Computer Architectures

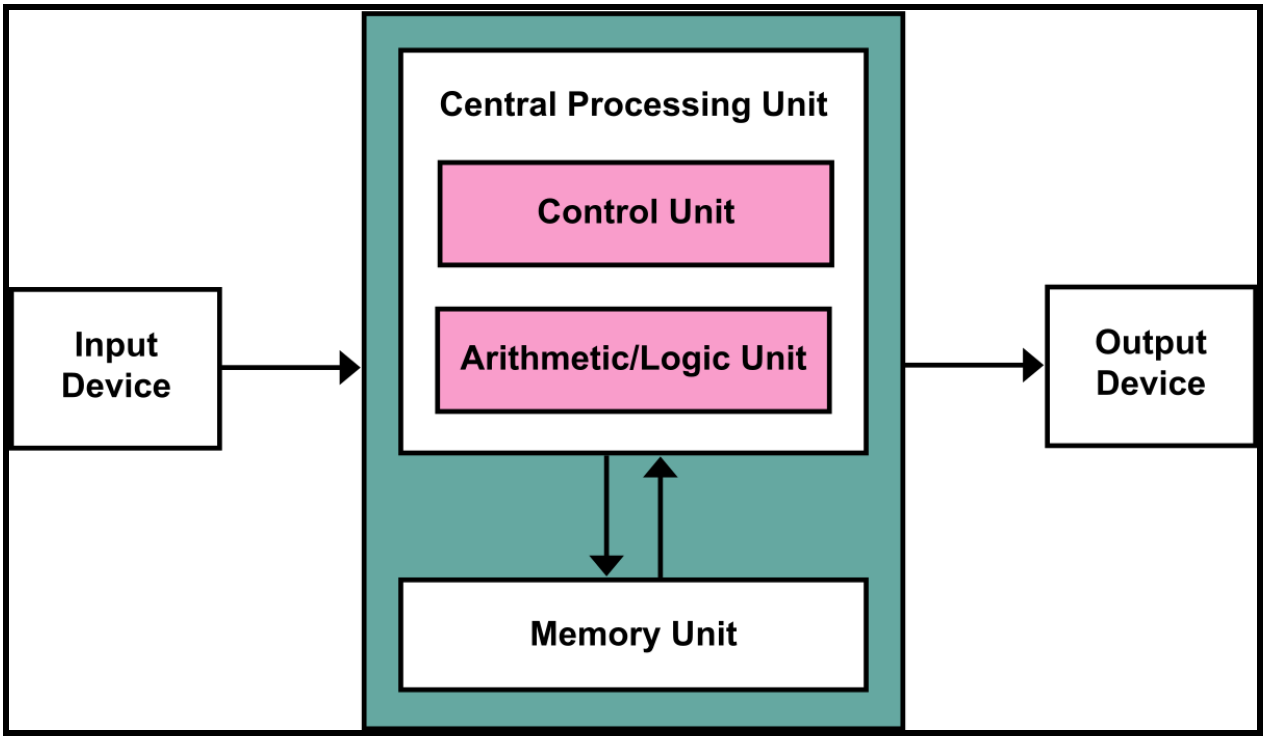
Submitted by,
Shashank Aital (19114076),
Batch O4,
Computer Science and Engineering (B. Tech. - II Year),
Indian Institute of Technology, Roorkee.

Table of Contents

Introduction

This article enlists some of the popular computer architectures. It was made as a submission for the “Reading Assignment - 1” from the course CSN-221 (Computer Architectures and Microprocessors) for the academic year 2020-21. The assignment instructed us to study various architectures apart from Von Neumann Model and Data Flow Model. But, before diving into any other architectures, I would like to write about those in short.

Von Neumann Model



Von Neumann architecture is based on the stored-program concept where instruction and program data are stored in the same memory. This design or its modifications are still used in computers today.

The CPU, in this model, is mainly divided into two main parts namely, Control Unit (CU) and Arithmetic and Logic Unit (ALU). This model uses a number of registers to store data for processing. In the Von Neumann Model, we have a Program Counter(PC) which stores the address of the next instruction to be executed. Once the execution of this instruction completes, the PC increments by 1.

An instruction is executed in two steps:

1. Fetching the instruction: In this step, the instruction corresponding to the address in the PC is copied to a register called the 'Instruction Register'(IR).
2. Execution: Here, the instruction stored in the Instruction pointer is executed by copying the required variables into different registers and sending them to the ALU for computation after which the result is sent to the specified memory address.

