9/4/21
EC413
Ivan Isakov

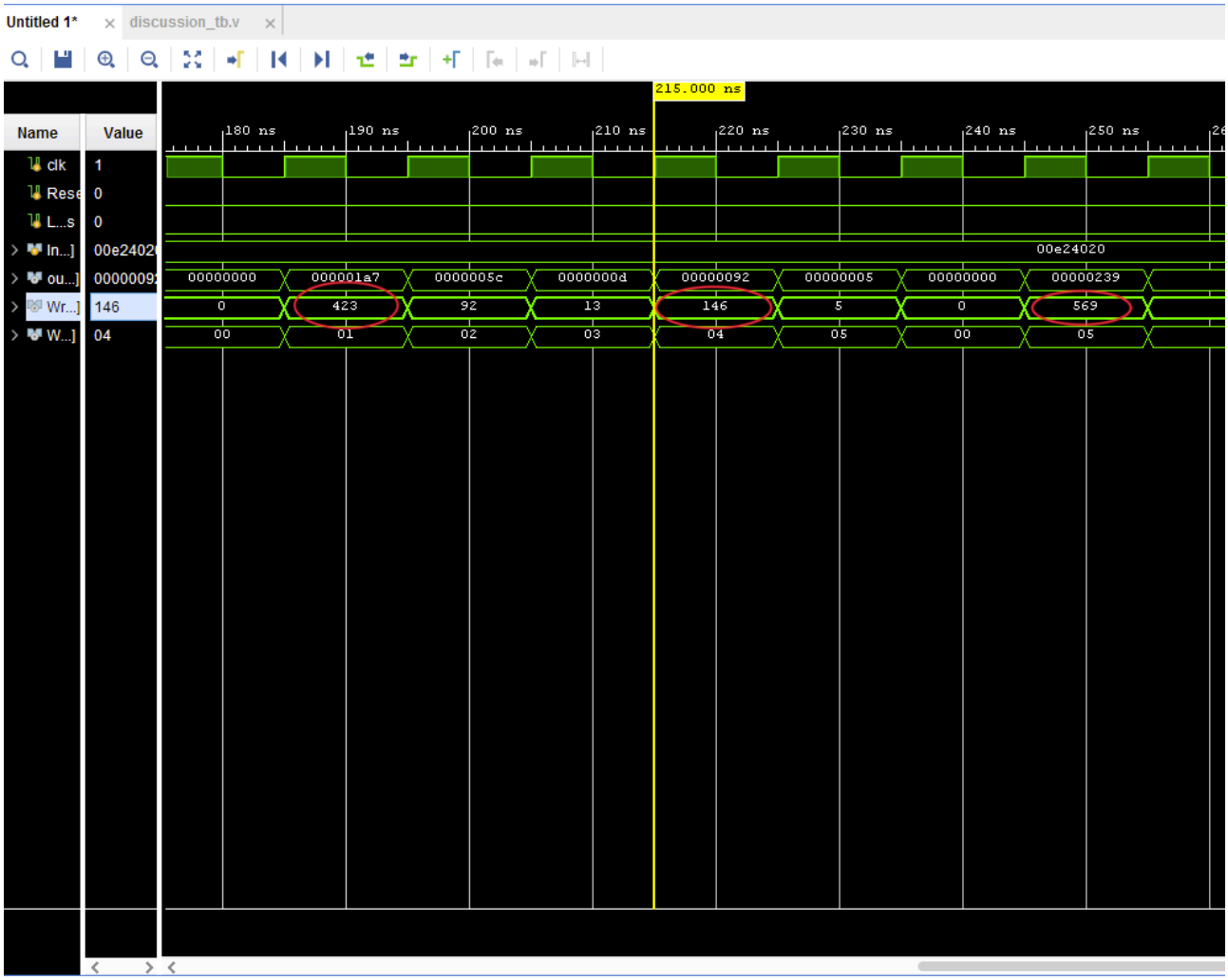
Lab 8

In the hazard detection module, I compare the registers that may have data dependency in the consecutive instructions. I use the specific bits of the instruction to check the destination registers and the source registers. And I also check whether the there is a RegWrite signal associated with each instruction that may cause a data hazard.

