# Models of Computer Architecture

CSN-221: Reading Assignment 1

Shreyas Dodamani
19114079

# Contents

The von-Neumann Architecture 2

Pure Harvard Architecture 4

Von-Neumann and Harvard Architecture comparison 5

Modified Harvard Architecture 6

Architecture of Parallel Computing 8

Quantum Computing Architecture 11

The Pipelining Technique 13

Systolic Architecture 14

Superscalar Architecture 16

References and Image Sources 18

# The von-Neumann Architecture

This model consists of:

1. The Central Processing Unit (CPU)
2. The Main Memory Unit
3. The Input/Output Device

**The Central Processing Unit (CPU) -** The CPU is the electronic circuit responsible for executing the instructions of a computer program. It consists of:

- **Control Unit**:
  This component handles the flow of all data (instructions and information). This includes fetching data and instructions from the memory with the given addresses. In short, it controls the movement of data throughout the system.
- **Arithmetic and Logic Unit:**
  This part of the CPU handles calculations like addition, subtraction, comparisons, etc.

**Memory -** In the Von Neumann model, instructions and data are both stored in the **same** memory space, and are read and written using the same signal pathways.

Apart from the permanent storage, data is also stored in places called **registers** for short periods of time. Types of registers are:

1. Accumulator: To store calculations made by the ALU
2. Program Counter: Stores the memory location of the next instruction (which is then passed to the memory address register for the subsequent step.
3. Memory Address Register: Stores memory locations of the instruction that needs to be fetched for execution.
4. Current Instruction Register: Stores the most recently fetched instruction.

5. Instruction Buffer Register: Used to store the instruction that will be executed at a later point in time.

**Buses -** Just like buses on the roads are used to transport people from one place to another, buses in the Von Neumann architecture are used to transmit data from one location in the computer to another. Depending on the type of information to be transmitted, there are different types of buses:

1. **Data Bus:** Carries data between memory unit, I/O modules, and the processor.
2. **Address Bus:** It carries the address of the data in the memory unit between memory and processor.
3. **Control Bus:** This bus carries the commands from the CPU that directs the flow data in the computer.

