Pipelining

Using pipelining we can execute more than one instruction at a time.

Suppose we have four stages as

IF - instruction fetch

ID → Instruction decode

Ex - Execute

WB - write back.

Suppose Each ✓ Stage takes one clock. So if we have to execute 3 instruction (without pipeline) we need $3 \times 4 = 12$ clock cycle

By using pipelining.

	1	2	3	4	5	6	7	8	9	10	11	12
IF	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9			
ID		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9		
~~EX~~			I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	
WB				I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
				↓	↓	↓	↓					

CPI = 1, in 12 clock we can execute 9 instruction.

Clock cycle = 4 + 8 = 12

Suppose n no. of instruction,

Clock cycle needed = 4 + (n-1)

So, If we have m stage and n no. of instruction.

Clock cycle needed = $m+(n-1)$

Speedup = $\frac{\text{Time without pipeline}}{\text{Time with pipeline}}$

Suppose time for one clock = t_p

So Time using pipeline

$$T_p = [m+(n-1)]\,t_p$$

without pipeline,

Clock cycle needed = $m*n$

Time $T_{wp} = m*n*t_p$

$$\text{Speedup} = \frac{m*n*t_p}{[m+(n-1)]t_p} = \frac{mn}{m+(n-1)}$$

$$= \frac{mn}{n+(m-1)} = \frac{m}{1+\frac{(m-1)}{n}}$$

n is large

Speedup $\cong m$

$$\text{Efficiency} = \frac{\text{Speedup}}{m}$$

Efficiency $\cong 1$

Q. Speedup $\leq m$

if we increase no of stages speedup will increase upto what will be the max limit of stage?

Q.2 Comparing the time T_1 taken for a single instruction on a pipelined cpu with time t_2 taken on a non-piplined but identical cpu, we can say that

(a) $T_1 \leq T_2$, b

(b) $T_1 \geq T_2$

(c) $T_1 < T_2$

(d)

Q. How many clock cycle needed?

	F	D	E	W
I_1	1	2	1	1
I_2	1	2	2	1
I_3	2	1	3	2
I_4	1	3	2	1
I_5	1	2	1	2

← stage

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I_1	F	D	D	E	W										
I_2	F	X	D	D	E	E	W								
I_3		F	F	X	D	X	E	E	E	W	W				
I_4				F	X	D	D	D	X	E	E	W			
I_5					F	X	X	X	D	D	X	E	W	W	

Clock

total 15 clock cycle.

How many clock cycle needed for following program

```
for (i=1 to 2)
{
    I1;
    I2;
    I3;
    I4; }
```

	S_1	S_2	S_3	S_4	← stage
I_1	1	1	1	1	
I_2	1	3	2	2	clock cycle
I_3	2	1	1	3	
I_4	1	2	2	2	

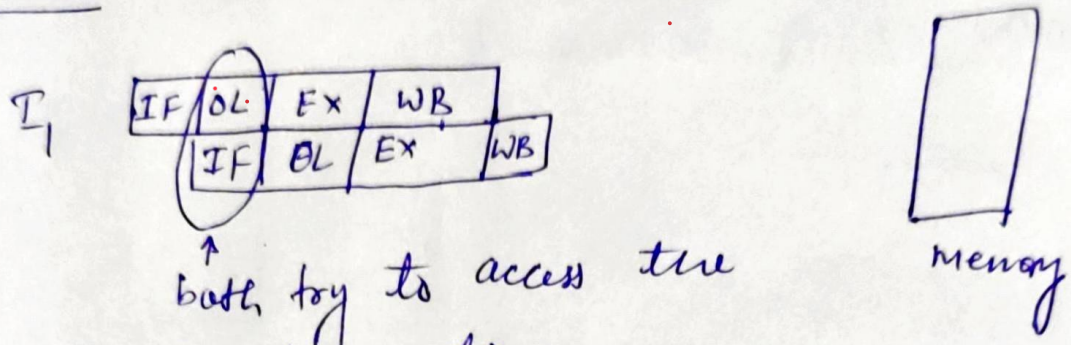
↑ Instruction

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S_1	S_1	S_2	S_3	S_4																		
		S_1	S_2	S_2	S_2	S_3	S_3	S_4	S_4													
			S_1	S_1	x	S_2	x	S_3	x	S_4	S_4	S_4										
					S_1	x	S_2	S_2	S_3	S_3	x	x	S_4	S_4								
						S_1	S_1	x	S_2	x	S_3	x	x	x	S_4							
								S_1	x	S_2	S_2	S_2	S_3	S_3	x	S_4	S_4					
									S_1	S_1	x	S_2	x	x	S_3	x	x	S_4	S_4	S_4		
											S_1	x	S_2	S_2	x	S_3	S_3	x	x	x	S_4	S_4

