MIPS Datapath and Control Unit Simulator

Done by:

Alfred Maheeb Yousry	16P8186
Andre Osama Raouf	16P6002
Andrew Raafat Adeeb Kuzman	16P8011
Daniella George Botros	16P1073
Monica Mourad Elia	16P8223
Nayer Nabil George	16P6054

Submitted to:

Dr. Cherif Ramzi Salama

1. IMPLEMENTATION

1.1. INTRODUCTION

1.1.1. Purpose of the Report

This report discusses the implementation of a low-level MIPS datapath and control unit simulator.

1.1.2. Background of the Report

MIPS is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems). The early MIPS architectures were 32-bit, with 64-bit versions added later. There are multiple versions of MIPS: including MIPS I, II, III, IV, and V; as well as five releases of MIPS32/64 (for 32- and 64-bit implementations, respectively). As of April 2017, the current version is MIPS32/64 Release 6. MIPS32/64 primarily differs from MIPS I–V by defining the privileged kernel mode System Control Coprocessor in addition to the user mode architecture.

1.2. Implementation Language

The programming language used in implementation is Java, as it supports graphical user interface (GUI) which was used in building the application.

1.3. Instruction Set Architecture (ISA)

- **Arithmetic:** add, addi, sub
- **Load/Store:** lw, sw, lb, lbu, sb, lh, lhu, sh, lui
- **Logic:** sll, nor, srl, and, andi, or, ori
- **Control flow:** beq, bne, j, jal, jr
- **Comparison:** slt, slti, sltu, sltui
- **Labels:** L1, L2, L3, L4
- **Pseudo instructions:** mul, move, blt

1.4. Simulation and Simulation Outputs

The simulator assumes a single-cycle datapath the supports all the included instructions. The program simulates the execution showing the values carried by all the wires of the datapath in the clock cycle. The simulator also keeps track of the register file contents and memory contents.

1.5. Assembler

An assembler is implemented and integrated to allow the user to supply programs in a suitable assembly language. The assembler supports labels, and pseudoinstructions (like move and blt).

1.6. GUI Application

The application is built as an educational GUI application. It includes and animated diagram of the datapath illustrating how the wire values propagate through it each clock cycle.

2. Datapath

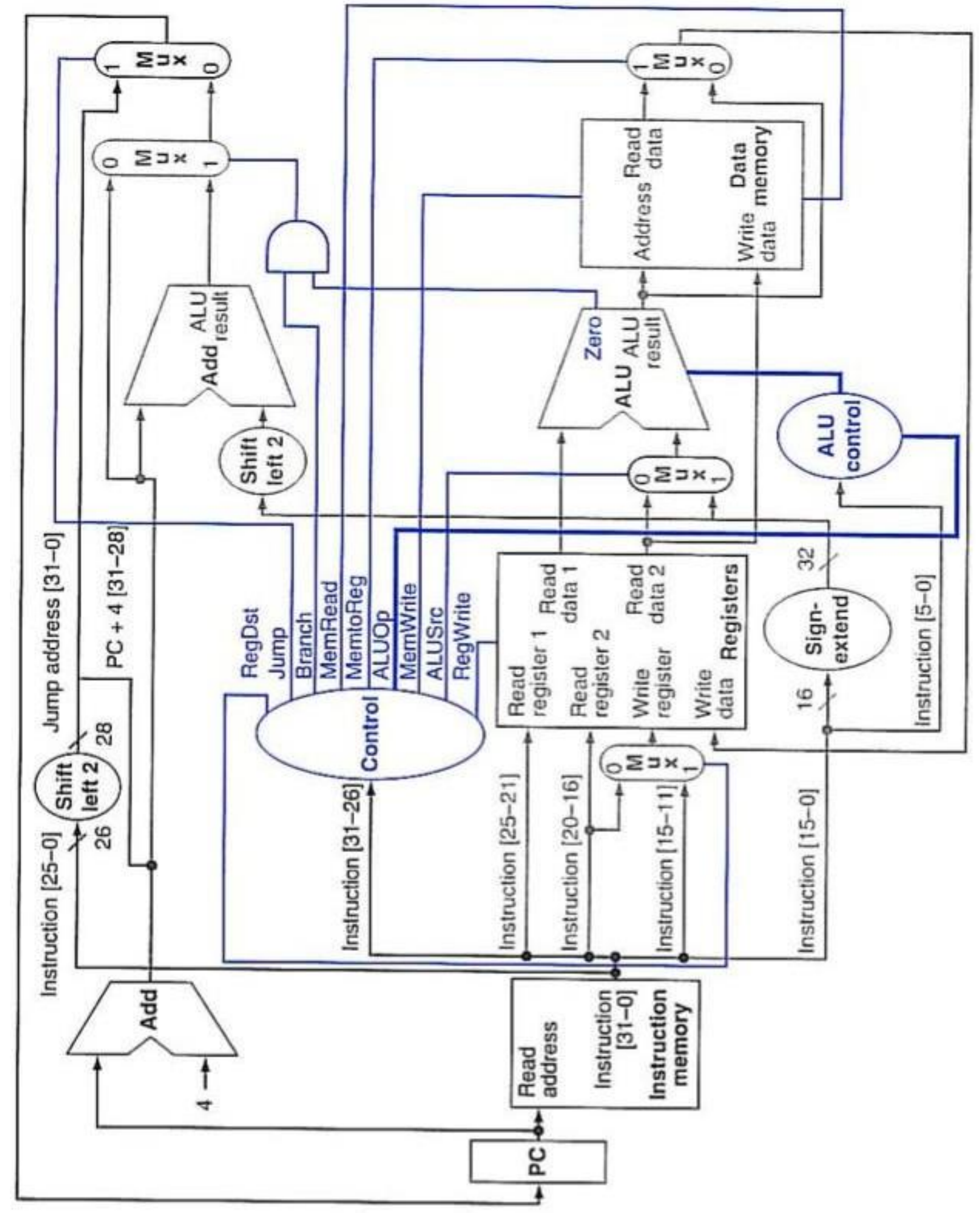


Fig 1: Single Cycle MIPS Datapath

3. CONTROL UNIT

3.1. Truth Table

Operation	Input	Output							
		RegDst	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
add	000000	1	0	0	0	10	0	0	1
addi	001000	0	0	0	0	00	0	1	1
sub	000000	1	0	0	0	10	0	0	1
lw	100011	0	0	1	1	00	0	1	1
sw	101011	X	0	0	X	00	1	1	0
lb	100000	0	0	1	1	00	0	1	1
lbu	100100	0	0	1	1	00	0	1	1
sb	101000	X	0	0	X	00	1	1	0
lh	100001	0	0	1	1	00	0	1	1
lhu	100101	0	0	1	1	00	0	1	1
sh	101001	X	0	0	X	00	1	1	0
lui	001111	0	0	1	1	00	0	1	1
sll	000000	1	0	0	0	10	0	0	1
nor	000000	1	0	0	0	10	0	0	1
srl	000000	1	0	0	0	10	0	0	1
and	000000	1	0	0	0	10	0	0	1
andi	001100	0	0	0	0	00	0	1	1
or	000000	1	0	0	0	10	0	0	1
ori	001101	0	0	0	0	00	0	1	1
beq	000100	X	1	0	X	01	0	0	0
bne	000101	X	1	0	X	01	0	0	0
j	000010	X	X	0	0	XX	0	X	0
jal	000011	0	X	0	0	XX	0	1	1
jr	000000	X	0	0	0	XX	0	0	0
slt	000000	1	0	0	0	10	0	0	1
slti	001010	0	0	0	0	00	0	1	1
sltu	000000	1	0	0	0	10	0	0	1
sltui	001011	0	0	0	0	00	0	1	1

