SVKM'S NMIMS, School of Technology Management & Engineering | Navi-Mumbai

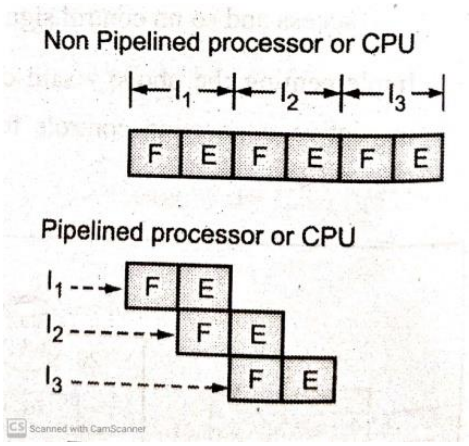
MBA-Tech (A.Y. 2020-21)

Lecture Notes

(Pipelining and Parallel Processing Unit)

INSTRUCTION PIPELINING:

- Instruction cycle of the processor or CPU consists of many functions and is operationally divided into its functional units. A single instruction in the processor life cycle at a time is an inefficient process.
- When a specific instruction completes its work on the specific functional unit and goes to the next functional unit then the earlier functional unit is free, and it can process subsequently next instruction in the same time frame.
- Therefore, at any given instance, different instructions are in the different stages of advancement at the same time allows a greater number of instructions to be executed in the same time. This feature is called as instruction Pipeline.
- Consider a case of a non-pipelined processor as against a pipelined processor as shown in the Fig. We consider two functional operations, fetching of the instructions (F) and execution of the instruction (E).



- In a non-pipelined processor case, it is seen that instructions require two time slots each two complete its fetching and execution.
- On the other hand, in the case of pipelined processor, it is possible to fetch the next instruction at the same time while the previous instruction is being executed.
- This functional overlap allows a greater number of instructions to be executed in the same time frame (the number of instructions executed would be one less than double).
- To increase the performance of the processor instructions are simultaneously fetched and executed this technique of fetching while executing is called as instruction pipelining.

➢ Pipelining Strategy:

- Every instruction in processor is divided into smaller task, some task in processor are independent to each other. For e. g. If the processor is performing arithmetic

operation with register operands, then the buses are free to fetch next instruction. Therefore, the task of execution and fetching can be performed simultaneously.

- This will reduce the waiting time also two tasks are completely in the same clock cycle which will reduce the required time to execute the next instruction.
- This technique of processing simultaneously is called as pipelining.

➢ Pipeline performance:

- As explained above, the instruction pipeline improves the performance of the system as it allows the simultaneous execution of instruction and thereby increasing the number of instructions executed per unit time as against any non-pipelined processor.
- The performance characterizes of pipeline are determined by different matrix consider a generic instruction pipeline that is having k stages in the pipeline. Let this pipeline be subjected to execution of a sequence of n instructions then, we can deduce that, A non-pipelined processor would require clock periods each to execute one entire instruction and therefore to execute a sequence of n instructions it would require k x n number of clock periods.
- An equivalent k stage pipelined processor would require k clock periods to execute first instruction however the remaining n-1 instructions would require only one clock period each to execute. Therefore, total number of clock period required to execute are k + n-1 which is much smaller as compared to (k x n).
- This clearly indicates that the pipeline performance will be improved.

➢ Speed Up (s):

- It is defined as the ratio bf the amount of time taken by a non-pipelined processor to the amount of time taken by an equivalent k stage pipelined processor evaluated over the execution of program sequence of n instructions.
- Now we have time taken by non-pipeline processor to be (k x n) and time taken by k stage pipeline processor to be (k + n-1). Therefore, Speed Up

$$s = (k \times n) / (k+n-1)$$

➢ Clocks per instruction (CPI):

- CPI is another important performance matrix of pipeline. It gives an indication that how many clock periods are needed to complete an instruction on an average.
- It is defined as the number of clock period required per instruction on an average taken over the sequence of n instructions in an instruction pipelined processor.
- We know that, to complete the execution of n instruction, a k stage pipelined processor requires (k + n-1) clock Periods. Therefore

$$CPI = (k+n-1) / n$$

➢ Throughput:- Instructions Per Second (IPS) and Million Instructions Per Seconds (MIPS):

- Since most of the processor operates in frequency range above megahertz, a more convenient unit is the million instructions executed in one second (MIPS). A higher value of MIPS indicates a better performing processor.

