High performance computing

what is high performance computing & how can do it?

- HPC - divide up the work among numerous linked system.

why HPC - Scientific simulation & modelling drive the need for greater computing power.

- single core processor can not be made that have enough resource for the simulation needed.

~~Computer architecture~~

Flynn's classification ⇒ michael Flynn (1972) introduced a classification of various computer architectures based on notions of instruction & data streams.

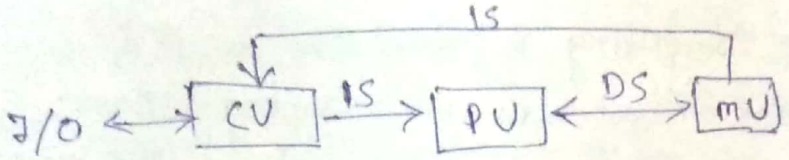
↳ is abstracted by its instruction set, which includes opcode, addressing modes, register; virtual memory etc.

He has classified computer architecture mainly into four catogery.

1- SISD (Single instruction stream over a single data stream)
2 SIMD (" " over multiple ")
3 - MISD (multiple " " " single ")
4 - MIMD. (multiple " " " multiple ")

multiple abbreviation. - IS - instruction set
DS - Data "
MM - memory module
SM - shared memory
CU - control unit
PU - processor unit.

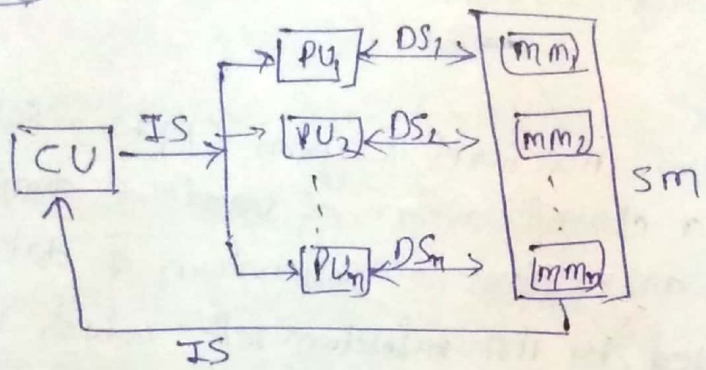
1- SISD Uniprocessor Architecture.



- Conventional sequential m/c are called SISD computer.