Von Neumann bottleneck

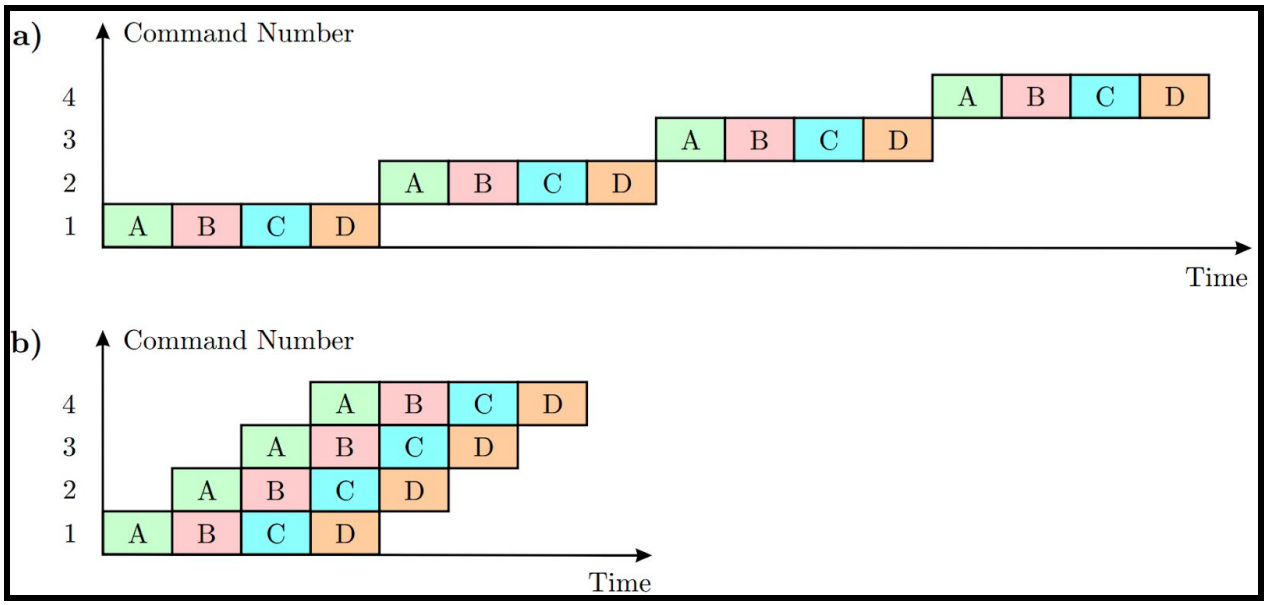
The shared bus between the program memory and data memory leads to the Von Neumann bottleneck, the limited throughput between the CPU and memory compared to the amount of memory. Hence, throughput rate is lower than the CPU working rate. Hence, the speed of the processor is restricted.

Dataflow Model

Dataflow architecture is the exact opposite of the Von Neumann architecture. Dataflow architectures do not have a program counter and the execution of instructions is done wholly based on availability of inputs i.e. as soon as a processing node gets all the inputs, it executes the instruction. An instruction, along with its required data operands, is transmitted to an execution unit as a packet, also known as 'instruction token'. Similarly, output data is transmitted back to the Content Addressable Memory (CAM) as a 'data token'. This packetization of instructions and results allows parallel execution of ready instructions on a large scale.

One of the issues of the Dataflow model is processing involved in determining the processor loads and allocation of tasks at run-time.

