****

**LAB GROUP: SS1**

**CZ3001:**

**ADVANCED COMPUTER ARCHITECTURE**

***Lab 4 (Brief Report)***

by

Alvin Lee Yong Teck (U1620768F)

Dr Kavallur Pisharath Gopi Smitha

A/P Anupam Chattopadhyay

[smitha@ntu.edu.sg](mailto:SMITHA@NTU.EDU.SG)

[anupam@ntu.edu.sg](mailto:anupam@ntu.edu.sg)

**Date of Submission: \_\_\_\_\_\_\_\_\_\_\_\_**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**NANYANG TECHNOLOGICAL UNIVERSITY**

**CE/CZ3001- Lab4 (Brief report)**

**Submission: Please submit the report when you come for Lab 5 (approx.: two weeks’ time)**

Write the MIPS assembly code for the computation of “e = (a+b)^(c\*d) ”. Note that all variables are integers. The addresses and data of the variables are given in table below. You can load this data to data memory.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data | a=A | b=A | c=A | d=A | e=A |
| Addresses | 0x00000002 | 0x00000003 | 0x00000004 | 0x00000005 | 0x00000006 |

**^ indicates XOR operation**

**\* Indicates multiply operation**

**+ indicates addition operation**

1. Write the MIPS assembly code for the computation of “e = (a+b)^(c\*d)” with minimum number of instructions.

**LW $4, 2($0) # load a to $4**

**LW $5, 3($0) # load b to $5**

**ADD $5, $4, $5 # $5 = $4 + $5**

**LW $4, 4($0) # load c to $4**

**LW $6, 5($0) # load d to $6**

**MUL $4, $6, $4 # $4 = $6 \* $4**

**XOR $4, $5, $4 # $4 = $5 ^ $4**

**SW $4, 6($0) # store result from $4 to e**

1. Modify the MIPS assembly code for the computation of “e = (a+b)^(c\*d)” for a five stage pipelined architecture given in lab 4, after including NOPs for removing data-dependencies.

**\*RAW Dependencies**

**LW $4, 2($0) # load a to $4**

**LW $5, 3($0) # load b to $5**

**NOP**

**NOP**

**ADD $5, $4, $5 # $5 = $4 + $5**

**LW $4, 4($0) # load c to $4**

**LW $6, 5($0) # load d to $6**

**NOP**

**NOP**

**MUL $4, $6, $4 # $4 = $6 \* $4**

**NOP**

**NOP**

**XOR $4, $5, $4 # $4 = $5 ^ $4**

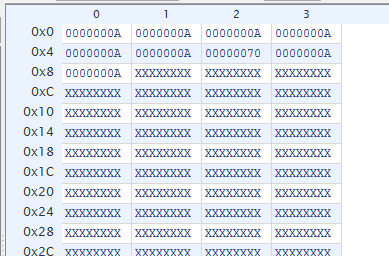
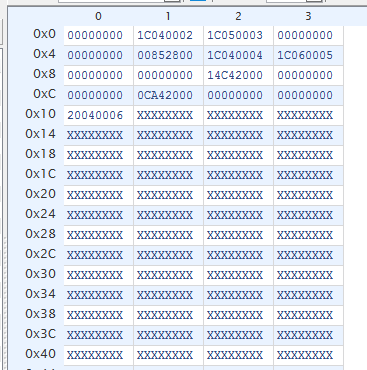
**NOP**

**NOP**

**SW $4, 6($0) # store result from $4 to e**

1. Show the snapshot of instruction and data-memory (all values in hexadecimal) from the ISIM simulation window.

**Data Memory Instruction Memory**

1. Explain the working of the five stage pipeline both for LW and SW instruction (used in this code) using ISIM window as reference.

**Load Word (LW) Instruction**

Example: **LW $4, 2($0)**

**\*Assume that the fetch stage and decode stage takes one cycle each.**

**(1) Fetch** : The LW instruction is fetch after the Program Counter (PC) provides the address 0x00000001 to the instruction memory.

**(2) Decode** : The 6-bit opcode was decoded from the instruction to generate the control signals. The source (+ offset) and destination registers are also decoded and pass in the value (**rdata1**) to the Register File. Afterwards, it will then pass the required values (**imm**, **alusrc** & **waddr**) to the ID\_EXE pipeline register.

\***rdata1** is the address 0x00000000 (the first source for the ALU)

**\*imm** contains the value 0x00000002 (sign-extended immediate value)

\***waddr** contains the value 0x04 (the address to write back to register $4)

\***alusrc** contains a one-bit value “1” (the signal is asserted, i.e. the second source for the ALU will be taken from **imm** instead of **rdata2**)

**(3) Execute** : ALU executes the **ADD** operation for the two sources (**rdata1** & **imm**) and produces the result to **aluout**. The values (**aluout** & **waddr**) will then be passed on to the EXE\_MEM pipeline register.

