CS 341 Assignment 3

Devansh Jain, 190100044

September 14, 2021

Contents

1 5 Stage, without forwarding or hazard detection

Code

```
.text

main:
    addi s0 zero 5
    add s1 s0 zero
```

Behaviour

Expectation: At the end of the program, we expect registers `s0` and `s1` to have value 5.
Reality: At the end of the program, only register `s0` has value 5. (`s1` is default to 0)

Explanation

The incorrect value is due to data hazard.
To be more specific, we face a **read-before-write (RAW) data hazard**.
`addi s0 zero 5` writes to register `s0` in WB stage during 4th cycle.
`add s1 s0 zero` reads from register `s0` in ID stage during 2nd cycle.
The value read here is not updated yet causing the incorrect computation.

Screenshots

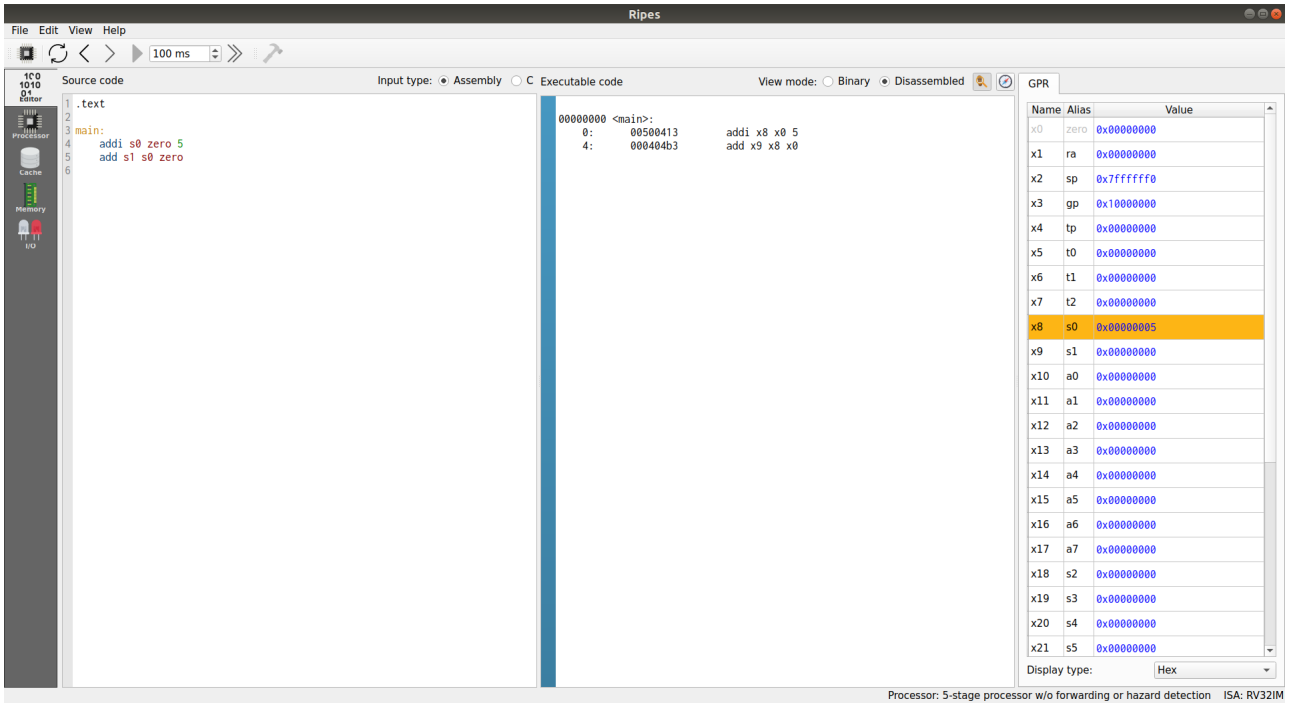

Figure 1: Without hazard detection; Without forwarding;

