Project Report

Team: Samir w Aly

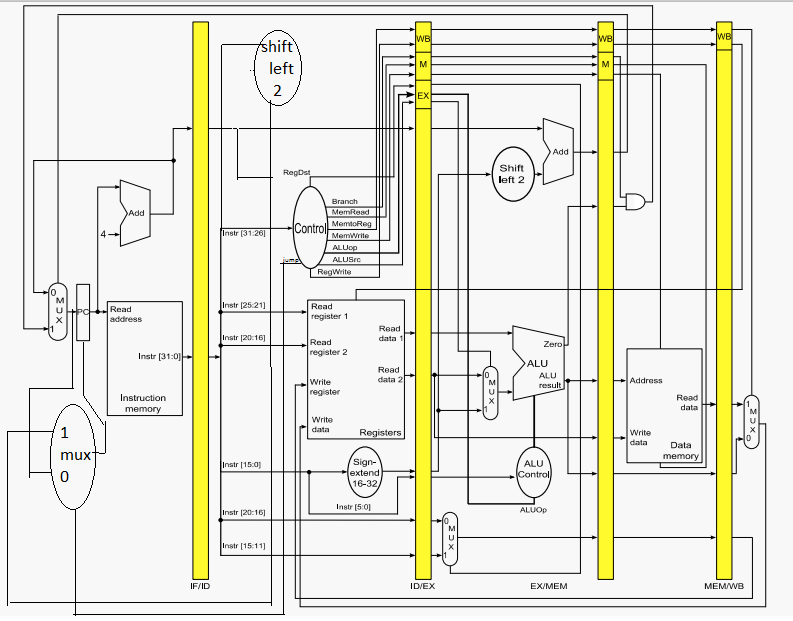
|  |  |
| --- | --- |
| **Team Members** | |
| **Aly ahmed ezz** | **28-1278** |
| **Mohamed badr** | 28- |
| **Sherif Ihab** | 28-0748 |
| **Ali Soliman** | 28-13627 |

**1.Brief Description:**

The programming language used was JAVA.

We created a super class for all the instructions containing control signals (each instruction has its own control signal). This class has three subclasses R-format, I-format, and J-format. Each one of those three contains their unique attributes. All instructions that should be supported are subclasses of one of the three super classes (R-format, I-format, and J-format). Creating an object of any instruction example: add, automatically generates its own control signals. We created another class called “memory” this class has two array lists, instruction memory and data memory. Data memory is an array list of integers, each entry represents a word. Instruction memory array list of instructions, each entry represents an instruction. We created another class called “registers” that has an array list of size 32 each entry represents a register, example: $t0 is 8. We created another class called “alu” that has execute method that execute a certain instruction, it also contains a method called “zero” that is used for branch instructions. We created a parser that simply takes the string that the user enters ( MIPS program ) and converts it to a set of instructions ( Instruction Memory). The “main” class is the one that uses all these components to simulate the mips data path.

**2.DataPath:**

****

**3.Control Unit:**

Control signals are set according to the class of the instruction when it is created.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **RegDest** | **Branch** | **MemRead** | **Memtreg** | **MemWrite** | **aluOp** | **aluSrc** | **RegWrite** | **jump** |
| **Add** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Addi** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **And** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Beq** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |
| **Bne** | **0** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |
| **J** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **Jal** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **Jr** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **Lb** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** |
| **Lbu** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** |
| **Lui** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
| **Lw** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** |
| **Nor** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Sb** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **0** |
| **Sll** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Slt** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Sltu** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Srl** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Sub** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Sw** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **0** |

**4.Assumptions:**

1) A single data entry in the data memory array is considered a word, not a byte.

2) A single entry in the instruction memory contains one instruction.

