



UNIT 5

COMPUTER ORGANIZATION

Dr. Ankita Markare

VirtualMemory

Virtual memory loosely describes a hierarchical storage system of at least two levels, which is managed by an operating system to appear to a user like a single large directly addressable main memory.

The three main reasons of using a virtual memory are as follows:

- 1.To free users from the need to carry out storage allocation.
- 2.To make the programs independent of the configuration and capacity of the memory systems used during their execution
3. To permit efficient sharing of memory space among different users and achieve the high access rates and low cost per bit that is possible with a memory hierarchy.

Locality of Reference

The predictability of logical memory addresses which is essential to the successful operation of a memory hierarchy is based on a common characteristic of a computer called as locality of reference.

Two different types of locality have been observed:

Temporal locality: states that recently accessed items are likely to be accessed in the near future.

Spatial locality: says that items whose addresses are near one another tend to be referenced close together in time.

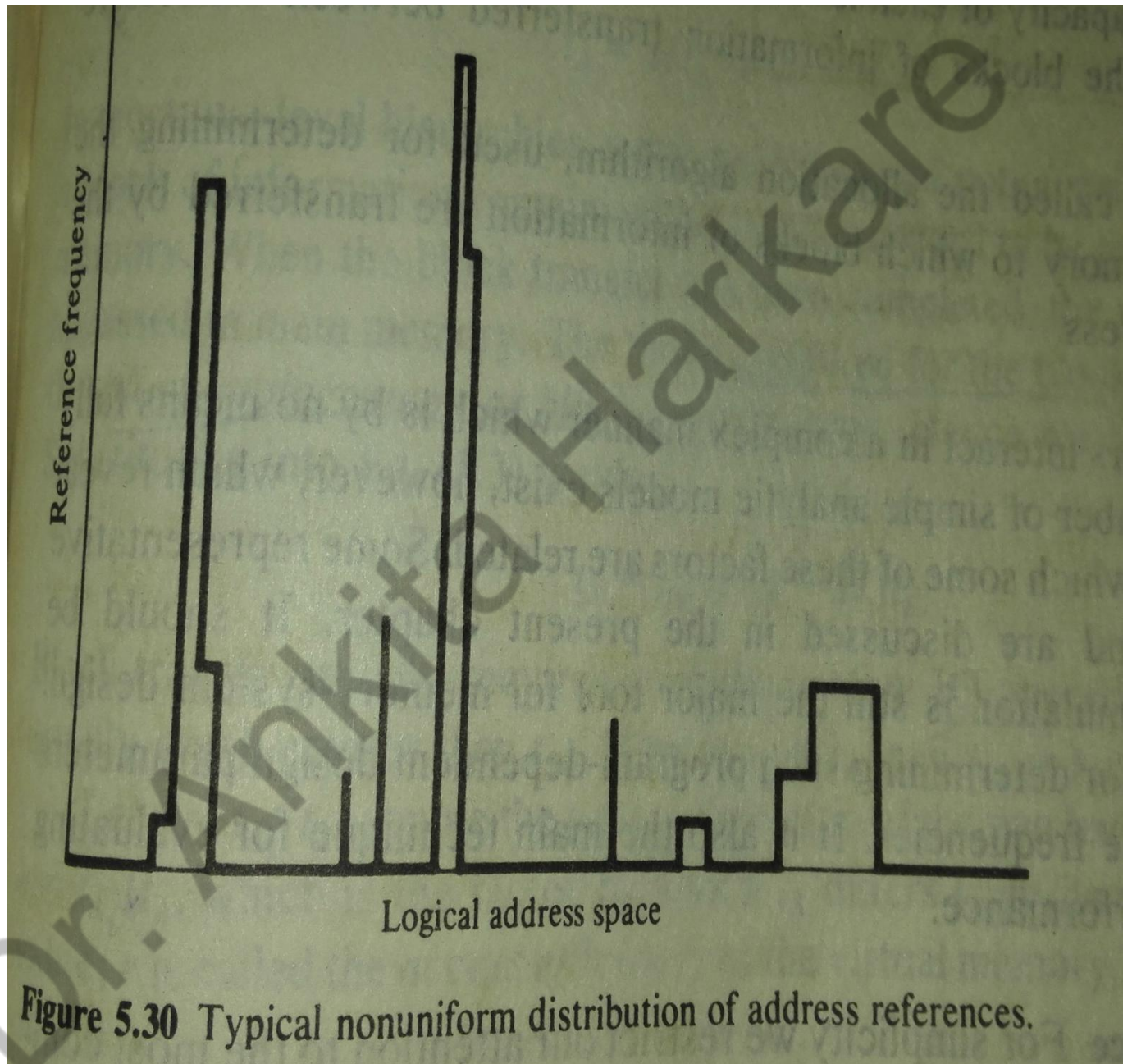


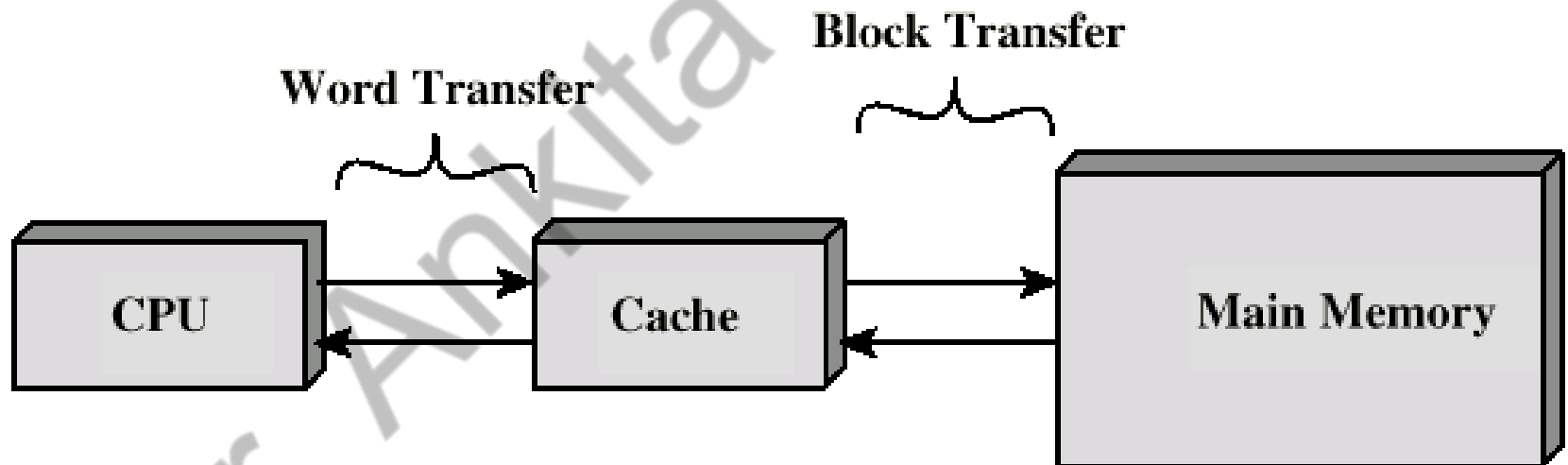
Figure 5.30 Typical nonuniform distribution of address references.

How can one get fast memory with less expense?

