CS 232: Digital Logic Design and Computer Architecture Lab

Lab-04

NAME: B V S S Prabandh

Roll_no: 210050037

MARCH 8 2023

Contents

1 Q1 **2**

1.1 part_a 2

1.2 part_b 3

2 Q2 **3**

2.1 part_a 3

2.2 part_b 4

3 Q3 **4**

3.1 part_a 4

3.2 part_b 6

4 Q4 **8**

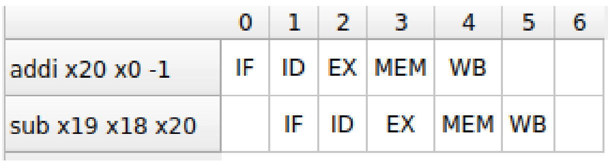
1 Q1

1.1 part_a

Read after Write Hazard(RAW):

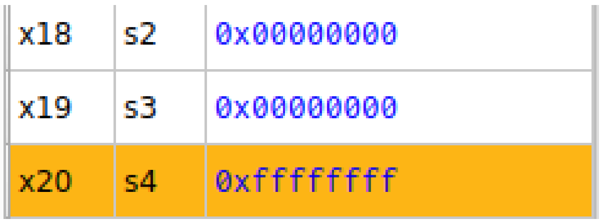
This type of Data Hazard occurs when a register is read before its being Written by an earlier Instruction .

- Ideally s4 should store the value -1 and as s2,s3 is intialized to 0 . s3 must store a value of 1
- In the pipeline we can see EX of instruction 2 happens before WB of instruction 1.



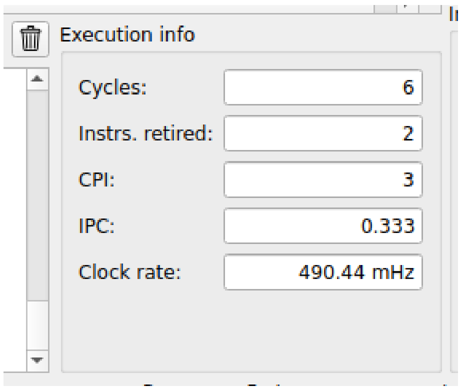
	0	1	2	3	4	5	6
addi x20 x0 -1	IF	ID	EX	MEM	WB		
sub x19 x18 x20		IF	ID	EX	MEM	WB	

- This leads to a data-Hazard, Instead of expected values (s4 = -1 , s2 = 0 , s3 = 1) we have (s4 = -1, s2 = 0 ,s3 = 0)



x18	s2	0x00000000
x19	s3	0x00000000
x20	s4	0xffffffff

- Execution Table :



Execution info

Cycles:	6
Instrs. retired:	2
CPI:	3
IPC:	0.333
Clock rate:	490.44 mHz

1.2 part_b

Write after Read Hazard(WAR):

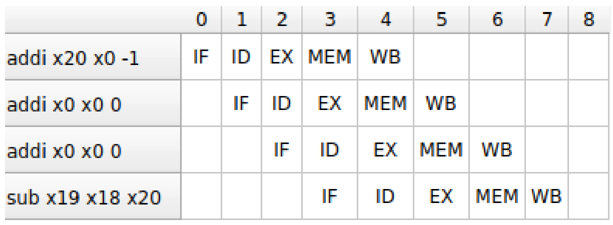
This type of Data Hazard occurs when a register is written before its being read by an earlier Instruction .

- This Type of Data-Hazard is not possible in 5-stage Vanilla Pipeline
- This is because WB stage is after ID stage (Registers are read before written)
- Therefore, There can be no chance of occurence of WAR Hazard

2 Q2

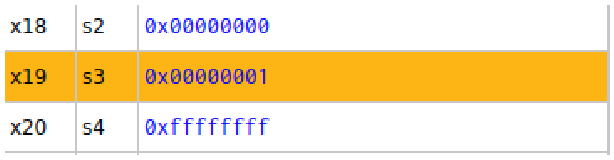
2.1 part_a

- To remove the Data-Hazard present in q1_a , We need to stall/NOPs the pipe until WB stage of instruction 1 is finished.
- We need to add 2 nop instructions in between instruction1 and instruction 2
- In the pipeline we can see EX of instruction 2 happens just after WB of instruction 1.



