FACULTY OF ENGINEERING

AIN SHAMS UNIVERSITY

CSE 116 : COMPUTER ARCHITECTURE

MIPS DATAPATH SIMULATOR

MAY 1, 2018

AIN SHAMS UNIVERSITY – FACULTY OF ENGINEERING
1 Al-Sarayyat st, Abbassiya, Cairo 11517, Egypt

Mips Datapath Simulator

Submitted by:

Abdelrahman Ibrahim ELGhamry
Ghamry98@hotmail.com
16P3043@eng.asu.edu.eg

Hossam ELDin Khaled Mohmed
hossampen97@gmail.com
16P3025@eng.asu.edu.eg

Abdelrahman Amr Issawi
aid-issawi@hotmail.com
16P6001@eng.asu.edu.eg

Submitted to:

Prof. Dr. Chrief Salama

MAY 2018

A report for Computer Architecture Course codded CSE116 with the requirements of Ain Shams University

Mips Datapath Simulator
Ain Shams University
Faculty of Engineering
Abbassiya
Cairo 11517, Egypt

Dear Prof. Dr. Chrief Salama,

We are submitting the attached report, done at your request, entitled ***trux datapath simulator*** to include the simulator datapath and all its components, how it works, how it's developed and some information that we would share with you – Our Dr.Chrief – and also with our Group. This report is assigned in 1 May 2018.

If you have any questions, please do not hesitate to contact us

Thank you for your co-operation during the lectures.

Sincerely yours,

Ghamry Hossam Issawi

Abdelrahman Ibrahim ELGhamry,

Hossam ELDin Khaled,

Abdelrahman Amr Issawi, Students,

Faculty of Engineering,

Ain shams University.

TABLE OF CONTENTS

1.0 BRIEF DESCRIPTION

1.1 Implementation

Classes:

- **ArchProj**

Main class that contains the gui initialization.
Methods: main(), reset().

- **ALU**

Performs all ALU operations.
Methods: getRes(), getZeroFlag().

- **ALUControl**

Generates ALU control signals.
Methods: checkFunct(), getALUControl().

- **Control**

Controls all other components by generating control signals.
Methods: getAluOp(), getAluSrc1(), getAluSrc2(), getAluSrcMux(), getBranch(), getJump(), getJumpAndLink(), getJumpReg(), getLoadByte(), getMemRead(), getMemToReg(), getMemUnsigned(), getMemWrite(), getRegDstMux(), getRegWrite(), getStoreByte(), getInstructionName().

- **InstMem**

Contains all instructions to be executed, and the program counter.
Methods: enterCode(), getPc(), instFetch(), setPc().

- **Instruction**

Divides the instruction and defines all its possible fields.
Methods: getOpCode, getRs(), getRt(), getRd(), getShamt(), getJumpAdd(), getFunct(), getConstant().

- **RegisterFile**

Represents the thirty-two registers of the mips processor.
Methods: getRegisterName(), getRegisterValue(), setRegisterValue().

- **Memory**

Data storage for the executing program.
Methods: getSize(), setSize(), getValue(), setValue(), load(), store(), toString().

- **GUI**

Contains all the user interface components, interacts with all the datapath components and represents the project in an interactive way.
Methods: getMemSize(), imageRefresh(), regRefresh(), runCode(), getMemToRegMuxOutput().

- **JNumberTextField**

It extends JTextField class and it only accepts digits.

- **JBinaryTextArea**

It extends JTextArea class and it only accepts binary digits.

- **memFrame**

Represents all values in the data memory.
Methods: memRefresh(), setMemory().

- **traceFrame**

Represents all values in the datapath.
Methods: setControlValues(), setWireValues(), traceRefresh(),

- **resFrame**

Shows that the program is successfully built.
Methods: setRes().

- **helpFrame**

Shows all supported instructions.

- **aboutFrame**

Shows the project's credits.