Pipelining

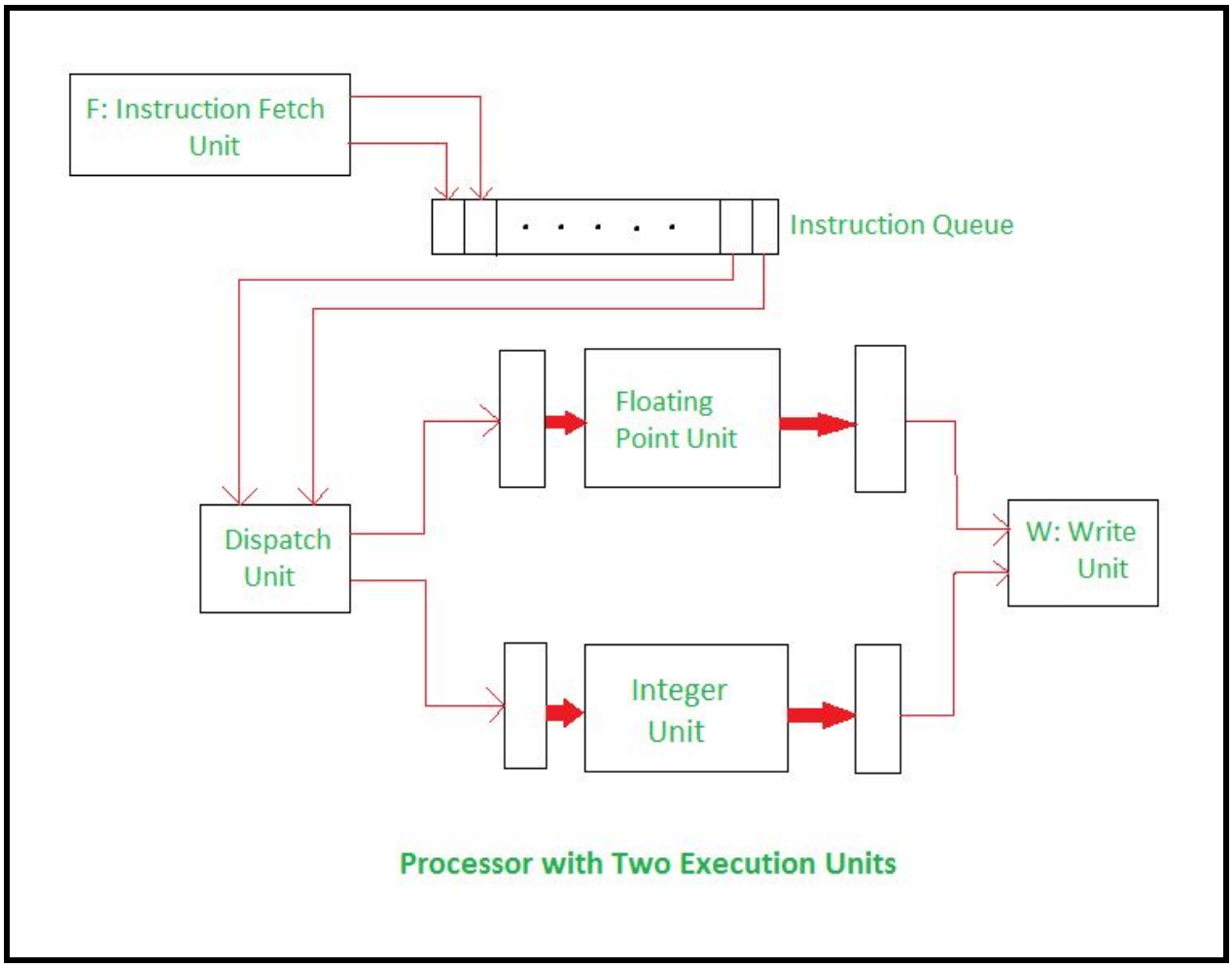


Pipelining is an implementation technique where multiple instructions are overlapped in execution. The program computer pipeline (a set of data processing elements) is divided into different stages. Each stage completes a part of an instruction in parallel.

For example, in the above figure, A, B, C and D are 4 stages of a program instruction. In the case of classical Von Neumann architecture, we would see a behaviour as shown in sub-figure (a). Hence, the total execution time is equal to the number of stages times the number of instructions. Sub-figure (b) shows the characteristics of pipelining. Hence, the total time taken is the difference between the number of stages and the number of instructions.

Please note that pipelining does not decrease the time for individual execution. Instead, it increases instruction throughput. The 'throughput' of the instruction pipeline is determined by how often an instruction exits the pipeline. Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time. The time required to move an instruction one step further in the pipeline is called a 'machine cycle'. The length of machine cycle is determined by the time required by the slowest pipe stage.

