Page No.
Date. / /

CPU Time Calculation / Program ET

Program ET = # seconds/Program
= # Instruction/Program × # cycle/instruction × # second/cycle

- # Instruction/Program → Instruction Count
- # cycle/instruction → Cycle per instruction
- # second/cycle → Cycle time

$$\text{Program ET} = Ic \times CPI \times \text{Cycle time}$$

* Program is a combination of data transfer, data manipulation, & transfer of control (TOC) instruction. Different instruction takes (consume) different cycle to complete the execution so,

$$\text{Program ET} = \left[\sum (ICi \times CPIi)\right] \text{Cycle time}$$

i = instruction.

Let CPI_i be the number of cycles required for the instruction type i and I_i be the number of executed instructions of type i for a given program. Then we can calculate an overall (average) CPI as follows:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{Ic}$$

The processor time T needed to execute a given program can be expressed as $T = Ic \times CPI \times \tau$ (where $\tau = 1/f$, and Ic = instr. count).

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{Ic}{T \times 10^6} = \frac{f}{CPI \times 10^6} \quad \text{MIPS}$$

Page No.
Date. / /

Q2. Common data question 1, 2, and 3. Consider a 1.5 GHz clock frequency processor used to execute the following program segment

Instruction type	Instruction count	CPI
Load	300	11
Store	200	9
Arithmetic	250	7
Shift	150	6
Branch	50	4

(i) What is the average instruction execution?

Ans Avg CPI = $\frac{300\times11 + 200\times9 + 250\times7 + 150\times6 + 50\times4}{950}$

= 8.36 cycles

Cycle time = $\frac{1}{1.5G}$ sec

Avg Instruction cycle = 8.36 cycle

= 8.36 × 0.66 nsec = 0.66 nsec

Avg Inst. ET = 5.51 nsec

(ii) What is the mips rate of a program?

Ans 1 Instruction Avg. ET = 5.51×10^{-9} sec

1 Instr. = 5.51×10^{-9} sec

To 1 sec = # Instr. = $\frac{1}{5.51\times10^{-9}}$ Instr/sec

= $\frac{1}{5.51}\times10^{9}$ Instr/sec = 0.1814×10^{9} Inst/sec

= 181.4 MIPS

(iii) What is the total program ET?

Ans Avg Inst. ET = 5.51 nsec, Program contains 950 instructions

Program ET = 950 × 5.51 nsec

= 5234.5×10^{-9}

= 5.234×10^{-6}

Program ET = 5.234 usec

Page No.
Date. / /

Q3. 1 nsec clock cycle Processor consume 4 cycle for load and store Operation and 6 cycle for ALU operation and 2 cycle for branch Operation. The relative frequency of these Operation are 40%, 40% and 20% respectively

(i) What is the average Instruction ET?

Ans Avg CPI = .40×4 + .40×6 + .20×2
= 1.6 + 2.4 + 0.4

Avg CPI = 4.4 cycle

Avg. Instruction ET = 4.4 × 1nsec
= 4.4 nsec

(ii) What is the Performance in terms of MIPS?

1 Instruction = 4.4×10^{-9} sec

In 1 sec = $\frac{1}{4.4} \times 10^9$ Inst^s/sec

= $\frac{1000}{4.4} \times 10^6$ Inst^s/sec

= 227.2 MIPS

(iii) If program contains 10^6 instructions then what is total program ET?

1 Instruction = 4.4×10^{-9} sec

Program contains = 10^6 Instructions

Program E.T. = $4.4 \times 10^{-9} \times 10^6$

= 4.4×10^{-3}

= 4.4 msec

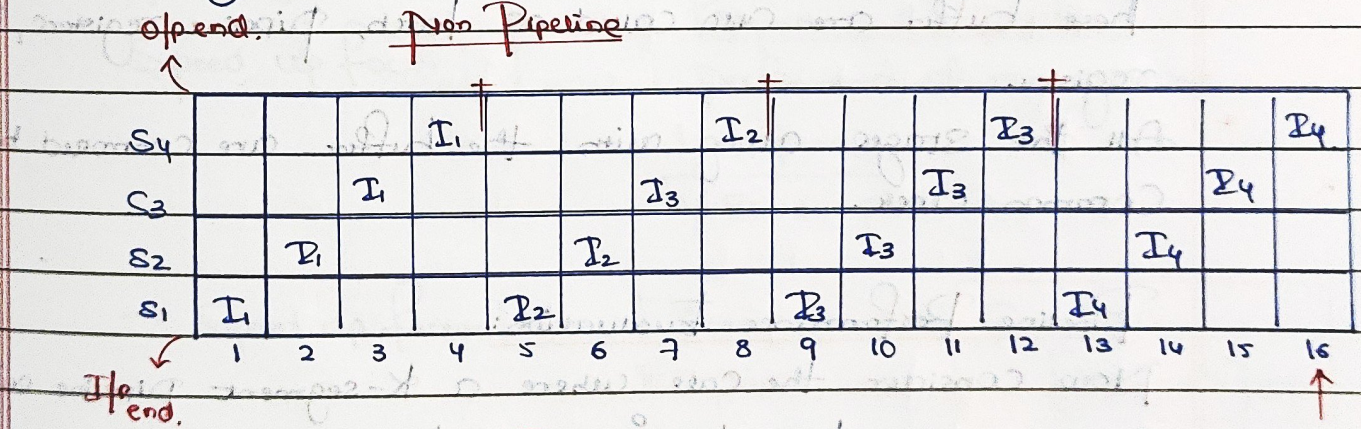
Q4. Consider a 2GHz clock cycle processor which consume 9 cycle for load and store instructions and 7 cycle for ALU instruction and 3 cycle for branch instructions. Runtime frequency of these instruction are 40%, 40% and 20% relatively. Processor is enhanced with, an average CPI of 1. During the enhancement

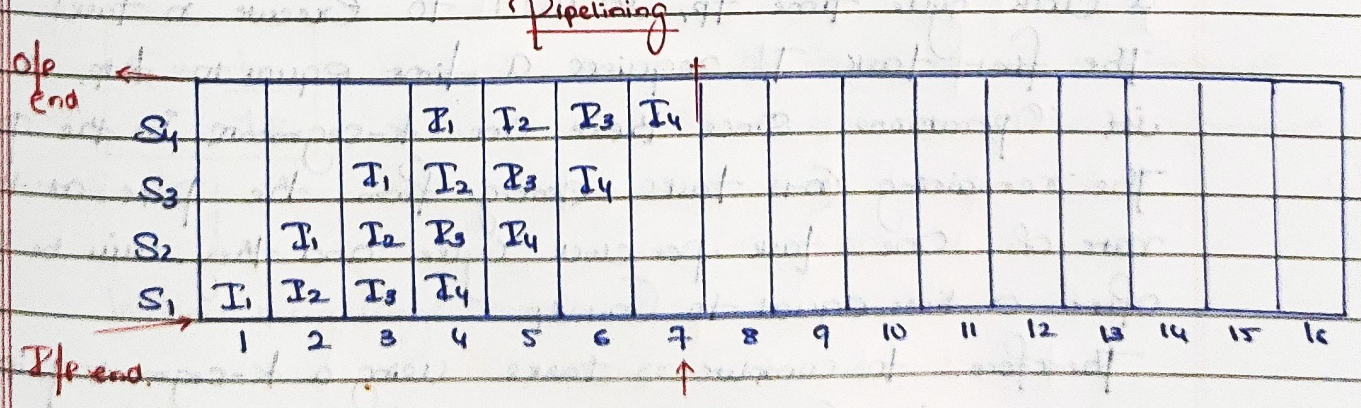
INSTRUCTION PIPELINING

Pipelining

* Pipeline is a mechanism which are used to improve the performance of the system, in which task are executed in an overlapping manner.
* Pipelining is a decomposition technique that means the problem is divided into sub problem and assign the sub problem to the pipes then operate the pipe under the same clock.
* Accepting new input before the previously accepted input appear as a output at other end. i.e it means new input are executed along with old input in a overlapping manner.

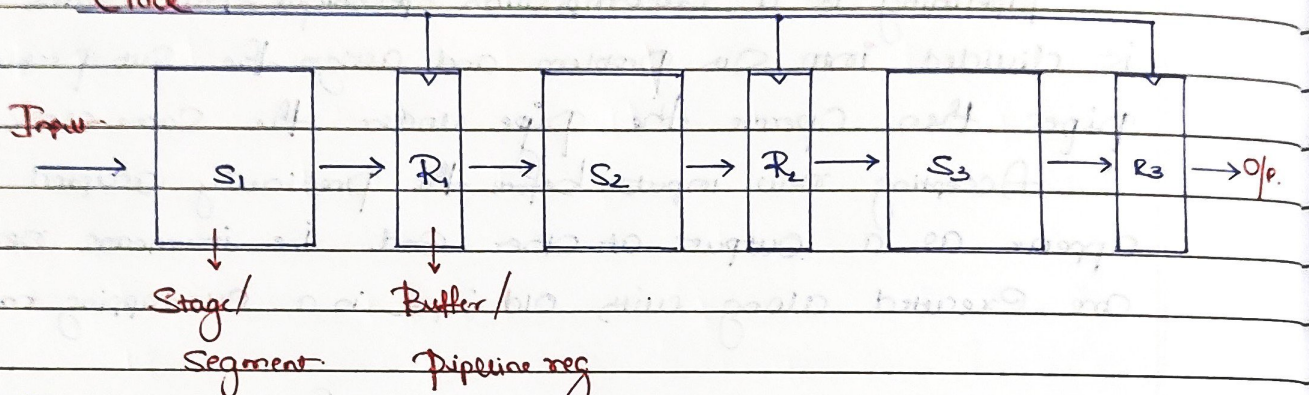
Non Pipeline: Accepting new input after the previously accepted input appear as a output at other end. In which non overlapping execution occur.





Pipeline Design:

* Pipeline has 2 ends → Input-end, Output-end
* Between the input and output end multiple Pipes are interconnected to satisfy the functionality of the pipeline.