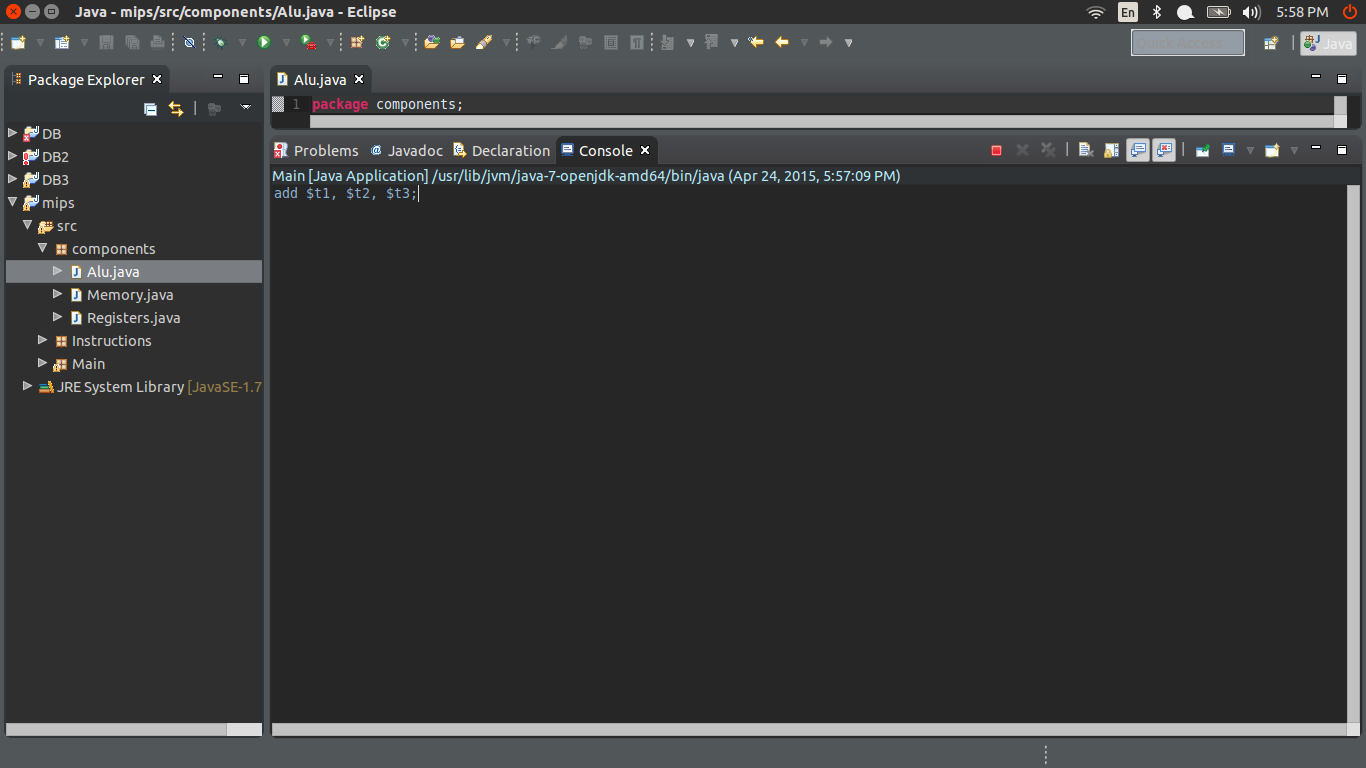
3) Instructions contain their own control signals, and therefore the pipelining registers each contain an instruction.

4) rs, rt, rd, immediate address, and shift amount are all integers.

5) aluOp control signal is determined by the class of the instruction.

