

CS341: Computer Architecture Lab

Lab 3: Pipelining Report

Richeek Das (190260036)



Department of Computer Science and Engineering
Indian Institute of Technology Bombay
2021-2022

Problems

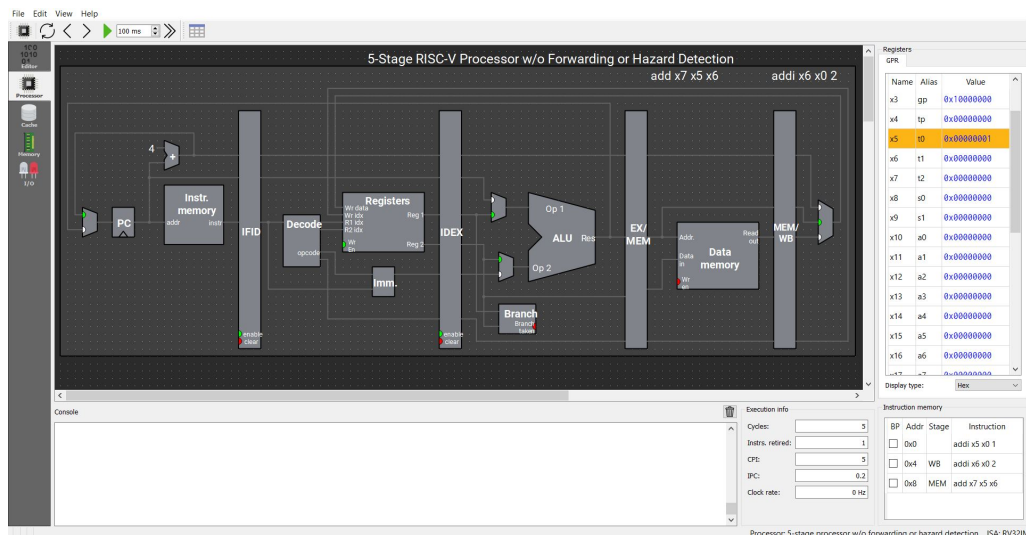
1. 5 Stage, without forwarding or hazard detection

Here we show an example of **data dependence hazard. read-after-write (RAW): True Dependence**

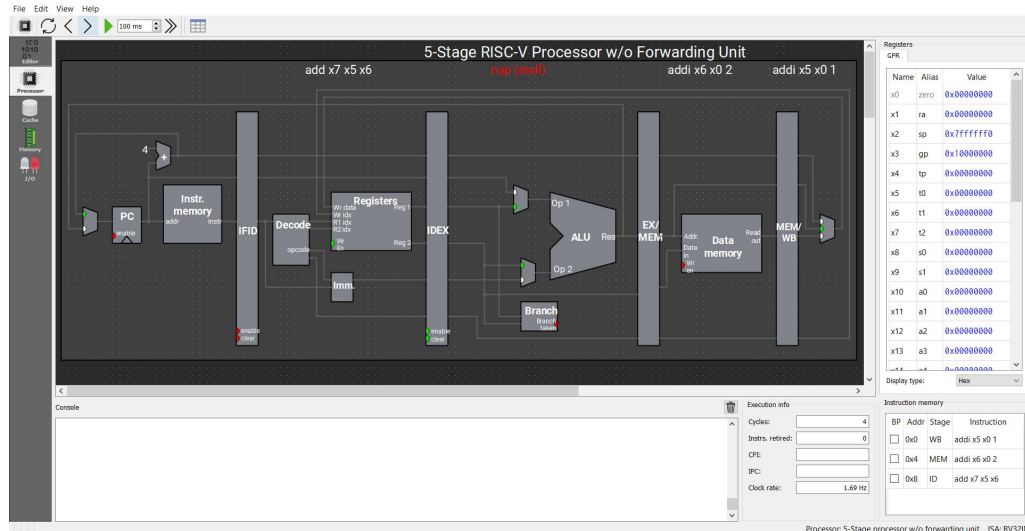
```
.text

main:
    li t0 1
    li t1 2
    add t2 t0 t1
```

Here without hazard detection or forwarding, this code will return a wrong result. The **ADD t2 t0 t1** instruction will use the old value of **t1** and **t2** i.e. 0. This is due to the lack of hazard detection which fails to ensure that **t0 = 1** and **t1 = 2** when **ADD** is in **EX** stage. We show how a pipeline with hazard detection solves this using **NOP** instructions:



Q1: Without Forwarding or Hazard Detection. Wrong output **t2=0x00000000**. ADD crosses EX stage but Load has not finished yet.