Superscalar Architecture

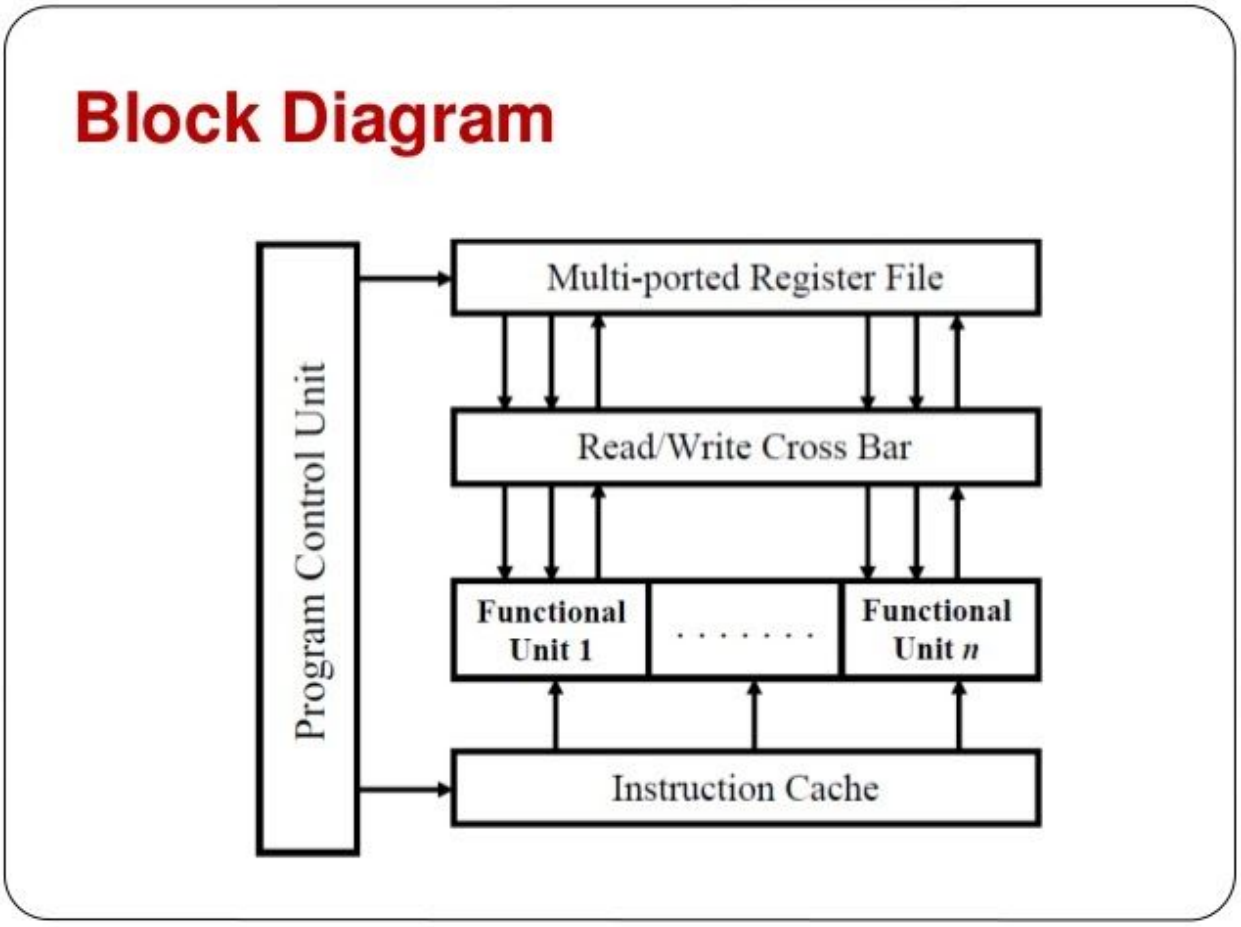


Superscalar architecture is a parallel computing architecture used in many processors. In a superscalar computer, the CPU manages multiple instruction pipelines through a number of execution units within the processor. So, there is a need to choose the instructions smartly as poor choices can lead to pipeline stalls, resulting in idle execution units.

For example, in the above image, there are two execution units - one for integer and the other for floating point operations. The instruction fetch unit is capable of reading the instructions at a time and storing them in the instruction queue. In each cycle, the dispatch unit retrieves and reads upto two instructions from the front of the queue. Hence, both the instructions are dispatched in the same clock cycle.

In general, high performance is achieved if the compiler is able to arrange the program instructions to take maximum advantage of the available hardware units. But, superscalar machines can face an issue in scheduling sometimes.

Very Long Instruction Word

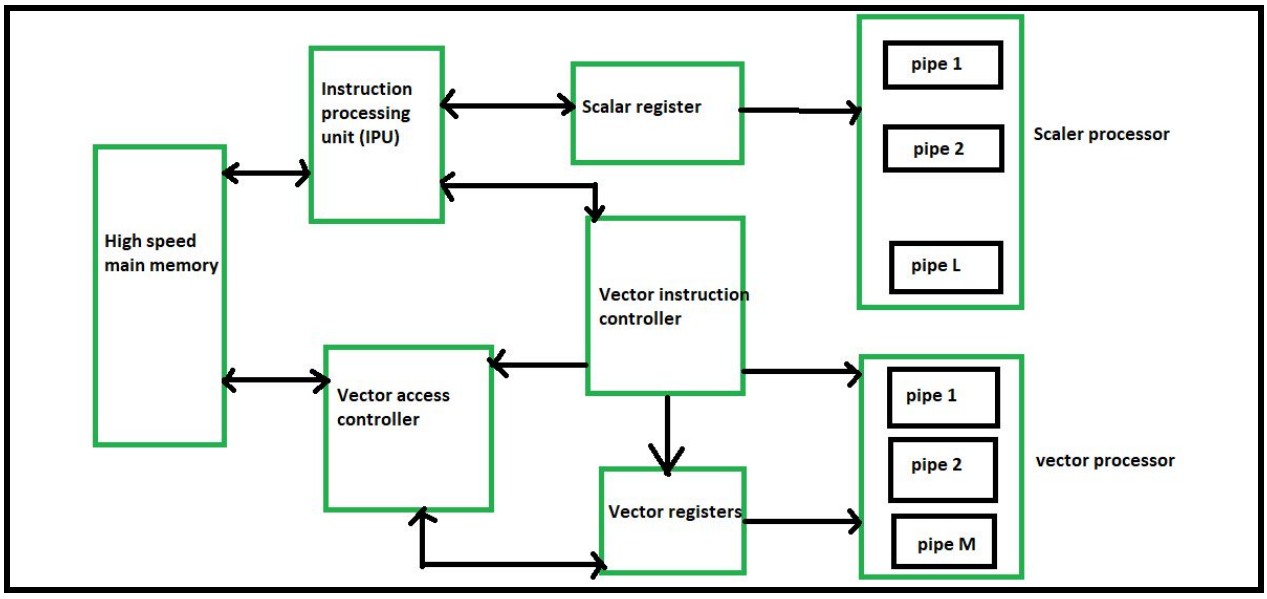


Very Long Instruction Word refers to Instruction Set Architectures designed to achieve instruction level parallelism. Whereas conventional CPUs allow programs to specify instructions in sequential order only, a VLIW processor allows programs to explicitly specify instructions to execute in parallel.

We have seen a couple of methods used to enhance speed of execution above. They include pipelining (Instructions can be executed partly at the same time), superscalar architecture (Individual instructions are executed in different parts of processor) and we could even try to change the order in which the instructions are executed using Out-of-Order Execution. These methods will complicate hardware because the processor must make all of the decisions internally for these methods to work. In contrast, VLSW depends on the program to provide all the decisions regarding which instructions to execute simultaneously, avoiding conflicts (i.e. compiler becomes more complex).

For example, if a VLIW machine has five execution units, then a VLIW instruction for the device has five operands, each field specifying what operation should be performed on that corresponding execution unit.

Vector Processor



A Vector Processor is a CPU that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors, compared to the scalar processors, whose instructions operate on single data items. Note that Vector Processors are different from SIMD based on the fact that vector computers processed the vectors one word at a time using pipelined processors, whereas modern SIMD computers process all the elements of the vector simultaneously.

