

***Computer Architecture Project***

***Submitted to:***

Dr. Cherif Salama Ramzi

***Submitted by:***

***NAME:*** ***ID:***

Mayar Wessam Hassan16P3008

Rowan Hazem Wagieh16P3023

Engy Samy Salah16P3004

Gina Emil Attia16P3022

**Implementation Description:**

* **Project Language:** We have implemented our project using JAVA language.
* **Implementing:** We have simulated all the components ( Registers , Alus , Memories , Multiplexers , Sign Extenders , Control Unit , Pc & Instruction Memory) as separated classes. Each class contains a method or more.

**Class Instruct\_mem**: contains a method that stores the instructions which the user has entered in array.

**Class PC:** contains a static variable that has the address which the user has entered. Also it contains a method that adds 4 on the address.

**Class Reg\_file**: contains 3 arrays for registers. One for the registers’ addresses, and one for the initial values (zero), and the last one is for the registers’ names. It also contains 2 methods, one for writing in a specific register, and the other one for reading the value in a specific register.

**Class Instruction:** contains a method that takes the instruction from the user (in 32-bit binary form). If it is R-type, it will be divided into 6 bits for the opcode, 5 bits for first source register, 5 bits for second source register, 5 bits for destination register, 5 bits for shift amount, and 6 bits for function code.

If it is I-type, it will be divided into 6 bits for opcode, 5 bits for source register, 5 bits for destination register (rt in this condition), and 16 bits for a constant or address.

If it is J-type, it will be divided into 6 bits for opcode and 26 bits for address.

**Class Control\_Unit:** contains a method that takes the opcode from the instruction class’s method and generate the control unit signals for each opcode.

**Class Sign\_Extend:** contains a method that takes the constant (or the address) from the instruction class’s method and extends it from 16 bits to 32 bits.

**Class ALU\_Control:** contains a method that takes the opcode and the function code (if it is R-type) from the instruction class’s method and create the needed function code.

**Class ALU:** contains a method that takes the function code from the ALU\_control class’s method and controls the ALU for which operation it will perform. Also it contains a variable (zeroflag) which its value is always (0), its value might change to (1) in case of (branch instruction) only according to the value of ALU.

**Class Memory:** contains a method that support SW, SB, LW, LB, LBU operations. In SW operation it takes the needs address from ALU and read the needed value from (rt) which is the word (32-bits).It divides this word into 4 bytes. For example, the needed address to store in is 200, So first byte will be stored in 200, the second in 201, the third in 202 and the last one in 203. In SB operation it takes the byte and store it in the address needed. In LW operation it takes the needed address from ALU. For example, the address is 200, it goes to 200 and load the first byte then to 201 and load the second byte then to 202 and load the third byte then to 203 and load the last byte, Finally, it concatenates the 4 bytes. In LB it takes the needed address from the ALU to load from it. And in LBU operation the same thing happens as in LB operation, but it converts the loaded bytes into a positive number.

|  |  |  |  |
| --- | --- | --- | --- |
| **200** | **201** | **202** | **203** |
| 204 | 205 | 206 | 207 |
| 208 | 209 | 210 | 211 |
| 212 | 213 | 214 | 215 |

**Class Muxes:** contains a method that takes RegWrite, Memtoreg, and Regdst signals from Control\_Unit class’s method to know where to write the output value from the ALU class’s method.

**Class ALU\_Branch**: contains a method that takes the constant (the address) from the Sign\_Extend class’s method and multiply it by 4.

**Class MUX\_Branch**: contains a method that controls if the new address which will be entered the pc will be the value from the (Pc class)’s method or the value from (ALU\_Branch class)’s method added to the value from the (Pc class)’s method.

**Class Project1:** contains 2 methods, one for calling all the previous classes, and the other one is for printing all the signal wires values. Also it contains the main of the program.

**Datapath:**

All the instructions are saved in the instruction memory, then each clock cycle a single instruction is loaded from it according to the pc address.

**Firstly** the instruction goes to class (Instruction) which divides the instruction (32-bit) into specific number of bits according to its format.

**In addition** we have supported an extension which is a wire for the shift-amount 5-bits that goes directly to class (ALU).

**Secondly** we use the Opcode in class (Contol\_Unit) to specify the right control signals for this instruction.

**Also** we have supported an extension which is a signal wire (1-bit) for the jump operation.

**Thirdly** the constant which is from Instruction[15-0] bit goes to class (Sign\_Extend) to extend it from 16 bit to 32 bit.

**Fourthly** the ALUOp from class (Control\_Unit) and the function code from class (Instruction) goes to class (ALU\_Control) to determine the needed ALU- function code for class (ALU).

**After that** we call the class (reg\_file) to determine from it the needed value of (RS) and read it.

**Then** It goes to class (ALU) to do the specific operation.

**Moreover**, it goes to class (Memory) to know if we need to read from the memory or write in it. Otherwise it writes the output value in (rd register) by using class (Muxes).