Q1: Without Forwarding but With Hazard Detection. Correct output **t2=0x00000003** [Notice the **NOP instr**]. ADD instruction waits for Load Imm to finish.

2. 5 Stage, without forwarding, with hazard detection

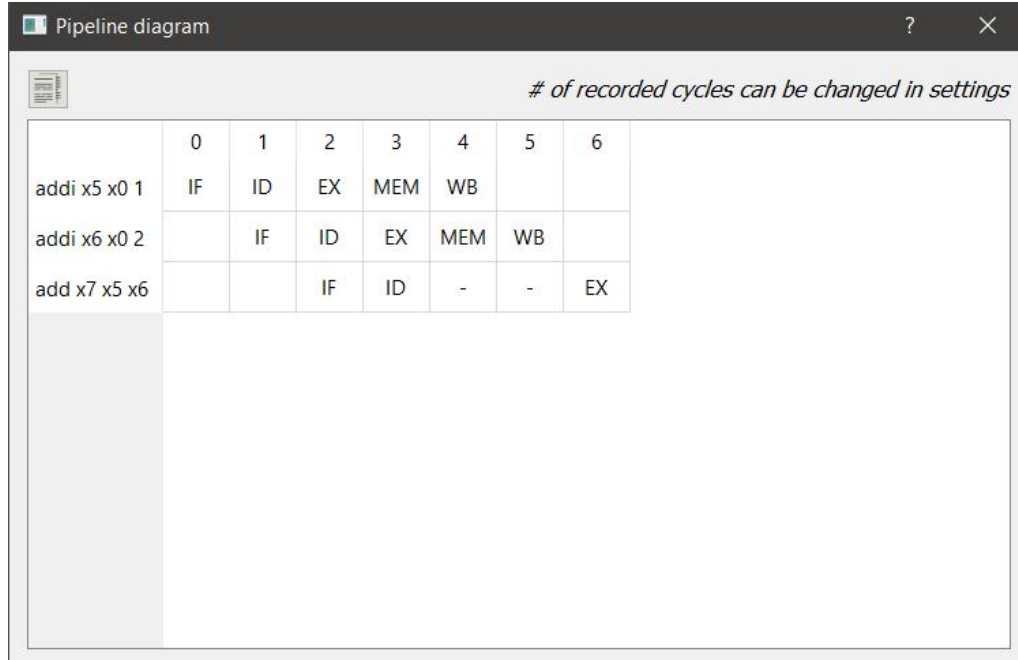
a: Here we show an example of **data dependence hazard**. **read-after-write (RAW): True Dependence**

```
.text
```