  ex. IBM 701, IBM 1620 etc

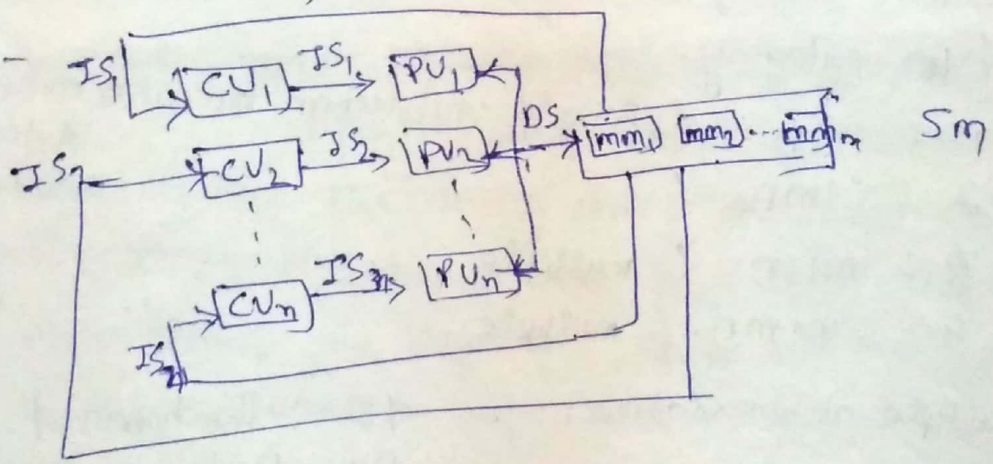
2- SIMD



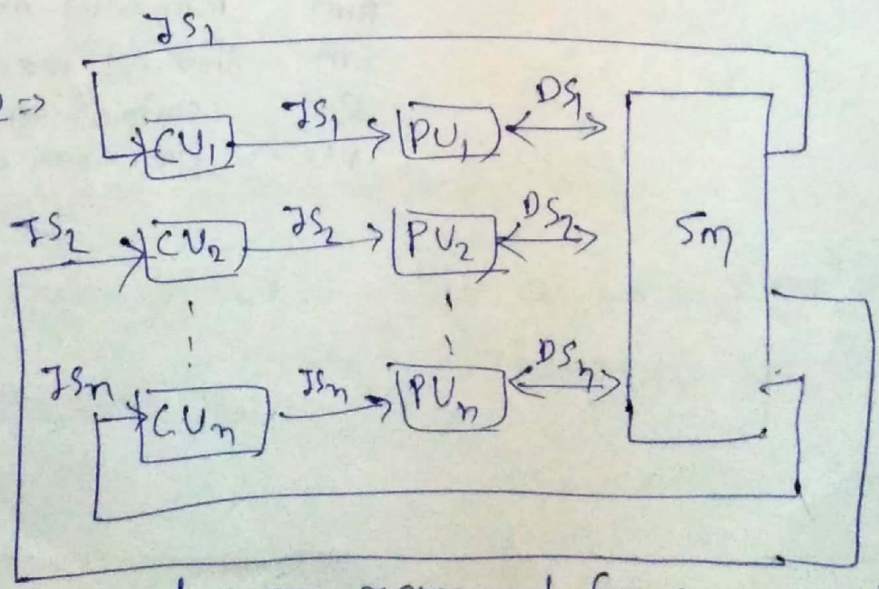
- Vector computers are equiped with scalar & vector h/w

  ex- Illiac IV, STARAN

3- MISD-



(4) MIMD =>



Parallel computer are reserved for MIMD m/c

Kinds of parallelism ⇒

① Data parallelism ⇒ The same task run on different data in parallel. → SIMD

Ex- convert all characters in an array to upper case

② Task parallelism ⇒ different tasks running on the same data. → MISD

Ex- average, minimum, binary & geometric mean.

- no dependencies b/w the task, so all can run in parallel

③ Hybrid data/task parallelism ⇒

- A parallel pipeline of tasks, each of which might be data parallel. (F, D, OF, E)

Dependencies & hazards ⇒

what is dependency ⇒ If two instructions are parallel, they can execute simultaneously in a pipline of ~~data~~ arbitrary length without causing any stalls, assuming the pipline has sufficient resources. if two instructions are dependent, they are not parallel and must be executed in sequential order.

There are Three types of dependecies.

① Data dependence
② Control dependence
③ Resource dependence.

① Data dependence ⇒ The ordering relationship b/w statements is indicated by the data dependence.

five type of data dependence :-

① flow dependence :- $S_1 \rightarrow S_2$

① the control will be coming from S_1 to S_2
② O/P of S_1 will be the I/P of S_2

A statement S_2 is flow dependent on statement S_1 if an execution path exists from S_1 to S_2 and if at least one output of S_1 feeds in an input to S_2

② Antidependence ⇒ $S_1 \not\rightarrow S_2$.

Statement S_2 is antidependent on statement S_1 if S_2 follows S_1 in program order and if the O/P of S_2 overlaps the input to S_1.

⇒ ① flow coming from S_1 to S_2
② I/P of S_1 will be the O/P of S_2

③ O/P dependency ⇒ Two statements are O/P dependent if they produce the same O/P.

– denoted by $S_1 \xrightarrow{O} S_2$

④ I/O dependence ⇒ $S_1 \xrightarrow{I/O} S_2$

Read and write are I/O statements. I/O dependence occure not because the same variable is involved but because the same file is offered by both I/O statements.

⇒ If both the devices are trying to access the same I/O device for read & write operation

ex. S_1 : Read (4), A(I) // Read Array A from tape unit 4

S_2 : Rewind (4) // Rewind tape unit 4

S_3 : write (4), B(I) // write Array B, in the tape unit 4

S_4 : Rewind (4) // Rewind tape unit 4

$S_1 \xrightarrow{I/O} S_2$

5 **Unknown dependence** :- The dependence relation b/w two statements cannot be determined in the following situation.