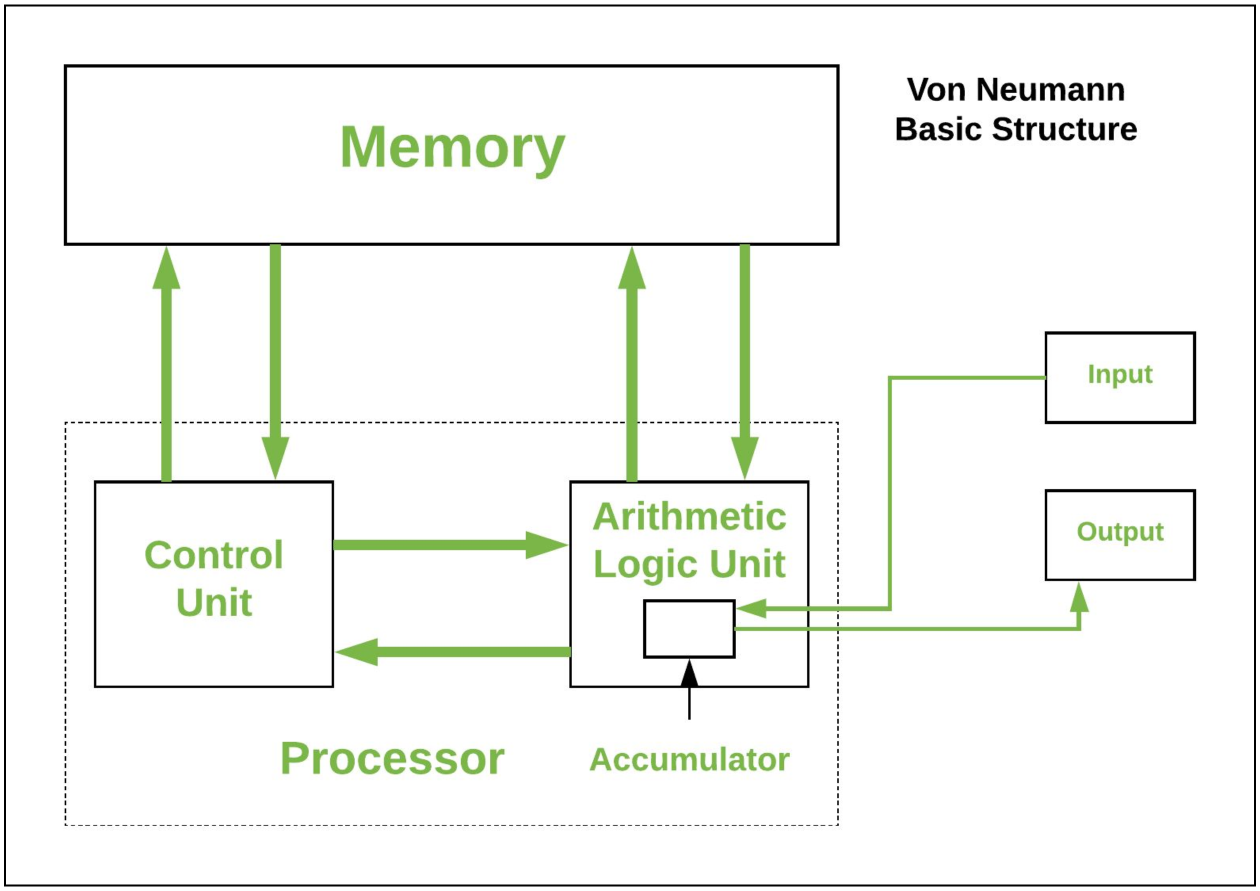

Figure 1: Von Neumann architecture

## Pure Harvard Architecture

Unlike the Von Neumann Architecture - which stores instructions and data in the same memory components, the program instructions and data use the same pathways in the Pure Harvard Architecture. This means that the CPU can fetch data and instructions at the same time.

The instructions and data are not only retrieved separately and independently, but are also stored in different address spaces. In other words, it won't suffice to know what the memory address is, it is also necessary to know which memory space to look in.

This architecture was first implemented in the *Harvard Mark I,* which was a relay based computer. It stored instructions on a 24-bit wide punched tape and data in electro-mechanical counters. The data storage resides entirely inside the CPU and has no means of direct communication with the instruction storage.

