

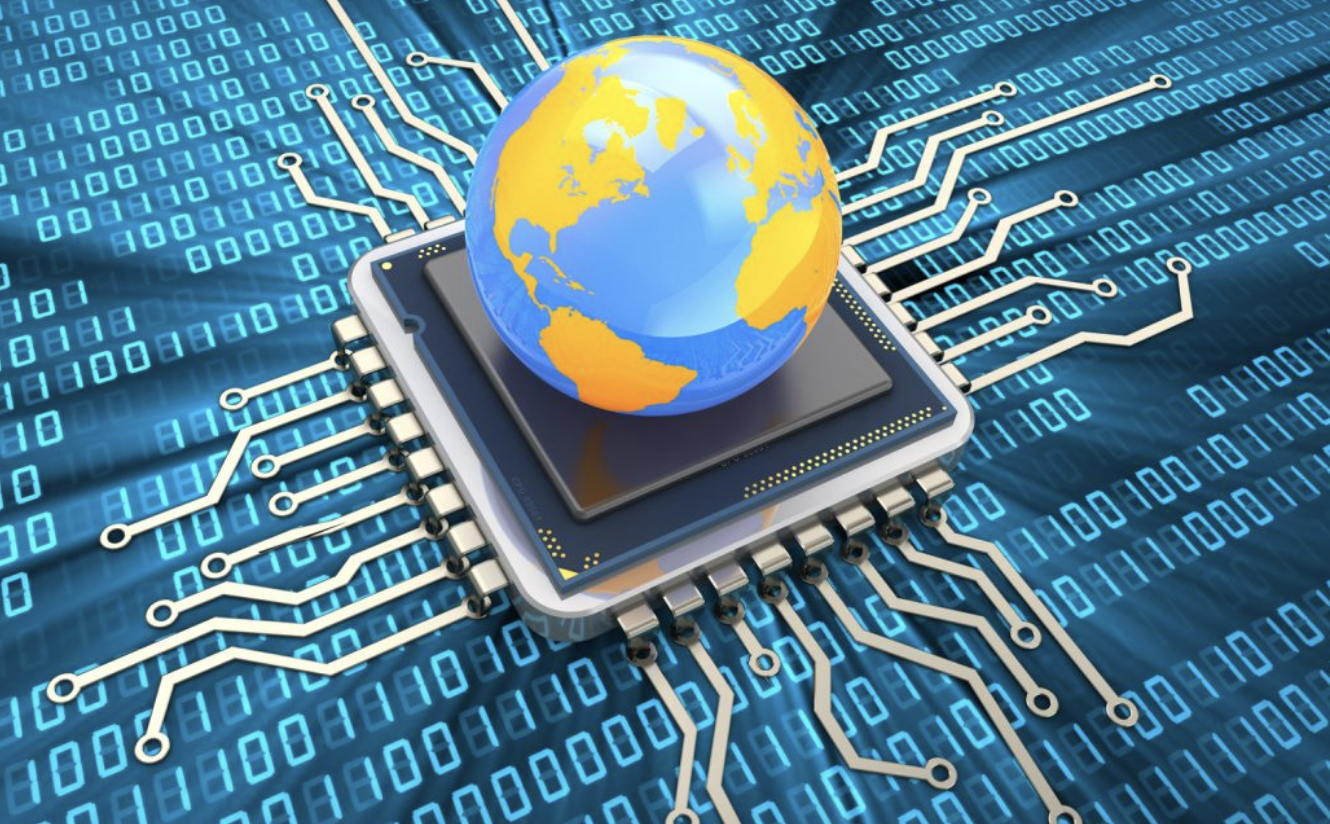
CE3001 Advanced Computer Architecture

Lab4 Report

Yang Yubei

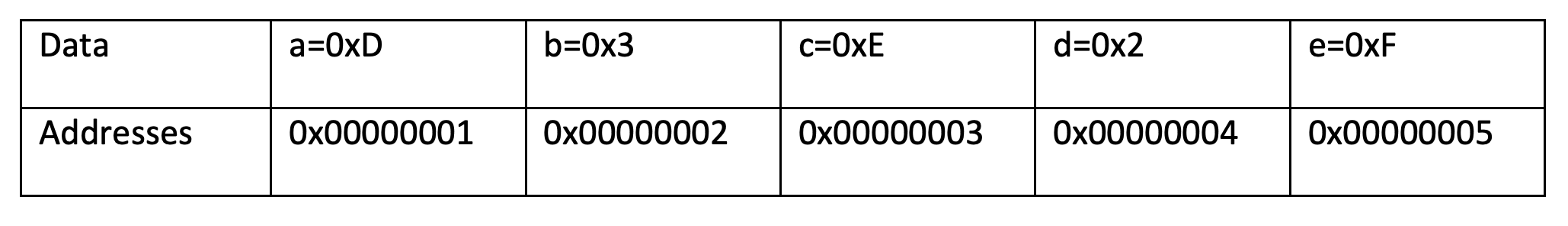
U1820525L

SEP1

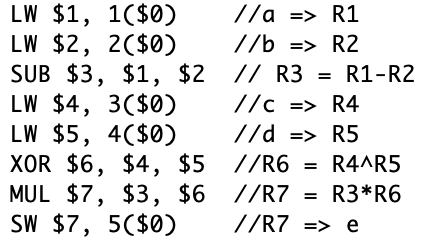


1. Lab Questions

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.



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



The minimum number of instructions: 8

**Analysis of Data dependencies**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No.Inst | Instruction | Clock Cycle | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 1 | LW $1, 1($0) | F | D | E | M | W |  |  |  |  |  |  |  |
| 2 | LW $2, 2($0) |  | F | D | E | M | W |  |  |  |  |  |  |
| 3 | SUB $3, $1, $2 |  |  | F | D | E | M | W |  |  |  |  |  |
| 4 | LW $4, 3($0) |  |  |  | F | D | E | M | W |  |  |  |  |