➢ Latency:

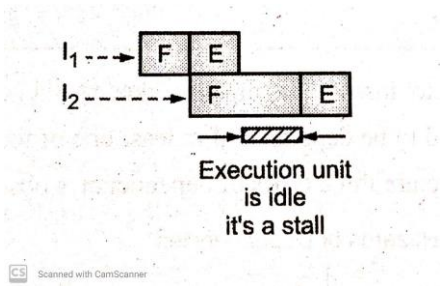
- The latency is time delay. In the context of Instruction pipeline, latency is defined as the time delay between the times at which instruction is fetched and loaded in the processor or CPU and the time at which the same specific instruction completes its execution on the processor. All latencies are measured in Sec. (or more appropriately in Micro-seconds or Nano-seconds).
- Issue Latency is the latency or time delay in issuing the instruction for execution.
- Execution Latency is the latency or time delay in completing the execution of an instruction over the processor.

➢ Pipeline Hazards:

- It assumes the pipeline is running absolutely smooth and every instruction requires exactly same time in each functional stage of the pipeline and therefore after the first instruction, every instruction is completing its execution in exactly one clock period.
- However, in the practical situation, it is not the case. There are many resource bottle necks and operational difficulties that don't allow the pipeline to operate in ideal condition.
- All such practical issues and resource bottle necks that make the instruction pipeline to deviate from its ideal characteristics are called as Hazards.
- Hazards introduce inefficiency and therefore increase the execution time of instructions in certain cases. Therefore, hazards in general degrade the instruction pipeline performance.
- The hazards can be classified into three types as follows:
  (a) Structural Hazard
  (b) Data Hazard
  (c) Control Hazard

❖ **Structural Hazards (or Resource Hazards):**

- Structural Hazards or resource hazards are the hazards introduced in an instruction pipeline by virtue of structural problems or resource bottle necks.
- Consider a situation where the fetch and execute stages of the pipeline are of dissimilar durations.



- Such situation is called as pipeline stall which pushes the timings forward making instructions to take longer times to execute and therefore degrading the performance.
- Alternately, if the current instruction that is under execution requires a memory reference (e. g. MOV AX, [Bx]) and at the same time fetch stage also requires access to the memory for fetching the next instruction in the program.
- In this case both fetch stage and execute stage are requiring the memory reference over the busses.
- However, the busses can be given to only one of them at a time.
- This forces the two operations to be performed one after the other due to the resource bottle on the busses.
- The result is that the instruction execution takes longer time and pipelined performance is degraded.

❖ **Data Hazards or Dependencies:**

- Data Hazards or Dependencies are introduced into instruction pipeline due to the consecutive instruction being dependent. The two consecutive instructions are said to be dependent if at least one of the source or destination operand referred in these two instructions are common.
- There are three types of dependencies observed in data hazards.

- Read after Write (RAW) dependency or flow dependency:
  - In this type of data dependency hazard, the destination operand of the previous instruction is one of the source operands in the subsequent instruction.
  - Therefore, it is termed as read after write dependency. This type of dependency always leads to the data hazard in instruction pipeline.
  - This happens because the subsequent instruction cannot fetch its source operands unless the previous instruction has stored its destination operand and generally results in one clock period stall as a subsequent instruction execution waits.
- Write after Read (WAR) dependency or Anti-Dependency:
  - In this type of data dependency hazard the one of the source operands of the previous instruction is the destination operand of the subsequent instruction.
  - Therefore, it is termed as write after read (WAR) dependency.
  - This type of dependency does not lead to the data hazard in instruction pipeline. Therefore, usually WAR dependency is not needed to be treated further.
- Write after Write (WAW) dependency or Output Dependency:
  - In this type of data dependency hazard, the destination operands of the previous instruction are same as destination operand in the subsequent instruction. Therefore, it is termed as Write after Write dependency.

❖ **Control Hazards or Instruction Hazards:**

- These are caused due to the control transfers encountered in the user programs or due to the system activity of events.