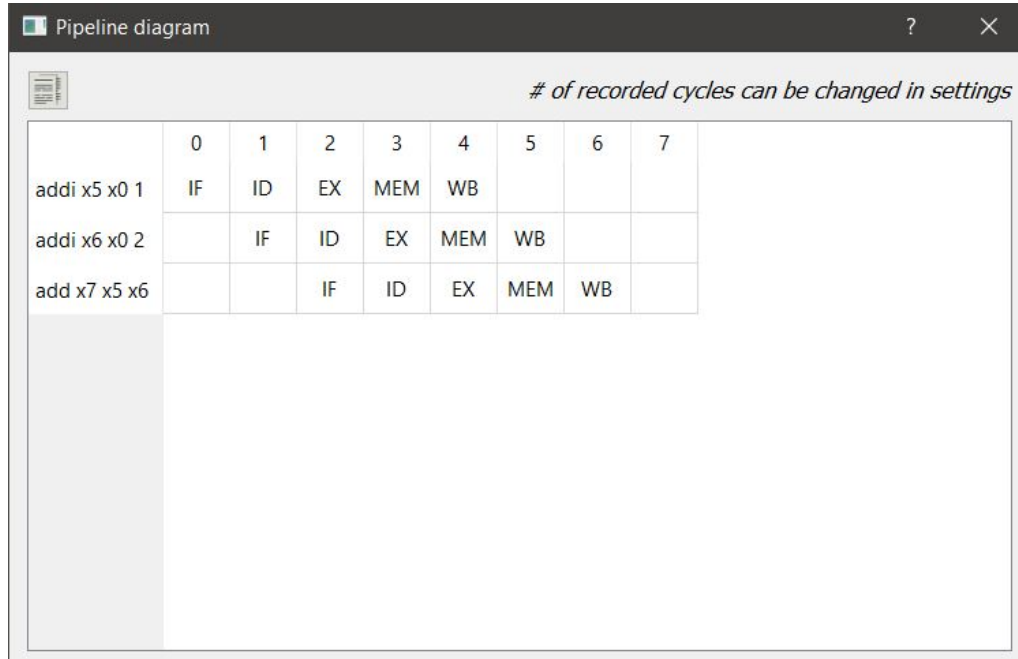
```
main:
    li t0 1
    li t1 2
    add t2 t0 t1
```

In this code, if forwarding is disabled the **ADD** instruction will need to wait in the **IDEX** latch till the **li t1 2** instruction reaches the **WB** stage. Only after its written in the register file **ADD** can proceed. This is an example of **True Dependence** or **RAW** hazard.

But if Forwarding is enabled, the registers with the required values 1 and 2 will be ready after they are done with the **EX** stage. They won't need to complete the **WB** stage. So as soon as **li t1 2** reaches **MEM** stage, **ADD** proceeds to **ALU** i.e. there is no stall.



Q2a: Pipeline without forwarding handles this hazard with stalls by introducing NOP instructions.



Q2a: Pipeline with forwarding handles this hazard by bypassing the output of load back to the ALU as it is required by the next ADD instruction. Hence there is no stall.

b: Example of sequence of instructions in which stalls can be avoided by rearranging the instructions (rearranging should not change the program logic i.e., final values in the registers):

Rearranging the code in the second column does not lead to any change in logic but provides the pipeline with “useful & independent” instructions to keep it filled before `t0` and `t1` get ready to be used in the `ADD` instruction in the `EX` stage.

Code leading to stalls (forwarding disabled)	Rearranged code working without stalls (forwarding disabled)
<pre>li t0 1 li t1 2 add t2 t0 t1 li t3 3 li t4 4</pre>	<pre>li t0 1 li t1 2 li t3 3 li t4 4 add t2 t0 t1</pre>

Execution of the code on Ripes: Hazard type: **RAW & True Dependence**

Pipeline diagram

of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10	11
addi x5 x0 1	IF	ID	EX	MEM	WB							
addi x6 x0 2		IF	ID	EX	MEM	WB						
add x7 x5 x6			IF	ID	-	-	EX	MEM	WB			
addi x28 x0 3				IF	-	-	ID	EX	MEM	WB		
addi x29 x0 4							IF	ID	EX	MEM	WB	

Q2b: Pipeline without forwarding handles this hazard with stalls by introducing NOP instructions.