* These pipes are called Stage or Segment.
* Between the Stages buffer are used to store the intermediate result.
* These buffer are also called as latch, pipeline registers, interface registers.
* All the stages along with the buffer are controlled by Common clock.

Pipeline Performance Evaluation:

Now consider the case where a k-segment pipeline with a clock cycle time t_p, is used to execute n tasks.
The first task T_1 requires a time equal to kt_p, to complete its Operations since there are k-segments in the pipe.
The remaining (n-1) tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to $(n-1)t_p$.
Therefore, to complete n tasks using a k-segment pipeline requires $k+(n-1)$ clock cycles

ET. pipe = $k * tp + (n-1)\, tp$

$$ET_{pipe} = [k + (n-1)]\, tp$$

k : # Stages (segment)
n : # instruction
tp : each Stage Delay in Pipeline.

Execution Time in Non-Pipeline :

$$ET_{Non\ pipeline} = n * t_n$$

t_n : each instruction Execution time in non pipeline
n : # instruction (tasks)

Performance Gain of Pipeline Processor

Performance Gain
or
Speed up factor [S] $= \frac{\text{Performance of Pipeline}}{\text{Performance of non pipeline}}$

$$= \frac{1/ET_{pipe}}{1/ET_{non\ pipe}}$$

→ Speed up factor $[S] = \frac{ET_{non\ pipe}}{ET_{pipe}} \Rightarrow S = \frac{n\, t_n}{[k + (n-1)]\, tp}$

$S = \frac{t_n}{tp}$ As the number of tasks increase, n becomes much larger than $k-1$, and $k+n-1$ approaches the value of n.

↓

When very large number of task/instruction executed
or
When no. of task is not given.

eg: $k=4,\ n=1$

t_p = (Stage delay) = 2nsec.

ET pipe = $[4+(1-1)] \times 2$nsec.

= 8nsec

Pipeline

$t_n = 2+2+2+2 = 8$nsec

ET non pipe = $n \cdot t_n$

$= 1 \times 8$nsec

= 8nsec

Non-pipeline

Note: In ideal case when pipeline each Stage are perfectly balanced then One task ET in non pipeline

$ET \cdot NP = K * t_p$

eg: $K=4$ 1 Task ET in non pipeline

t_p = 2 cycle (2 cycle in each Stage)

$ET_{NP} = 2+2+2+2$

ET_{NP} = 8 cycle

$ET \cdot NP = K * t_p$

$= 4 \times 2$

$ET \cdot NP$ = 8 cycle

eg: 1 Task ET in non Pipeline

(1 cycle in each Stage)

$ET_{NP} = 1+1+1+1 =$ 4 cycle

$ET \cdot NP = K * t_p$

$= 4 \times 1 \implies ET \cdot NP = 4$ cycle

Note: This only happens in the condition when Pipeline are perfectly balanced

↓

Each Stage takes same amount of time/cycle.

$S = \frac{K * t_p}{t_p}$

$S = K$

Throughput

$[k+(n-1)]\,tp$ time unit n task completed

$$\text{In 1 time unit (Throughput)} = \frac{n}{[k+(n-1)]\,tp}$$

Note: In pipeline when number of tasks are not given Then number of task completed per unit time

$$\text{Throughput} = \frac{1}{tp}$$

Efficiency: $\eta = \frac{S}{k}$ S: Speed up factor, k: number of Storage (segment)

Types of Pipeline:

1. Uniform delay Pipeline
2. Non uniform delay Pipeline

1. Uniform Delay Pipeline: In uniform pipeline each storage taking same amount of time/delay to execute the assigned Operations/task.

1 Task ET in pipeline	1 Task ET in non pipeline	If buffer delay is included:
$k = 4$ (stage)	$ET_{NP} = n\,tn$	$tp = (\text{Storage delay} + \text{buffer delay})$
$tp = (\text{storage delay}) 2$	$tn = 2+2+2+2$	$tp = 2+1 = 3\,nsec$
$n = 1$	$= 8\,nsec$	$ET_{pipe} = [4+(1-1)]\,3$
$ET_{pipe} = [k+(n-1)]\,tp$	$ET_{NP} = 1 \times 8\,nsec$	$ET_{p} = 12\,nsec$
$= [4+(1-1)] \times 2\,nsec$	$ET_{NP} = 8\,nsec$	$ET_{NP} = 8\,nsec$
$= 8\,nsec$		

1 Task ET in Pipeline = 1 Task ET in non pipeline.

② Non Uniform delay pipeline: In non uniform delay pipeline each stages take different amount of time (delay) to complete the assigned task/operation.

1 Task ET in pipeline	1 Task ET in non pipeline	If buffer delay is included:
$k=4,\ n=1$	$ET_{NP} = n t_n$	$t_p = \max(\text{stage delay} + \text{buffer delay})$
$t_p = \max(\text{stage delay})$	$t_n = 2+8+4+2$	$= \max((2+1),(8+1),(4+1),(2+1))$
$t_p = \max(2,8,4,2)$	$= 16$ nsec	$= 9$ nsec
$t_p = 8$ nsec	$ET_{NP} = 1 \times 16$ nsec	$ET_{pipe} = [4+(n-1)] \times 9$
$ET_{pipe} = [4+(1-1)]\,8$ nsec	$ET_{nonpipe} = 16$ nsec	$ET_{pipe} = 36$ nsec
$ET_{pipe} = 32$ nsec		

When number of task/instructions increase:

if $n = 1000$ in previous question. (above one)

$ET_{pipe} = [k+(n-1)]\,t_p$

$= [4+(1000-1)] \times 8$

$ET_{pipe} = 8024$ nsec

$ET_{nonpipe} = n t_n$

$t_n = 2+8+4+2 = 16$ nsec

$ET_{nonpipe} = 1000 \times 16$

$ET_{nonpipe} = 16000$ nsec

Important Points:

① When pipelines are perfectly balanced (uniform delay) then 1 task (instruction) execution time in pipelined is same as 1 task E.T. in non pipelined.

② When pipeline are not perfectly balanced (non uniform delay) then 1 task execution in pipeline is generated/equal to 1 Task ET in non pipeline

T_1: one task ET in pipeline $\quad T_1 \geq T_2$

T_2: One task ET in non pipeline

③ When the number of task are increased then pipeline performance is best compared to non pipeline.

4. Buffer delay is not included in non pipeline, because in non pipeline we are not storing the intermediate result. So buffer delay is included only in pipeline when it is given in question.

Uniform delay	Non uniform delay
t_p = Storage delay + buffer delay	t_p = max (Storage delay + buffer delay)

Q1. Consider an instruction pipeline which a speed up factor 20 while operating with 80% efficiency. What could be the number of stages in the pipeline?

Ans $\eta = \frac{S}{K} \Rightarrow 80\% = \frac{20}{K}$

$K = \frac{20}{80} \times 100 \quad K = 25$

Q2. A pipeline is having speed factor as 10 and operating with efficiency of 80%. What will be the number of stages in the pipeline?

Ans $\eta = \frac{S}{K} \Rightarrow 80\% = \frac{10}{K}$

$K = \frac{10}{80} \times 100 \quad K = 12.5 \simeq 13$

Q3. 4 segment pipeline have the respective stage delay of 10ns, 15ns, 20ns, and 30ns. What is the efficiency of the pipeline when very large number of task are executed?

Ans $K = 4, \quad t_p = 10 + 15 + 20 + 30 = 75$ nsec

$S = \frac{t_n}{t_p} = \frac{75}{30} = 2.5$

t_p = max (Stage delay)

= max (10, 15, 20, 30)

$t_p = 30$ nsec.

$\eta = \frac{S}{K} = \frac{2.5}{4} = 0.625 = 62.5\%$

Q4. A 4 segment instruction pipeline has the respective stage delay of 8ns, 11ns, 20ns, 2ns respectively. The interface register are used between the stages have a delay of 2ns. What is the approx Speed up value when very large number of tasks are pipelined?

Ans $k=4$. Stage delays = (8, 11, 20, 2) Buffer delay = 2nsec.

$t_n = 8+11+20+2 = 41$ nsec.

$t_p = \max(\text{stage delay} + \text{buffer delay}) = 20+2 = 22$

$S = \frac{t_n}{t_p} = \frac{41}{22} = 1.86$

Q5. Consider 4 stage pipeline with the respective stage delay of 20ns, 80ns, 50ns, and 10ns. To the enhancement process of the pipeline largest stage is split into 2 equal stage delay:

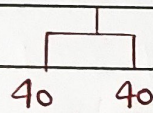
(i) What is the Speedup between new and old design?

Ans 4 stages. Stage delay = (20ns, 80ns, 50ns, 10nsec)

$t_p = 80$ nsec.

New design: $k=5$ stage.

larger space



Stage delay = [20 nsec, 40 nsec, 40 nsec, 50 nsec, 10 nsec]

t_p new = 50 nsec.

$S = \frac{\text{performance of old}}{\text{performance of new}} = \frac{80}{50}$

$S = 1.6$

(ii) What is the clock frequency of the new pipeline?

Ans Frequency $\propto \frac{1}{\text{cycle time}}$

$\text{Frequency} = \frac{1}{50 \times 10^{-9}} = \frac{1}{50} \times 10^9 = \frac{1000 \times 10^6}{50}$

$= 20$ MHz

Q6. The Stage delay in 5 Stage Pipeline are 900, 600, 550, 450 and 400 nanoseconds. The larger Stage delay is replaced with a functionally equivalent design involving two Stages with respective Stage delay 440 and 460 nanoseconds. What is the throughput increase of the Pipeline?

Ans

Old design:

$k=5$, Stage delay = (900, 600, 550, 450, 400)

t_p = max(Stage delay)

= 900 nsec.