| 5 | LW $5, 4($0) |  |  |  |  | F | D | E | M | W |  |  |  |
| 6 | XOR $6, $4, $5 |  |  |  |  |  | F | D | E | M | W |  |  |
| 7 | MUL $7, $3, $6 |  |  |  |  |  |  | F | D | E | M | W |  |
| 8 | SW $7, 5($0) |  |  |  |  |  |  |  | F | D | E | M | W |

1. Cycle 4:

At decode stage of Inst3, values from memory is not yet stored in $1 and $2.

1. Cycle 7:

At decode stage of Inst6, values from memory is not yet stored in $4 and $5.

1. Cycle 8:

At decode stage of Inst7, value of $4^$5 is not yet written back to $6.

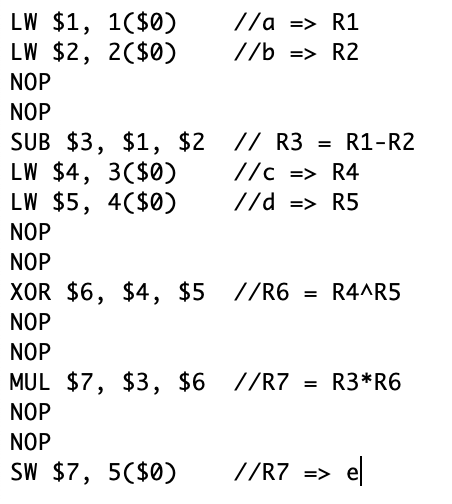
1. Cycle 9:

At decode stage of Inst8, value of $3\*$6 is not yet written back to $7.

We need add NOP instructions to eliminate data dependencies.

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.