Pipeline diagram

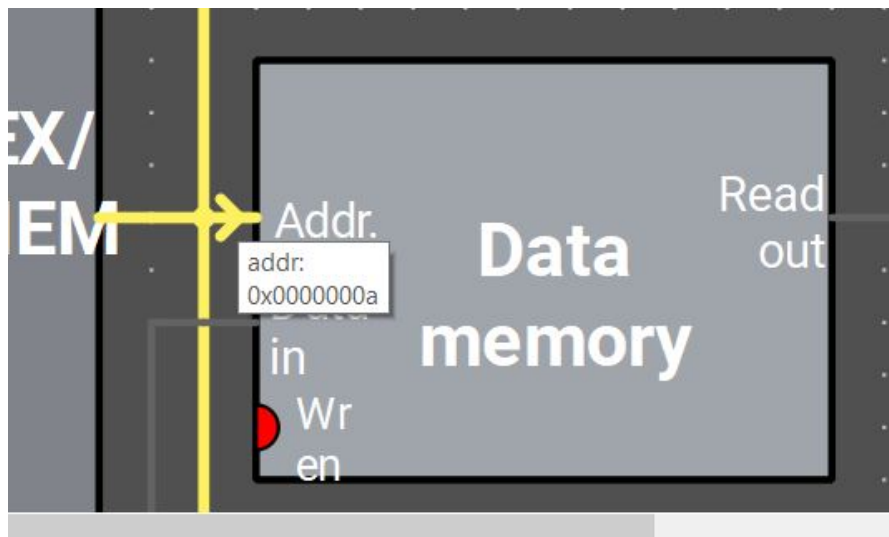
of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9
addi x5 x0 1	IF	ID	EX	MEM	WB					
addi x6 x0 2		IF	ID	EX	MEM	WB				
addi x28 x0 3			IF	ID	EX	MEM	WB			
addi x29 x0 4				IF	ID	EX	MEM	WB		
add x7 x5 x6					IF	ID	EX	MEM	WB	

Q2b: The rearranged code ensures `t0` and `t1` are written back into the register files when `ADD` reaches `EX` stage. Hence there are no stalls.

