

CE3001-Advance Computer Architecture

Experiment 2 Lab Report

By Chandra Bhavya

LAB REPORT 3

# Explain the function of three-stage pipelined datapath with testbench for the execution of Rtype instruction with the necessary RTL-block diagram.

**16-BIT R-TYPE INSTRUCTION FORMAT:**

|  |  |  |  |
| --- | --- | --- | --- |
| Op-code | Rs | Rt | Rd |
| 4-bit | 4-bit | 4-bit | 4-bit |

The 3-stage pipeline for R-type instruction can be seen from figure 1 below. It consists of *Instruction Fetch, Instruction Decode, and Execute Stage.* The following demonstrates an example of the pipeline process.Since I-type instructions are not included, imm\_ID\_EXE is ignored.

**FIRST CLOCK CYCLE:**

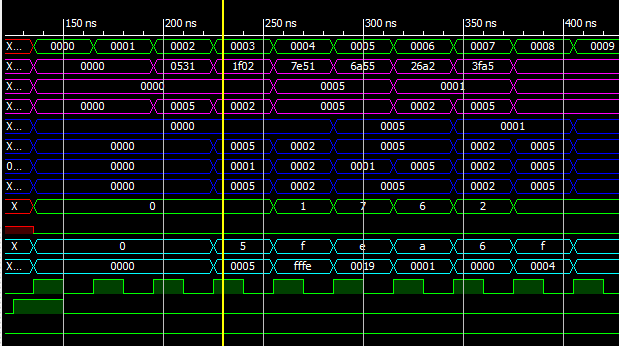
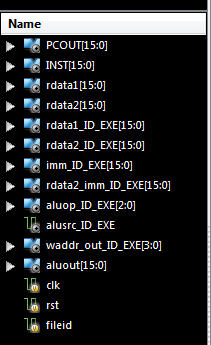
|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x0001 * Instruction fetched=0x0531 | NA | NA |

**SECOND CLOCK CYCLE:**

|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x0002 * Instruction Fetched=0x1F02 | * **INST** 0x0531 is decoded as: reg5=reg3+reg1 * **Rdata1** stores 0 [from register 3] * **Rdata2** stores 5 [from register 1] * **Alu\_Op** is set to 0000 for add * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. |  |