**Without Data dependencies**



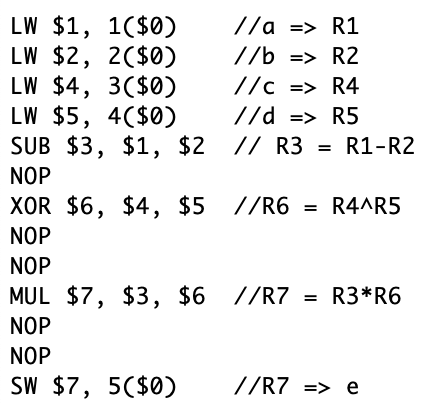
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No.Inst | Instruction | Clock Cycle | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | LW $1, 1($0) | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | LW $2, 2($0) |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | NOP |  |  | S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | NOP |  |  |  | S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | SUB $3, $1, $2 |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |
| 6 | LW $4, 3($0) |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |
| 7 | LW $5, 4($0) |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |
| 8 | NOP |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | NOP |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |  |  |  |
| 10 | XOR $6, $4, $5 |  |  |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |
| 11 | NOP |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |  |
| 12 | NOP |  |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |
| 13 | MUL $7, $3, $6 |  |  |  |  |  |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |
| 14 | NOP |  |  |  |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |
| 15 | NOP |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |
| 16 | SW $7, 5($0) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | E | M | W |

The number of instructions: 16

Clock Cycle : 20

We can reorder then instruction to make it more efficient.

**Minimum Clock Cycle & Without Data dependencies**



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No.Inst | Instruction | Clock Cycle | | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 | LW $1, 1($0) | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | LW $2, 2($0) |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |
| 3 | LW $4, 3($0) |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |
| 4 | LW $5, 4($0) |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |
| 5 | SUB $3, $1, $2 |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |
| 6 | NOP |  |  |  |  |  | S |  |  |  |  |  |  |  |  |  |  |  |
| 7 | XOR $6, $4, $5 |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |
| 8 | NOP |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |  |
| 9 | NOP |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |  |  |
| 10 | MUL $7, $3, $6 |  |  |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |
| 11 | NOP |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |  |
| 12 | NOP |  |  |  |  |  |  |  |  |  |  |  | S |  |  |  |  |  |
| 13 | SW $7, 5($0) |  |  |  |  |  |  |  |  |  |  |  |  | F | D | E | M | W |

Minimum number of instructions: 13

Minimum Clock Cycle : 17

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

Data is stored in text file ‘dmem\_txt.txt’

First column: address; Second column: data value

|  |  |
| --- | --- |
| dmem\_test.txt | ISIM simulation window |
|  |  |

This is Initialize txt. This is data memory after execution.