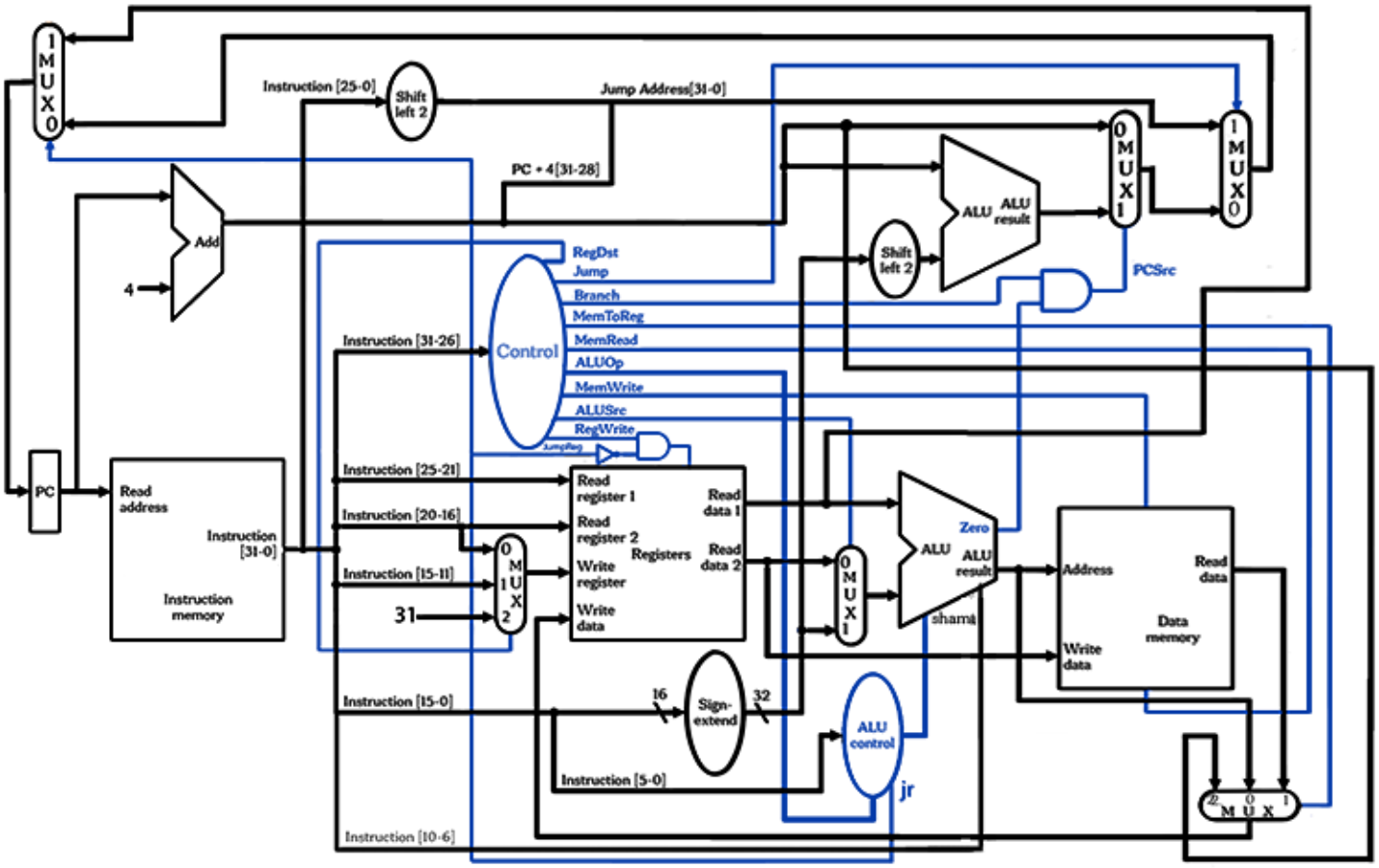
- **contactFrame**

Shows developers contact info.

1.2 Bonus Features

- Building the application as a GUI application.
- Building the application as an educational GUI application. In this case the GUI should include an animated diagram of the datapath illustrating how the wire values propagate through it each clock cycle.

2.0 DATAPATH

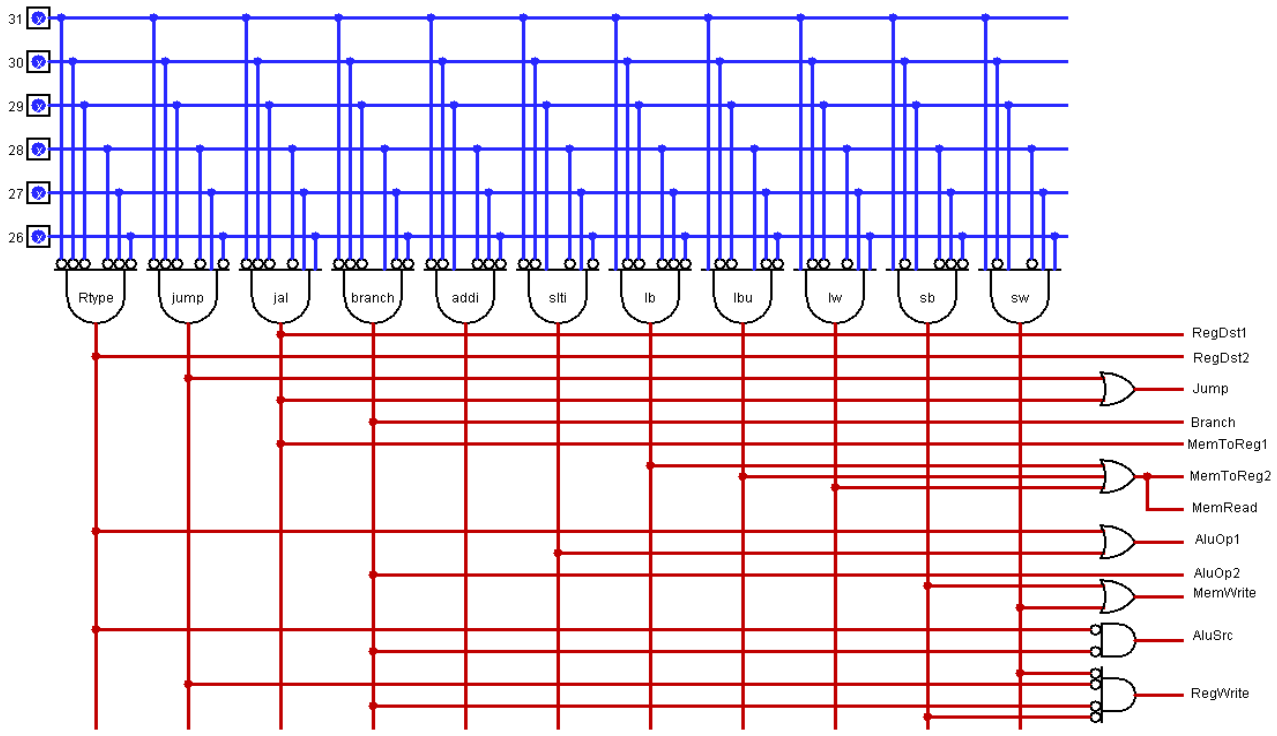


3.0 CONTROL UNIT TRUTH TABLE AND LOGIC DIAGRAM

3.1 Truth Table

OpCode	Reg Dst1	Reg Dst2	Jump	Branch	Memto Reg1	Memto Reg2	Mem Read	AluOp1	AluOp2	Mem Write	AluSrc	Reg Write
000000	0	1	0	0	0	0	0	1	0	0	0	1
000010	x	x	1	0	x	x	0	x	x	0	x	0
000011	1	0	1	0	1	0	0	x	x	0	x	1
000100	x	x	0	1	x	x	0	0	1	0	0	0
001000	0	0	0	0	0	0	0	0	0	0	1	1
001010	0	0	0	0	0	0	0	1	0	0	1	1
100000	0	0	0	0	0	1	1	0	0	0	1	1
100100	0	0	0	0	0	1	1	0	0	0	1	1
100011	0	0	0	0	0	1	1	0	0	0	1	1
101000	x	x	0	0	x	x	0	0	0	1	1	0
101011	x	x	0	0	x	x	0	0	0	1	1	0

3.2 Logic Diagram



4.0 ADOPTED ASSUMPTIONS

1. The ALU Control detects the jr instruction and enables the jr control signal.

2. Since jr is an R-Format instruction so the normal register write signal is enabled. Therefore NOT gate (JumpReg) is added to control the register write signal to prevent writing in register file in case jr instructions is being executed.

