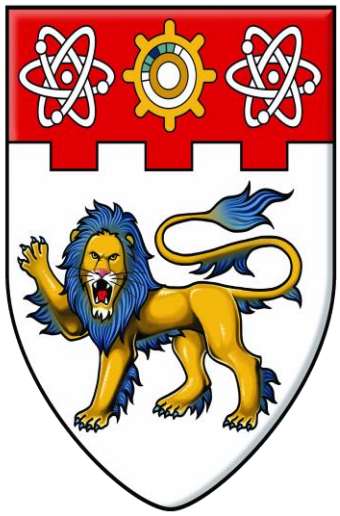


NANYANG TECHNOLOGICAL UNIVERSITY

CZ3001 ADVANCED COMPUTER ARCHITECTURE

Lab 3 Report

Abstract

Pipelining is a process in which successive steps of an instruction sequence are executed in turn by a sequence of modules able to operate concurrently, so that another instruction can be begun before the previous one can finish execution. This report explains the process and implementation of a 3-staged and 4-staged pipelined data path and their synthesis report.

Seshadri Madhavan
Matric. No: U1322790J
Lab Group: SSP3

1. Explain the function of three-stage pipelined data path with test bench for the execution of Rtype instruction with the necessary RTL-block diagram

A typical instruction in a program uses multiple clock cycles (namely, fetch, decode, execute, write back and so on). In a single pipelined system all the actions occur in a sequential manner and take multiple clock cycles to execute and all the circuitry in the processor is also not in use. Thereby to improve the efficiency of the system by fully utilising all the functional units, a multi-staged pipelined data-path helps in full utilization of the functional units. The major advantage of pipelined data path is the fact that multiple instructions are concurrently running in the system. A typical example would be when the 1st instruction is in the execution phase, the 2nd instruction is in the decode phase and the 3rd instruction is in the fetch phase. In a 3-staged pipelined data path, up-to 3 instructions are running concurrently.

Illustration with an Example

16-BIT R-TYPE INSTRUCTION FORMAT:

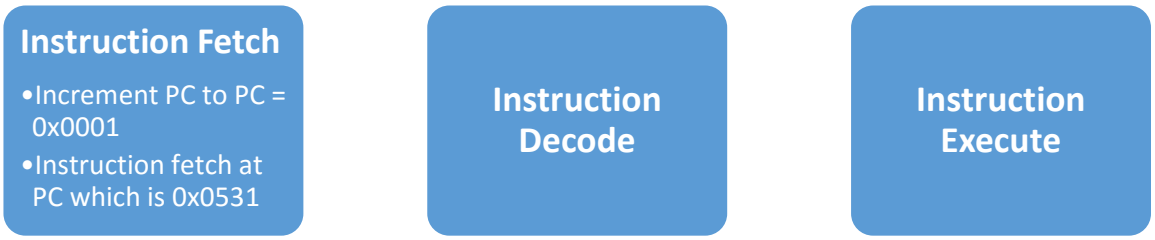
Op-code	Rs	Rt	Rd
4-bit	4-bit	4-bit	4-bit

A three stage pipeline for R type instructions consists of the following stages: namely Instruction Fetch, Instruction Decode and Instruction Execute. Let us a see a typical example by using the first three instructions from the test bench.

INSTRUCTIONS

1. 0x0531 (Perform operation with operation ID 0x0000 and the destination register 0x0005 and source registers 0x0003 and 0x0001)
2. 0x1F02 (Perform operation with opcode 0x0001 with the destination register 0x00F and the source registers 0x000 and 0x002)
3. 0x7E51(Perform operation with opcode 0x0007 with the destination register 0x00E and the source registers 0x005 and 0x001)

FIRST CLOCK CYCLE



SECOND CLOCK CYCLE

Instruction Fetch

- Increment PC to PC = 0x0002
- Instruction fetch at PC which is 0x1F02

Instruction Decode

- INST 0x0531 is decoded as reg5 = reg3+reg1
- RData1 reads 0 from register 3
- Rdata2 reads 5 from register 1
- ALU_op is set to 0x0000 for add operation

Instruction Execute

THIRD CLOCK CYCLE

Instruction Fetch

- Increment PC to PC = 0x0003
- Instruction fetch at PC which is 0x7E51

Instruction Decode

- INST 0x0531 is decoded as regF = reg0-reg2
- RData1 reads 0 from register 0
- Rdata2 reads 2 from register 2
- ALU_op is set to be set 0x0001 for SUB operation