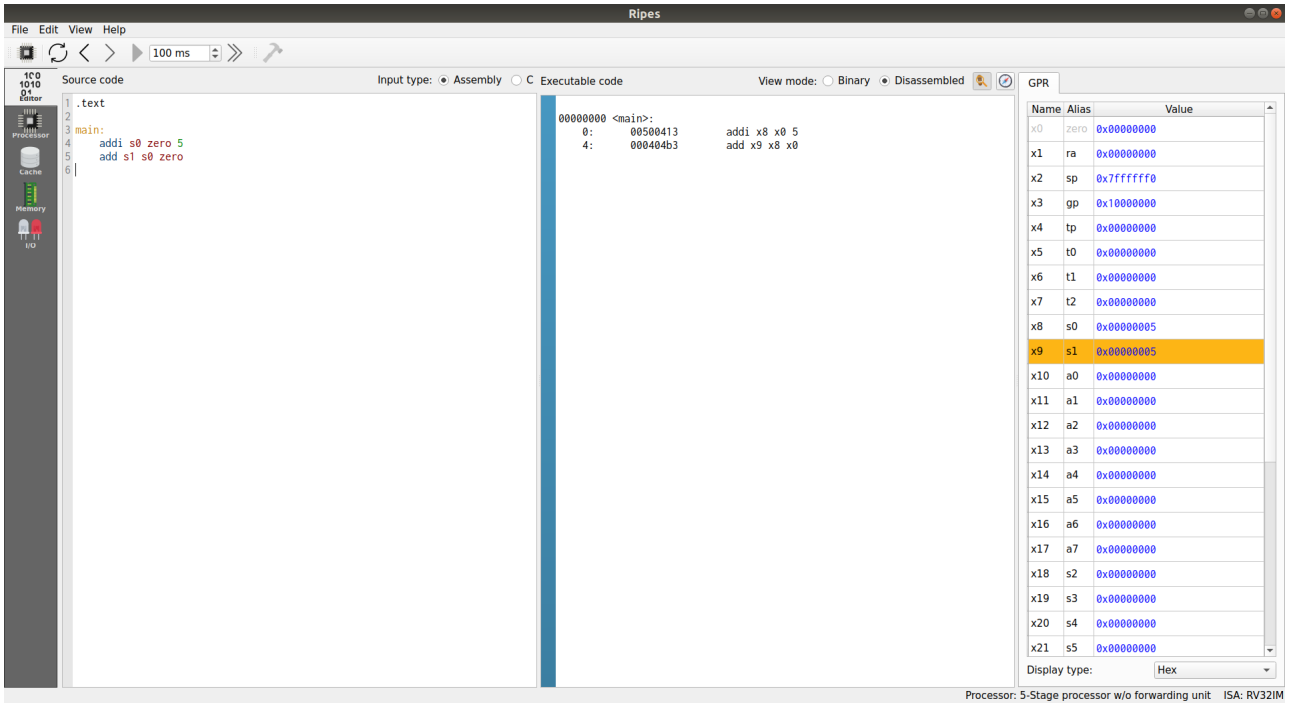

Figure 2: With hazard detection; Without forwarding;

2 5 Stage, without forwarding, with hazard detection

a.

Code

```
.text

main:
    addi s0 zero 5
    add s1 s0 zero
```

Behaviour

At the end of the program, we expect registers `s0` and `s1` to have value 5.
Without forwarding, we observe that ID for second instruction takes 3 cycles (2 stalls).
There are no stalls with forwarding.

Screenshots

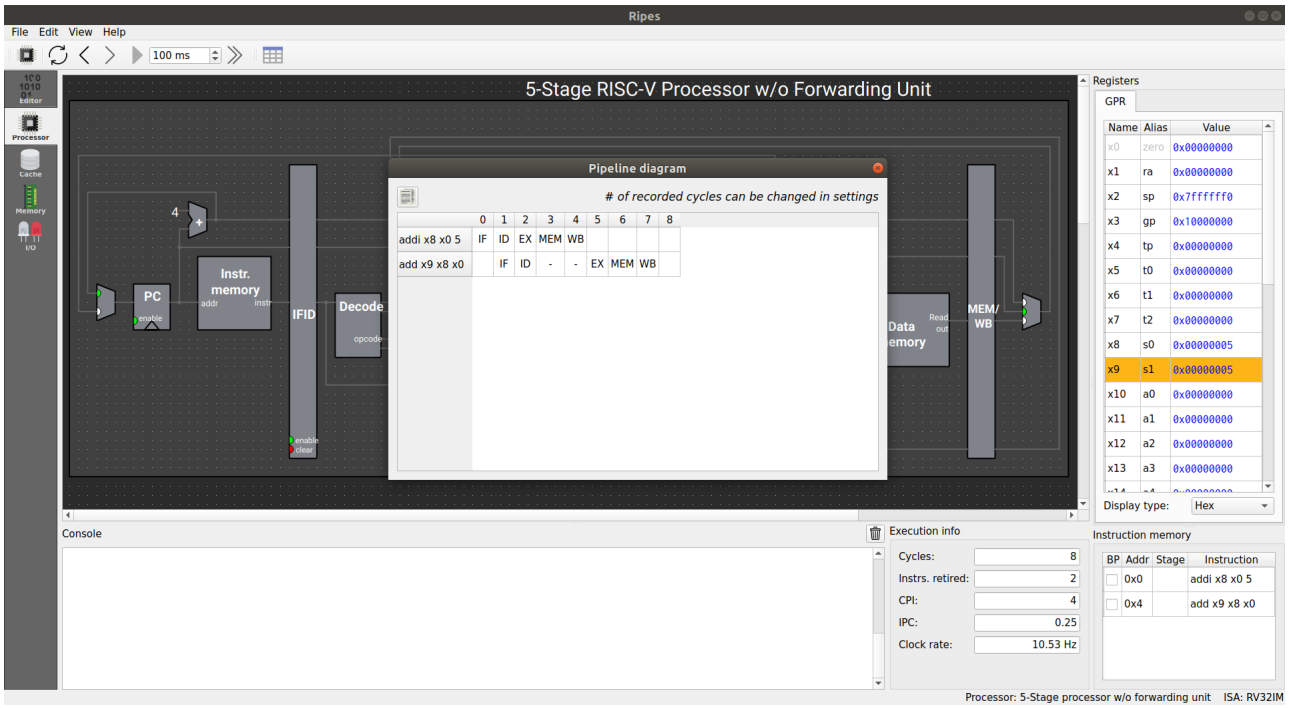

Figure 3: With hazard detection; Without forwarding;