1 Instr in old pipe = 900×10^{-9} sec

In 1 sec # instr. ext. = $\frac{1}{900 \times 10^{-9}}$

$$\text{Throughput} = \frac{1}{t_p} = \frac{1}{900 \text{ nsec}}$$

New design:

$k=6$

Stage delay = (440, 460, 600, 550, 450, 400)

t_p = max(delay)

$t_{p\,new}$ = 600 nsec

$$\text{Throughput}_{new} = \frac{1}{600}$$

$$\% \text{ of throughput increment} = \frac{new - old}{old.} \Rightarrow \frac{1/600 - 1/900}{1/900} = \frac{1/6 - 1/9}{1/9}$$

$$= \frac{9-6}{54} \Big/ 9 = \frac{3}{54} \times \frac{9}{1} = \frac{27}{54} = \frac{1}{2}$$

$$= 50\%$$

Q7. A 4-stage pipeline has the stage delays as 150, 120, 160 and 140 nsec respectively. Registers that are used between the stages have a delay of 5 ns each. Assuming constant clocking rate, the total time taken to process 1000 data items on the pipeline will be

Ans $n=1000$ $k=4$ t_p = max(Stage delay + buffer delay)

t_p = 165 nsec. (160+5).

$ET_{pipe} = [k+(n-1)]t_p = [4+(1000-1)] \times 165 \times 10^{-9}$

$= 1003 \times 165 \times 10^{-9}$

$= 165.5 \times 10^{-6} \Rightarrow$ 165.5 micro seconds

Q8. We have two designs D1 and D2 for a synchronous pipeline processor. D1 has 5 pipeline stages with execution times of 3, 2, 4, 2, and 3 nsecs while the design D2 has 8 pipeline stages each with 2nsec execution time. How much time can be saved using design D2 over design D1 for executing 100 instructions?

Ans

Design D1

$K = 5$, Stage delays $= (3, 2, 4, 2, 3)$

$t_p = \max(\text{stage delays}) = 4$ nsec.

$ET_{D1} = [K + (n-1)]\, t_p$

$= [5 + (100-1)] \times 4$

$ET_{D1} = 416$ nsec

Design D2

$K = 8$, $t_p = 2$ns $n = 100$

$ET_{D2} = [K + (n-1)]\, t_p$

$= [8 + (100-1)] \times 2$

$ET_{D2} = 214$ nsec

Time Saved $= 416 - 214 \Rightarrow 202$ nsec.

Q9. A five stage pipeline has stage delays of 150, 120, 150, 160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each.
The total time to execute 100 independent instructions on the pipeline, assuming there are no pipeline stalls, is 17160 nsec.

Ans $K = 5$ Stage delay $= (150, 120, 150, 160, 140)$

buffer delay $= 5$ nsec

$t_p = \max(\text{stage delay} + \text{buffer delay}) = 160 + 5$

$= 165$ nsec

$ET_{pipe} = [K + (n-1)]\, t_p$

$= [5 + (100-1)] \times 165$

$= 104 \times 165 = 17,160$ nsec

Q10. Consider a 3-stage pipelined processor having a delay of 100s, 20ns and 14ns for the first, second and third stages respectively. Assume that there is no other delay and the

Processor does not suffer from any pipeline hazards. Also assume that one instruction is fetched every cycle.
The total execution time for executing 100 instructions on this processor is _____ ns.

Ans K=3, Stage delay = (10, 20, 14 nsec) n=100

$t_p = \max(10, 20, 14) = 20$

$ET_{pipe} = [k + (n-1)] t_p$

$= [3 + (100-1)] \times 20$

$= 102 \times 20 = 2040$ nsec

Q11. Consider the following processors. Assume that the pipeline registers have zero latency.

P_1: Four stage pipeline with stage latencies 1ns, 2ns, 2ns, 1ns.

P_2: " " " " " " 1ns, 1.5ns, 1.5ns, 1.5ns

P_3: Five stage " " " " 0.5, 1, 1, 0.6, 1 ns

P_4: " " " " " 0.5, 0.5, 1, 1, 1.1 ns.

Which processor has the highest clock (peak) frequency?

Ans $P_1 \Rightarrow t_{p1} = \max(1, 2, 2, 1) = 2$ nsec

$P_2 \Rightarrow t_{p2} = \max(1, 1.5, 1.5, 1.5) = 1.5$ nsec

$P_3 \Rightarrow t_{p3} = \max(0.5, 1.1, 0.6, 1) = 1.1$ nsec

$P_4 \Rightarrow t_{p4} = \max(0.5, 0.5, 1.1, 1.1) = 1.1$ nsec.

$\text{Frequency} \propto \frac{1}{\text{Time}}$ $\therefore$ P_3 has highest clock frequency.

Q12. The stage delays in a 4-stage pipeline are 800, 500, 400 and 300 picoseconds. The first stage (with delay 800 picoseconds) is replaced with a functionally equivalent design involving two stages with respective delays 600 and 350 picoseconds. The throughput increase of the pipeline is 33.33 percent.

Ans Old design: $t_{pold} = \max(800, 500, 400, 300) = 800$ psec

$\text{throughput} = \frac{1}{800}$

New design: $t_{p\,new} = \max(600, 350, 500, 400, 300) = 600$ ps

$\therefore$ Throughput $= \frac{1}{600}$

% Increment in throughput $= \frac{\text{New} - \text{Old}}{\text{Old}} = \frac{1/600 - 1/800}{1/800}$

$= \frac{8-6}{48} \times \frac{8}{1} = \frac{2}{48} \times \frac{8}{1} = \frac{16}{48_3} = \frac{1}{3}$

$= 33.3\%$

Note: In the Successful Pipeline

Cycle Per instructions = 1.

$\because$ for n instructions $ET_{pipe} = [k + (n-1)]$ cycle

1 instructions $= \frac{k + n - 1}{n} = \frac{n}{n} = 1$ ($k-1$) is ignored

$\therefore CPI = 1$

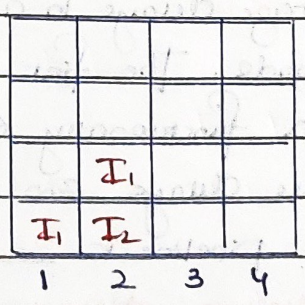
Q1. How to Construct (design) the pipeline?

Ans If we want to construct 'N' Stage Pipeline, the entire CPU (Control Unit) is divided into 'N' functional unit (Independent functional Unit) which is independent from each other.

Independent function Unit means when One functional Unit performing one task, at the same time other functional Unit performing the other task/operation.

eg: Instruction Pipeline

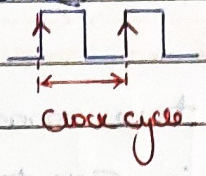
At clock cycle no.2 I_1 is in decode Stage and I_2 is in fetch Stage at the same time (clock cycle 2)



Q2. What is the meaning of CPI=1 in Pipeline?

Ans Characteristics of Pipeline is in every new cycle new input must be inserted into the pipeline.

CPI = 1.



Q3. Why Clock is required?

Ans Because whenever we enable the clock then operations are performed and to provide the synchronization between the stages.

Q4. How to set this CPI in uniform delay Pipeline?

Ans. Uniform delay: **t_p = Stage delay**

or

If buffer delay is given: **t_p = Stage delay + buffer delay.**

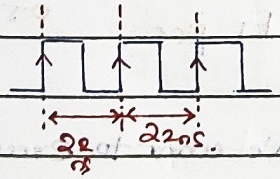
Q5. How to set this CPI in non uniform delay Pipeline?

Ans **t_p = max (Stage delay)**

If buffer delay is included: **t_p = max (Stage delay + buffer delay)**

Q6 Why don't we take minimum in calculating t_p?

Ans



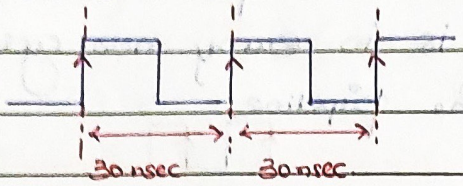
$t_p = 20 + 2 = 22$ nsec

$F = \frac{1}{22}$ nsec.

If we take minimum (22 nsec) then Stage 1 work finish and data move from Stage 1 to stage 2. But in 22 nsec Stage 2, Stage 3 and Stage 4 work not finish.

So for proper functioning and working we take maximum

If we take maximum



$t_p = 28 + 2$
$= 30 nsec.$

$F = \frac{1}{30 nsec}$ Hz

If we take maximum 30 ns but stage 1 task finish at 22 nsec But clock is set to be 30 nsec. Every phase output will be available after 30 nsec. } Synchronization

When one cycle complete the data given from stage 1 to stage 2 so Synchronization

Q1. Consider a non-pipelined processor with a clock rate of 2.5 GHz and average cycles per instruction of four. The same processor is upgraded to a pipelined processor with five stages but due to the internal pipeline delay the clockSpeed is reduced to 2 GHz Assume that there are no Stalls in the pipeline. The speedup achieved in the pipelined processor is _____

Ans

Non pipeline

Each instruction takes = 4 cycle

Cycle time = $\frac{1}{2.5}$ GHz

Cycle time = 0.4 nsec

ET in non pipeline (t_{np}) = 4 x 0.4 nsec
= 1.6 nsec

Pipeline

Each instruction takes 1 cycle

CPI = 1 cycle

Cycle time = $\frac{1}{2 GHz}$ = 0.5 nsec

t_p = 0.5 nsec

$S = \frac{ET_{non\ pipe}}{ET_{pipe}} = \frac{1.6}{0.5} = 3.2$

Q2. Consider a 4 stage Pipeline processor. We want to execute a loop: For (i=1; i<1000; i++) {I_1, I_2, I_3, I_4} where the time taken (in ns) by instruction I_1 to I_4 for stage S_1, S_2, S_3, S_4 is shown below

	S_1	S_2	S_3	S_4
I_1	1	2	1	2
I_2	2	1	2	1
I_3	1	1	2	1
I_4	2	1	2	1

The output of I_1 for i=2 will be available after ?

Ans.

	S_1	S_2	S_3	S_4
I_1	1	3	4	6
I_2	3	4	6	7
I_3	4	5	8	9
I_4	6	7	10	11
I_1	7	9	11	(13) ← Ans.

Output of I_i for $i=2$ will be 13.

Q3 Consider a 4-stage pipeline processor. The number of cycles needed by the four instructions I_1, I_2, I_3, I_4 in stages S_1, S_2, S_3, S_4 is shown below:

	S_1	S_2	S_3	S_4
I_1	2	1	1	1
I_2	1	3	2	2
I_3	2	1	1	3
I_4	1	2	2	2

What is the number of cycles needed to execute the following loop: for (i=1 to 2) {I_1, I_2, I_3, I_4}.