According to from where the operands are retrieved in a vector processor, pipelined vector computers are classified into two configurations:

1. Memory to memory architecture -
   In memory to memory architecture, source operands, intermediate and final results are retrieved directly from the main memory. Hence, there is no limitation of size. But, the speed is comparatively slow in this architecture.

2. Register to register architecture -
   In register to register architecture, operands and results are retrieved indirectly from the main memory through the use of a large number of vector registers or scalar registers. Hence, they are extremely fast compared to the previous architecture. However, it has some issues too. Using this type of architecture means we have restricted size and the hardware cost is comparatively high.

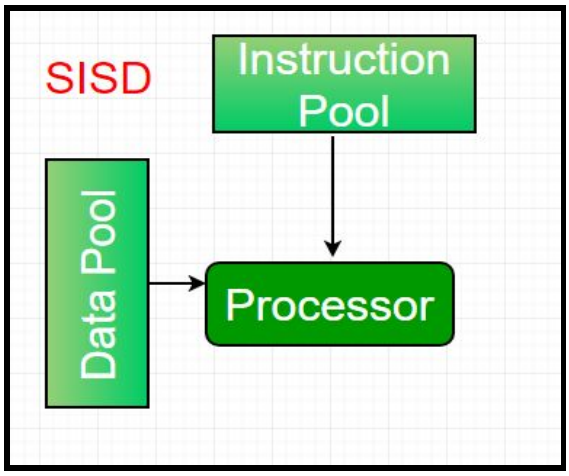
A block diagram of a modern multiple pipeline vector computer is shown in the figure above. It follows Register to Register architecture.

Parallel Computing - Flynn's Taxonomy

Parallel Computing is a computing where the jobs are broken down into discrete parts that can be executed concurrently. Each part is further broken down into a series of instructions. Instructions from each part execute simultaneously on different CPUs. Based on the number of Instructions and data streams that can be processed simultaneously, computing systems are classified into four major categories:

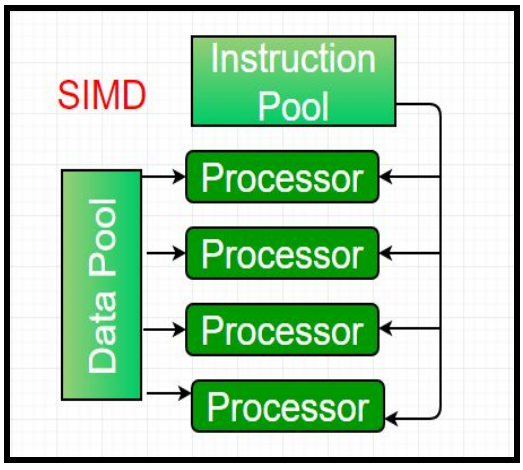
1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instructions Single Data (MISD)
4. Multiple Instructions Multiple Data (MIMD)

1. SISD systems



An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream. These computers are called sequential computers. This is the traditional Von Neumann single CPU computer. The speed of the processing element in the SSID model is limited by the rate at which the computer can transfer information internally.

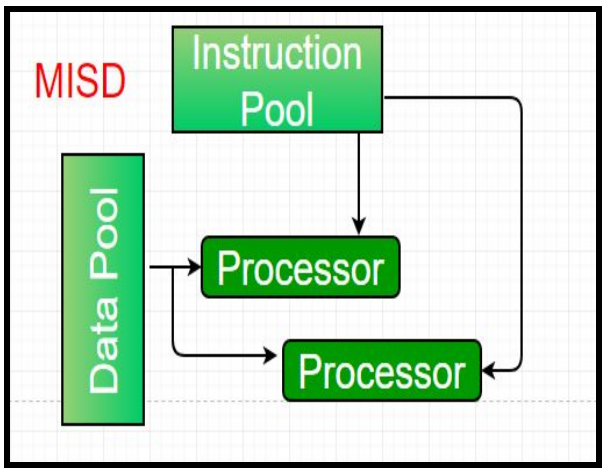
2. SIMD systems



An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPU but operating on different data streams. Machines based on an SIMD model are well suited for scientific computations as they involve vector and matrix calculations. A matrix can be divided into small parts and fed to one of the many Processing Elements (PE)

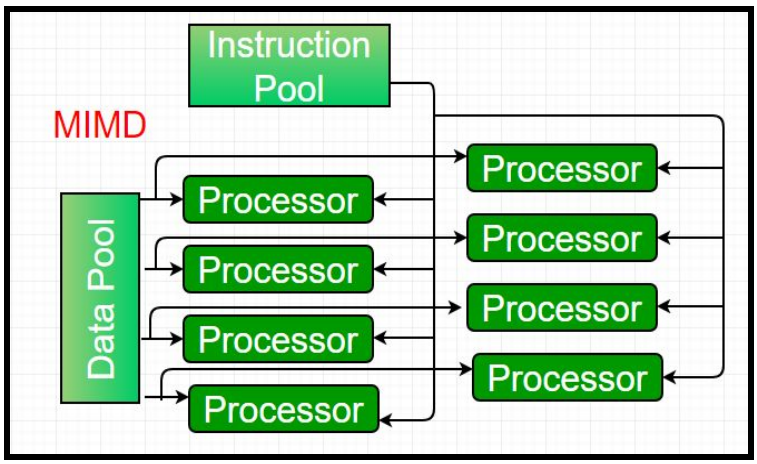
after which a single operation would be performed on all the parts simultaneously.