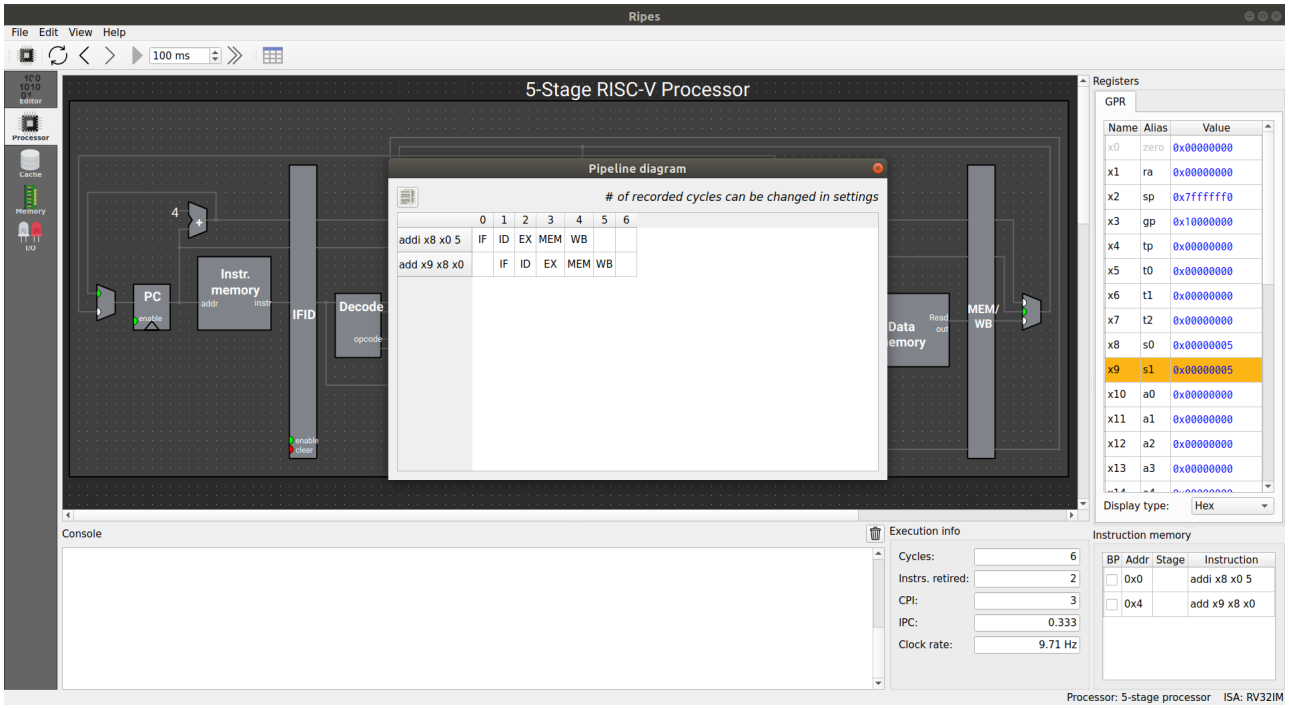

Figure 4: With hazard detection; With forwarding;

b.

Initial Code

```
.text

main:
    addi s0 zero 5
    add s1 s0 zero
    addi s2 zero 6
    addi s3 zero 7
```

Optimized Code

```
.text

main:
    addi s0 zero 5
    addi s2 zero 6
    addi s3 zero 7
    add s1 s0 zero
```

Behaviour

At the end of both the program, we expect registers `s0` and `s1` to have value 5, register `s2` to have value 6 and, register `s3` to have value 7.