Ans

	S_1	S_2	S_3	S_4
I_1	2	3	4	5
I_2	3	6	8	10
I_3	5	7	9	13
I_4	6	9	11	15
I_1	8	10	12	16
I_2	9	3	15	18
I_3	11	14	16	21
I_4	12	16	18	(23) ← Ans

23 cycles will be needed to execute the loop.

Q4. Consider a 3GHz processor with a three-stage pipeline and stage latencies τ_1, τ_2, τ_3 such that $\tau_1 = 3\tau_2/4 = 2\tau_3$. If the longest pipeline stage is split into two pipeline stages of equal latency, the new frequency is 4 GHz, ignoring delays in the pipeline registers.

Ans. $T_1 = \frac{3T_2}{4} = 2T_3$

$T_1 : T_2 : T_3 = 6 : 8 : 3$

$T_1 = \frac{3T_2}{4}$ $\quad \frac{3T_2}{4} = 2T_3$

$\frac{T_1}{T_2} = \frac{3}{4}\left[\frac{6}{8}\right]$ $\quad \frac{T_2}{T_3} = \frac{8}{3}$

Assume time = x

$T_1 = 6x$, $T_2 = 8x$, $T_3 = 3x$

$tp = \max(6x, 8x, 3x) = 8x$

Frequency $= \frac{1}{\text{Time}}$

$3\text{ GHz} = 1/8x$

$\frac{1}{x} = 24\text{ GHz}$ — eq①

New design:

8x → 6x, 4x

$tp_{new} = \max(6x, 4x, 4x, 3x)$

$tp_{new} = 6x$

Frequency $= \frac{1}{\text{Time}}$

$= \frac{1}{6x} \Rightarrow \frac{1}{6} \times \frac{1}{x}$

$= \frac{1}{6} \times 24\text{ GHz}$

$\therefore 4\text{ GHz}$

Qs. Consider two processors P_1 and P_2 executing the same instruction set. Assume that under identical conditions, for the same input, a program running on P_2 takes 25% less time but incurs 20% more CPI (clock cycles per instruction) as compared to the program running on P_1. If the clock frequency of P_1 is 1GHz, then the clock frequency of P_2 (in GHz) is

Ans $ET = IC \times CPI \times \text{Cycle time}$

In the question same no. of instructions

$ET = \text{cycle/instruction} \times \text{cycle time}$

$ET_{P1} = CPI \times \text{cycle time}_1$

$ET_{P2} = 1.2\ CPI \times \text{cycle time}_2$

$ET_{P3} = 0.75 \times ET_{P1}$

Cycle time $P_1 = \frac{1}{1\text{ GHz}} = 1\text{ nsec}$

$0.75 \times ET_{P1} = 1.2CPI \times \text{cycle time}_2$

$0.75 \times CPI \times 1\text{nsec} = 1.2CPI \times \text{cycle time}_2$

Cycle time $= \frac{0.75}{1.2} = 0.625$ nsec

Clock frequency $= \frac{1}{0.625 \text{ nsec}} = 1.6$ GHz

Q6. Consider a machine with 40 MHz processor which has run a benchmark program. The executed program consist of 100,000 instruction executions, with the following instruction mix and clock cycle count. What will be the effective CPI, MIPS rate and execution time.

Instruction Type	Instruction Count	Cycle/Instruction
Integer arithmetic	45000	1
Data transfer	32000	2
Floating point	15000	2
Control transfer.	8000	2

Ans. Avg CPI $= 0.45 \times 1 + .32 \times 2 + .15 \times 2 + .8 \times 2$

Avg CPI $= 1.55$ cycle

Cycle time $= \frac{1}{40 \text{ MHz}}$

Avg. Instruction ET $= 1.55 \times \frac{1}{40} \times 10^{-6}$ sec $\rightarrow 0.03875 \times 10^{-6}$

1 Instruction ET $= 0.3875 \times 10^{-6}$ sec

In 1 sec no of instructions $= \frac{1}{0.03875} \times 10^6$ Instr/sec

$= 25.8$ MIPS

Total program ET:

$= 0.03875 \times 10^{-6} \times 10^5$

$= 3.875 \times 10^{-3}$

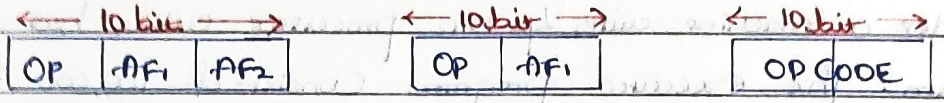
$= 3.875$ msec

Q7. Which of the following is not a form of main memory?

Ans Instruction Opcode

Q8. In a 10-bit computer instruction format, the size of the address field is 3 bits. The computer uses expanding OP code technique and has 4 two-address instructions and 16 one-address instructions. The number of zero-address instructions it can support is

Ans



Primitive Derived Further derived

Primitive: Total number of operations = 2^4 = 16 operations.

Given 2AI = 4

Free opcode after allocating 2AI = 16 - 4 = 12

Total # operations in 1AF = free opcode × 2

$= 12 \times 2^{7-4} \Rightarrow 12 \times 2^3$

$\Rightarrow$ 96

Given 1AF = 16

Free opcode after allocating 1AI = 96 - 16 = 80

Total # operations in OAF = $80 \times 2^{10-7} = 80 \times 2^3$

$\Rightarrow$ 640

Q9.

Design D_1	500ns	450ns	600ns	800ns
Design D_2	600ns	820ns	750ns	650ns

What is the cycle time latency of one instruction and throughput in the pipeline design D_1 and D_2 if in both design D_1 and D_2 having buffer register between each stages have a delay of 20 nsec? Buffer delay = 20ns

Ans

Design D_1

tp_{D_1} = max (stage delay + buffer delay)

= max (800 + 20)

tp_{D_1} = 820

Design D_2

tp_{D_2} = max (Stage delay + buffer delay)

= (820 + 20)

tp_{D_2} = 840

classmate
Date 9/10/23
Page

Latency of one instruction
$ET_{D1} = 4 \times 820 = 3280$ nsec

Throughput $= \frac{1}{tp} = \frac{1}{820ns}$

Latency of one instruction
$ET_{D2} = 4 \times 840 = 3360$ nsec

Throughput $= \frac{1}{840ns}$

→ $D_1 = (820ns, 3280ns, 1/820)$
→ $D_2 = (840ns, 3360ns, 1/840)$ ∴ Pipeline D_1 performs better than D_2.

Pipeline Stages

1. 2-Stage Pipeline | FI | EX |

2. 4-Stage Pipeline

I_1:	IF	ID	EX	WB	
I_2:		IF	ID	EX	WB
I_3:			IF	ID	EX	WB
I_4:					

IF : Instruction fetch
ID : Instruction decode
EX : Execute
WB : Write back

3. 6-Stage Pipeline ⇒ | FI | DI | CO | FO | EI | WO |

Additional Stages:

1. Fetch Instruction:
Read the next expected instruction from buffer.

2. Decode Instruction:
* Determine the opcode and the operand specifiers

3. Calculate Operands (CO)
* Calculate the effective address of each source operand
* This may involve displacement, register indirect or other forms of address calculations.

4. Fetch Operands (FO)
* Fetch each operand from memory.

* Operands in register need not be fetched

⑤ Executed Instruction (EI)

* Perform the indicated operation and store the result, if any in the specified destination operand location.

⑥ Write Operand (WO)

* Store the result in memory.

Note: Generally we use classical pipelined:

Classical Pipeline	MIPS Pipeline	Standard Pipeline	RISC Pipeline

RISC Pipeline: 5 Stage

In RISC pipeline 5 Stages:

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Execute (EX)
4. Memory Access (MA)
5. Write Back (WB)

1. Instruction fetch: In this stage memory instruction is fetched from memory.

2. Instruction decode: In this stage 2 operations are performed:
   (i) Decode the instruction
   (ii) Operand loading (fetching) from the register file.
   This stage also contain comparator circuit to evaluate the branch condition.

3. Execute: In this stage data processing (ALU operations) are performed.

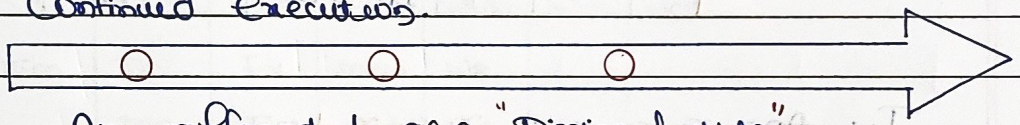
4. Memory-Access : In this stage operand (Data) will be accessed from memory (load or store).

5. Write Back : In this stage register write (Operand storing into reg. file) Operations performed.

Pipeline Hazards

Occur when the pipeline, or some portion of the pipeline must stall because conditions do not permit continued execution.

There are 3 types of hazards:
1. Resource
2. Data
3. Control.



Also reffered to as a "pipeline bubble".

→ Hazard is a situation that makes the pipeline to idle or stall.
1. Structural Hazards (Resource Conflict)
2. Data Hazards (Data/operand)
3. Control Hazards. (Branch instructions)

→ In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation:
* Resource conflicts caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
* Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but the result is not yet available.
* Branch difficulties arise from branch and other instructions that change the value of pc.

Hazards / Dependencies in the pipeline:

* Dependency is a major problem in the pipeline, Causes extra cycle.
* Cycle in the pipeline without new input is called as extra cycle. Also named as "Stall".
* When Stall is present in the pipeline then **CPI ≠ 1**

eg: 5 Stage Pipeline: Normal Executing.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I1	IF	ID	EX	MA	WB		
I2		IF	ID	EX	MA	WB	

I1: ADD R1, R2, R3 : R1 ← R2 + R3

I2: MUL R4, R1, R5 : R4 ← R1 * R5.

Instruction I2 data dependent on I1

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I1	IF	ID	EX	MA	WB		
I2		IF	ID	"	"		

R1 — Stall — Bubble / Extra Cycle.

