Organization of Digital Computer LAB

EECS 112L

Lab2: Mips Single Cycle Processor

Student Name:TeHsun Wang

Student ID: 24944958

Date: Due 10/27

1 Objective

The objective of implementing pipelining in a processor, specifically following the MIPS instruction set architecture with its five stages (IF, ID, EXE, MEM, WB), is to enhance performance by overlapping the execution of multiple instructions. The pipeline design aims to keep each stage busy, thereby increasing instruction throughput. To address potential hazards such as structural, data, and control hazards that may disrupt the smooth flow of instructions, techniques like forwarding and hazard detection units are employed. Forwarding is utilized to resolve data dependencies by detecting and bypassing missing data, while a hazard detection unit helps identify situations where forwarding is insufficient, leading to pipeline stalls. The overall goal is to maximize the efficiency of instruction execution and minimize delays caused by various hazards in the pipelining process.

2 Procedure

Pipeline Stage Completion:

- Begin by comprehensively implementing each pipeline stage (IF, ID, EXE, MEM, WB) based on the provided design figures.
- Establish connections between stages by creating necessary wires to connect each module.

Pipeline Register Integration:

- Introduce pipeline registers strategically between each pair of consecutive stages. These registers will store essential data required for the execution of instructions in various stages.

Hazard Detection Integration:

- Import a hazard detection unit into the pipeline to identify potential data hazards during execution. This unit will play a crucial role in recognizing situations where data dependencies may disrupt the pipeline flow.

Forwarding Module Implementation:

- Develop the forwarding module to address data hazards efficiently. Detect and handle cases where a subsequent instruction relies on data not yet available by utilizing forwarding paths. Ensure that the forwarding unit is adept at bypassing the necessary information.

Connection to Data Memory:

- Establish connections to the data memory to determine addresses for read and write operations. This step is essential for accessing and storing data in memory during the MEM stage.

ALU Result Connection:

- Connect the ALU result to the pipeline stages where relevant. This connection is crucial for storing the results of arithmetic and logic operations performed in the EXE stage.

Write Back Data Assignment:

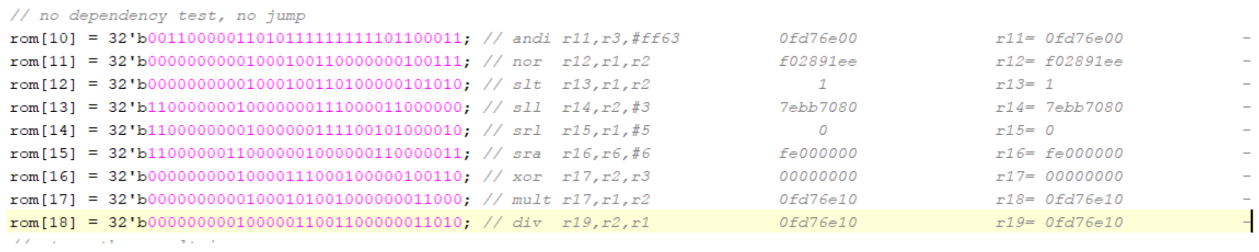
- Complete the pipeline by assigning the write-back data to the appropriate results. This step ensures that the final output of an instruction is correctly stored in the designated registers.

By following these steps, the MIPS-32 pipeline with forwarding will be effectively implemented, enhancing instruction throughput and addressing potential data hazards. This structured approach ensures a clear and organized development process for the pipeline architecture.

3 Simulation Results

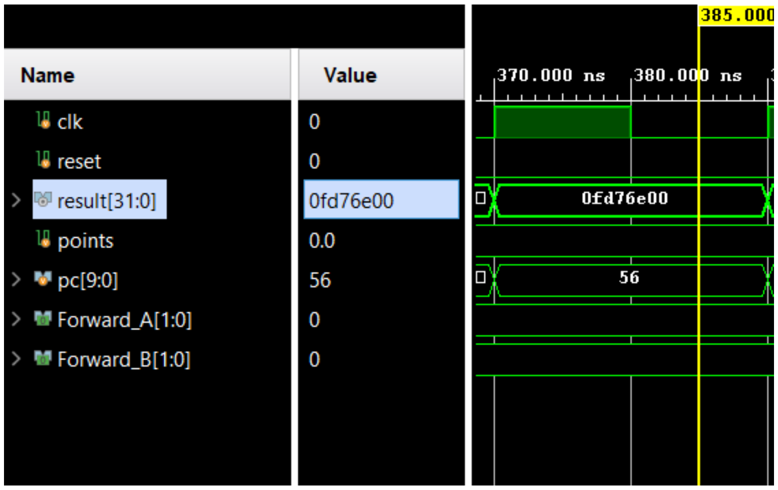
To test the result, we have to first find the binary code of that instruction and add it to the instruction memory. Even though we have all the information in data. I still added opcode, reg_write, reg_read_addr1 and 2 to find which instruction we are doing easier and also we can read rd,rs, and rt at the same time. Next, I went to find the instruction in the instruction memory and confirmed each field depending on the instruction format. Finally, we can observe the result easily in the waveform.