- Control transfers are generally caused by the conditional and unconditional branch instructions and subroutine calls in the user programs.
- They can also be caused by system events such as interrupts, exceptions and task switches.
- The branch instruction comes for execution, next few instructions are already in the pipe line but, since the next instruction is to be executed from the branch address (target address), all the instructions in the pipeline are flushed.
- Therefore, in total heavy penalty in terms of pipeline stalls and delay is required to be paid.
- Control hazards are the hazards that contribute maximum to the pipeline stalling and degradation of the system performance.

Introduction to Parallel Processors:

Flynn's Classification:

- The processor or CPU has primary functionality of processing the data by executing programs. Program is sequence of instructions. Both programs and data are stored in the memory.
- When the processor or CPU executes the program it is needed to fetch instructions in the program from the memory. Once the instruction is fetched, it is decoded and executed in the processor or CPU.
- While executing it reads the input data operands from the memory and it stores output data operands to the memory.
- The flow or movement of instructions from memory to the CPU is called as instruction stream. The two-way flow of input and output data operands between the CPU and the memory is called as data stream.
- Therefore, there are two types of streams observed: Instruction stream and Data stream
- Based on the multiplicity of the instruction and data stream, the computers are classified into four different types. This classification or taxonomy is called as Flynn's Classification or Taxonomy. The four types are:

  (i) SISD: Single Instruction Stream, Single Data Stream

  (ii) SIMD: Single Instruction Stream, Multiple Data stream

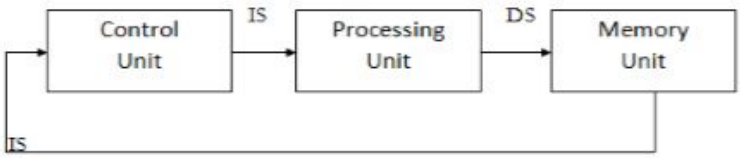
  (iii) MISD: Multiple Instruction stream, single Data stream

  (iv) MIMD: Multiple Instruction Stream, Multiple Data Stream

(i) Single Instruction Stream, Single Data stream (SISD):

In this organization, there is only one instruction stream and one data stream. The CPU can be visualized to be made up of two parts namely: Data processing unit (DPU/PU) and control unit (CU). The DPU deals with data and consists of ALU and the data path. Therefore, DPU accesses the data stream from the memory bidirectional.

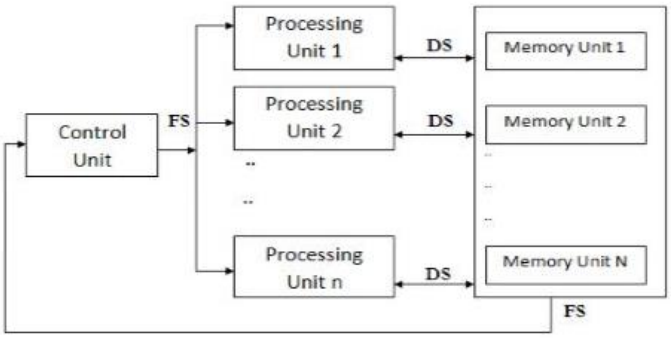
The control unit deals with the instruction stream as it receives instructions from the memory and decodes them. Subsequently, the CU gets these instructions executed from the DPU. It has only one data stream and one instruction stream. This stream is the most widely used computer type. The fig. shows SISD



(ii) Single Instruction Stream, Multiple Data Stream (SIMD):

In this type of organization of computer systems, there are multiple DPU's. They are controlled by a single control unit. The CU receives instructions from the memory over a single instruction stream and generates controls on the basis of decoded instruction to different DPU and each DPU processes different data element. The data element is accessed from the memory over different data streams.

Therefore, SIMD processors are capable of executing the same instruction on multiple data elements simultaneously. These types of computer systems are widely used in vector processor and find their applications in multimedia and signal processing. The Fig. shows organization of SIMD computers.



