27/2/24

Thread safeness :-

It refers to an application's ability to execute multiple threads simultaneously without clobbering shared data or creating race condition while accessing shared global memory.

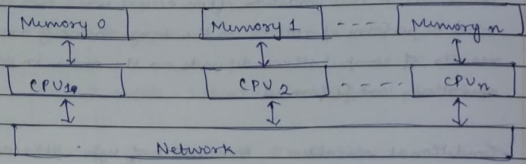
Limitations of pthreads :-

Because of pthreads, a program that runs fine on one platform may fail or produce wrong results on another platform. Eg:- The maximum no. of threads permitted and the default thread stacksize are two important limits to consider when designing your program.

⇒ SPMD Model (Single Program Multiple data Model)

It is a special case of MIMD model.

Tasks are split up and run simultaneously on multiple processors with different inputs in order to obtain results faster.



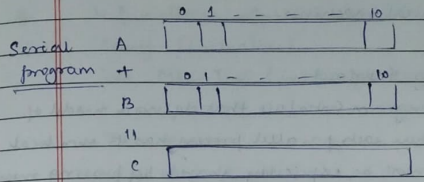
SIMD V/S SPMD ⟶ on your own

Execution process :-

The program is written once but replicated many times with each processing element executing the same program but with different data elements. The processing elements (PE) operate independently of each other and can execute conditional branches or loops differently depending on their assigned data elements.

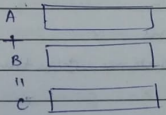
Good Write

```
for (i=0 : 11) {
        c[i] = A[i] + B[i]
}
```



SPMD :- for (i=0:3) (4:7) (8:11)

(for loop ke parts mein divide kariya)



Applications :- used where there is massive data processing Vector multiplication, weather forecasting, scientific simulations,

Advantages

(i) Locality :- Data locality is essential to achieving good performance on large scale machines where communication across the network is very expensive.

(iii) Structured parallelism :- The set of threads is fixed throughout computation. It is easier for compilers to reason about SPMD code resulting in more efficient program analysis than in other models

(iii) Simple runtime implementation :- It has a local view of execution and parallelism is exposed directly to the users. Compilers and runtime systems require less effort to implement than any other MIMD model.

Good Write

DATE: __/__/____
PAGE: _____

Disadvantages

(i) SPMD is a flat model which makes it difficult to write hierarchical code such as divide and conquer algorithms as well as programs optimized for hierarchical machines.

⇒ MPI (Message Passing Interface)

It is an application program interface that defines a model of parallel computing where each parallel process has its own local memory and data must be explicitly shared by passing messages between processes.

MPI communication functions:-

(i) Blocking communication functions

Blocking communications are routines where the completion of the call is dependent on certain events.

MPI_Send (void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

MPI_Recv (void * buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)

For sends, the data must be successfully sent or safely copied to system buffer space and for receives, the data must be safely stored in the receive buffer.

19/3/24

(a) • Communication domain :- All the processes running on different CPUs that are going to communicate with one another.

(b) • Stored in communicator → yo bhi process use krne hain voh yaha store hoga

(c) • Communicator type: MPI_Comm

(d) • Pre-defined default communicator: MPI_COMM_WORLD
This is global. (Sb processes can access it)

4 functions when using MPI

(i) Setup (initialize)
int MPI_Init (int* argc, char***argv);

(ii) Tear down
int MPI_Finalize ();
This allows MPI to turn off all the communication channels

(iii) Total processes
int MPI_Comm_Size (MPI_Comm comm, int *size);

(iv) Local processing index
int MPI_Comm_rank (MPI_Comm comm, int *rank);
(communicator ke andar har process ko index mil jayega. aur wo global_index hoga)

(i) Blocking communication func^n s:- (continued)
Send has 4 modes:-

(i) Standard
(ii) Buffered
(iii) Synchronous
(iv) Ready

} We have blocking & non-blocking fn^s for each

The buffer passed to MPI_Send() can be reused either because MPI saved it somewhere or because it has been received by destination.