For No dependency checking:



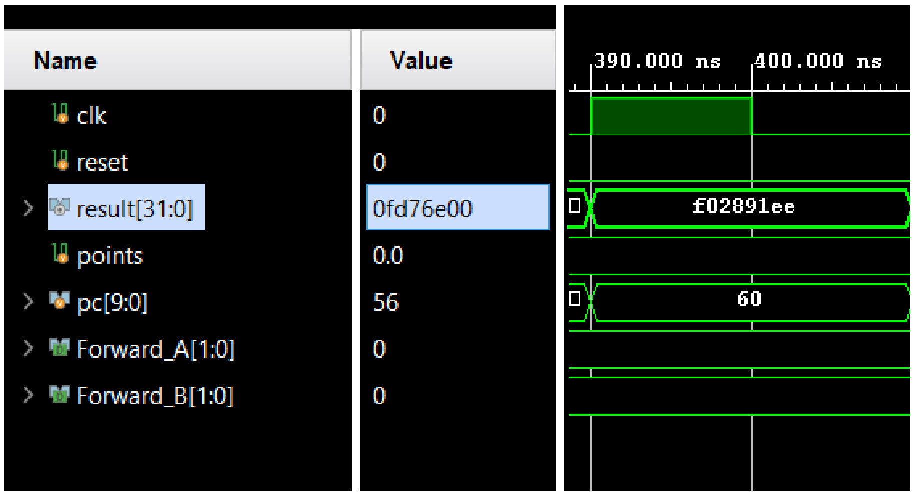
```
// no dependency test, no jump
rom[10] = 32'b00110000011010111111111101100011; // andi r11,r3,#ff63        0fd76e00          r11= 0fd76e00          -
rom[11] = 32'b00000000001000100110000000100111; // nor  r12,r1,r2           f02891ee          r12= f02891ee          -
rom[12] = 32'b00000000001000100110100000101010; // slt  r13,r1,r2               1             r13= 1                 -
rom[13] = 32'b11000000010000000111000011000000; // sll  r14,r2,#3           7ebb7080          r14= 7ebb7080          -
rom[14] = 32'b11000000001000000111100101000010; // srl  r15,r1,#5               0             r15= 0                 -
rom[15] = 32'b11000000110000001000000110000011; // sra  r16,r6,#6           fe000000          r16= fe000000          -
rom[16] = 32'b00000000010000111000100000100110; // xor  r17,r2,r3           00000000          r17= 00000000          -
rom[17] = 32'b00000000001000101001000000011000; // mult r17,r1,r2           0fd76e10          r18= 0fd76e10          -
rom[18] = 32'b00000000010000011001100000011010; // div  r19,r2,r1           0fd76e10          r19= 0fd76e10          -
```

1. No dependency andi



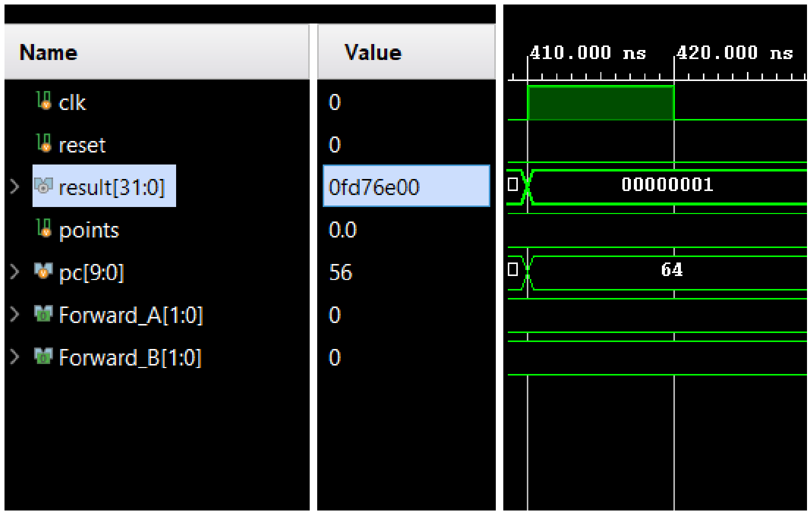
Name	Value
clk	0
reset	0
result[31:0]	0fd76e00
points	0.0
pc[9:0]	56
Forward_A[1:0]	0
Forward_B[1:0]	0

2. No dependency nor



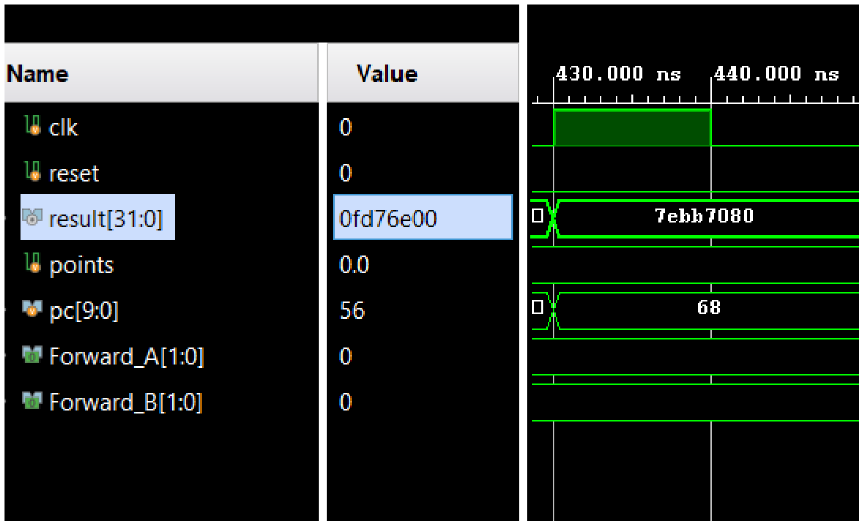
Name	Value
clk	0
reset	0
result[31:0]	0fd76e00
points	0.0
pc[9:0]	56
Forward_A[1:0]	0
Forward_B[1:0]	0