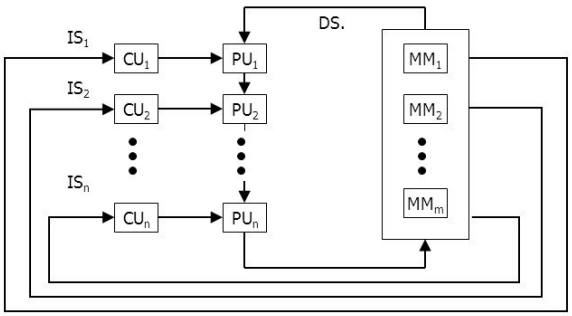
(iii) Multiple Instruction Stream, Single Data Stream (MISD):

In the organization of this type of computer system, there are multiple data processing units and multiple control units.

The data processing units are arranged in a sequential manner in such a way that output of one DPU is input of the next DPU in order. Therefore, all DPUs together

handle only one data element, which is accessed from the memory over single data stream.

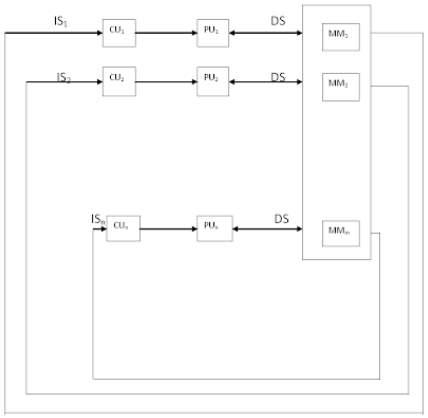
Each DPU receives controls from its independent CU and each CU processes its different instruction. Each of these instructions is fetched from the memory over different instruction streams. Therefore, this type of processor processes the same data elements sequentially by multiple instructions.



(iv) Multiple Instruction Stream, Multiple Data Stream (MIMD):

In this type of computer systems there are multiple Instruction control units and multiple data processing units. Each DPU processes different data element using different data streams. Each DPU receives controls from its independent CU and each CU processes its different instruction

Each of these instructions is fetched from the memory over different instruction streams. Therefore, these kinds of processors are capable of handling multiple independent instructions on multiple different data elements simultaneously.

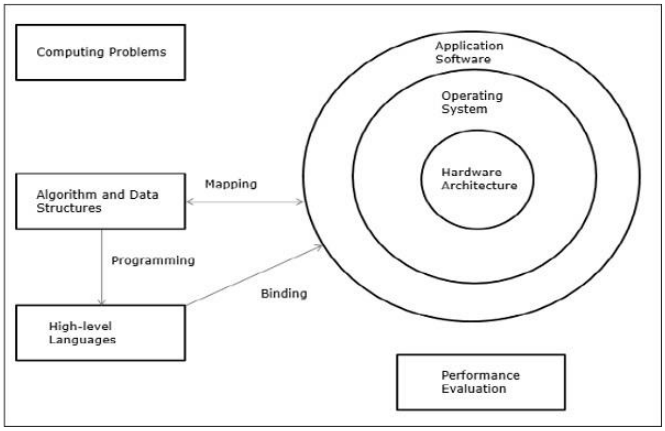


Concurrent Access to Memory:

Parallel processing has been developed as an effective technology in modern computers to meet the demand for higher performance, lower cost and accurate results in real-life applications. Concurrent events are common in today's computers due to the practice of multiprogramming, multiprocessing, or multi-computing.

Modern computers have powerful and extensive software packages. To analyze the development of the performance of computers, first we have to understand the basic development of hardware and software.

- Computer Development Milestones − There is two major stages of development of computer - mechanical or electromechanical parts. Modern computers evolved after the introduction of electronic components. High mobility electrons in electronic computers replaced the operational parts in mechanical computers. For information transmission, electric signal which travels almost at the speed of a light replaced mechanical gears or levers.
- Elements of Modern computers − A modern computer system consists of computer hardware, instruction sets, application programs, system software and user interface.



The computing problems are categorized as numerical computing, logical reasoning, and transaction processing. Some complex problems may need the combination of all the three processing modes.