3. MISD systems



An MISD system is capable of executing different instructions in different Processing Elements (PE). But, all of them operate on the same dataset.
Machines built using the MISD model are not useful in most of the applications. Hence, none of them are available commercially.

4. MIMD systems



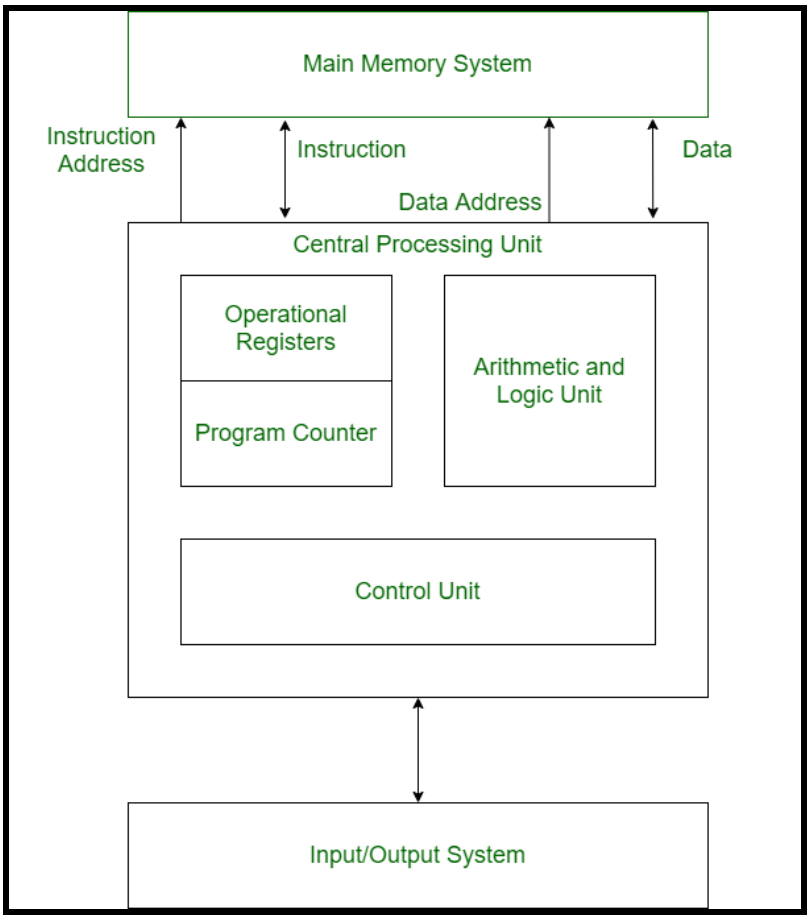
An MIMD system is a multiprocessing machine which is capable of executing multiple instructions on multiple data sets. Each PE in the MIMD has separate instruction and data streams. Therefore, machines built using this model are capable of any kind of application. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

MIMD machines are categorized into shared-memory MIMD and distributed-memory MIMD based on the way PEs are coupled to the main memory.

Shared-memory MIMD - Here, all the PEs are connected to a single global memory and they all have access to it. The communication between PEs takes place through the shared memory. They are also known as ‘tightly coupled multiprocessor systems’.

Distributed-memory MIMD - Here, all PEs have a local memory. The communication between different PEs takes place with the help of interconnection network (IPC - Inter Process Communication channel).

Harvard Architecture



In a normal computer that follows Von Neumann architecture, instructions and data are stored in the same memory. Hence, as described in the introduction, the processor can fetch either instruction or the data in one cycle. Hence, each instruction requires two clock cycles to execute. This reduces throughput. Harvard Architecture contains separate storage and separate buses for instruction and data. Hence, it overcomes the Von Neumann bottleneck. The main advantage of this architecture is that it can read instructions and modify data at the same time.

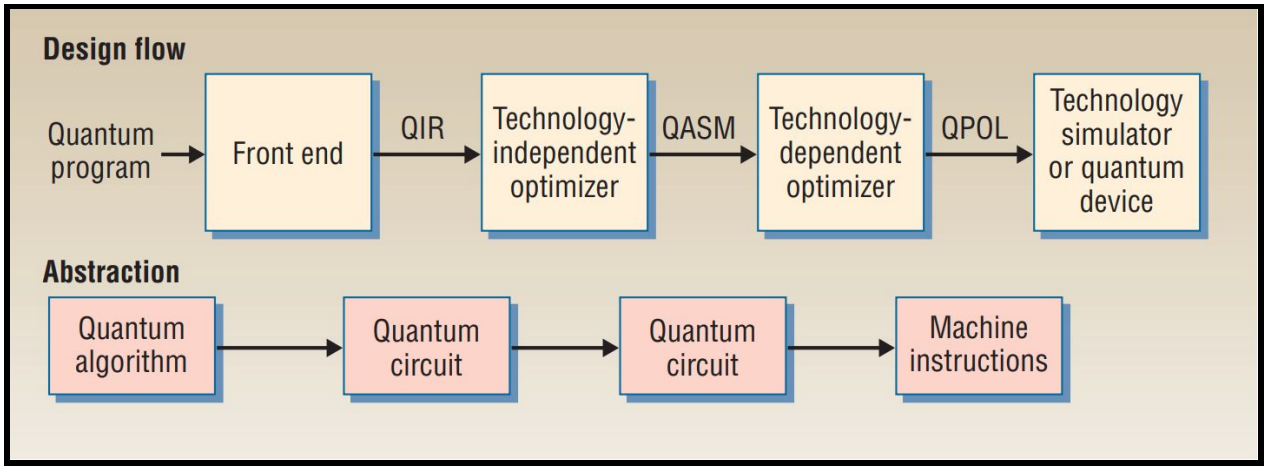
As we can see in the figure above, there are separate buses for fetching instruction and data. Rest of the function is similar to the Von Neumann architecture.