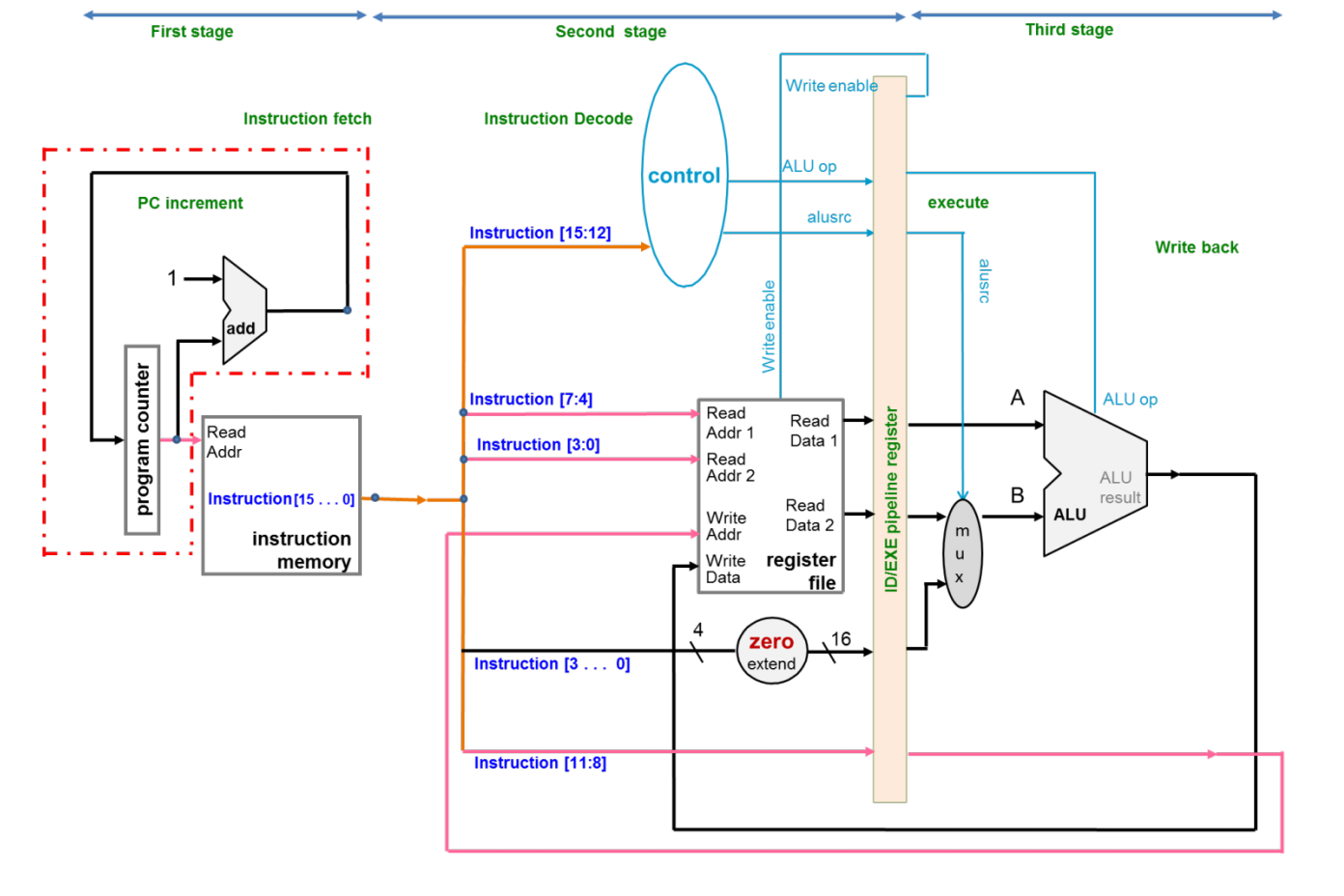
**THIRD CLOCK CYCLE:**

|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x0003 * Instruction Fetched=0x7E51 | * **INST** 0x1F02 is decoded as:   regF=reg0-reg2   * **Rdata1** stores 0 [from register 0] * **Rdata2** stores 2 [from register 2] * **Alu\_Op** is set to 1 for SUB * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. | reg5=reg3+reg1   * Rdata1\_ID\_EXE=0 * Rdata2\_ID\_EXE=5 * Imm\_ID\_EXE=1 * Rdata2\_imm\_ID\_EXE=5 * Alu\_out=0+5=5 |



*Figure 1. Test bench of a 3-stage pipeline for R-type instructions*

The above figure shows the test bench for 3-stage R type instructions. The first instruction is shown by the yellow area, the second instruction is shown by the red area and third instruction by the white area at different stages of the pipeline. Each instruction is completed in 3 clock cycles, but with the help of pipeline, the throughput can be increased i.e.the number of instructions completed per unit of time. But it does not reduce the execution time of an individual instruction. In fact, it usually slightly increases the execution time of each instruction due to overhead in the pipeline control. **This is the core function of pipelining**. The RTL diagram below shows a snippet of the pipeline when all three stages are full.

*Figure 2. RTL Diagram for 3-stage pipeline of R-Type instructions*

|  |  |
| --- | --- |
|  | Executing 0531 |
|  | Decoding 1F02 |
|  | Fetching 7E51 |

Legend

0010

0000

0101

0000

0101

0101

1111

0010

0001

7E51

0003

# What modification you made for converting the datapath from R –type to be used for both R& I type? Explain the testbench output for the execution of R & I type instructions for threestage pipelined datapath (clearly show the R and I type instructions in the testbench simulation)

**16-BIT I-TYPE INSTRUCTION FORMAT:**

|  |  |  |  |
| --- | --- | --- | --- |
| Op-code | Rs | Rt | Immediate |
| 4-bit | 4-bit | 4-bit | 4-bit |

The code was modified to sign-extend the immediate value of I-type-

.imm\_in({{12{INST[3]}},INST[3:0]})

The 3-stage pipeline for R&I-type instruction can be seen from figure 3 below. It consists of *Instruction Fetch, Instruction Decode, and Execute Stage.* The following demonstrates an example of the pipeline process.