\***aluout** contains the value 0x00000002

\***waddr** contains the value 0x04 (from **(2) Decode**)

**(4) Memory** : The **aluout** which contains the address 0x00000002 will be used to read the content from the data memory. The value 0x0000000a is then retrieved out from the data memory. The **output of DMEM** is not passed on to the MEM/WB pipeline register, and only the **waddr** is being passed on to the MEM\_WB pipeline register.

\***waddr** contains the value 0x04 (from **(3) Execute**)

**(5) Write Back** : The ALU output or the data memory output is chosen with the help of the control signal. Hence, **wdata** now holds the value 0x0000000a. It is then written back to the register file using the **waddr** 0x04. The register $4 is then loaded with the value 0x0000000a.

**Store Word (SW) Instruction**

Example: **SW $4, 6($0)**

**\*Assume that the fetch stage and decode stage takes one cycle each.**

**(1) Fetch** : The SW instruction is fetch after the PC provides the address 0x00000010 to the instruction memory.

**(2) Decode** : The 6-bit opcode was decoded from the instruction to generate the control signals. The source (+ offset) and destination registers are also decoded and pass in the values (**rdata1** & **rdata2**) to the Register File. Afterwards, it will then pass the required values (**imm**, **rdata2** & **alusrc**) to the ID\_EXE pipeline register.

\***rdata1** is the address 0x00000000 (the first source for the ALU)

\***rdata2** is the data 0x00000070 (the content from $4) (It will serve as the write data for Data Memory)

**\*imm** contains the value 0x00000006 (sign-extended immediate value)

\***alusrc** contains a one-bit value “1” (the signal is asserted, i.e. the second source for the ALU will be taken from **imm** instead of **rdata2**)

**(3) Execute** : ALU executes the **ADD** operation for the two sources (**rdata1** & **imm**) and produces the result to **aluout**. The values (**aluout** & **rdata2**) will then be passed on to the EXE\_MEM pipeline register.

\***aluout** contains the value 0x00000006

\***rdata2** is the data 0x00000070 (from **(2) Decode**)

**(4) Memory** : The data 0x00000070 will be written to memory location 0x00000006 (**aluout**). Thus the required result is successfully stored there eventually.

