Chapter-6 Pipelining

Pipelining – an implementation technique in which multiple instr are overlapped in execution.

eg: laundry (Wash, Dry, Fold)

eg: ordering food item in canteen.

if 2 people wants to buy food, [place order (P) + buy (B)] → 2 stages

in non pipelined :-

5M 5M 5M 5M 5M 5M.

P1 P B

P2 P B

P3 P B

Total time – $10 \times 3 = \underline{\underline{30}}$

In pipelined :

5M 5M 5M 5M

P1 P B → 2×5

P2 P B → $2 \times 5 + 5$

P3 P B → $2 \times 5 + 5 + 5$

∴ Total time = $\underline{\underline{20m}}$.

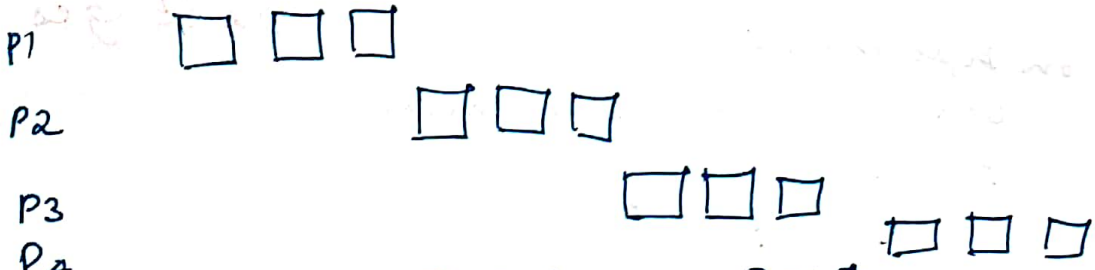
Throughput – executing time of n tasks.

speed up ratio = $\frac{\text{time non pipe}}{\text{time pipe}}$.

Q) Consider a k stage pipeline with t_p as time reqd for each stage to process for n no. of processes. Calculate the total no. of cc to complete pipelined & non-pipelined exec in. & also speed up ratio.

eg: if 3 stages. & 4 processes.

non-pipelined



Total cc = 3×4

$= 12$

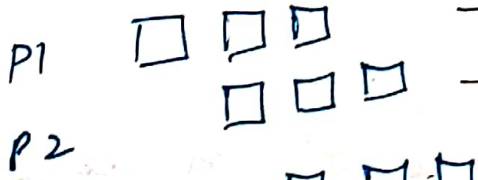
ie $(k \times n)$

Total time = no: of cc x time for each stage

$= k \times t_p$.

$= k \times n \times t_p$.

pipelined.



no. of cc

$\rightarrow 3$

$\rightarrow 3+1$

$\rightarrow 3+2$

$\rightarrow 3+3$ ie $k+n-1$

$\therefore$ Total $= 3+3 = 6$ $(k+n-1)$

$\therefore$ time $= (k+n-1) \times t_p$.

$\therefore$

Speed up ratio = $\frac{\text{Time nonpipe}}{\text{Time pipe}}$

Speed up $= \frac{k \times n \times t_p}{(k+n-1) \times t_p} = \frac{k \times n}{k+n-1}$

Nonpipelined:

Time $= k \times n \times t_p$, CC $= k \times n$

Pipelined

Time $= (k+n-1)\, t_p$, CC $= k+n-1$

Q) Calculate no: of cc reqd by a 4 stage pipeline to execute 100 instr if it takes 30ps for each stage. Also calculate speed up ratio.

No: of CC $= k+n-1 = 4+99 = \underline{\underline{103}}$

speed up $= \frac{kn}{k+n-1} = \frac{4 \times 100}{4+99} = \underline{\underline{3.88}}$.

Q) Assume operation time for major functional units: Memory – 200ps, ALU – 100ps, Reg read/write – 50ps. Consider a 5 stage pipeline in MIPS with stages Instr Fetch, Reg Read, Execution, Mem Access and Reg write. Calculate no: of cc reqd to execute 200 instr in pipelined and non pipelined. and in SCI and also speed up ratio?

