Que :- Consider A Non-Pipeline Processor with a clk cycle time of 2.3 ns which consumes 12 cycles for Load / Store inst^n, 8 cycles for ALU and 5 cycles for Branch inst^n. Relative frequencies of these inst^n are 40%, 30%, and 30% respectively.

Non Pipeline System is enhanced with a Pipeline to make the Average CPI of 1. In this implimentation Skew time overhead is 0.8 ns what is the Performance gain in the Pipeline design.

E.T. Non Pipeline $= (0.4 * 12 + 0.3 * 8 + 0.3 * 5) * 2.3$

$= 20.01$ ns

E.T. Pipeline $= [(0.4 * 1) + (0.3 * 1) + (0.3 * 1)] * (1 * 2.3) = 2.3$ ns + overhead

$= 2.3 + 0.8$

$= 3.1$

$S = \frac{ET_{Non\ Pipeline}}{ET_{Pipeline}} = \frac{20.01}{3.1} = 6.45$

Que :- Consider 4 stage Pipeline where different Inst^n are speding different cycles at different stages.

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

a.) How many cycles are required

b.) Performance gain

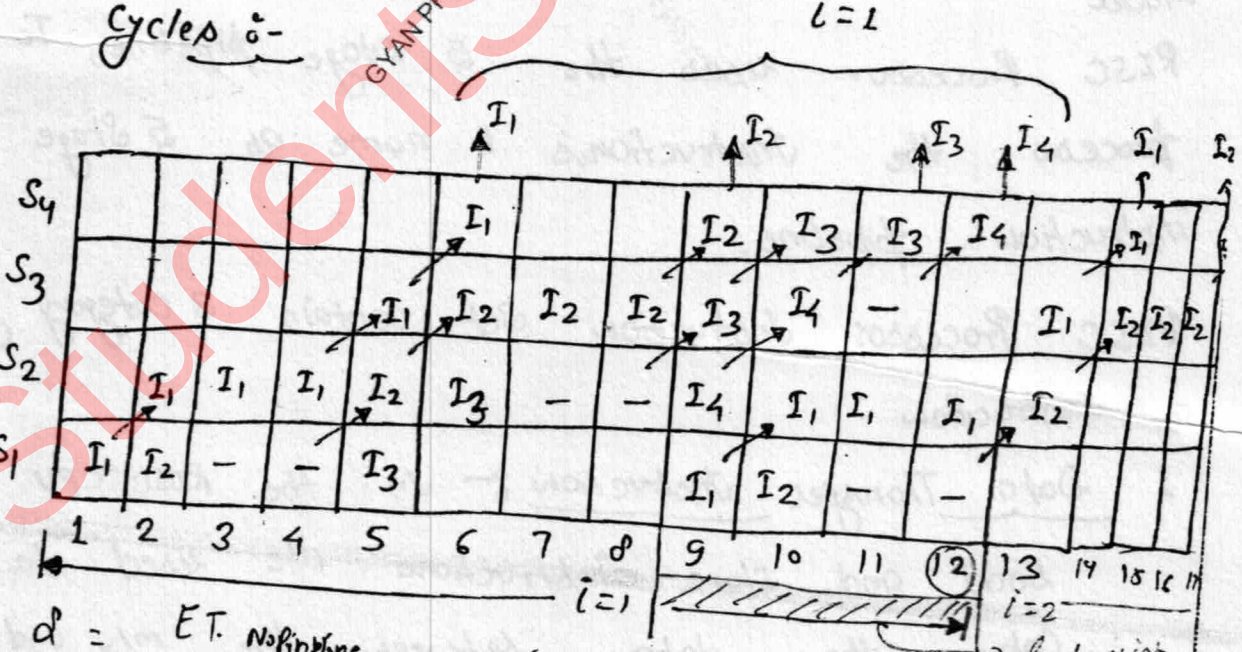
c.) following code is executed in the Pipeline

```
for ( i=1; i<=100 ; i++)
{ I1;
  I2;
  I3;
} I4;
```

o/p of I_2 for $(i=2)$ will be available after 17 cycle.

d.) Program E.T. without Loop level parallelism in the Pipeline.

a.) Cycles :-



b.) $d = \frac{\text{ET. Nonpipeline}}{\text{E.T Pipeline}} = \frac{(S_1+S_2+S_3+S_4)}{} = \frac{21}{12} = 1.75$

(c) 17.