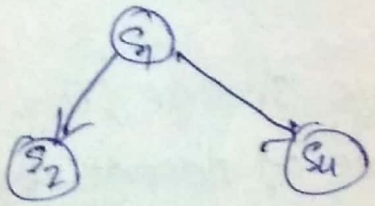
- The subscript of a variable is itself subscribed.
- The subscript does not contain the loop index variable.
- A variable appears more than once with subscripts having different coefficients of the loop variable.
- The subscript is non linear in the loop index variable.

When one or more of these conditions exist, a conservative assumption is to claim unknown dependence among the statement involved.

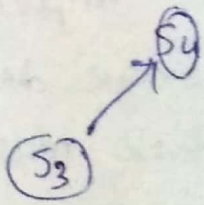
~~Control Dependence~~ ⇒ Ex ⇒ Dependency graph.

S_1 : Load R_1, A // $R_1 \leftarrow$ memory (A)
S_2 : Add R_2, R_1 // $R_2 \leftarrow (R_2) + (R_1)$
S_3 : move R_1, R_3 // $R_1 \leftarrow (R_3)$
S_4 : Store B, R_1 // memory (B) $\leftarrow R_1$

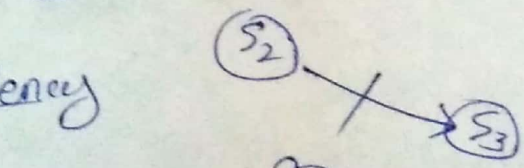
1



⇒

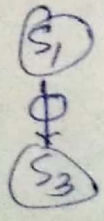


Flow dependency



Anti dependency

O/P dependency



~~Dependen~~

Control dependency ⇒ This refers to the situation where the order of execution of statements can not be determine before run time. for example, conditional statements will not be resolved until run time.

eg ⇒

```
ADD   R1, R1, R2
BEQ   R1, R3, Label
ADD   R2, R3, R4
SUB   R5, R6, R8
  :
Label:
MUL   R5, R6, R8
  :
```

Resource Dependency ⇒ This is different from data or control dependence, which demands the independence of the work to be done.

- Resource dependence is concerned with the conflicts in using shared resources, such as integer units, floating point units, registers & memory areas, among parallel events.
- when the conflicting resource is an ALU, we call it ALU dependence.
- if the conflicts involve workplace storage, we call it storage dependence. In the case of storage dependence, each task must work on independent storage locations or use ~~access~~ protected access to shared writable data.

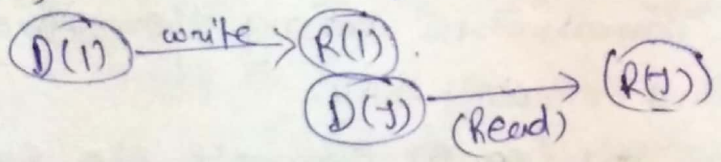
Hazard ⇒ The situation when a dependence can cause a problem in a pipeline is called hazard

- So hazard occures when dependence result in incorrect execution.

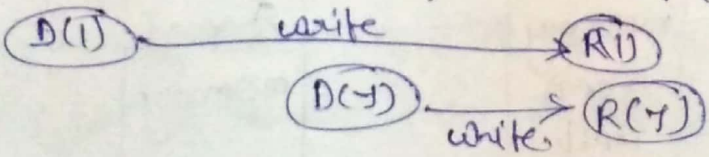
- The Three types of hazard are possible } Due to read & write of shared variables by different instruction

- Consider two instruction I & J. instruction J is assumed to logically follow instruction I according to program order.

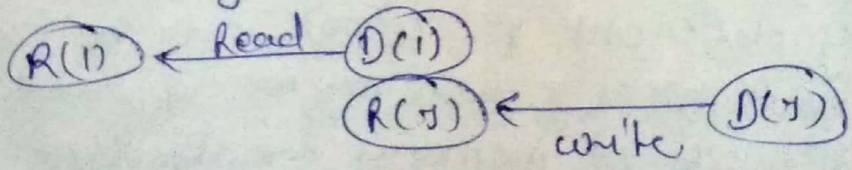
(1) RAW (Read after write): J tries to read a source before i writes it, so J incorrectly gets old value, this hazard is due to true data dependency or flow dependency.



