Lab 4 Report

CZ3001 – Advanced Computer Architecture

Name : Xavier Tan

Matriculation Number : U1720697B

Lab group: SSP2

Brief report for 5 stage pipeline.

A) Write the MIPS Assembly code for the computation d = (a+b)*(b-c)

00000000 00000000 //NOP at the start to clear previous instructions just incase
00000001 1C040001 //LW $4, 1($0);
00000002 1C050002 //LW $5, 2($0);
00000003 1C060003 //LW $6, 3($0);
00000004 00855000 //ADD $10, $4, $5;
00000005 04A65800 //SUB $11, $5, $6;
00000006 154B5000 //MUL $10, $10, $11;
00000007 200A0004 //SW $10, 4($0);

No Stalls or NOP instruction is added here yet for the 5 stage pipelining architecture. This gives the minimum number of instruction, as

B) Instructions after the including NOPs to take care of the data dependencies found.

There is data hazard due to data dependencies between the instructions

1. Identify the data dependencies

Instruction 2 (LW $5, 2($0)) and Instruction 4 (ADD $10, $4, $5) has a read after write dependency, as instruction 4 needs to read register 5 after instruction 2 loads the data in address 0x00000002 into the same register, 5. So instruction 4's decode stage need to be at the write back stage's cycle of instruction 2. Hence NOP is added in between. There is also dependency between instruction 1 and 4. However, instead of adding no operations(not), we can reorder the arrangement of the 2 other load word so 2 clock cycle would have passed when instruction 4 tries to get the data from register $4. This reduces the number of NOP needed, hence optimising the code.

There is also data dependency between instruction 6(MUL $10, $10, $11), and 4(ADD $10, $4, $5)/5(SUB $11, $5, $6). Both dependency arises from read after write(RAW). As instruction 6 would need to read data from registers thats instruction 4 and 5 will write to. The data will only be correct after the execution stage hence the decode stage of instruction 6 have to be the cycle of that stage. So to simulate hardware stall, NOP instructions is given.

2 NOP is also needed before Store word instruction as there is read after write data dependency between instruction 6 and 7. SW $10, 4($0); needs to read data from register $10 which is also used by MUL instruction for write back. So the decode stage of instruction 7 needs to be at the same cycle as write back cycle of instruction 6.

This results in the instruction below. The first NOP is added just incase if there is any remaining instruction left behind from instructions before.

```
00000000 00000000 // NOP
00000001 1C040001 // LW $4, 1($0); load a
00000002 1C050002 // LW $5, 2($0); load b
00000003 1c060003 // LW $6, 3($0); load c
00000004 00000000 // NOP
00000005 00855000 // ADD $10, $4, $5; a + b put in $10
00000006 04A65800 // SUB $11, $5, $6; b-c put in a temp $11
00000007 00000000 // NOP
00000008 00000000 // NOP
00000009 154B5000 // MUL $10, $10, $11; (a+b)*(b-c) store $10
0000000A 00000000 // NOP
0000000B 00000000 // NOP
0000000C 200A0004 // SW $10, 4($0); Store value of $10 to 0x00000004
```