**Then** detecting the address which passes to PC in class (Mux\_branch).

**Finally** The signal wires, register file and memory are printed.

**Truth table:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP-CODE** | | | | | | **Control Signals** | | | | | | | | | |
|  | | | | | | **RegDst** | **Branch** | **Mem-**  **Read** | **Mem-**  **toReg** | **ALUOp** | | **Mem-**  **Write** | **ALU-**  **Src** | **Reg-**  **Write** | **Jump** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | X | 1 | 0 | X | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | X | 0 | 0 | X | x | x | 0 | x | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | X | 0 | 0 | X | x | x | 0 | x | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | X | 0 | 0 | X | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | X | 0 | 0 | X | 0 | 0 | 1 | 1 | 0 | 0 |

**K-maps:**

Case 1: (RegDst)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 1 | 0 |  |  | 0 | X |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | X |  |  |  | 0 | X |  |  |
| **010** | X | 0 |  |  |  |  |  |  |
| **100** | X |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

RegDst **=** A’B’C’D’F’



Case 2: (Branch)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 0 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | 0 |  |  |  | 0 | 0 |  |  |
| **010** | 0 | 0 |  |  |  |  |  |  |
| **100** | 1 |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

Branch =A’B’C’DE’F’



Case 3: (MemRead)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 1 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | 0 |  |  |  | 1 | 0 |  |  |
| **010** | 0 | 0 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 1 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

MemRead =AB’C’E’F + AB’C’D’EF



Case 4: (MemtoReg)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 1 | X |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | X |  |  |  | 1 | X |  |  |
| **010** | X | 0 |  |  |  |  |  |  |
| **100** | X |  |  |  | 1 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

MemtoReg = AB’C’E’F’ + B’C’D’EF



Case 5: ( ALU-Op “First bit ”)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 1 | 0 |  |  | 0 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | X |  |  |  | 0 | 0 |  |  |
| **010** | X | 1 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

ALU-Op = A’B’C’D’F’ + A’B’D’EF’



Case 6: (ALU-Op “Second bit ”)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 0 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | X |  |  |  | 0 | 0 |  |  |
| **010** | X | 1 |  |  |  |  |  |  |
| **100** | 1 |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

ALU-Op = A’B’D’EF’ + A’B’C’DE’F’



Case 7: (MemWrite)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 0 | 1 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | 0 |  |  |  | 0 | 1 |  |  |
| **010** | 0 | 0 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

MemWrite = AB’CD’E’F’ + AB’CD’EF



Case 8: (ALUSrc)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 1 |  |  | 1 | 1 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | X |  |  |  | 1 | 1 |  |  |
| **010** | X | 1 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 1 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

ALUSrc = A’B’CD’F’ + B’CD’E’F’ + AB’C’E’F’ + AB’D’EF



Case 9: (RegWrite)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 1 | 1 |  |  | 1 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | 1 |  |  |  | 1 | 0 |  |  |
| **010** | 0 | 1 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 1 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

RegWrite=A’B’D’E’F’+B’C’D’EF+ A’B’CD’F’ + AB’C’E’F’



Case 10: (Jump)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ABC**  **DEF** | **000** | **001** | **011** | **010** | **100** | **101** | **111** | **110** |
| **000** | 0 | 0 |  |  | 0 | 0 |  |  |
| **001** |  |  |  |  |  |  |  |  |
| **011** | 1 |  |  |  | 0 | 0 |  |  |
| **010** | 1 | 0 |  |  |  |  |  |  |
| **100** | 0 |  |  |  | 0 |  |  |  |
| **101** |  |  |  |  |  |  |  |  |
| **111** |  |  |  |  |  |  |  |  |
| **110** |  |  |  |  |  |  |  |  |