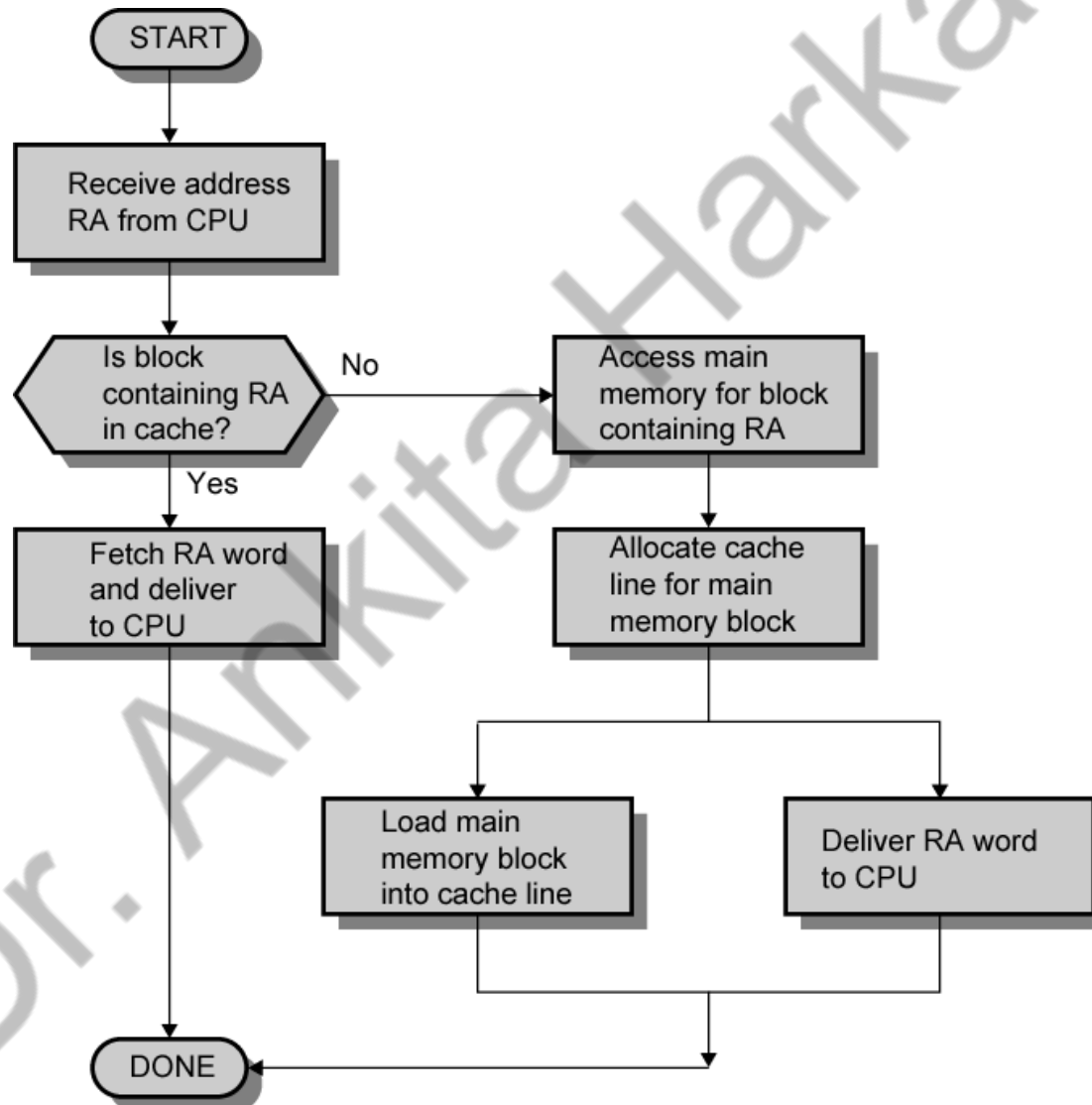
- It is possible to build a computer which uses only static RAM (large capacity of fast memory)
 - This would be a very fast computer
 - But, this would be very costly
- It also can be built using a small fast memory for present reads and writes.
 - Add a Cache memory

Cache Memory Organization

- Cache - Small amount of fast memory
 - Sits between normal main memory and CPU
 - May be located on CPU chip or in system
 - Objective is to make slower memory system look like fast memory.



Cache Read Operation - Flowchart



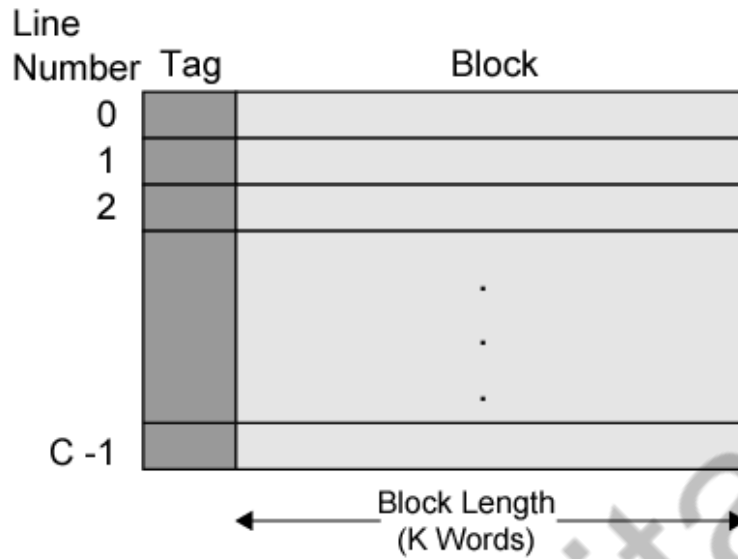
Cache Design Parameters

- Size of Cache
- Size of Blocks in Cache
- Mapping Function – how to assign blocks
- Write Policy - Replacement Algorithm when blocks need to be replaced

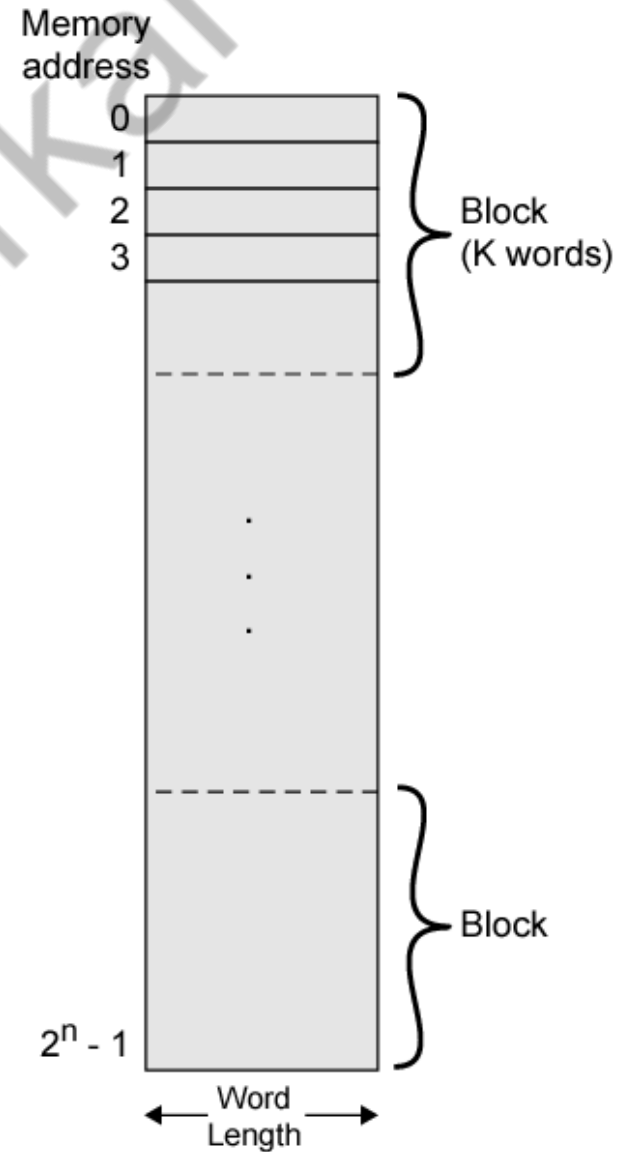
Size Does Matter

- Cost
 - More cache is expensive
- Speed
 - More cache is faster (up to a point)
 - Checking cache for data takes time

Cache/Main Direct Caching Memory Structure



(a) Cache



(b) Main memory

Write Policy Challenges

- Must not overwrite a cache block unless main memory is correct
- Multiple CPUs/Processes may have the block cached
- I/O may address main memory directly ?
(may not allow I/O buffers to be cached)

Write through

- All writes go to main memory as well as cache

(Typically 15% or less of memory references are writes)

Challenges:

- Multiple CPUs *MUST* monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic – may cause bottlenecks
- Potentially slows down writes

Write back

- Updates initially made in cache only
(Update bit for cache slot is set when update occurs – Other caches must be updated)
- If block is to be replaced, memory overwritten only if update bit is set
(15% or less of memory references are writes)
- I/O must access main memory through cache or update cache

Comparison of Cache Sizes

Processor	Type	Year of Introduction	Primary cache (L1)	2 nd level Cache (L2)	3 rd level Cache (L3)
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA _b	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

UNIT 6

- An Overview of Parallel Processing
- Organization of Multiprocessor
 - Flynn's Classification
 - System Topologies
- Multiprocessors
 - Symmetric Multiprocessors
 - Asymmetric Multiprocessors
- Pipelining
 - Construction
 - Systolic Arrays
- Vector Processors



Parallel Processing

Dr. Ankita Markare

An Overview of Parallel Processing

What is parallel processing?

- Parallel processing is a method to improve computer system performance by executing two or more instructions simultaneously.

The goals of parallel processing.

- One goal is to reduce the “wall-clock” time or the amount of real time that you need to wait for a problem to be solved.
- Another goal is to solve bigger problems that might not fit in the limited memory of a single CPU.

An Analogy of Parallelism



The task of ordering a shuffled deck of cards by suit and then by rank can be done faster if the task is carried out by two or more people. By splitting up the decks and performing the instructions simultaneously, then at the end combining the partial solutions you have performed parallel processing.

Parallelism in Uniprocessor Systems

- It is possible to achieve parallelism with a uniprocessor system.
 - Some examples are the instruction pipeline, arithmetic pipeline, I/O processor.
- Note that a system that performs different operations on the same instruction is not considered parallel.
- Only if the system processes two different instructions simultaneously can it be considered parallel.