- Evolution of Computer Architecture − In last four decades, computer architecture has gone through revolutionary changes. We started with Von Neumann architecture and now we have Multicomputers and multiprocessors.
- Performance of a computer system − Performance of a computer system depends both on machine capability and program behaviour. Machine capability can be improved with better hardware technology, advanced architectural features and efficient resource management. Program behaviour is unpredictable as it is dependent on application and run-time conditions

Multiprocessors and Multicomputers

In this section, we will discuss two types of parallel computers −

- Multiprocessors
- Multicomputers

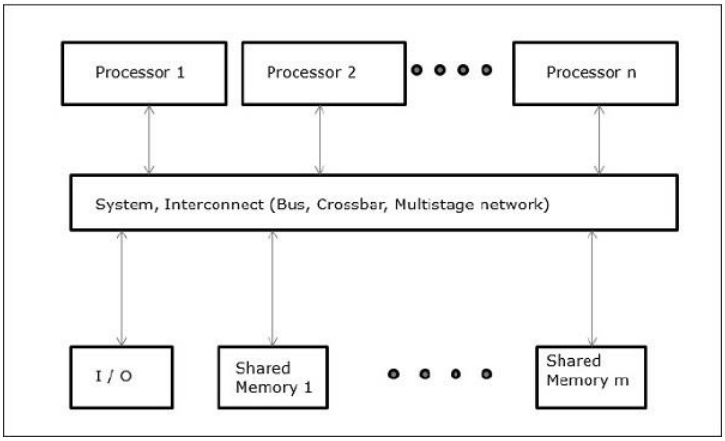
Shared-Memory Multicomputers

Three most common shared memory multiprocessors models are −

Uniform Memory Access (UMA):

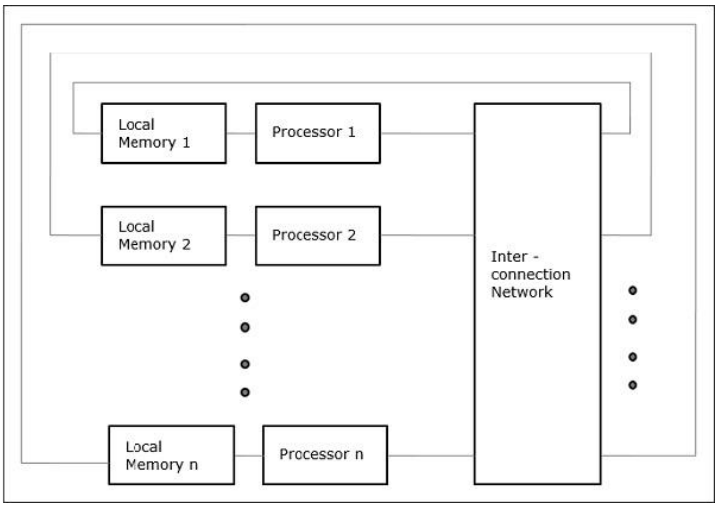
In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

When all the processors have equal access to all the peripheral devices, the system is called a symmetric multiprocessor. When only one or a few processors can access the peripheral devices, the system is called an asymmetric multiprocessor.



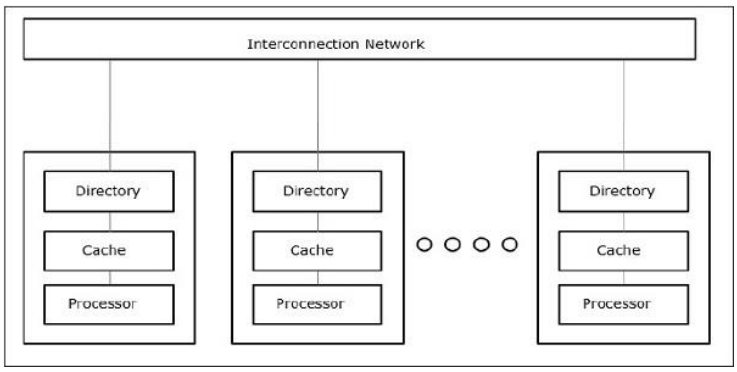
Non-uniform Memory Access (NUMA):

In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.