Structural Dependency / Hazards

* Structural dependency is created in the pipeline when two or more phase require the same resource at the same time and they are not able to run simultaneously.
* Structural dependency is created in the pipeline due to resource conflict.
* Resource maybe registers, functional unit, ALU, memory etc.

eg:1.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	IFEm	ID	EX	MEm	WB		
I_2:		IFEm	ID	EX	MEm	WB	
I_3:			IFEm	ID	EX	MEm	WB
I_4:				IFEm	IFEm	IFEm	IFEm
				" ① "	" ② "	" ③ "	

3 extra cycles / bubbles / stalls.

Here I_1 is accessing the memory to store the data and I_4 is accessing the memory to fetch the instruction.

eg:2.

CC1	CC2	CC3	CC4	CC5	CC6	CC7
IFEm	ID	EX	MEm	WB		
	IFEm	ID	EX	MEm	WB	
		IFEm	ID	EX	MEm	WB
			IFEm			

↳ Resource conflict

* In clock cycle (CC4) 4 both I_1 and I_4 accessing the same resource 'memory'.
* This situation is called resource conflict so (I_1 & I_4) not go for executing.
* So keep I_4 into the waiting until the resource become available.
* This waiting create stall / bubble / extra cycle in pipeline.

Memory → Instruction / code memory [Im/em] (Instruction Cache)
Memory → Data Memory [Dm] (Data Cache)

* To minimize the stall due to structural dependency one hardware m/c instruction mechanism is used called as "renaming".

* Renaming State then divide the memory into independent module to store the instructions and data seperately called Instructions/Code Memory [Cm/Im] and data memory (Dm) respectively.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I_1 :	Cm	Id	Ex	Dm	WB			
I_2 :		Cm	Id	Ex	Dm	WB		
I_3 :			Cm	Id	Ex	Dm	WB	
I_4 :				Cm	Id	Ex	Dm.	WB

No Stall.

Data Dependency:

I_1: ADD r_1 r_2 r_3; $r_1 \leftarrow r_2 + r_3$

I_2: MUL r_4 r_1 r_5; $r_4 \leftarrow r_1 * r_5$

Instruction I_2 is data dependent on instruction I_1 because I_2 is using the result of instruction I_1 as Operand

Data dependency is created in the pipeline when next (I_j or I_{i+1}) instruction using the result of previous instruction.

1. Data Dependency/Flow dependency → RAW hazards
2. Anti dependency → WAR hazards
3. Output/Write dependency → WAW hazards

Data Hazard: A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason. Or any condition that cause the pipeline to stall is called as hazards

Data Hazards: 1. RAW (Read after write)
2. WAR (write after read)
3. WAW (Write after write)

Important Points:

* Every dependency not creates Stall.
* ANTI and OUTPUT dependency never create Stall.
* Data dependency may or may not creates Stall/bubbles/extra cycle.

Data Dependency
- (1.) True Data Dependency (Adjacent instruction dep. which creates Stalls)
- (2.) Data dependency (Which may or maynot Create Stall)

Q1. Consider the following program segment, executed on a Risc pipeline Find how many data dependency?

		Data dependency
I_1	Add r0, r1, r2	
I_2	Sub r3, r0, r2	$I_2 - I_1$
I_3	Mul r4, r0, r5	$I_3 - I_1$
I_4	Div r0, r6, r7	$I_5 - I_4$
I_5	Mul r8, r0, r9	

Q2. How to check that dependency creates hazards or not?

Ans Execution of instruction in a normal pipeline manner if there is a Stall/bubbles/extra cycle occur then data dependency creates Stall.

Q4. I_1 : Add r0 r1 r2; r0 ← r1 + r2

I_2 : Mul r3 r0 r4; r3 ← r0 * r4.

Now Check this dependency ($I_2 - I_1$) Create hazards or not?

Ans Risc : 5 Stage IF, ID, EX, MA, WB.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	IF	ID	EX	MA	WB				
I_2		IF	ID	ID	ID	ID	EX	MA	WB

3 Stalls

ID → Correct value.

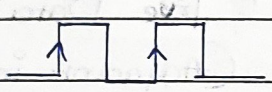
At CC3 I2 read wrong value of r0

At CC4 I2 read wrong value of r0.

At CC5 I2 read wrong value of r0

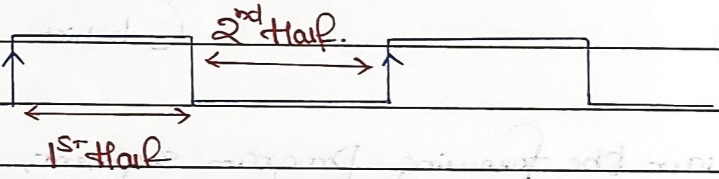
Because r0 Available at end of CC5

So 3 extra Stall/extra cycle/bubbles



At clock cycle 5 the value of r0 getting [write back into register]

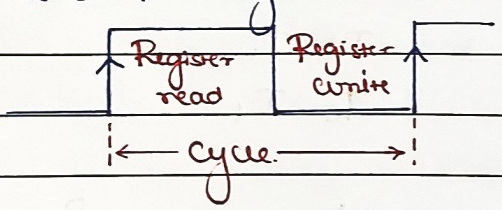
Split Phase



I Case

1st half : Reg Read

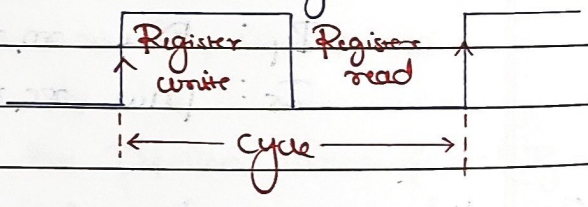
2nd half : reg write



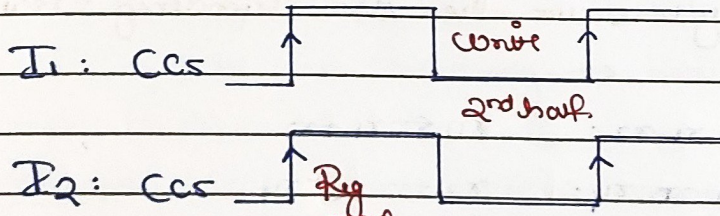
II Case

1st half : register write

2nd half : register read

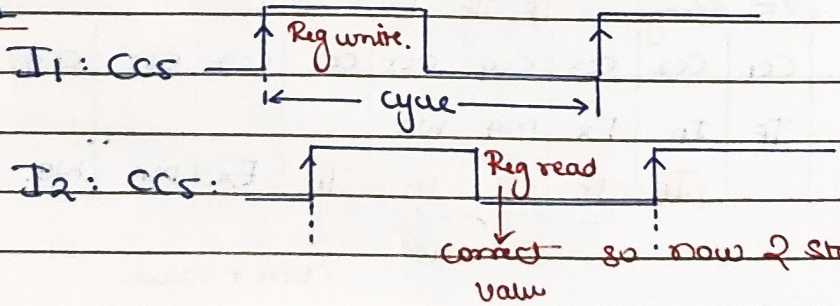


Case I in previous Question:



Old (wrong) value So 3 stalls in previous Question.

Case II



If Case of previous Question (Q4)

RISC : 5 Stage

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I_1	IF	ID	EX	MA	WB			
I_2		IF	ID	ID	ID	EX	MA	WB

2 Stalls

At CC3 I_2 read old value of r_0

At CC4 I_2 read old (wrong) value of r_0

At CC5 (2nd part) of cycle I_2 read correct value of r_0.

So 2 Stalls.

Data Dependency (Formal Definition)

* Data dependency will be occurred when the instruction 'j' try to read the data before 'i' writes it.

eg: I_1 : Add r_0, r_1, r_2 : $r_0 \leftarrow r_1 + r_2$

I_2 : mul r_3, r_0, r_2 : $r_3 \leftarrow r_0 * r_2$

* When the above instruction are executed in a non-pipelined system then data dependency condition does not occur because "I_2" executing after "I_1" so, I_2 reads the register (r_0) data I_1 writes it (Read-After-Write)

* When the above instructions are executed in a pipeline system then data dependency condition will be occurred because (I_2) will be executing along with (I_1) so (I_2) reads the (r_0) data before I_1 writes it. Therefore I_2 incorrectly get the old value from (r_0) (data loss).

	CC1	CC2	CC3	CC4	
I_1	IF	ID	EX	MA	WB
I_2		IF	ID		

So: r_0, r_2

D: r_3 r_0 = value - - - - - - Read-before write.

r_2 = value

To handle above problem, some logic will be maintained in the "Is" Stage of pipeline to stop the accessing of a dependent data. Structure of the logic is as follows: "TOMASULO ALGO".

S/No	Function Unit	Destination	Independent Source 1	Independent Source 2	Dependent Source 1	Dependent Source 2	Status	Reg. file access
I_1	ADD	R_0	r_1	r_2	—	—	0	r_1 = value r_2 = value Ex-stage started
I_2	MUL	r_3	—	r_2	$r_0(I_1)$	—	0	r_2 = value Ex-Stage not started.

"0" execution yet not complete

"1" execution complete

By using the above logic, we need to stop the access of dependent data until the data becomes available.

This waiting created extra cycle/bubble/Stall in the pipeline.

Solution of Data Dependency (Raw Hazard)

→ Hardware Solution

→ Software Solution (Compiler based)

→ No operating instruction inserted.

Hardware Solution of Data Dependency:

(1) Hardware interlocks: The most straightforward method is to insert hardware interlocks. An interlock is a circuit that detects instructions whose source operands are destinations of instructions further up in the pipeline.

(2) Operand forwarding: Another technique called operand forwarding uses special hardware to detect a conflict and then avoid

it by routing the data through special paths between pipeline segments. For eg: instead of transferring an ALU result into a destination register, the hardware register checks the destination operand, and if it is needed as a source in the next instruction, it passes the result directly into the ALU input, bypassing the register file.

* To minimize the data dependency stall in the pipeline hardware technique is used i.e Operand forwarding. Also named as By-passing or Short circuit.