In [illegible] SCI,

as in page 62.

600ps – total time for an instr

(here $(k \times t_p) = 600ps$)

∴ no need to find $k \times t_p$

$$Time_{SCI} = 600ps \times n \quad \left(\begin{array}{l} IC \times CPI \times CCT, \\ (= 200 \times 1 \times 600) \end{array} \right)$$

$$= 600 \times 200 = \underline{\underline{120ns}} = \underline{\underline{120000ps}}$$

Pipelined :

max time for each stage = 200ps

$$\therefore t_p = 200ps$$

$$\therefore time_{pipe} = (k+n-1) t_p$$

$$= (5+99) \times 200$$

$$= 104 \times 200 = \underline{\underline{20800}} ps$$

∴ [illegible]

$$Non\ pipelined = n \times k \times t_p$$

$$= 200 \times 5 \times 200$$

(200 = max time for each stage)

$$= \underline{\underline{200000}} ps.$$

[illegible]

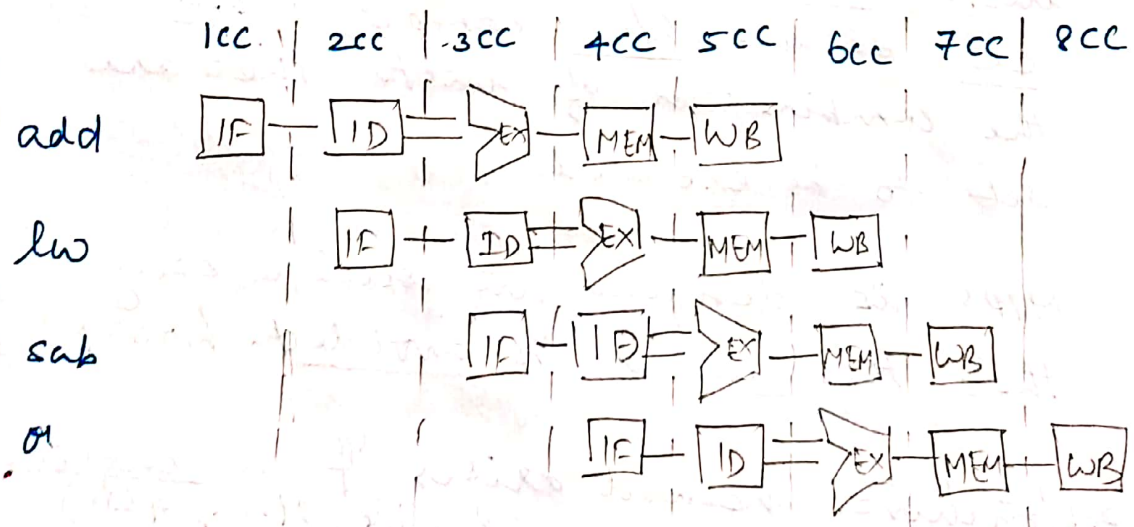
In MIPS, instr execution can be split into 5 stages –

(IF) Instr fetch – uses IM
(ID) Instr decode – uses Reg file
(EX) Execution – ALU
(MEM) Memory access – DM
(WB) Write Back – Reg file

If we have ~~three~~ four instrs,

```
add $s1 $s2 $s3
lw  $s4 4($s6)
sub $s2 $s7 $s3
or  $s3 $s7 $s7
```

multiple clock cycle pipeline diagram can be drawn as.



→ altogether it requires 8cc

If not in pipeline, how much?

$5 \times 4 = 20cc$

Pipeline hazards:

→ events in pipelining when the next instr cannot execute in the following cc.

① ↳ structural hazard

② ↳ data hazard
  ↳ load use hazard

③ ↳ control or branch hazard.

① Structural hazard.

An occurence in which a planned instr cannot be executed in a proper cc because h/w cannot support the combination of instr that are set to be executed.

MIPS is designed in such a way that this hazard is avoided. How?

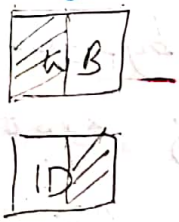
* structural hazard arises if a conflict (Regfile, Mem, ALU) arises when the same unit is used by diff instr in the same cc.