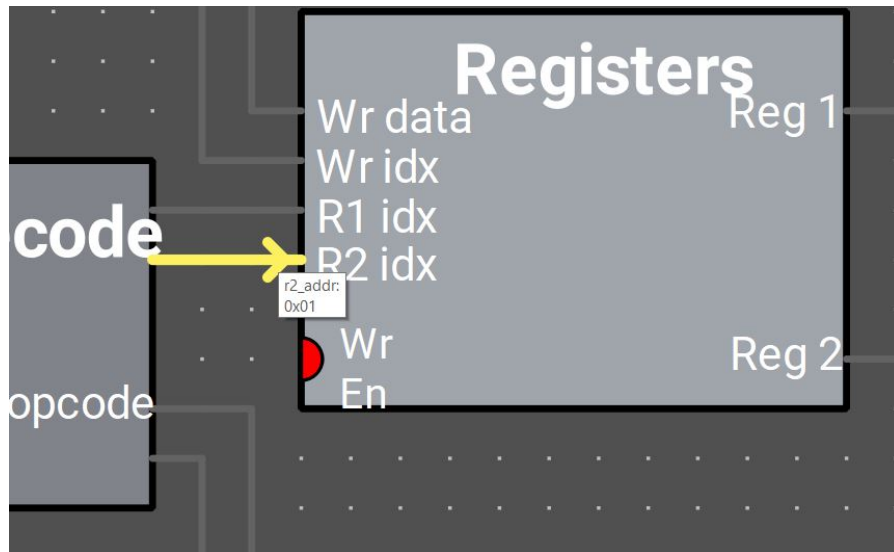
3. Stalls and Forwarding

- a. Address input of data memory after 4th cycle: `0x0000000a`



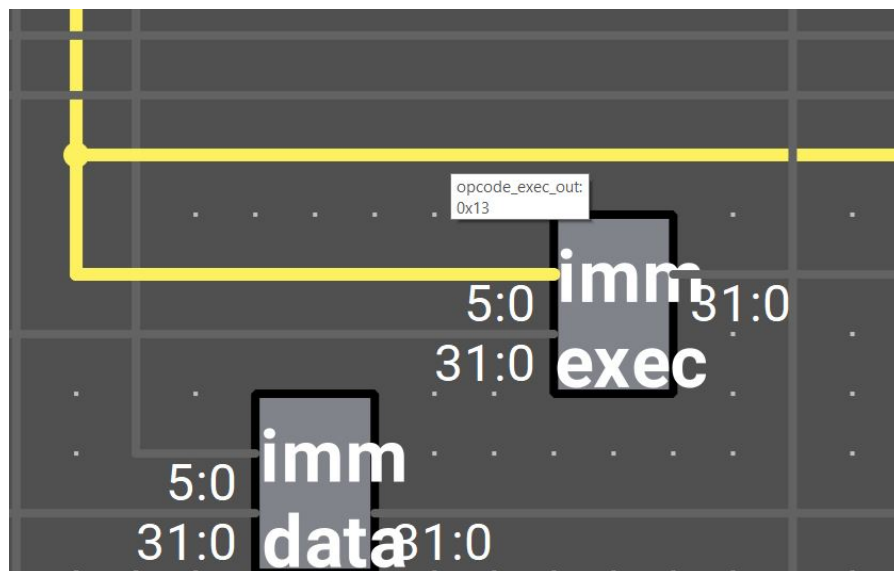
Q3a: addr

- b. R2 idx input of registers block after 10 cycles: `0x01`



Q3b: r2_addr

c. opcode_exec_out after 4 cycles: 0x13



Q3c: opcode_exec_out

d. a: 14 cycles, b: 19 cycles, c: 13 cycles


Pipeline diagram

of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
addi x5 x0 2	IF	ID	EX	MEM	WB										
addi x6 x0 10		IF	ID	EX	MEM	WB									
add x5 x5 x6			IF	ID	EX	MEM	WB								
addi x7 x0 5				IF	ID	EX	MEM	WB							
add x7 x5 x7					IF	ID	EX	MEM	WB						
addi x17 x0 1						IF	ID	EX	MEM	WB					
add x10 x7 x0							IF	ID	EX	MEM	WB				
ecall								IF	ID	EX	-	-	MEM	WB	

Q3d: 5-stage RISC-V with hazard detection and forwarding. **14 cycles**

Pipeline diagram



of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
addi x5 x0 2	IF	ID	EX	MEM	WB															
addi x6 x0 10		IF	ID	EX	MEM	WB														
add x5 x5 x6			IF	ID	-	-	EX	MEM	WB											
addi x7 x0 5				IF	-	-	ID	EX	MEM	WB										
add x7 x5 x7							IF	ID	-	-	EX	MEM	WB							
addi x17 x0 1								IF	-	-	ID	EX	MEM	WB						
add x10 x7 x0											IF	ID	-	EX	MEM	WB				
ecall												IF	-	ID	EX	-	-	MEM	WB	