Parallelism in a Uniprocessor System

A reconfigurable arithmetic pipeline is an example of parallelism in a uniprocessor system.

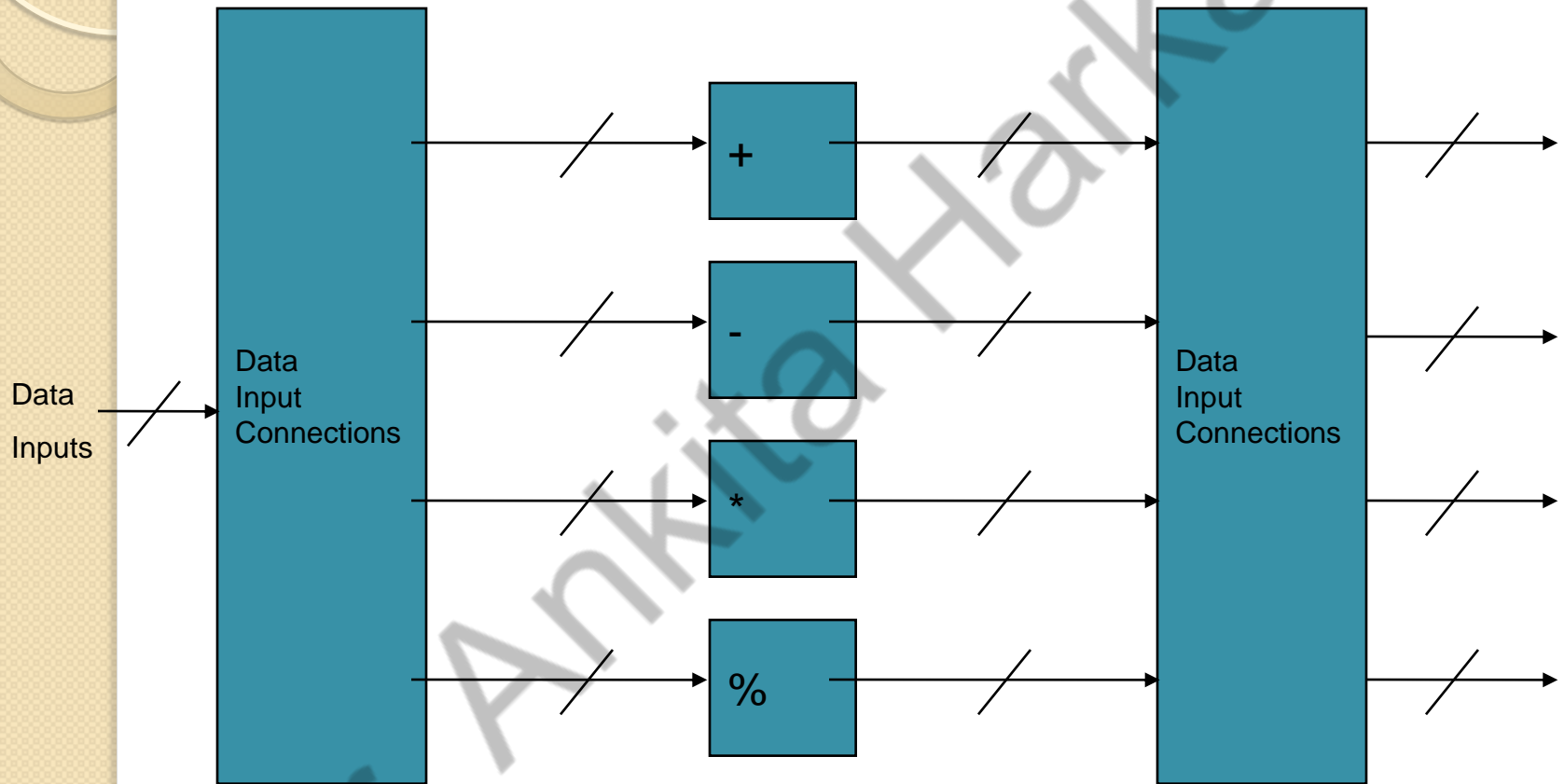
Each stage of a reconfigurable arithmetic pipeline has a multiplexer at its input. The multiplexer may pass input data, or the data output from other stages, to the stage inputs. The control unit of the CPU sets the select signals of the multiplexer to control the flow of data, thus configuring the pipeline.

Vector Arithmetic Unit

A vector arithmetic unit contains multiple functional units that perform addition, subtraction, and other functions. The control unit routes input values to the different functional units to allow the CPU to execute multiple instructions simultaneously.

For the operations $A \leftarrow B + C$ and $D \leftarrow E - F$, the CPU would route B and C to an adder and then route E and F to a subtractor for simultaneous execution.

A Vectored Arithmetic Unit



$A \leftarrow B + C$

$D \leftarrow E - F$



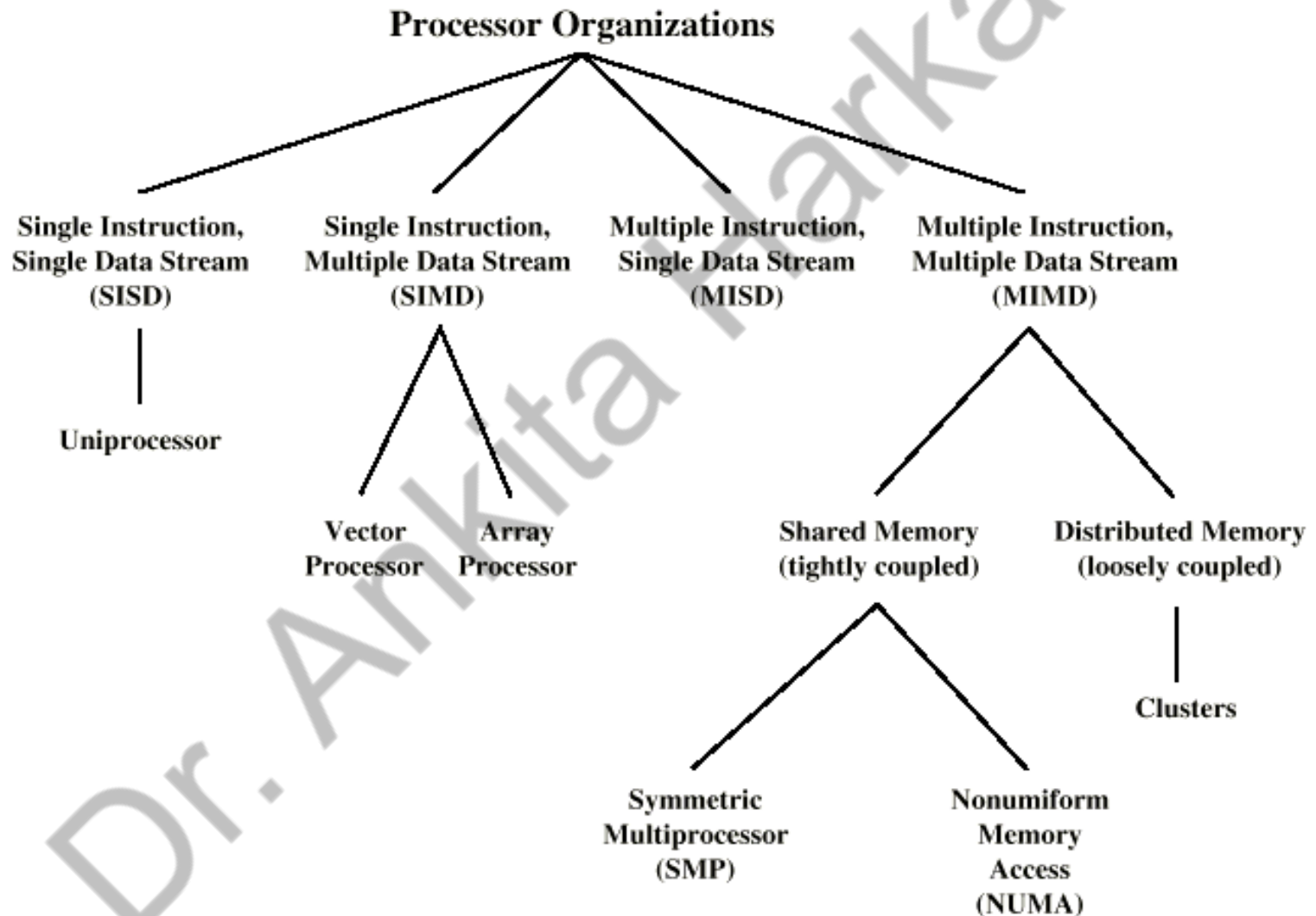
Organization of Multiprocessors

Dr. Ankita Markare

Organization of Multiprocessor Systems

- Flynn's Classification
 - Was proposed by researcher Michael J. Flynn in 1966.
 - It is the most commonly accepted taxonomy of computer organization.
 - In this classification, computers are classified by whether it processes a single instruction at a time or multiple instructions simultaneously, and whether it operates on one or multiple data sets.