2 cases:

ⓐ Consider 4 cc
If same mem was used for IF and MEM access, structural hazard

will arise, because both instr and data is being fetched/written in the same unit.
But we have separate mem — IM & DM
∴ It not the same unit. Hence no hazard

(b) Consider the 4th cc.
4th instr uses Reg file to decode
1st instr " " to write value
But this hazard is also avoided because, reading the file happens in the 2nd half of cc & writing in 1st half.



∴ MIPS has no structural hazard.

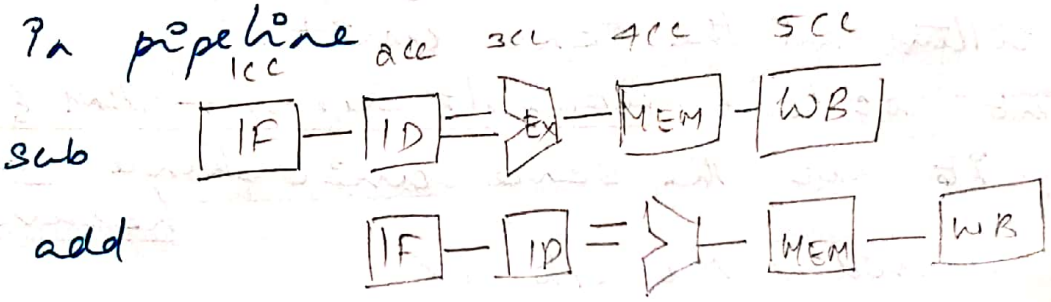
[Note: ALU is also used by only one instr in pipeline in any cc.]
∴ no hazard.

(2) Data hazard.

An occurrence in which a planned instr cannot execute in the proper cc because data that is needed to execute the instr is not yet available.

eg: sub $s1 $s2 $s3
add $s3 $s1 $s2

In pipeline



In the 4th CC, add instr requires the value of $s1 which is written to reg file only in 5th CC

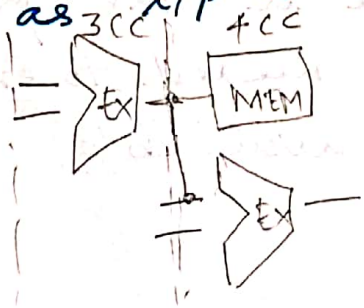
→ data hazard.

Solution:

Forwarding / bypassing.

→ a method of resolving the hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from registers or memory.

Eg: at the end of 3rd CC, the value of $s2 - $s3 to be written to $s1 is executed & ready in ALU. (for sub)
This value is forwarded to 4 CC & given as i/p to ALU to for add instrs



* whatever value was decoded by add ~~sub~~ instr in 3rd cc is overwritten by the value forwarded from above instrs & this is forwarded to ALU for execution.

2.1 load use hazard. — ~~an occurrence in which data required by load~~ [an occurrency in which data requested from a load instr has not yet become available]

Consider the below set of instrs.

```
lw  $s1  4($s2)
add $s3  $s1  $s4
```

	1cc	2cc	3cc	4cc	5cc	6cc
lw	IF	ID	EX	MEM	WB	
add		IF	ID	EX	MEM	WB

As in the prev case, here forwarding alone will not work. Because the actual value to be written to $s1 is available only at the end of 4cc. But add needs $s1 in the beginning of 4cc.

How to resolve this?

Ans: Stalling & forwarding.

* If add instr can wait for 1cc, the value can be forwarded. — stall — no operation.