Q3d: 5-stage RISC-V with hazard detection but no forwarding. **19 cycles**

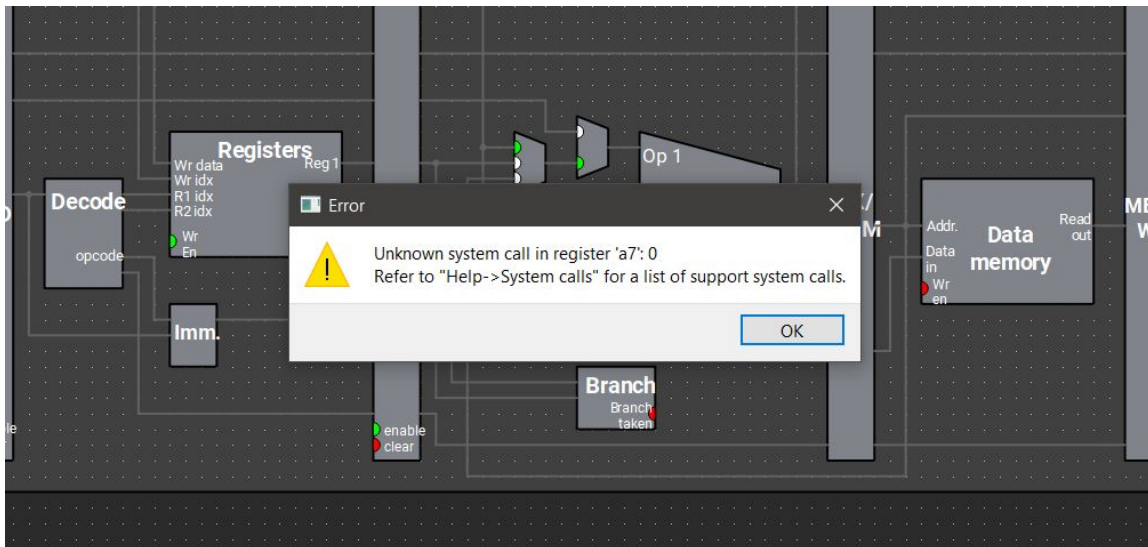
Pipeline diagram

of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
addi x5 x0 2	IF	ID	II	EX	MEM	WB								
addi x6 x0 10	IF	ID	II	EX	MEM	WB								
add x5 x5 x6			IF	ID	II	EX	MEM	WB						
addi x7 x0 5			IF	ID	-	II	EX	MEM	WB					
add x7 x5 x7				IF	-	ID	II	EX	MEM	WB				
addi x17 x0 1				IF	-	ID	II	EX	MEM	WB				
add x10 x7 x0					IF	ID	II	EX	MEM	WB				
ecall					IF	ID	-	II	EX	-	-	MEM	WB	

Q3d: 6-stage dual-issue processor. **13 cycles**

- e. **Output:** error: Unknown system call in register 'a7':0. Since the **printInt** syscall is not loaded into the **a7** register, we don't see any output in the console.



Q3e: Screenshot of Output.

Reason: a7 is in the MEM stage when the **ecall** accesses it in the ID stage. If we don't have hazard detection, we need to somehow ensure that **a7** is ready when **ecall** accesses it.

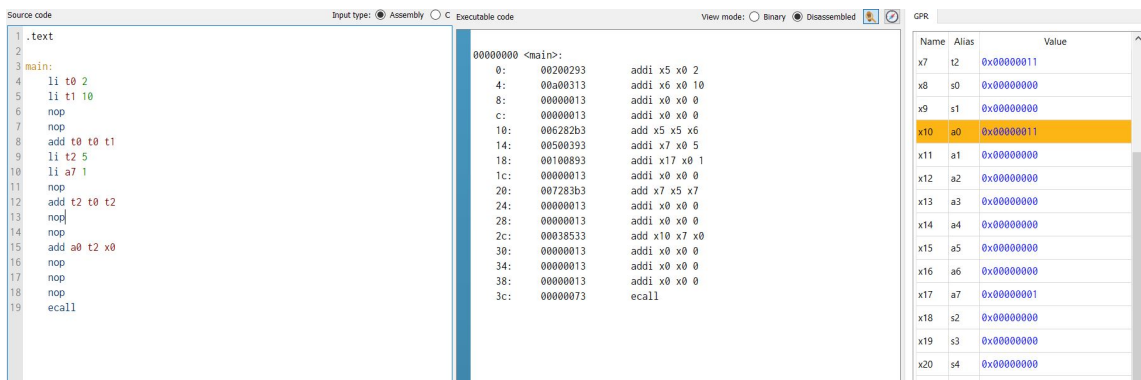
Solution: We can fix this by reordering some statements and introducing **NOP** instructions where required. We are using 5-stage RISC-V w/o Forwarding or Hazard Detection.

```

.text

main:
    li t0 2
    li t1 10
    nop                # introducing two NOPs here so that t0 & t1
    nop                # get loaded into WB before ADD
    add t0 t0 t1
    li t2 5
    li a7 1
    nop                # so that t2 enters WB when ADD enters EX
    add t2 t0 t2
    nop                # so that t2 enters WB when ADD a0 t2 x0 enters EX
    nop
    add a0 t2 x0
    nop                # so that a0 enters WB when ECALL enters IF
    nop
    nop
    ecall

```



Q3e: Screenshot of a working solution.

