Computer Architecture Project's Report

Ain Shams University

Computer Engineering and Software Systems

Prepared by:

Aly Mohammed Aly

Marina Gamal Ibrahim

Moussa Mohsen Moureed

Mohammed Said Mohamed Said

Submitted to:

Dr. Cherif Salama

A brief description of the implementation :

We wrote a class for each component in the circuit, ex: class for data memory, class for ALU.

A final class for the processor containing objects of each component, the processor behaves as a normal circuit where we use getters and setters as wires to connect components together.

Bonus features:

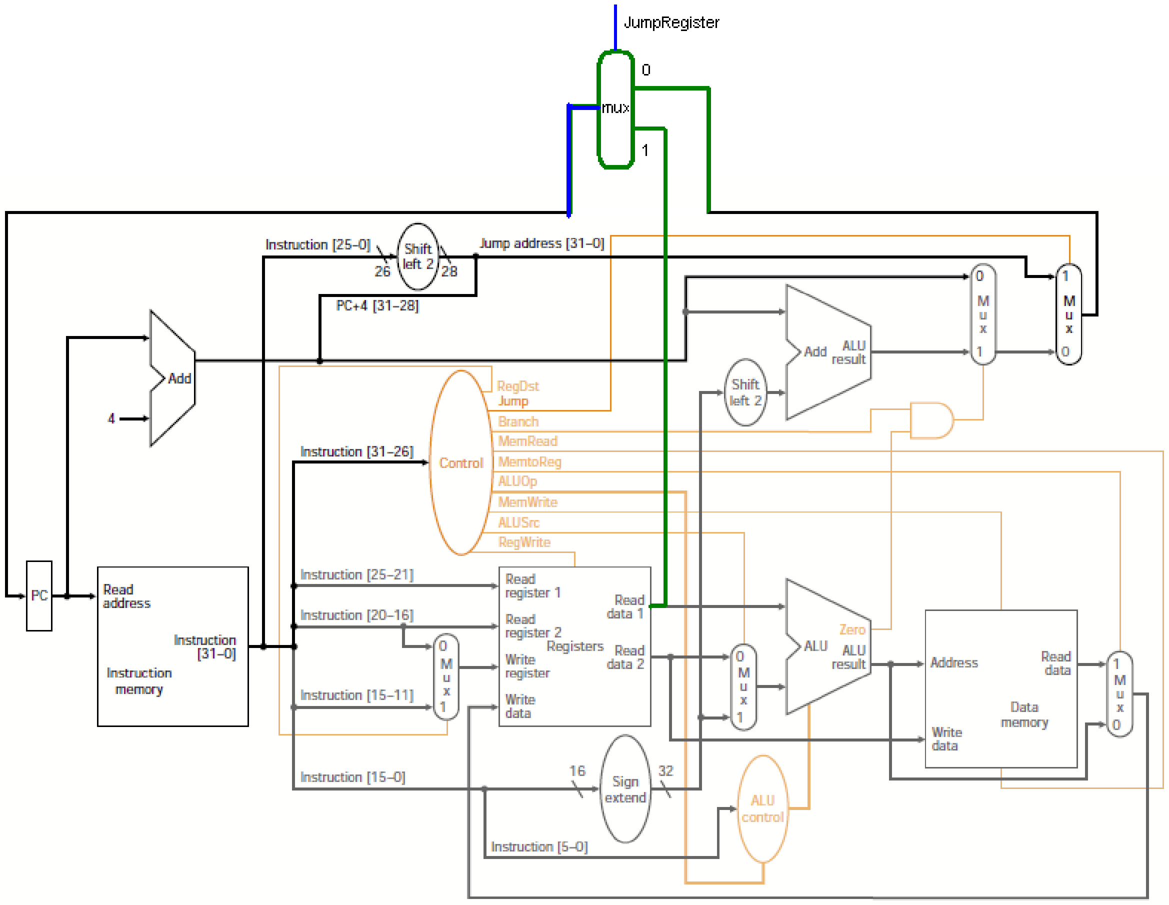
Included in the project's folder 6 programs (one of them is the sort program in lecture 5).

The simulator can take a written instruction however it is not yet completed as it only takes one instruction at a time.