(d) Program E.T. without loop level parallelism in the Pipeline:

Program E.T. = 100 * iterations * 12 cycles per iteration

= 1200 cycles

* If the 'option' in the answer is not matching then switch from overlapped loop parallelism to Non-overlapped loop parallelism.

RISC Pipeline :-

To analyze the implimentation issues of a pipeline let us consider RISC pipeline as a reference model.

RISC Processor used the 5 stage pipeline, To process the instructions & name as 5 stage instruction pipeline.

RISC Processor Instruction Set contain 3 category of a instruction:

1 Data Transfer Instruction :- In the RISC CPU Load and store instructions are used to copy the data between the m/y and register respectively.

** In these instructions, memory is ~~pred~~ referred with a indexed addressing mode.

Syntax:

Load	r_0	$3(r_1)$

↓

$r_0 \leftarrow M[3+[r_1]]$

Store	$3(r_1)$	r_0

↓

$M[3+[r_1]] \leftarrow r_0$

2.) Data manipulation Instructions :-

In the RISC CPU ALU operations are performed only on a Register data.

Syntax :

Add	r_0	r_1	r_2

↓

$r_0 \leftarrow r_1 + r_2$

3.) Transfer of control Instn :-

a) unconditional Toc :-

Syntax :

Jmp	2000

↓

PC : seg Addr
2000

*b) conditional Toc :- Condition evaluated in current Inst^{n} itself

Syntax :

DJNZ	r_0	2000

↓

Dec r_0

↓

NZ Compare "r_0" status → True → PC : seg Addr. 2000

→ false → PC : seg Addr

★ To Execute the above set of Instructions, 5 Stage Pipeline is used in the RISC CPU. Different Stages in the Pipeline is as follows:

Stage 1 (Instruction fetch) :- In this stage CPU reads the Instruction from the memory based on the PC. simultaneously PC will be increamented to next sequential address.

Stage 2 (Instruction Decode) :-

In this stage two operations are performed:

① Decode the Instruction (ID)

② Access the Register file to fetch the operand. (OF)

This stage also contain comparator circuit to evaluate the Branch condition.

Stage 3 (Execute) :-

In this stage ALU operations are performed.

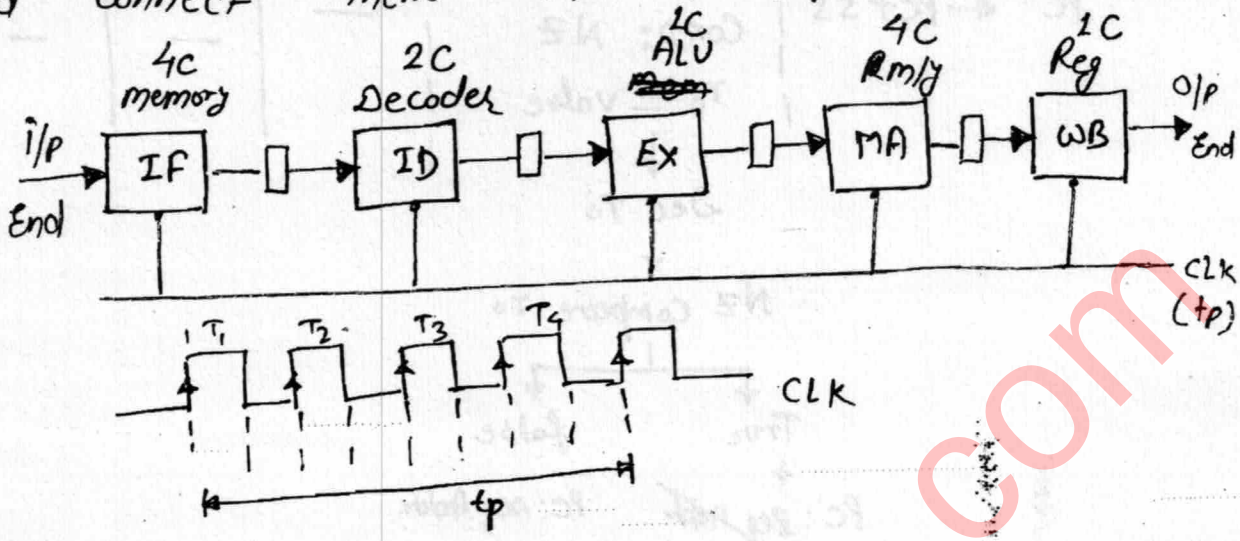
Stage 4 (Memory Access) :-

In this stage, m/y read and m/y write operations are performed to access the Data.