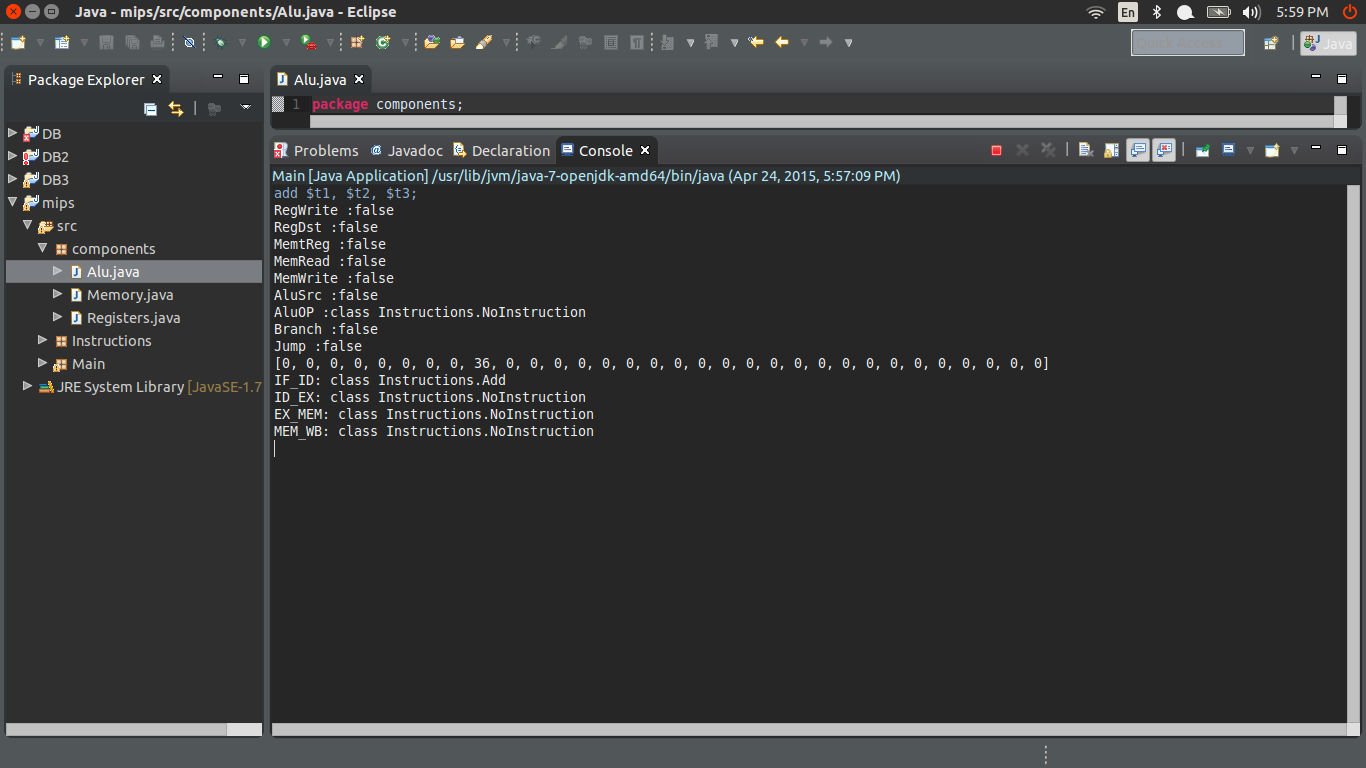
**5.User guide:**

Enter the desired program in a string with each instruction ended by a semicolon. Note: you should follow the string written in the screen shot exactly including the spaces and commas and semicolon. Press enter when you are done.