[will see later]

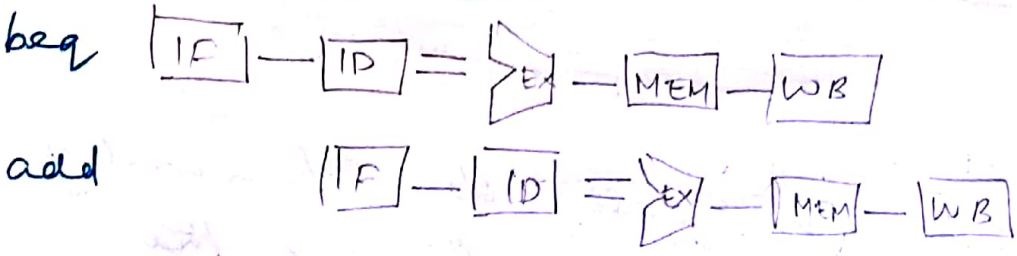
Control hazard.

An occurrence in which proper instr cant be executed in a cc because instr fetched is not the one needed.

Soln: stalling.

eg:

```
        beq  $s1  $s2  L
③      add  $s1  $s2  $s3
       sub  $s2  $s3  $s4
       lw   ...
       ...
L:     and  ...
       1cc  2cc  3cc
```



for beq instr, if $s1 & $s2 are same, then the next instr to be executed is and (L), but in pipeline, in 2nd cc, add instr is automatically fetched. ∴ we need to stall until the decision is made.

[will see later]

* actual decision making happens in the 4cc.

soln: * predict branch will not be taken and proceed with normal sequential execution.

* ~~in 4cc, if it~~

Consider the instrs given in prev eg:

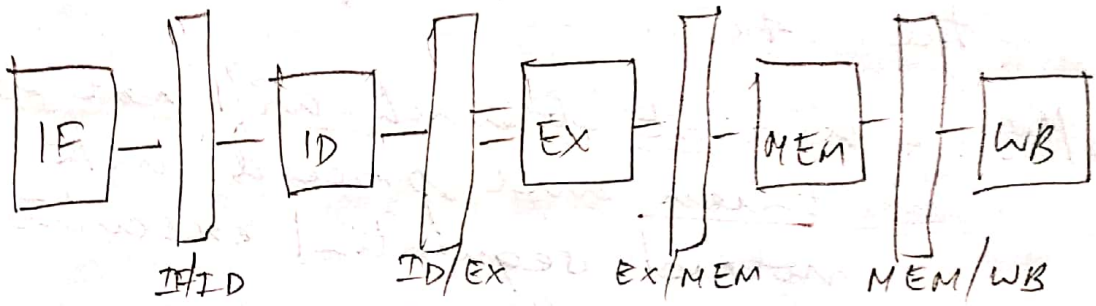
* if s1 & s2 are same then next instr to be executed should be L: and...

∴ if beq — then stall the instrs pushed to pipeline till 4cc, else proceed with exec

(decision made here)

if our prediction goes wrong, then flush the three instrs & start executing L instr 5th cc

Pipelined Datapath.



Data stored in the pipeline registers:

IF/ID:
- PC+4, (32bits)
- Instr (32bits)

} = 64 bits (max size)

ID/EX:
- PC+4 (32)
- RD1 (32)
- RD2 (32)
- Sign extended (32)
- rt/rd no: (5)

} = 133 bits

EX/MEM:
- branch addrs (32)
- zero flag (1)
- ALU Result (32)
- RD2 (32)
- rt/rd no: (5)

} = 102 bits

MEM/WB:
- RD (DM) (32 bits)
- ALU res (32 bits)
- rt/rd no: (5)

} = 69 bits

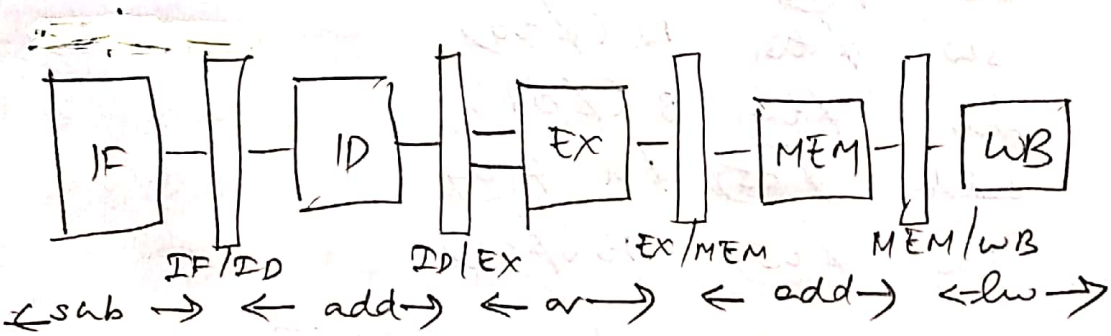
a) Consider the below set of mc/ns.