3. A new Mux is added to select the next pc value, to choose between jr address and the normal path.

4. The regDst Mux has an extra input which is thiry-one, to select ra register to write in it (pc+4), while jal instruction is being executed, so the regDst signals are now two bits.

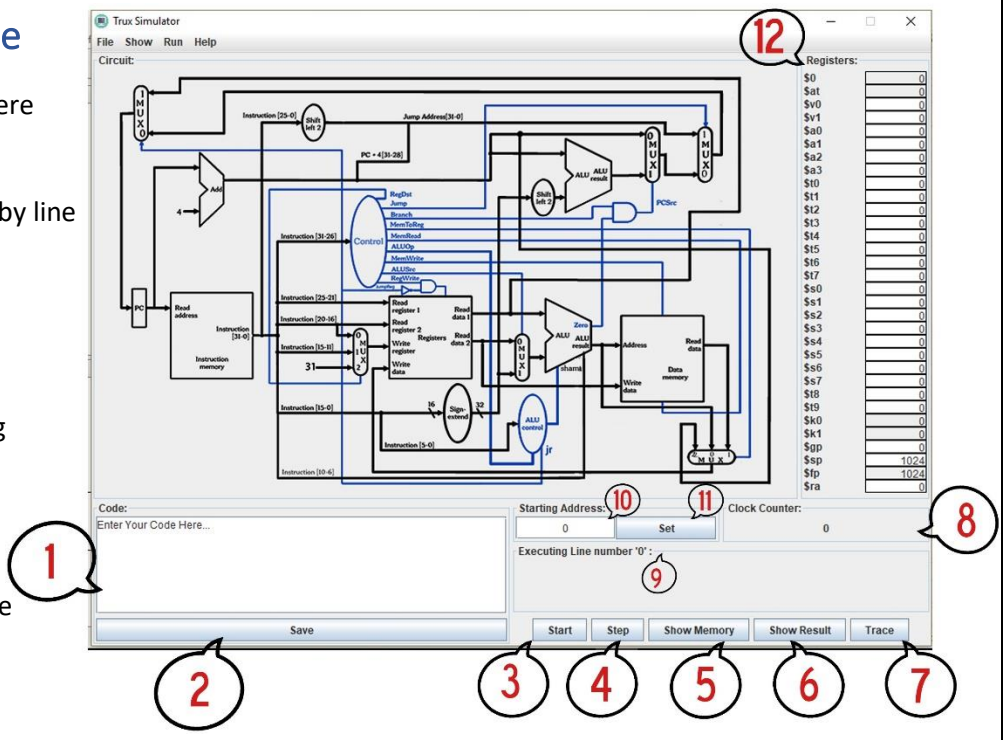
5. The memToReg Mux has an extra input which is the (pc+4), so the memToReg signals are now two bits.

6. Instruction [10-6] (shamt) is wired to the ALU, to perform shift instruction.

5.0 USER GUIDE

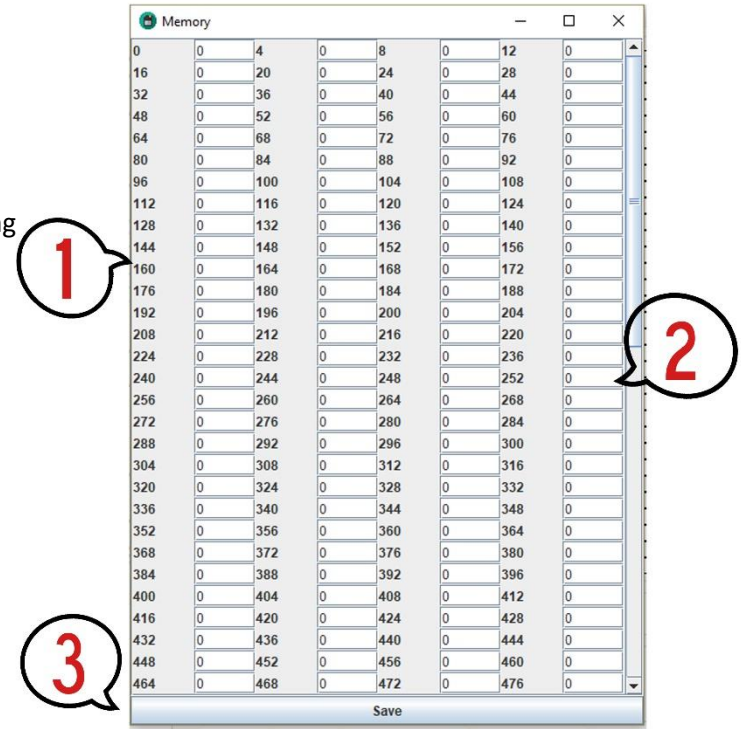
5.1 Main Frame

1. Enter your code here
2. Save the code
3. Run all the code
4. Execute code line by line
5. Show memory
6. Show final result
7. Show trace frame
8. The clock counter
9. Line of execution
10. Enter your starting Address here
11. Set the starting address
12. Set your register values here, before starting execution