(ii) Non blocking functions
These functions return immediately even if the communication is

not finished yet. You must call up MPI_WAIT() & MPI_TEST() to see whether the communication has finished.

20/3/24

Unit-3

High performance Architectures

Instⁿ level parallelism
↳ Delays in instⁿ pipeline
↳ Mechanisms to tackle stalls

Advanced Processor Tech
Clock rate, CPI → clock/cycles per instⁿ
↳ how many instns are executed per unit of time

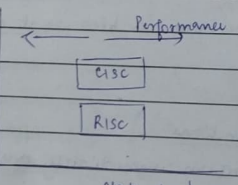
Higher or lower clock rate? → Higher
CPI? → lower

Clock rates moved from lower to higher speeds. CPI is lowered.

Broad categorization

CISC
~~Complex instruction~~ set computer
Eg:- Intel, AMD

RISC → in syll.
~~Reduced instruction~~ set computer
Eg:- MIPS, SPARC, arm
↳ hybrid of both CISC & RISC (not purely RISC)



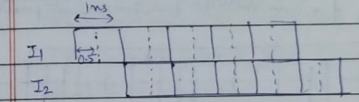
Good Write

scalar → pipelining nhi hoti (jo ek instⁿ each cycle per unit time ko execute hoga)

→ Instⁿ pipeline
- Instⁿ cycle (basic stages)
- Pipeline / Pipeline cycles
- Instⁿ issue latency
- Instⁿ issue rate → execute hone mein kitna time le rha hai instⁿ
  - ideally it is 1 (for scalar)
  - For superscalar, it is > 1
- Resource conflicts
- Base scalar processor (no pipeline)
  - ↳ non-pipelined case

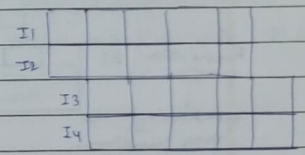
→ Advanced instⁿ pipelining

Superpipelining
increase the depth of the pipeline to ↑ the clock rate.



Pehle fetching mein 1 ns lag rha tha (suppose),
Humne 5 stages to 10 stages mein divide kr diya.
(operations ki suboperations bana diye).
Pehle jo 1 clock cycle mein chalte the, ab ½ clock cycle mein hogi
(isliye hum isse clock rate increase krdenge)

Superscalar
fetch (& executes) more than 1 instⁿ at one time.



- Here CPI < 1
ek clock cycle mein zyada instⁿ execute krte hain.
- Multiple functional units should be there in a processor.
(taaki multiple instⁿs ||ely execute hoskaye)
Instruction 1, 2 → run ||ely
" 3, 4 → run ||ely
} in diff

Superpipelining mein zyada bada processor nhi chahiye.

Good Write

DATE: __/__/____
PAGE: _____

2 ⅋ Methods of superscaling

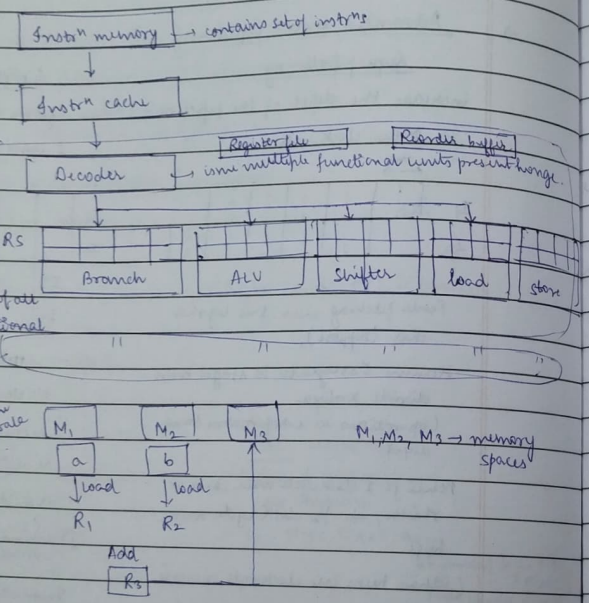
1) ~~Statically~~ Statistically scheduled superscalar processor
2) ~~VLIW (very long instⁿ word) processor~~
3) Dynamically scheduled superscalar processor.