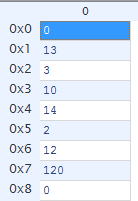
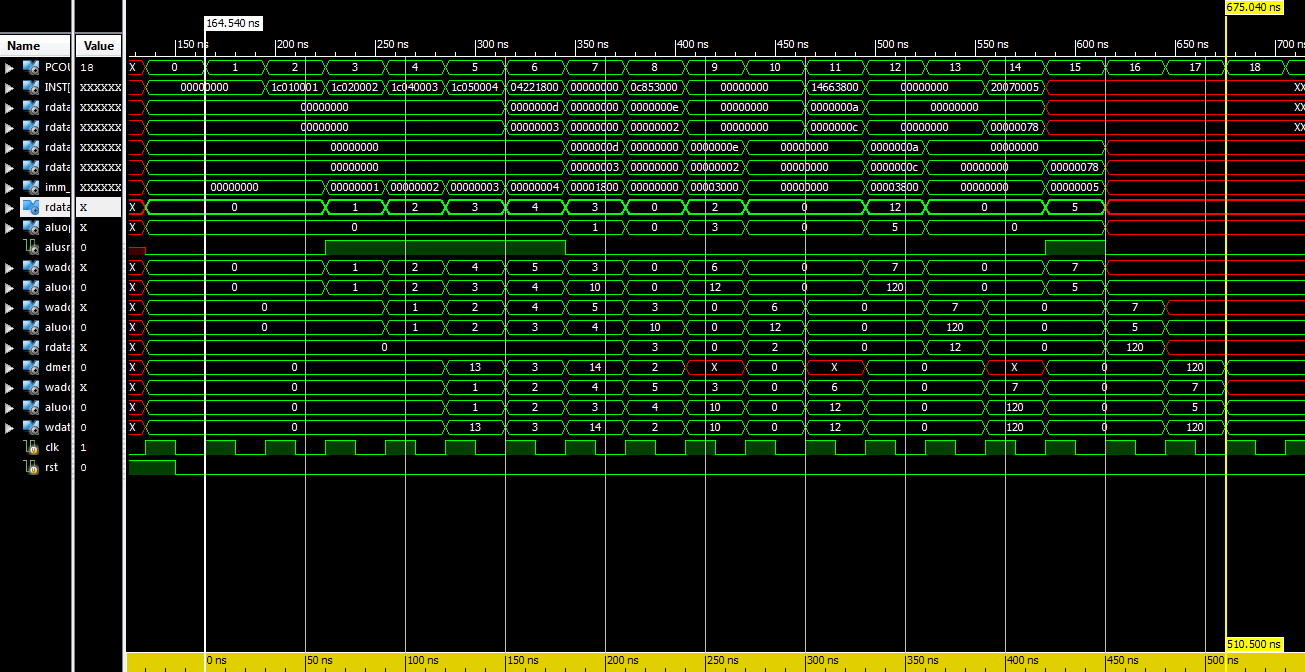
The set of instructions is stored in a text file ‘imem\_test.txt’.

Note: An extra NOP is needed at the beginning of instruction

|  |  |
| --- | --- |
| imem\_test.txt | ISIM simulation window |
|  |  |

The left graph shows the full execution in a 5-stage pipeline architecture without data forwarding.

The right graph shows the register memory after execution.



Further illustration will be in next part on LW, SW instruction and execution.

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

Take 1C010001 //LW $1, 1($0) as an example for illustration.

The addressing of LW (I-type instruction)

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | rs | rt | immediate |
|  | $0 | $1 | 1 |
| 000111 | 00000 | 00001 | 0000 0000 0000 0001 |

1. Fetch

Instruction memory of address 0x01 is fetched to **INST**[31:0] with value of 0x1C010001