Screenshots

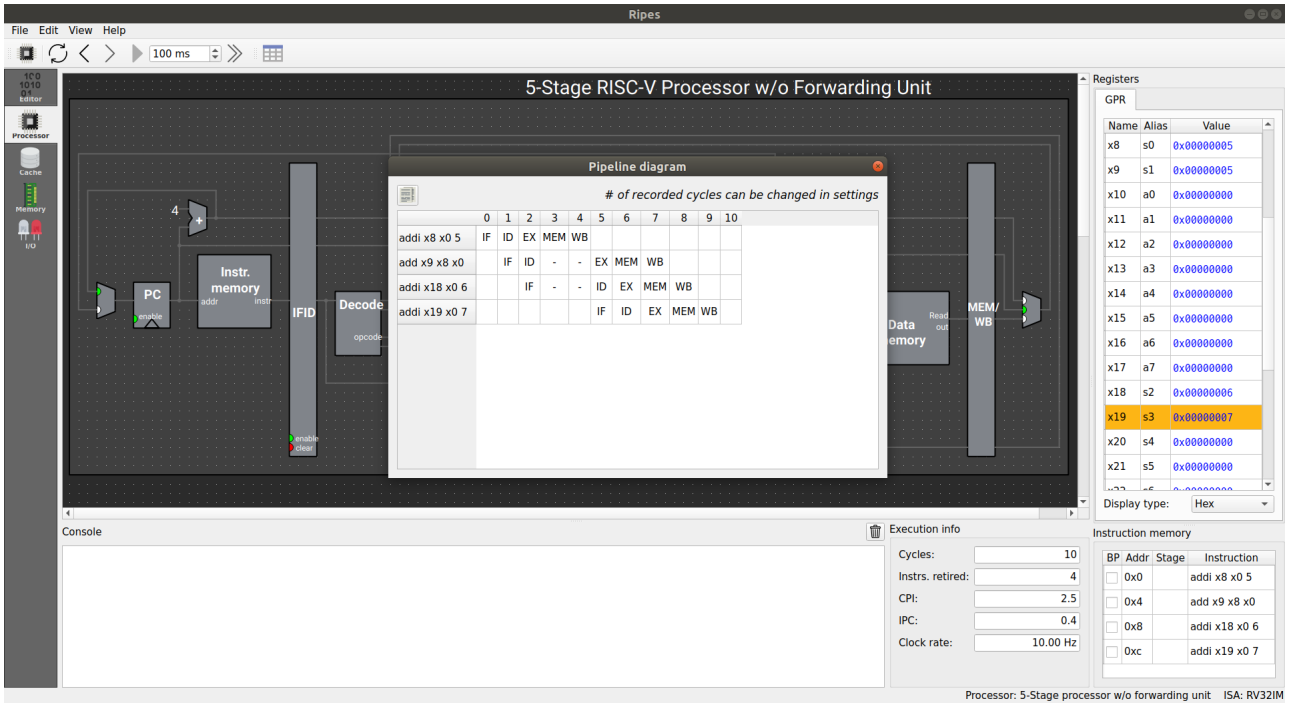

Figure 5: With hazard detection; Without forwarding; Initial code

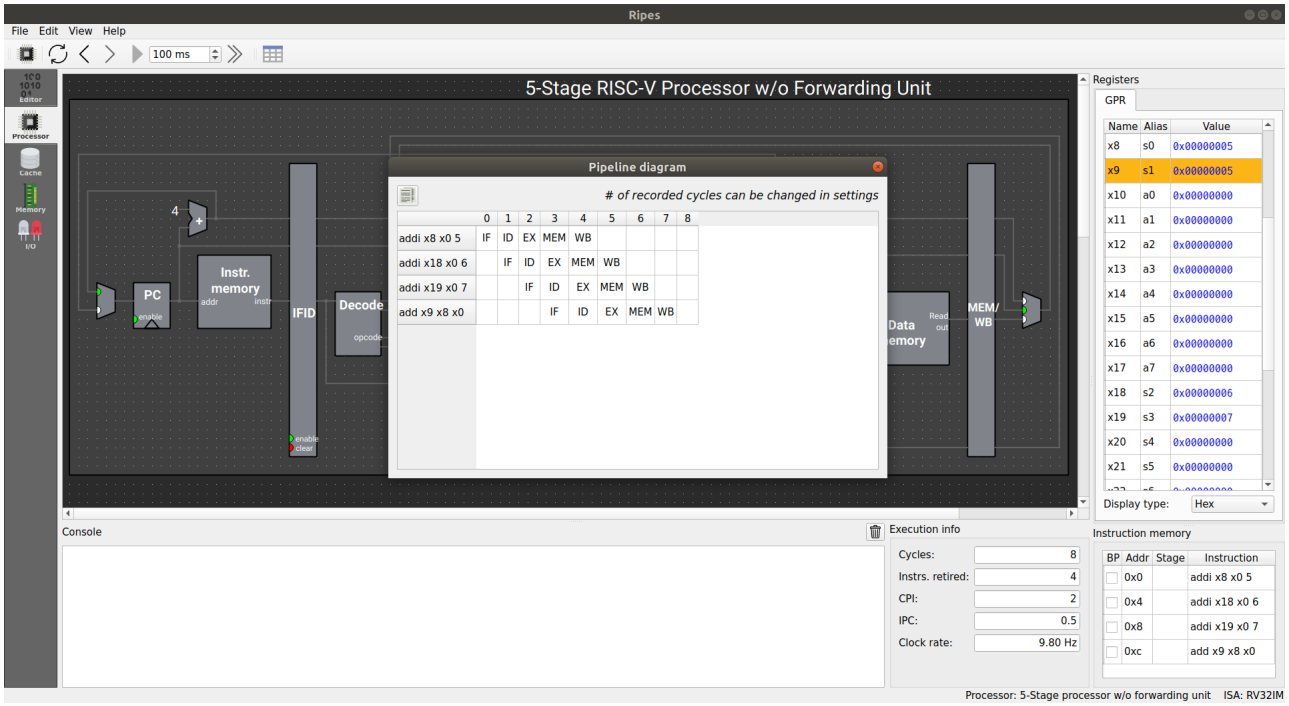

Figure 6: With hazard detection; With forwarding; Optimized code

3 Stalls and Forwarding

a.