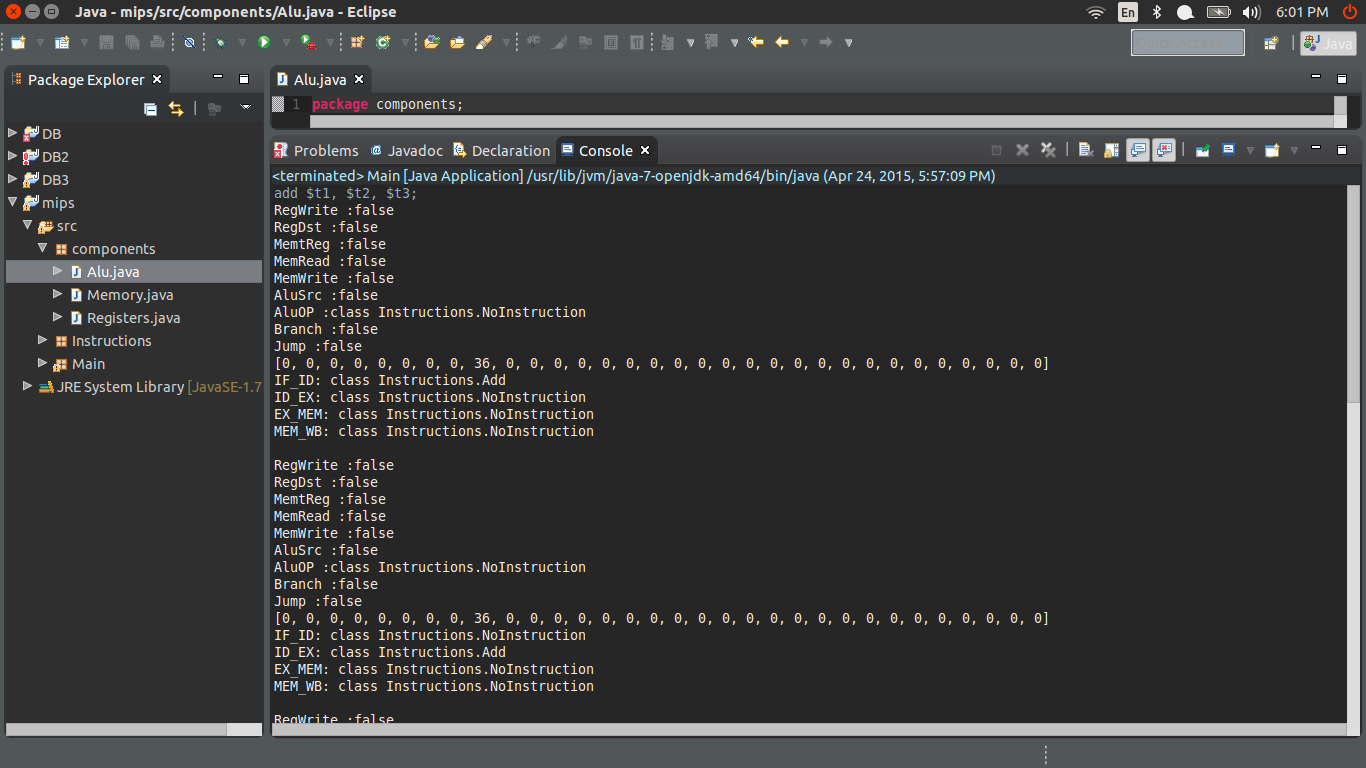


The program is now running step by step (cycle by cycle) at each step, it shows all the control signals as well as the registers and the content (instruction) in the pipelining registers. Press enter to run the program one more cycle.