84 i1 lw $s5 40($s2) - WB.
88 i2 add $s6 $s3 $s4 - MEM
92 i3 or $s7 $s2 $s1 - EX
96 i4 add $s0 $s4 $s3 - ID
100 i5 sub $s1 $s2 $s1 - IF

At CC5, right before the mc/ns are executed, the processor state is as follows. PC has address 100. Every reg has an initial value of 10 + reg no:. Determine the value of every field in the 4 pipeline reg at the end of 5CC.

$s0 - 26 (10+16)
$s1 - 27 (10+17)
$s2 - 28 (10+18)
$s3 - 29
$s4 - 30

IF/ID : a) sub mc/ns
b) 104 (PC+4)



ID/EX : a) PC+4 of i4 - 100 (add mc/ns)
b) RD1 - $s4 - 30
c) RD2 - $s3 - 29
d) rd no: - of $s0 - 16.

EX/MEM :
a) ALU Result of (or) mc/ns (i3)
$s2 || $s1 = 28 || 27 = 31
b)

b) rt/rd no: = of $s7 = 23 (76)

MEM/WB:

a) ALURes of I2 (add instr)
$s3 + $s4 = 29 + 30 = 59

b) rt/rd no: of $s6 = 22

Code - reordering to avoid pipeline stall.

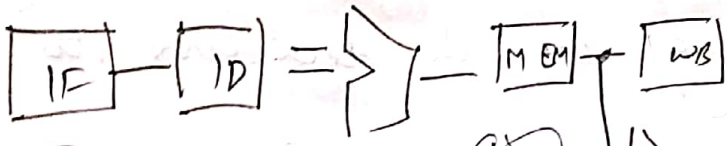
Consider the below set of instrs. Identify the dependencies, hazards and solⁿ. Reorder the instr to avoid any pipeline stalls.

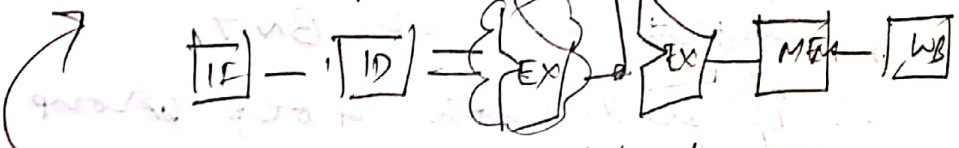
```
1  lw   $t1  0($t0)
2  lw   $t2  4($t0)
3  add  $t3  $t1  $t2
4  sw   $t3  12($t0)
5  lw   $t4  8($t1)
6  add  $t5  $t1  $t4
7  sw   $t5  16($t0)
```

	dependencies	hazard	solⁿ
(a)	b/w 2 & 3 ($t2)	load-use	stalling + forwd.
(b)	3 & 4 ($t3)	data	forwd.
(c)	5 & 6 ($t4)	load use	stalling + forwd
(d)	6 & 7 ($t5)	data	forwd

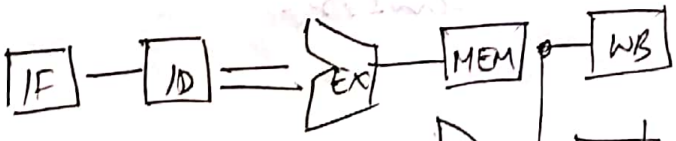
To avoid stalling b/w 2 & 3, (of lcd)

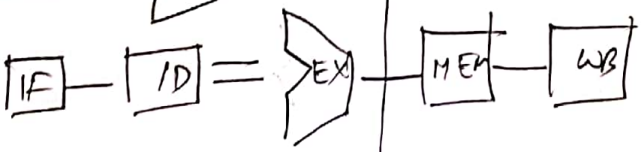
* If we can put some independent instr b/w them, stalling can be avoided.
* check for one such instr.