Table 1: Control Unit Truth Table

3.2. Logic Diagram:

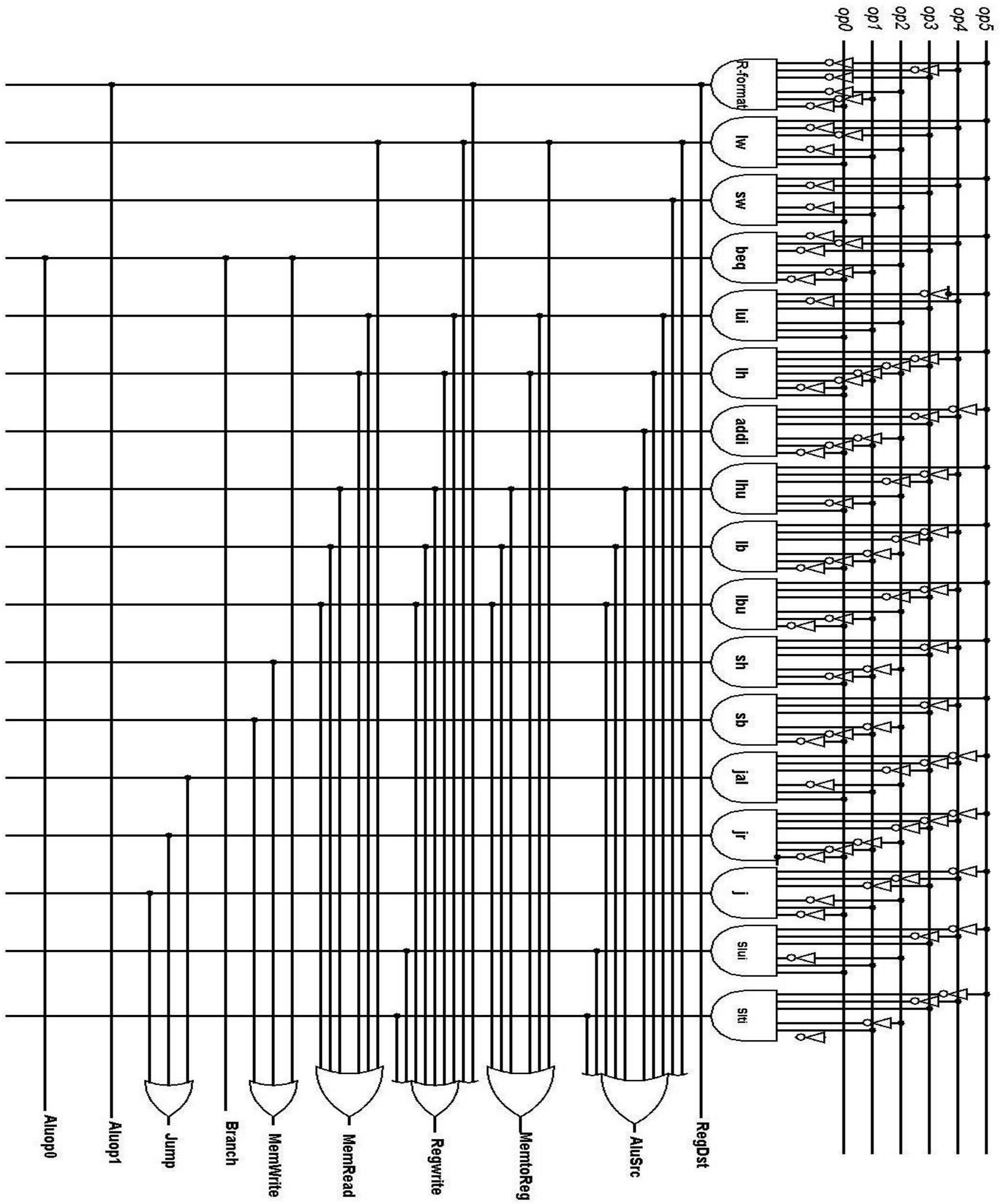


Fig 2: Control Unit Logic Diagram

4. ASSUMPTIONS

4.1. Memory Related Assumptions

For the Datapath (See Fig 1) we assumed that the memory is divided into three parts:

- A part that deals with the two instructions: lw, and sw.
- A part that deals with the instructions: lb, lbu, lh, lhu, and lui.
- A part that deals with the instructions: sb, and sh.

4.2. Assumptions Regarding Our Simulator Program

- We assume that the user will enter a value that is either less than or equal to the exact number of instructions that the user needs. Thusly, we assume that the memory array's size will be proportional to the number of instructions. Meaning that, if the user wishes to enter for example 10, we assume that the user will not enter a memory offset greater than four times the number of instructions for words and double the number for half words and equal to the number of instructions for byte , so the size of the memory array will be about 40 for word 20 for half word and 10 for byte.
- lb, lh have the same Datapath of lw.
- sb, sh have the same datapath of sw.
- Assumption maximum 4 labels to be used.
- All registers has an initial value of zero (including the stack pointer **$sp**).

