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.

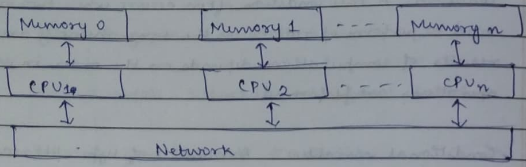
Limitations of pthreads :-

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.

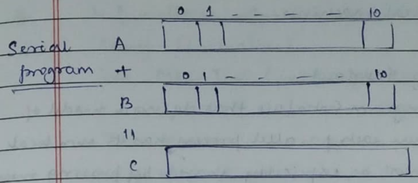


SIMD v/s SPMD → on your own

Execution process :-

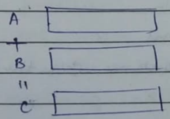
The program is written once but replicated many times with each processing element executing the same program but with different data elements. The processing elements (PE) operate independently of each other and can execute conditional branches or loops differently depending on their assigned data elements.

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



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

(for loop ko parts ~~main~~ divide kardiya)



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.

(ii) Structured parallelism :- The set of threads is fixed throughout computation. It is easier for compiler 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.

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 → jo bhi process use krrhe 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 channe

(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 vo global index) hoga)

(i) Blocking communication funcⁿs :- (continued)

Send has 4 modes :-

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

} we have blocking & non-blocking fⁿs 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() & ~~or~~ MPI_TEST() to see whether~~a~~ the communication has finished.

20/3/24

Unit - 3

High performance Architecture

Instrn level parallelism
↳ Delays in instrn pipeline
↳ Mechanisms to tackle stalls

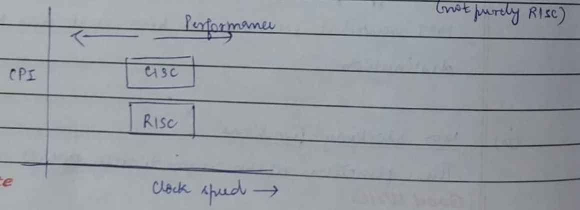
Advanced Processor Tech
Clock rate, CPI ⟶ clock/cycles per instrn
↳ how many instrns are executed per unit of time

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

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

Broad categorization

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



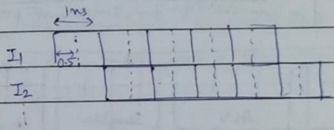
scalar → pipelining nhi horhi (jo ek instrn aarhi hai per unit time vo execute hogi)

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

→ Advanced instrn pipelining

Superpipelining

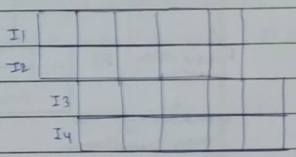
increase the depth of the pipeline to ↑ the clock rate.



Pehle fetching mein 1ns lag rha tha (suppose),
Humne 5 stages to 10 stages mein divide krdiya.
(operations ki suboperations bana diye).
Pehle jo 1 clock cycle mein cheez ho rhi thi, ab 1/2 clock cycle mein hogi.
(idhar hum isse clock rate increase krdenge)

Superscalar

fetch (& execute) more than 1 instrn at one time.



- Here CPI < 1
ek clock cycle mein zyada instrs execute horhe hain.
- Multiple functional units should be there in a processor.
(taaki ~~ea~~ multiple instrns ||ely execute hojaaye)
Instruction 1, 2 → run ||ely } in diag
" 3, 4 → run ||ely

Superpipelining mein zyada bada processor nhi chahiye.

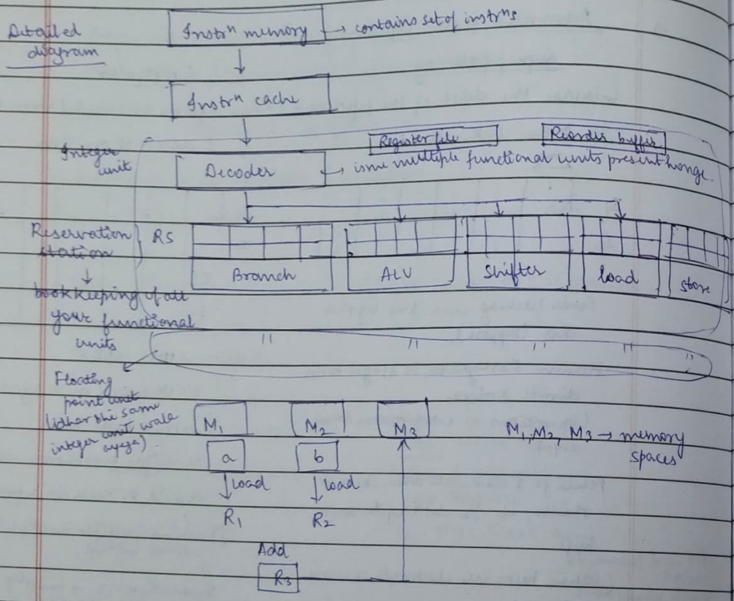
DATE: __/__/__
PAGE: ____

2.7 Methods of superscaling

1) ~~Statically~~ statistically scheduled superscaler processor
2) ~~VLIW (Very long instrn word) processors.~~
3) Dynamically scheduled superscalar processor.