**FIRST CLOCK CYCLE:**

|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x0008 * Instruction fetched=0x5BD1 | * INST at 0x0007 is being decoded | * INST at 0x0006 is being executed |

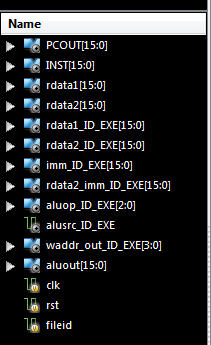
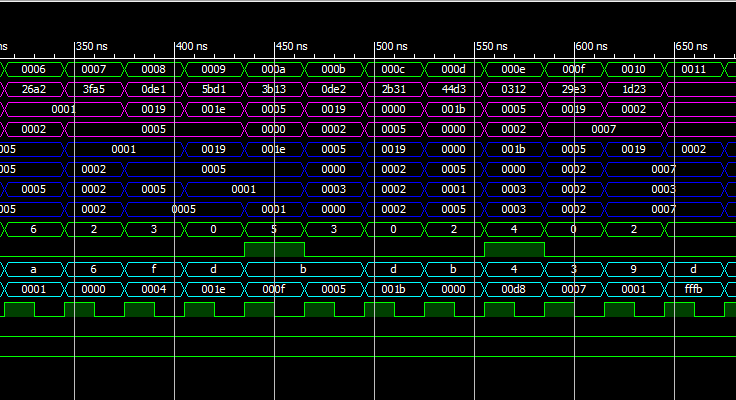
**SECOND CLOCK CYCLE: *(decoding I-type instructions)***

|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x0009 * Instruction Fetched=0x3B13 | * **INST** 0x5BD1 is decoded as:  regB=regD SRL 1 * **Rdata1** stores 0x001E [from register D] * **Rdata2** stores 5 [from register 1] * **Alu\_Op** is set to 0101 for SRL * **Alu\_src** is set to 1 to choose immediate value instead of Rdata2. | * INST 0x0007 is being executed |

**THIRD CLOCK CYCLE:**

|  |  |  |
| --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** |
| * PC increments to 0x000A * Instruction Fetched=0x0DE2 | * **INST** 0x0DE2 is decoded as:   regD=regE+reg2   * **Rdata1** stores 25 [from register E] * **Rdata2** stores 2 [from register 2] * **Alu\_Op** is set to 0001 for ADD * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. | regB=regD SRL 1   * Rdata1\_ID\_EXE=0x001E * Rdata2\_ID\_EXE=5 * Imm\_ID\_EXE=1 * Rdata2\_imm\_ID\_EXE=1 * Alu\_out=000F |

In the third stage when the I-type instruction is decoded, the value of Rdata2\_ID\_EXE is not selected but the immediate value is selected because the Alu\_src is set to 1 by the Control. This is how I-type instructions are decoded. In R-type instructions, the Alu\_src is set to 0. This causes Rdata2\_ID\_EXE to be selected over the immediate value, as shown in the decoding stage of the 000A instruction



*Figure 3. Test bench of a 3-stage pipeline for R&I-type instructions*

The above figure shows the test bench for 3-stage R&I type instructions. The instruction shown in yellow is an I-type instruction. Note how Aluop\_ID\_EXE rises to 1 to indicate the use of immediate values.   
In the above diagram there are two I type instructions since Alu\_Src rises to 1 twice i.2. 5bd1 and 44d3

# Explain the function of four-stage pipelined R & I type datapath with the help of testbench output. (Clearly show the R & I instructions in the test bench simulation for 4-stage pipeline)

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** | **WRITE BACK** |
| * PC increments to 0x0001 * Instruction fetched=0x0531 | NA | NA |  |

The 4-stage pipeline for R&I-type instruction can be seen from figure 4 below. It consists of *Instruction Fetch, Instruction Decode, Execute Stage and write back.* The following demonstrates an example of the pipeline process.