Stage 5 (write Back) :-

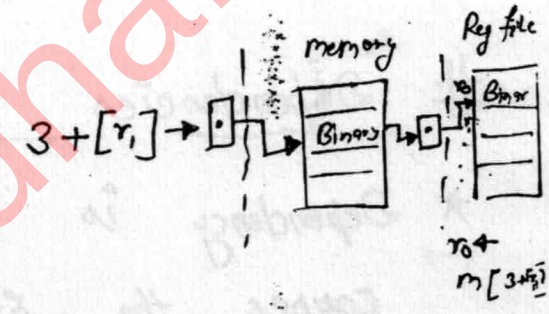
In this stage Register write operation is performed to store the result in the Register file.

Above sub-operations are assign to independent H/w and connect them in a pipeline sequence i.e.



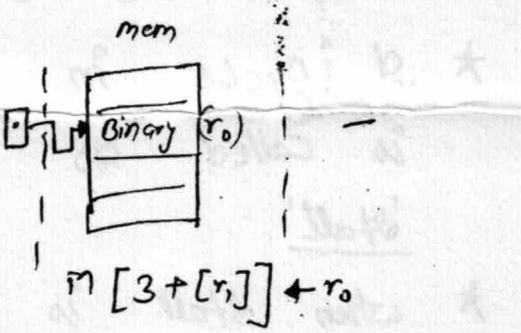
① Load $r_0, 3(r_1)$ | PC ← PC+SS

memRD | S: mem (3, r_1) | D: reg (r_0) | r_1 = value



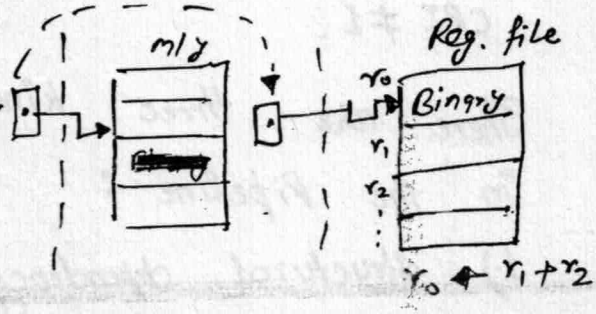
② Store $3(r_1), r_0$ | PC ← PC+SS

memWR | S: reg (r_0) | D: mem (3, r_1) | r_0 = value | r_1 = value



③ Add r_0, r_1, r_2 | PC ← PC+SS

S: reg (r_1, r_2) | D: reg (r_0) | r_1: value | r_2: value



④ Jmp 2000 | PC ← PC+SS | uncond Toc | TA = 2000 ⇓ PC: ~~200~~ 2000

(5) DJNZ r_0, 2000

PC ← PC + SS

Cond To C	—	—	—
TA: 2000			
Cond: NZ			
r_0 = value			

↓

Dec r_0

↓

NZ Compare r_0

↓

True → PC: ~~Reg Addr.~~ 2000

False → PC: Seq. Addr.

Dependencies in the Pipeline :-

★ Dependency is a major problem in the Pipeline, causes the Extra cycles.

★ A cycle in the Pipeline without new Input is called as Extra cycle. also name as 'Stall'

★ when Stall is present in the Pipeline then CPI ≠ 1.

There are three kinds of dependencies possible in the Pipeline :

1.) Structural dependency
2.) Data dependency
3.) Control dependency.

1) Structural dependency :-

★ Structural dependency will be occured in the Pipeline due to a resource conflict.

★ Resource may be a Register, or memory or function unit (ALU).

	CC1	CC2	CC3	CC4
I_1:	Mem	ID	ALU	Mem
I_2:		Mem	ID	ALU
I_3:			Mem	ID
I_4:				Mem

Resource Conflict

↳ cycle diagram 1

★ In the above execution sequence I_1 and I_4 both of the instructions are tries to access the same resource (memory) in the same cycle (CC4). This situation in the Pipeline is called as resource conflict.

★ Conflict is a unsuccessful operation so we need to keep the I_4 into a waiting until the resource become available.