C) Snap shot of instruction memory and data memory

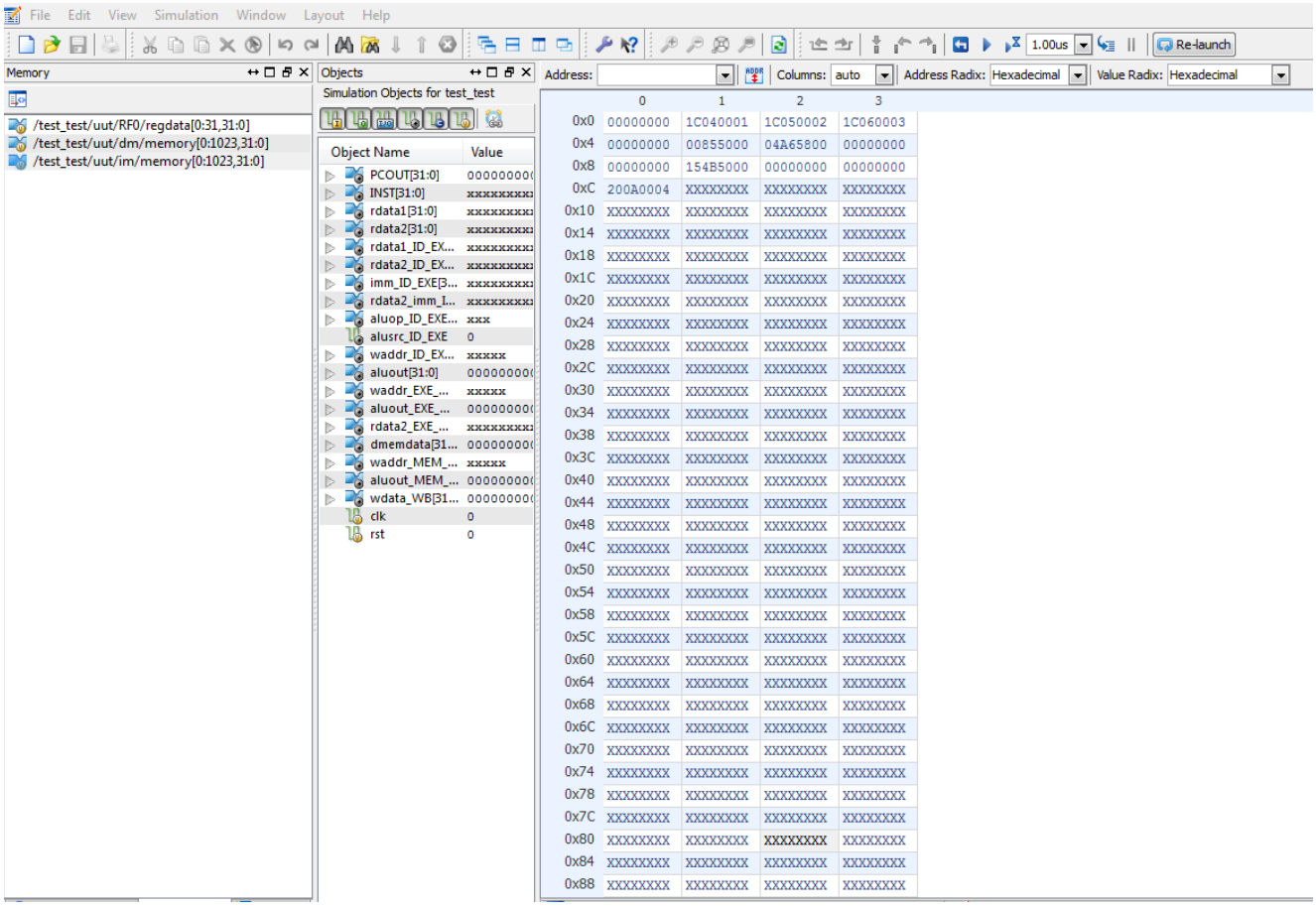

Figure 1. Instruction memory in hexadecimal

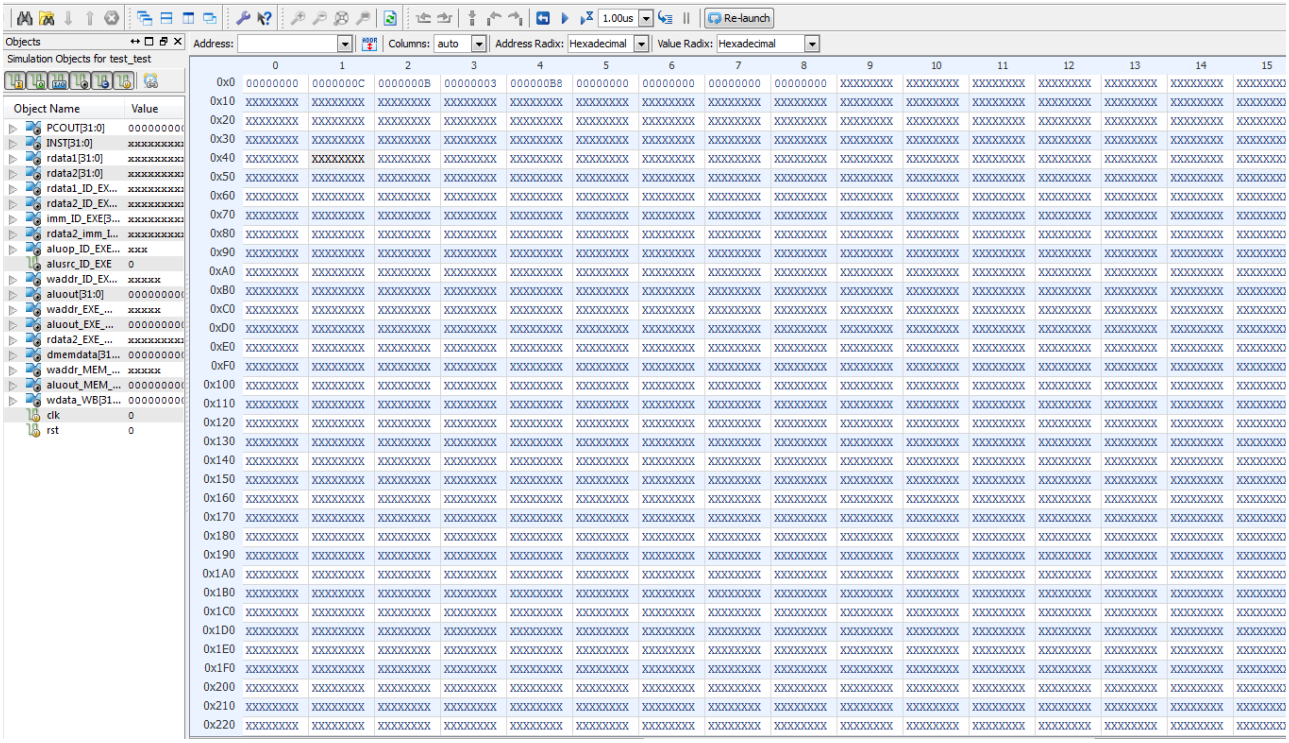

Figure 2. Decimal Memory in hexadecimal

D) Explanation five stage pipeline for LW and SW.

5 stage pipeline => Fetch Decode Execute Memory Write-Back

Load word instruction: Using the first load instruction as reference, LW $4, 1($0); load a.

1) Fetch : CPU fetches instruction from the memory address that the program counter (PC) points to and the corresponding instruction stored in the instruction memory(IMEM).

2) Decode : The CPU will decode the 6 bit opcode i.e 1C040001. This will generate the control signals like write enable(wen), ALUSrc for the other components like ALU, register file (regfile), Data Memory. This some of these control signals that is only needed in later stage will be propagated through the pipeline registers. For example the **memwrite_cntrl** = 0 (LW don't need to write to data memory file) which is generated in the second cycle will be propagated through ID/EXE pipeline register, until EXE/MEM register and then passed to the memory. The source and destination registers will also be decoded and passed into regFile. Within reg file, it will fetch the data from the respective registers and output it to the ID/EXE pipeline registers. So that another instruction can occupy this part of the hardware while LW instruction continues to the execution part. In ISIM data **rdata1** (read data 1)will be the first register source $rs, **rdata2** (read data 2) will be the second register source, $rt. However, because this is a load word instruction is takes the source from **rdata2_imm** instead, contains the value 0x00000001(sign extended). **Waddr_EXE_MEM** = 0x04, which is the address to write back for register $4. **Alusrc_cntrl** = 1, the signal is asserted as seen in ISIM window.