1) Statistically scheduled superscalar processor

↳ these are compiler dependent (bez compiler decide krega kis order main execute honge instrns)
↳ some overhead zyada hota hai (that's why not popular)

2) Dynamically scheduled superscalar processor

Detailed diagram

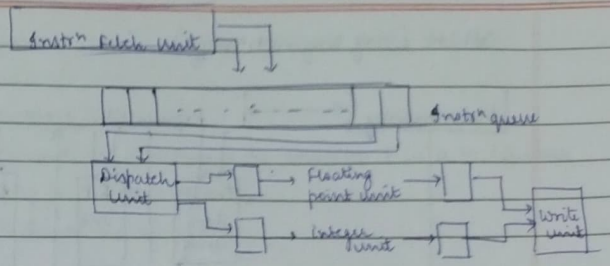


load, store → memory pr hota hai
baaki operations → occurs on registers.

Good Write

DATE: __/__/____
PAGE: _____

exam main yeh banana hai.



Processor with 2 execution units

This is a processor with 2 execution units, one for integer and one for floating point operations.
The instⁿ fetch unit is capable of reading the instⁿ at a time & storing them in a instⁿ queue. In each cycle, the dispatch unit retrieves and decodes upto 2 instrns from the front of the queue. If there is one integer, one floating point instⁿ & no hazards both the instructions are dispatched in the same clock cycle. In this single centralized register file is used to read operands from it and write results into it by each execution unit.

⇒ Advantages of superscalar :-

i) It implements instruction level parallelism in a single processor.
~~through judicious~~
ii) It can avoid many hazards through judicious sch^n & ordering of instrns.

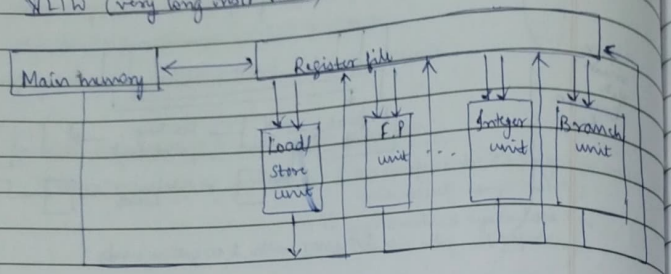
⇒ Disadvantages of superscalar :-

i) Not used much in small embedded systems due to power usage
ii) Enhances the complexity level in the designing of hardware.
iii) Problem in scheduling can occur.

Good Write

23/3/24

VLIW (very long instⁿ word)



for {
for {
for { C = a+b;
d = a×b;

opcode	op1	op2

L/S	ALU	FPU

64 - 1024 bits

Eg:- ADD R_1, R_2 ; SUB R5, R6 ; Load L_7, Data ; Store R_8, data ; } all three instrⁿs must be independent

One VLIW instrⁿ word encodes multiple operations which allows them to be initiated in a single clock cycle.

Eg:- $Z = Ax_1 + Bx_2 + Cx_3$

Sequential Cycles	VLIW
1. Load A	Cycle 1 : Load A
2. Load x_1	Load x_1
3. Multiply y_1, A, x_1	Cycle 2 : Load B
4. Load B	Load x_2
5. Load x_2	Multiply y_1, A, x_1
6. Multiply y_2, B, x_2	Cycle 3 : Load C
7. Add y_3, y_1, y_2	Load x_3
8. Load C	Multiply y_2, B, x_2
9. Load x_3	Cycle 4 : Add y_3, y_1, y_2
10. Multiply y_1, C, x_3	Multiply y_1, C, x_3
11. Add Z, y_1, y_3	Cycle 5 : Add Z, y_1, y_3

Good Write

VLIW :- Instrⁿ long hoggi hai but functional units ek ek hi hain.