★ This waiting creates 'stalls' in the Pipeline that is described below :

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	mem	ID	ALU	mem	WB		
I_2:		mem	ID	ALU	mem	WB	
I_3:			mem	ID	ALU	mem	WB
I_4:							mem

Stalls

7 cycles → 4 i/p = 3 stalls

★ To minimize the Structural Stalls in the Pipeline, Hardware Technique is used that is : **Re-Naming**

★ In the cycle diagram 1, I_1 refers the memory in 4th stage of a Pipeline. to access the 'Data' simultaneously I_4 refers the memory in a 1st stage of Pipeline to access the Instruction in the same cycle CC4.

★ when the Instruction and data both are present in the same memory chip then above situation creates conflict.

★ Renaming mechanism states that split the memory unit into a two independent modules used to store the Instruction and data seperately. called as code memory (CM) and Data mry (DM) respectively.

★ Refer the code-memory in 1st stage and refer the Data-memory in 4th stage of a pipeline so Accessing of these two modules in the same cycle does not create the conflict. that is described below :-

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	CM	ID	ALU	DM	WB		
I_2:		CM	ID	ALU	DM	WB	
I_3:			CM	ID	ALU	DM	WB
I_4:				CM	ID	ALU	DM
I_5:					CM	ID	ALU
I_6:						CM	ID
I_7:							CM

{ 7 cycles − 7 i/p = 0 Stalls }

Date
19/08/2017 (★★★) v. Imp.

Data Dependency :-

★ Consider Program segment where instruction 'J' follows instruction 'I' in a program order i.e.

Prog :-

I :	Instn
J :	Instn
⋮	⋮

★ Data dependency condition will be occured when the instn 'J' tries to read the data before instruction 'I' writes it i.e.

Read - before - write

Eg: I_1 : Add r_0, r_1, r_2 ; $\quad r_0 \leftarrow r_1 + r_2$

I_2 : Sub r_3, r_0, r_4 ; $\quad r_3 \leftarrow r_0 - r_4$

★ when the Above instructions are executed in a Non-pipeline System then Data dependency condition doesn't occured because I_2 executing after completion of I_1 So I_2 reads the Register r_0 'after' I_1 writes it (No data Loss).

★ when the Above Instructions are executed in a Pipelined Processor then Data dependency condition will be occured. becaused I_2 overlapping with I_1 so I_2 tries to read the the Register r_0 "before" I_1 writes it. So I_2

incorrectly read the old value from the r_0 ('Data Loss').

	CC1	CC2	CC3	CC4
I_1	IF	ID	Ex	
I_2		IF	ID S: r_0, r_4 r_0 = value r_4 = value	
I_3			IF	

→ Read before write
{Data Loss}

★ To detect the Data dependency condition, some logic will be maintained in the 'ID Stage' of a Pipeline.

Structure of the logic is

ID-Stage :- (Tomasulo soln.)

S/No.	Function unit	Destination	Independent Source 1	Independent Source 2	Dependent Source 1	Dependent Source 2	Status
I_1	Add	r_0	r_1	r_2	—	—	0
I_2	Sub	r_3	—	r_4	r_0 (I_1)	—	0

Of {Reg. file Accessing}
r_1 = value, r_2 = value } ⇒ Ex is allocated
r_4 = value } ⇒ Ex is not allocated

Status → 0 ; execution not yet completed
1 ; execution completed.

* By using the above logic we need to stop the accessing of a dependent data until the data become available in the Register. This waiting creates 'Stalls' in the pipeline that is described below :

**

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	IF	ID	EX	MA	WB r_0: updated [Status: 1]		
I_2:		IF	ID S: r_0, r_4 (2) (I) I_1 r_4 = value	ID	ID	ID r_0: value (Read)	EX
I_3:			IF	IF	IF	IF	ID
I_4:							IF

Stalls

7 Cycles – 4 i/p's → 3 stalls

* To minimize the Data depency stalls H/w technique is used that is :

→ Operand forwarding (Short circuiting) or (By-Passing)

This technique states that use the Buffer the between the Stages to hold the intermediate Result so that Dependent Instn will be accessing

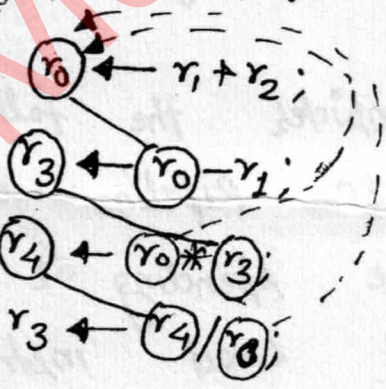
the New value from the Buffer before updating the Register-file.