4. Maximising Efficiency

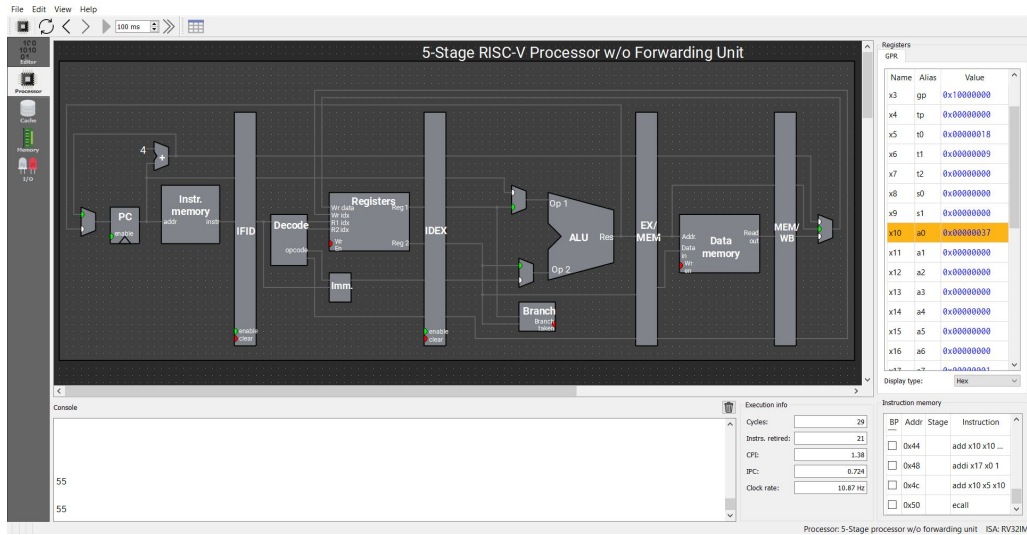
Number of cycles taken = 29

Number of registers taken = 4 [t0,t1,a0,a7]

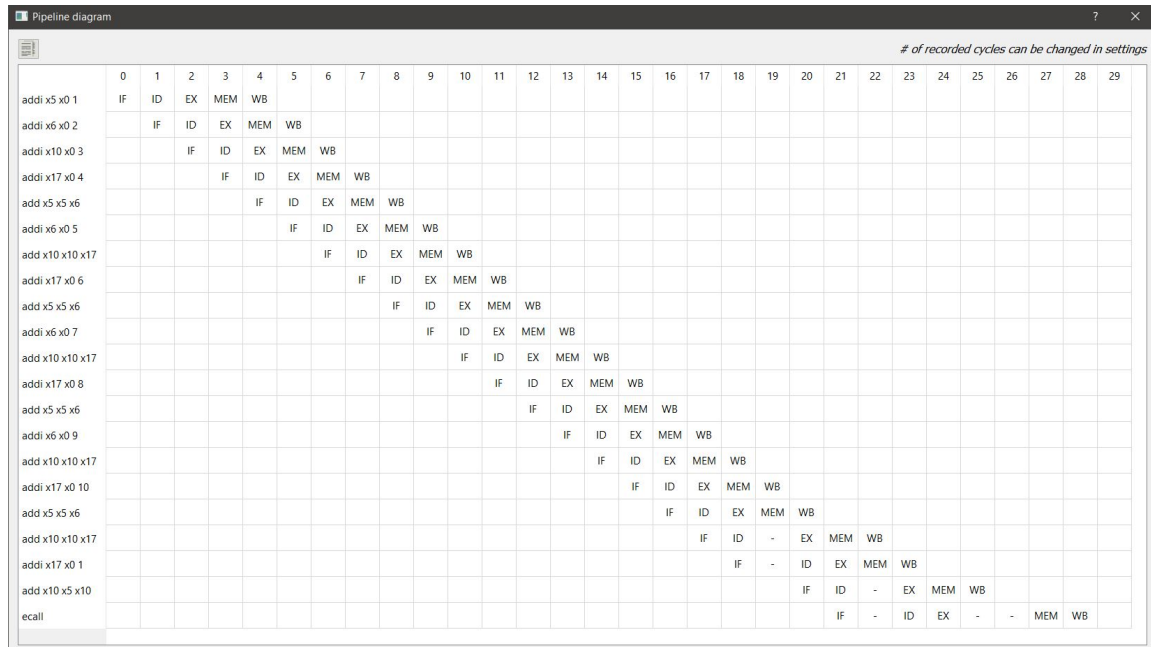
Numbers hardcoded = [1,2,3,4,5,6,7,8,9,10]

Output = 55

Code is available in the file: 190260036_A3.s



Q4: Screenshot of the number of cycles consumed. **29 cycles**



Q4: Pipeline Diagram **29 cycles**