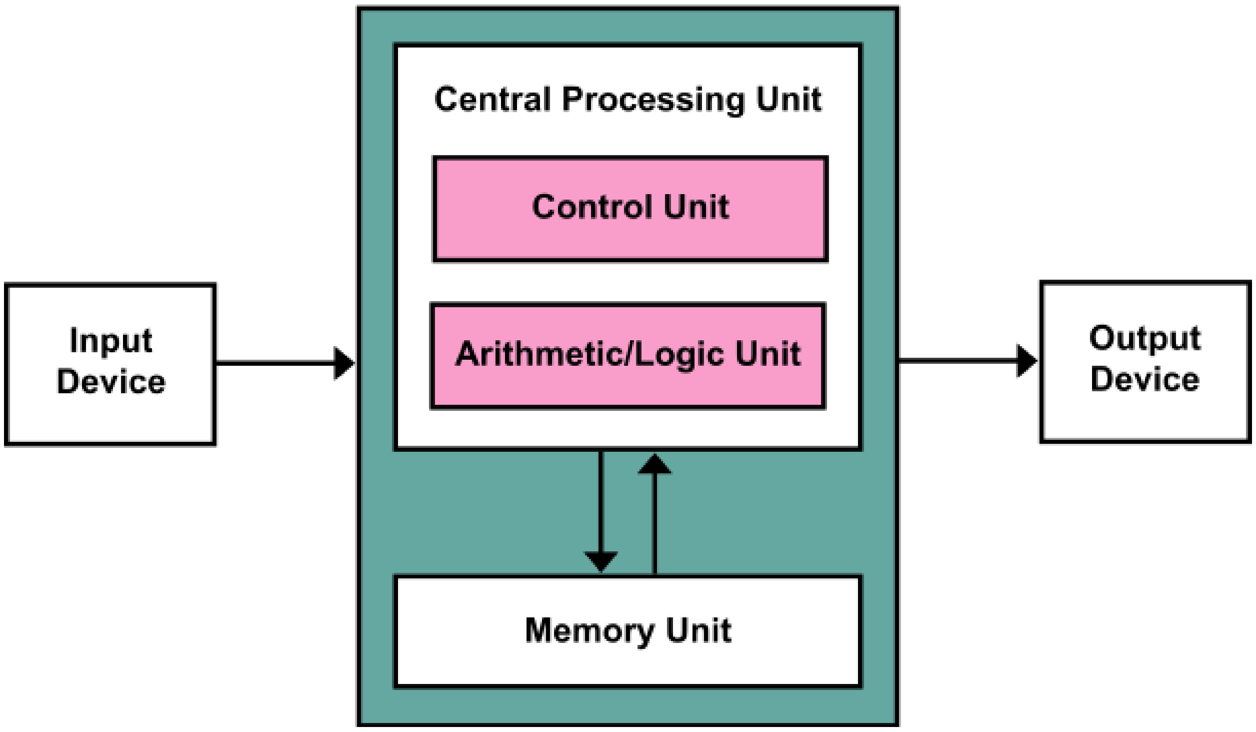

Figure 2: Harvard Architecture

## von-Neumann Architecture vs Harvard Architecture

| **Aspect of comparison** | **Harvard Architecture** | **Von Neumann Architecture** |
|---|---|---|
| Arrangement | The CPU is separately connected to the data memory and program memory. | A single memory component is used to store instructions and data both. |
| Hardware requirements | It requires more hardware because it requires separate data and address bus for each memory. | This requires less hardware since only a common memory space needs to be accessed. |
| Speed of execution | Faster than von neumann because the processor fetches data and instructions simultaneously. | Speed of execution is slower than Harvard Architecture because this model uses a single bus to access both instructions at the same time. Hence, only one of instructions and data can be retrieved at one time. |
| Space usage | Space is wasted because if there is space available in the data memory, the instruction memory cannot access it and vice versa. | Space is not wasted because there is a common memory space for data and instructions both. |
| Controlling | Controlling becomes complex because data and instructions are retrieved simultaneously. | Controlling is simpler than Harvard because only one piece of information is fetched at a time. |

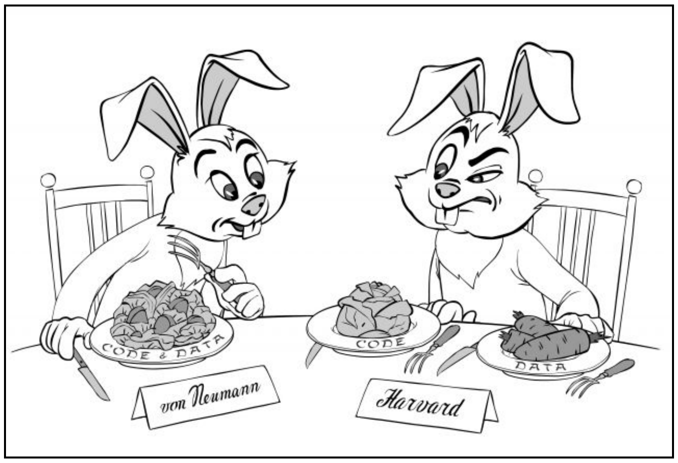

Figure 3: Comical Representation of von-Neumann vs. Harvard Architecture

# Modified Harvard Architecture

This architecture, a combination of both Von Neumann and Pure Harvard Architectures, is one of the most widely used architectures today. In fact, most modern computers that are registered as Harvard architecture, are actually Modern Harvard Architecture. It weaves the best parts of Von Neumann architecture (common data storage) and the Harvard architecture (separate buses for data and instructions) into one complete structure.

A modified Harvard architecture is a lot like its ancestor, the Harvard architecture, however, it blurs the distinction between instruction and data, while still allowing the central processing unit to concurrently access multiple memory buses.

## What exactly were the modifications made to the Harvard architecture that resulted in the Modified Harvard architecture?

- Split cache (a.k.a. almost-von-Neumann) architecture:

  This is the most common modification made to the Harvard Architecture, and it rectifies a major flaw in the von-Neumann architecture. In the bare von-Neumann model, every time the CPU needs information, it needs to make a call to the storage unit. Since this happens very often, the process is very expensive in terms of time and energy. To fix this, CPU caches are implemented for instructions and data. The information that is being used frequently is stored in a small memory storage inside the CPU.