Address input of data memory and EX/MEM pipeline after 4th cycle is `0x0000000a`.

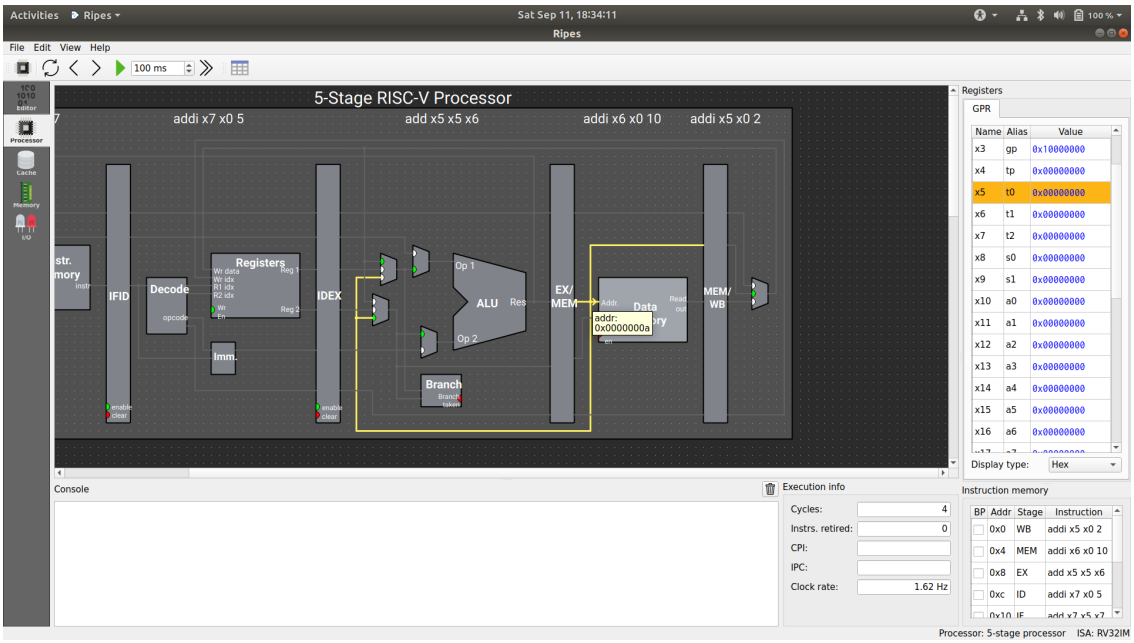

Figure 7: With hazard detection; With forwarding;

b.

R2 idx input of registers block and decoder after 10th cycle is `0x01`.

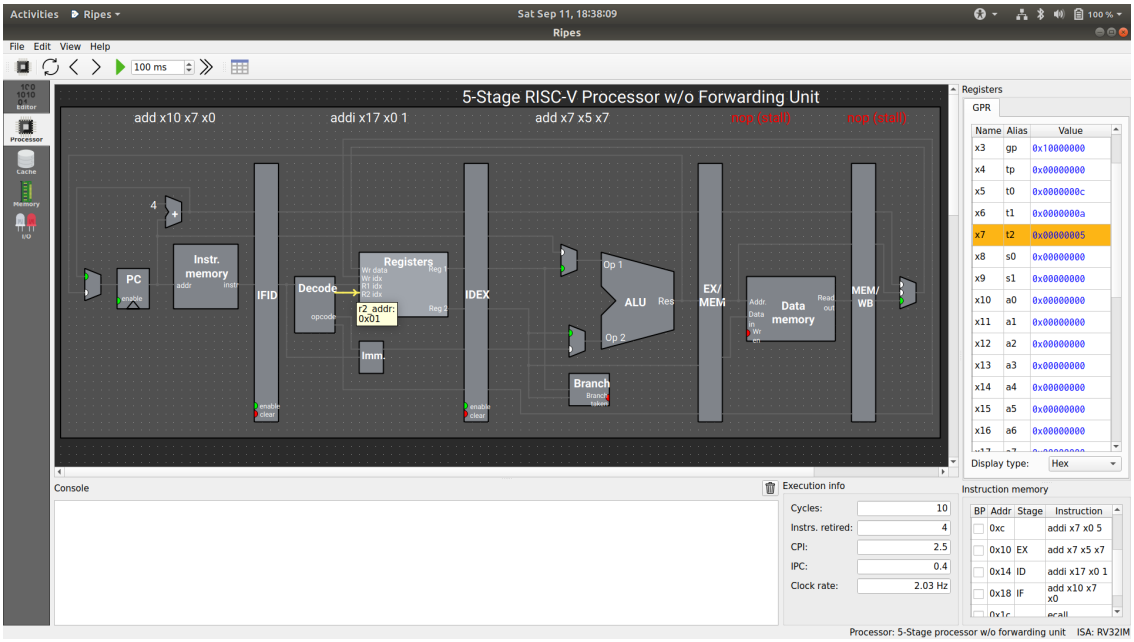

Figure 8: With hazard detection; Without forwarding;

c.

The value stored in `opcode_exec_out` datapath after 4 cycles is `0x13`.