5. USER GUIDE

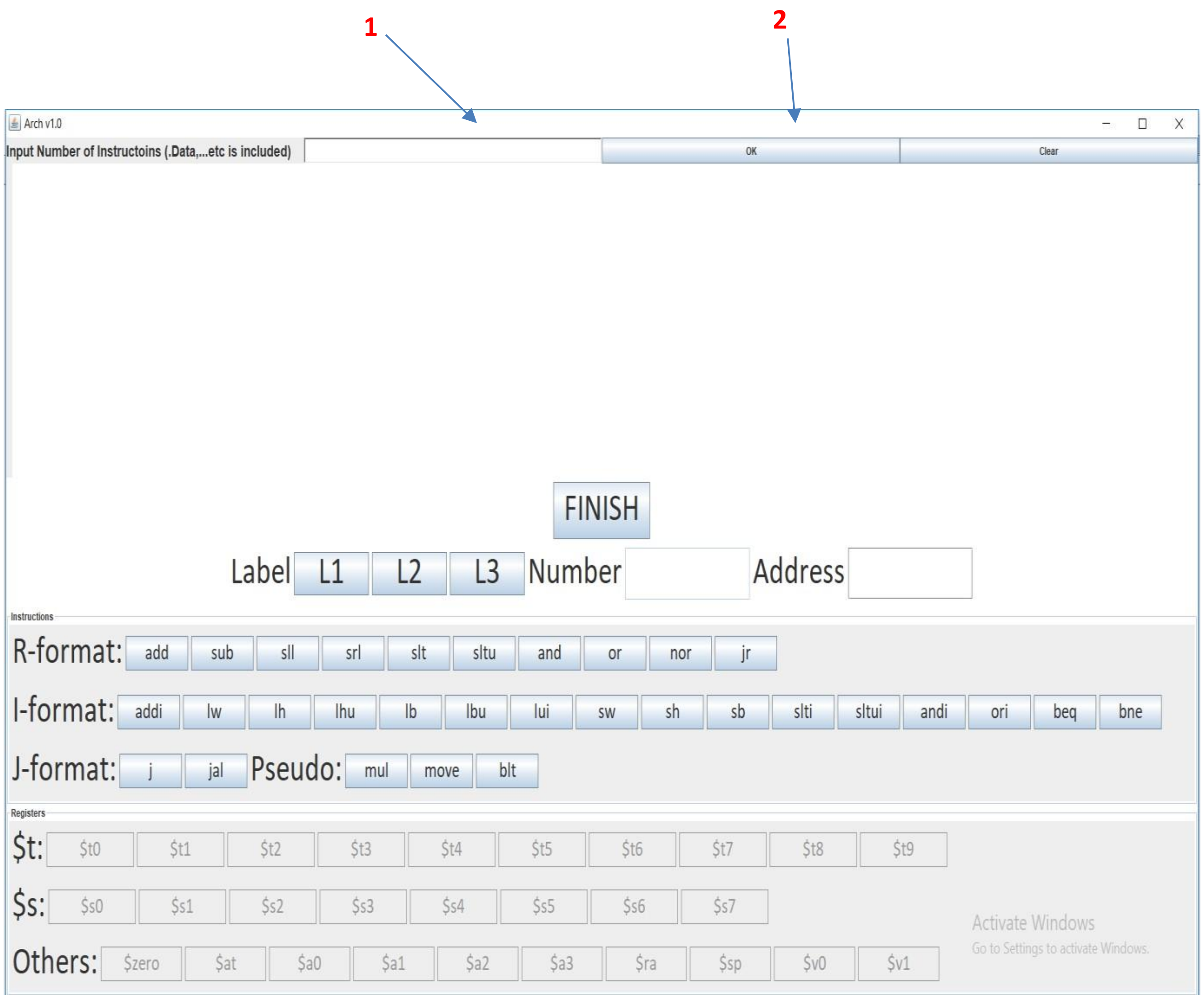
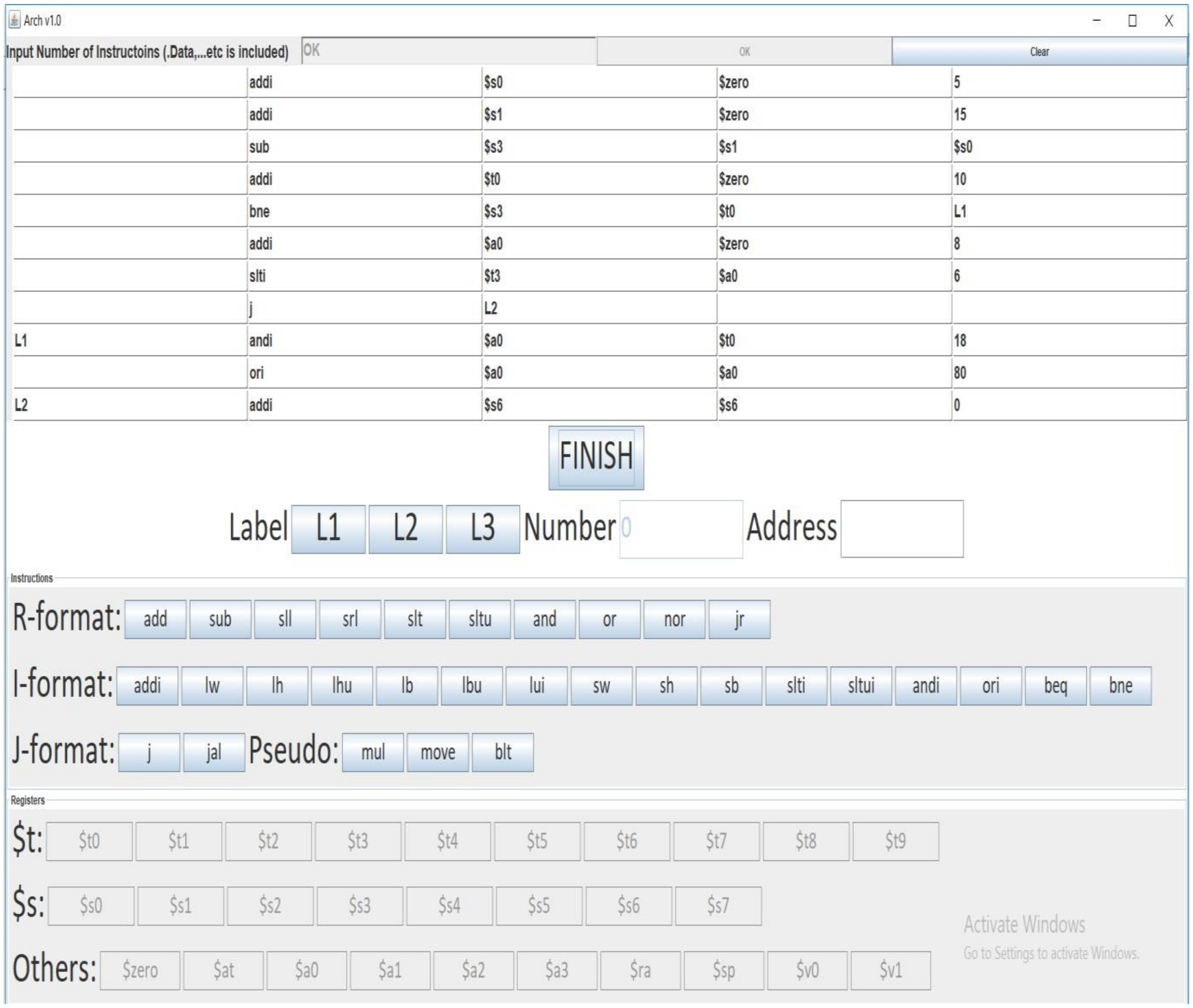

1
2
Arch v1.0
Input Number of Instructoins (.Data,...etc is included)
OK
Clear
FINISH
Label L1 L2 L3 Number Address
Instructions
R-format: add sub sll srl slt sltu and or nor jr
I-format: addi lw lh lhu lb lbu lui sw sh sb slti sltui andi ori beq bne
J-format: j jal Pseudo: mul move blt
Registers
$t: $t0 $t1 $t2 $t3 $t4 $t5 $t6 $t7 $t8 $t9
$s: $s0 $s1 $s2 $s3 $s4 $s5 $s6 $s7
Others: $zero $at $a0 $a1 $a2 $a3 $ra $sp $v0 $v1
Activate Windows
Go to Settings to activate Windows.