(2) WAW (write after write): J tries to write an operand before it is written by i.
WAW hazard causes from output dependence.



(3) WAR (write after Read): J tries to write a destination before it is read by i, so that I incorrectly get the new value. It occures due to antidependency.



$R(I) \cap D(J) \neq \phi$ for RAW hazard

$R(J) \cap R(J) \neq \phi$ for WAW "

$D(I) \cap R(J) \neq \phi$ for WAR "

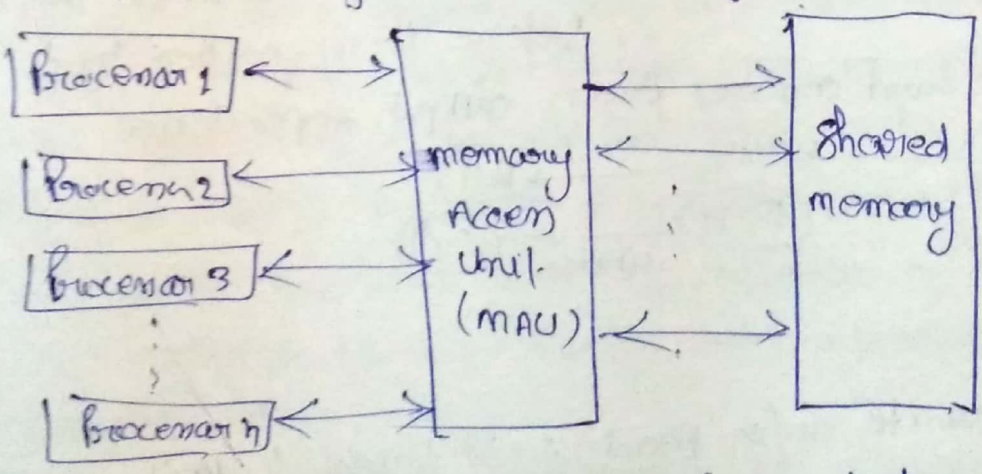
These conditions are necessary but not sufficient.

notation $D(I) \rightarrow$ Domain $(J) \rightarrow$ input set

$R(I) \rightarrow$ Range $(J) \rightarrow$ O/p set

PRAM models ⇒ is a model, which is considered for most of the parallel algorithms. Here multiple processors are attached to a single block of memory. A PRAM model contains —

- A set of similar type of processors.
- All the processors share a common memory unit.
- Processors can communicate among themselves through the shared memory only.
- A memory access unit (MAU) connects the processors with the single shared memory.



Here, n number of processors can perform independent operations on n number of data in a particular unit unit of time. This may result in simultaneous access of same memory location by different processors. with the shared memory, the model must specify how concurrent read and concurrent write of memory are handled.

four memory-update option are possible.

- Exclusive read (ER) ⇒ This allows at most one processor to read from any memory location in each cycle.
- Exclusive write (EW) ⇒ This allows at most one processor to write into a memory location at a time.
- Concurrent read (CR) ⇒ This allows multiple processor to read the same information from the same memory cell in the same cycle.
- Concurrent write (CW) ⇒ This allows simultaneous writes to the same memory location.

In order to avoid confusion, some policy must be set up to resolve the write conflicts.

Various combinations of the above options lead to several variants of the PRAM model as specified below.

PRAM variants ⇒ Described below are four variants of the PRAM model, depending on how the memory reads and writes are handled.

1. EREW ⇒ Here no two processors are allowed to read from or write to the same memory location at the same time. This is the most restrictive PRAM model
2. ERCW ⇒ Here no two processors are allowed to read from the same memory location at the same time, but are allowed to write to the same memory location at the same time.
3. CREW ⇒ Here all the processors are allowed to read from the same memory location at the same time, but are not allowed to write to the same memory location at the same time.
4. CRCW ⇒ All the processors are allowed to read from or write to the same memory location at the same time.