Sequential takes 11 cycles, VLIW takes 5 cycles.

⇒ Advantages :-

i) Dependencies are determined by compiler & used to schedule according to functional units.

ii) Reduces hardware complexity.

⇒ Disadvantages :-

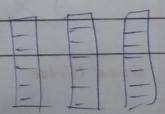
i) Cache misses in one pipeline will force all pipelines to stall in a pure VLIW machine.

ii) The number of instructions in a VLIW instrⁿ word is usually fixed, so padding VLIW instrⁿs with no operations is needed in case the full issue bandwidth is not being met.

(#) VLIW v/s Superscalar

(i) ~~Decoding~~	VLIW	Superscalar
(i) Decoding :	• Easier simple decoding	• Complex
(ii) Code density :	• When instrⁿ level ||ism is lesser, code density worsens	• Better code density, when instrⁿ level ||ism is less
(iii) CPI :	• Lower	• better
(iv) Effectiveness :	• Depends on efficiency of code compaction	• Dynamic behaviour (depends on processor to processor)
(v) Detecting parallelism :	• Requires explicit coding of parallelism. However, no additional h/w or s/w to detect parallelism.	• Complex h/w design. Requires (h/w or s/w) support.

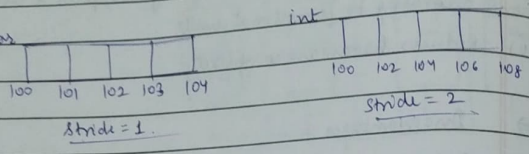
⇒ Scalar data } a=5, addition, sub, multiplication, shifting ops on single data
vector data ⇒ arrays . $a_i + b_i = c_i$



Good Write

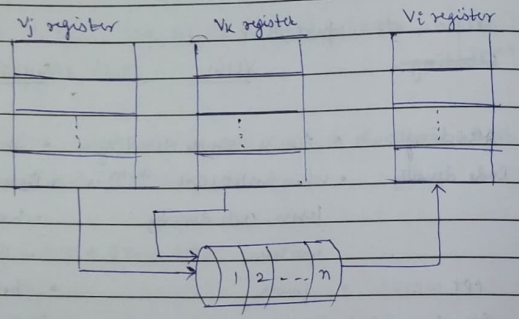
DATE: __/__/____
PAGE: _____

→ Vector :- A vector is an ordered set of scalar data items, all of the same type stored in memory. Vector elements are ordered to have a fixed addressing increment b/w successive elements called stride.



Vector instruction types :-

① Vector - vector instructions : one or two vector operands are fetched from the respective vector registers enter through a functional pipelined unit & produce results.

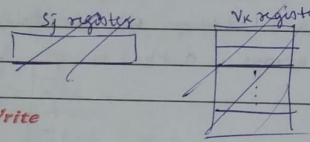


2 mappings : $f_1 : V_i \rightarrow V_j$ & $f_2 : V_j \times V_k \rightarrow V_i$

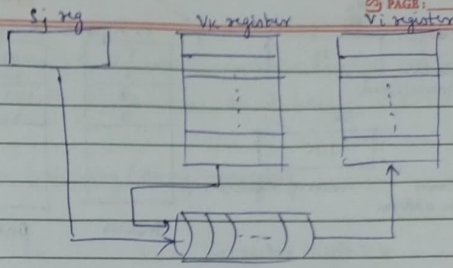
eg: $V_1 = \sin(V_2)$ eg:- $V_3 = V_1 + V_2$

Vector vector instⁿ → 2 types → (i) loading / storing (ii) operations

② Vector scalar instructions :