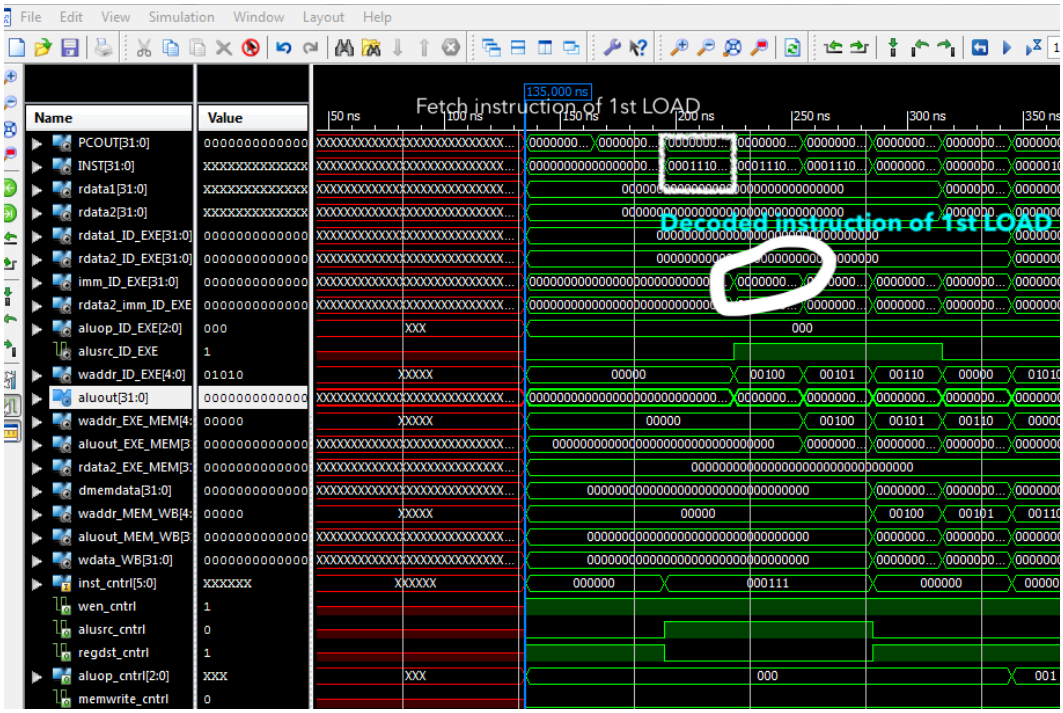

Figure 3. LW instruction in ISIM window

3) Execute: The ALU will execute an ADD operation on the two sources, rdata1 and imm, i.e. 0x00000001 + 0x00000000 = 0x00000001(address of variable a) . The result will be stored in EXE/MEM Pipeline register.

4) Memory: The aluout which contains the address to read the content from the data memory. In our case, the value retrieved from DMEM (0x0000000C) is not taken through MEM/WB pipeline register, only MemToReg signal which is **waddr_MEM_WB** is passed on. waddr should contain value 0x04 for this instruction.

5) The **wdata_WB** = 0x0000000C now and should be written back to the register file $4 since **waddr** = 0x04. The **wdata_WB** is chosen between ALU result or the read data by the multiplexer(MUX) with the value given by control unit. Thus now register $4 has value 0x0000000C.

Store word instruction: Using the last store instruction as reference, SW $10, 4($0)

1) Fetch : CPU fetches instruction from the memory address that the program counter (PC) points to and the corresponding instruction stored in the instruction memory(IMEM).

2) Decode : The CPU will decode the 6 bit opcode i.e 200A0004. Like before, this will generate the corresponding control signals. Now the **wen_cntrl** will be = 1, **alusrc** = 1. The source and destination registers are also decoded. The immediate data in **imm_ID_EXE** will be sign extended to 0x00000004, **rdata1** = 0x00000000 since it is reading from $0. The **rdata2** will read from register $10(0X0A). The rdata2 and and immediate number is will be channeled correctly due to the multiplexer between IMEM and regFile, and **regdst_cntrl** = 0. These values are then passed to the ID/EXE pipeline register for the instruction to use in the next clock cycle.

3) Execute: ALU will execute the ADD operation on the two sources **rdata1** and imm i.e 0x00000004 + 0x00000000 = 0x00000004(address of variable d). This value will be stored in EXE/MEM pipeline register.

4) Memory: The data from waddr_MEM_WB = 0x0A, register $10 will be read. This data will then be stored to address 0x00000004 as the control signal for **memwrite_cntrl** = 1.

5) Write Back: No need to write back to register file in SW, as data is stored in data memory.