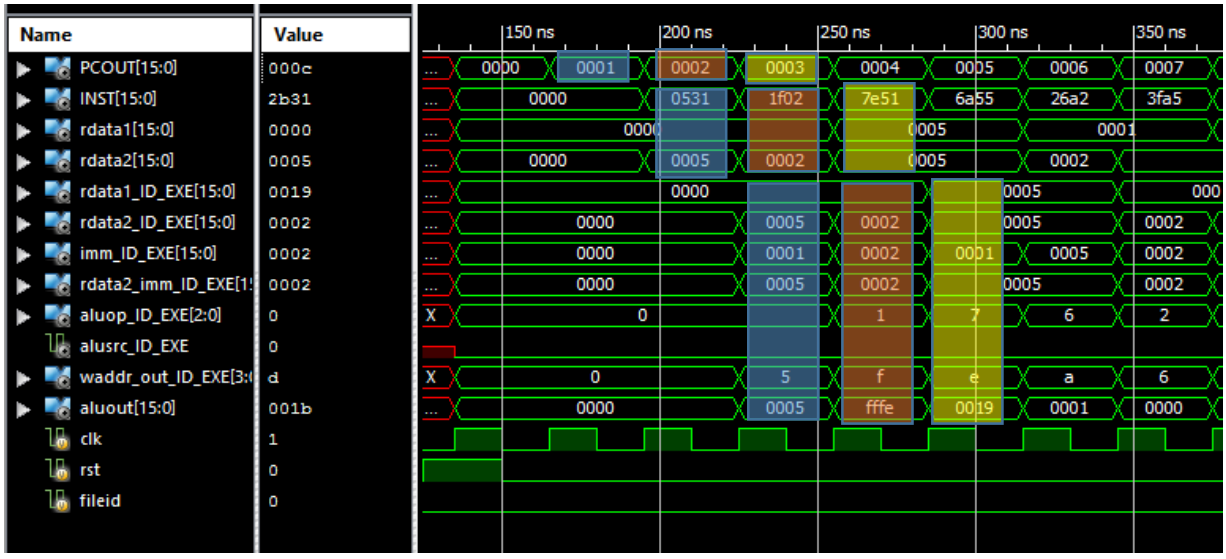
Instruction Execute

- **reg5 = reg3+reg1**
- **ALU_out = 5**
- **Imm_ID_EXE = 1**

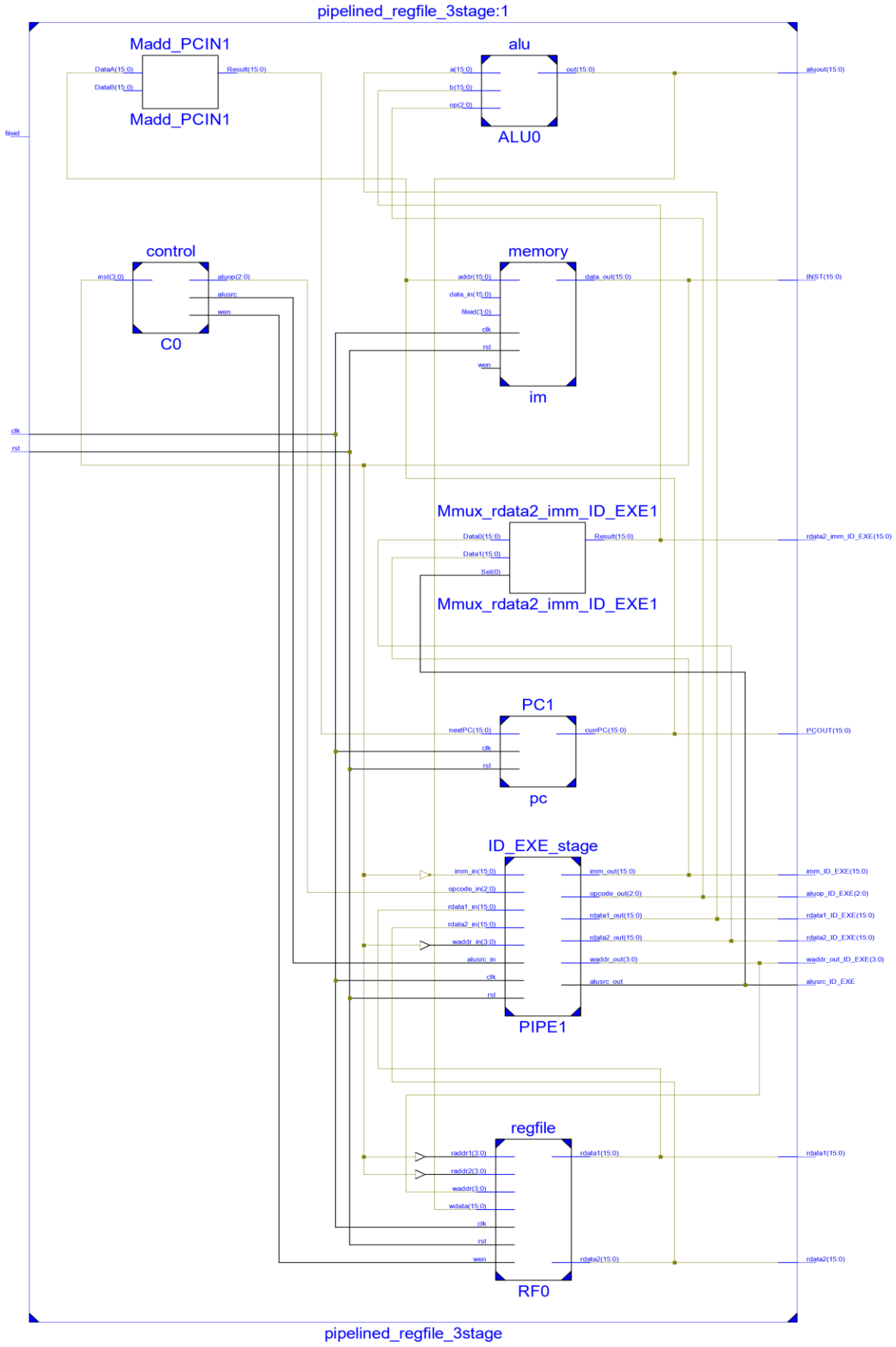
Test Bench Explanation

The simulation figure illustrates the instruction FETCH, DECODE and EXECUTE as explained in the above process. In the first Clock cycle the first instruction is fetched from memory. In the second clock cycle, the second instruction is fetched from memory and the first instruction is decoded. In the third clock cycle, the first instruction is executed while the second instruction is decoded and the third instruction is fetched and values in test bench confirm the same.

The first instruction is shaded by BLUE colour, second by Orange colour and third by Yellow colour.



RTL Diagrams for R type instructions



2. What modification you made for converting the data path from R –type to be used for both R& I type? Explain the test bench output for the execution of R & I type instructions for three-stage pipelined data path

To convert the data path to be able to execute both R&I type instructions and not just R type instructions we need to add sign extension for the immediate value. This is done by adding the following code for sign extension,

.imm_in({{12{INST[3]}},INST[3:0]})

The following explains the demonstration of the pipeline:

FIRST CLOCK CYCLE

Instruction Fetch
- Increment PC to PC = 0x0008
- Instruction fetch at PC which is 0x5BD1

Instruction Decode
- Instruction at 0x0007 is being decoded

Instruction Execute
- Instruction at 0x0006 is being executed

SECOND CLOCK CYCLE

Instruction Fetch
- Increment PC to PC = 0x0009
- Instruction fetch at PC which is 0x3B13

Instruction Decode
- INST 0x5BD1 is decoded as regB = regD SRL 1
- RData1 reads 0x001E from register D
- Rdata2 reads 0x005 from register 1
- ALU_op is set to 0x0101 for the operation SRL