  The CPU caches are separate for instructions and data (hence the name split-cache), however, the main storage unit for them is unified. The CPU always retrieves instructions and data independently from caches (in separate buses), and the caches pre load the information through a common bus connected to the main memory.

- Instruction-memory-as-data architecture:

  This modification provides special machine operations to read and write the contents of the instruction memory as data. Initial data values can be copied from the instruction memory to the data memory at the start of the program (before any instructions are run).

## Comparing Modern Harvard, Pure Harvard, and von Neumann architectures:

1. **Structure of instruction and data memory storage:**
   In Pure Harvard machines, there is a memory address 'X' in instruction memory space as well as the data memory space, and they are both different locations in the CPU which store different data. Contrary to the pure harvard machines, von Neumann and modified harvard machines store both instructions and data in a single address space, so, there is only one location corresponding to address 'X' in the entire computer, and it can store either instruction or data.

2. **Pathways for transporting instructions and data:**
   The modified Harvard shares qualities from both von-Neumann model and the pure Harvard model in this aspect. Like the von-Neumann model, the modified Harvard machine accesses the memory using a single pathway for instructions and data, and loads into the respective caches. The CPU in turn accesses the caches separately, as done in the pure Harvard architecture.

## Architecture of Parallel Computing

Simply put, parallel computing is a type of computing architecture in which multiple processors simultaneously execute small, and low level calculations (instructions) which are then combined to solve an overall larger and more complex problem.

Parallel computing systems can be classified under one of the following types:

1. **Single-instruction, multiple-data (SIMD) systems:**
   SIMD system architecture is used to apply the same instruction over multiple data sets. For example, if we are multiplying two matrices, then we need to multiply (same operation) multiple numbers (multiple data) to get the result.

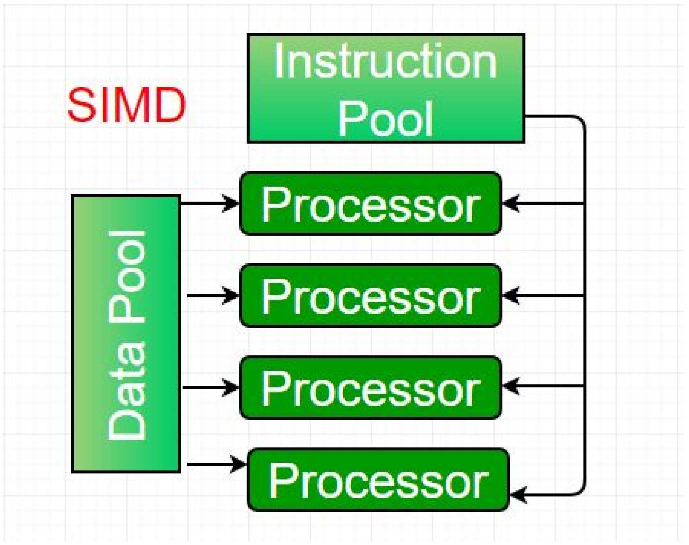

Figure 4

2. **Multiple-Instruction, single-data (MISD) systems:**
   MISD computing systems are used to carry out different operations on the same data. For example, if you are calculating:

$$Y = \sin(x) + \cos(x) + \tan(x)$$

In this case, there is a single piece of data (x), on which we are applying three different trigonometric functions and taking their sums. The tasks can be assigned to three different processing elements at the same time, instead of calculating them one by one.