Fig 3: Screenshot of user interface

1. The number of instructions of the tested program should be written here
2. Press **OK**



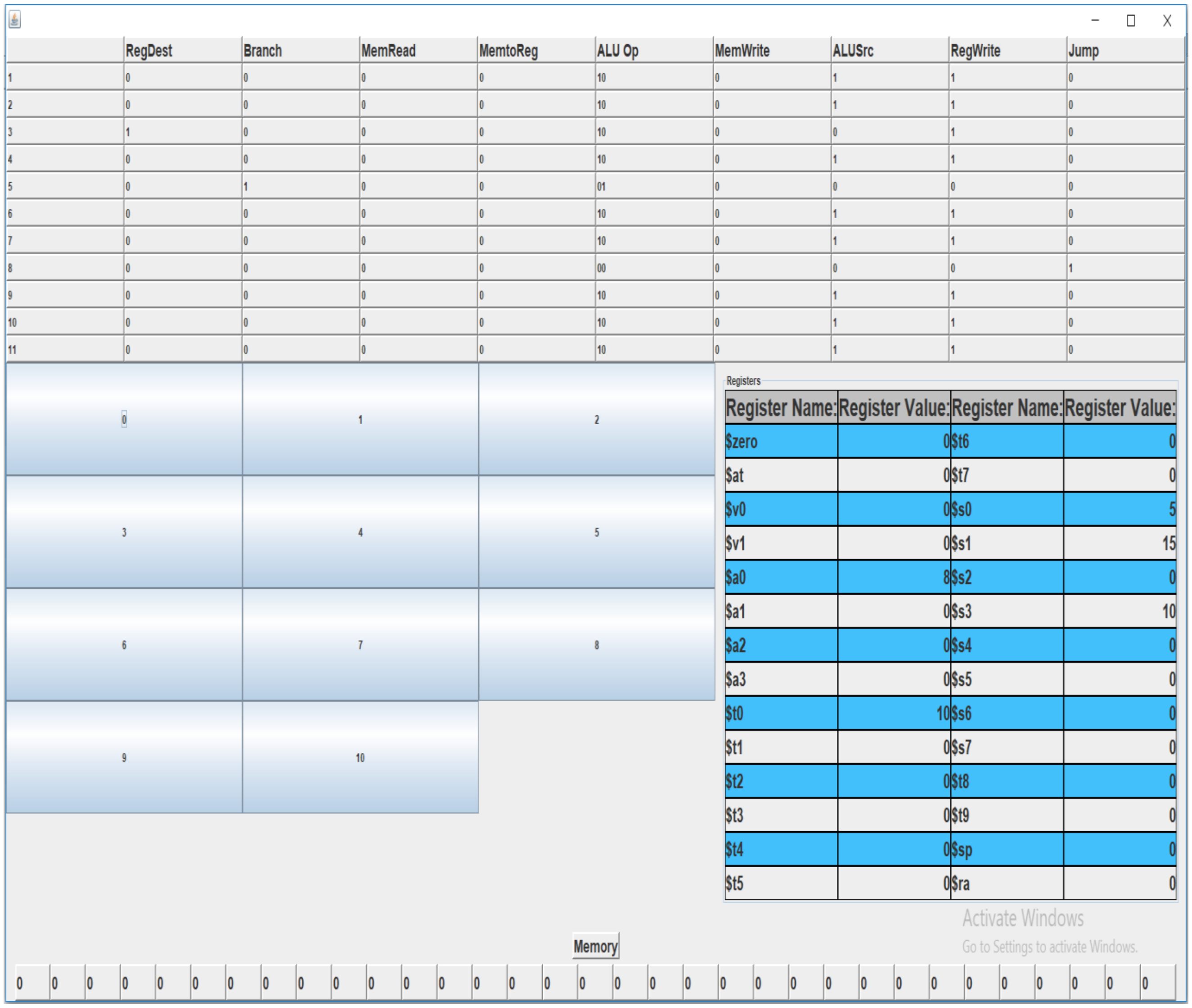
Arch v1.0
Input Number of Instructoins (.Data,...etc is included)
OK
OK
Clear

	addi	$s0	$zero	5
	addi	$s1	$zero	15
	sub	$s3	$s1	$s0
	addi	$t0	$zero	10
	bne	$s3	$t0	L1
	addi	$a0	$zero	8
	slti	$t3	$a0	6
	j	L2		
L1	andi	$a0	$t0	18
	ori	$a0	$a0	80
L2	addi	$s6	$s6	0

FINISH
Label L1 L2 L3 Number 0 Address
Instructions
R-format: add sub sll srl slt sltu and or nor jr
I-format: addi lw lh lhu lb lbu lui sw sh sb slti sltui andi ori beq bne
J-format: j jal Pseudo: mul move blt
Registers
$t: $t0 $t1 $t2 $t3 $t4 $t5 $t6 $t7 $t8 $t9
$s: $s0 $s1 $s2 $s3 $s4 $s5 $s6 $s7
Others: $zero $at $a0 $a1 $a2 $a3 $ra $sp $v0 $v1
Activate Windows
Go to Settings to activate Windows.

Fig 4: Screenshot of user interface of written program

3.The user should write the wanted program by pressing on the buttons -except for entering numbers it should be written in the textbox next to number label (followed by an enter). After writing the main program the user should enter a number in the address textbox which defines the address of the first instruction in the instruction memory.