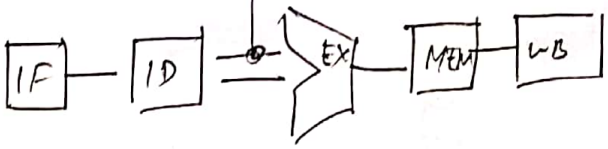
lw 

add 

If an instr is added,

lw 

? 

add 

ans:
If instr (E) is put b/w (2) & (3)

```
1   lw   $t1   0($t0)
2   lw   $t2   4($t0)
3   lw   $t4   8($t0)
4   add  $t3   $t1   $t2
5   sw   $t3   12($t0)
6   add  $t5   $t1   $t4
7   sw   $t5   16($t0)
```

→ (d) and (c) → avoided.

Branch hazards.

soln:

① Assume branch not taken.

② Reducing delay of branches.

— moving branch decision to ID stage

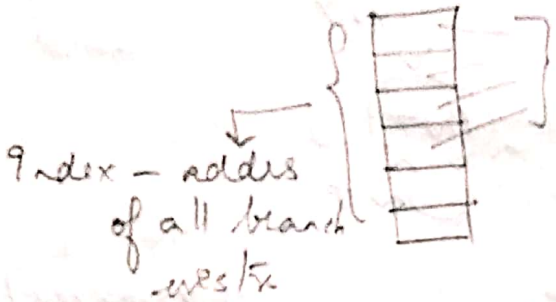
& again predict BNT

∴ if prediction goes wrong, we need to flush only one instr.

Dynamic branch prediction.

① 1-bit prediction scheme.

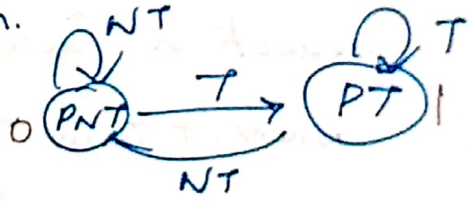
* a branch history table



0 — not taken

1 — taken.

1 bit FSM



for an addrs - check the bit
& if 1 — assume will be taken
& more forward
0 — assume will not be taken

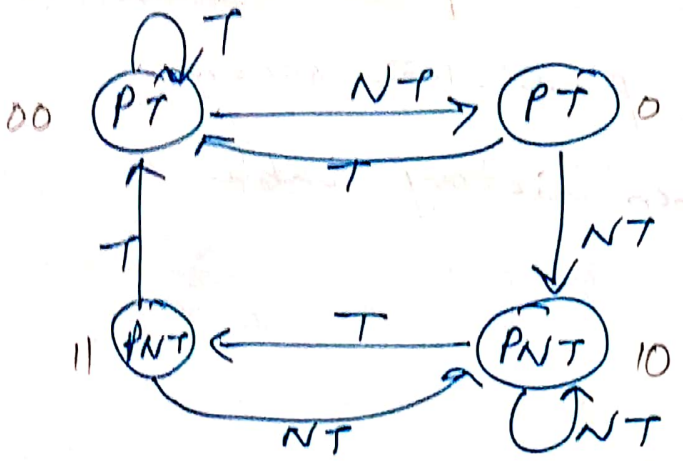
If prediction goes wrong, then invert the bit.

∴ every time, prediction goes wrong, bit is inverted.

② 2-bit prediction scheme.

— prediction must be wrong twice, b4 it is changed.

2-bit FSM:



For the foll eg: (branching sequences) calculate the prediction accuracy. (assume initial state is not taken)

(1) NNN TTNNTTN

actual seq	predict	H/M
N	N	H
N	N	H
N	N	H
T	N	H
T	T	M
N	T	H
N	N	M
T	N	H
T	T	M
N	T	H

accuracy = $\frac{7}{10} = \underline{0.7}$

≈ Calculate for

TTTNNTTNNT