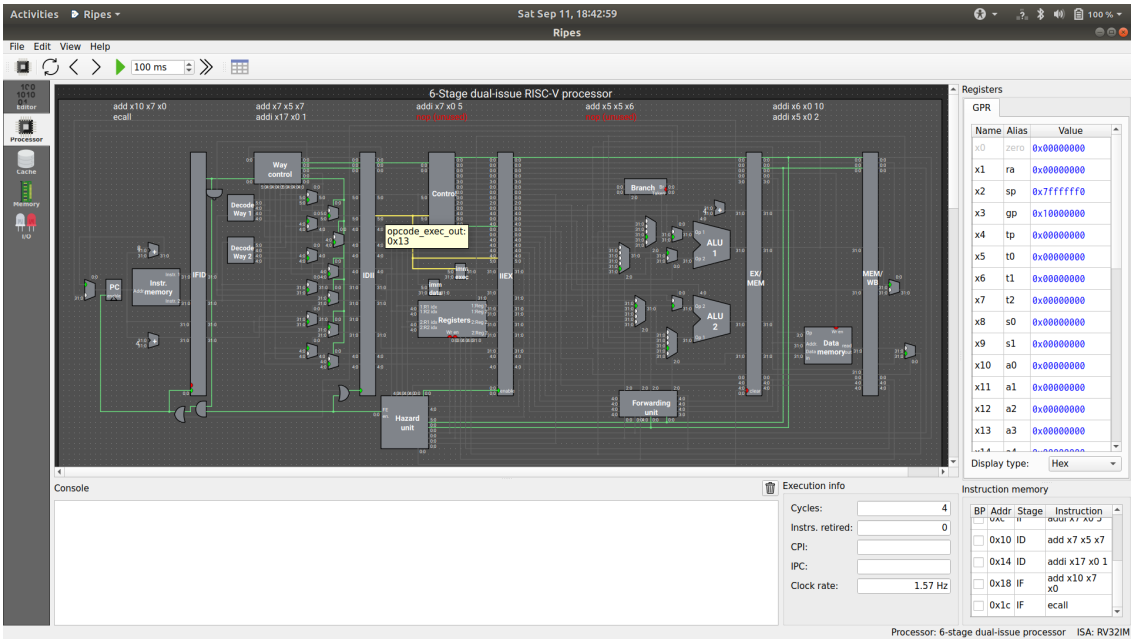

Figure 9: 6-stage dual-issue processor

d.