1) Statistically scheduled superscalar processor
↳ these are compiler dependent (bcz compiler decide krega kis order mein execute honge instrns)
↳ isme overhead zyada hota hai (that's why ~~not popular~~)

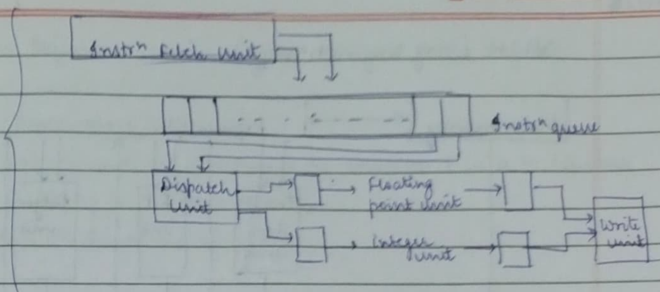
2) Dynamically scheduled superscalar processor



Load, store → memory pe hota hai
baaki operations → occur on registers.

DATE: __/__/__
PAGE: ____

exam mein 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 instrn fetch unit is capable of reading the instrn at a time & storing them in a instrn queue. In each cycle, the dispatch unit retriev and decodes upto 2 instrns from the front of the queue. If there is one integer, one floating point instrn & 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 soln & ordering of instrns.

⇒ Disadvantages of superscalar :-

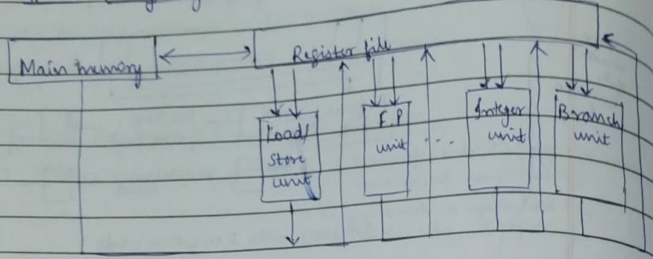
i) Not used much in small embedded systems due to power usage
ii) Enhances the complexity level in the designing of hardware.
iii) Problem in scheduling can occur.

Good Write

Good Write

23/3/24

VLIW (very long instrn 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 these 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

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

VLIW

Cycle 1 : Load A
load x_1

Cycle 2 : Load B
Load x_2
Multiply y_1, A, x_1

Cycle 3 : Load C
load x_3
Multiply y_2, B, x_2

Cycle 4 : Add y_3, y_1, y_2
Multiply y_1, C, x_3

Cycle 5 : Add Z, y_1, y_3

VLIW :- Instrn long hogyi 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.

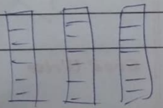
ii) The number of instructions in a VLIW instrn word is usually fixed, so padding VLIW instrns with no operations is needed in case the full issue bandwidth is not being met.

(#) VLIW v/s Superscalar

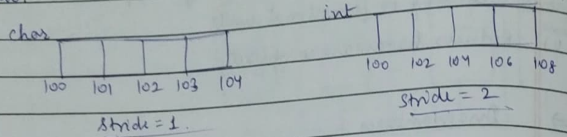
		VLIW	Superscalar
(i)	~~Decoding~~		
(i)	Decoding :	• Easier: simple decoding	• Complex
(ii)	Code density :	• when instrn Level 11^{elism} is lesser, code density worsens	• Better code density, when instrn level 11^{elism} is less
(iii)	CPI :	• Lower	• better
(iv)	Effectiveness :	• Depends on efficiency 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 . $a_i + b_i = c_i$

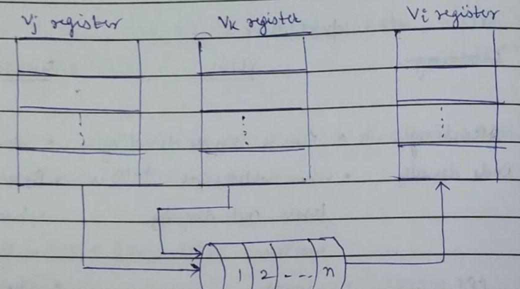


→ Vector :- A vector is an ordered set of scalar data items, all of the same type stored in memory. Vector elements are ordered to have a fixed addressing increment b/w successive elements called stride.