Hazard in Pipeline

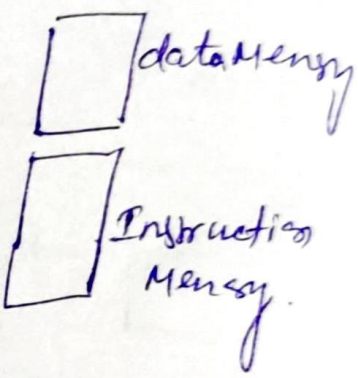
1. Structural
2. Control
3. data

Structural:- because of resource conflicts

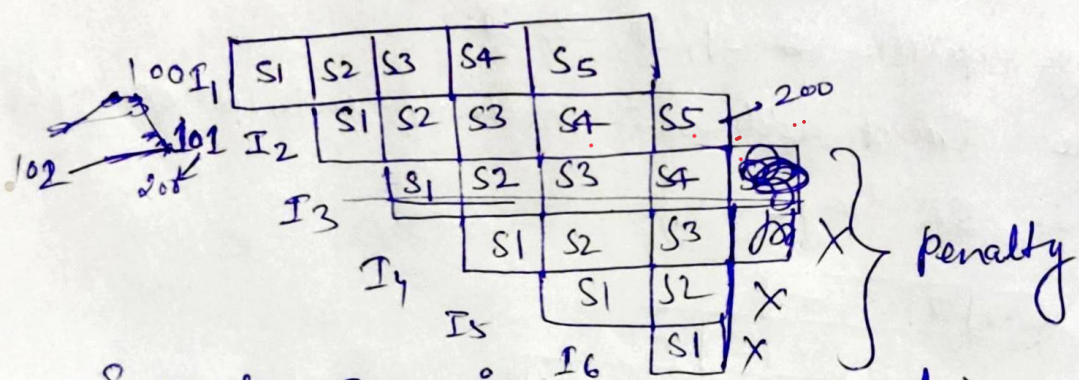


Solution: ~~diff~~ Replicate Resource

Memory — data memory
Instruction Memory



Control ~~depends~~ dependencies



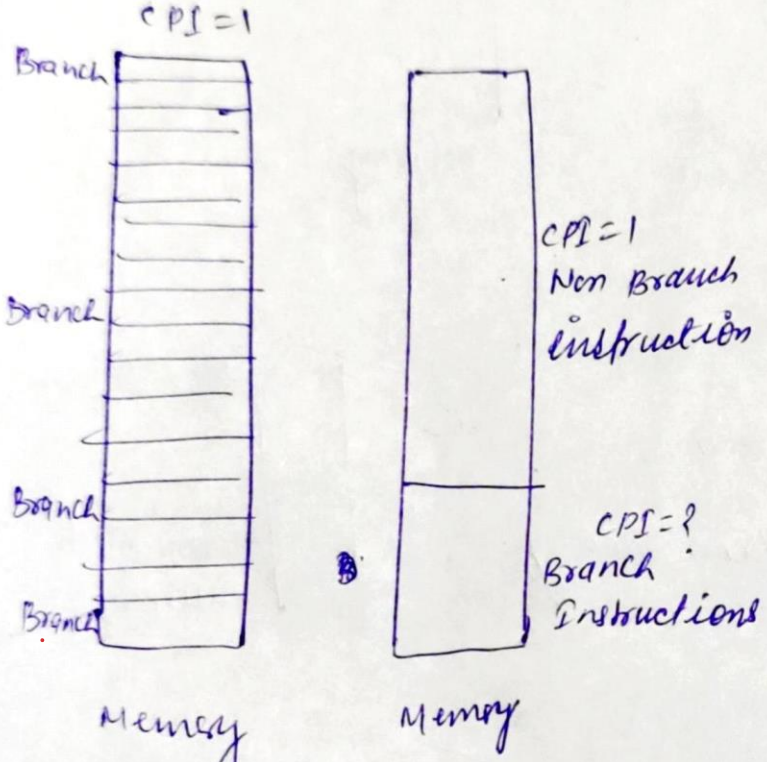
Suppose I2 is Branch instruction.
due to branching we have to flush I_3, I_4, I_5 and I_6 execution.