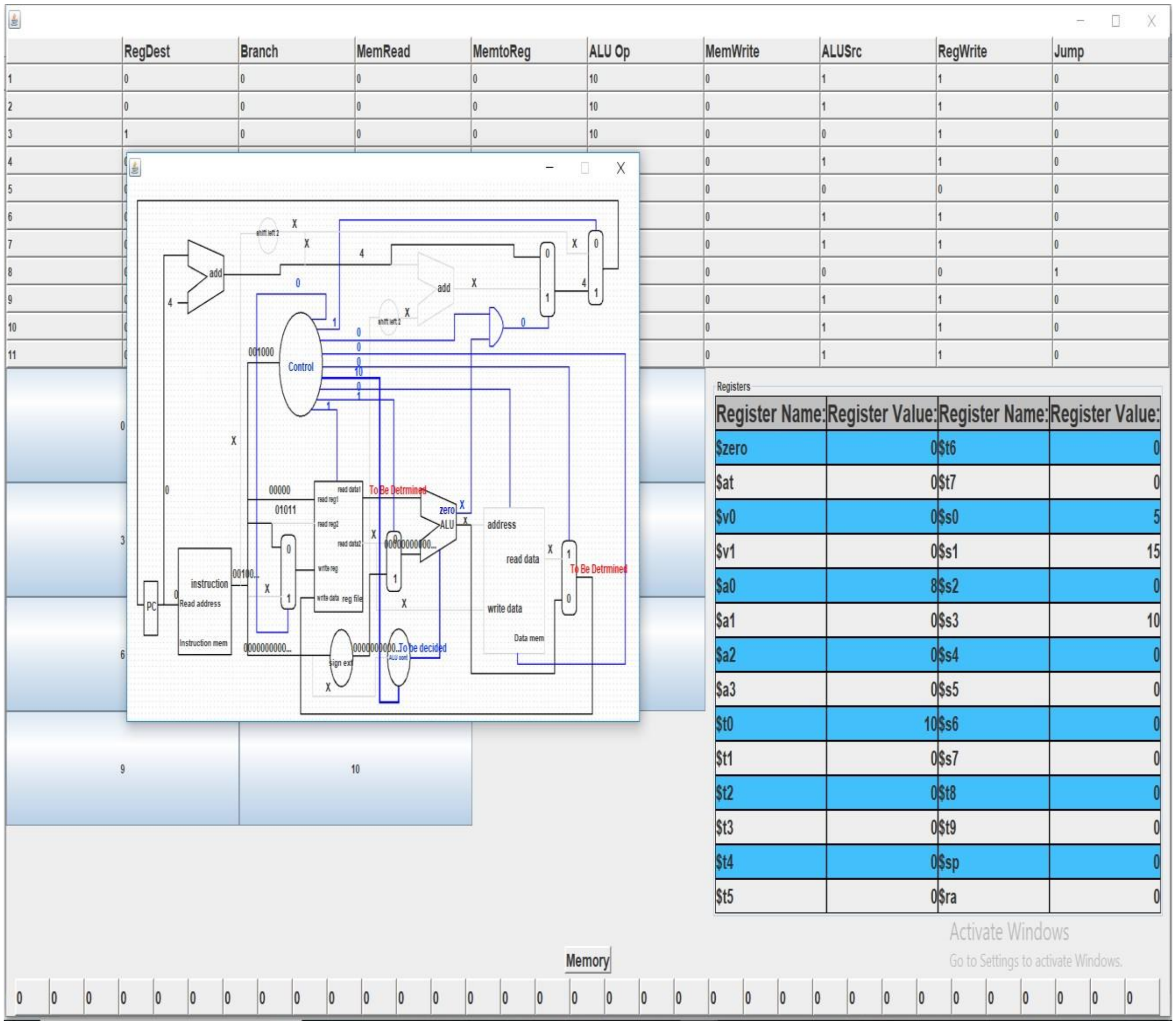
	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	0	0	0	0	10	0	1	1	0
3	1	0	0	0	10	0	0	1	0
4	0	0	0	0	10	0	1	1	0
5	0	1	0	0	01	0	0	0	0
6	0	0	0	0	10	0	1	1	0
7	0	0	0	0	10	0	1	1	0
8	0	0	0	0	00	0	0	0	1
9	0	0	0	0	10	0	1	1	0
10	0	0	0	0	10	0	1	1	0
11	0	0	0	0	10	0	1	1	0

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	5
$v1	0	$s1	15
$a0	8	$s2	0
$a1	0	$s3	10
$a2	0	$s4	0
$a3	0	$s5	0
$t0	10	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig 5: Screenshot of program after pressing finish

After pressing **Finish** a table of registers appears containing the edited registers' current values; the memory values is also updated, the ALU control values for every instruction appear and also buttons that show the datapath of the number of instruction presse



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	0	0	0	0	10	0	1	1	0
3	1	0	0	0	10	0	0	1	0
4						0	1	1	0
5						0	0	0	0
6						0	1	1	0
7						0	1	1	0
8						0	0	0	1
9						0	1	1	0
10						0	1	1	0
11						0	1	1	0

Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	5
$v1	0	$s1	15
$a0	8	$s2	0
$a1	0	$s3	10
$a2	0	$s4	0
$a3	0	$s5	0
$t0	10	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

Activate Windows
Go to Settings to activate Windows.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

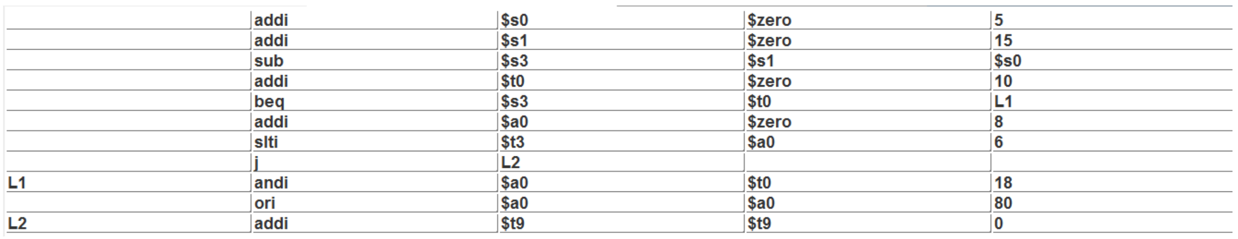
Fig 6: Screenshot of the datapath of a certain instruction

When an instruction number is pressed its datapath appears. The number of buttons should be equal to the number of instructions entered, and each instruction should show a picture when pressed.

If the user want to re-use the program, they must close it and then open it again.