★ ~~consider the following code~~

★ In the last stage of a pipeline, "Register-write" and "Register-Read" sequence of operations are present so the updated value is immidiately available for other instructions to access therefore Buffer is not required at the End of last stage.

★ Consider the following code used to execute in the Pipeline, uping operand forwarding technique

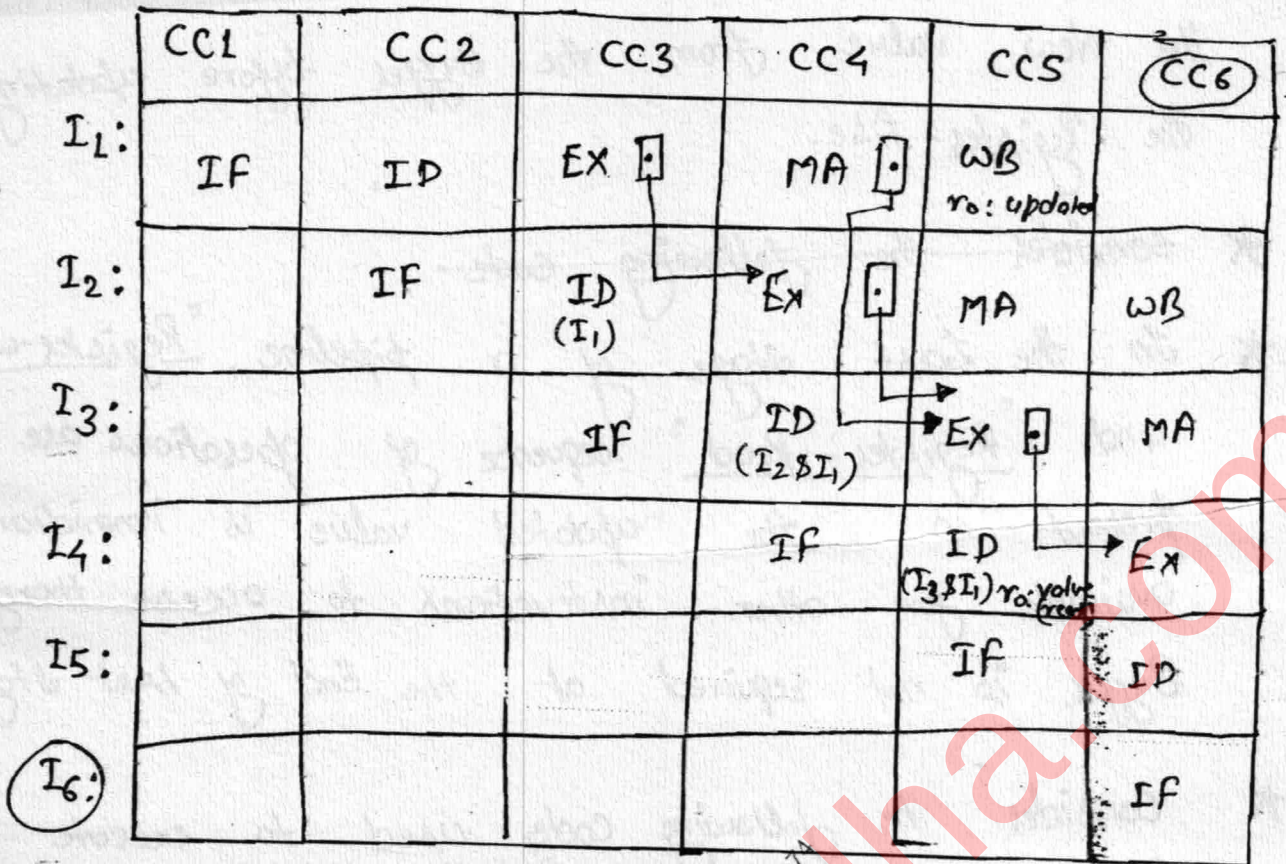
I_1: Add r_0, r_1, r_2;
I_2: Sub r_3, r_0, r_1;
I_3: mul r_4, r_0, r_3;
I_4: Div r_3, r_4, r_0;



★ when the Data dependency condition occured between the adjacent instructions then the corresponding dependency is name as "True data dependency":-

→ True data dependency :-

$$\begin{Bmatrix} I_2 - I_1\ [r_0] \\ I_3 - I_2\ [r_3] \\ I_4 - I_3\ [r_4] \end{Bmatrix}$$



{ 6 cycles – 6 i/p's – 0 stalls }

Que:- Consider the following code, executed on a RISC pipeline where all the instructions are spending 1 cycle on all the stages but Load instruction spending 4 cycles on MA stage (4th stage). How many cycles are required to complete the code.