Solution

(i) Flushing
(ii) Stalling
(iii) Nop (no operation)
Compiler insert Nop

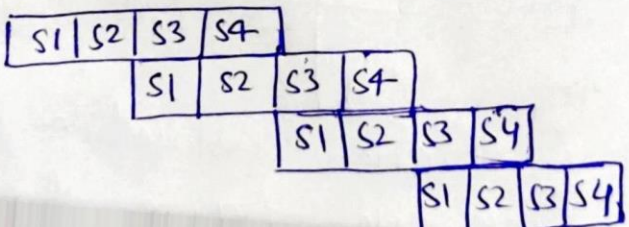
```
101  BZ  200
102  NOP
103  NOP
104  NOP
105  —
```

(iv) Reschedule & Rearrange
Compiler job to reschedule

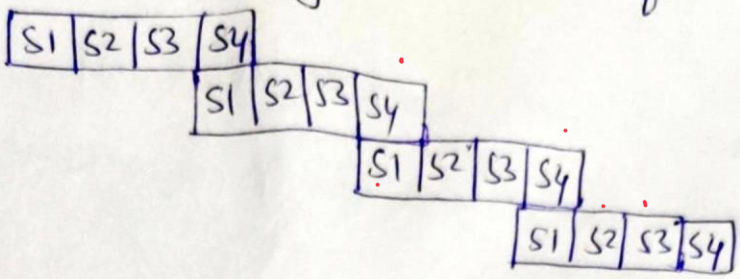


Suppose 4 stage pipline ~~an~~ S1, S2, S3, S4

If we know the traget address of Branch instruction after Completion of S2 Stage



If we know the target address after S3 stage



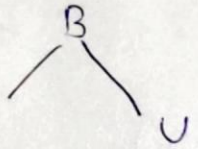
CPI = 3

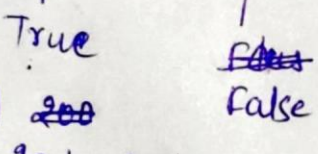
If we know the target address after S4 stage

CPI = 4

Branch instructions ~~are~~ if → Conditional (C)

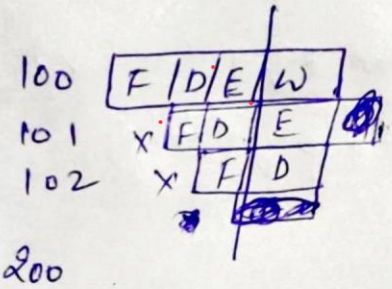
→ Unconditional (U)





if (a>b) ~~goto~~ goto 200

CPI ≠ 1 → C(T) Conditional True



If we know the target after E.

Assume, there are n no of instruction in program

probability of Conditional Branching = p

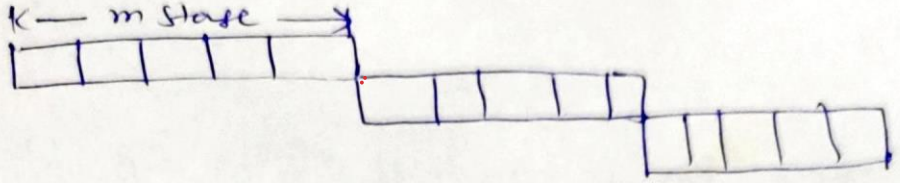
probability of Condition true = q

no of Stage = m, We get target Address after m^{th} stage

effective CPI ?

Total Conditional branch instruction = np

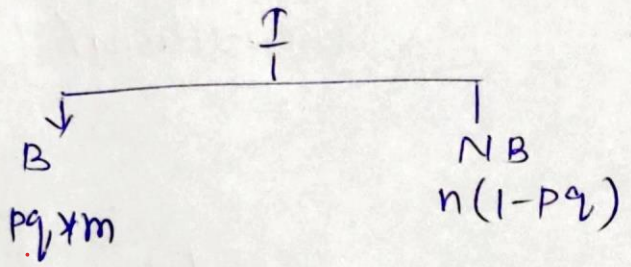
Total instruction for which condition true = npq



Total instruction which will not branch

$= n - npq$

$= n(1-pq)$



B: Branch

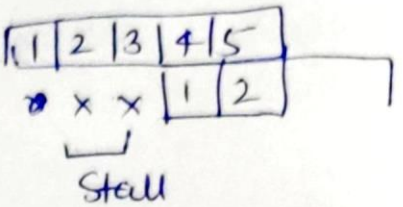
NB: Non Branch

$$\text{effective CPI} = \frac{npq * m + n(1-pq)}{n}$$

$$= mpq + (1-pq)$$

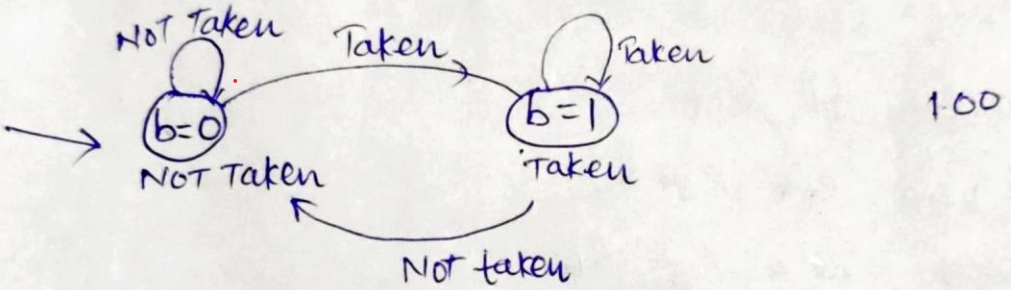
or $= 1 + pq(m-1)$

Ways to avoid Stall Cycles.