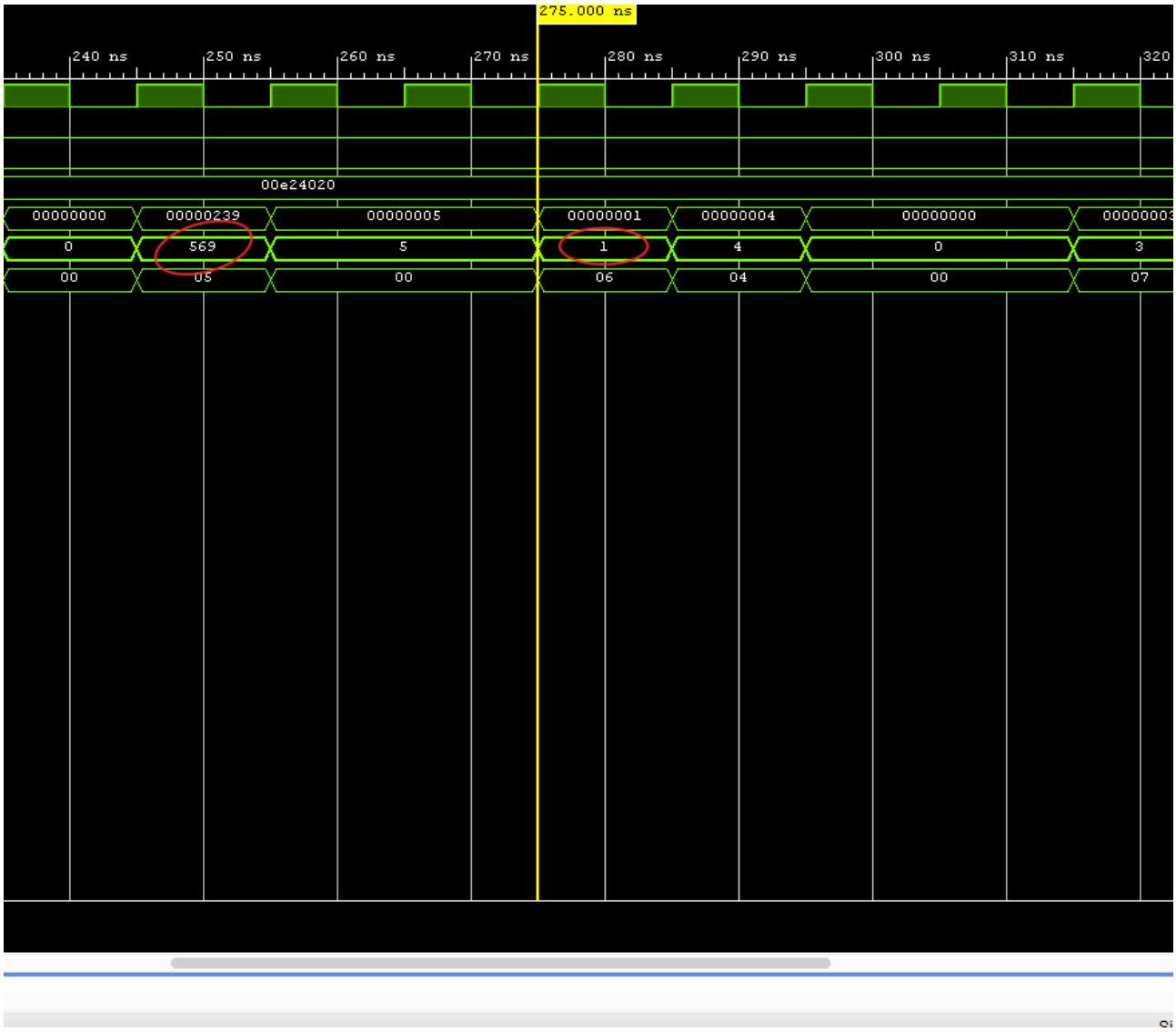
This is the testbench that I am using,

```
51          // begin by loading values into registers
52          Instruction = 32'b001000_00000_00001_0000000110100111; //addi $R1, 423
53          #10;
54          Instruction = 32'b001000_00000_00010_0000000001011100; //addi $R2, 92
55          #10;
56          Instruction = 32'b001000_00000_00011_0000000000001101; //addi $R3, 13
57          #10;
58          Instruction = 32'b001000_00000_00100_0000000010010010; //addi $R4, 146    // 2
59          #10;
60          Instruction = 32'b001000_00000_00101_0000000000000101; //addi $R5, 5
61          #10;
62          Instruction = 32'b000000_00001_00100_00101_00000_100000; //add  $R5, $R1, $R4     // 2 & 1
63          #10;
64          Instruction = 32'b000000_00011_00101_00110_00000_101010; //slt  $R6, $R3, $R5     // 1
65          #10;
66          Instruction = 32'b100011_00000_00100_00000_00000_000100;    // LW $R4, 4(R0)     // 3
67
68          #10;
69          Instruction = 32'b000000_00100_00110_00111_00000_100010; //sub  $R7, $R4, $R6     // 3
70          #10;
71          Instruction = 32'b101011_00000_00111_00000_00000_000000;    // SW R7, 0(R0)
72          #10;
73          Instruction = 32'b000000_00111_00010_01000_00000_100000;    // add R8, R7, R2
74          #10;
75
76          LoadInstructions = 0;
77          Reset = 1;
78          #10;
79          Reset = 0;
80          #100;
81
```