Jump = A’B’C’D’E



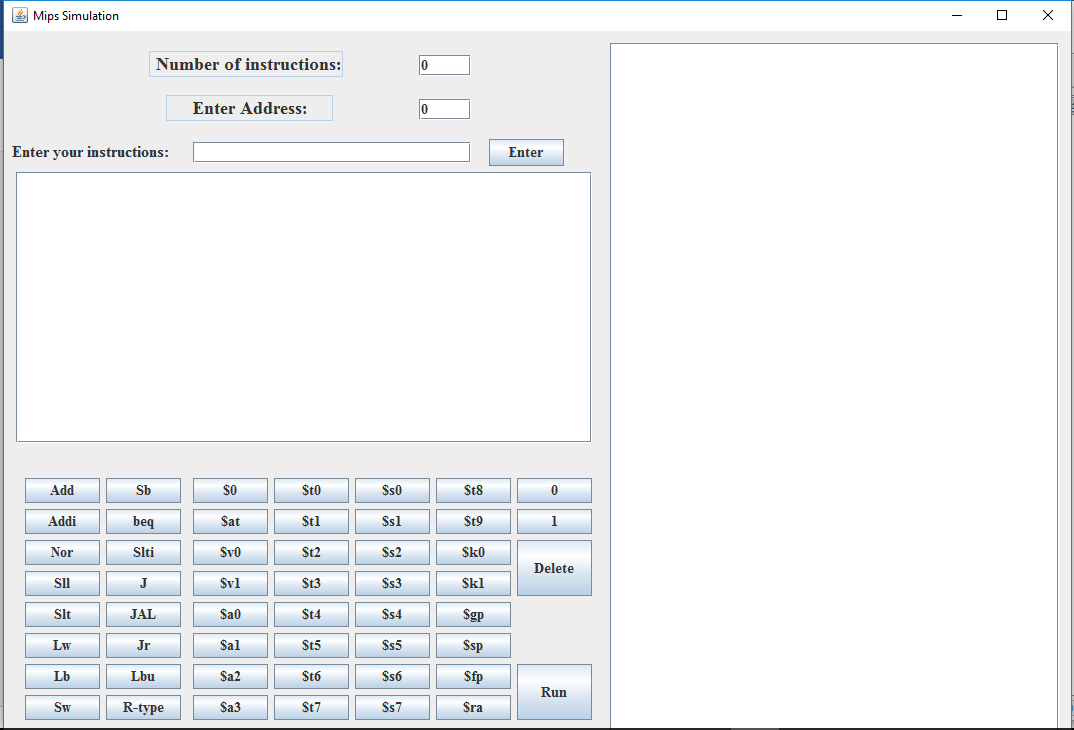
**Assumptions:**

We assumed:

The ALU control code of :

1. “NOR” = 0011
2. “SLL” = 0101
3. “Jump” & “Jal” = 0000

**User Guide:**

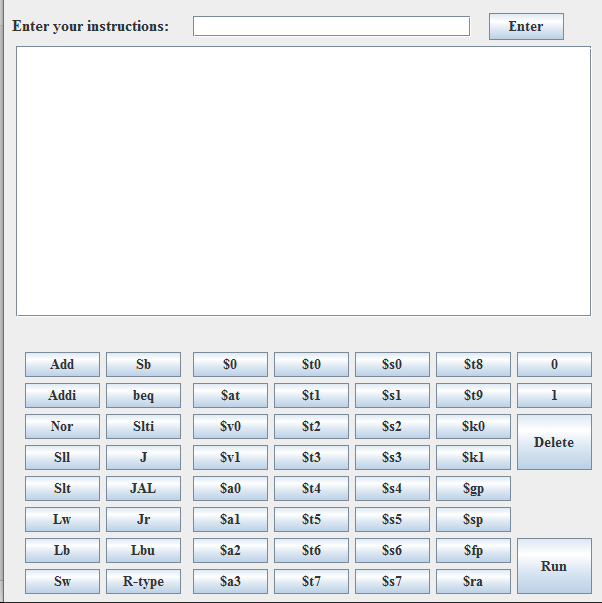
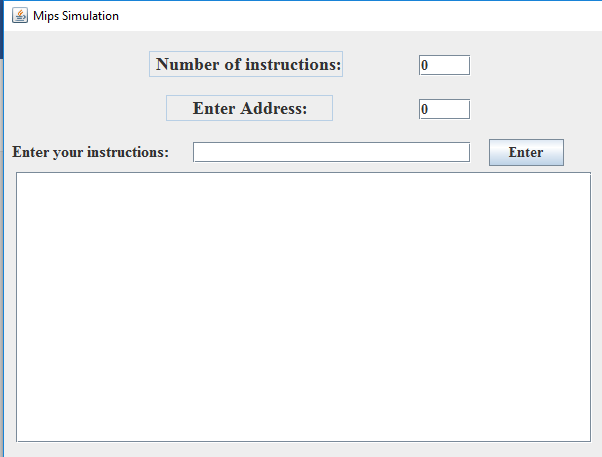
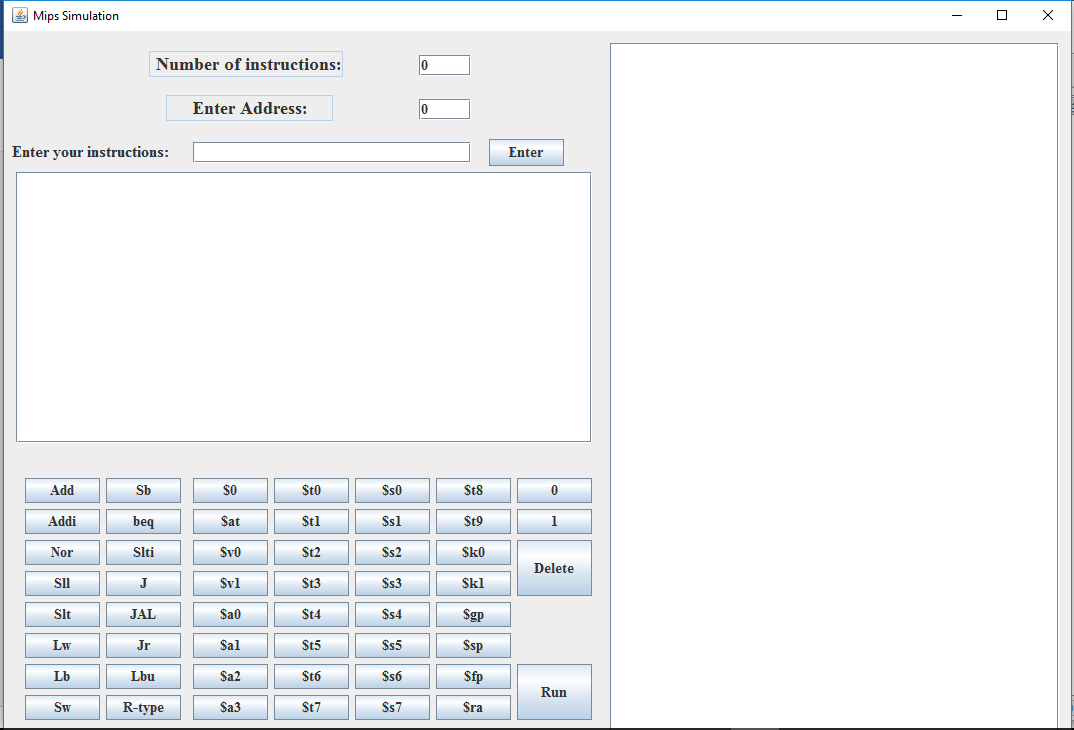
****