I_1: Load $r_0, 3(r_1)$; $r_0 \leftarrow m[3+[r_1]]$

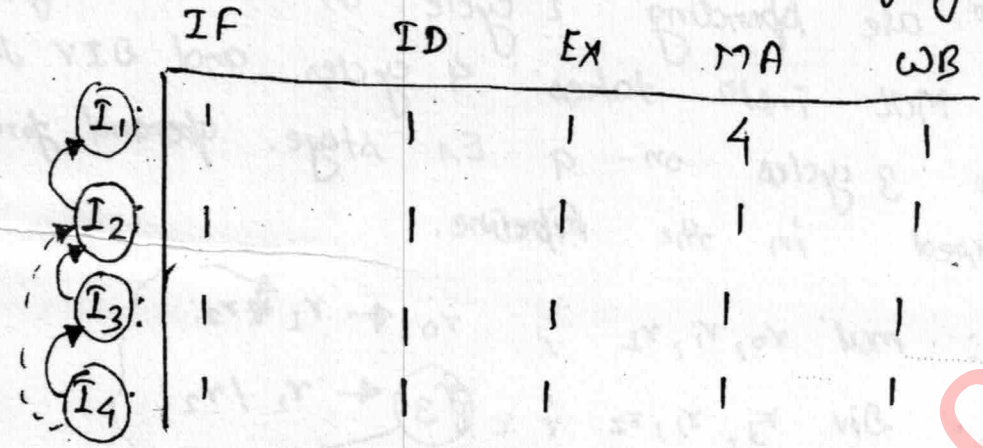
I_2: Add r_2, r_0, r_1 ; $r_2 \leftarrow r_0 + r_1$

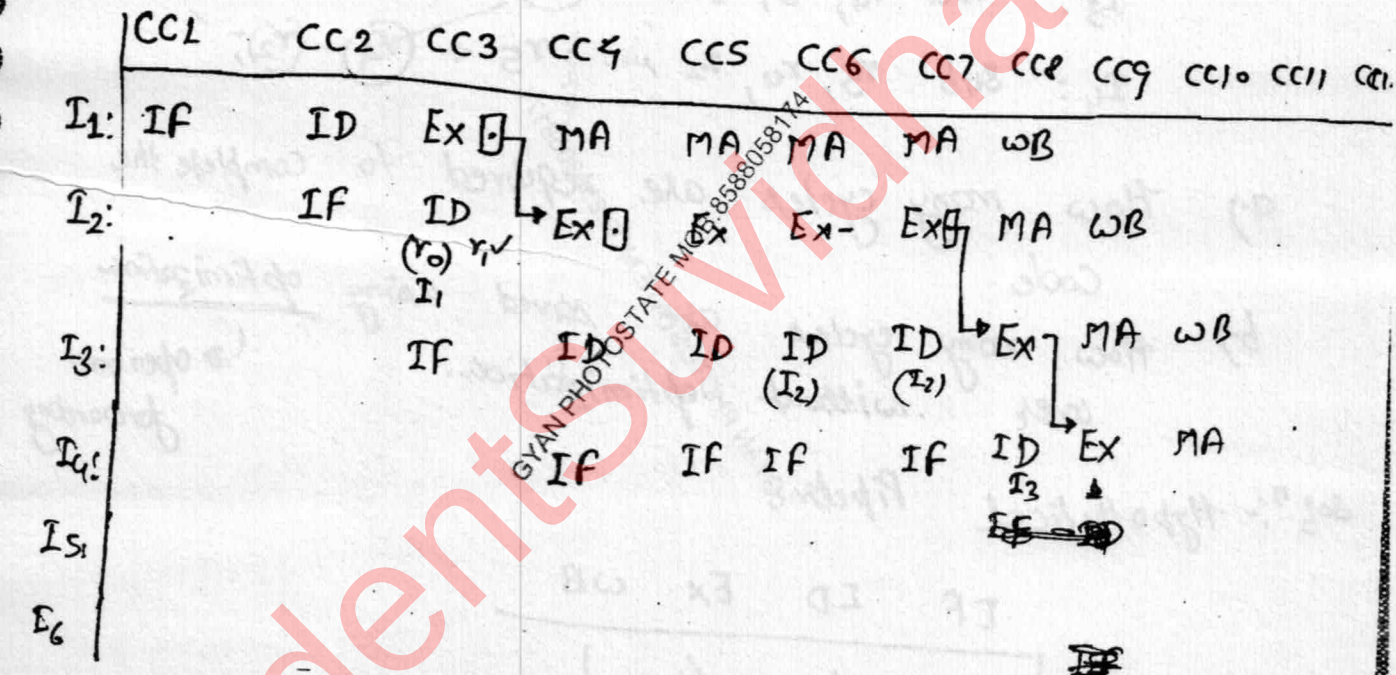
I_3: Sub r_3, r_2, r_1 ; $r_3 \leftarrow r_2 - r_1$