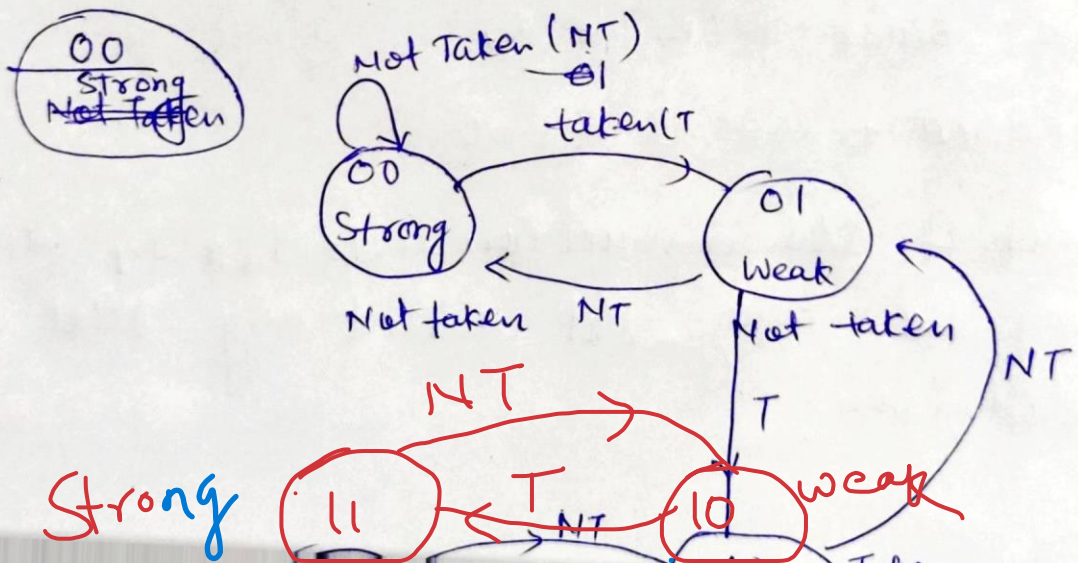


- Branch Prediction and speculative Execution.

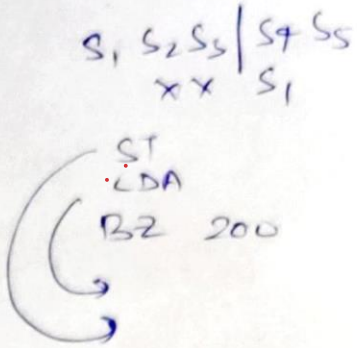
- Branch always taken (100, 200, 201, 202 . . .)
- Branch Never taken
- Dynamic Prediction
  - Bit implementation
  - BPB implementation

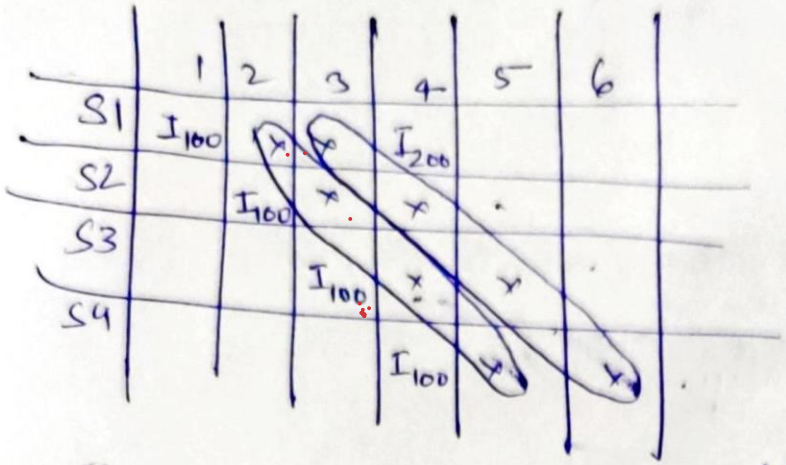


100



Delayed Branching





(i) Program Meaning should not change

(ii) Smart compiler.

The following code is run on a pipelined processor with one delay slot

I1 :- ADD $R_2 \leftarrow R_7 + R_8$

I2 : SUB $R_4 \leftarrow R_5 - R_6$

I3 : ADD $R_1 \leftarrow R_2 + R_3$

I4 : STORE $M[R_4] \leftarrow R_1$

BRANCH to label if $R_1 == 0$

Which of the instruction I_1, I_2, I_3 & I_4 can legitimately occupy the delay slot without any other program modification?

Ans: I_4