3. No dependency slt



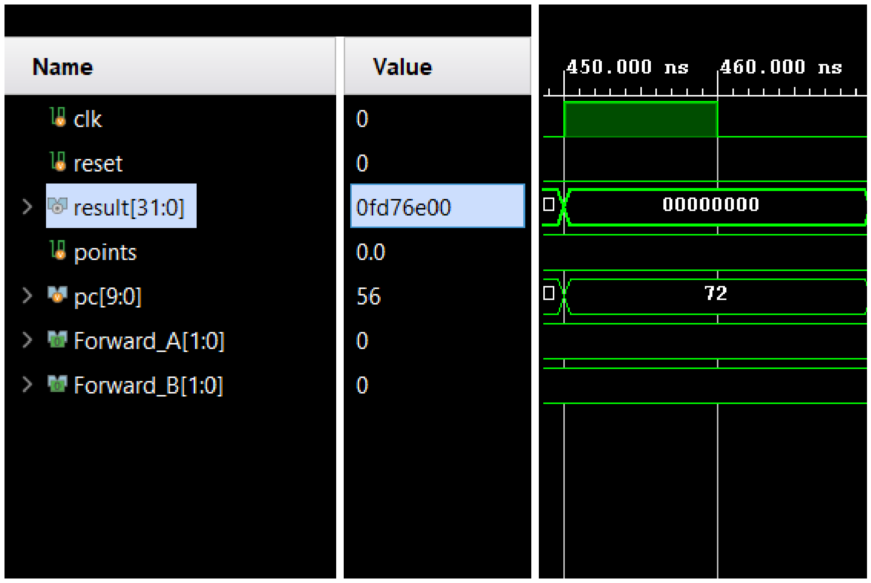
Name	Value
clk	0
reset	0
result[31:0]	0fd76e00
points	0.0
pc[9:0]	56
Forward_A[1:0]	0
Forward_B[1:0]	0

4. No dependency sll



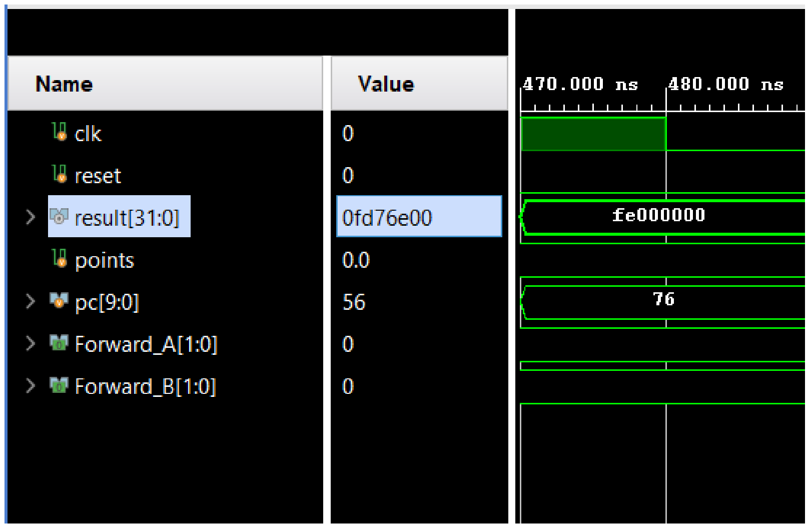
Name	Value
clk	0
reset	0
result[31:0]	0fd76e00
points	0.0
pc[9:0]	56
Forward_A[1:0]	0
Forward_B[1:0]	0

5. No dependency srl



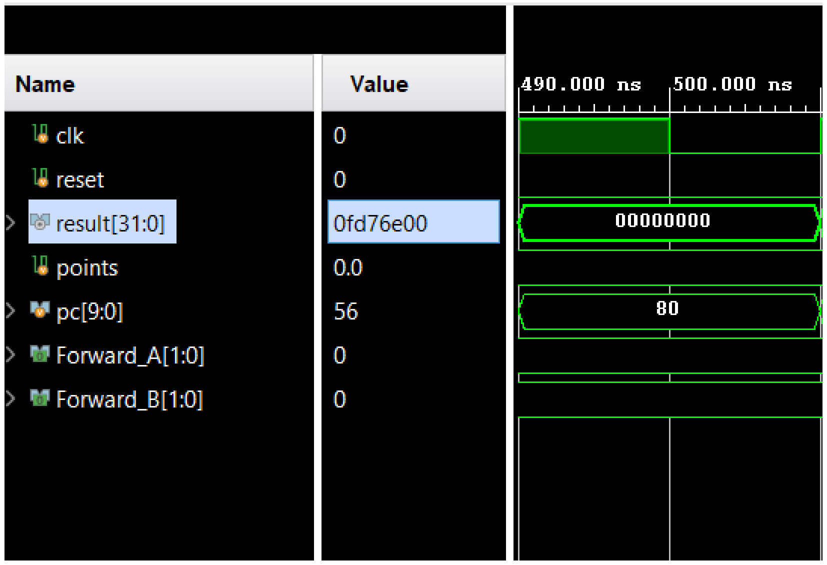
Name	Value	
clk	0	
reset	0	
result[31:0]	0fd76e00	00000000
points	0.0	
pc[9:0]	56	72
Forward_A[1:0]	0	
Forward_B[1:0]	0	