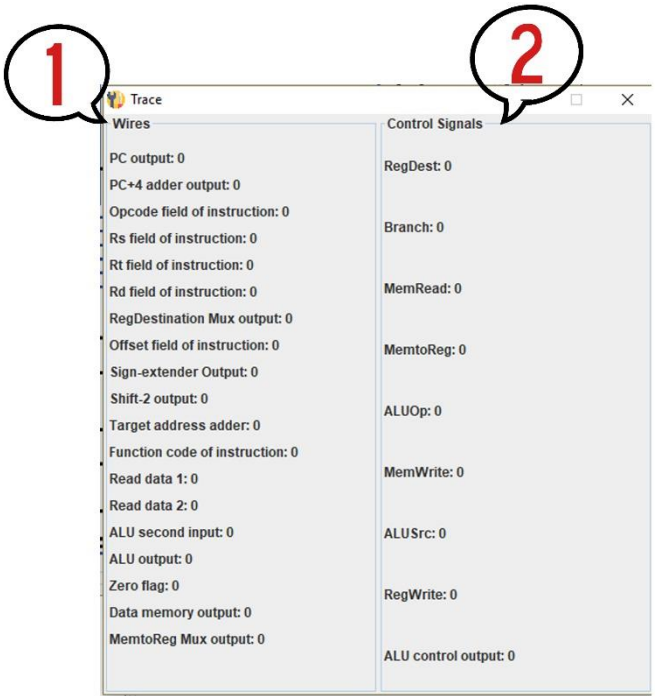
5.2 Memory Frame

1. Memory addressing
2. Memory values, you can initialize them before starting the program execution
3. Save your memory values before starting The program execution



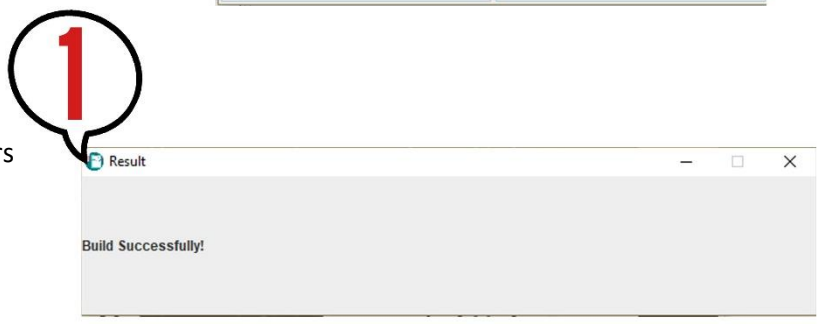
5.3 Trace Frame

1. Shows all the datapath wires values instruction by instruction
2. All the control signal values instruction by instruction



5.4 Result Frame

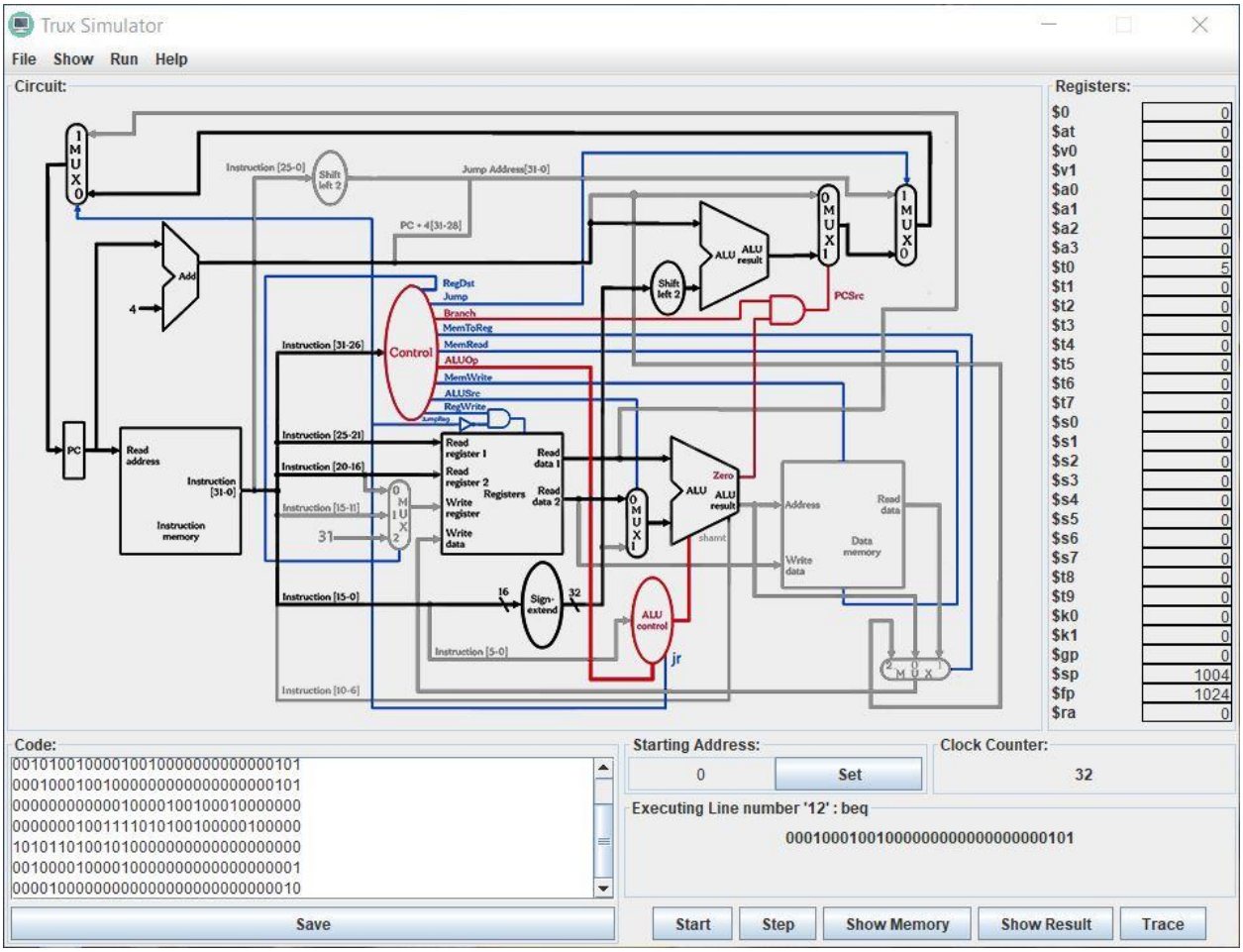
1. "Build Successfully!" appears when the program finishes execution