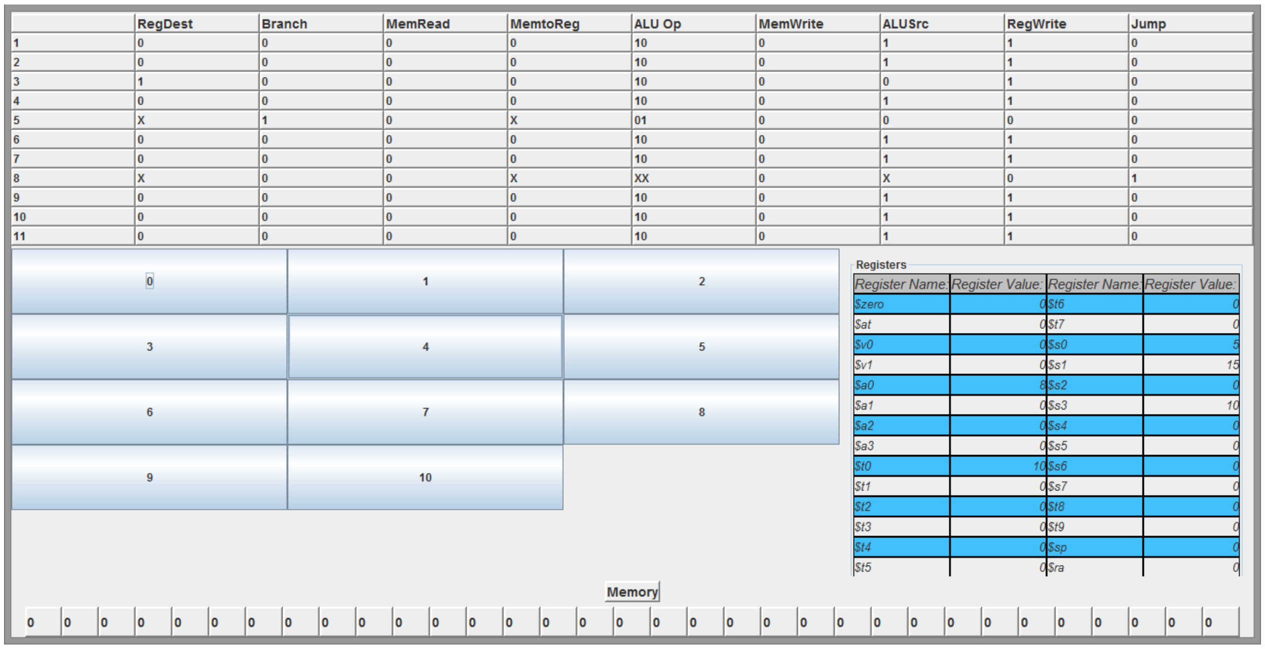
6. TEST CASES

6.1. Program #1



	addi	$s0	$zero	5
	addi	$s1	$zero	15
	sub	$s3	$s1	$s0
	addi	$t0	$zero	10
	beq	$s3	$t0	L1
	addi	$a0	$zero	8
	slti	$t3	$a0	6
	j	L2		
L1	andi	$a0	$t0	18
	ori	$a0	$a0	80
L2	addi	$t9	$t9	0

Fig 7: Program #1



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	0	0	0	0	10	0	1	1	0
3	1	0	0	0	10	0	0	1	0
4	0	0	0	0	10	0	1	1	0
5	X	1	0	X	01	0	0	0	0
6	0	0	0	0	10	0	1	1	0
7	0	0	0	0	10	0	1	1	0
8	X	0	0	X	XX	0	X	0	1
9	0	0	0	0	10	0	1	1	0
10	0	0	0	0	10	0	1	1	0
11	0	0	0	0	10	0	1	1	0

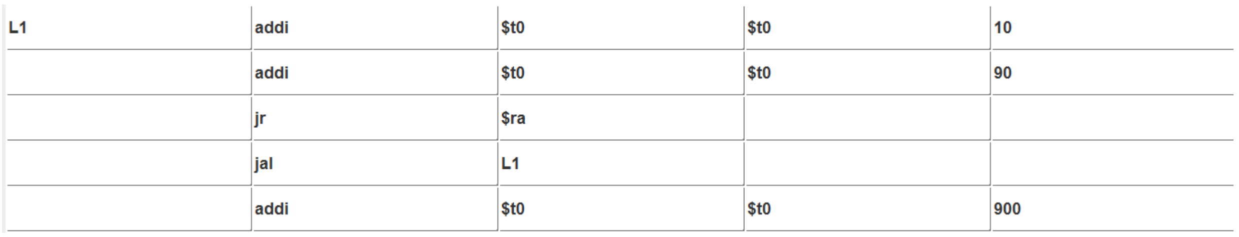
Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	5
$v1	0	$s1	15
$a0	8	$s2	0
$a1	0	$s3	10
$a2	0	$s4	0
$a3	0	$s5	0
$t0	10	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

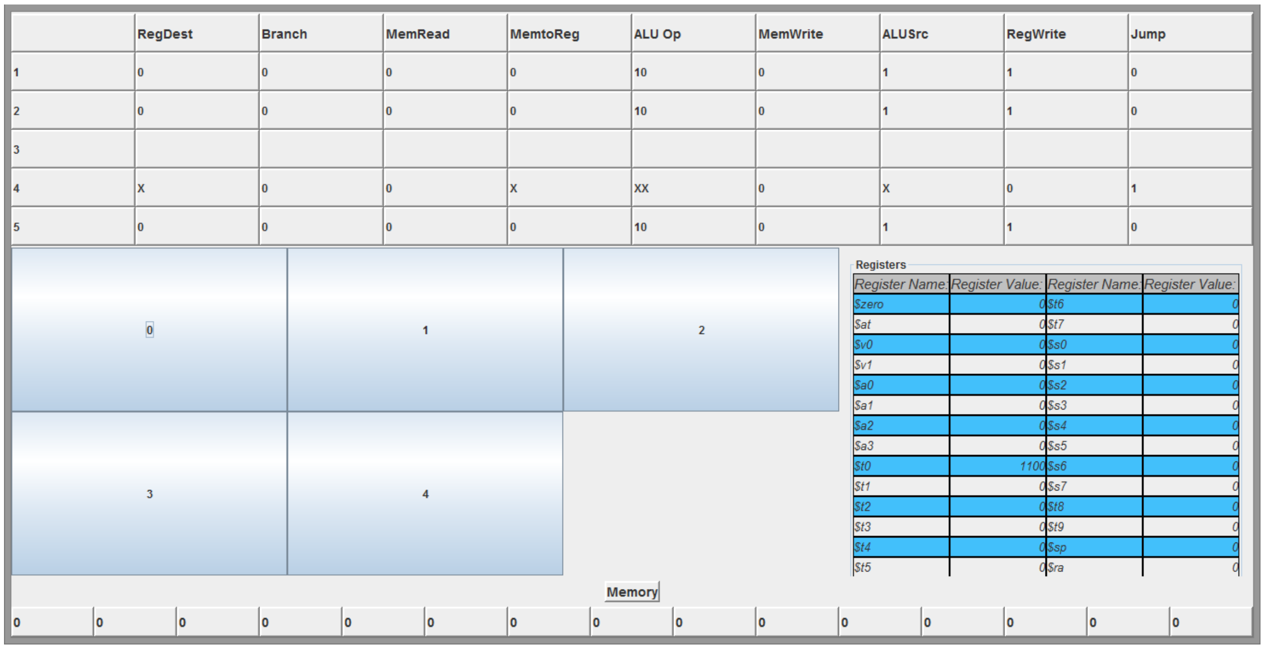
Fig 8: Output #1

6.2. Program #2:



L1	addi	$t0	$t0	10
	addi	$t0	$t0	90
	jr	$ra		
	jal	L1		
	addi	$t0	$t0	900

Fig 9: Program #2



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	0	0	0	0	10	0	1	1	0
3									
4	X	0	0	X	XX	0	X	0	1
5	0	0	0	0	10	0	1	1	0