* This mechanism state that use the buffer between the stage to hold the immediate result so that, dependent instructions will be accessing the new data from the buffer before update register file.

eg: I_1 : $r0 \leftarrow r1 + r2$

I_2 : $r3 \leftarrow r0 * r4$.

Without Operand forwarding:

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7	
I_1	IF	ID	EX	MA	WB			
I_2		IF	ID	ID	ID	ID	EX	WB

3 Stalls

With Operand forwarding:

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7
I_1	IF	ID	EX	MA	WB		
I_2		IF	ID	EX	MA	WB	

Buffer.

If here wrong (old) value of r0 then in CC_4

Correct value of r0 using

1. Data/Raw Dependency (Read After Write)
* An instruction modifies a register or memory location.
* Succeeding instructions reads data in memory or register location.
eg. I_i: Add $r0, r1, r2$ I_i $r0 \leftarrow r1 + r2$
I_2: Sub $r3, r0, r2$ I_j $r3 \leftarrow r0 * r2$

2. Anti Dependency/WAR Dependency (Write After Read)
* An instruction reads a register or memory location.
* Succeeding instructions writes to the location.
eg: I_1: Mul $r3, r1, r2$ I_i $r3 \leftarrow r1 * r2$
I_2: Add $r1, r4, r5$ I_j $r1 \leftarrow r4 + r5$

3. Output Dependency/WAW Dependency (Write After Write)
* Two instructions both write to the same location.
eg: I_1: Mul $r3, r1, r2$ I_i = $r3 \leftarrow r1 * r2$
I_2: Add $r2, r4, r5$ I_j = $r3 \leftarrow r4 + r5$

Types of Data Hazard:

1. Read after Write, or true dependency
* An instruction modifies a register or memory location.
* Succeeding instruction reads data in memory or register location.
* Hazards occurs if the read takes place before write operation is complete.

2. Write After Read, or antidependency
* An instruction reads a register or memory location.
* Succeeding instruction writes to the location.
* Hazards occurs if the write operation completes before the read operation takes place.

3. Write After Write or Output dependency.
* Two instructions both write to the same location.

* Hazards occur if the write operations take place in the reverse order of the intended sequence.

Q1. Consider the following Program Segment, Executed on a RISC pipeline:
(i) How many RAW dependency? (ii) How many Raw Hazards?

Ans Data (Raw) Dependency:

$I_2 - I_1$

$I_3 - I_4$ 4 Raw dependency

$I_4 - I_1$ 3 Raw Hazards.

$I_5 - I_1$

I_1: Add r0, r1, r2.
I_2: Sub r3, r0, r4.
I_3: Mul r5, r0, r6
I_4: XOR r7, r0, r8
I_5: Mul r9, r0, r10

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	IF	ID	EX	MA	WR				
I_2		IF	(ID)	EX	MA	WR			
I_3			IF	(ID)	EX	MA	WR		
I_4				IF	(ID)	EX	MA	WR	
I_5					IF	(ID)	EX	MA	WR

4 Raw Dependency
3 Raw Hazards.

After CC5 ⟹ r0 output available.

Q2. Consider the following Program Segment, Executed on a RISC Pipeline:
Check Raw dependency Create RAW Hazards?

Ans I_1: Add r0, r1, r2
I_2: Sub r3, r4, r5
I_3: Mul r6, r0, r7
I_4: Add r7, r5, r8
I_5: Add r9, r0, r10.

Raw Dependency
$I_1 - I_3$
$I_1 - I_5$

Raw Hazard : $I_3 - I_1$

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	IF	ID	EX	MA	WB				
I_2		IF	ID	EX	MA	WB			
I_3			IF	(ID)	EX	MA	WB		
I_4				IF	ID	EX	MA	WB	
I_5					IF	(ID)	EX	MA	WB

r0 available after CC5
Raw hazard : $I_3 - I_1$

Q3. How to check RAW Dependencies create RAW hazards or not?

Ans Pipeline executing then check if that result is required earlier or not by next instructions.

Q4 Consider a pipelined processor with the following four stages: IF, ID, EX, WB.

The IF, ID and WB stages take one clock cycle each to complete the operations. The number of clock cycles for the EX stage depends on the instructions. The ADD and SUB instructions need 1 clock cycle and the MUL instructions needs 3 clock cycles in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?

ADD R2, R1, R0 $R2 \leftarrow R1 + R0$

MUL R4, R3, R4 $R4 \leftarrow R3 * R2$

SUB R6, R5, R4 $R6 \leftarrow R5 - R4$

Ans Without operand forwarding:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12
I1	IF	ID	EX	WB								
I2		IF	////	////	ID	EX	EX	EX	WB			
I3			IF	//	//	//	//	//	//	ID	EX	WB

⇒ 12 clock cycles without Operand forwarding

WB				I1					I2			I3
EX			I1			I2	I2	I2			I3	
ID		I1	I2	I2	I2					I3		
IF	I1	I2	I3									
	1	2	3	4	5	6	7	8	9	10	11	12

Without Operand forwarding ⇒ 12 clock cycles.

With Operand forwarding:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	IF	ID	Ex	WB					
I_2		IF	ID	Ex	Ex	Ex	WB		
I_3			IF			ID	Ex	WB	

8 Clock cycles with Operand forwarding.

WB				I_1			I_2	I_3	
Ex			I_1	I_2	I_2	I_2	I_3		
ID		I_1	I_2	I_2	I_2	I_3			
IF	I_1	I_2							

Ans: 8 Clock Cycles

Qs. Consider the sequence of machine instructions given below:

MUL R5, R0, R1
DIV R6, R2, R3
ADD R7, R5, R6
SUB R8, R7, R4.

In the above sequence, R0 to R8 are general purpose registers. In the instructions shown, the first register shows the result of the operation performed on the second and third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages:

(1) Instruction fetch and decode
(2) Operand fetch
(3) Perform Operation
(4) Write back the result

The PO Stage takes 1 clock cycle for ADD or SUB instructions. 3 Clock cycles for MUL instruction and 5 clock cycles for DIV instruction. The pipelined processor uses operand forwarding from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instructions is 13 clock cycles.

Ans:

	1	2	3	4	5	6	7	8	9	10	11	12	13
WB						I_1					I_2	I_3	I_4
PO			I_1	I_1	I_1	I_2	I_2	I_2	I_2	I_3 ↓	I_3 ↓	I_4	
OF		I_1	I_2							I_3	I_4		
IF	I_1	I_2	I_3	I_4									

∴ 13 clock cycle

Another Approach:

n=4 (I_1, I_2, I_3, I_4) CPI=1

k=4 (4 Stage)

		Extra
IF	1	MUL 3 2
OF	1	DIV 5 4
PO	1	6 Stall
WB	1	

$ET_{pipe} = [k + (n-1)]tp$

$= [4 + (4-1)]$ cycle

$= 7$ cycle.

$ET = ET_{pipe} + Stall$

$= 7 + 6$

$= 13$ cycles (Ans)

Q6. The instruction pipeline of a RISC processor has the following stages. IF, ID, OF, PO and WB. The IF, ID, OF and WB stage take 1 clock cycle for every instruction. Consider a sequence of 100 instructions, In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycle each and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards.

The number of clock cycles required for completion of execution of the sequence of instructions is 219.

Ans. k=5, n=100

IF : 1 | ID : 1 | OF : 1 | PO : 1 | WB : 1

PO : 40 Instr. : 3
35 Instr. : 2
25 Instr. : 1

Extra (Stalls)	
2	Total stalls = 40×2 +
1	35×1
0	= 115 stalls

E1 pipe = [k+(n-1)] cycle
= [5+(100-1)] cycle
= 104 cycle

Total ET = ET pipe + Stalls
= 104 + 115
=> 219

Q7. The performance of the pipelined processor suffer if:
A. The pipeline stages have different delays.
B. Consecutive instructions are dependent on each other
C. The pipeline stages share hardware resources.
D. All of the above

Note: ANTI and OUTPUT Dependency does not create any stalls but in Anti and output we are getting the problem or for reading/updating the correct value.

The solution of Anti and Output dependency is "Register Renaming".

Register Renaming: Use some temporary storage register to store the result of the out of order instructions output.

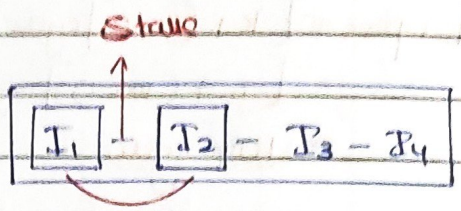
Q8. When and how Anti dependency and Output dependency create problem?

Ans (i) If the previous instruction taking more time (very high time) compare to next instruction. (Uneven delay).
(ii) When there is true data dependency between two instructions then to minimise the stalls, we try to reschedule the instructions (Out of order instruction execution).
(iii) In Super Scalar System, multiple instructions inserted in the stages of so next instruction if read/write the value that are used in previous instruction, then it create a problem.

* CPU always execute a sequence the program from top to bottom in a sequence called as in-order sequence execution.
In this execution sequence if any instruction is dependent then the remaining instructions are also sharing the stall cycles even

They are independent:

eg: I_1 : Add r0, r1, r2
I_2 : Sub r3, r0, r4
I_3 : Mul r4, r5, r6
I_4 : Div r3, r7, r8.



* In the above execution sequence I_2 is data dependent on I_1 so I_2 will be waiting until the I_1 execution is completed when the hardware does not support operand forwarding.
This waiting created stall in the pipeline this stall also shared by I_3 and I_4 instruction even they are independent.

* To utilize this point instruction scheduling concept stated that insert the independent instruction first.
It causes out of order execution (Re-order execution)

* Out of order execution sequence is $I_1 - I_3 - I_4 - I_2$
Out of order execution sequence create two more dependencies in the pipeline...
(i) Anti data dependency
(ii) Output data dependency.

* To handle the above dependency problem hardware technique is used i.e "register renaming".
This technique stated that, use the re-order buffer to store the out-of-order ~~buffer~~ instruction output later update the register file with a re-order buffer content after the completion of a dependent instruction therefore data is safe.