Vector instruction types :-

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

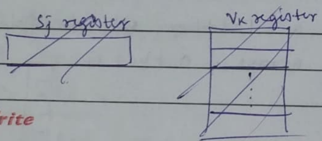


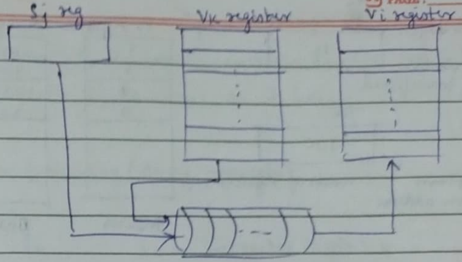
2 mappings : $f_1 : V_i \rightarrow V_j$ & $f_2 : V_j \times V_k \rightarrow V_i$

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

Vector vector instr^n → 2 types → (i) loading / storing (ii) operations

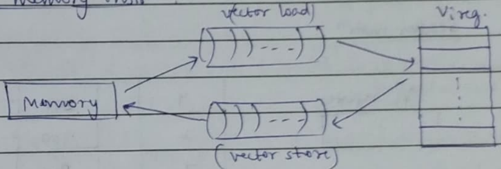
② Vector scalar instructions :





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

③ Vector memory instr^n :



Mapping :- $f_4: M \rightarrow V$

$f_5: V \rightarrow M$

④ Vector Reduction instr^n :

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

$f_7 : V_i \times V_j \rightarrow S_K$

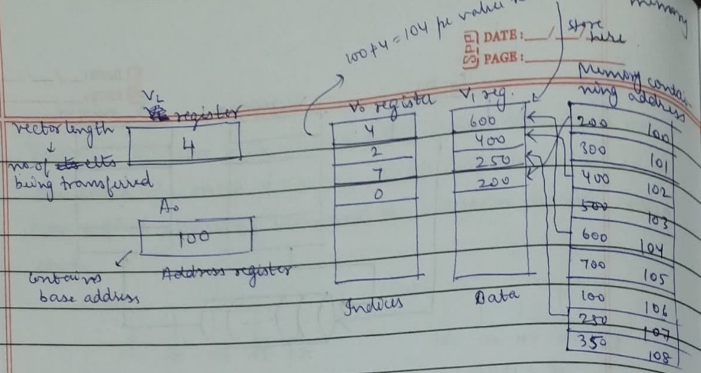
↳ eg:- Dot product

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

⑤ Gather & Scatter instr^{n}s :-

Gather instr^n : $f_8 : M \rightarrow V_1 \times V_0$

Scatter instr^n : $f_9 = V_1 \times V_0 \rightarrow M$

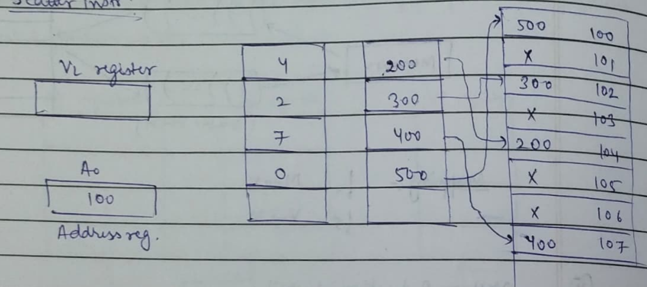


Effective address = Base address + Indices.

Gather instrn

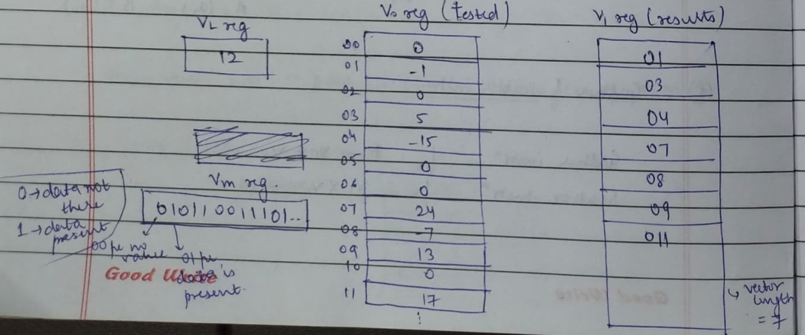
V0 register → given values hongi ki in index pe value chahiye.

Scatter instrn



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

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



jo elts empty the, unko compress krdiya.
(pehle vector length 12 thi, ab 7 hogyi)

Ⓐ Vector Processing / Array processing

$V = [v_1 \dots\dots v_n]$ → 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 instrns~~ one by one.

vector processing → $\boxed{c(1:10) = a(1:10) + b(1:10)}$ → 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.

operation code	Base address (source 1)	Base address (source 2)	Base addr. (destination)	vector length
ADD	A	B	C	10

Eg: ADD A B C 10 .