6. No dependency sra



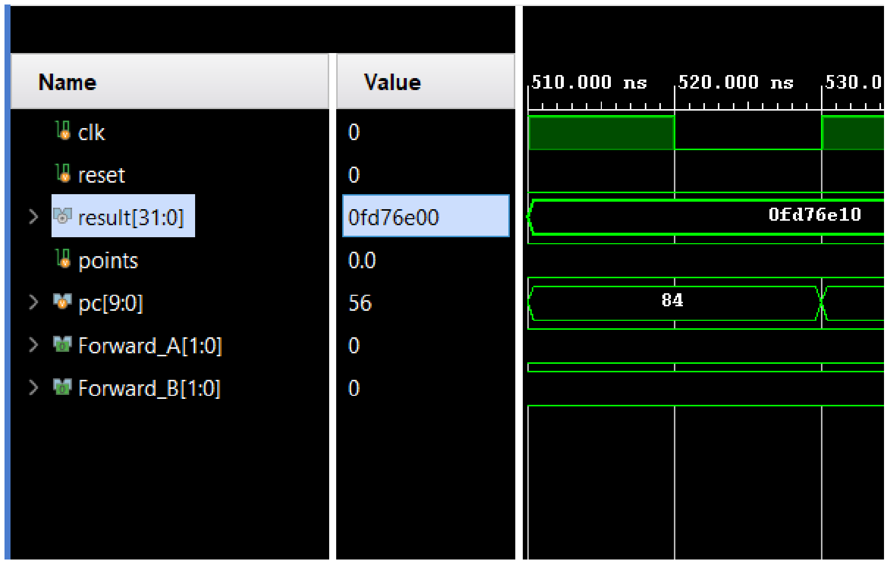
Name	Value	
clk	0	
reset	0	
result[31:0]	0fd76e00	fe000000
points	0.0	
pc[9:0]	56	76
Forward_A[1:0]	0	
Forward_B[1:0]	0	

7. No dependency xor



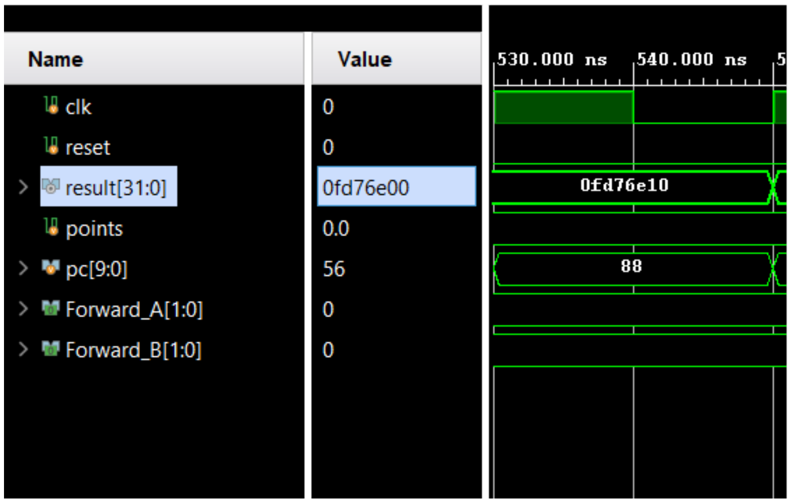
```
clk              0
reset            0
result[31:0]     0fd76e00
points           0.0
pc[9:0]          56
Forward_A[1:0]   0
Forward_B[1:0]   0
```

8. No dependency mult



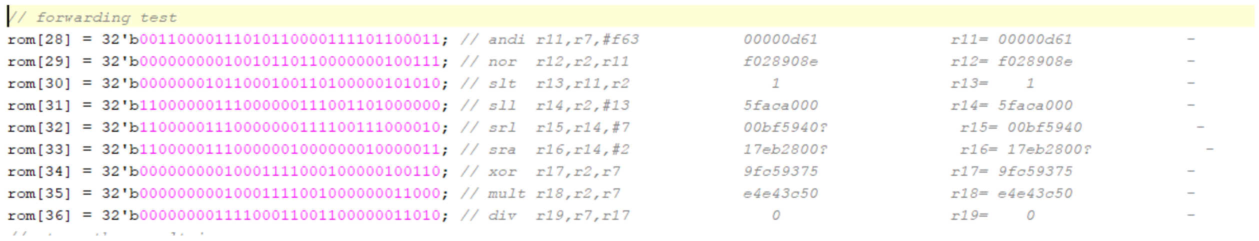
```
clk              0
reset            0
result[31:0]     0fd76e00
points           0.0
pc[9:0]          56
Forward_A[1:0]   0
Forward_B[1:0]   0
```

9. No dependency div



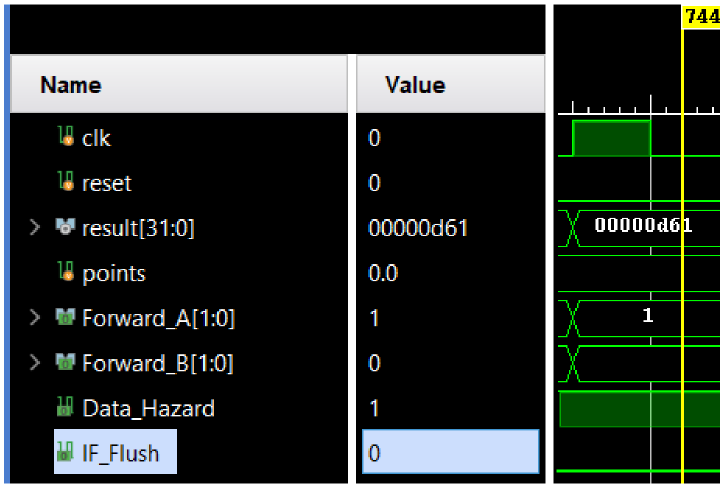
Name	Value
clk	0
reset	0
result[31:0]	0fd76e00
points	0.0
pc[9:0]	56
Forward_A[1:0]	0
Forward_B[1:0]	0