**FIRST CLOCK CYCLE:**

**SECOND CLOCK CYCLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** | **WRITE BACK** |
| * PC increments to 0x0002 * Instruction Fetched=0x1F02 | * **INST** 0x0531 is decoded as: reg5=reg3+reg1 * **Rdata1** stores 0 [from register 3] * **Rdata2** stores 5 [from register 1] * **Alu\_Op** is set to 0000 for add * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. | NA | NA |

**THIRD CLOCK CYCLE:**

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** | **WRITE BACK** |
| * PC increments to 0x0003 * Instruction Fetched=0x7E51 | * **INST** 0x1F02 is decoded as:   regF=reg0-reg2   * **Rdata1** stores 0 [from register 0] * **Rdata2** stores 2 [from register 2] * **Alu\_Op** is set to 1 for SUB * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. | reg5=reg3+reg1   * Rdata1\_ID\_EXE=0 * Rdata2\_ID\_EXE=5 * Imm\_ID\_EXE=1 * Rdata2\_imm\_ID\_EXE=5 * Alu\_out=0+5=5 | NA |

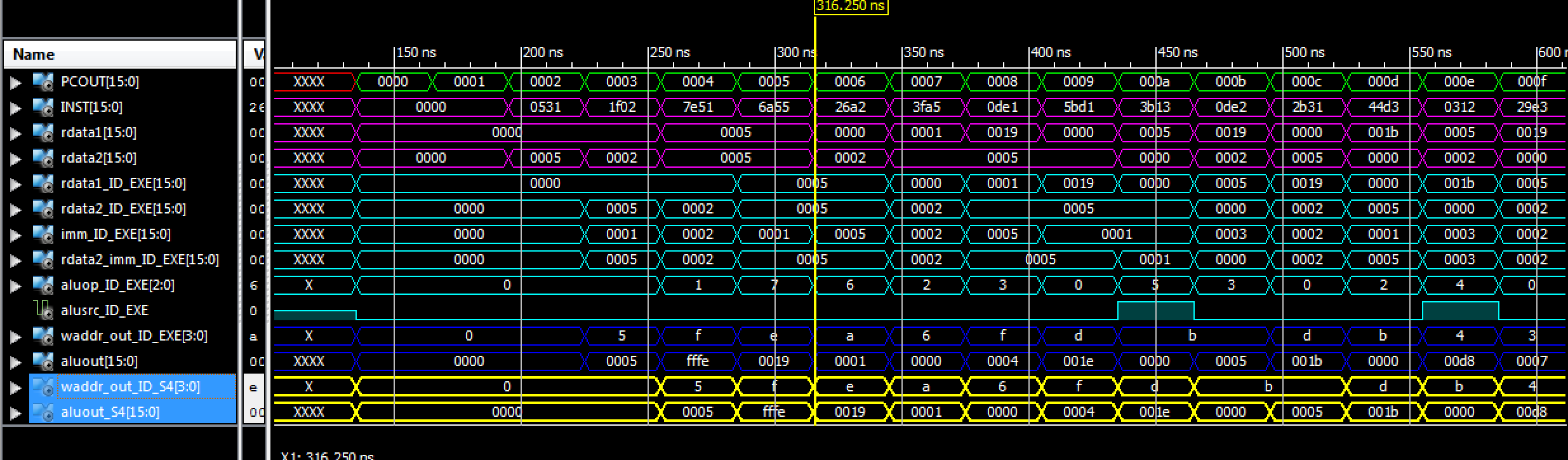
**FOURTH CLOCK CYCLE:**

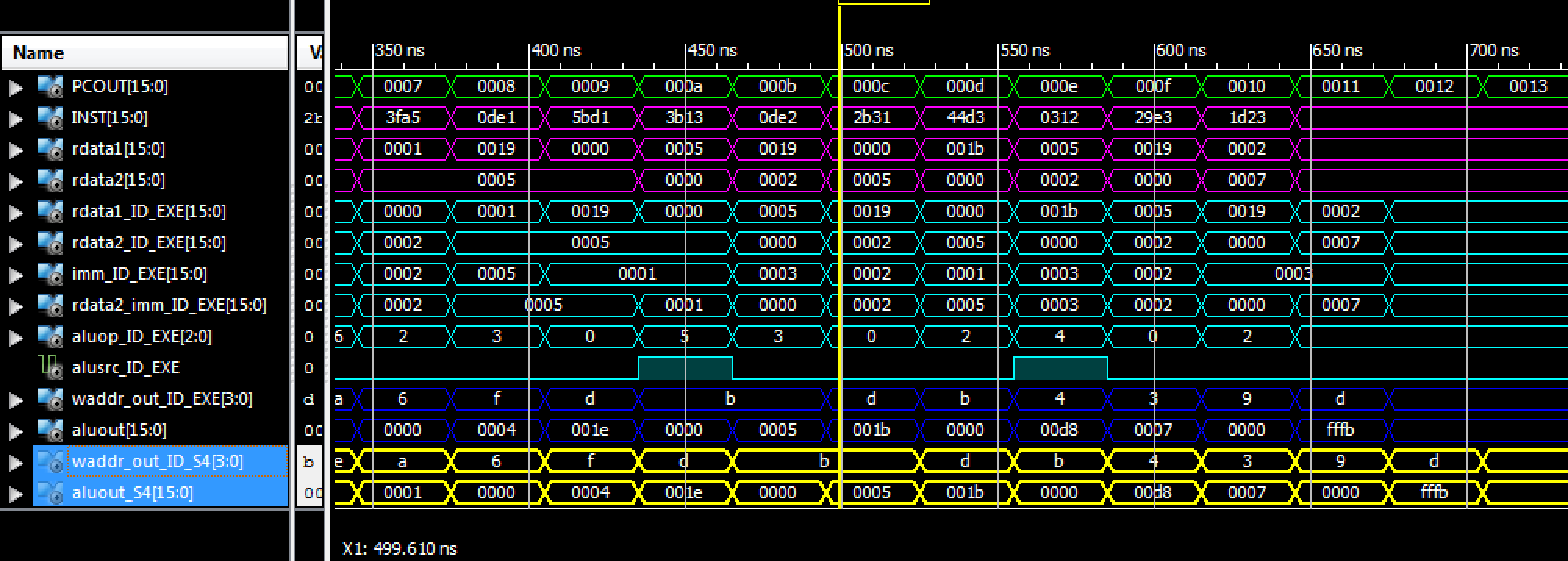
|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH** | **DECODE** | **EXECUTE** | **WRITE BACK** |
| * PC increments to 0x0004 * Instruction Fetched=0x6a555 | * **INST** 0x7E51 is decoded as:   RegE=reg5 x reg1   * **Rdata1** stores 5 [from register 5] * **Rdata2** stores 5 [from register 1] * **Alu\_Op** is set to 0111 for MULT * **Alu\_src** is set to 0 to choose Rdata2 instead of immediate value. | regF=reg0-reg2   * Rdata1\_ID\_EXE=0 * Rdata2\_ID\_EXE=2 * Imm\_ID\_EXE=1 * Rdata2\_imm\_ID\_EXE=2 * Alu\_out=0-2=-2 | Alu\_out\_s4=5 w\_Addr\_out\_ID\_S4=0x5  Register 5 is written with the value of Alu\_out |