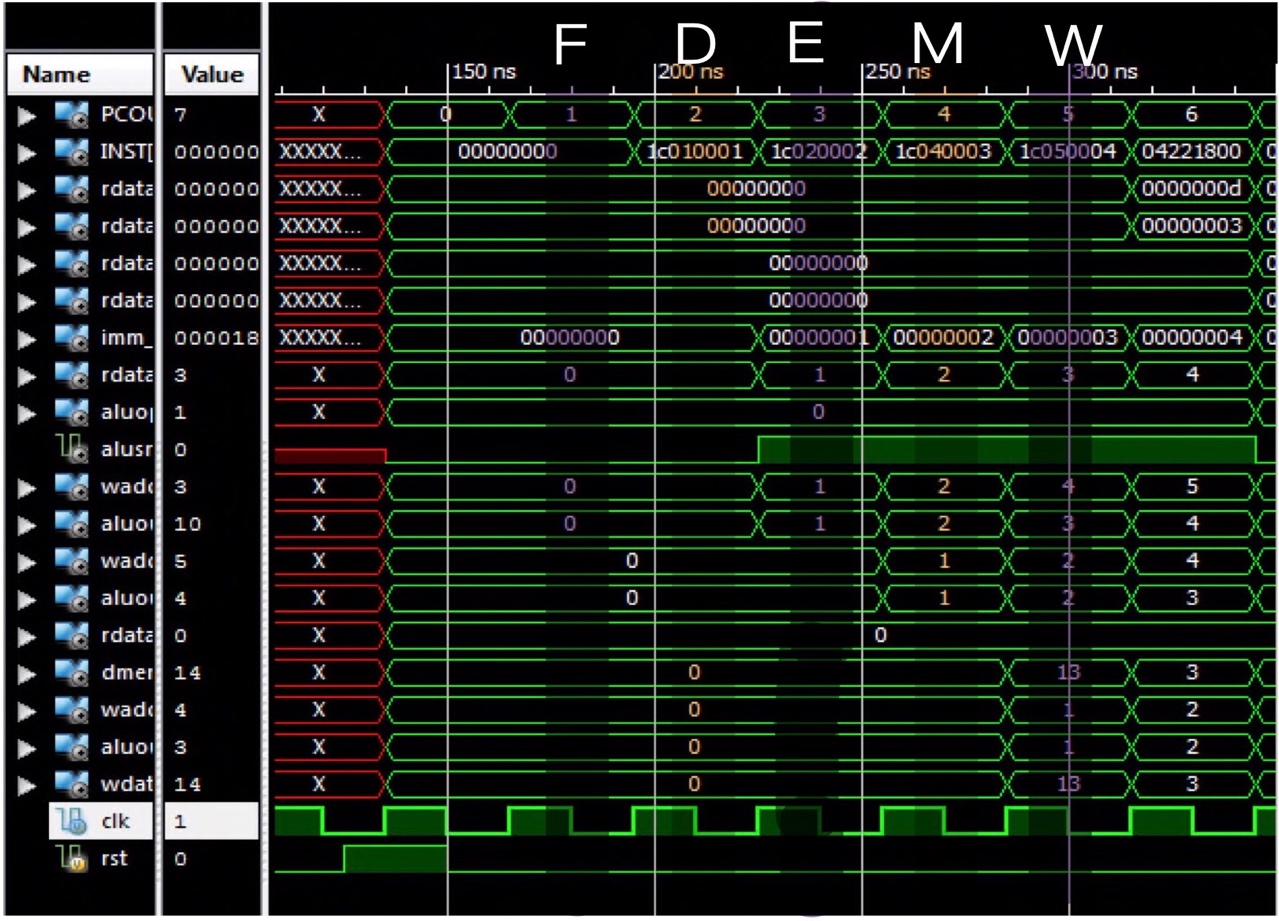
1. Decode
2. Address of $1 is write register address, stored in pipeline register **waddr\_ID\_EXE**[4:0].
3. $0 is read from register file, pass value 0 to rdata1, then is stored in pipeline register : **rdata1\_ID\_EXE**[31:0].
4. Immediate offset, 1, is sign extended and store in pipeline register **imm\_ID\_EXE**[31:0].
5. Execute
6. **aluop\_ID\_EXE**[2:0] = 000 indicates an ADD operation.
7. **alusrc\_ID\_EXE** is set to logic 1 indicate a LW instruction, which ALU should retrieve and add up **imm\_ID\_EXE**[31:0] and **rdata1\_ID\_EXE**[31:0].
8. **aluout**[31:0] = **imm\_ID\_EXE**[31:0] + **rdata1\_ID\_EXE**[31:0]= 1.
9. **aluout**[31:0] is passed to pipeline register **aluout\_EXE\_MEM**[31:0] = 1.

**waddr\_ID\_EXE**[4:0] is passed to **waddr\_EXE\_MEM**[4:0].

1. Memory
2. **aluout\_EXE\_MEM**[31:0] = 1 indicates read from data memory of address 0x0001.
3. MemWrite is logic 0 indicates no writing to memory in LW instruction.
4. **waddr\_EXE\_MEM**[4:0] is passed to **waddr\_MEM\_WB**[4:0].
5. Write Back
6. The value retrieved from address 0x01 of data memory is 0x0D. (In diagram is 13 since decimal radix is used.) This value is straightly output to **dmemdata[31:0]**.
7. MemtoReg is set to logic 0. Hence, **wdata\_wb** gets data from **dmemdata[31:0]** (not aluout\_MEM\_WB).
8. WriteEnable is set to logic 1.