Taxonomy of Parallel Processor Architectures

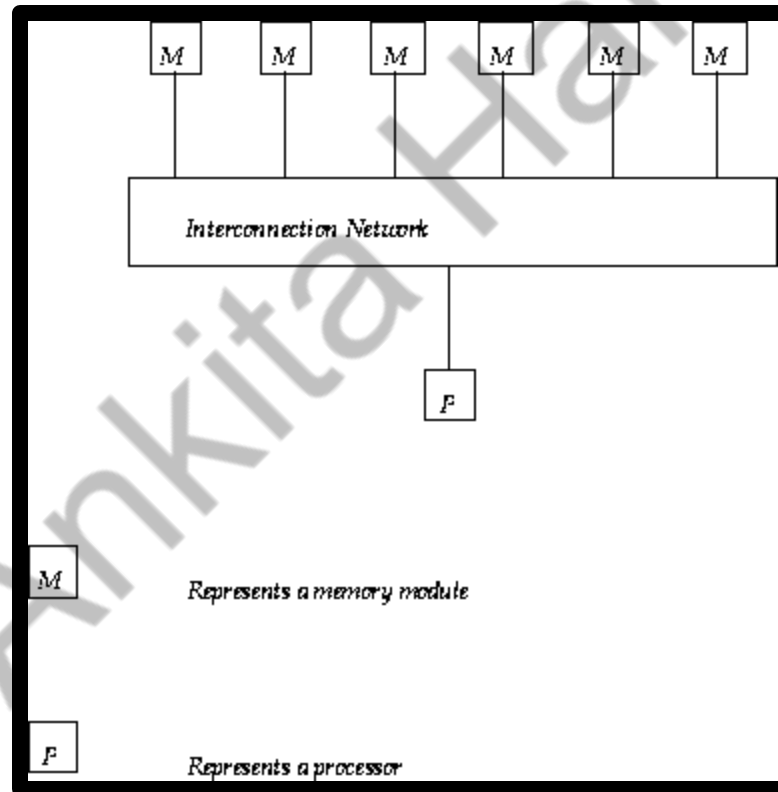


Single Instruction, Single Data (SISD)

- SISD machines executes a single instruction on individual data values using a single processor.
- Based on traditional Von Neumann uniprocessor architecture, instructions are executed sequentially or serially, one step after the next.
- Until most recently, most computers are of SISD type.

SISD

Simple Diagrammatic Representation

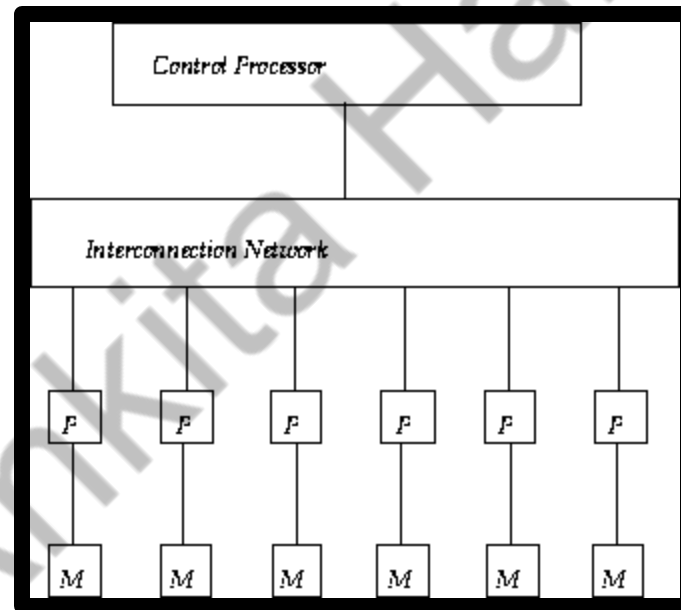


Single Instruction, Multiple Data (SIMD)

- An SIMD machine executes a single instruction on multiple data values simultaneously using many processors.
- Since there is only one instruction, each processor does not have to fetch and decode each instruction. Instead, a single control unit does the fetch and decoding for all processors.
- SIMD architectures include array processors.

SIMD

Simple Diagrammatic Representation

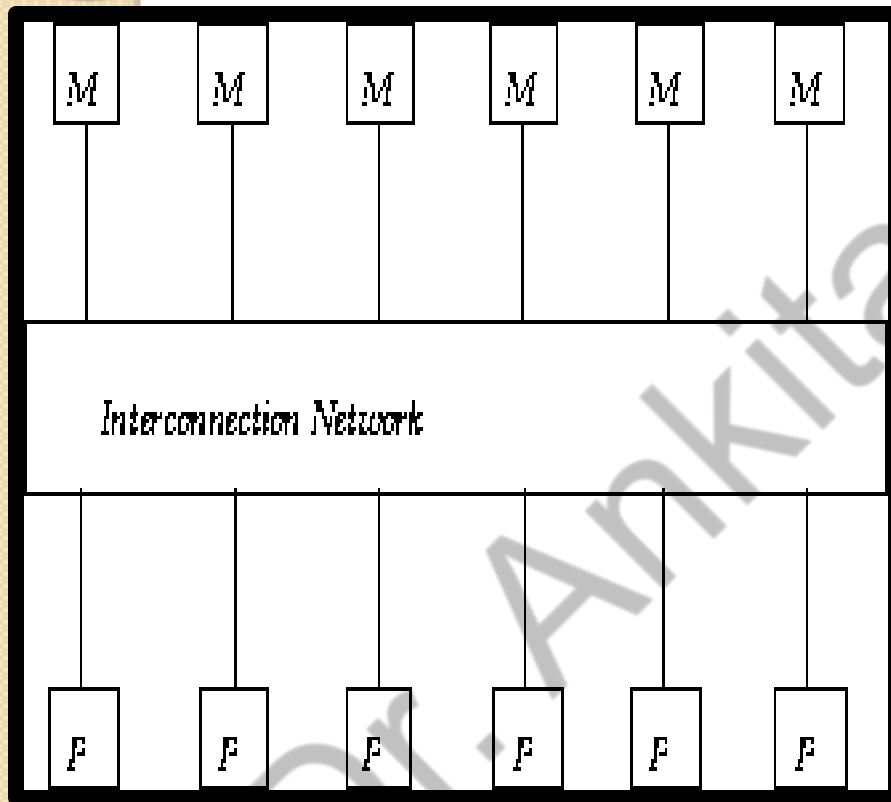


Multiple Instruction, Multiple Data (MIMD)

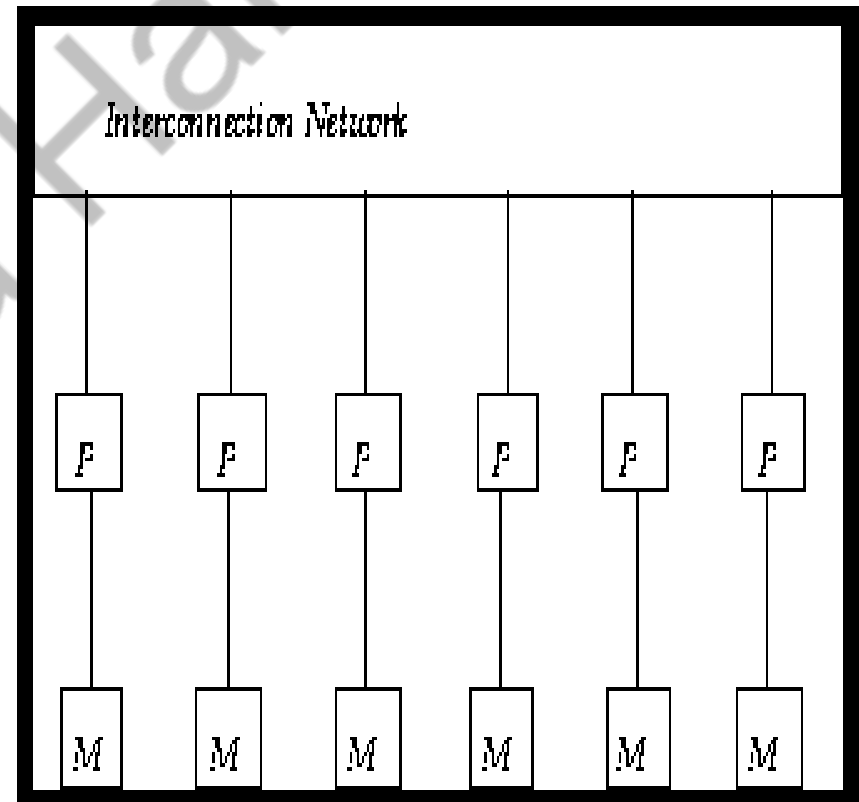
- MIMD machines are usually referred to as multiprocessors or multicomputers.
- It may execute multiple instructions simultaneously, contrary to SIMD machines.
- Each processor must include its own control unit that will assign to the processors parts of a task or a separate task.
- It has two subclasses: Shared memory and distributed memory

MIMD

Simple Diagrammatic Representation
(Shared Memory)



Simple Diagrammatic Representation
(Distributed Memory)



Multiple Instruction, Single Data (MISD)

- This category does not actually exist. This category was included in the taxonomy for the sake of completeness.