* Enter the number of instructions of the program which you want to build.



* Enter the address where the program’s first instruction should be loaded.



* Enter your instructions one by one either by clicking on the buttons (Not as assembly language, the button will only write for you the code in binary) or writing in the text field using **binary language**, after each instruction press enter.
* For entering a constant number or address in your instruction, use 0 and 1 buttons
* Only for R-type instructions, **press on R-type button** first then enter your instruction.
* If you want to delete what has been written in the text field, press delete.
* All your instructions that you have entered will appear here.
* Press run button to execute your program.
* Here will appear the output.

**List Of Programs**

**Program 1:**

addi $t1,$0,1

loop1:

beq $t0,$t1,finish

sll $t3,$t0,2

add $s1,$s0,$t0

sb $s1,0($t3)

lb $t5 ,0($t3)

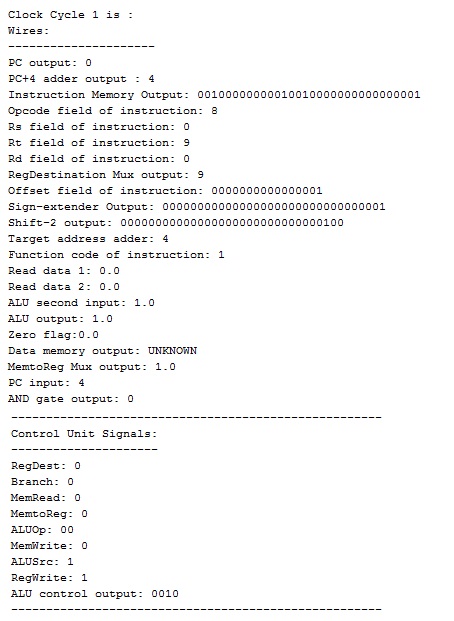
lbu $t5 ,0($t3)