However, there is a drawback with pure harvard architecture. As the memory locations of instruction employ different memories, the instruction and data had to be retrieved using different instructions.

Hence, in practice, we use Modified Harvard Architecture where we have two separate caches. This architecture also relaxes the strict separation between instruction and data while still letting the CPU concurrently access two or more memory buses.

Quantum Computer

As of now Quantum computers use the *uniform quantum circuit model*. There is a Quantum Von Neumann Model under research too in which the Control Unit (CU) is classical but the ALU, memory and buses are implemented using principles followed by quantum mechanics. But here, I would like to discuss a proposed model named Four-Phase Design Flow. As the name suggests, The Four-Phase Design Flow consists of four phases as described below:



We use a high level Quantum Programming Language to encapsulate the mathematical abstractions of quantum mechanics and linear algebra (examples can be qiskit, Q#, etc.). The design flow's first three phases are a part of the Quantum Computer Compiler (QCC). The last phase implements the algorithm on a quantum device or simulator.

In the first phase, the compiler front end maps a high-level specification of a quantum algorithm into a Quantum Intermediate Representation (QIR) - a quantum circuit with gates drawn from some universal set.

In the second phase, a technology-independent optimizer maps the QIR into an equivalent lower-level circuit representation of single qubit and controlled-NOT (CNOT) gates. The compiler optimizes this Quantum Assembly Language (QASM) according to a cost function such as circuit size, circuit depth or accuracy.

The third phase consists of optimizations suited to the quantum computing technology and outputs Quantum Physical Operations Language (QPOL), a physical-language representation with technology-specific parameters.

The final phase utilizes technology-dependent tools such as layout modules, circuit and physical simulators, or interfaces to actual quantum devices.

Quantum Assembly Language

QASM is a classical reduced-instruction-set computing assembly language extended by a set of quantum instructions based on the quantum circuit model. It uses qubits and registers of classical bits (cbits) as static units of information that must be declared at the program's beginning. Quantum Instructions in QASM consist solely of single-qubit unitary gates, CNOT gates, and measurements. Any quantum circuit can be constructed using these instructions.

Quantum Physical Operations Language

QPOL precisely describes the execution of a given quantum algorithm expressed as a QASM program on a particular technology, like trapped-ion systems. QPOL includes physical operations as well as technology specific modules. In particular, it organizes physical operations into five instruction types:

- *Initialization* instructions specify how to prepare the initial system state. It includes loading qubits into a quantum computer, initializing auxiliary physical states used in computations and setting qubits to $|0\rangle$.
- *Computation* instructions include quantum gates and measurements.
- *Movement* instructions control the relative distance between qubits to bring them together to undergo simultaneous operations or move them apart. (This could be done using *quantum teleportation* which uses two classical bits to teleport qubits instead of using physical buses which introduce a lot of error into quantum computation.
- *Classical control* instructions provide simple logic operations and allow quantum gates to be applied based on classical bit values stored in classical memory.
- *System-specific* instructions control physical parameters of the system that do not explicitly fall into the other categories.