Multiprocessors

Dr. Ankita Harkare

Why Choose a Multiprocessor?

- A single CPU can only go so fast, use more than one CPU to improve performance
- Multiple users
- Multiple applications
- Multi-tasking within an application
- Responsiveness and/or throughput
- Share hardware between CPUs

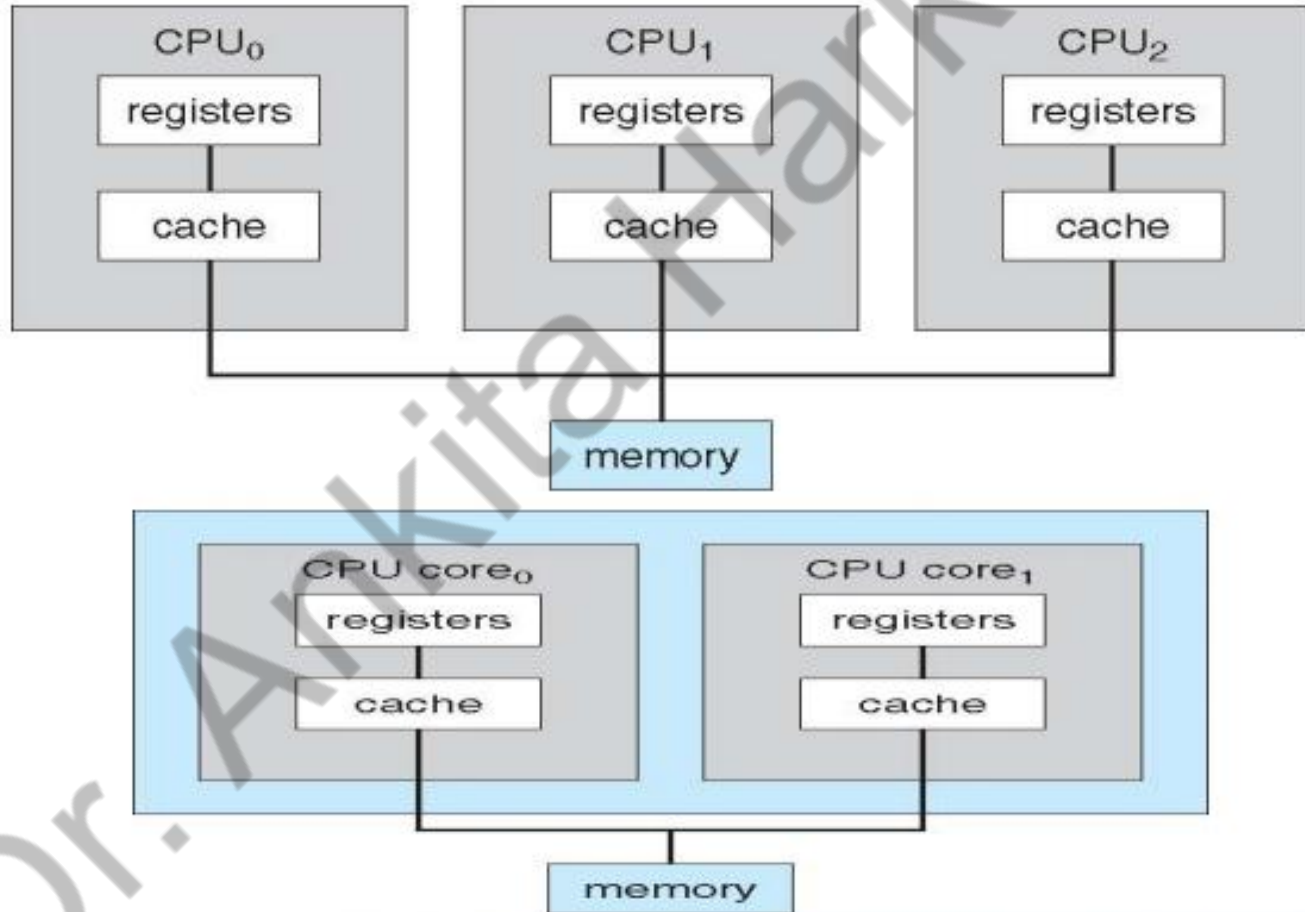
Multiprocessor Symmetry

- In a multiprocessing system, all CPUs may be equal, or some may be reserved for special purposes.
- A combination of hardware and operating-system software design considerations determine the symmetry.
- Systems that treat all CPUs equally are called symmetric multiprocessing (**SMP**) systems.
- If all CPUs are not equal, system resources may be divided in a number of ways, including asymmetric multiprocessing (**ASMP**), non-uniform memory access (**NUMA**) multiprocessing, and clustered multiprocessing.

Multiprocessor Systems

- Parallel Systems / Tightly coupled systems
- More than one processor in close communication, sharing the computer bus, the clock sometimes memory and IO devices.
- Advantages
 - Increased throughput
 - Economy of scale
 - Increased reliability

Multiprocessor Architecture



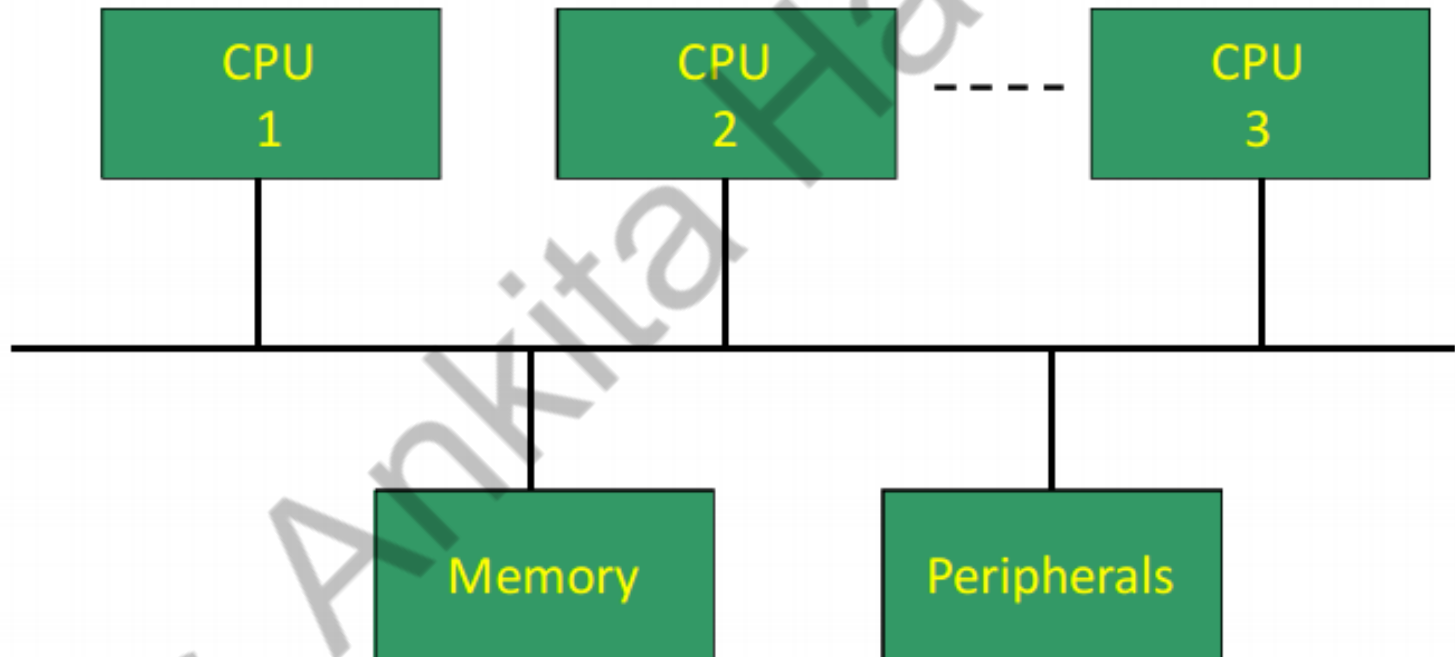
Types of Multiprocessor Systems

- **Asymmetric Multiprocessing**
 - master processor schedules and allocates work to slave processors.
- **Symmetric Multiprocessing (SMP)**
 - Each processor runs an identical copy of the operating system.
 - Typically each processor does self-scheduling from the pool of available process.
 - Most modern operating systems support SMP.

Symmetric Multiprocessing (SMP)

- Each processor can perform the same functions and share same main memory and I/O facilities (symmetric).
- The OS schedule processes/threads across all the processors (real parallelism).
- Existence of multiple processors is transparent to the user.
- Incremental growth: just add another CPU!
- Robustness: a single CPU failure does not halt the system, only the performance is reduced.

