Lecture 1-Computer Architectures

Modern computers follow the Von Neumann architecture with *fetch, execute, store* control flow. More specficially, the computer contains

- A processing unit with both an arithmetic logic unit and processor registers
- A control unit that includes an instruction register and a program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

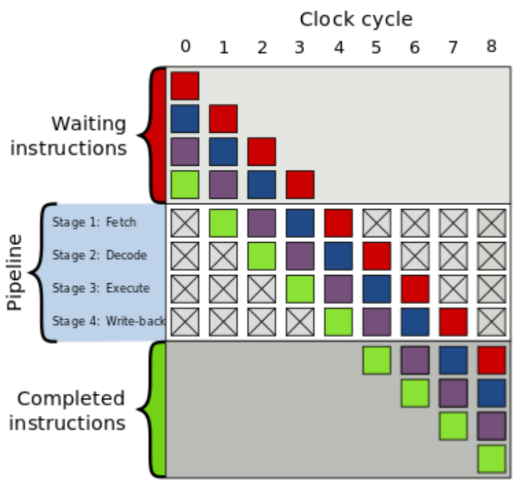
The control unit manages the four basic operations as follows:

- Fetch: gets the next program command from the computer's memory
- Decode: deciphers what the program is telling the computer to do.
- Execute: carries out the requested action
- Store: saves the result to a register or memory

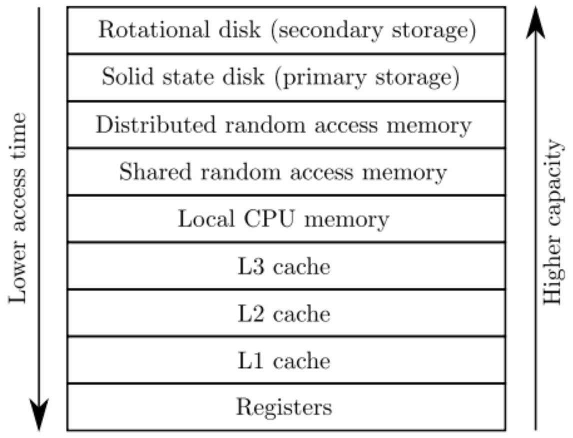
The arithmetic/logic unit will perform logic/mathematical operations. The processor also comes with an internal clock that dictates when instructions start. The register is a type of memory that is very fast and very close to the cpu but very limited in space. It is used to save intermediate calculations.

Instruction Level Parallelism: is the approach in which CPUs operate on multiple instructions simultaneously, which are said to be 'in flight'.

- Multiple Floating Point Units (supporting parallel execution)
- Fused Multiply-Add: x $\leftarrow$ ax + b in same time as addition or multiplication.
- Multiple-issue: instructions that are independent can be started at the same time.
- Pipelining: arithmetic units can deal with multiple operations in various stages of completion.
- Branch prediction and speculative execution: A compiler can "guess" whether a conditional instruction will evaluate to true, and exectue those instructions accordingly.
- Out-of-order execution: instructions can be rearranged if they are not dependent on each other, and if the resulting execution will be more efficient.
- Prefetching: data can be speculatively requested before any instruction needing it is actually encountered.



Memory Aspects


Rotational disk (secondary storage)
Solid state disk (primary storage)
Distributed random access memory
Shared random access memory
Local CPU memory
L3 cache
L2 cache
L1 cache
Registers
Lower access time
Higher capacity

There are two quantities to check:

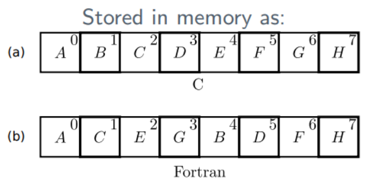
- Latency: Delay from the request item in memory until arrival.
- Bandwidth: The rate at which data arrives at its destination, after the initial latency is overcome. Measured in bytes per second or per clock cycle.

Vectorial processing is when you use a single operation to add multiple FPs for example. It only works if the data you stored in memory is contiguous.

Memory layout

- Contiguous: Where elements are neighbours. Easy prefetching; efficient memory transfer especially for vector units.
- Noncontiguous: Where elements are fragmented in memory. Prefetching is challenging and less efficient wide memory bus utilization; less efficient transfer.

$$A = \begin{bmatrix} A & B \\ C & D \\ E & F \\ G & H \end{bmatrix} \in \mathbb{R}^{4\times 2}$$


Stored in memory as:
(a)
A 0 B 1 C 2 D 3 E 4 F 5 G 6 H 7
C
(b)
A 0 C 1 E 2 G 3 B 4 D 5 F 6 H 7
Fortran

If memory allows, it is often an advantage to store both versions.

Cache Keep often used data close for easy retrieval. When the CPU needs some specific data it will first check the L1, L2, L3 cache to see if it was recently used. If the data is not found it's a 'miss'. There are three types of misses

- Compolsary: On first access. Unavoidable
- Capacity: Too much data that is often used filling the cache. Solution is to partition the problem into chunks that can be contained in cache.
- Conflict: Multiple items are mapped to same cache location and hence overwritten.

Something about cache lines and mapping

Algorithmic intensity

Refers to the number