecode, which runs in a JVM (Java Virtual Machine). JVM bytecode is like x86 and ARM machine code however it's Java specific (Java being an architecture unto itself). This means that any compiled Java program can run on any machine with a JVM.

Von Neumann, Harvard and Contemporary Architectures

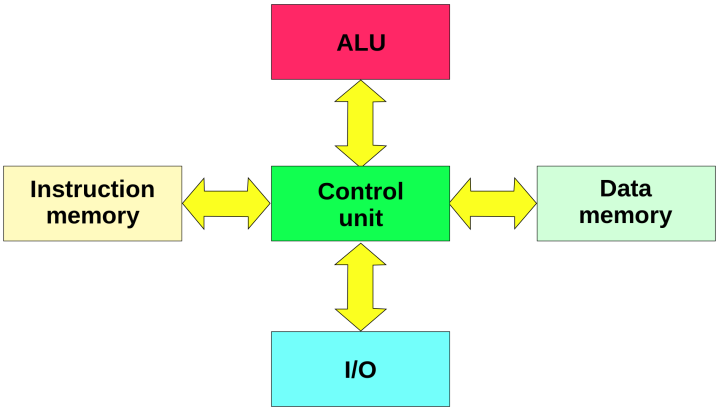


The *Von Neumann Architecture*, created by John Neumann is a simple architecture which has single control unit can sequentially work through a stream of instructions.

There is a single data bus and single storage unit, where both instructions and data are stored.

The *Von Neumann Bottleneck* is an issue posed by the architecture due to instructions and data not being able to sent and received in parallel as a single data bus can only send data sequentially in one way at a time.

The *Harvard Architecture* is an improvement on the Von Neumann Architecture due to having two data busses and two memory units, one for data and one for instructions, meaning instructions can be fetched whilst data is being sent or received from storage.



Contemporary Architecture uses a unified model of the Harvard architecture, with multiple memory units as instruction and data caches, but a unified address.

All architectures support multiple cores that are all linked using data busses, which connect all CPU caches, however these architectures can also work with just a single core.