For Forwarding checking



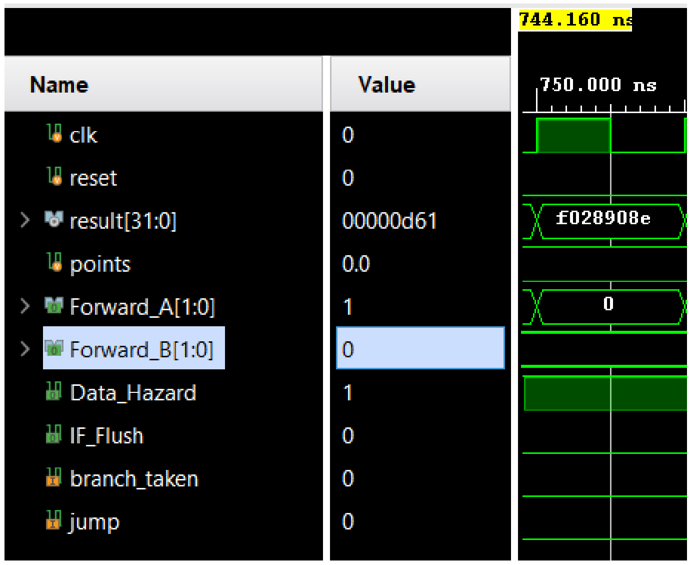
```
// forwarding test
rom[28] = 32'b00110000111010110000111101100011; // andi r11,r7,#f63          00000d61          r11= 00000d61          -
rom[29] = 32'b00000000010010110110000000100111; // nor  r12,r2,r11           f028908e          r12= f028908e          -
rom[30] = 32'b00000001011000100110100000101010; // slt  r13,r11,r2              1              r13=    1              -
rom[31] = 32'b11000000111000000111001101000000; // sll  r14,r2,#13           5faca000          r14= 5faca000          -
rom[32] = 32'b11000001110000000111100111000010; // srl  r15,r14,#7           00bf5940?          r15= 00bf5940           -
rom[33] = 32'b11000001110000001000000010000011; // sra  r16,r14,#2           17eb2800?          r16= 17eb2800?           -
rom[34] = 32'b00000000010001111000100000100110; // xor  r17,r2,r7            9fc59375          r17= 9fc59375          -
rom[35] = 32'b00000000010001111001000000011000; // mult r18,r2,r7            e4e43c50          r18= e4e43c50          -
rom[36] = 32'b00000000111100011001100000011010; // div  r19,r7,r17              0              r19=    0              -
```

10. ANDI No Forwarding



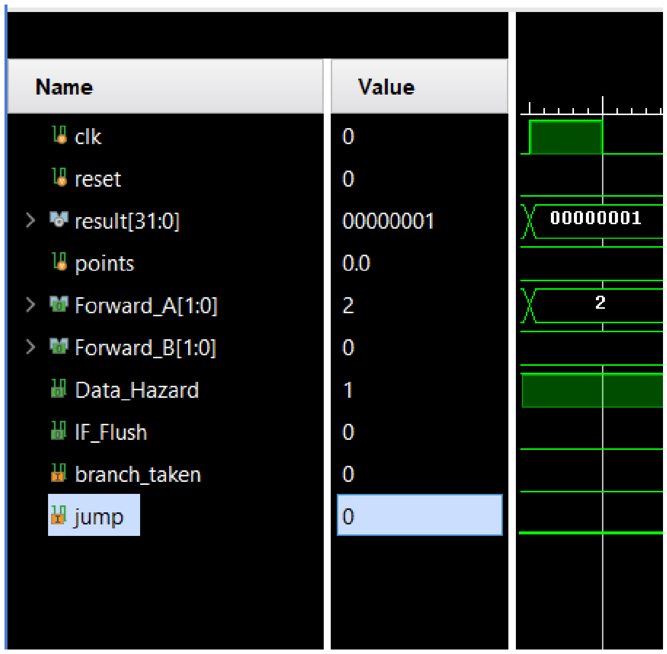
Name	Value
clk	0
reset	0
result[31:0]	00000d61
points	0.0
Forward_A[1:0]	1
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0

11. Forward EX/MEM to EX B



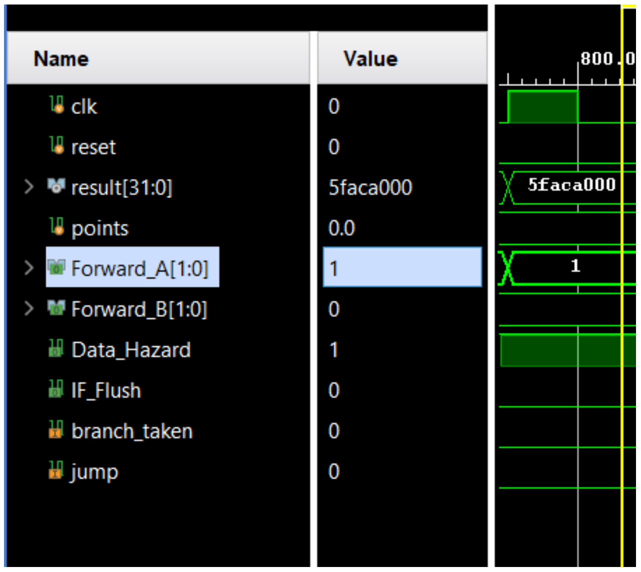
Name	Value
clk	0
reset	0
result[31:0]	00000d61
points	0.0
Forward_A[1:0]	1
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

12. Forward MEM/WB to EX A



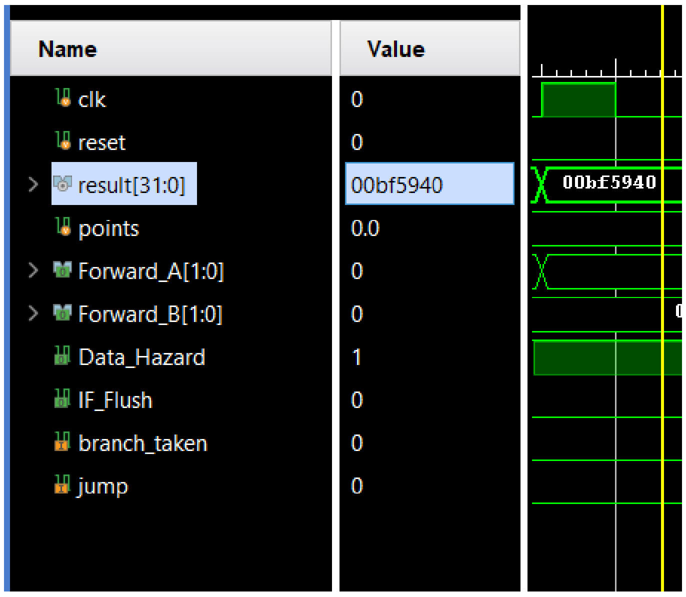
Name	Value
clk	0
reset	0
result[31:0]	00000001
points	0.0
Forward_A[1:0]	2
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