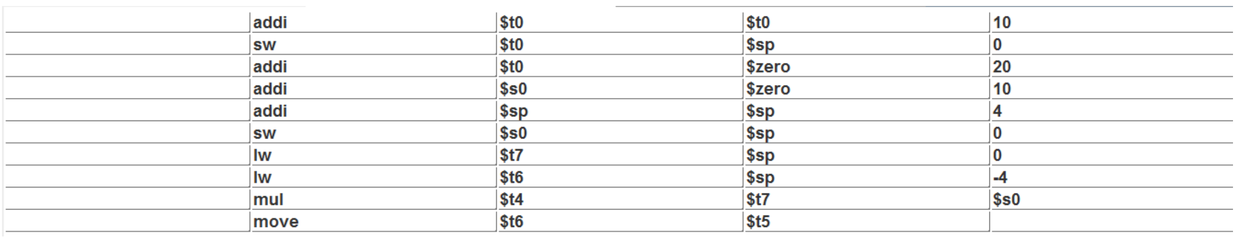
Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	0
$v1	0	$s1	0
$a0	0	$s2	0
$a1	0	$s3	0
$a2	0	$s4	0
$a3	0	$s5	0
$t0	1100	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

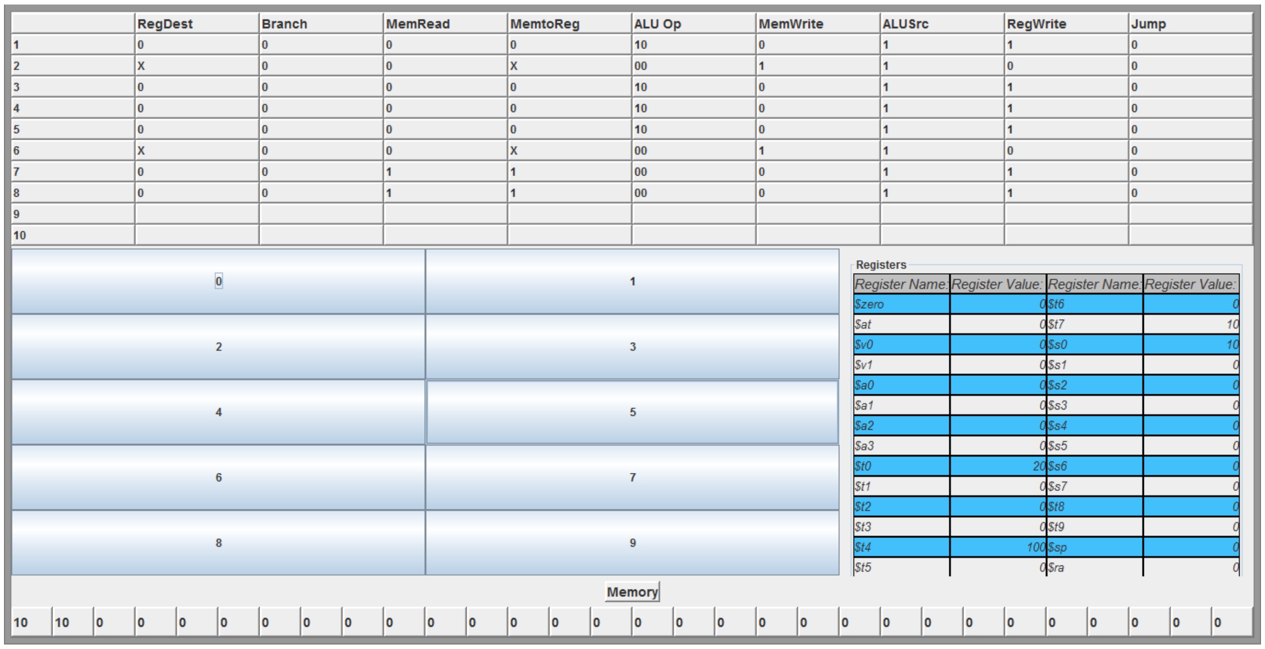
Fig 10: Output #2

6.3. Program #3



	addi	$t0	$t0	10
	sw	$t0	$sp	0
	addi	$t0	$zero	20
	addi	$s0	$zero	10
	addi	$sp	$sp	4
	sw	$s0	$sp	0
	lw	$t7	$sp	0
	lw	$t6	$sp	-4
	mul	$t4	$t7	$s0
	move	$t6	$t5	

Fig 11: Program #3



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	X	0	0	X	00	1	1	0	0
3	0	0	0	0	10	0	1	1	0
4	0	0	0	0	10	0	1	1	0
5	0	0	0	0	10	0	1	1	0
6	X	0	0	X	00	1	1	0	0
7	0	0	1	1	00	0	1	1	0
8	0	0	1	1	00	0	1	1	0
9									
10									

Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	10
$v0	0	$s0	10
$v1	0	$s1	0
$a0	0	$s2	0
$a1	0	$s3	0
$a2	0	$s4	0
$a3	0	$s5	0
$t0	20	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	100	$sp	0
$t5	0	$ra	0

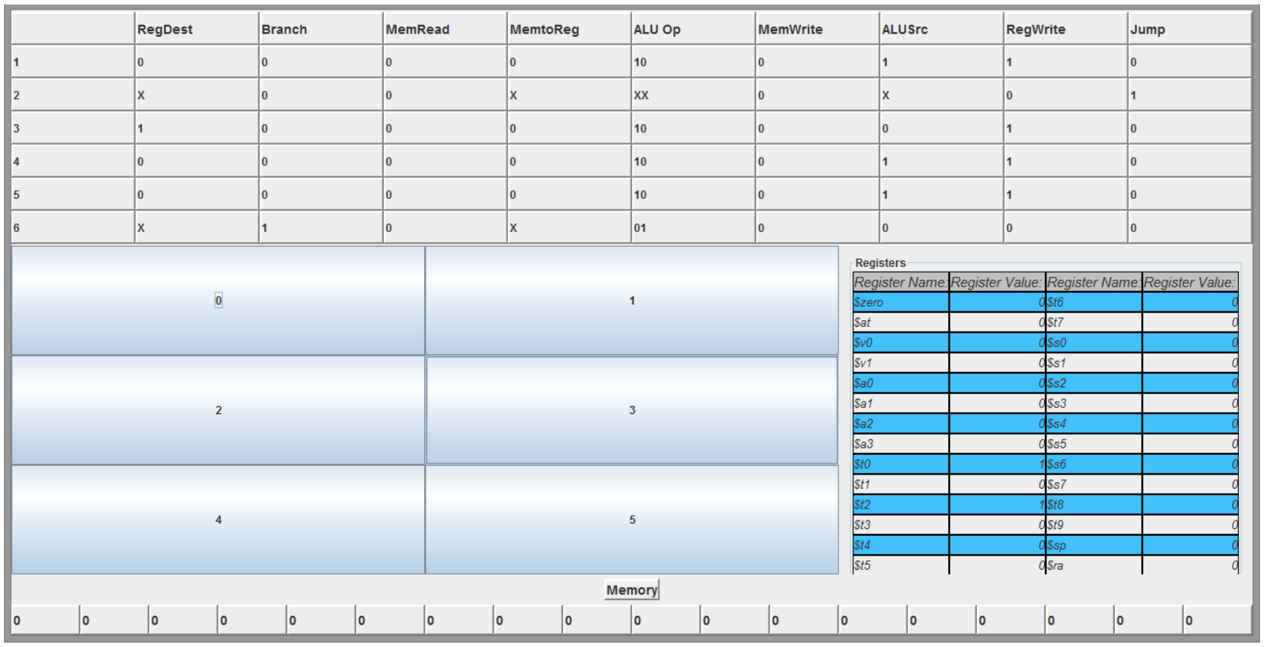
10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 12: Output #3