**waddr\_MEM\_WB**[4:0] = 1 and **wdata\_wb** is written to $1.

The below graph is a snapshot of LW example.



1. **SW instruction**

Take 20070005 //SW $7, 5($0) as an example for illustration.

The addressing of SW (I-type instruction)

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | rs | rt | immediate |
|  | $0 | $7 | 5 |
| 001000 | 00 000 | 0 0111 | 0000 0000 0000 0101 |

1. Fetch Instruction memory of address 0x0D is fetched to **INST**[31:0] with value of 0x20070005.
2. Decode
3. $0 is read from register file, pass value 0 to rdata1, then is stored in pipeline register: **rdata1\_ID\_EXE**[31:0].
4. $7 is read from register file, pass value 70(decimal) to rdata2, then is stored in pipeline register: **rdata2\_ID\_EXE**[31:0].
5. Immediate offset, 5, is sign extended and store in pipeline register **imm\_ID\_EXE**[31:0].
6. Execute
7. **aluop\_ID\_EXE**[2:0] = 000 indicates an ADD operation.
8. **alusrc\_ID\_EXE** is set to logic 1 indicate a LW instruction, which ALU should retrieve and add up **imm\_ID\_EXE**[31:0] and **rdata1\_ID\_EXE**[31:0].

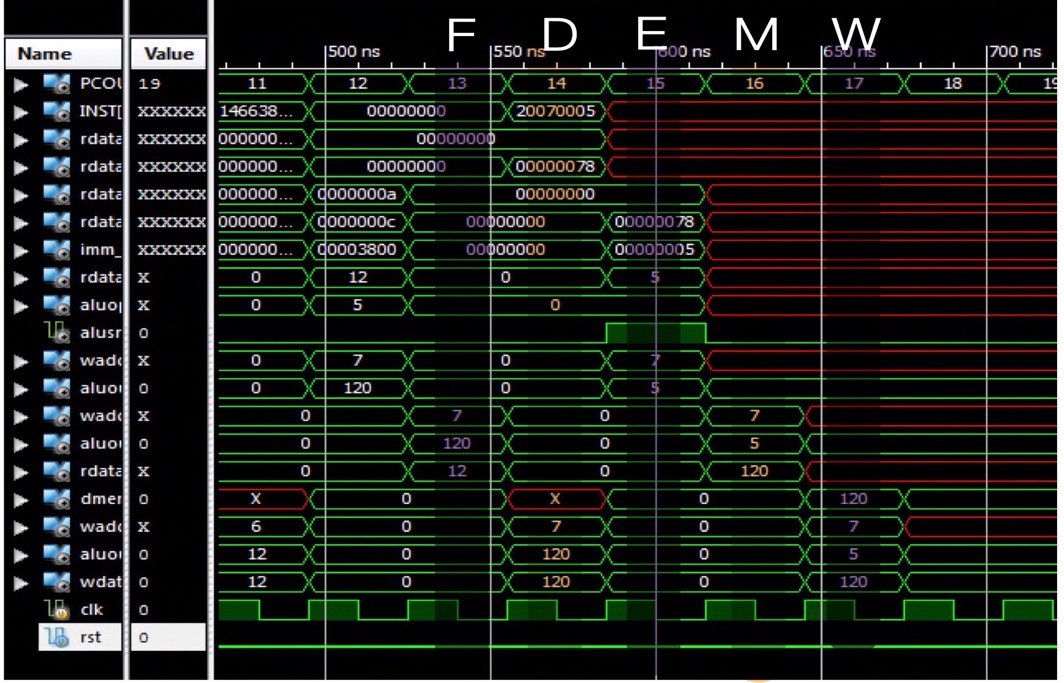
**aluout**[31:0] = **imm\_ID\_EXE**[31:0] + **rdata1\_ID\_EXE**[31:0]= 5.