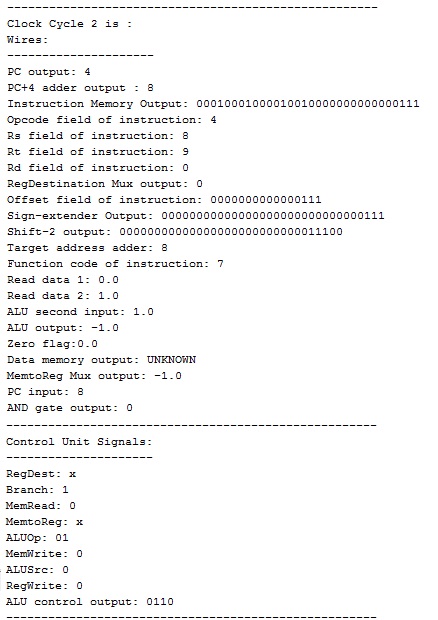
addi $t0,$t0,1

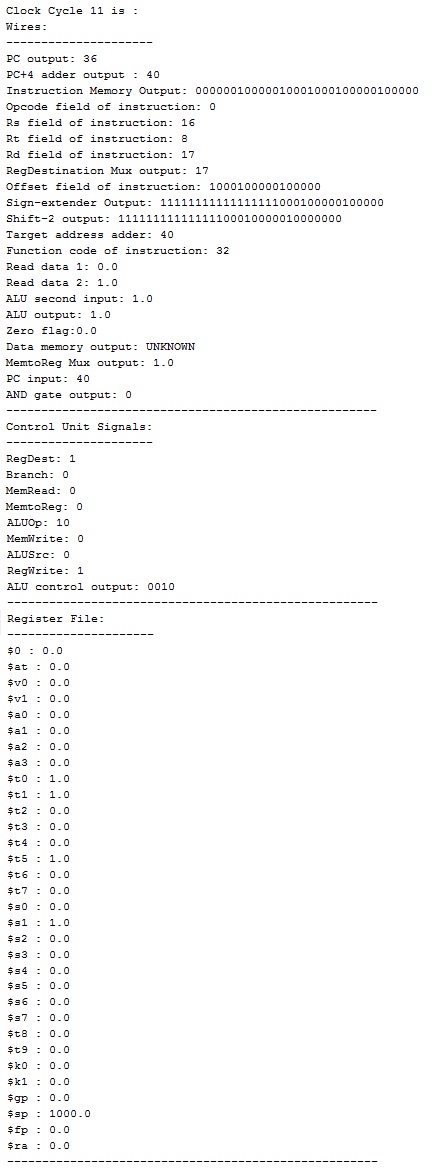
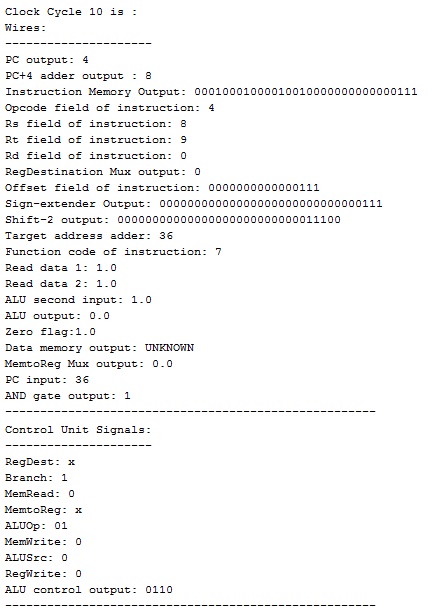
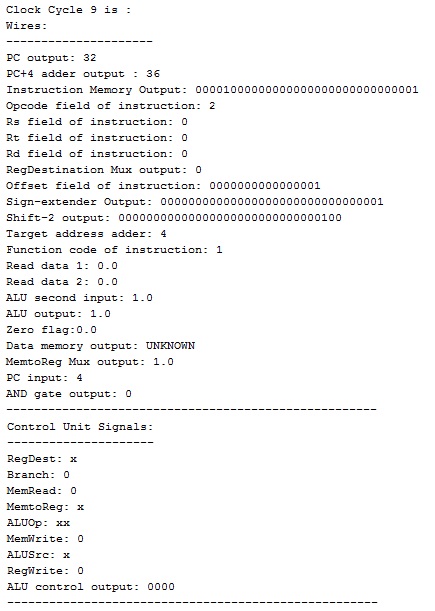
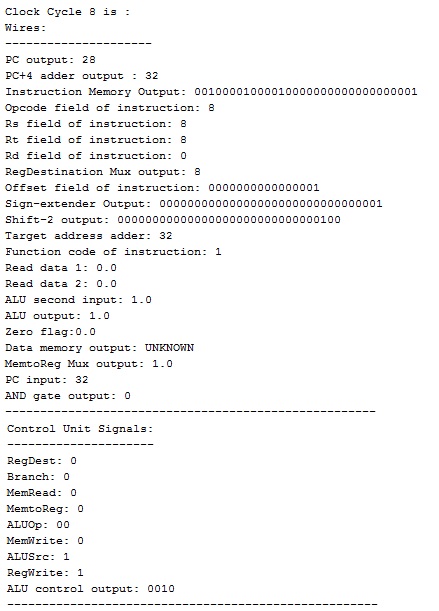