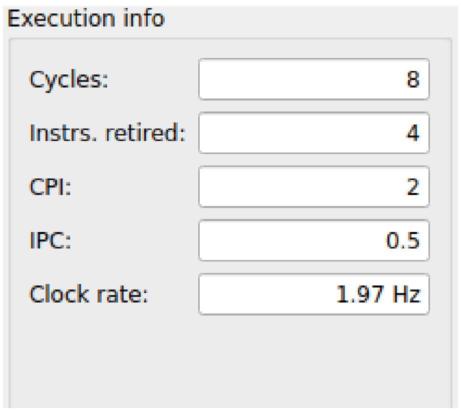
	0	1	2	3	4	5	6	7	8
addi x20 x0 -1	IF	ID	EX	MEM	WB				
addi x0 x0 0		IF	ID	EX	MEM	WB			
addi x0 x0 0			IF	ID	EX	MEM	WB		
sub x19 x18 x20				IF	ID	EX	MEM	WB	

- This mitigates data-Hazard, expected values (s4 = -1 , s2 = 0 , s3 = 1)



x18	s2	0x00000000
x19	s3	0x00000001
x20	s4	0xffffffff

- Execution Table :



Execution info

Cycles:	8
Instrs. retired:	4
CPI:	2
IPC:	0.5
Clock rate:	1.97 Hz

- Although we removed the hazard but it was on the expense of increase in cycles :(, Increase from 6 to 8 cycles.

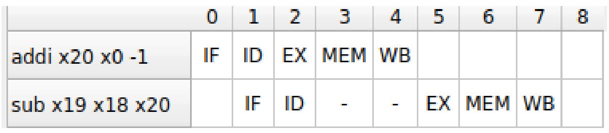
2.2 part_b

- No hazard possible $\implies$ no need for fix :)

3 Q3

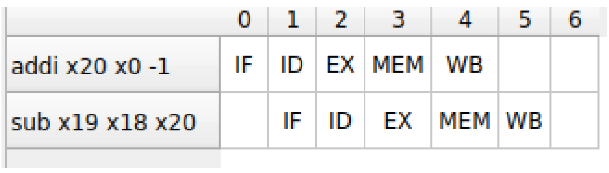
3.1 part_a

- 5-stage Processor uses forwarding and stall for (lw/sw)
- 5-stage Processor w/o forwarding uses only stalls for Data-Hazards
- Any Hazard arising from RAW can be resolved using forwarding in 5-stage Processor but stalls are needed for 5-stage Processor w/o forwarding
- Pipeline:



	0	1	2	3	4	5	6	7	8
addi x20 x0 -1	IF	ID	EX	MEM	WB				
sub x19 x18 x20		IF	ID	-	-	EX	MEM	WB	

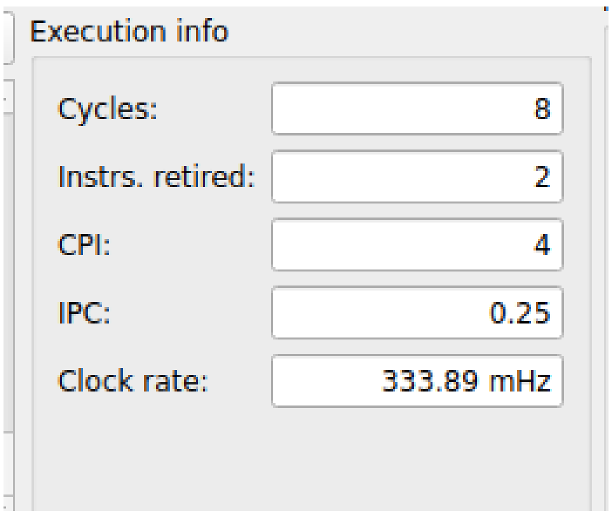
Figure 1: W/O FORWARDING



	0	1	2	3	4	5	6
addi x20 x0 -1	IF	ID	EX	MEM	WB		
sub x19 x18 x20		IF	ID	EX	MEM	WB	

Figure 2: W/ FORWARDING AND HAZARD CONTROL

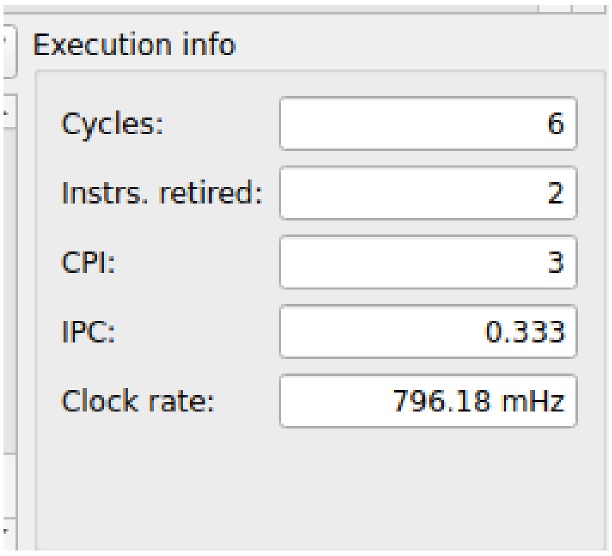
- EXECUTION TABLE:



Execution info

Cycles:	8
Instrs. retired:	2
CPI:	4
IPC:	0.25
Clock rate:	333.89 mHz

Figure 3: W/O FORWARDING



Execution info

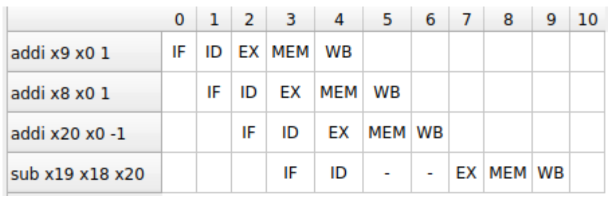
Cycles:	6
Instrs. retired:	2
CPI:	3
IPC:	0.333
Clock rate:	796.18 mHz

Figure 4: W/ FORWARDING AND HAZARD CONTROL

- Number of cycles w/o forwarding is 8 , while with forwarding is 6

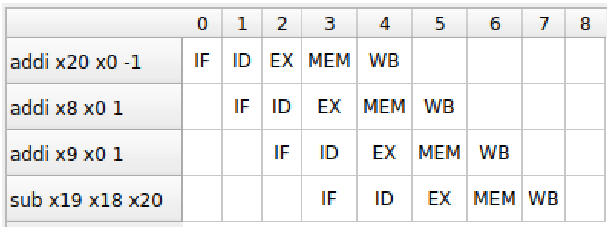
3.2 part_b

- Same set of instructions, when executed in a different order, can take different number of cycles on 5 stage processor without forwarding.
- This can happen if interchanging of order removes (or) adds Data-Hazards in the pipeline
- The set with Lower Hazards needs less cycles to complete the instructions set whilst the later take more cycles to complete.
- Pipeline:



	0	1	2	3	4	5	6	7	8	9	10
addi x9 x0 1	IF	ID	EX	MEM	WB						
addi x8 x0 1		IF	ID	EX	MEM	WB					
addi x20 x0 -1			IF	ID	EX	MEM	WB				
sub x19 x18 x20				IF	ID	-	-	EX	MEM	WB	

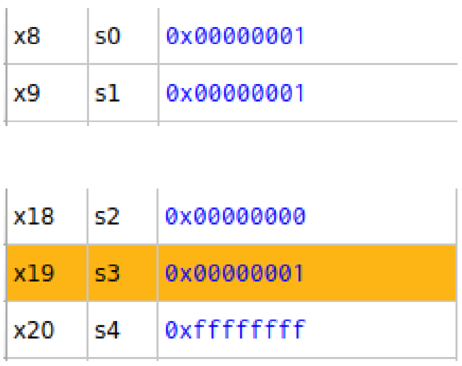
Figure 5: Pipeline with Hazards



	0	1	2	3	4	5	6	7	8
addi x20 x0 -1	IF	ID	EX	MEM	WB				
addi x8 x0 1		IF	ID	EX	MEM	WB			
addi x9 x0 1			IF	ID	EX	MEM	WB		
sub x19 x18 x20				IF	ID	EX	MEM	WB	

Figure 6: Pipeline with no Hazards

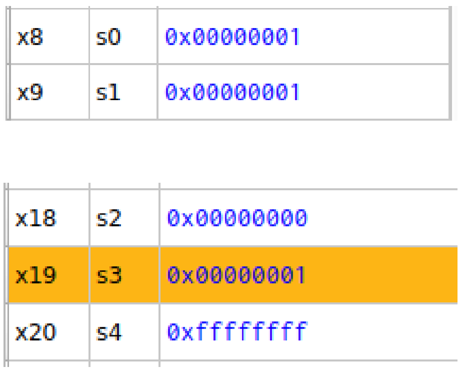
- Although number of cycles taken are different, values stored in registers are same i.e, effectively both are same.