6.4. Program #4

	addi	$t0	$zero	0
	j	L1		
L2	add	$s0	$s0	$s1
	addi	$t0	$t0	1
L1	slti	$t2	$t0	10
	bne	$t2	$zero	L2

Fig 13: Program #4



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2	X	0	0	X	XX	0	X	0	1
3	1	0	0	0	10	0	0	1	0
4	0	0	0	0	10	0	1	1	0
5	0	0	0	0	10	0	1	1	0
6	X	1	0	X	01	0	0	0	0

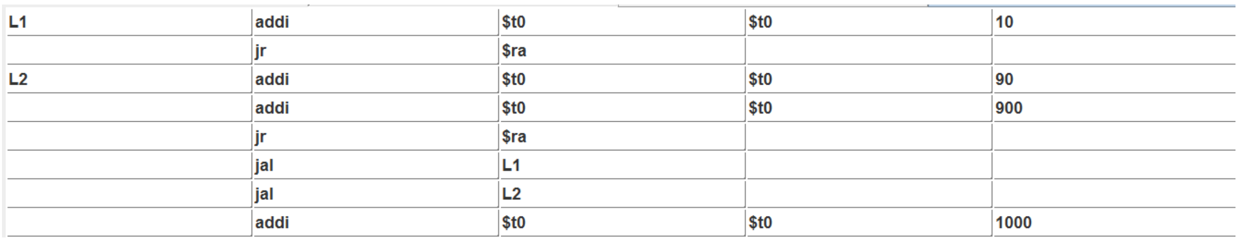
Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	0
$v1	0	$s1	0
$a0	0	$s2	0
$a1	0	$s3	0
$a2	0	$s4	0
$a3	0	$s5	0
$t0	1	$s6	0
$t1	0	$s7	0
$t2	1	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

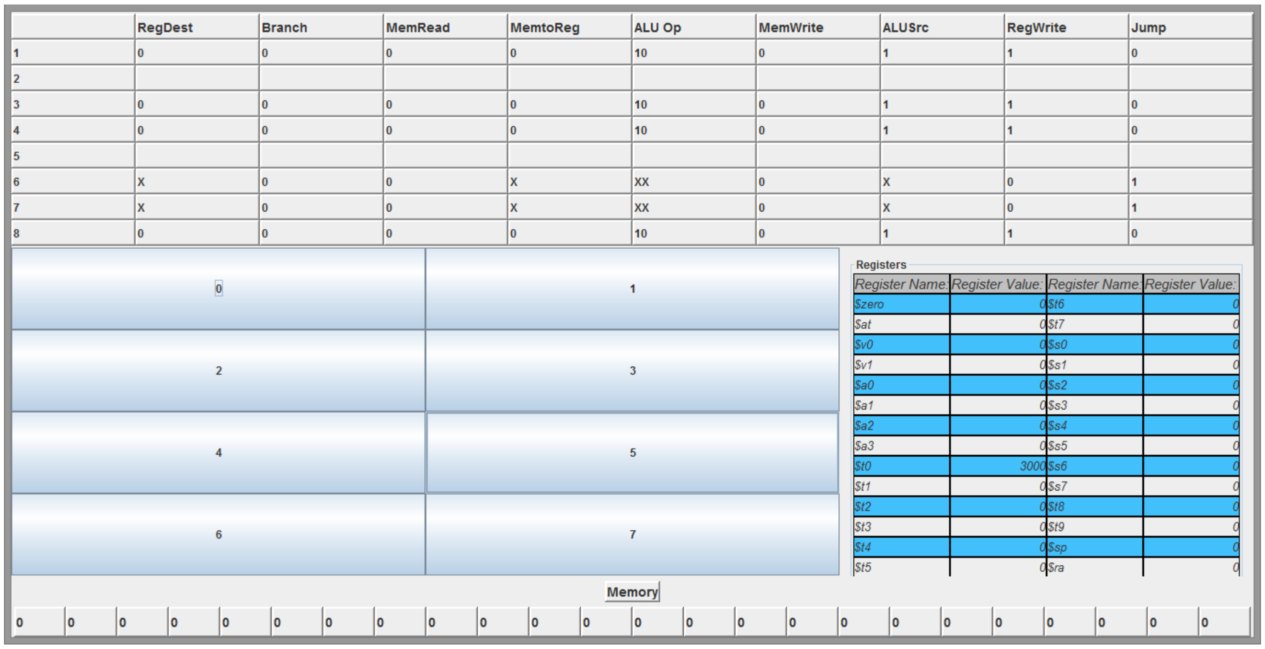
Fig 14: Output #4

6.5. Program #5



L1	addi	$t0	$t0	10
	jr	$ra		
L2	addi	$t0	$t0	90
	addi	$t0	$t0	900
	jr	$ra		
	jal	L1		
	jal	L2		
	addi	$t0	$t0	1000

Fig 15: Program #5



	RegDest	Branch	MemRead	MemtoReg	ALU Op	MemWrite	ALUSrc	RegWrite	Jump
1	0	0	0	0	10	0	1	1	0
2									
3	0	0	0	0	10	0	1	1	0
4	0	0	0	0	10	0	1	1	0
5									
6	X	0	0	X	XX	0	X	0	1
7	X	0	0	X	XX	0	X	0	1
8	0	0	0	0	10	0	1	1	0

Registers

Register Name:	Register Value:	Register Name:	Register Value:
$zero	0	$t6	0
$at	0	$t7	0
$v0	0	$s0	0
$v1	0	$s1	0
$a0	0	$s2	0
$a1	0	$s3	0
$a2	0	$s4	0
$a3	0	$s5	0
$t0	3000	$s6	0
$t1	0	$s7	0
$t2	0	$t8	0
$t3	0	$t9	0
$t4	0	$sp	0
$t5	0	$ra	0

Memory

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 16: Output #5

7. WORK SUMMARY

- **Alfred Maheeb:** worked on input GUI, buttons, implementing function codes and buttons' action listeners.
- **Andre Osama:** worked on the report, and the files representing the test programs and data.
- **Andrew Raafat:** was responsible for building the registers database, the memory database and writing the program functions.
- **Daniella George:** worked on creating the GUI for the table and the report.
- **Monica Mourad:** worked on input GUI, buttons, implementing function codes and buttons' action listeners.
- **Nayer Nabil:** worked on functions, buttons, memory, output GUI and helped in testing.