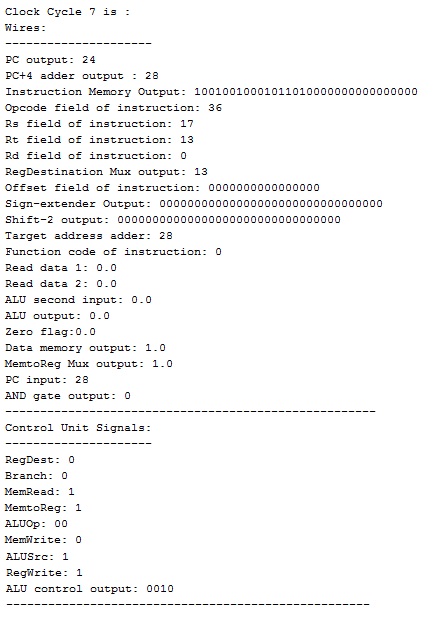
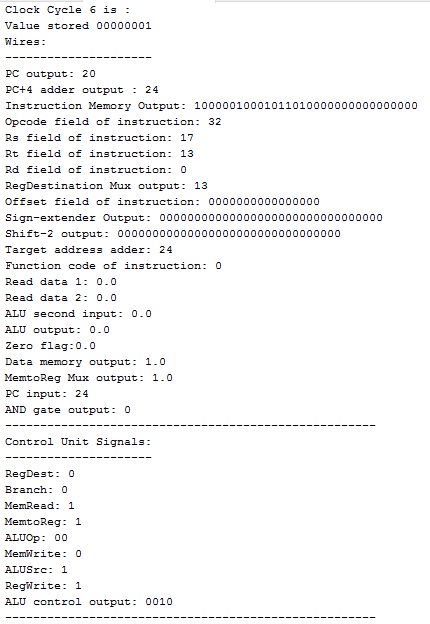
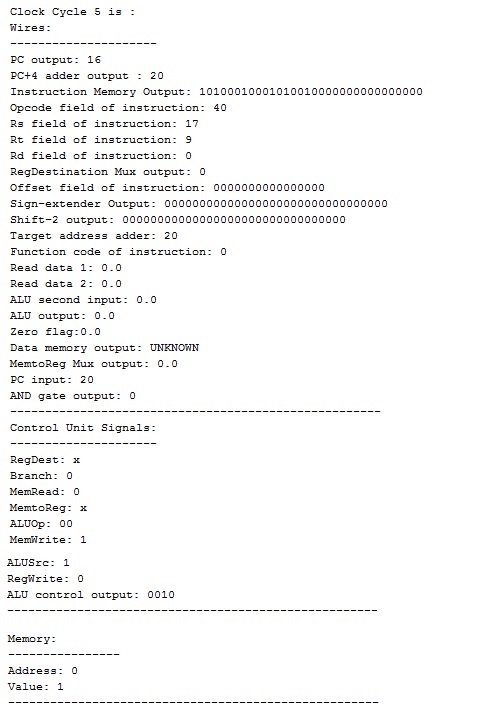
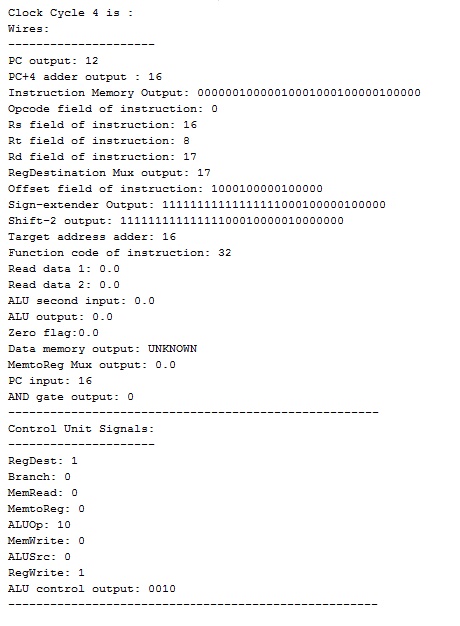
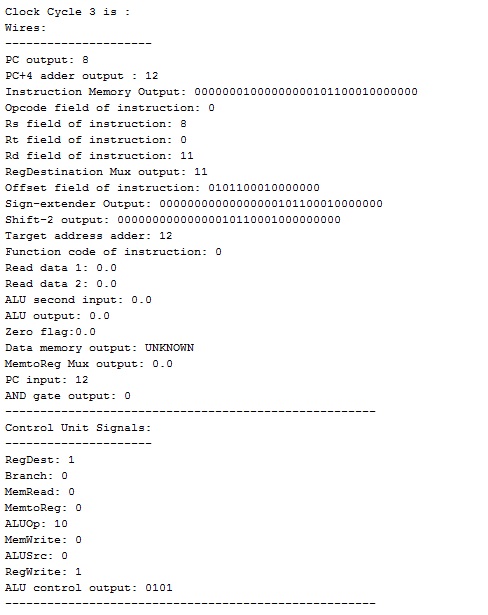
j loop1