x8	s0	0x00000001
x9	s1	0x00000001

x18	s2	0x00000000
x19	s3	0x00000001
x20	s4	0xffffffff

Figure 7: Pipeline with Hazards

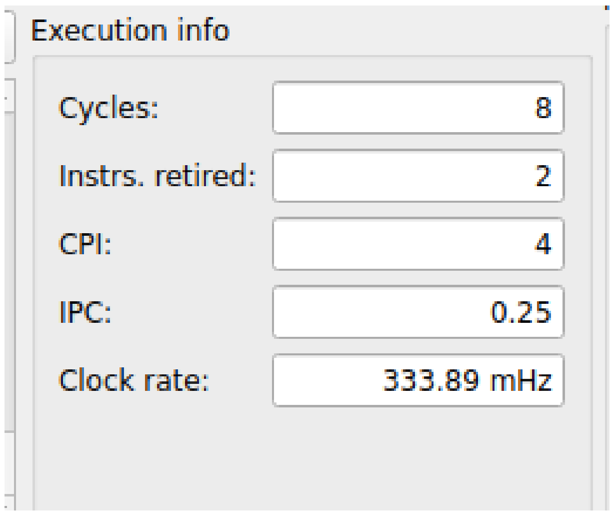


x8	s0	0x00000001
x9	s1	0x00000001

x18	s2	0x00000000
x19	s3	0x00000001
x20	s4	0xffffffff

Figure 8: Pipeline with no Hazards

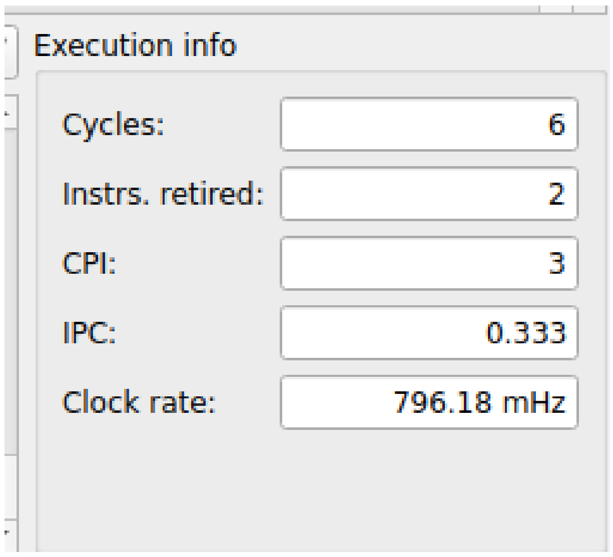
- EXECUTION TABLES:



Execution info

Cycles:	8
Instrs. retired:	2
CPI:	4
IPC:	0.25
Clock rate:	333.89 mHz

Figure 9: Pipeline with Hazards



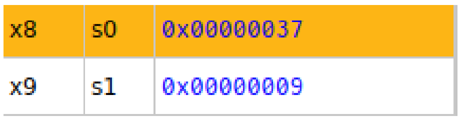
Execution info

Cycles:	6
Instrs. retired:	2
CPI:	3
IPC:	0.333
Clock rate:	796.18 mHz

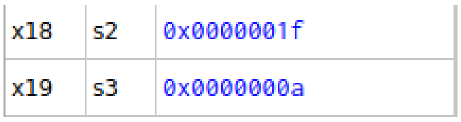
Figure 10: Pipeline with no Hazards

4 Q4

- Adding 10 integers with minimum number of clock cycles and minimal registers in use.
- We need use a register as soon as we know we dont need it anymore
- For avoiding hazards (RAW) leave atleast two-instructions in between writing and reading in a register
- U can maitain without stall till cycle 17, then you are forced to make stalls
- U would need 3 stalls to remove the hazards and 4 registers to compute the sum
- Registers: Here s0 stores the sum which is $(37)_{16} = (55)_{10}$ (sum of first 10 natural numbers)

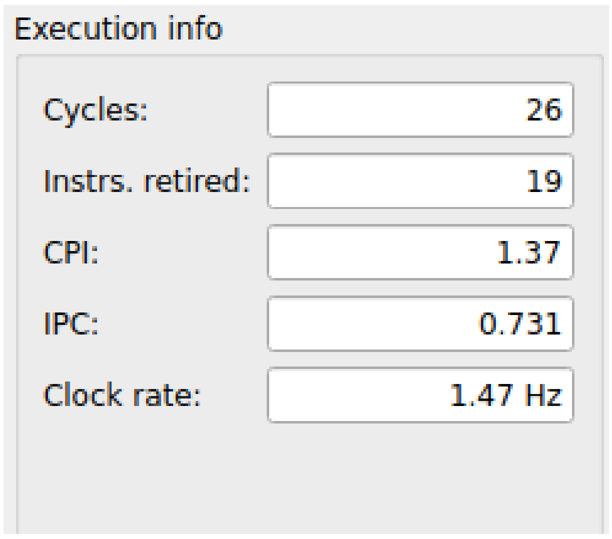


x8	s0	0x00000037
x9	s1	0x00000009



x18	s2	0x0000001f
x19	s3	0x0000000a

- I have done this in 26 clock cycles (which is the minimum I could achieve , idk If its possible to do better than this)



Execution info

Cycles:	26
Instrs. retired:	19
CPI:	1.37
IPC:	0.731
Clock rate:	1.47 Hz

Figure 11: Pipeline with no Hazards