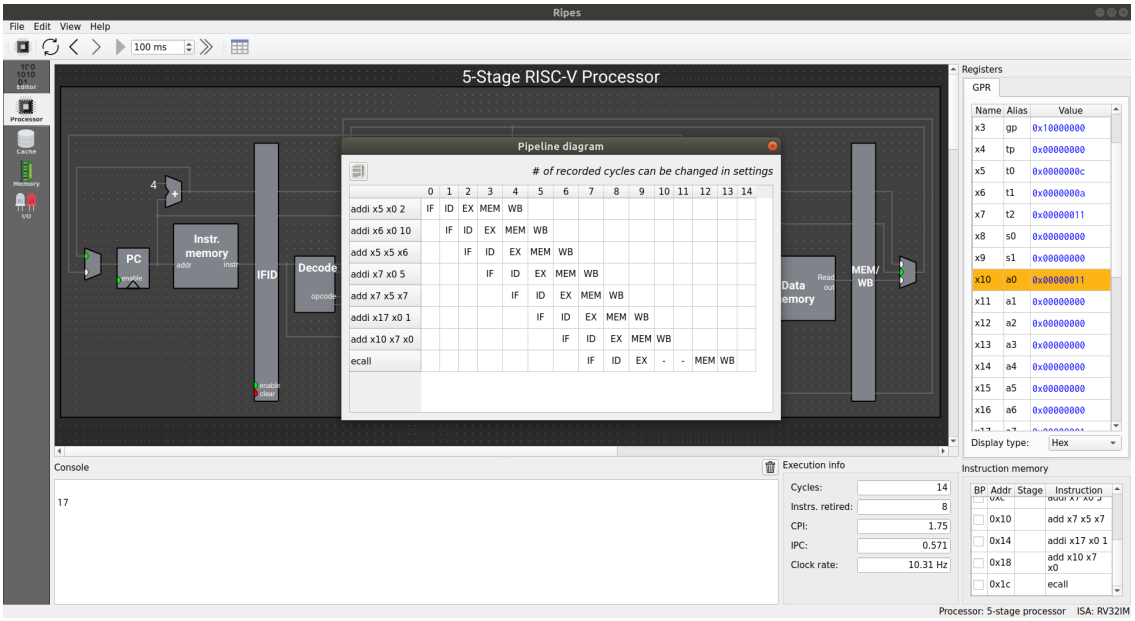

Figure 10: With hazard detection; With forwarding; **Takes 14 cycles**

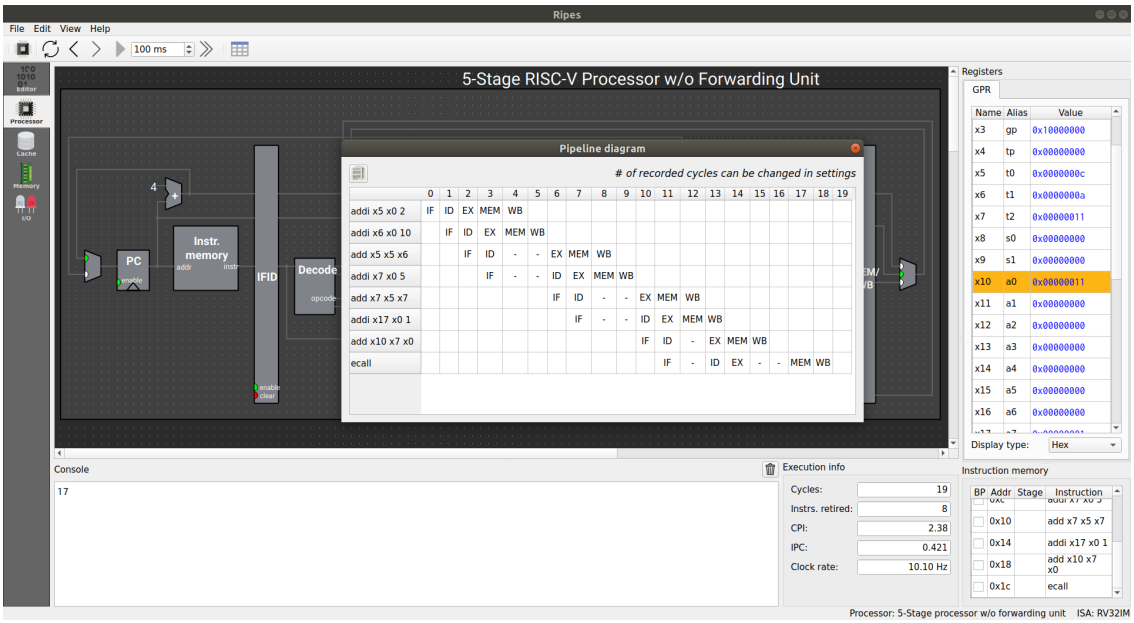

Figure 11: With hazard detection; Without forwarding; **Takes 19 cycles**

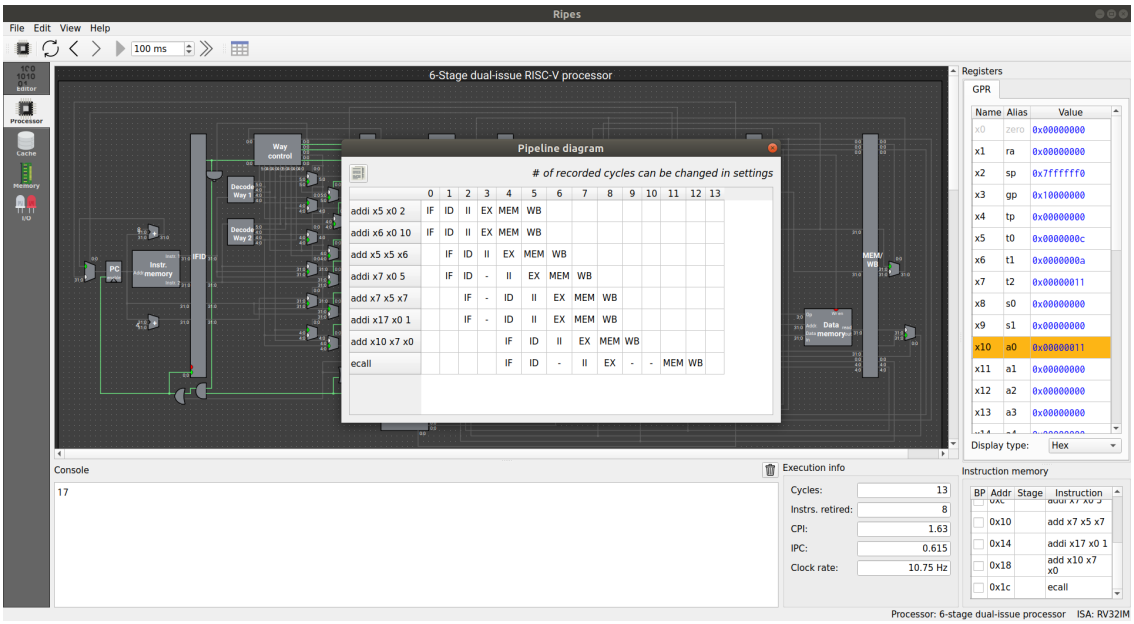

Figure 12: 6-stage dual-issue processor; **Takes 13 cycles**

e.

The major issue is that the value of register `a7` is not updated before `ecall` reaches ID stage. There is no system call with `a7` as 0, so an error occurs.