Symmetric Multiprocessing



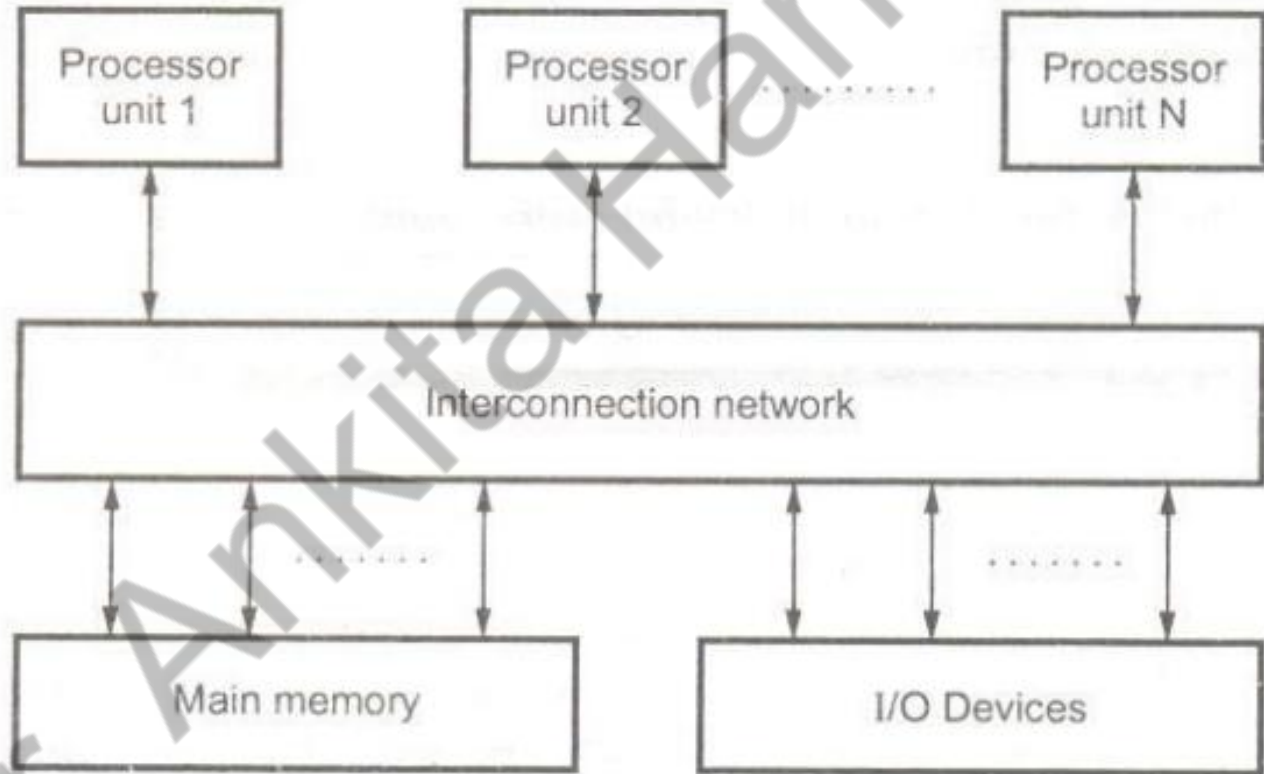
Characteristics

- Two or more similar processes are employed in a stand-alone system.
- Processors share same memory and I/O facilities.
- Processors accesses memory and I/O in approx. equal amount of time.
- Processors are capable of performing the same functions.
- The OS controls the interaction between processors and their programs at different levels.

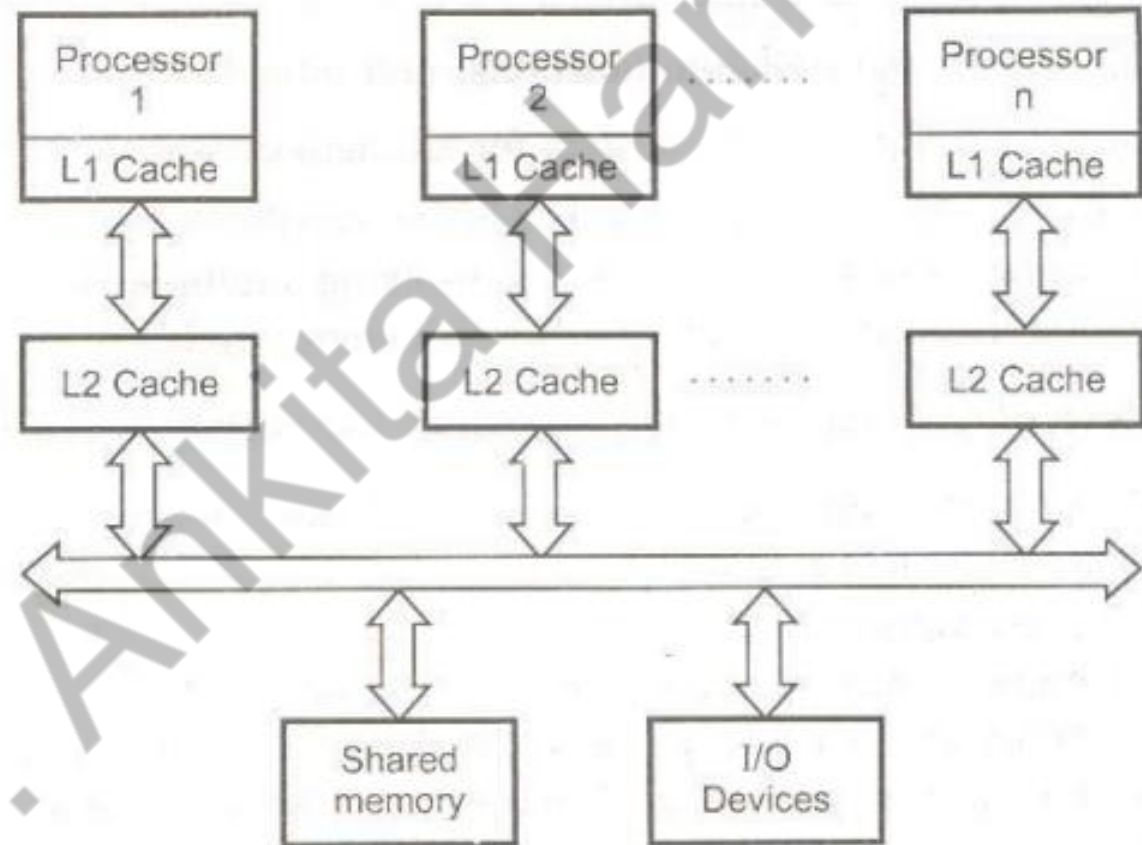
SMP Advantages

- Performance
 - If some work can be done in parallel
- Availability
 - Since all processors can perform the same functions, failure of a single processor does not halt the system
- Incremental growth
 - User can enhance performance by adding additional processors
- Scaling
 - Vendors can offer range of products based on number of processors