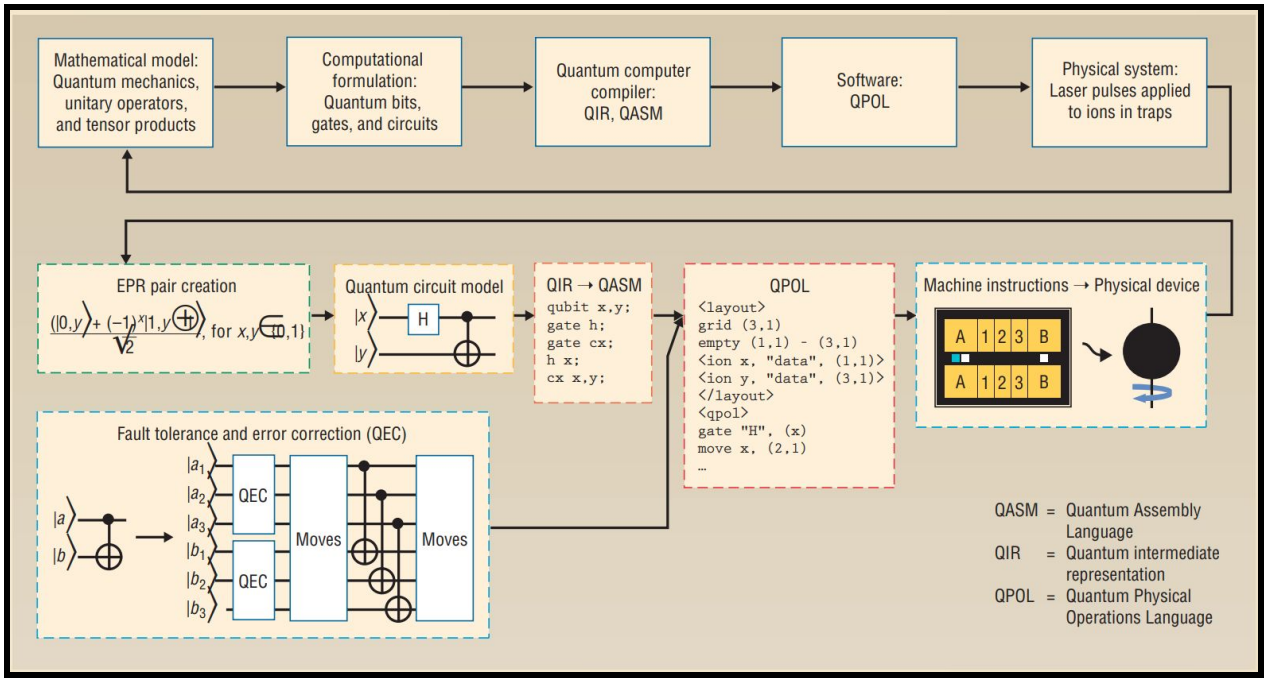
In case of trapped-ion computers,

Initialization has 3 stages: Loading of multiple ions into a loading region, laser cooling to reduce ion temperatures and optical pumping to put all qubits into a known state.

Computation is naturally described in terms of single-qubit rotations and a controlled-phase gate between ions in the same trap, both achieved using a laser pulse sequence.

An external classical processor controls the execution of QPOL instructions, stores measurement results and performs conditional instructions based on stored cbits.

System-specific instructions recool ions when they heat due to movement operations. Certain laser pulses also accomplish recooling, but the lasers are applied differently for cooling than for gates, requiring different programming and pulse-sequence optimization.



The example above demonstrates the production of Einstien-Podolsky-Rosen (EPR) Pairs for implementation on a trapped-ion computer. Trapped ion systems are one of the technologies that has potential as a future quantum computing technology. Transmon qubits, used by IBM quantum computers are one of the state of the art quantum architectures too. The transmon qubits have a Josephson Tunnel Junction which is trapped between two superconductors in order to form a perfect LC circuit. Trapped ion systems on the other hand, use charged, electromagnetically trapped atoms as qubits. Ions can be shot in and out of ion traps to increase the quantum computer's effective size.

For EPR pair creation, we abstract the mathematical representation in a quantum circuit composed of a Hadamard (H) and a CNOT gate. The figure shows sample QASM and QPOL representations. When and where to insert fault tolerance and error correction is an open research question.

QPOL instructions for creating an EPR pair can be translated into a sequence of laser pulses - in this case, for performing a CNOT gate on an ion-trap device.

Conclusion

Through this assignment, I was able to explore different computer architectures and compare them based on their advantages and disadvantages. This exercise was really informative and let me enjoy different architectures including a Quantum Computing architecture which is still under research. Completing this assignment has not only incubated an interest in computer architectures but also provided me with an opportunity to enhance my knowledge of quantum computing hardware and the recent development in the field of quantum computing.

I am certainly looking forward to more of these hands-on exercises in the future especially in Quantum Computing Architectures.

References

- *Von Neumann Model and Von Neumann Bottleneck - https://en.wikipedia.org/wiki/Von_Neumann_architecture*
- *Data Flow Model - https://en.wikipedia.org/wiki/Dataflow_architecture*
- *Pipelining - http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html#:~:text=Pipelining%20is%20an%20implementation%20technique,of%20an%20instruction%20in%20parallel.&text=We%20call%20the%20time%20required,the%20pipeline%20a%20machine%20cycle%20.*
- *Superscalar architecture - https://kb.iu.edu/d/aett#:~:text=Superscalar%20architecture%20is%20a%20method,concurrently%20during%20a%20clock%20cycle.*
- *VLIW Architecture - https://en.wikipedia.org/wiki/Very_long_instruction_word*
- *Vector Processing - https://en.wikipedia.org/wiki/Vector_processor*
- *Flynn's taxonomy - https://www.geeksforgeeks.org/parallel-processing-systolic-arrays/ , https://en.wikipedia.org/wiki/SIMD*
- *Harvard Architecture - https://en.wikipedia.org/wiki/Harvard_architecture, https://en.wikipedia.org/wiki/Modified_Harvard_architecture, https://www.geeksforgeeks.org/harvard-architecture/*
- *The Four Phase Design Flow and Quantum Computing - https://web.eecs.umich.edu/~imarkov/pubs/jour/computer06-q.pdf, https://arxiv.org/pdf/1702.02583.pdf#:~:text=A%20quantum%20computer%20is%20a,4%2C%205%2C%206%5D., https://en.wikipedia.org/wiki/Quantum_computing#Quantum_computing_models, http://www.research.ibm.com/*