Control Dependency:
Control dependency is created in the pipeline due to:
* Branch instruction
* Program Control

* Function Call
* Transfer of control instructions → Conditional Toc
  → Unconditional Toc

Assume 5 Stage Pipeline and I_2 is the branch instruction. Assume the target address of the branch instruction is available at the end of the last stage.

	1000	I_1	Unconditional Toc
I_2 is a branch instr	1001	I_2	Jmp 2000
	1002	I_3	
	1003	I_4	
	1004	I_5	
	1005	I_6	
	1006	I_7	
	2000	BI_1	
	2001	BI_2	

	1	2	3	4	5	6
I_1	S_1	S_2	S_3	S_4	S_5	
I_2		S_1	S_2	S_3	S_4	S_5
I_3			S_1	S_2	S_3	S_4
I_4				S_1	S_2	S_3
I_5					S_1	S_2
I_6						S_1

4 Stalls

I_3, I_4, I_5, and I_6 is unwanted instruction

B_1 is the wanted instruction.

Target address ←

	1	2	3	4	5	6	7	8	9	10
S_5					I_1	I_2				
S_4				I_1	I_2	I_3				
S_3			I_1	I_2	I_3	I_4			BI_1	
S_2		I_1	I_2	I_3	I_4	I_5		BI_1		
S_1	I_1	I_2	I_3	I_4	I_5	I_6	BI_1			

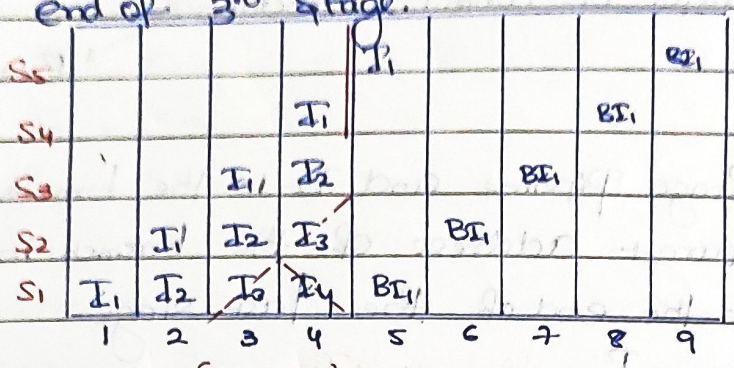
4 Stalls

Assume the target address of the branch address of branch instruction is available at end of 4th stage.

S_5					I_1	I_2				BI_1
S_4				I_1	I_2	I_3			BI_1	
S_3			I_1	I_2	I_3	I_4		BI_1		
S_2		I_1	I_2	I_3	I_4	I_5	BI_1			
S_1	I_1	I_2	I_3	I_4	I_5	I_6				

3 Stalls

Assume the target address of the branch instruction is available at the end of 3rd stage.



If the target address of branch instruction is available at k^{th} stage.

$$\#\text{Stalls from branch instruction} = k-1$$

Ideal CPI = 1

But due to control hazard / Control dependency.

$$\text{Average } \underset{ET}{\text{Instruction}} = 1 + \#\text{instructions which} \times \#\text{Stalls from branch}$$
(Create stalls)

$$\text{Average } \underset{ET}{\text{Instruction}} = 1 + \left(\text{Branch Instr. Frequency} \times \#\text{ Stalls due to branch instr.}\right)$$

Control Dependency

* Control dependency is created in the pipeline due to execution of branch (Jac instruction). So program execution will be changed.
* In this the next sequential instructions become wanted or unwanted, it depends on the output previous instructions decoding.
* If the previous instructions decode the instruction as simple instructions (Arithmetic, data transfer etc) then the next instruction become wanted instructions.

* If the previous instruction decode the instruction as branch instruction, Unconditional Toc, Conditional Toc with True then next sequential instruction become unwanted

	1000	I_1		
I_1 is a branch instruction.	1001	I_2	Jmp 2000	When I_1 execute then I_2 is wanted instruction.
	1002	I_3		But when I_2 execute (decode) then
	1003	I_4		$I_3, I_4 \ldots$ is a wanted instruction
	1004	I_5		Because I_2 decode as unconditional
	1005	I_6		Toc (Jmp 2000) so BI_1 is the new
	1006	I_7		wanted instruction.

* The number of stalls created due to branch operation is called Branch Penalty
* This branch penalty value depends on the outcome of the target address.

Note: In the Risc pipeline branch penalty value is '1' because target address is available at the end of 2nd stage.

Branch penalty = $k-1 = 2-1 = 1$.

* Generally target address is available in decoding stage/phase but condition checking done in execution phase/stage.
* If target address available at 'k^th' stage

Branch penalty = $k-1$

Branch penalty = (At which stage Target address available) -1

→ Avg CPI = 1, But due to Control Hazards

CPI = 1 + (Branch instr freq × # stalls due to branch)

Date 11/10/23
Page

Q1. Suppose cpu have 100 instructions. Out of 100 instructions 30 are branch type, cubics creates 3 stalls cycles then avg CPI?

Ans 100 Instructions

ET = 100 x 1 = 100

Total # Stalls = no. of branch instructions x # stalls due to branch

⇒ 30 x 3

⇒ 90

For total 100 instr = 100 + 90

= 190

Avg CPI = 190/100 = 1.9 (Ans)

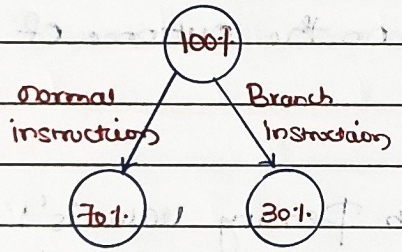
→ Average instructions ET = 1 + (branch instruction frequency x # Stalls due to branch instructions)

= 1 + (.30 x 3)

= 1 + 0.9

⇒ 1.9 (Ans)

→



(overlapped manner)

Stalls / Instructions = 0.7 x 0 + 0.30 x 3

= 0.9

Avg Inst. TIME = [1 + # Stalls / Instruction]

⇒ 1.9 (Ans)

- Stalling: Design a pipeline in such a manner, if it detect the branch instruction, then cpu have to stall (stop fetching the next instruction into the pipeline) until branch instruction evaluation (execution) not completed

Solution of Control Dependency:

- Hardware Solution → Branch prediction
- Software Solution → Delayed Branch

Branch Prediction : In this we predict that the branch is taken or branch is not taken and if prediction is correct then accordingly instruction will execute.

(i) Branch is always taken : In this we assume everytime branch is taken, so in this next instruction is target address instructions for execution.

(ii) Branch is never taken : In this we assume every time branch is not taken, so in this next instruction is the next sequential instruction for execution.

Note : If prediction goes wrong then we have to suffer from stalls.

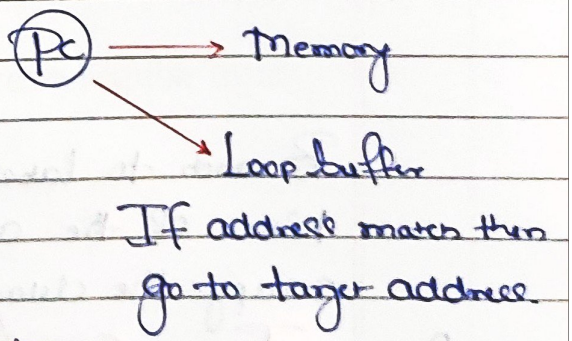
Dynamic Predictions

1. Bit implementation
2. Branch target buffer / Branch loop buffer / Branch prediction buffer

Branch loop buffer : It is a high speed buffer maintained in the instruction fetch stage of pipeline.

It's just like a database.

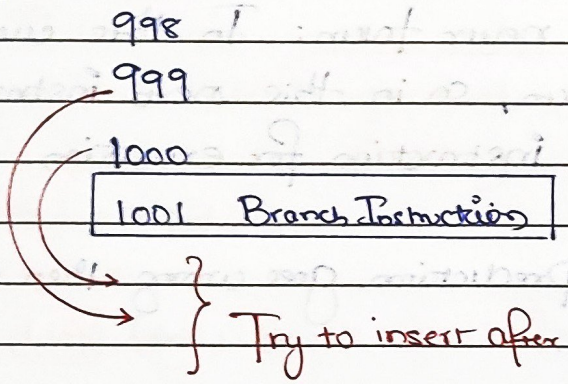
Branch instruction address	Target address
1001	2000



When the next time this type of instruction come then if match exists the address directly getting the target address.

Delayed Branch: It is a compile technique (Software Solution) in which trying to rearrange the instructions if possible.
If it is not possible to rearrange, then insert 'Nop' instructions after the branch instructions.

Nop Instruction: This is the instructions that fetch and execute but not update any register or not affect our desired output.



→ 'K' Stage Target Address
Delay slot: K-1 delay slot.

Q1. Delayed branching can help in the handling of control hazards. The following code is to run on a pipelined processor with one branch delay slot:

I_1 : ADD $R_2 \leftarrow R_7 + R_8$
I_2 : SUB $R_4 \leftarrow R_5 - R_6$
I_3 : ADD $R_1 \leftarrow R_2 + R_3$
I_4 : STORE MEMORY $[R_4] \leftarrow R_1$
Branch to label if $R_1 == 0$

Which of the above instructions I_1, I_2, I_3 or I_4 can legitimately occupy the delay slot without any other program modification?

Ans I_4 — Can insert in delay slot because no changes in program.