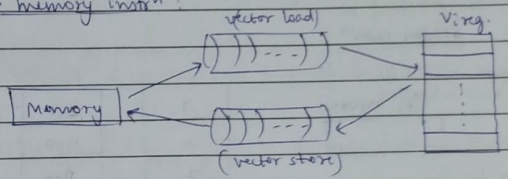


DATE: __/__/____
PAGE: _____



Mapping :- $f_3 : S \times V_k \rightarrow V_i$ eg:- $2 \times [3\ 2\ 1]$
$= [6\ 4\ 2]$

③ Vector memory instⁿ :



Mapping :- $f_4 : M \rightarrow V$
$f_5 : V \rightarrow M$

④ Vector Reduction instⁿ :

Mappings :- $f_6 : V_i \rightarrow S_j$ → eg:- min, max, sum, mean of all elements of vector

$f_7 : V_i \times V_j \rightarrow S_k$

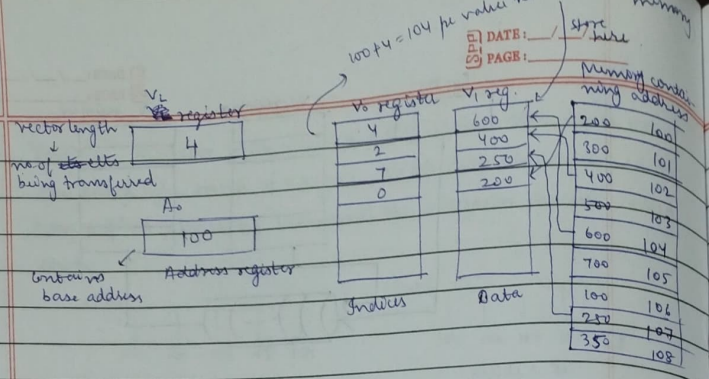
↳ eg:- Dot product

dot product : $S = \sum_{i=1}^{n} a_i \times b_i$ from 2 vectors
$A = (a_i)$ & $B = (b_i)$

⑤ Gather & scatter instrⁿs :-

Gather instrⁿ : $f_8 : M \rightarrow V_i \times V_o$

Scatter instrⁿ : $f_9 = V_i \times V_o \rightarrow M$

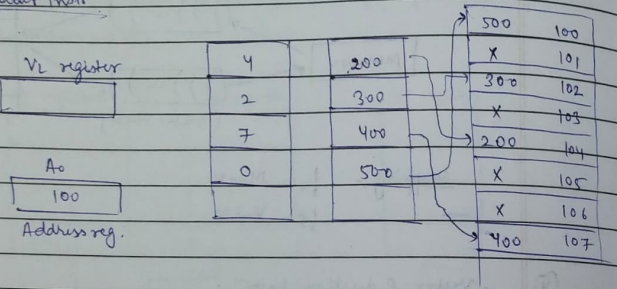


Effective address = Base address + Indices.

Gather instrⁿ

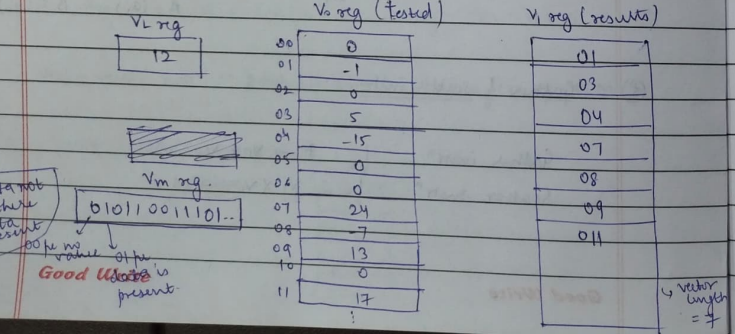
V_0 register → given values hongi ki in index pe value chahiye.

Scatter instrⁿ



⑥ Masking instrⁿ :- It uses a mask vector to compress/expand a vector to a shorter/longer index vector respectively.