Instruction Execute
- INST at 0x0007 is being executed

THIRD CLOCK CYCLE

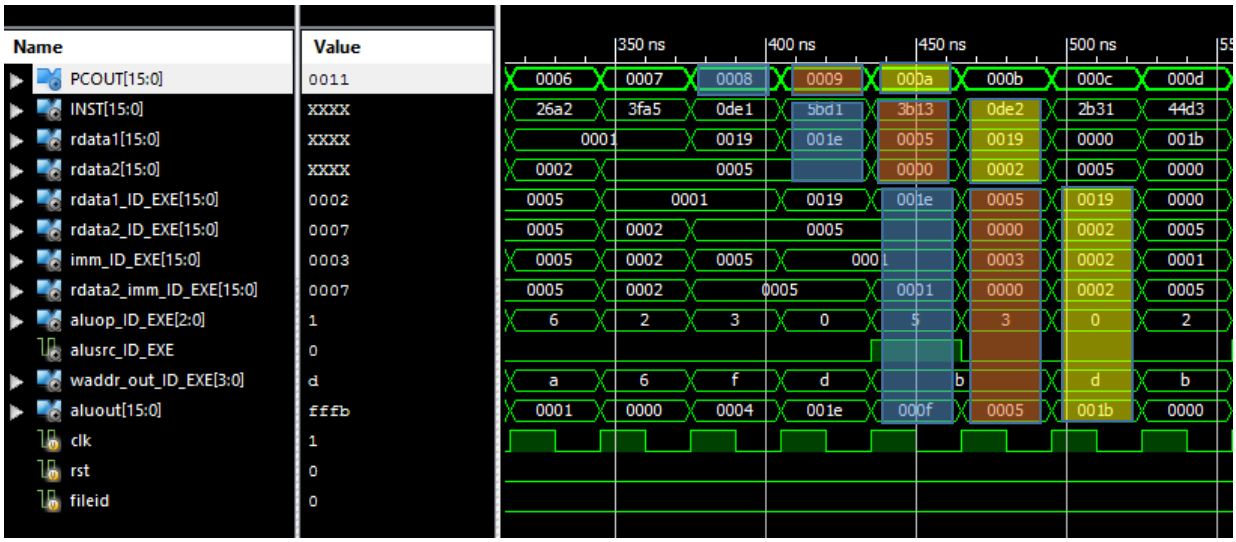
Instruction Fetch
- Increment PC to PC = 0x000A
- Instruction fetch at PC which is 0x0DE2

Instruction Decode
- INST 0x2B13 is decoded as regB = reg1 SRL 1
- RData1 reads 0x001E from register 1
- Rdata2 reads 0x005 from register 3
- ALU_op is set to 0x0101 for the operation SRL

Instruction Execute
- INST at 0x0008 is being executed
- Rdata1_ID_EXE=0x001E
- Rdata2_ID_EXE=5
- Imm_ID_EXE=1
- Rdata2_imm_ID_EXE=1
- Alu_out=000F

In the third clock cycle, the execution phase of the instruction 0x008 is ongoing. The value of Rdata2_ID_EXE is not selected but the immediate value is selected because it is, I type

instruction and ALU_src is set to 1 by the control of the processor. In the R type instructions the value of ALU_src is set to 0 so the value present in the Rdata2_ID_EXE is used instead of the value from the immediate register.



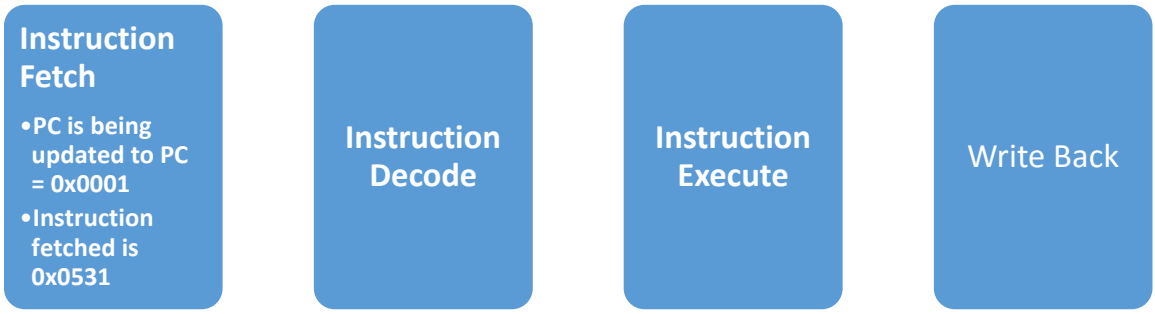
In the above figure, the instruction 0x0008 is I (which is highlighted by the blue shading) type instruction and 0x009 and 0x000a are the R type instructions in the current instruction set (which is highlighted by the orange and yellow shading). We can observe that the alusrc_ID_EXE changes to 1 to show that the data to be used is the immediate data provided in the instruction and not the Rdata2.