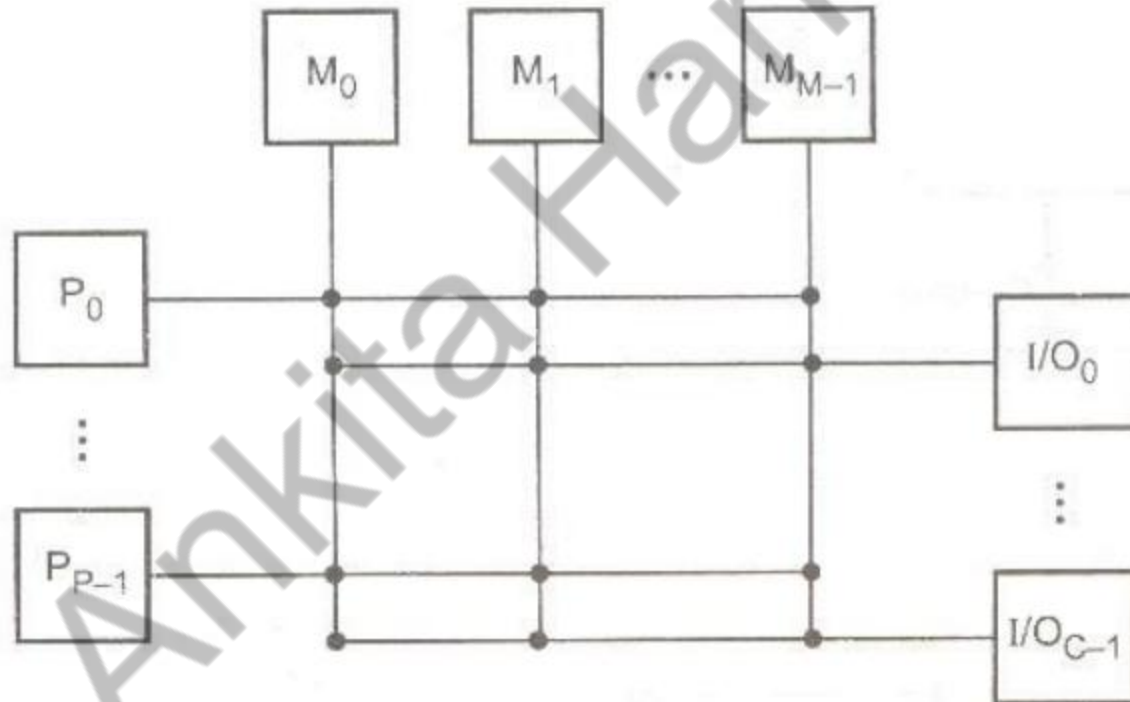
General architecture



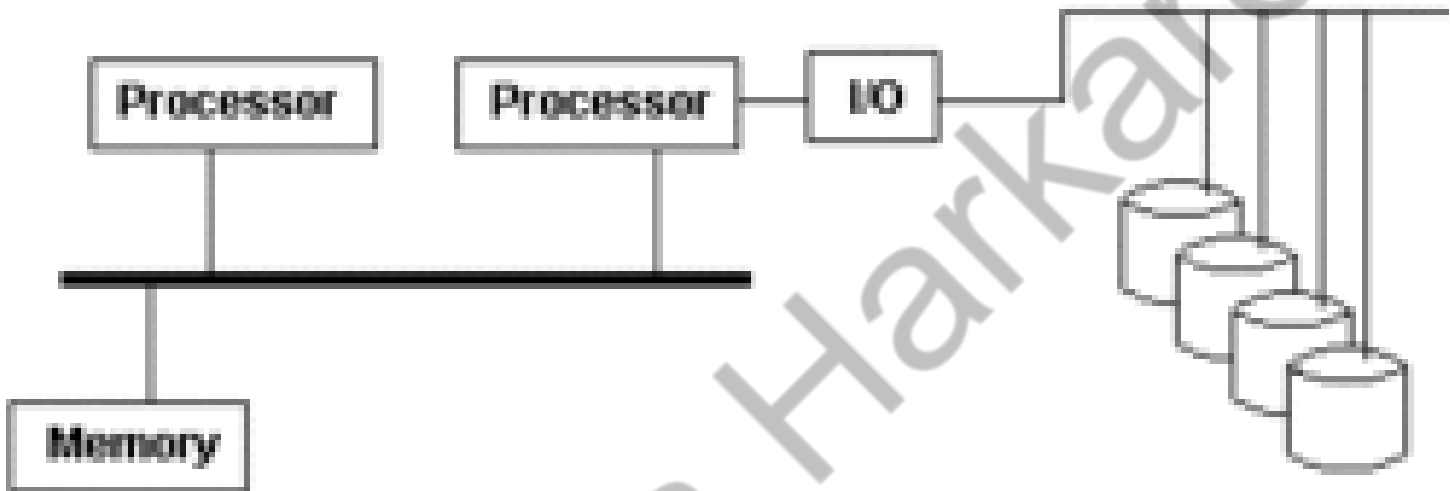
SMP with time shared bus




Crossbar switch system



Asymmetric multiprocessing (AMP) was a software stopgap for handling multiple CPUs before symmetric multiprocessing (SMP) was available. It has also been used to provide less expensive options on systems where SMP was available. In an asymmetric multiprocessing system, not all CPUs are treated equally; for example, a system might only allow (either at the hardware or operating systems level) one CPU to execute operating system code or might only allow one CPU to perform I/O operations. Other AMP systems would allow any CPU to execute operating system code and perform I/O operations, so that they were symmetric with regard to processor roles, but attached some or all peripherals to particular CPUs, so that they were asymmetric with regard to peripheral attachment.



Dr. Ankita Harkare



Multiprocessor OS design considerations

- Simultaneous concurrent processes.
- Scheduling
- Synchronization
- Memory management
- Reliability and fault tolerance



Pipelining

Dr. Ankita Harkare

Characterize Pipelines

- Hardware or software implementation – pipelining can be implemented in either software or hardware.
- Large or Small Scale – Stations in a pipeline can range from simplistic to powerful, and a pipeline can range in length from short to long.
- Synchronous or asynchronous flow – A synchronous pipeline operates like an assembly line: at a given time, each station is processing some amount of information. A asynchronous pipeline, allow a station to forward information at any time.
- Buffered or unbuffered flow – One stage of pipeline sends data directly to another one or a buffer is place between each pairs of stages.
- Finite Chunks or Continuous Bit Streams – The digital information that passes though a pipeline can consist of a sequence or small data items or an arbitrarily long bit stream.
- Automatic Data Feed Or Manual Data Feed – Some implementations of pipelines use a separate mechanism to move information, and other implementations require each stage to participate in moving information.

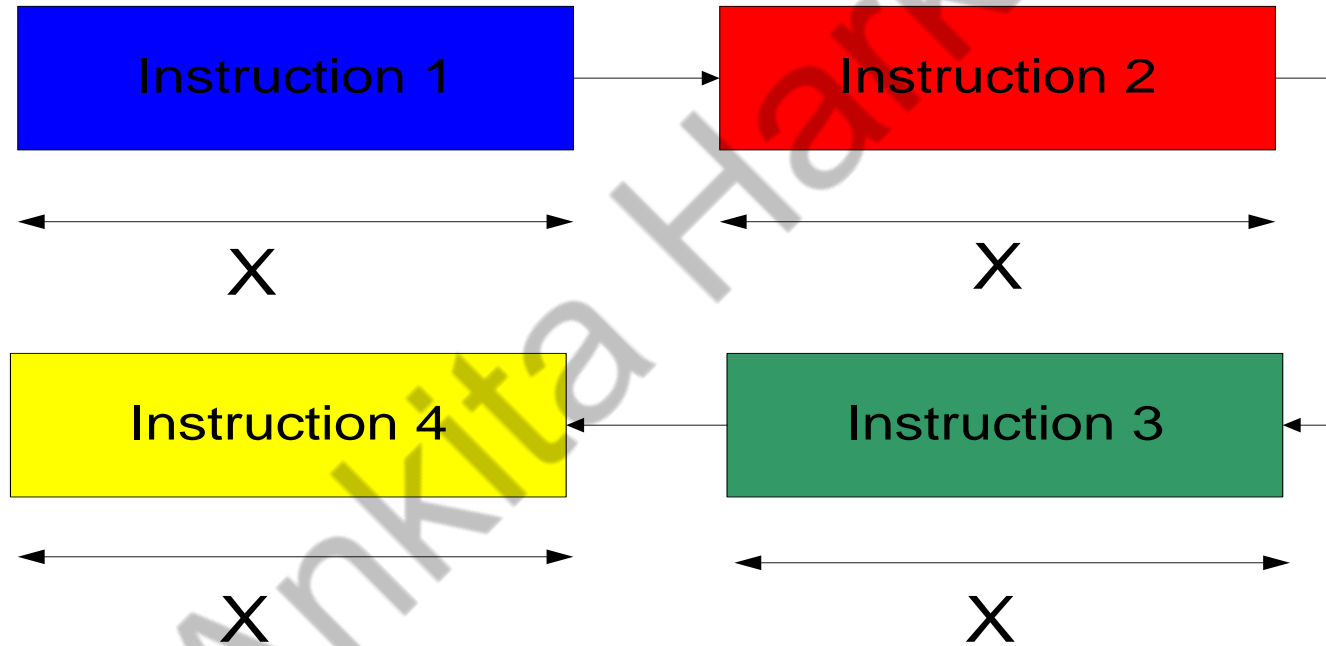
What is Pipelining

- A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed.
- A Pipeline is a series of stages, where some work is done at each stage. The work is not finished until it has passed through all stages.
- With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor until each instruction operation can be performed.