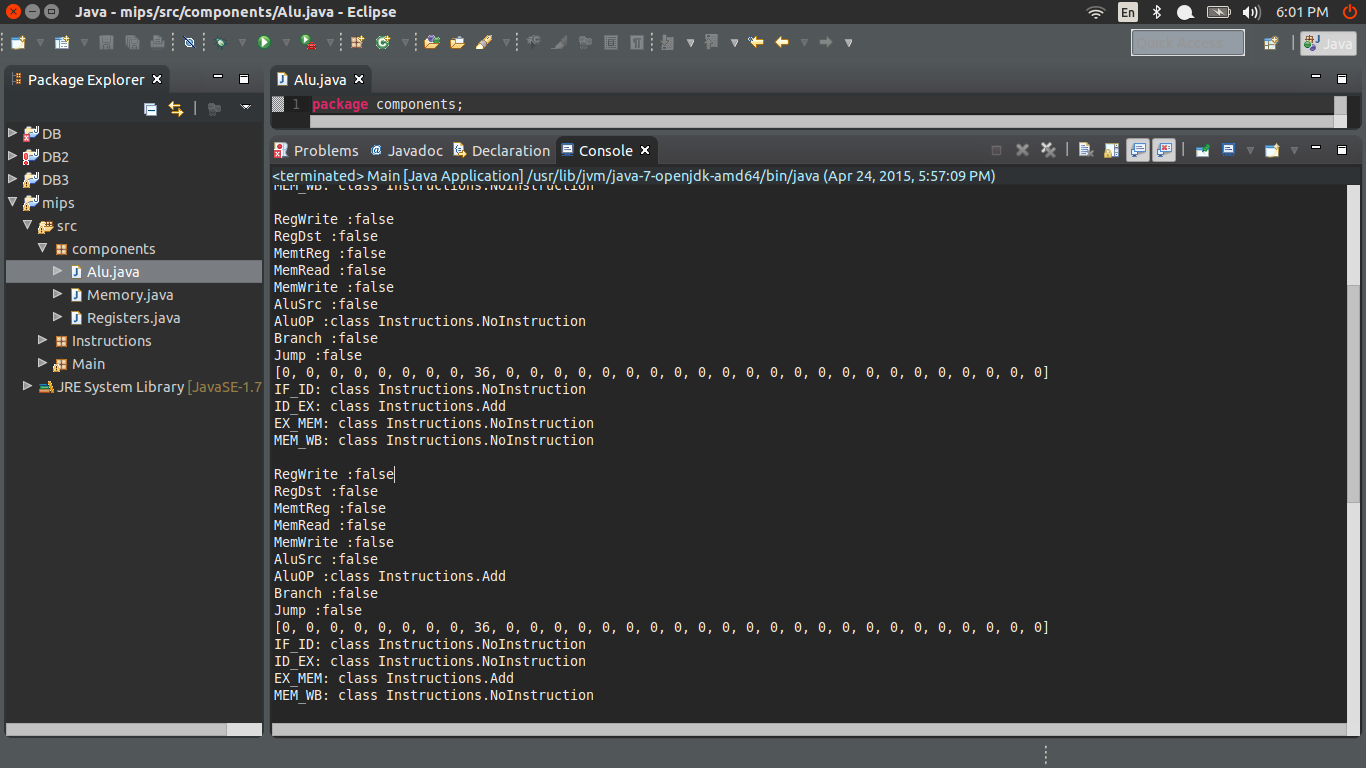
1. Fetching the instruction



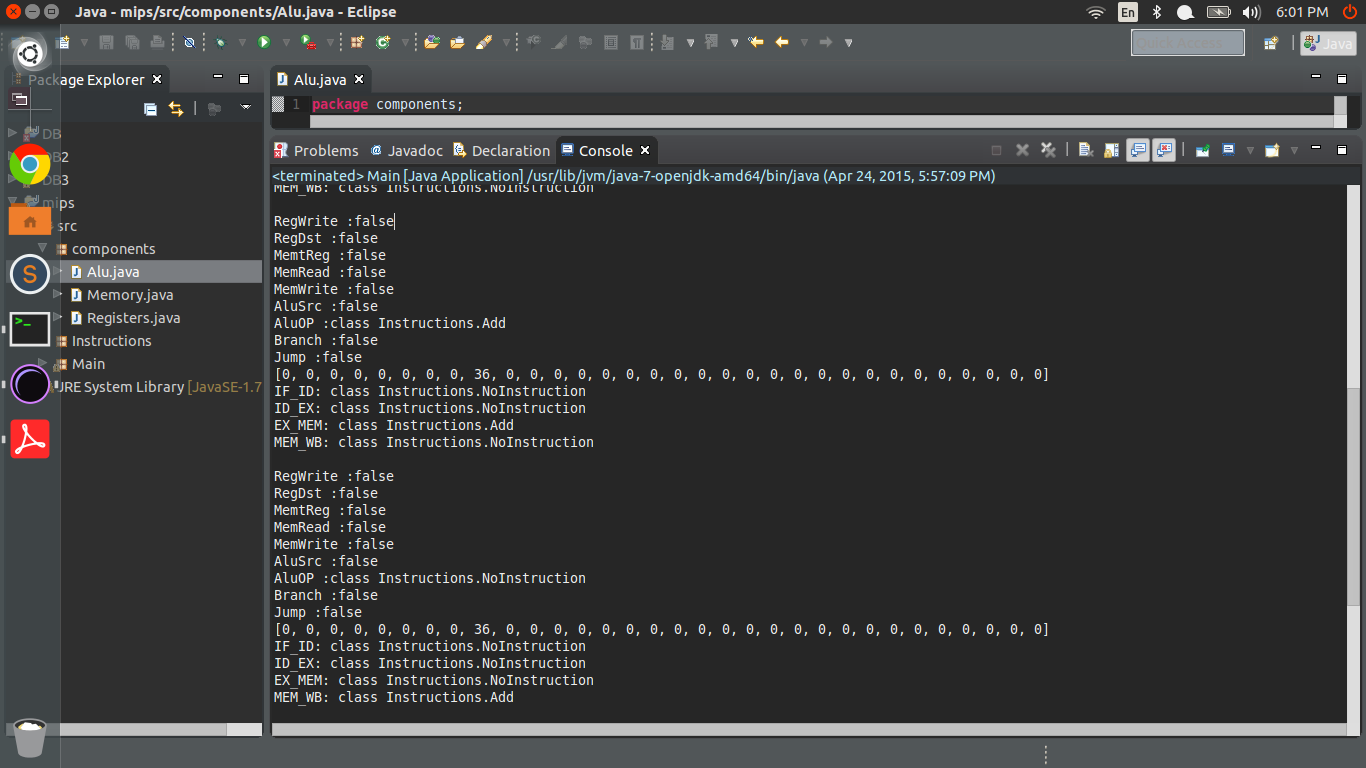
1. Decoding



1. Execute

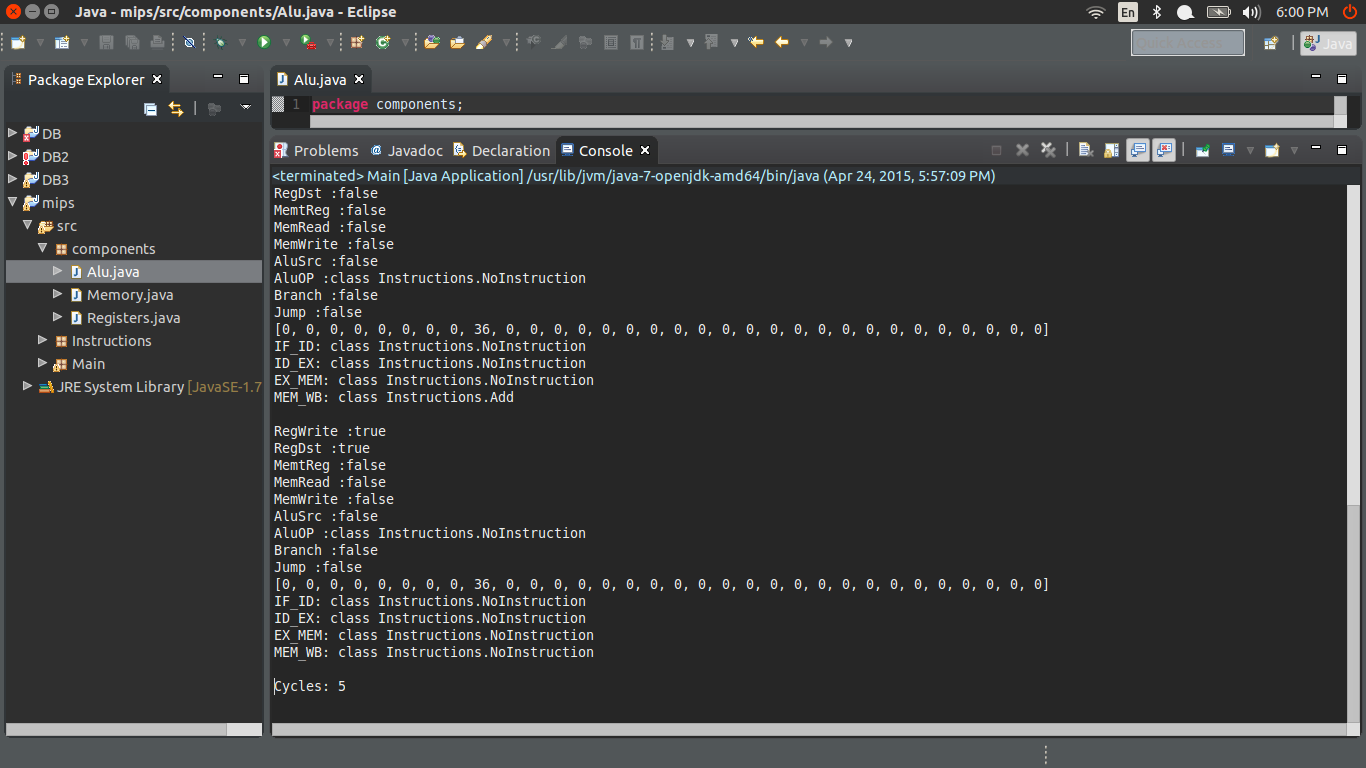


1. Memory



The program has finished running the instruction(s). And its shows the number of cycles.

1. Write back



**6.Testing:**

A)

addi $t4, $t4, -6; addi $t0, $t0, 5; addi $t1, $t1, 10; addi $t6, $t6, 11; add $s0, $t0, $t0; sub $s1, $t1, $t0; sll $s3, $t0, 2; srl $s2, $t1, 1; and $s4, $t1, $t2; nor $s5, $t1, $t2; sw $t1, 0($s0); sb $t1, 0($t2); lw $s6, 0($s0); lb $s7, 0($t2); slt $t3, $t0, $t1; sltu $t5, $t4, $t0;

B) Loop

addi $t1, $t1, 3; add $s7, $s7, $s7; add $s7, $s7, $s7; add $s7, $s7, $s7; addi $t4, $t4, 1; add $s7, $s7, $s7; add $s7, $s7, $s7; add $s7, $s7, $s7; bne $t1, $t4, -4; beq $t1, $s0, -4;

C) Function call

addi $t1, $t1, 3; add $s7, $s7, $s7; add $s7, $s7, $s7; add $s7, $s7, $s7; sb $t1, 0($s0); add $s7, $s7, $s7; add $s7, $s7, $s7; add $s7, $s7, $s7; lui $t0, 1; lbu $t1, 0($s0); j 12; sub $t0, $t0, $t1; jal 14; sll $t0, $s1, 2; addi $t6, $t6, 10; addi $t5, $t5, 10; jr $ra;