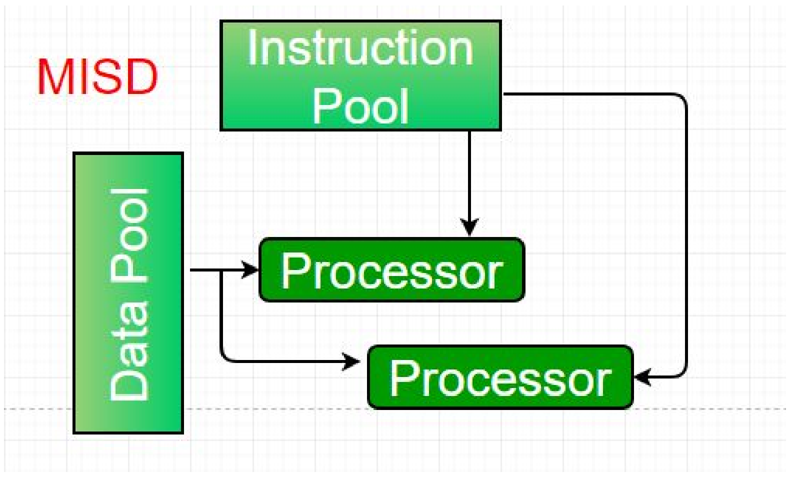

Figure 5

3. **Multiple-Instruction, multiple-data (MIMD) systems:**

   MIMD systems are used for doing different types of operations on different datasets at the same time.

   MISD and SIMD machines work synchronously. This means that once a task is assigned to multiple processors, new tasks are not assigned to any processor until all processors have completed the current assigned tasks. In contrast to MISD and SIMD machines, MIMD machines are asynchronous. Each processor is given a different instruction and dataset, so processors don't have to wait for each other to complete their tasks.

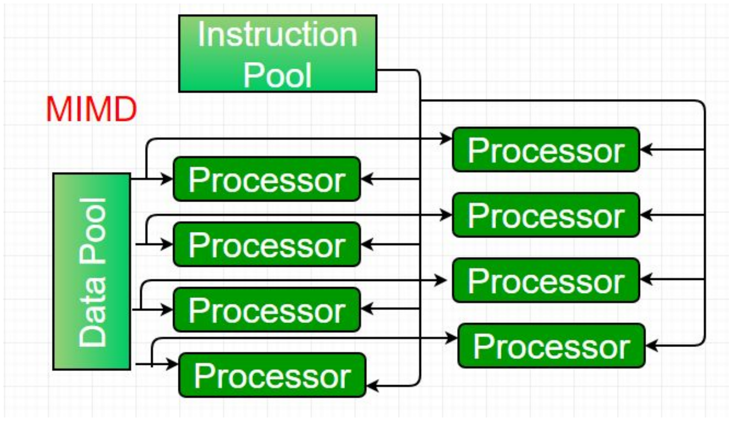

Figure 6

Based on the processors' connection to the memory, MIMD computer can be classified into:

a. **Shared Memory MIMD:**

   Also known as a tightly coupled multiprocessor system, in this model, all processors are connected to the same (global) memory unit, hence the name 'tightly coupled'. Any communication between the processors is also done through the same memory unit, and any modifications done by one processor to the memory unit can be seen by all other processors in the system.

b. **Distributed Memory MIMD:**

   Also known as a loosely coupled multiprocessor system, in this model, despite the existence of a global memory, each processor is connected to its own local memory. Communication between each processor takes place through the inter process communication channel or IPC.

## The Quantum Computer

Just like any computer, the quantum computer is also composed of multiple subsystems that are interconnected with each other. The quantum computer model is built on top of the von-Neumann model.

A quantum computer is made up of both classical and quantum parts. The programs are all classical (stored in classical components). The classical controller reads the classical instructions and feeds it into the quantum computer controller, which then runs quantum computing algorithms on quantum data. The quantum data is stored in *quantum registers*.

One way to understand this concept is to treat the **quantum component as a coprocessor to the classical system**. The classical component is essentially a von-neumann system. Instructions from this component are fed into the quantum component, which executes algorithms that the classical component would take ages to complete.

The basic model of the quantum computer architecture is:

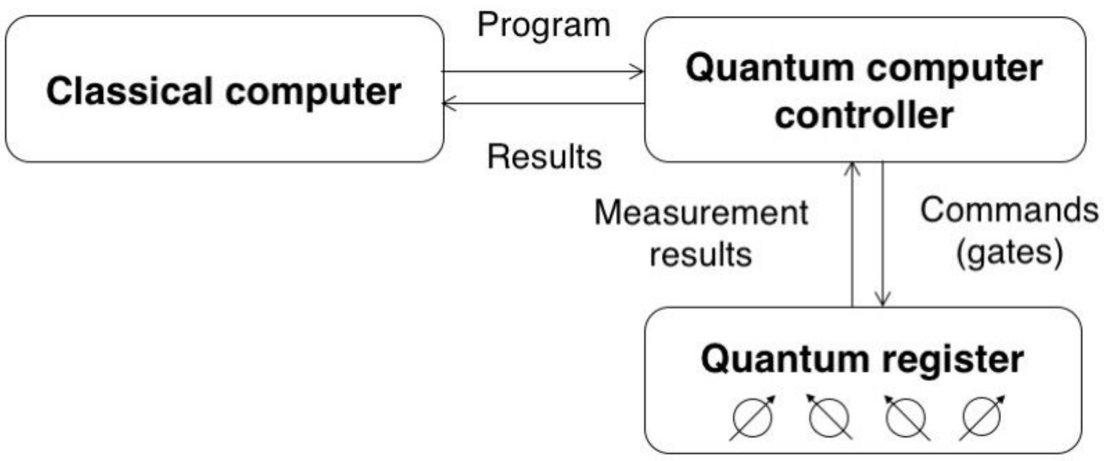

Figure 7: Quantum Computer Architecture

## Memory in the quantum computing architecture

The quantum computer stores temporary information in spaces called *quantum registers*. These registers can be divided into multiple sub-registers. In classical computer architectures, there are different types of memory, depending on the usage and type of data. Registers are used to store data that are going to be used almost immediately - like a buffer space. RAM (main memory) is used to store information that won't be required later on in time, but has no immediate requirement. Unlike classical computers, quantum computers do not have a clear distinction between registers and main memory, but can be considered similar to a classical computer's RAM.

## Pipelining

If you have heard of the assembly line, then you already know what pipeline computing is.

Pipelining is a *technique* - not a full-fledged architecture - where several instructions are overlapped during execution:

- Take a large problem
- Break it down into multiple steps (instructions)
- Start executing one instruction on a piece of data.
- As soon as the first instruction is finished, start executing the second instruction on the first piece of data, and bring in a new piece of data to execute the first instruction.

As can be seen from the diagram below, the simple process of fetching, decoding, executing and writing data takes 8 units of time, while, if they are pipelined, the time reduces to 5 clock units. (only 1 unit more than it would take to execute one instruction cycle).

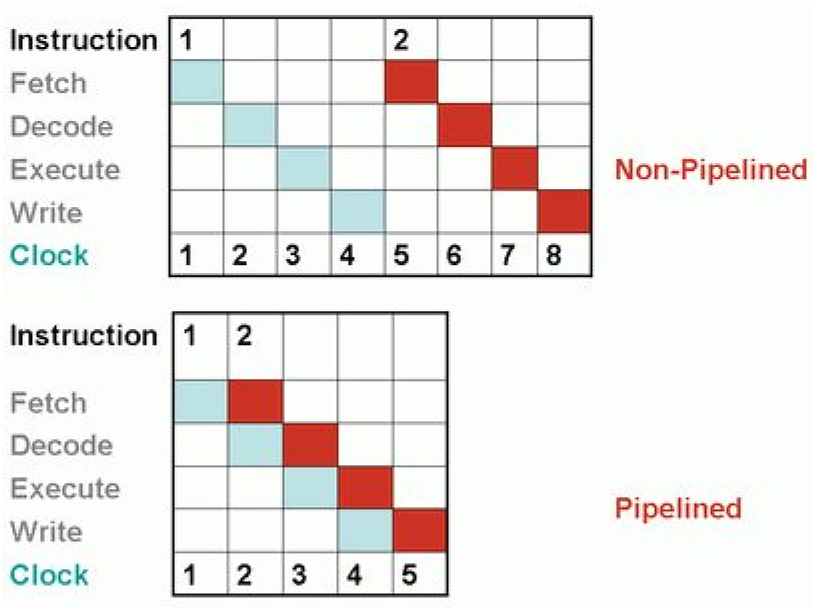

Figure 8: Example of how pipelining accelerates computation

## Systolic Architecture

In a systolic machine, data flows from the memory in a rhythmic fashion - technically called pipeline fashion - passing through many processing elements and finally returns to the memory. This architectural concept was developed at Carnegie Mellon University.