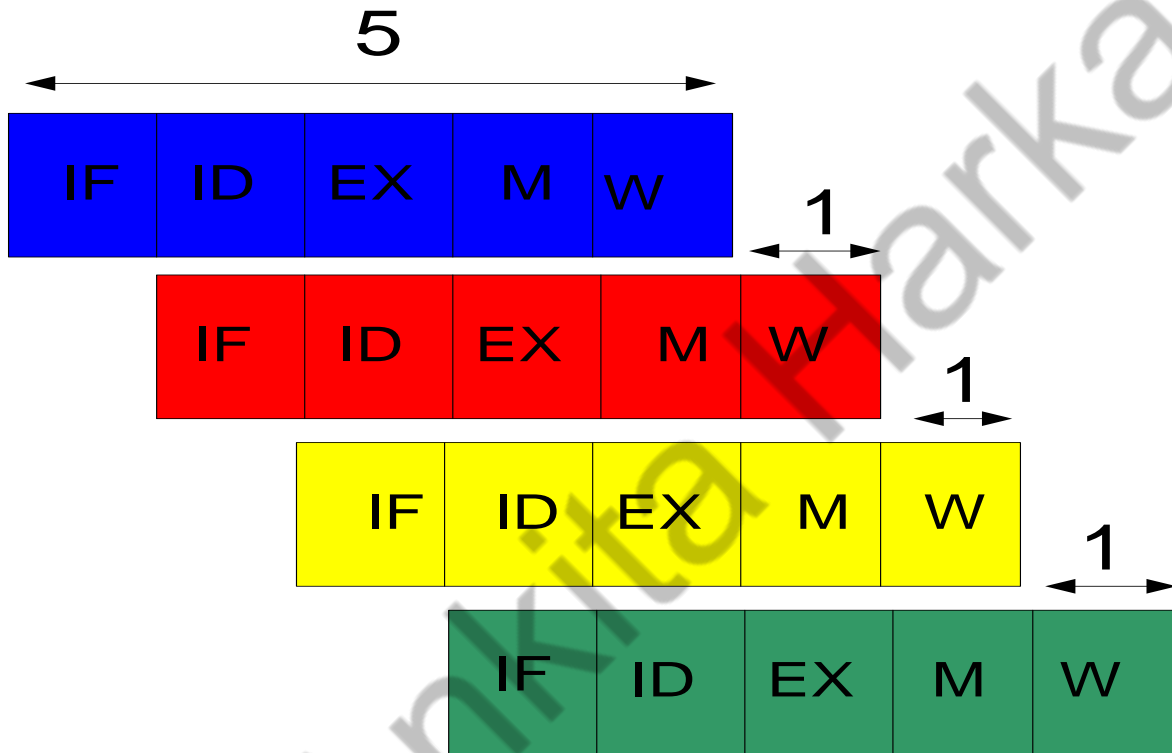
How Pipelines Works

- The pipeline is divided into segments and each segment can execute its operation concurrently with the other segments. Once a segment completes an operation, it passes the result to the next segment in the pipeline and fetches the next operations from the preceding segment.

Example



Four sample instructions, executed linearly



Four Pipelined Instructions

Instructions Fetch

- The instruction Fetch (IF) stage is responsible for obtaining the requested instruction from memory. The instruction and the program counter (which is incremented to the next instruction) are stored in the IF/ID pipeline register as temporary storage so that may be used in the next stage at the start of the next clock cycle.

Instruction Decode

- The Instruction Decode (ID) stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor. The instruction is sent to the control unit where it is decoded and the registers are fetched from the register file.

Execution

- The Execution (EX) stage is where any calculations are performed. The main component in this stage is the ALU. The ALU is made up of arithmetic, logic and capabilities.

Memory and IO

- The Memory and IO (MEM) stage is responsible for storing and loading values to and from memory. It also responsible for input or output from the processor. If the current instruction is not of Memory or IO type than the result from the ALU is passed through to the write back stage.

Write Back

- The Write Back (WB) stage is responsible for writing the result of a calculation, memory access or input into the register file.

Operation Timings

- Estimated timings for each of the stages:

Instruction Fetch	2ns
Instruction Decode	1ns
Execution	2ns
Memory and IO	2ns
Write Back	1ns

Arithmetic Pipelining

- The pipeline structures used for instruction pipelining may be applied in some cases to other processing tasks. If pipelining is to be useful, however, we must be faced with the need to perform a long sequence of essentially similar tasks. Large numerical applications often make use of repeated arithmetic operations for processing the elements of vectors and arrays. Architectures specialized for applications of this type often provide pipelines to speed processing of floating-point arithmetic sequences. This type of pipelining is called **arithmetic pipelining**.
- Arithmetic pipelines differ from instruction pipelines in some important ways. They are generally **synchronous**. This means that each stage executes in a fixed number of clock cycles. In a synchronous pipeline, moreover, no buffering between stages is provided. Each stage must be ready to accept the data passed from a previous stage when that data is produced.
- Another important difference is that an arithmetic pipeline may be **nonlinear**. The "stages" in this type of pipeline are associated with key processing components such as adders, shifters, etc. Instead of a steady progression through a fixed sequence of stages, a task in a nonlinear pipeline may use more than one stage at a time, or may return to the same stage at several points in processing.

Arithmetic Pipelining

- As an example of a pipelined arithmetic unit we consider a floating point adder. This pipeline accepts as input two normalized floating point numbers of the form:

- $A = a \times 2^p$

- $B = b \times 2^q$

Here a and b are 2's complement fractions in the range $0.5 < f < 1.0$. p and q are corresponding base 2 exponents. The normalized sum is to be computed. Four stages can be identified for this pipeline:

- Input the original fractions and exponents. Compute the larger exponent and the exponent difference. Shift the fraction corresponding to the smaller exponent right for a number of places equal to the difference. Both fractions are now adjusted to match the same (larger) exponent. Output the exponent and the two fractions.
- Add the two fractions, producing a sum. Pass through the exponent unchanged. Output the exponent and fractions.
- Count leading zeros in the result fraction. Shift the fraction to normalize. Output the original exponent, the fraction, and the count.
- Add the exponent and count. Output the adjusted exponent and the normalized fraction

Advantages/Disadvantages

Advantages:

- More efficient use of processor
- Quicker time of execution of large number of instructions

Disadvantages:

- Pipelining involves adding hardware to the chip
- Inability to continuously run the pipeline at full speed because of pipeline hazards which disrupt the smooth execution of the pipeline.

Systolic Architectures

- Basic principle: Replace a single PE with a regular array of PEs and carefully orchestrate flow of data between the PEs → achieve high throughput w/o increasing memory bandwidth requirements

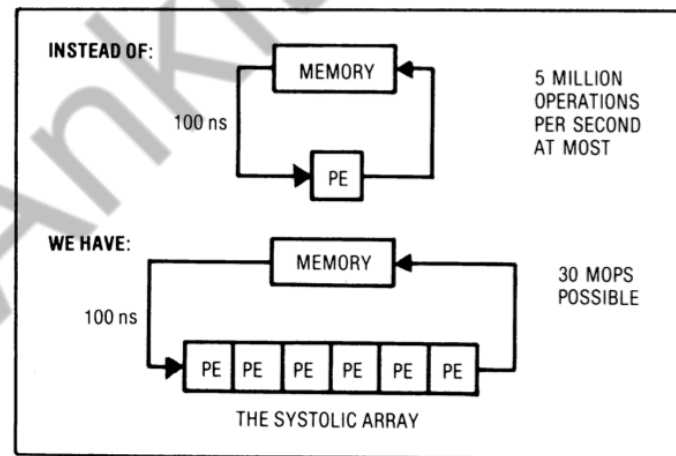


Figure 1. Basic principle of a systolic system.

Memory: heart
PEs: cells
PE: Processing Element

Memory pulses
data through
cells

Systolic Architectures

- Differences from pipelining:
 - Array structure can be non-linear and multi-dimensional
 - PE connections can be multidirectional (and different speed)
 - PEs can have local memory and execute kernels (rather than a piece of the instruction)



Vector Processors

Dr. Ankita Harkare

Overview

- History
- Description
- Advantages
- Disadvantages
- Applications
- Conclusions

What is a Vector Processor?

- Also called an Array Processor.
- Runs multiple mathematical operations on multiple data elements simultaneously.
- Common in supercomputers of the 1970's 80's and 90's.
- Today most CPU designs contains at least some vector processing instructions, typically referred to as SIMD.
- Typically operate on a few vectors elements per clock cycle in a pipeline.
- Common examples include VIS, MMX, SSE, AltiVec and AVX

History

- 1962 University of Illinois Illiac IV - completed 1972 with 64 ALUs 100-150 MFlops (massively parallel computer)
- (1973) TI's Advance Scientific Computer (ASC) 20-80 MFlops
- (1975) Cray-1 first to have vector registers instead of keeping data in memory (8 registers with 64 64-bit words in each)
- Cray-1 had separate pipelines for different instruction types allowing vector chaining. 80-240 MFlops

Advantages

- Each result is independent of previous results - allowing deep pipelines and high clock rates.
- A single vector instruction performs a great deal of work - meaning less fetches and fewer branches (and in turn fewer mispredictions).
- Vector instructions access memory a block at a time which allows memory latency to be amortized over many elements.
- Vector instructions access memory with known patterns, which allows multiple memory banks to simultaneously supply operands.
- Less memory access = faster processing time.

Disadvantages

- Not as fast with scalar instructions
- Complexity of the multi-ported VRF
- Difficulties implementing precise exceptions
- High price of on-chip vector memory systems
- Increased code complexity

Applications

- Servers
- Home Cinema
- Super Computing
- Cluster Computing
- Mainframes
- “Astrophysicist Replaces Supercomputer With 8 PS3’s” ²

Dr. Ankita Harkare

Thank You



Dr. Ankita Harkare



Dr. Ankita Harkare



Dr. Ankita Harkare



Dr. Ankita Harkare



Dr. Ankita Harkare