13. SLL No Forwarding



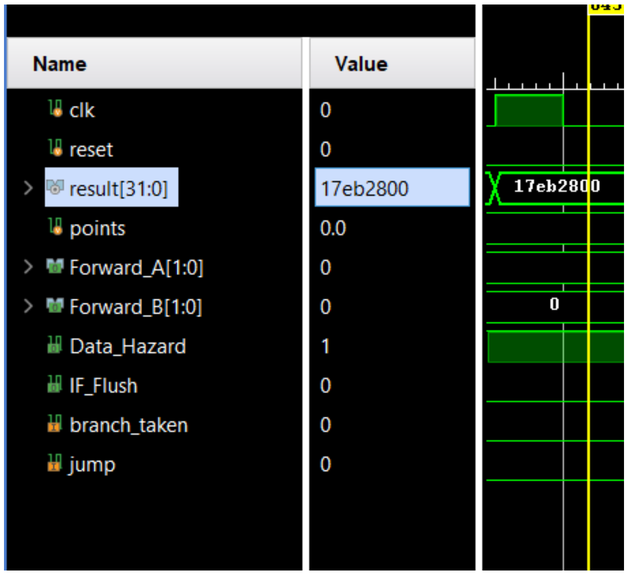
Name	Value
clk	0
reset	0
result[31:0]	5faca000
points	0.0
Forward_A[1:0]	1
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

14. Forward EX/MEM to EX A



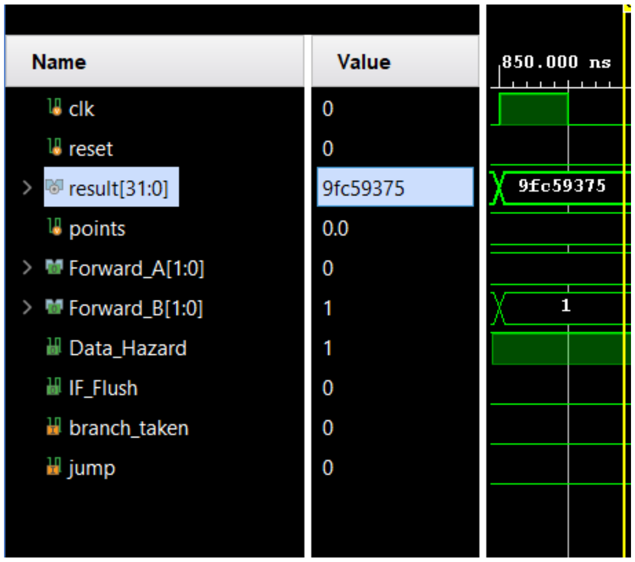
Name	Value
clk	0
reset	0
result[31:0]	00bf5940
points	0.0
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

15. Forward MEM/WB to EX A



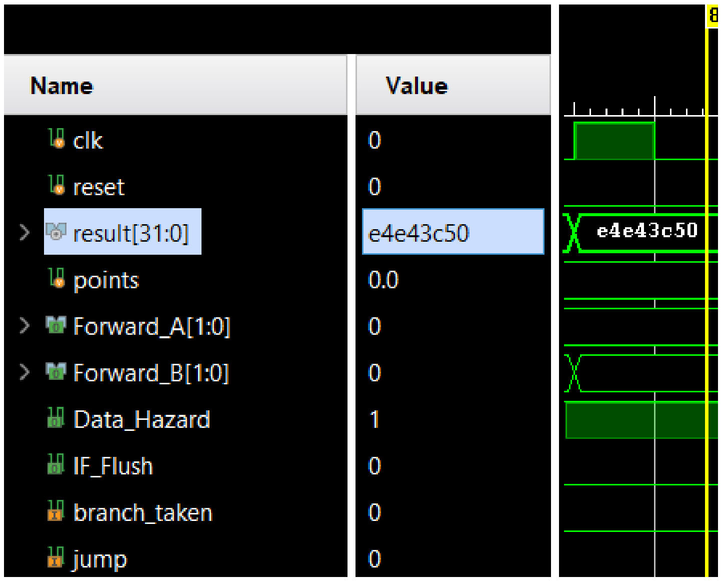
Name	Value
clk	0
reset	0
result[31:0]	17eb2800
points	0.0
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

16. XOR No Forwarding



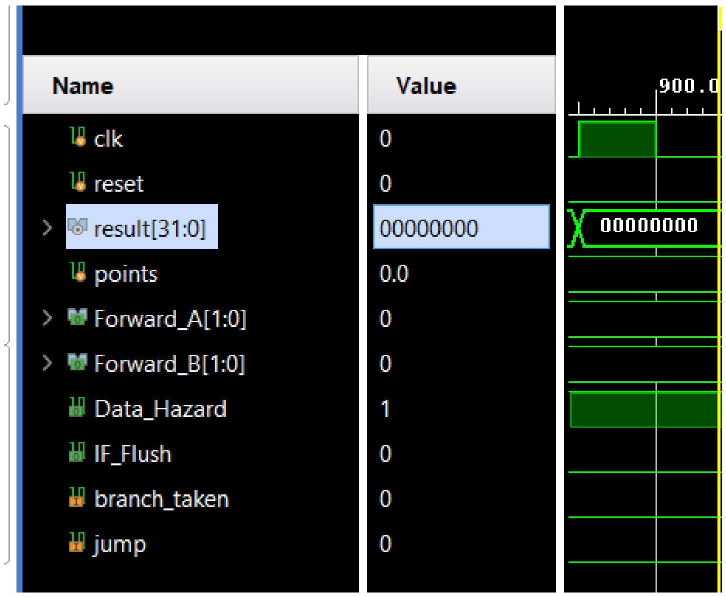
Name	Value
clk	0
reset	0
result[31:0]	9fc59375
points	0.0
Forward_A[1:0]	0
Forward_B[1:0]	1
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