~~There exist~~ The read causes no discrepancies while the concurrent write is further defined as:

Common - all processors ~~write the same value~~ are allowed to complete write if all the values to be written are equal. Any algorithm for this model has to make sure that this condition is satisfied. if not, the algorithm is illegal and the m/c state will be undefined.

Priority - the processors are assigned fixed distinct priorities and the processor with the highest priority is allowed to complete write.

Arbitrary - one randomly chosen processor is allowed to complete write. The algorithm may make no assumptions about processor was chosen.

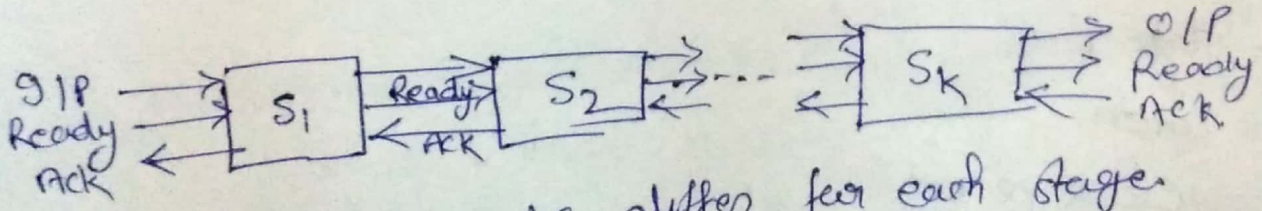
Pipelining and superscalar Techniques.

Linear pipeline processors ⇒ A linear pipeline processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other. For modern computers, linear pipelines are applied for instruction execution, arithmetic computation, and memory access operations.

- A linear pipeline processor is constructed with K processing stages. External inputs (operands) are fed into the pipeline at the first stage S_1. The processed results are passed from stage S_i to S_{i+1} for all $i = 1, 2 \ldots K-1$. The final result emerges from the pipeline at the stage S_K.
- Depending on the control of data flow along the pipeline, we model linear pipelines in two categories
  - asynchronous
  - synchronous

Asynchronous model: Data flow b/w adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol. When stage S_i is ready to transmit, it sends a ready signal to stage S_{i+1}. After stage S_{i+1} received the incoming data, it returns an acknowledge signal to S_i.

- Asynchronous pipelines are useful in designing communication channels in message passing multicomputers.



- Delay may be differ for each stage.
- throughput rate may be vary.

Synchronous model ⇒ - clocked latches are used to interface between stages.

- Upon the arrival of a clock pulse, all latches transfer data to the next stage simultaneously.
- The pipeline stages are combinational logic circuits.
- It is desired to have approximately equal delays in all stages.

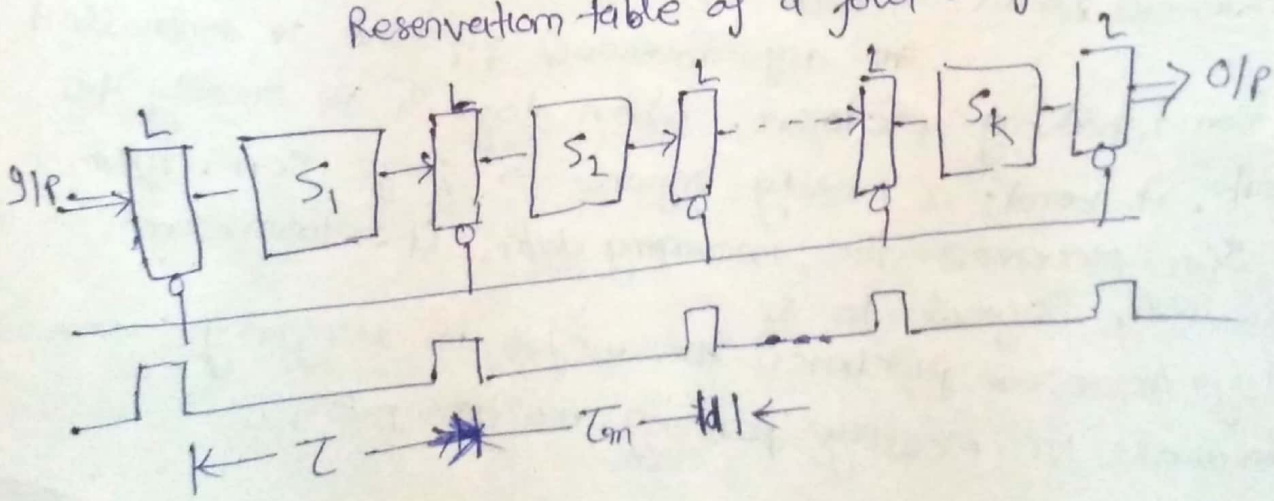
The utilization pattern of successive stages in a synchronous pipeline is specified by a reservation table.