**(5) Write Back** : For SW instruction, there is not a need to write back to the register file.

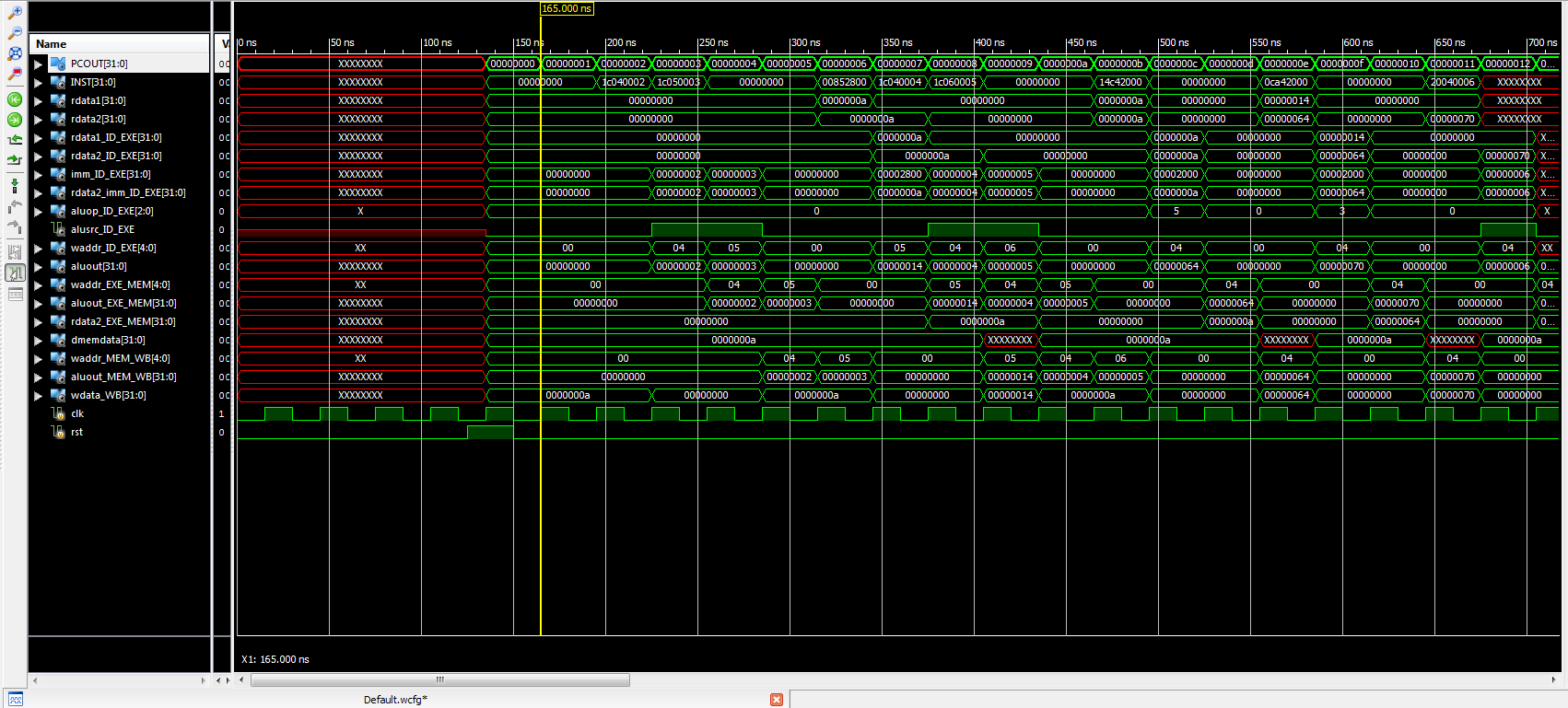
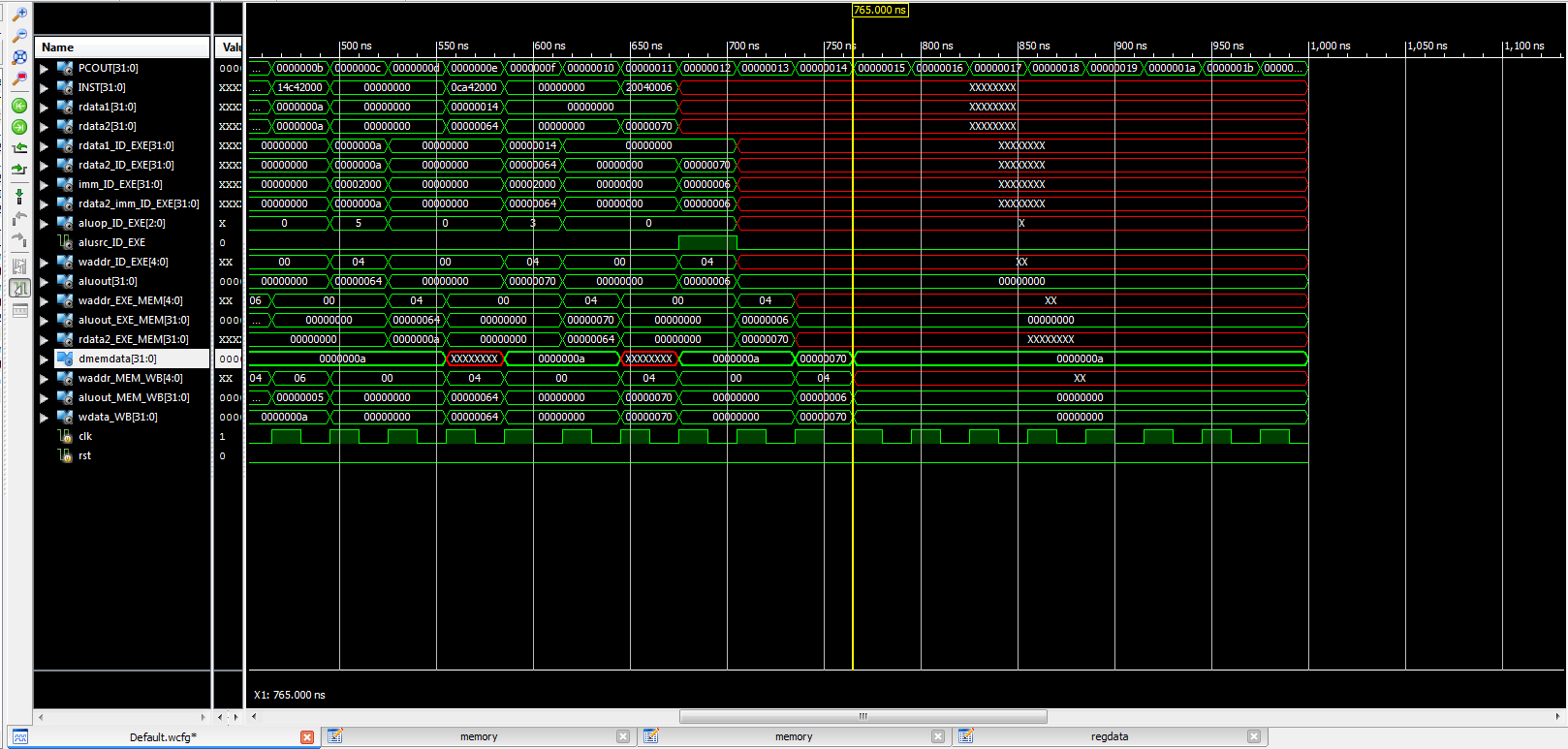
1. Indicate the execution time for running this program along with a snapshot of starting and ending time of the code in the ISIM simulator.

**Execution Time** = Ending Time – Starting Time

= 765ns – 165ns

= **600ns**

**Starting Time** **Ending Time**

 ****

1. Calculate the steady state CPI of the code while running in a five stage pipelined architecture.