Similar procedure can be applied for I-type instructions. Alu\_Src is used to choose between register and immediate values. The figure 4 show the 4-stage pipeline with the help of color-codes as shown in the previous examples.

**Additional Note:** *Often pipelining can lead to data hazards. For e.g. in the previous I-type instruction 0x5BD1 the value stored was 0x000f but here the value stored is 0x0000. This is because, due to an increase in pipeline instruction, the instruction before 0x5BD1 (i.e. 0DE1) is not able to successfully write back and update the value of register D. Similar is the case with instruction 29E3 where the value stores is 0x0000 instead of 0x0001*

*Figure 4. 4-Stage pipeline implementation test bench*





# Synthesize the three-stage and four-stage pipelined R & I type CPU and find the number of slices and minimum period.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CPU | BITWIDTH | No of LUT slices | No of registers | minimum period in ns |
| Three-stage Pipelined  CPU(R and I type)  Implementation | 16 | 642 | 330 | 8.103ns |
| Four-stage Pipelined CPU implementation | 16 | 635 | 348 | 6.404ns |

# Justify the synthesis result.

>>There is an increase in the number of registers used when increased from 3-stage to 4-stage pipeline implementation because we need additional registers to hold the values in an extra stage.

>>There is a deccrease in the number of LUT slices from the 3-stage pipeline to 4-stage pipeline. This depends on the internal optimisation of the verilog code by the Xilinx compiler.

>> When the pipeline stages are increased, the throughput also increases. When the pipeline is increased from 3-stage to a 4-stage, the time consumed for the execution of ALU and write back is now split. Each function can now be completed in lesser time, decreasing our minimum period.