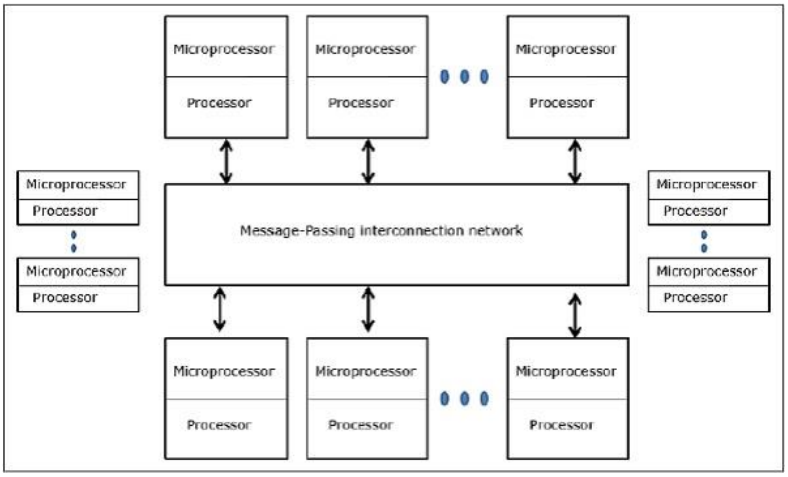


Cache Only Memory Architecture (COMA):

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.



- Distributed - Memory Multicomputers − A distributed memory multicomputer system consists of multiple computers, known as nodes, inter-connected by message passing network. Each node acts as an autonomous computer having a processor, a local memory and sometimes I/O devices. In this case, all local memories are private and are accessible only to the local processors. This is why, the traditional machines are called no-remote-memory-access (NORMA) machines.



Cache Coherency:

In multiprocessor system where many processes needs a copy of same memory block, the maintenance of consistency among these copies raises a raises a problem referred to as **Cache Coherence Problem.**
This occurs mainly due to these causes: -

- Sharing of writable data.
- Process migration.
- Inconsistency due to I/O.

Cache Coherence Protocols:

These are explained as following below:

1. MSI Protocol:

This is a basic cache coherence protocol used in multiprocessor system. The letters of protocol name identify possible states in which a cache can be. So, for MSI each block can have one of the following possible states:

- **Modified –**
  The block has been modified n cache, i.e., the data in the cache is inconsistent with the backing store (memory). So, a cache with a block in "M" state has responsibility to write the block to backing store when it is evicted.
- **Shared –**
  This block is not modified and is present in at least one cache. The cache can evict the data without writing it to backing store.
- **Invalid –**
  This block is invalid and must be fetched from memory or from another cache if it is to be stored in this cache.

2. MOSI Protocol:

This protocol is an extension of MSI protocol. It adds the following state in MSI protocol:

- **Owned –**
  It indicates that the present processor owns this block and will service requests from other processors for the block.

3. MESI Protocol –

It is the most widely used cache coherence protocol. Every cache line is marked with one the following states:

- **Modified –**
  This indicates that the cache line is present in current cache only and is dirty i.e its value is different from the main memory. The cache is required to write the data back to main memory in future, before permitting any other read of invalid main memory state.
- **Exclusive –**
  This indicates that the cache line is present in current cache only and is clean i.e its value matches the main memory value.
- **Shared –**
  It indicates that this cache line may be stored in other caches of the machine.
- **Invalid –**
  It indicates that this cache line is invalid.

4. MOESI Protocol:

This is a full cache coherence protocol that encompasses all of the possible states commonly used in other protocols. Each cache line is in one of the following states:

- **Modified –**
  A cache line in this state holds the most recent, correct copy of the data while the copy in the main memory is incorrect and no other processor holds a copy.
- **Owned –**
  A cache line in this state holds the most recent, correct copy of the data. It is similar to shared state in that other processors can hold a copy of most recent, correct data and unlike shared state however, copy in main memory can be incorrect. Only one processor can hold the data in owned state while all other processors must hold the data in shared state.
- **Exclusive –**
  A cache line in this state holds the most recent, correct copy of the data. The main memory copy is also most recent, correct copy of data while no other holds a copy of data.
- **Shared –**
  A cache line in this state holds the most recent, correct copy of the data. Other processors in system may hold copies of data in shared state as well. The main memory copy is also the most recent, correct copy of the data, if no other processor holds it in owned state.
- **Invalid –**
  A cache line in this state does not hold a valid copy of data. Valid copies of data can be either in main memory or another processor cache.