1. **aluout**[31:0] is passed to pipeline register **aluout\_EXE\_MEM**[31:0] = 5.

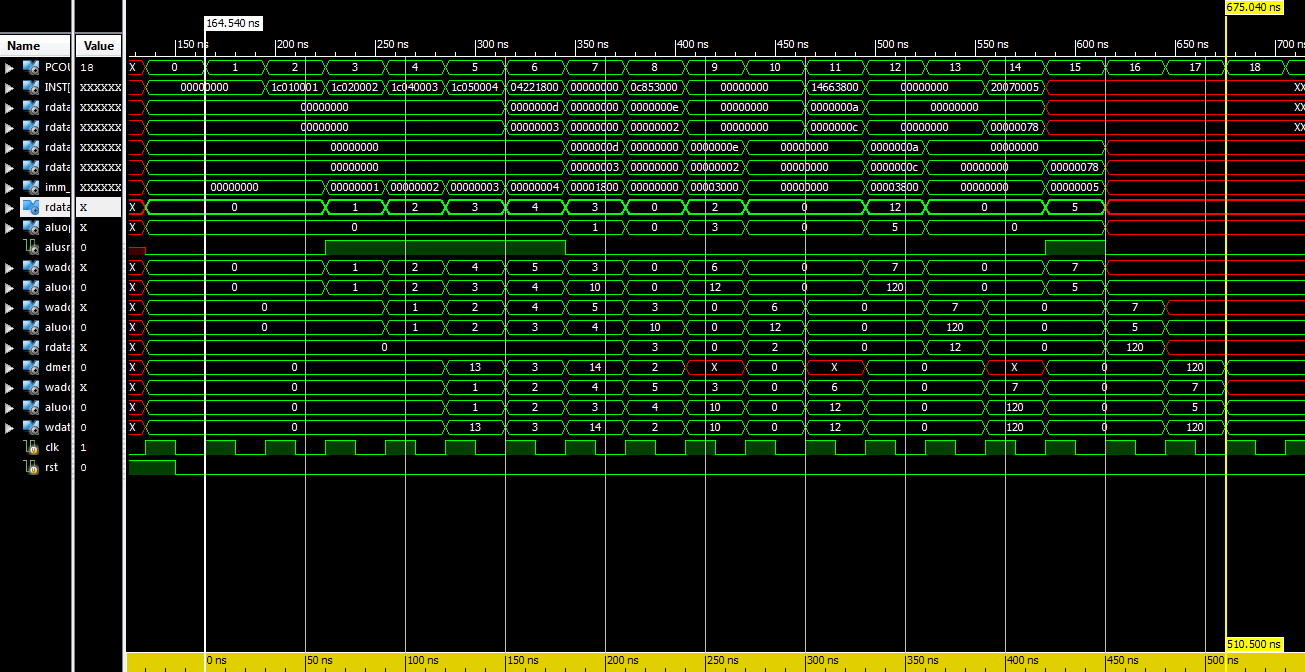
Value of **rdata2\_ID\_EXE**[31:0] is passed to **rdata2\_EXE\_MEM**[31:0]

1. Memory
2. **aluout\_EXE\_MEM**[31:0] = 5 indicates write to data memory of address 0x0005.
3. MemWrite is logic 1 indicates writing to memory in SW instruction.
4. **rdata2\_EXE\_MEM**[31:0] is written to data memory of address 0x0005.
5. Write Back

This stage is not skipped but there is no write back to register file in SW instruction. The WriteEnable is set to logic 0.



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.



From the above snapshot, the execution time is **510.500 ns**.

The formula is:

Execution time = Clock Period \* CPI \* No.of Instructions

Clock period is 30ns. Then, Execution time = 30ns \* 1 \* 17 = 510 ns.

The cursor measurement is close to what we expected.

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