Mapping:- $V_0 \times V_m \rightarrow V_1$



Good Write

jo bits empty the, unko compress krdiya.
(pehle vector length is this, ab 7 hogyi)

(A) Vector Processing / Array processing

$V = [v_1 \dots v_n]$ → vector length = n

$c_i = a_i + b_i$

i → index
↳ memory address.

simple programming

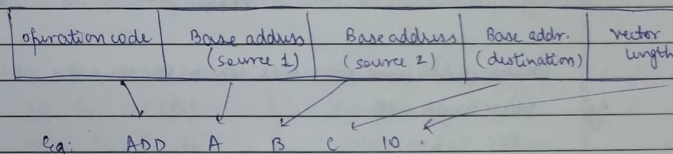
```
int a[10], b[10], c[10];
for (i=0; i<n; i++){
    c[i] = a[i] + b[i];
}
```

↳ fetching instrⁿ one by one.

vector processing → $C(1:10) = a(1:10) + b(1:10)$ → ek hi baar mein kaam hogya (no need of loop)

In computing, vector processor is a CPU that implements an instrⁿ set containing instrⁿs that operate on one-dimensional array of data called vector.

Vector processors have the ability to remove overhead of instrⁿ fetch & execution in loop.



⇒ Array Processors types:-

↓ ↓

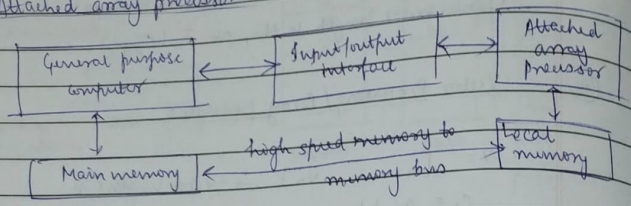
Attached Array Processor SIMD array processor.

Good Write

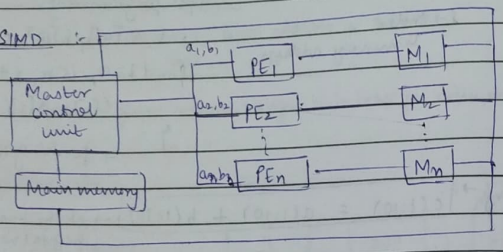
Attached → single PE
SIMD → multiple PEs.

DATE: __/__/____
PAGE: ____

- Attached array processor :-

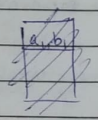


- SIMD :-



ek hi time pe (a_1, b_1) (a_2, b_2), $(a_n b_n)$ ki addition hojayegi

$C_i = a_i + b_i$



	a	b	c	
PE_1	1	2	3	PEs min simultaneously execute hojayegi
PE_2	4	5	9	
PE_3	2	2	4	

Master control unit will check which PEs are active, which are inactive

Eg:- vector length = 30
PEs = 60
jb V.L = 30, to 30 PEs use main hongy (rest ko MCU inactive kardega)

→ Masking are used to control status of each PE during the execution of vector instrⁿ.

Good Write

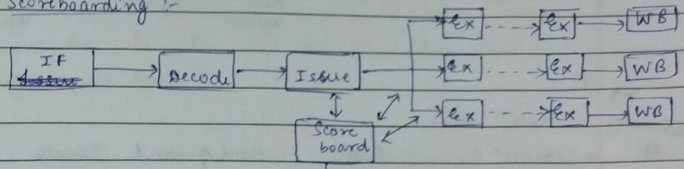
2/4/24

DATE: __/__/____
PAGE: ____

(4) Implementation of ILP (Instrⁿ Level Parallelism)

3 types :-

(I) Scoreboarding :-



↳ It is a centralized control unit

Multiple functional units appear as multiple execution pipelines. So the IInd units allow the instrⁿs to complete out of order program.