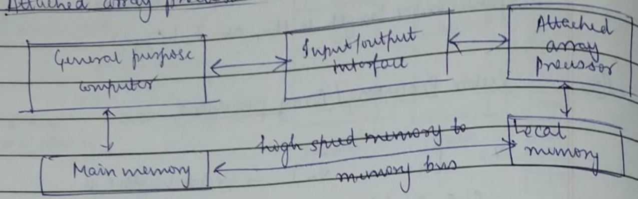
⇒ Array Processors types :-

↓ Attached Array Processor

↓ SIMD array processor.

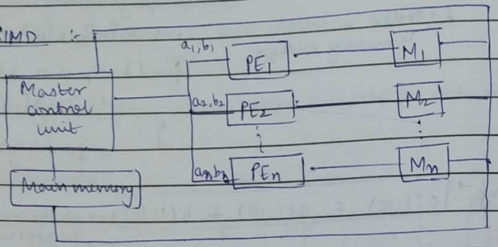
Attached → single PE
SIMD → multiple PEs.

- Attached array processor :-



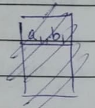
- SIMD :-

PE
↓
processing element



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

$c_i = a_i + b_i$



	a	b	c
PE1	1	2	3
PE2	4	5	9
PE3	2	2	4

PEs mein simultaneously execute hojayegi

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

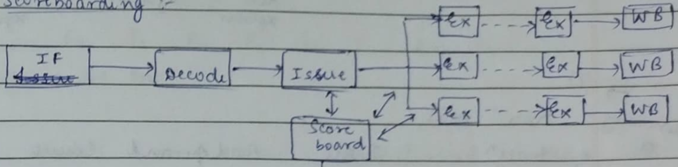
eg:- vector length = 30
PEs = 60
jb V.L = 30, to 30 PEs use mein hongey (rest ko MCU inactive krdega)

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

2/4/24

(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 ||el 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.

Sol:
LD F_6, 34 (R_2)
LD F_2, 45 (R_3)
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

Ans:- Instrn status

Instrn	Issue	Read operand	Execute	W.B.
LD	1			

Rj Rk ← Pehle Rk ki value fill kro ~~[illegible]~~, then Rj ko fill

Flags indicating when Fj & Fk ~~are available~~

Functional unit status

FU Busy Op Fi Fj Fk Qj Qk Rj Rk

FU ↓ functional unit

Op ↓ operation

Fi ↓ destn register

Fj, Fk → src registers

Qj Qk → functional units producing the src register

⇒ (i)

Instrn unit

done using IU

Instrn	Issue	Read operand	Execute	WB
LD F6, 34(R2)	1 ↳ 1 cycle	2	3	4

4 cycles mein complete hogi

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer unit	Yes	LD	F6		R2				Yes
MULT1									
MULT2									
ADD									
DIV									

Register Result status

	F0	F2	F4	F6	F8	F10	F12	F14
FU ~~Int~~				Int				

(ii) ⇒

Instrn	Issue	Read operand	Execute	WB
LD F2, 45(R3)	5	6	7	8

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int unit	Yes	LD	F2		R3				Yes

(iii) ⇒

Instrn	Issue	Read operand	Execute	WB
MUL F0, F2, F4	6	9	19	20

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
MULT1	Yes	MUL	F0	F2	F4	Int		No	Yes

↓ ~~Abhi tk result~~ WB nhi hua Instrn(2) ka

	F0	F2	F4	F6	F8	F10	F12	F14
FU	Mult1	Int						

(iv) ⇒

Instrn	Issue	Read operand	Execute	WB
SUB F8, F6, F2	7	9-10	11	12

we will use adder for this

↳ multiplier F2 ka wait krsha hai ab

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
ADD	Yes	SUB	F8	F6	F2		Int	Yes	NO

Ab F2 ka wait krga

(v) ⇒

Instrn	~~Read operand~~ Issue	Read operand	Ex	WB
DIV F10, F0, F6	8	21	61	62

for (iv) ⇒

	F0	F2	F4	F6	F8	F10	F12	F14
	Mult1				Add			

when F2 has completed

FU	Busy	op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Int	Yes	LD	F2		R3				Yes
MULTI	Yes	MUL	F0	F2	F4			Yes	Yes
ADD	Yes	SUB	F8	F6	F2			Yes	Yes
DIV	Yes	DIV	F10	F0	F6	MULT		No	Yes

jb tk sub poora nhi hota, tb tk add (last instrn) nhi execute ho skti.

(vi) ⇒

Instrn	Issue	Read operand	Ex	WB
ADD F6, F8, F2	13	14-15	16	22

jb tk div F6 ko read nhi krta, tb tk add F6 ko update nhi krskta.

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 instrns)

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

⇒ Limitations of scoreboarding :- → from ppt.

- No frwding
- Limited to instrns in basic clock
- No. of FUs ↑es sometimes
- Waiting time is more