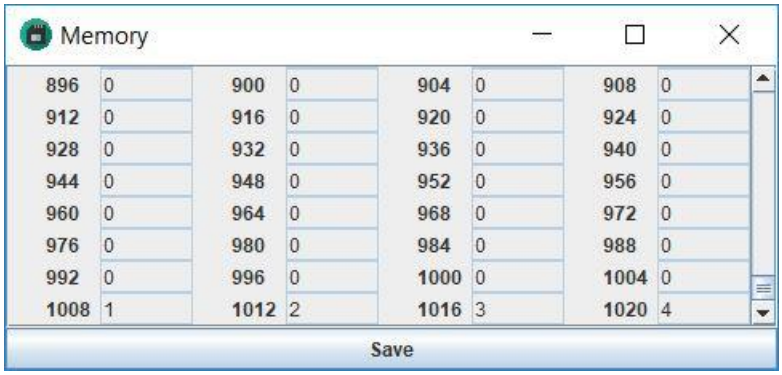


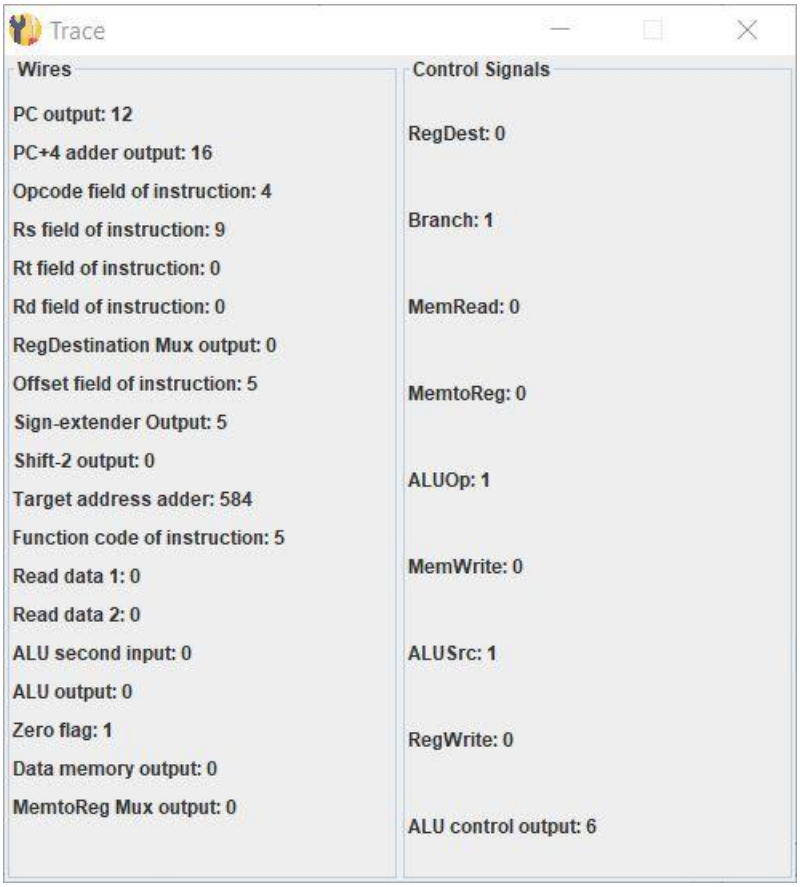
6.0 TEST CASES

6.1 Test Case 1

Assembly	Machine
addi $sp, $sp, -20	00100011101111101111111111101100
addi $t0, $0, 1	00100000000010000000000000000001
loop:	
slti $t1, $t0, 5	00101001000010010000000000000101
beq $t1, $0, exit	00010001001000000000000000000101
sll $t1, $t0, 2	00000000000010000100100010000000
add $t1, $t1, $sp	00000001001111010100100000100000
sw $t0, 0($t1)	10101101001010000000000000000000
addi $t0, $t0, 1	00100001000010000000000000000001
j loop	00001000000000000000000000000010
exit:	




Memory
896 0 900 0 904 0 908 0
912 0 916 0 920 0 924 0
928 0 932 0 936 0 940 0
944 0 948 0 952 0 956 0
960 0 964 0 968 0 972 0
976 0 980 0 984 0 988 0
992 0 996 0 1000 0 1004 0
1008 1 1012 2 1016 3 1020 4
Save