I_4: mul r_L, r_3, r_2 ; $r_L \leftarrow r_3 * r_2$

Soln :- RISC Pipeline { K=5 { IF, ID, Ex, MA, WB } }

{ operand forwarding By default }





	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12
I1:	IF	ID	Ex	MA	MA	MA	MA	WB				
I2:		IF	ID (I1)	ID	ID	ID	ID	Ex	MA	WB		
I3:			If	If	If	If	If	ID (I2)	Ex	MA	WB	
I4:								If	ID (I3 & I2)	Ex	MA	WB

Que:- Consider 4 stage Pipeline (IF, ID, Ex, WB) used to Execute the following code. all the Instn are spending 1 cycle on all the stages but MUL instn takes 4 cycles and DIV Instn takes 3 cycles on a Ex stage. operand forwarding is used in the Pipeline.

I_1: mul r_0, r_1, r_2 ; $r_0 \leftarrow r_1 * r_2$;

I_2: Div r_3, r_1, r_2 ; $r_3 \leftarrow r_1 / r_2$;

I_3: Add r_4, r_3, r_1 ; $r_4 \leftarrow r_3 + r_1$;

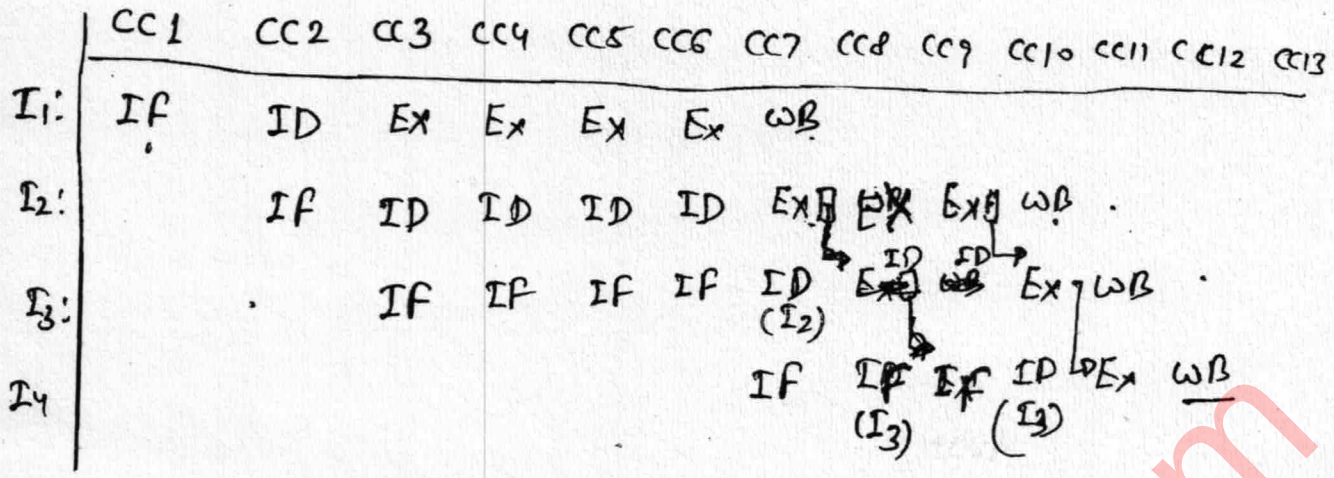
I_4: Sub r_5, r_4, r_2 ; $r_5 \leftarrow r_4 - r_2$;

a.) How many cycles are required to complete the code

b.) How many cycles are saved using optimization over without optimization. → operand forwarding