**The easy way to understand it:**

Take blood for example. It is pumped out from your heart, travels around the entire body while periodically releasing discrete amounts of oxygen to keep all the organs functioning, and returns back to the heart as all the oxygen is used up.

Similarly, in the systolic architecture, data exits the memory at the beginning of the cycle and goes through several processing units. The systolic system consists of multiple interconnected cells, each capable of performing some simple operation. Multiple, simple computations have a huge advantage over one large, complex computation, because it increases modularity.

**Problems faced by computer architects which the systolic model overcomes:**

- Cost:
  Cost effectiveness has always been a major concern while designing special-purpose architectures, because the cost must be low enough to justify their limited applicability. Cost can be classified into two types: non-recurring (design) and recurring (parts) costs. Although part costs are dropping due to advancements in integrated circuit technologies, this advantage applies to both special-purpose and general-purpose systems. It wouldn’t entice people to buy special-purpose systems of general-purpose. Hence, design cost must be relatively small to beat the competition of general-purpose architecture.

- Concurrency and Communication:

  There are essentially two ways to build a fast computation system. One is to use faster components, and the other is to improve the algorithms used for computation. Although the computer parts (especially transistors) have shrunk significantly in the past decades, they have almost reached the physical limit. The next best solution now is concurrency. Data is pipelined into the cells of the systolic system and churned out the other end back into the memory. The pipelined nature of the architecture allows you to make more calculations in a limited amount of time.

Computational tasks can be broadly classified into two groups. There's compute-bound problems, and I/O-bound problems. In a computation, if the total number of operations is larger than the total number of input/output elements, it is called compute-bound problems, else, an I/O-bound problem. Acceleration of I/O-bound computations can only be done by including more memory bandwidth. Speeding up a compute-bound calculation, can be accomplished by using a systolic approach. Unlike the I/O-bound problem, improving the speed for compute-bound calculations is simple and inexpensive.

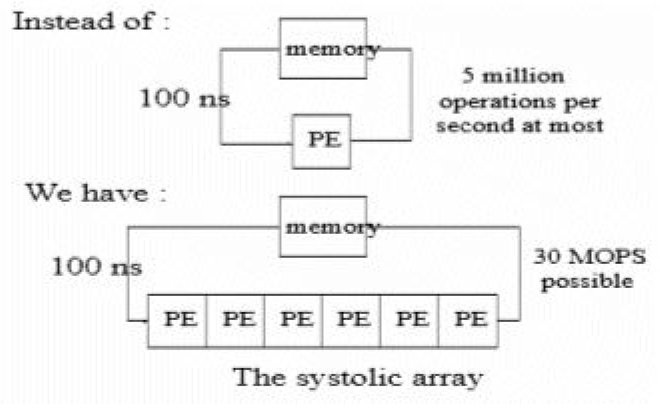

Figure 9: Systolic Architecture Representation

## Superscalar Architecture

The superscalar architecture is a combination of parallel processing and pipelining. In other words, it is a method of parallel computing with the use of multiple processors. The CPU executes a series of instructions concurrently in the same clock cycle.

**Let's understand this architecture using an example:**

Imagine a busy restaurant. Contrary to a single person at home, the restaurant receives multiple orders at the same time from several tables. Since everyone in the restaurant is already very hungry and impatient, everyone expects the service to be quick.

First, multiple waiters take orders from the tables and pass them on to the chefs. Inside the kitchen, the chefs prepare the dishes for each order, and once they're ready, send them back out with the waiters.

In this scenario, we can relate the waiters to the CPU's instruction fetching mechanism - to find out what food to prepare, or what instruction it is that needs to be executed. The different chefs can be seen as the pipelines. The greater the number of chefs in the kitchen, the more tables they can serve - more computations can be done in a shorter amount of time.

To picture the kinds of problems one might face while designing the superscalar architecture, we can again refer to the restaurant example:

- Suppose there are three chefs in the restaurant. Assuming one chef serves one table at a time, we can say that the three chefs can take orders for three tables. If one of the chefs burns his hand, then he has to stop working for a period of time. This is similar to a processor breaking down while in use. This would create a

**bottleneck** in the pipeline. Insead of taking three instructions at a time, the CPU can only handle two, which will slow down the process of computation.