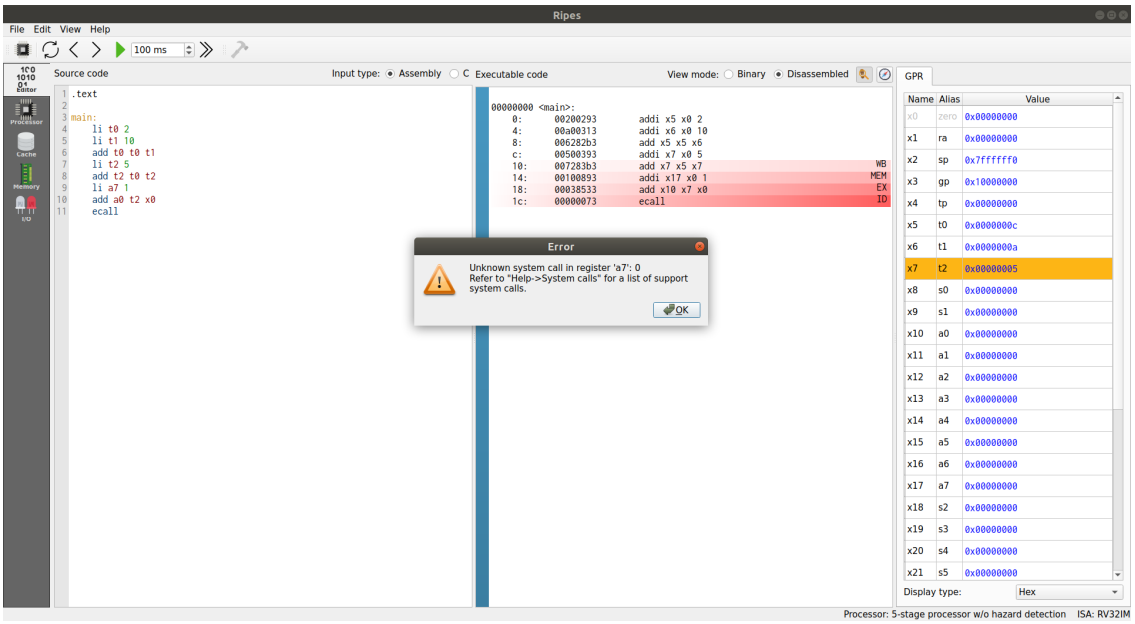

Figure 13: Without hazard detection; With forwarding; Error

One simple solution is to add `nop`s before `ecall`.
If we add two `nop`s, the error goes away but we end up with output 0 instead of 17.
This happens because the value of register `a0` is not updated before `ecall` sends to signal to the console.
If we add one more `nop`, the output is 17.

Updated code:

```
.text

main:
    li t0 2
    li t1 10
    add t0 t0 t1
    li t2 5
    add t2 t0 t2
    li a7 1
    add a0 t2 x0
    nop
    nop
    nop
    ecall
```

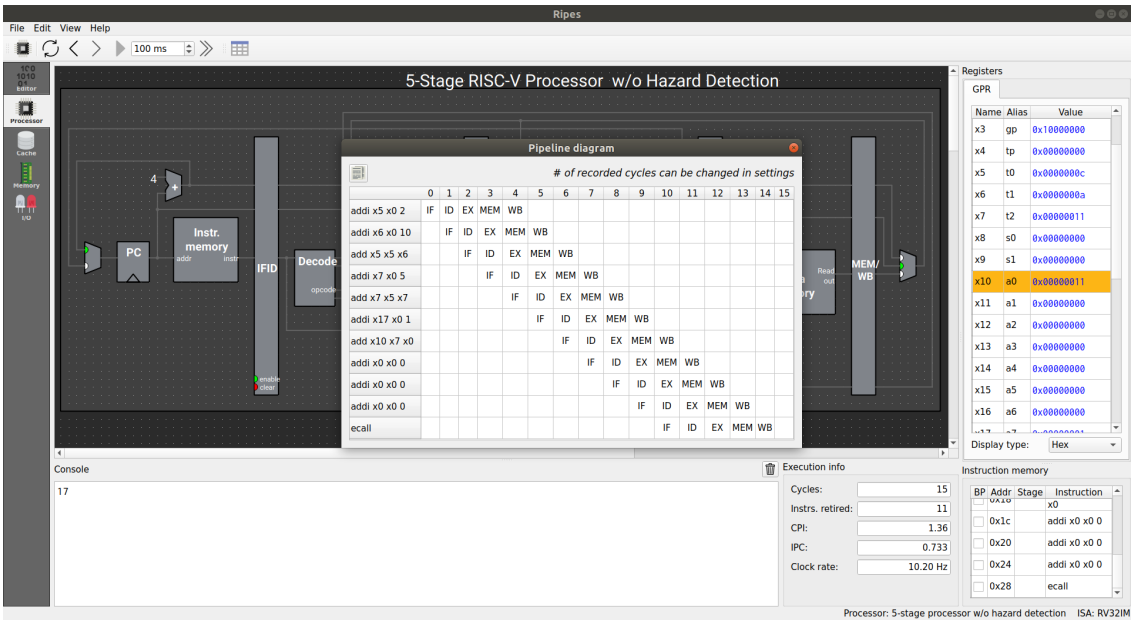

Figure 14: Without hazard detection; With forwarding; Fixed bug; **Takes 15 cycles**