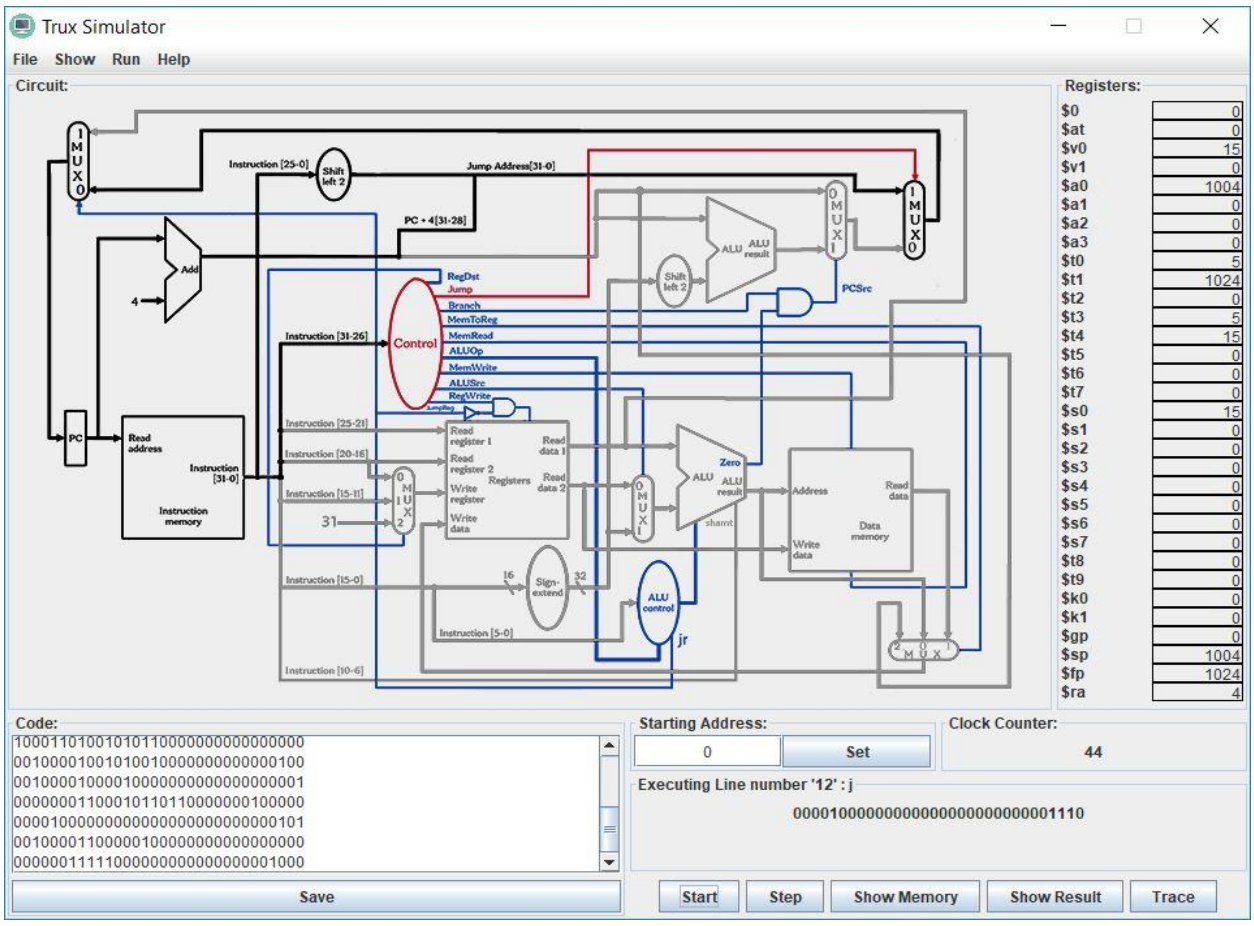

Trace
Wires
PC output: 12
PC+4 adder output: 16
Opcode field of instruction: 4
Rs field of instruction: 9
Rt field of instruction: 0
Rd field of instruction: 0
RegDestination Mux output: 0
Offset field of instruction: 5
Sign-extender Output: 5
Shift-2 output: 0
Target address adder: 584
Function code of instruction: 5
Read data 1: 0
Read data 2: 0
ALU second input: 0
ALU output: 0
Zero flag: 1
Data memory output: 0
MemtoReg Mux output: 0
Control Signals
RegDest: 0
Branch: 1
MemRead: 0
MemtoReg: 0
ALUOp: 1
MemWrite: 0
ALUSrc: 1
RegWrite: 0
ALU control output: 6

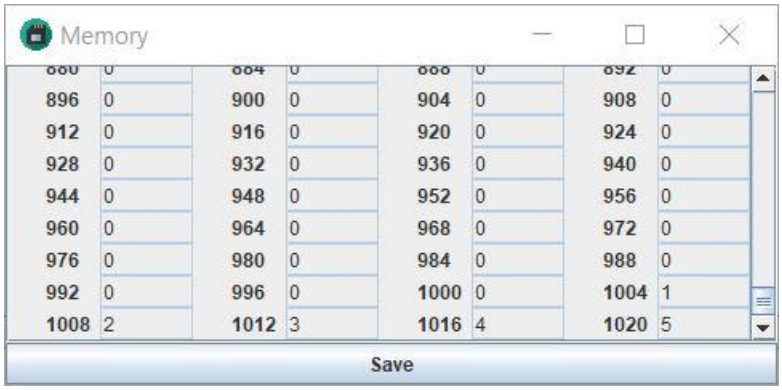

Result
Build Successfully!