Scoreboarding keeps track of registers needed by instrⁿs waiting for various instrⁿ units. It consists of 3 parts :-

(i) Instruction status :- It has 4 steps
(ii) Functional unit status :- 9 fields for each functional unit.
(iii) Register result status :-

Eg:- Write the steps of execution with scoreboarding approach

LD F_6, 34(R_3)
LD F_2, 45(R_2)
MUL F_0, F_2, F_4
SUB F_8, F_6, F_2
DIV F_{10}, F_0, F_6
ADD F_6, F_8, F_2

Given :- (LD → load)
Integer unit : 1 cycle
Adder unit : 2 cycles
Multiplier : 10 cycles
Divider : 40 cycles

Sol :- Instrⁿ status

Instrⁿ	Issue	Read operand	Execute	W.B.
LD	1			

Good Write

R_j R_k

← Pehle R_k ki value fill hoga, then R_j ko fill. Flags indicating when F_j & F_k are available

Functional unit status

FU	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Functional unit		operation	dest^n register	src registers		functional units producing the src register			

(i)

⇒ Instr^n unit doing IU

Instr^n	Issue	Read operand	Execute	WB
LD F_6, 34(R_2)	1	2	3	4

↳ 1 cycle

4 cycles ← main complete hogi

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer unit	Yes	LD	F_6		R_2			Yes	
MULT1									
MULT2									
ADD									
DIV									

Register Result Status

	F_0	F_2	F_4	F_6	F_8	F_{10}	F_{12}	F_{14}
FU	~~Int~~			Int				

(ii) ⇒

Instr^n	Issue	Read operand	Execute	WB
LD F_2, 45(R_3)	5	6	7	8

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int unit	Yes	LD	F_2		R_3			Yes	

(iii) ⇒

Instr^n	Issue	Read operand	Execute	WB
MUL F_0, F_2, F_4	6	9	19	20

Good Write

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
MULT1	Yes	MUL	F_0	F_2	F_4	Int		No	Yes

↓ Abhi tk result WB nhi hua instr^n(2) ka

	F_0	F_2	F_4	F_6	F_8	F_{10}	F_{12}	F_{14}
FU	Mult1		Int					

(iv) ⇒ we will use adder for this

Instr^n	Issue	Read operand	Execute	WB
SUB F_8, F_6, F_2	7	9–10	11	12

↳ multiplier F_2 ka wait krrha hai ab

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Add F_2 ka wait krega ← ADD	Yes	SUB	F_8	F_6	F_2		Int	Yes	No

(v) ⇒

Instr^n	Issue	~~Read operand~~	Read operand	Qx	WB
DIV F_{10}, F_0, F_6	8		21	61	62

for (iv) ⇒

	F_0	F_2	F_4	F_6	F_8	F_{10}	F_{12}	F_{14}
	Mult1				Add			

When F_2 has completed

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	Yes	LD	F_2		R_3			Yes	
MULT1	Yes	MUL	F_0	F_2	F_4			Yes	Yes
ADD	Yes	SUB	F_8	F_6	F_2			Yes	Yes
DIV	Yes	DIV	F_{10}	F_0	F_6	MULT		No	Yes

jb tk SUB poora nhi hota, tb tk add (last instr^n) nhi execute ho skti.

(vi) ⇒

Instr^n	Issue	Read operand	Ex	WB
ADD F_6, F_0, F_2	13	14–15	16	22

jb tk div F_6 ko read nhi krta, tb tk add F_6 ko update nhi krskta.

Good Write

FU	Busy	op	Fi	Fj	Fj	Qj	Qk	Rj	Rk
ADD	Yes	ADD	F6	F8	F2			Yes	Yes

	F0	F2	F4	F6	F8	F10	F12	F14
FU		Add				Div		

∴ Time for all instrns = 62 cycles.