Instruction Pipelining (Continued)

M T W T F S S
Page No.:
Date: 11/10/23
YOUVA

Control Dependency

*Generally CPI = 1.

$$\text{Avg Instruction Et.} = CPI \times \text{Cycle Time}$$

But due to Control Hazard CPI ≠ 1.

$$CPI = (1 + \#\text{Stalls/Instruction})$$

$$\#\text{Stalls per instruction} = \text{Branch instruction frequency} \times \text{Branch penalty}$$

$$\text{Average Instruction Et} = (1 + \#\text{Stalls/instruction}) \times \text{cycle time}$$

Speed up factor with Stalls:

$$S = \frac{\text{Avg Instruction Et non pipe}}{\text{Avg Instruction Et pipe.}}$$

Ideal CPI of pipeline is always '1' due to a dependency Problem Some Stalls are Created in the pipeline.

$$S = \frac{CPI_{\text{non pipe}} \times \text{Cycle-time non pipe.}}{(1 + \text{no. of stalls/Instruction})(\text{Cycle time}_{\text{pipe}})}$$

When the pipeline Stages are perfectly balanced then 1 task execution time in the non-pipeline is also equal to number of Stages in the pipeline i.e... $t_n = k.t_p$ under this Condition:

$$\Rightarrow S = \frac{k.t_p}{(1 + \text{no. of Stalls/instruction})(\text{Cycle time pipe})}$$

$$\Rightarrow S = \frac{k}{(1 + \text{number of stalls/Instruction})}$$

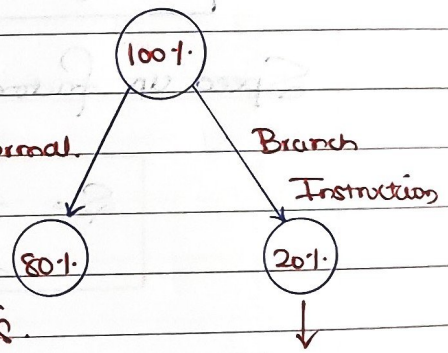
Q1. A CPU has five stages pipeline and runs at 1GHz frequency. Instruction fetch happens in the first stage of pipeline. A conditional branch instruction computes the target address and evaluates the condition in the third stage of the pipeline. The processor stops fetching new instructions following a conditional branch until the branch outcome is known. A program executes 10^9 instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, then total execution time of the program

Ans Frequency = 1GHz → Cycle time = $\frac{1}{1GHz}$ sec = 1nsec

5 Stage pipeline

Total = 10^9 instructions.

Target address = 3rd Stage.



Branch penalty = 3-1 ⇒ 2.

(Overlapped manner)

stall/instruction = 0.8×0 + 0.2×2

= 0.4

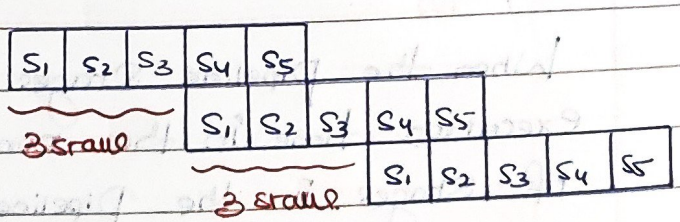
Avg Instr. = (1+0.4)×1

ET = 1.4 nsec

Program ET = $1.4 \times 10^{-9} \times 10^9$

= 1.4 sec (Ans)

Another Approach:



Avg CPI = .80×1 + 0.2×3

= 0.8 + 0.6

= 1.4 cycle

Avg Instruction ET = 1.4 × 1nsec

= 1.4 nsec.

Program ET = $1.4 \times 10^{-9} \times 10^9$

= 1.4 sec (Ans)

Q2. Consider a 6-stage pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution if 25% of the instructions incur 2 pipeline stalls cycle is _____.

Ans. $k = 6$ stage

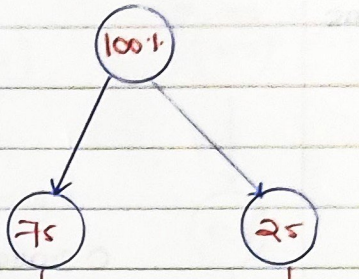
$t_c = 6 t_c$

$S = \frac{ET_{non\,pipe}}{ET_{pipe}}$ $\quad t_n = k * t_p$

$= \frac{6 * t_c}{(1 + \#stall/Inst^{n}) \times cycle\,time}$

$= \frac{6 * t_c}{(1+0.5) * t_c}$

$= \frac{6}{1.5} \Rightarrow \underline{4}$ (Ans)



#stall/instruction = $0.75 \times 0 + 0.25 \times 2$

$= \underline{0.5}$

Q3. Consider a 5-stage instruction pipeline, where all stages are perfectly balanced and have a cycle time of 20ns. Target address of branch instruction is available at end of the execution (last stage). There are 30% instructions.

(i) What is average instruction execution time (ignore the fact some are conditional)?

Ans. #stage = 5

Cycle time = 20nsec.

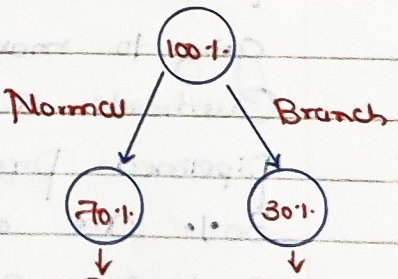
Target address available at end of 5th stage.

Branch penalty = $5 - 1 = 4$.

Avg Instruction ET_{pipe} = $(1 + 1.2) \times 20$

$= 2.2 \times 20$

$= \underline{44\,nsec}$ (Ans)



#stall/instr = $0.7 \times 0 + 0.30 \times 4$

$= \underline{1.2}$

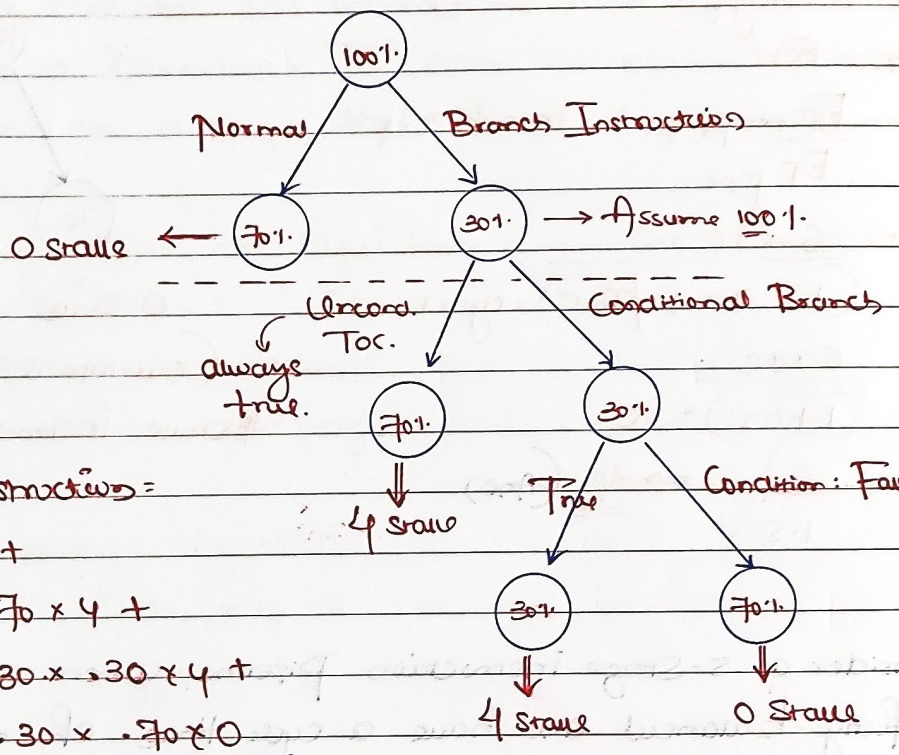
(ii) What is Speed up factor?

Ans. $S = \frac{ET.NP}{ET.P} \rightarrow \frac{5 \times 20\,nsec}{(1 + \#stall/instr) \times cycle}$

$= \frac{5 \times 20}{(1+1.2)\times 20} \rightarrow \frac{5}{2.2}$ $\therefore$ S = 2.27 (Ans)

(iii) Among the branch instruction 30% are conditional and 70% of them does not satisfy the condition then what is avg instruction ET?

Ans



#Stall/instruction =

$0.70 \times 0 +$

$.30 \times .70 \times 4 +$

$.30 \times .30 \times .30 \times 4 +$

$.30 \times .30 \times .70 \times 0$

$\Rightarrow$ 0.980 (Ans)

Q4. Consider a non pipelined Processing/operating at 2.5 GHz. It takes 5 clock cycles to complete an instruction. You are going to make a 5-Stage Pipeline out of this Processor. Overheads associated with pipelining force you to operate the pipelined Processor at 2GHz. In a given program, assume that 30% are memory instructions, 60% are ALU instructions and rest are branch instructions. 5% of the memory instructions cause Stalls of 50 clock cycles each due to cache misses and 50% of the branch instructions cause Stalls of 2 cycles each. Assume that there are no stalls associated with the execution of ALU instructions. For this program, the Speedup achieved by the pipelined processor

Over the non-pipelined processor (round off to two decimal places) is _____

Cycle time = $\frac{1}{2\text{GHz}}$ = 0.5 nsec.

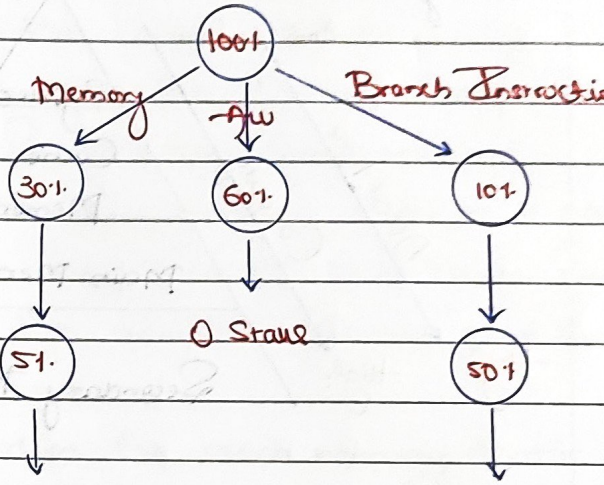
stalls/instructions =

.30 × 0.05 × 50

+ .60 × 0

+ .10 × .50 × 2

⟹ 0.85



Avg Inst. ET = (1 + 0.85) × 0.5

= 1.85 × 0.5

= 0.925 nsec

Nonpipeline: 5 clock cycle

2.5 GHz

Cycle time = $\frac{1}{2.5}$ sec = 0.4 nsec

ET nonpipe = 5 × 0.4 nsec

= 2 nsec

$S = \frac{ET_{NP}}{ET_P} = \frac{2}{0.925}$ ∴ S = 2.16 (Ans)