- Another example would be if a waiter hands the orders to a chef who is already serving a table, instead of the chef that is free. Since the first chef is busy, he cannot handle the new orders, and the customers will have to wait longer. This is analogous to the instruction fetcher assigning the instruction to a processor that is already busy, and a bottleneck would appear once again. Hence, it is crucial for the CPUs to carefully manage the order in which they process instructions.

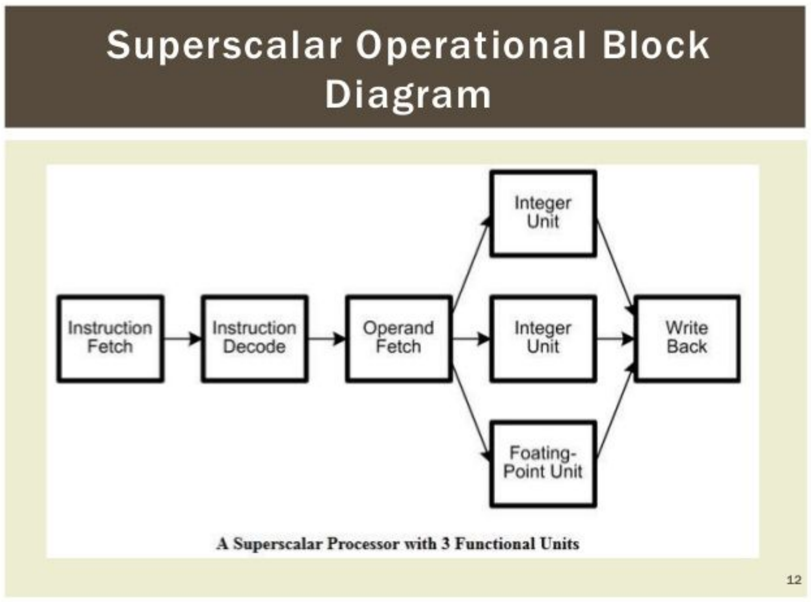

Figure 10: Systolic Operational Block Diagram

## References:

- von-Neumann Architecture:
    - https://whatis.techtarget.com/definition/von-Neumann-bottleneck#:~:text=The%20von%20Neumann%20bottleneck%20is,the%20standard%20personal%20computer%20architecture.&text=In%20the%20von%20Neumann%20architecture,data%20moves%20between%20the%20two.

- Harvard Architecture:
    - https://www.slideshare.net/CarmenUgay/harvard-architecture-12019907

- Comparing von-Neumann and Harvard Architecture:
    - https://www.slideshare.net/matungaolson/von-neumann-vs-harvard-architecture?next_slideshow=1
    - https://www.microcontrollertips.com/difference-between-von-neumann-and-harvard-architectures/

- Modified Harvard Architecture:
    - http://ithare.com/modified-harvard-architecture-clarifying-confusion/

- Parallel Computing Architecture:
    - https://www.omnisci.com/technical-glossary/parallel-computing#:~:text=Parallel%20computing%20refers%20to%20the,part%20of%20an%20overall%20algorithm.
    - https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

- Quantum Computing Architecture:
    - https://arxiv.org/pdf/1702.02583.pdf#:~:text=A%20quantum%20computer%20is%20a,4%2C%205%2C%206%5D.

- Systolic Architecture:
    - http://www.eecs.harvard.edu/~htk/publication/1982-kung-why-systolic-architecture.pdf

- Superscalar Architecture:
    - https://kb.iu.edu/d/aett#:~:text=Superscalar%20architecture%20is%20a%20method,concurrently%20during%20a%20clock%20cycle.

## Image sources:

- Figure 1: https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/
- Figure 2: http://www.differencebetween.net/technology/difference-between-von-neumann-and-harvard-architecture/
- Figure 3: http://ithare.com/modified-harvard-architecture-clarifying-confusion/
- Figures 4,5,6: https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/
- Figure 7: https://www.futurelearn.com/courses/intro-to-quantum-computing/0/steps/31566
- Figure 8: https://stackpointer.io/hardware/how-pipelining-improves-cpu-performance/113/
- Figure 9: https://www.design-reuse.com/articles/19106/systolic-fir-filter-based-fpga.html
- Figure 10: https://www.slideshare.net/NusratMary/superscalar-architectureaiub