3. Explain the function of four-stage pipelined R & I type data path with the help of test bench output.

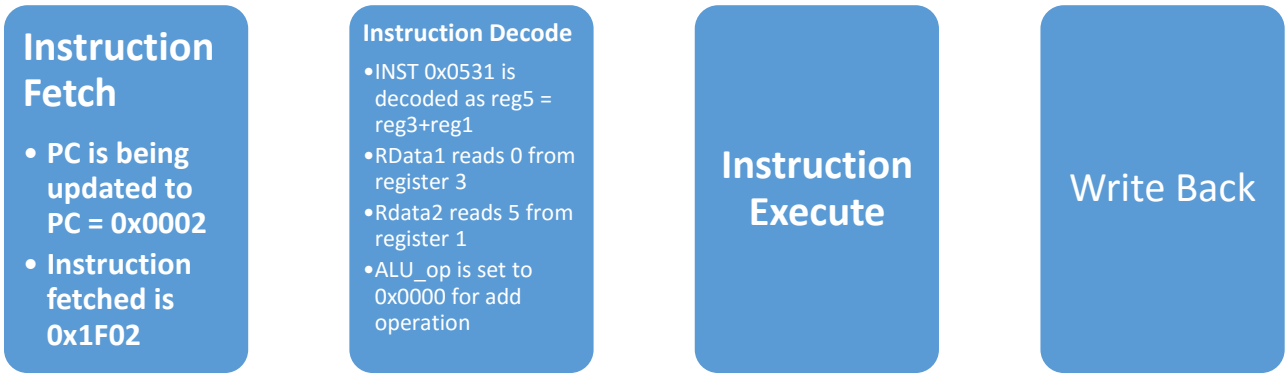
There are 4 stages in a four staged pipeline namely Fetch Instruction, Decode Instruction, Execute Instruction and Write Back the result if the write enabled signal is high. In this lab experiment, write enabled is always high. In a four staged pipelined architecture, up to 4 instructions can be executed simultaneously in processor.

When the first instruction is writing back the result, the second instruction is executing, the third instruction is in the decoding stage and fourth instruction is in the fetching stage. Thus the four instructions can be executed simultaneously.