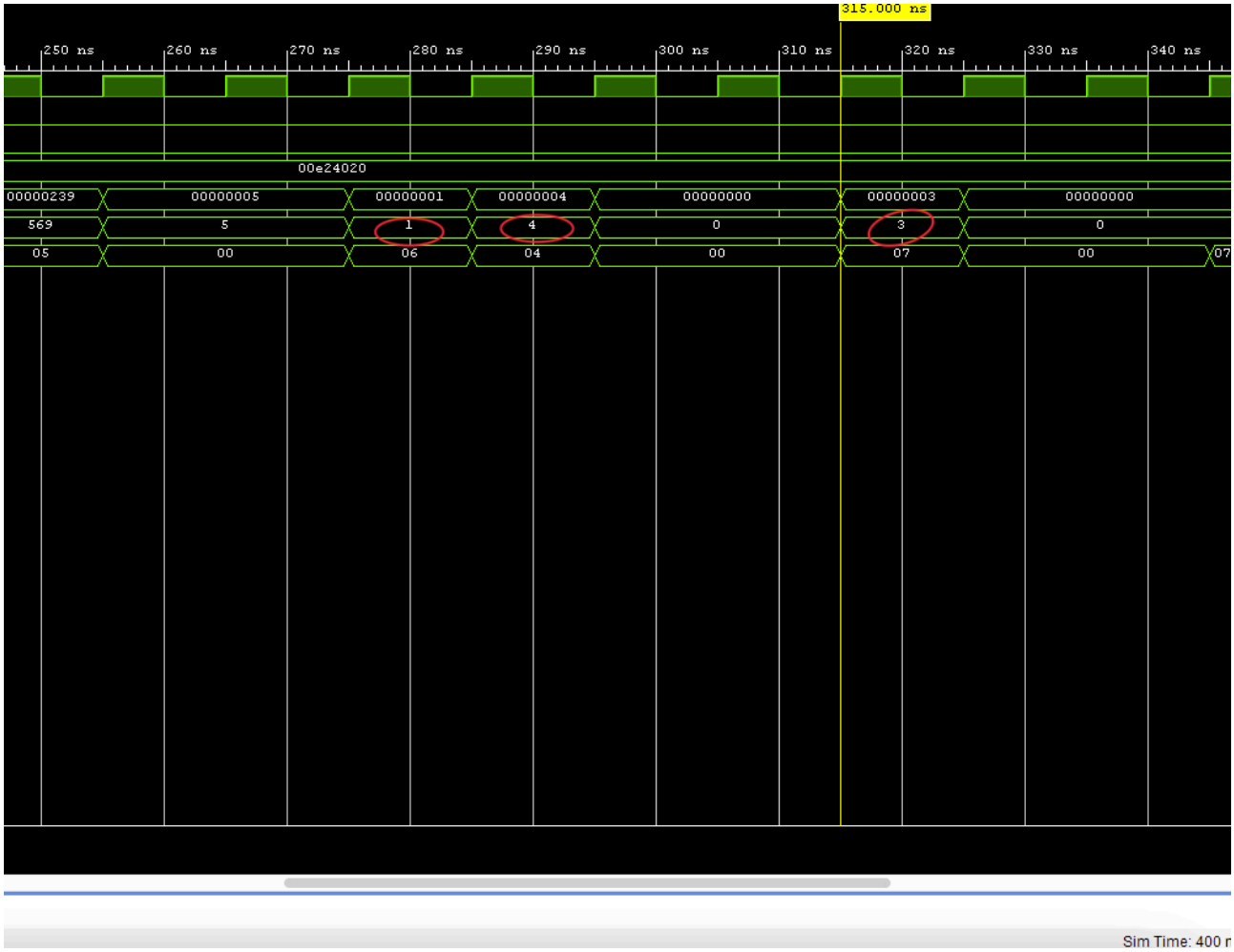
As we can see there is a 2-ahead hazard occurring, indicated by the // 2. This means that the CPU has to wait 2 clock cycles for the value in the destination register to be the correct one. The instruction where we are trying to prevent a hazard from happening means add 423 and 146, which equals 569 as is confirmed in the waveform below



As we can see there is a 1-ahead hazard occurring, indicated by the // 1. This means that the CPU has to wait 1 clock cycles for the value in the destination register to be the correct one. The instruction where we are trying to prevent a hazard from happening means if(13 < 569) -> 1, else -> 0 as is confirmed in the waveform below



As we can see there is a load-use hazard occurring, indicated by the // 3. This means that the CPU has to wait 2 clock cycles for the value in the destination register to be the correct one. The instruction where we are trying to prevent a hazard from happening means 4 - 1, which equals to 3 as is confirmed in the waveform below


315.000 ns
250 ns
260 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
330 ns
340 ns
00e24020
00000239
00000005
00000001
00000004
00000000
00000003
00000000
569
5
1
4
0
3
0
05
00
06
04
00
07
00
Sim Time: 400 n