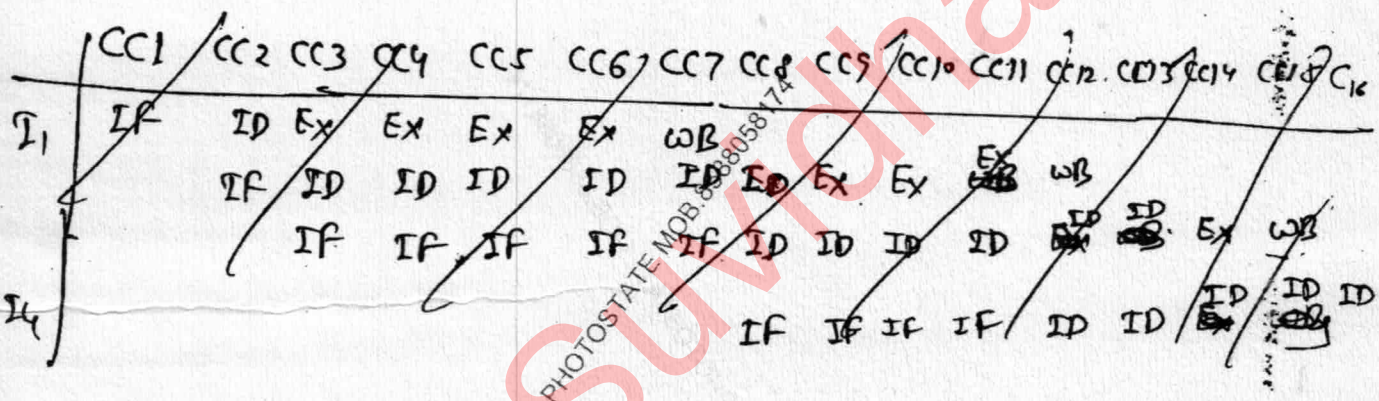
Soln:- Hypothetical Pipeline

	IF	ID	Ex	WB
I_1	1	1	4	1
I_2	1	1	3	1
I_3	1	1	1	1
I_4	1	1	1	1

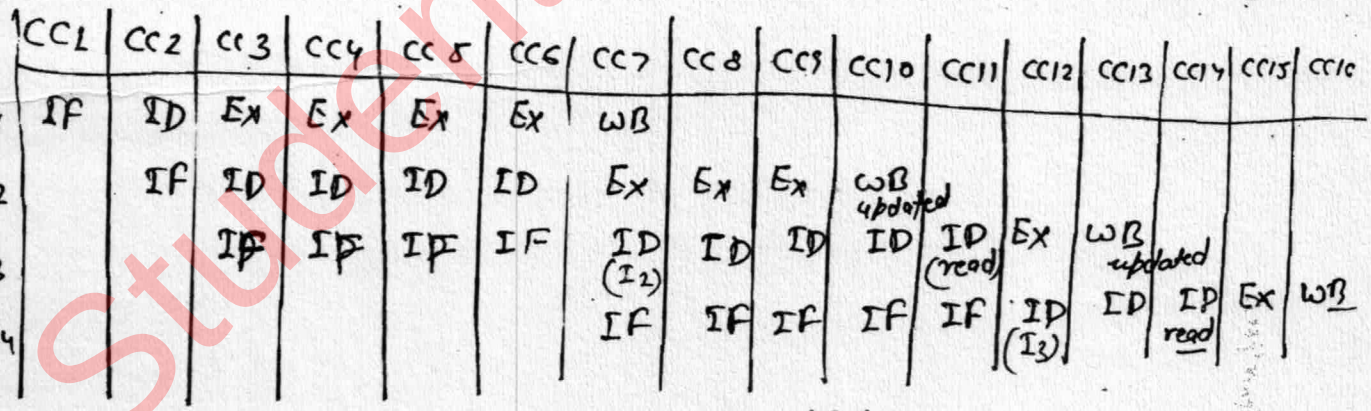


a) 12

b) w/o operand forwarding :-



CC17 CC18
Ex WB



cycle saved = 16 - 12 = 4