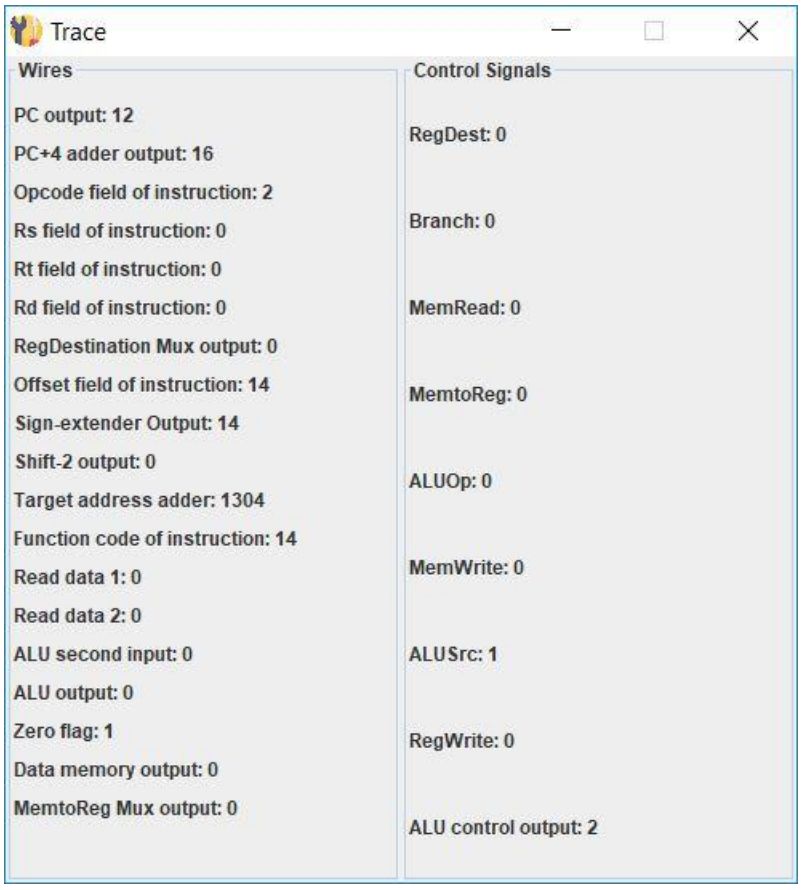
6.2 Test Case 2

Initially starting with: $sp = 1004, Memory locations 1004 to 1020 = 1,2,3,4,5

Assembly	Machine
addi $a0, $sp, 0	00100011101001000000000000000000
jal sum	00001100000000000000000000000100
addi $s0, $v0, 0	00100000010100000000000000000000
j exit	00001000000000000000000000001110
sum:	
addi $t1, $a0, 0	00100000100010010000000000000000
loop:	
slti $t2, $t0, 5	00101001000010100000000000000101
beq $t2, $0, exitLoop	00010001010000000000000000000101
lw $t3, 0($t1)	10001101001010110000000000000000
addi $t1, $t1, 4	00100001001010010000000000000100
addi $t0, $t0, 1	00100001000010000000000000000001
add $t4, $t4, $t3	00000001100010110110000000100000
j loop	00001000000000000000000000000101
exitLoop:	00100001100000100000000000000000
addi $v0, $t4, 0	00000011111000000000000000001000
jr $ra	00100011101001000000000000000000
exit:	



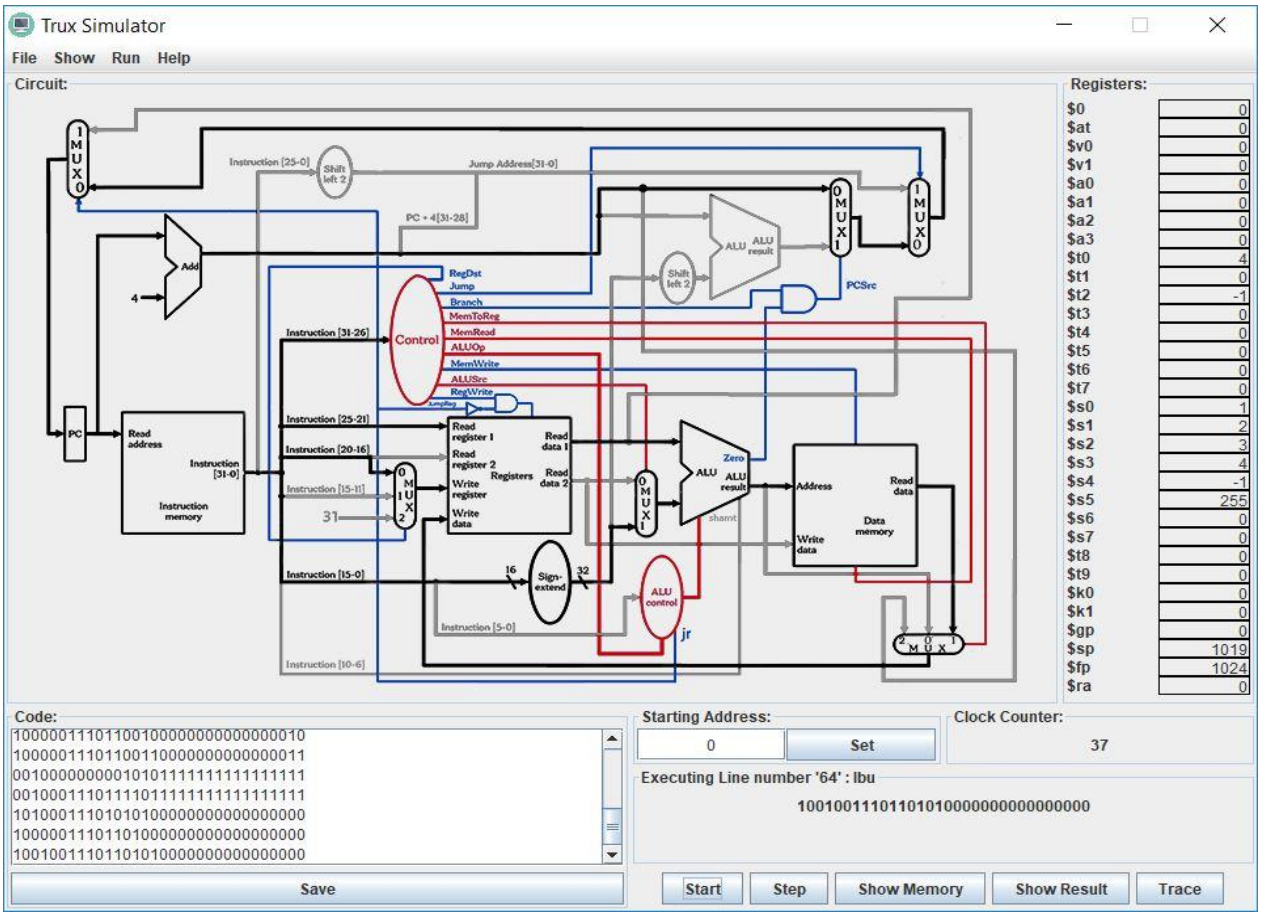

Memory
896 0 900 0 904 0 908 0
912 0 916 0 920 0 924 0
928 0 932 0 936 0 940 0
944 0 948 0 952 0 956 0
960 0 964 0 968 0 972 0
976 0 980 0 984 0 988 0
992 0 996 0 1000 0 1004 1
1008 2 1012 3 1016 4 1020 5
Save