- The utilization follows the diagonal streamline pattern
- for a K-stage linear pipeline, k clock cycles are needed are needed for data to flow through the pipeline

→ Time (clock cycles)

Stages	1	2	3	4
S_1	X			
S_2		X		
S_3			X	
S_4				X

Reservation table of a four stage linear pipeline



S_i = stage i

L = Latch

τ = clock period

τ_m = maximum stage delay

d = Latch delay

<u>clock cycle</u> $\tau = \max_i \{\tau_i\}_1^k + d = \tau_m + d$

In general $\tau_m \gg d$

This implies that maximum stage delay τ_m dominates the clock period.

- The pipeline frequency is defined as the inverse of the clock period

$$f = \frac{1}{\tau}$$

- if one result is expected to come out of the pipeline per cycle, f represents the maximum throughput of the pipeline

<u>Clock skewing</u> => Ideally, we expect the clock pulses to arrive at all stages at the same time. However, due to a problem known as clock skewing, the same clock pulse may arrive at different stages with a time offset of s.

To avoid a race in two successive stage, we must choose

$$\tau_m \geq \tau_{max} + s \quad \& \quad d \leq t_{cmin} - s$$

when clock skew takes effect

$$d + t_{cmax} + s \leq \tau \leq \tau_m + t_{cmin} - s$$

In the ideal case (without effect of clock skewing)

$$s = 0, \quad t_{cmax} = \tau_m, \quad t_{cmin} = d$$

here. t_{cmax} – time delay of the longest logic path within a stage.

t_{cmin} – time delay of the shortest logic path within a stage

=> Total time in linear pipelined ~~in~~ of k stages with n tasks.

first task need k clock cycle

rest (n-1) " (n-1) "

Thus the total time required

$$T_k = [k + (n-1)]\tau$$

where τ is a clock period.

=> for non pipelined, $T_1 = nk\tau$

Speedup factor ⇒

$$S_K = \frac{T_1}{T_K} = \frac{nK\tau}{K\tau + (n-1)\tau} = \frac{nK}{K+(n-1)}$$

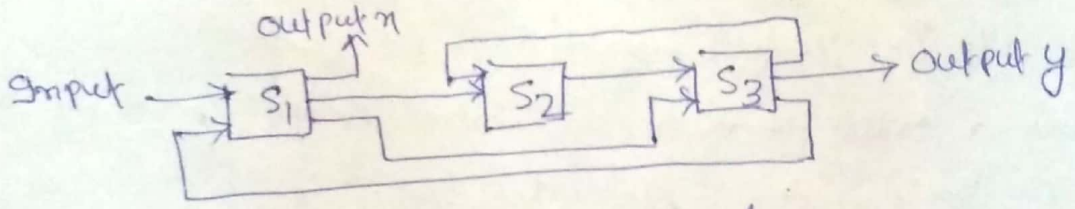
Efficiency ⇒

$$E_K = \frac{S_K}{K} = \frac{n}{K+(n-1)}$$

Throughput ⇒ Throughput defined as the number of tasks performed per unit time.

$$H_K = \frac{n}{K+(n-1)\tau} = \frac{nf}{K+(n-1)}$$

Nonlinear pipeline ⇒ A dynamic pipeline can be reconfigured to perform variable functions at different time. A dynamic pipeline allows feedforward and feedback connections in addition to streamline connections.



A three stage pipeline

Reservation table for o/p x.

	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

Reservation table for function y.

	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y