17. MULT No Forwarding



Name	Value
clk	0
reset	0
result[31:0]	e4e43c50
points	0.0
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

18. Forward MEM/WB to EX B

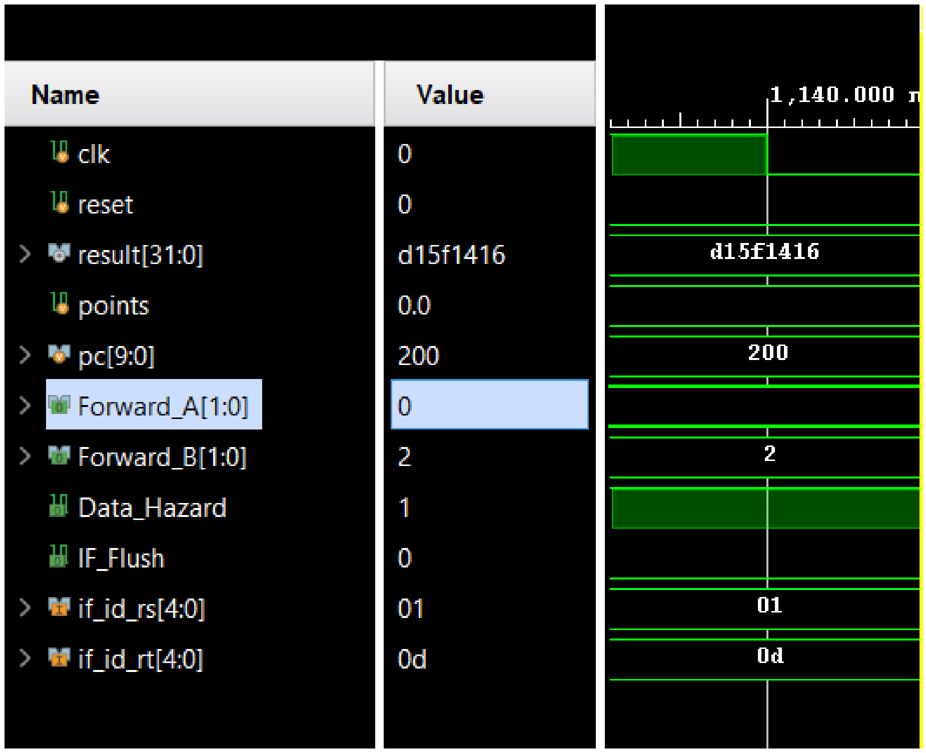


Name	Value
clk	0
reset	0
result[31:0]	00000000
points	0.0
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
branch_taken	0
jump	0

For Data Hazard

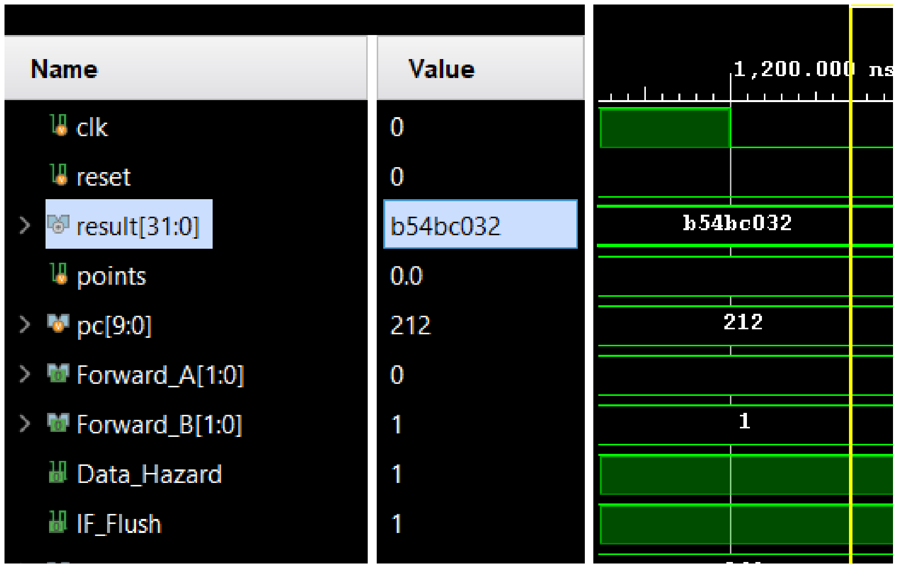
```
rom[47] = 32'b00000001011000100110000000100000; // add r12,r11,r2          d15f1416          r12= d15f1416

rom[49] = 32'b00000000001011010111000000100000; // add r14,r1,r13          b54bc032          r14= b54bc032
```

1. Data Hazard RS Dependency



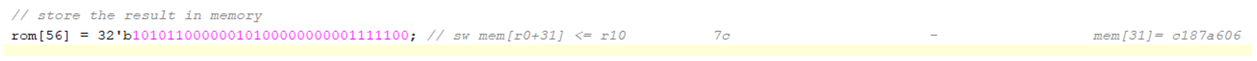
```
Name              Value
clk               0
reset             0
result[31:0]      d15f1416
points            0.0
pc[9:0]           200
Forward_A[1:0]    0
Forward_B[1:0]    2
Data_Hazard       1
IF_Flush          0
if_id_rs[4:0]     01
if_id_rt[4:0]     0d
```