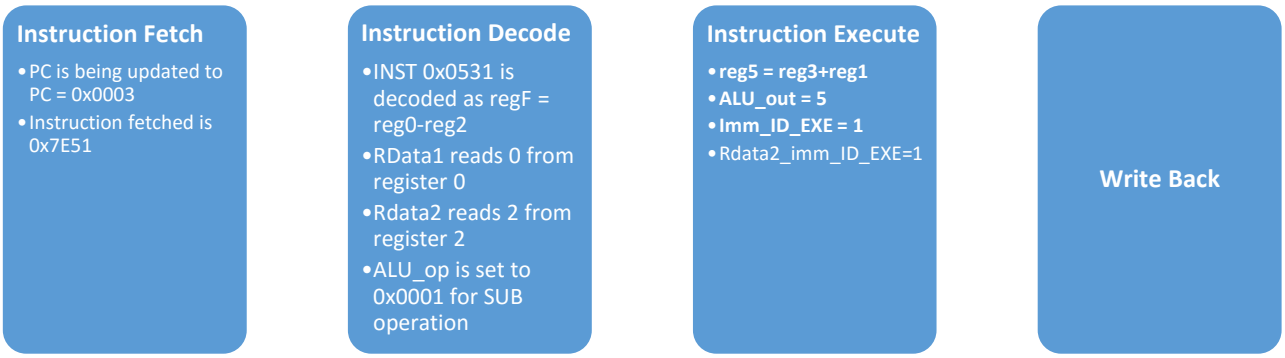
FIRST CLOCK CYCLE



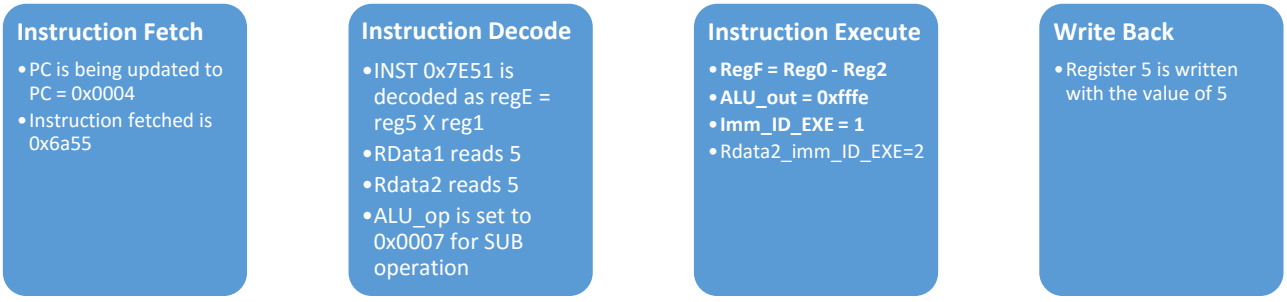
SECOND CLOCK CYCLE



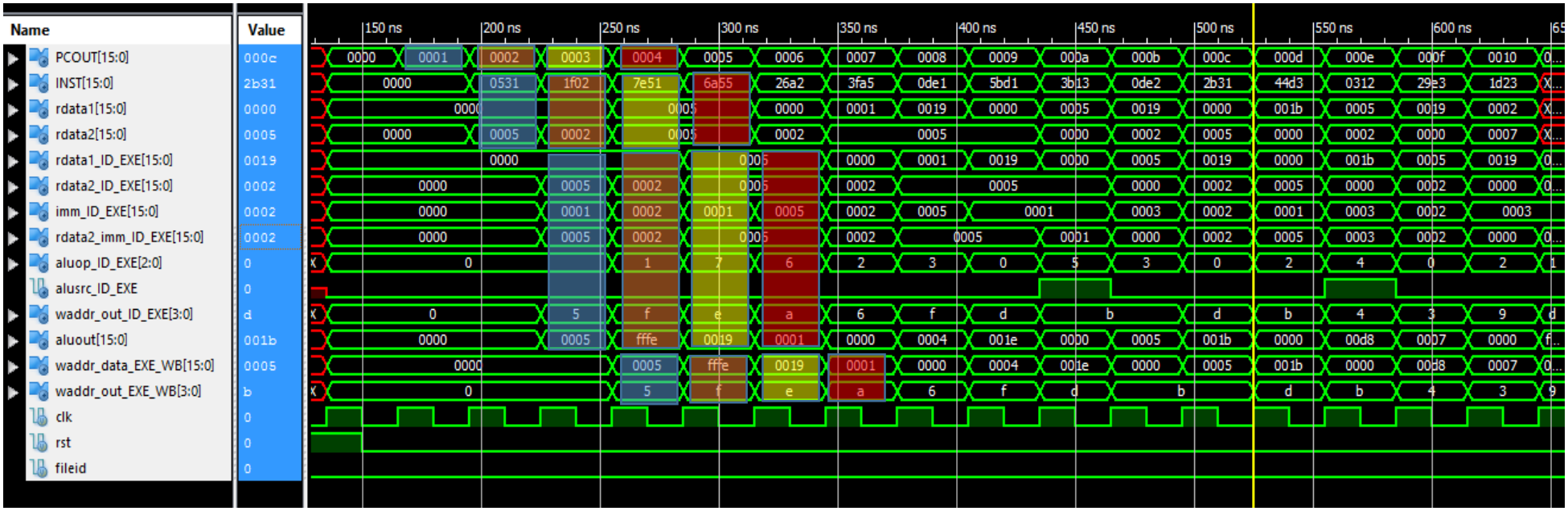
THIRD CLOCK CYCLE



FOURTH CLOCK CYCLE



The example above illustrates the stages in the four-staged pipeline process in which up to four instructions can be executed at any point of time. Just as illustrated above, when the first instruction is writing back result into memory, the second instruction is Executing, the third instruction is Decoding and the fourth instruction is in the Fetching stage.



The above figure illustrates the test bench simulation of the four staged pipelined data path. When the first instruction is writing back result into memory, the second instruction is Executing, the third instruction is Decoding and the fourth instruction is in the Fetching stage.

4. 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

5. Justify the synthesis result

From the above synthesis result it can be observed that the number of registers increases from the three staged to four staged pipeline as more register are required to hold the values to be used to store in the additional stage of the pipeline.

It can also be observed that the number of LUT slices actually decreases from the 3 staged to the four staged pipelined implementation (from 642 to 635) because Xilinx has different optimization process for each of the individual circuits. In an ideal scenario, the number of LUT slices are supposed to increase when we increase the number of stages by one, in the pipelined data path because more circuitry is required to implement the additional stage in the pipelined data path.