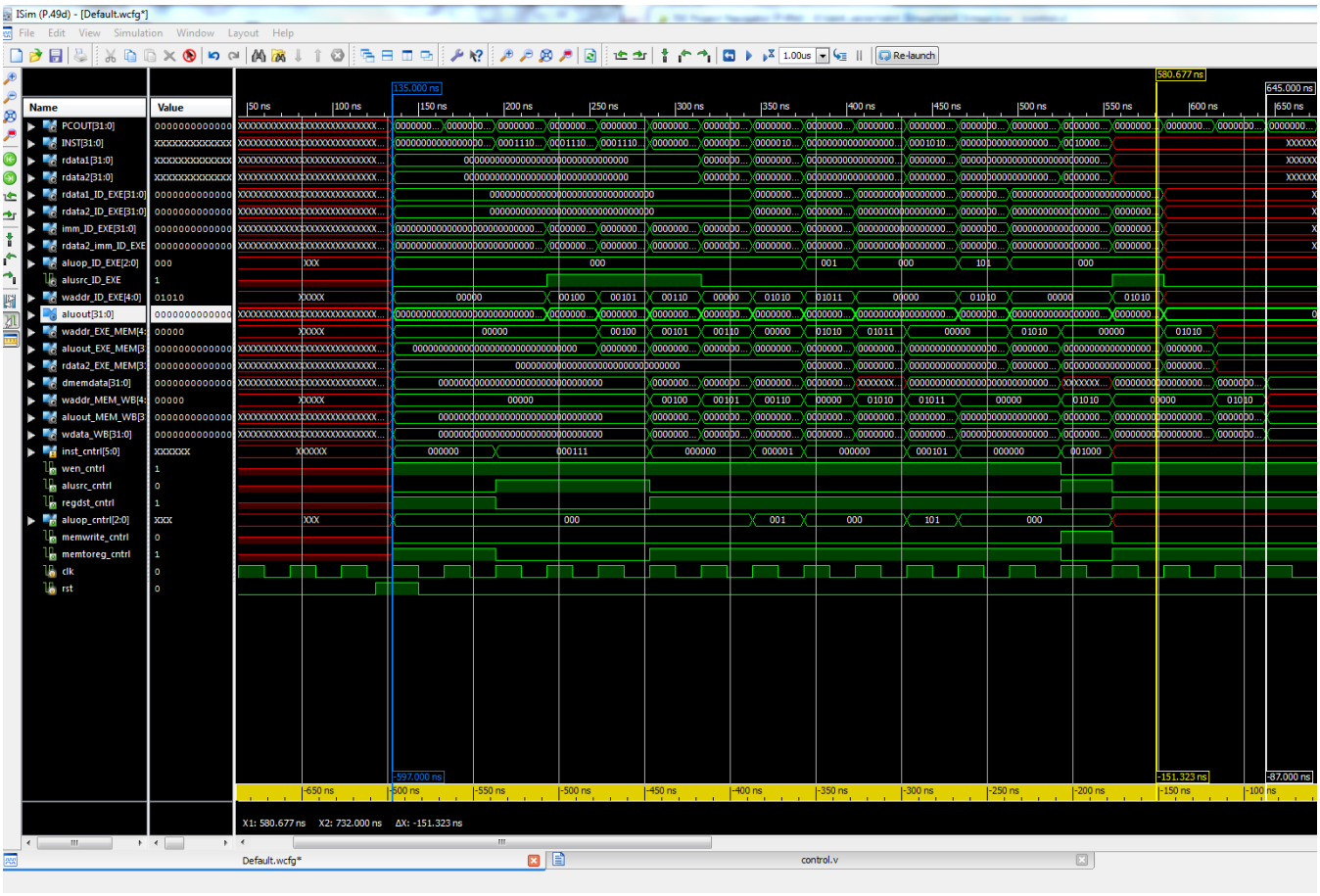

Figure 4. Execution time

E) Execution time :

End of the execution is defined as after the store word instruction finishes storing in the data memory. Including the addition NOP before the first LW instruction.

The starting time is **135.000ns** and ends at **645.000ns.**

As seen from the two markers in figure 4. Hence the **Execution time = 510.0 ns.**

If excluding the 1st NOP which will not affect any instructions.
Starting time **165.00ns** and ends at **645.00ns**. **Execution time = 480.0ns.**

Execution time = CPI * number of instruction * clock period

Practical CPI = 480/(12*9.126) = 4.383

F) Steady state CPI

Steady State CPI = (number of stalls + number of instructions) / number of instructions
= (5 + 7)/ 7 =1.714285*
*this number of stalls excludes the first stall before the load instruction.

G)Conclusion.

Theoretical CPI is way lesser that the real CPI. This is probably because of the high propagation delay from storing in the pipeline registers hence the execution time is higher from the ISIM window is higher then what the formula would calculate. Therefore the theoretical CPI is way lower than practical.