2. Data Hazard RT Dependency

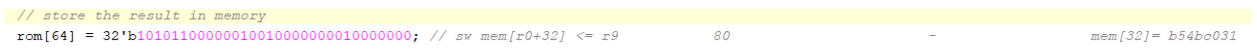


```
Name              Value
clk               0
reset             0
result[31:0]      b54bc032
points            0.0
pc[9:0]           212
Forward_A[1:0]    0
Forward_B[1:0]    1
Data_Hazard       1
IF_Flush          1
```

For Control Hazard

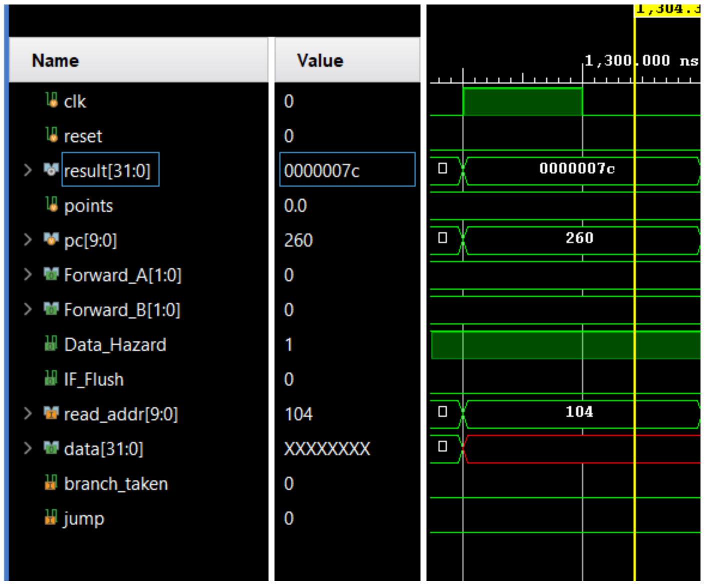


```
// store the result in memory
rom[56] = 32'b10101100000001010000000000111100; // sw mem[r0+31] <= r10          7c          -          mem[31]= c187a606
```



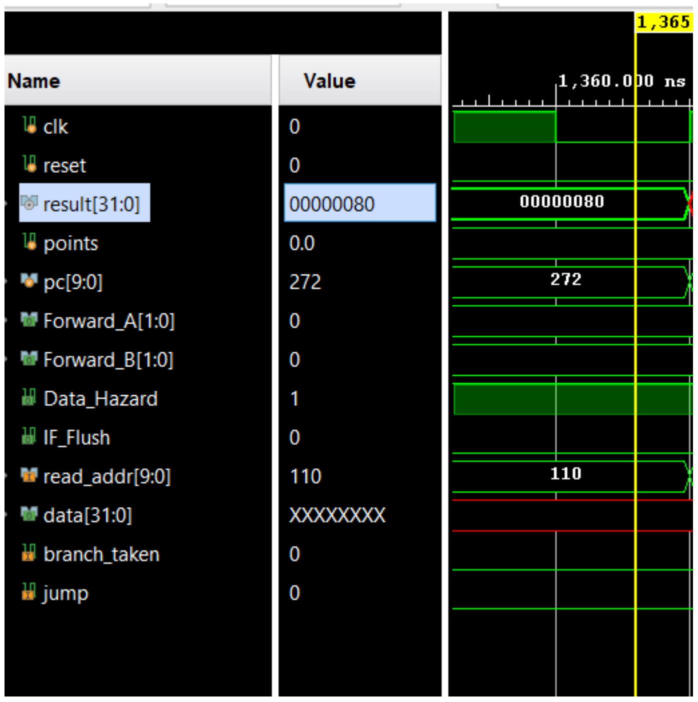
```
// store the result in memory
rom[64] = 32'b10101100000001001000000000100000000; // sw mem[r0+32] <= r9          80          -          mem[32]= b54bc031
```

1. Control Hazard Branch



Name	Value
clk	0
reset	0
result[31:0]	0000007c
points	0.0
pc[9:0]	260
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
read_addr[9:0]	104
data[31:0]	XXXXXXXX
branch_taken	0
jump	0

2. Control Hazard Jump



Name	Value
clk	0
reset	0
result[31:0]	00000080
points	0.0
pc[9:0]	272
Forward_A[1:0]	0
Forward_B[1:0]	0
Data_Hazard	1
IF_Flush	0
read_addr[9:0]	110
data[31:0]	XXXXXXXX
branch_taken	0
jump	0

This is my result after run 2000:

```
NO DEPENDENCY ANDI  success!

NO DEPENDENCY NOR   success!

NO DEPENDENCY SLT   success!

NO DEPENDENCY SLL   success!

NO DEPENDENCY SRL   success!

NO DEPENDENCY SRA   success!

NO DEPENDENCY XOR   success!

NO DEPENDENCY MULT  success!

NO DEPENDENCY DIV   success!

ANDI No Forwarding       success!

Forward EX/MEM to EX B   success!

Forward MEM/WB to EX A  success!

SLL  No Forwarding      success!

Forward EX/MEM to EX A  success!

Forward MEM/WB to EX A  success!

XOR  No Forwarding      success!

MULT No Forwarding      success!

Forward MEM/WB to EX B  success!

DATA HAZARD RS DEPENDENCY    success!

DATA HAZARD RT DEPENDENCY   success!

CONTROL HAZARD BRANCH   success!

CONTROL HAZARD JUMP success!
```