→ Register Result status (for all instr^n^s)

	F0	F2	F4	F6	F8	F10	F12	F14	
FU			Int						Instr^n^ 1
		Int							" 2
	Mult								" 3
				ADD					" 4
				~~SUB~~					
						DIV			" 5
				ADD					" 6

⇒ Limitations of scoreboarding :- → from ppt

- No fwrding
- Limited to instr^n^s in basic block
- No. of FUs ↑es sometimes
- Waiting time is more

DATE : __/__/____
PAGE : _____

Good Write

⇒ Limitations of scoreboarding :- → from ppt.

- No fwding
- Limited to instr's in basic clock
- No. of FUs ↑es sometimes
- Waiting time is more

1/24

(#) Multiprocessing v/s Parallel processing

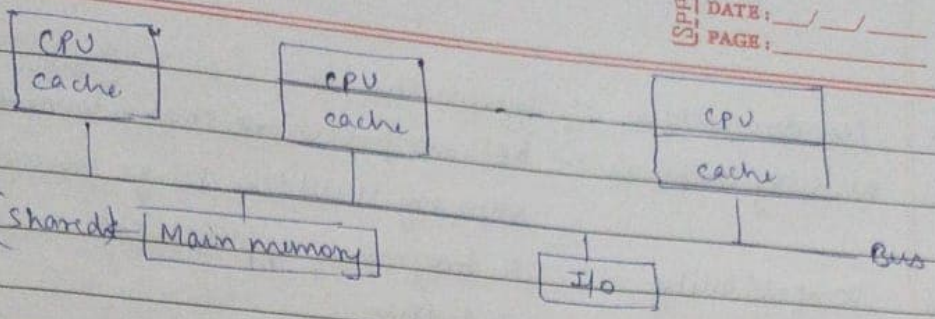
- multiple CPU cores
- usually have only one OS.
- does multiple jobs at same time

- can have different OS of diff. CPUs.
- instr's from same job at (tasks) same time

Multiprocessing :-

- Symmetric
- Asymmetric

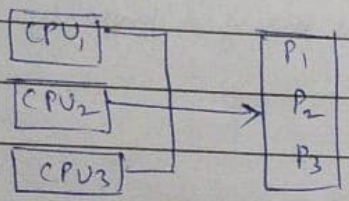
Symmetric



Multiprocessor is a computer with more than one CPU, each share bus, main memory & peripherals in order to simultaneously execute programs.

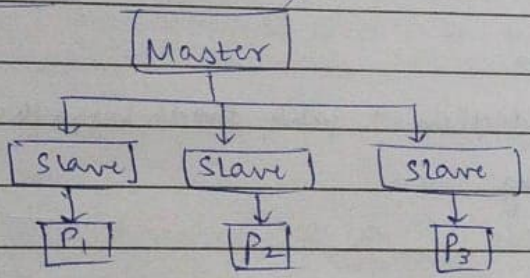
also known as Tightly coupled systems.

⇒ Symmetric multiprocessor :- In this, each processor runs on identical copy of OS & these copies communicate with each other.



⇒ Asymmetric :-

not in syllabus



⇒ Advantages of symmetric :-

(i) Increased throughput → time taken to do jobs decreases, throughput ↑

(ii) Reliability :- If a processor fails, whole system does not fail.

(iii) Performance :- It is higher

(iv) Programming & executing code → Program har ek processor pe run ho sakta hai (bcz they are all identical)

(v) Multiple processors :- If task takes too long, multiple processors can be added to speed up.

Good Write

→ Disadvantages of symmetric :-

(i) Memory expense :- bcz all processors share, common memory, main memory should be large enough.

(ii) Compatibility :- OS, programs/ applications need to support the architecture.

(iii) Complicated OS :- bcz ek hi OS ko sb cheezo ko manage krna hai.
6. The OS manages all the processors in an SMP system. This means the design and mgmt. of OS can be complex, as the OS needs to handle all the available processors, while operating in resource-intensive computing environment.

(iv) Limit on processor count :-

(v) Risk of bottlenecks → if not manged properly, bcz memory is shared.