finish:

add $s1,$s0,$t0

****

****

****

**program 2:**

funct:

sw $s0,0($t1)

lw $t5,0($t1)

slti $s1,$s0,3

beq $s2,$0,Exit

addi $s1,$s1,-1

Exit:

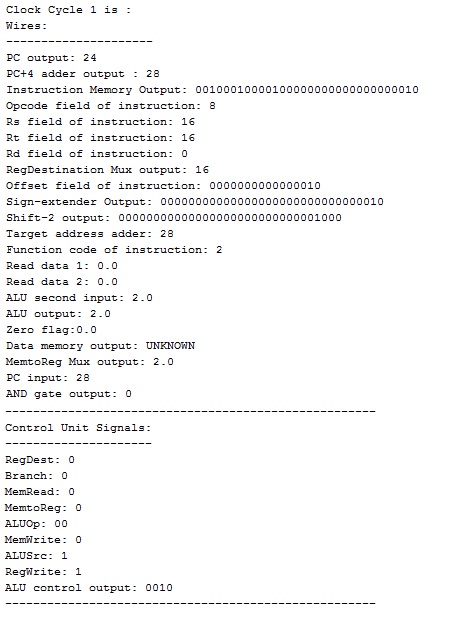
jr $ra

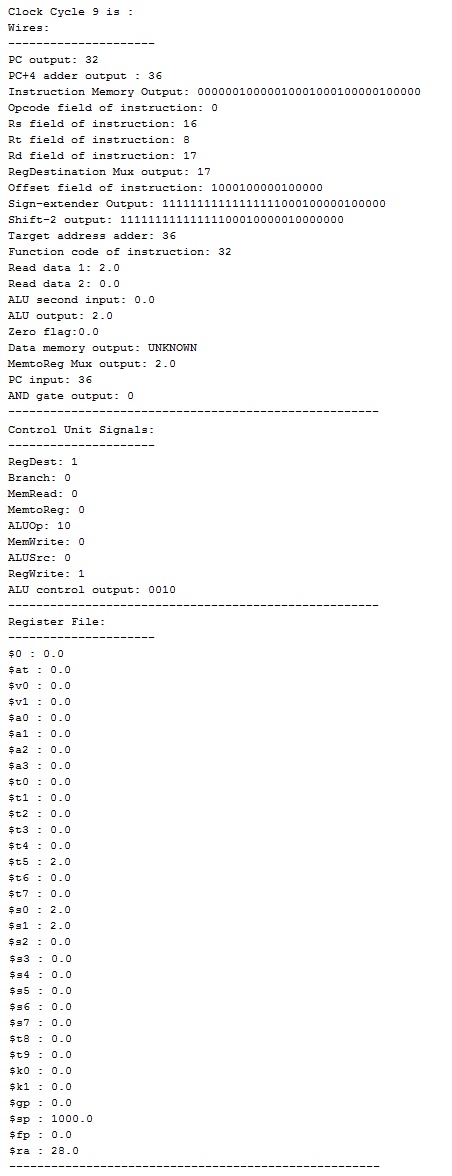
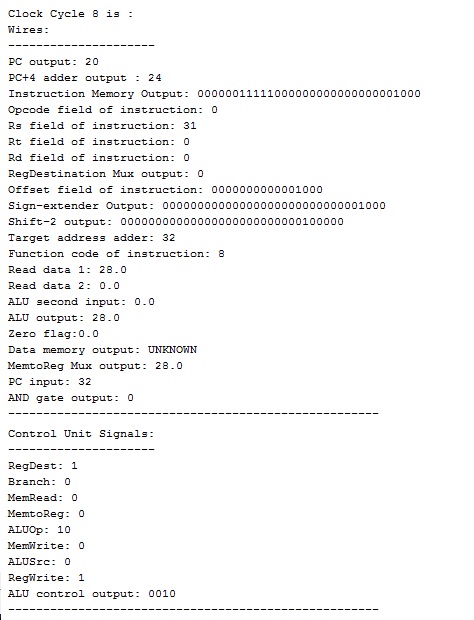
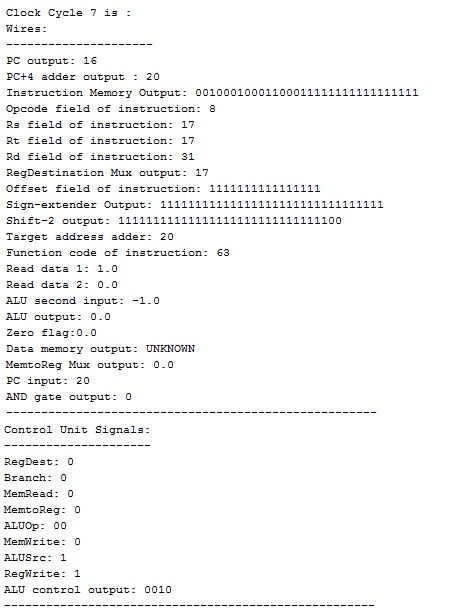
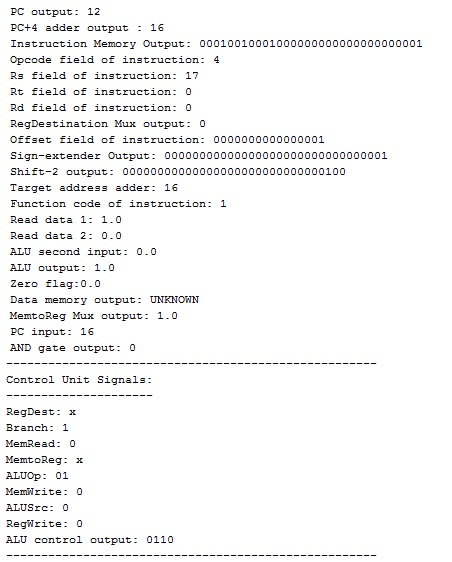
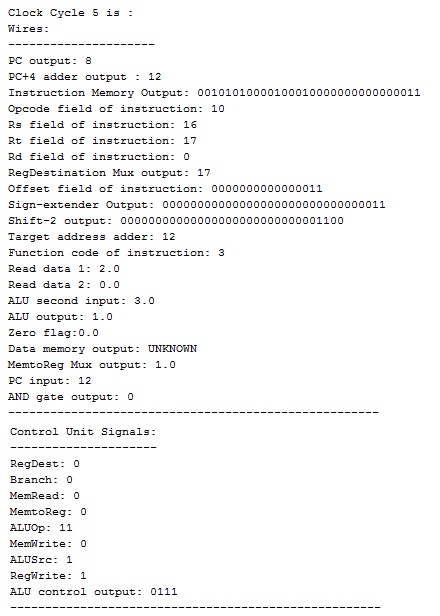
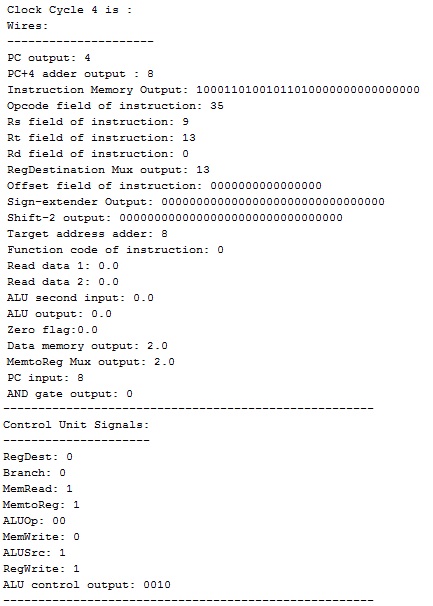
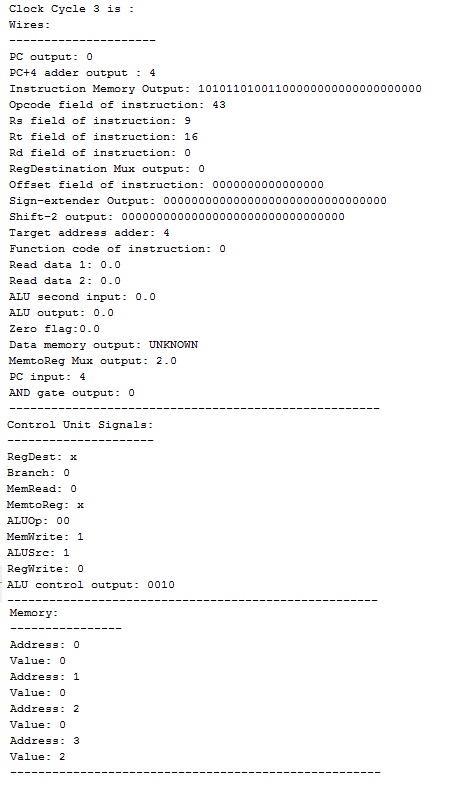
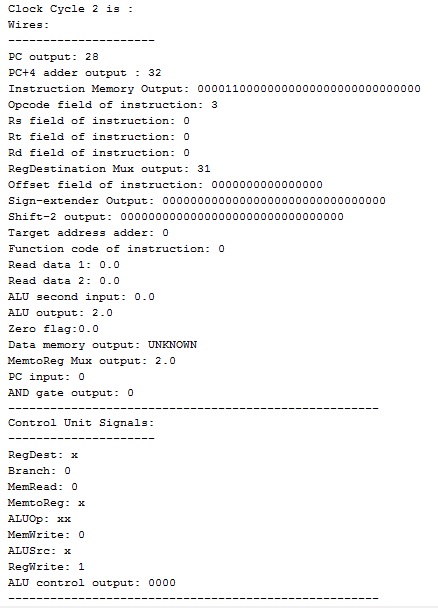
Main:

addi $s0,$0,2

jal funct

add $t5,$s0,$t0

****

****

**Teamwork**

The whole team would sit together and brainstorm ideas about each part of the code and about how we are going to implement such ideas. We divided the data-path into classes. We worked together class by class, each class we would think about how we will implement it, then after creating all classes we brainstormed how we will integrate them all together. Finally, when we connected all the classes together the code worked perfectly but there wasn’t any bonus features. Then we thought about adding the GUI to make the program more user-friendly.