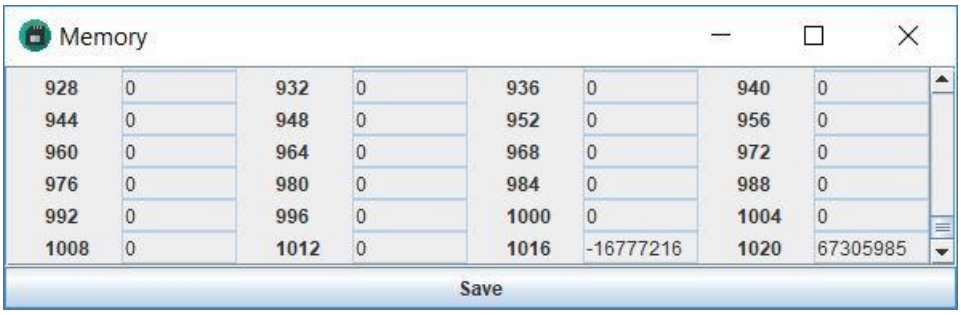

Trace
Wires
PC output: 12
PC+4 adder output: 16
Opcode field of instruction: 2
Rs field of instruction: 0
Rt field of instruction: 0
Rd field of instruction: 0
RegDestination Mux output: 0
Offset field of instruction: 14
Sign-extender Output: 14
Shift-2 output: 0
Target address adder: 1304
Function code of instruction: 14
Read data 1: 0
Read data 2: 0
ALU second input: 0
ALU output: 0
Zero flag: 1
Data memory output: 0
MemtoReg Mux output: 0
Control Signals
RegDest: 0
Branch: 0
MemRead: 0
MemtoReg: 0
ALUOp: 0
MemWrite: 0
ALUSrc: 1
RegWrite: 0
ALU control output: 2

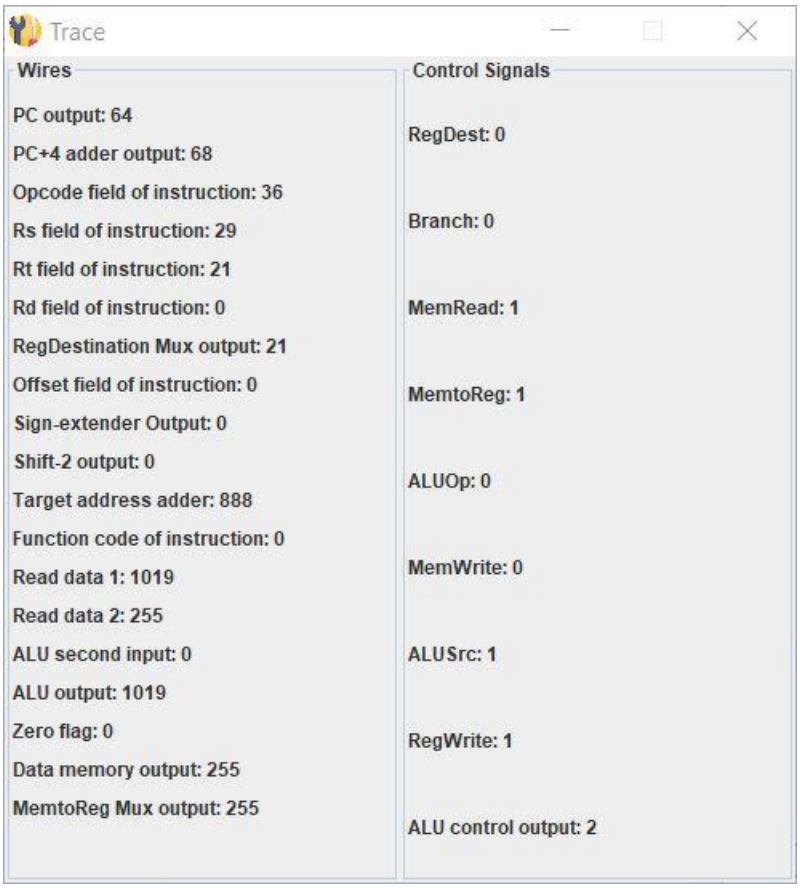

Result
Build Successfully!

6.3 Test Case 3

Assembly	Machine
addi $sp, $sp, -4	00100011101111011111111111111100
addi $t0, $0, 0	00100000000010000000000000000000
loop:	
slti $t1, $t0, 4	00101001000010010000000000000100
beq $t1, $0, exit	00010001001000000000000000000100
add $t1, $sp, $t0	00000011101010000100100000100000
addi $t0, $t0, 1	00100001000010000000000000000001
sb $t0, 0($t1)	10100001001010000000000000000000
j loop	00001000000000000000000000000010
exit:	
lb $s0, 0($sp)	10000011101100000000000000000000
lb $s1, 1($sp)	10000011101100010000000000000001
lb $s2, 2($sp)	10000011101100100000000000000010
lb $s3, 3($sp)	10000011101100110000000000000011
addi $t2, $0, -1	00100000000010101111111111111111
addi $sp, $sp, -1	00100011101111011111111111111111
sb $t2, 0($sp)	10100011101010100000000000000000
lb $s4, 0($sp)	10000011101101000000000000000000
lbu $s5, 0($sp)	10010011101101010000000000000000




Memory
928 0 932 0 936 0 940 0
944 0 948 0 952 0 956 0
960 0 964 0 968 0 972 0
976 0 980 0 984 0 988 0
992 0 996 0 1000 0 1004 0
1008 0 1012 0 1016 -16777216 1020 67305985
Save


Trace
Wires
PC output: 64
PC+4 adder output: 68
Opcode field of instruction: 36
Rs field of instruction: 29
Rt field of instruction: 21
Rd field of instruction: 0
RegDestination Mux output: 21
Offset field of instruction: 0
Sign-extender Output: 0
Shift-2 output: 0
Target address adder: 888
Function code of instruction: 0
Read data 1: 1019
Read data 2: 255
ALU second input: 0
ALU output: 1019
Zero flag: 0
Data memory output: 255
MemtoReg Mux output: 255
Control Signals
RegDest: 0
Branch: 0
MemRead: 1
MemtoReg: 1
ALUOp: 0
MemWrite: 0
ALUSrc: 1
RegWrite: 1
ALU control output: 2


Result
Build Successfully!

7.0 WHO DID WHAT EXACTLY

Instead of assigning individual roles to each member, we decided it would be more efficient and beneficial for us to work as a team on every aspect of the project.