The datapath we have used:

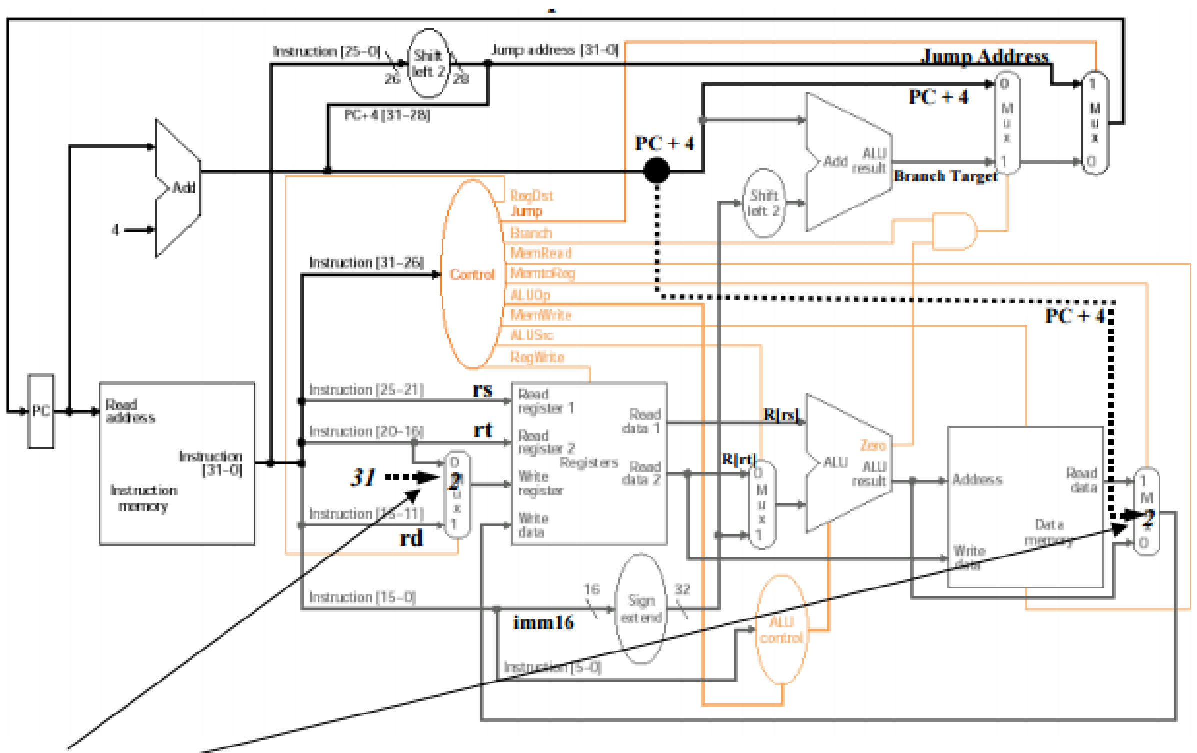
We initially used the datapath with jump supported as in lecture 7, however during implementation we faced a problem in implementing jr and jal, to solve this we extended the circuit to support jr by adding a detector circuit that takes the function code of the instruction and gives a flag of 1 if it is equal to the function code of jr, this flag becomes an input along with the op code in the control unit such that if the flag is 1 and the op code is zero the control unit sets the control signals to those of the jr, that's because although jr is an R-Format instruction it's control signals are completely different of those of the standard R-Format instructions.

Finally a multiplexer is added to select between read data 1 from source register, ex: $ra, and the output of the last multiplexer in the jump datapath.



As for the jal, we extended the register destination signal and the corresponding multiplexer into a 2 bit signal and a 4-1 multiplexer, were input 0 and 1 of the multiplexer remains unchanged while adding the $ra register as input 2 and a zero as input 3 since it is a don’t care.

The memory to register signal and the corresponding multiplexer were also extended such that inputs 0 and 1 remains unchanged and the pc+4 address is added to input 2 and a zero as input 3 since it is a don't care, therefore when a jal instruction is fetched, the control unit sets the correct signals including those for the 2 multiplexers mentioned above such that the pc+4 address in written in the $ra register.



1. Expand the multiplexor controlled by RegDst to include the value 31 as a new input 2.
2. Expand the multiplexor controlled by MemtoReg to have PC+4 as new input 2.

The truth table of the control unit and its logic diagram:

The Control Unit:

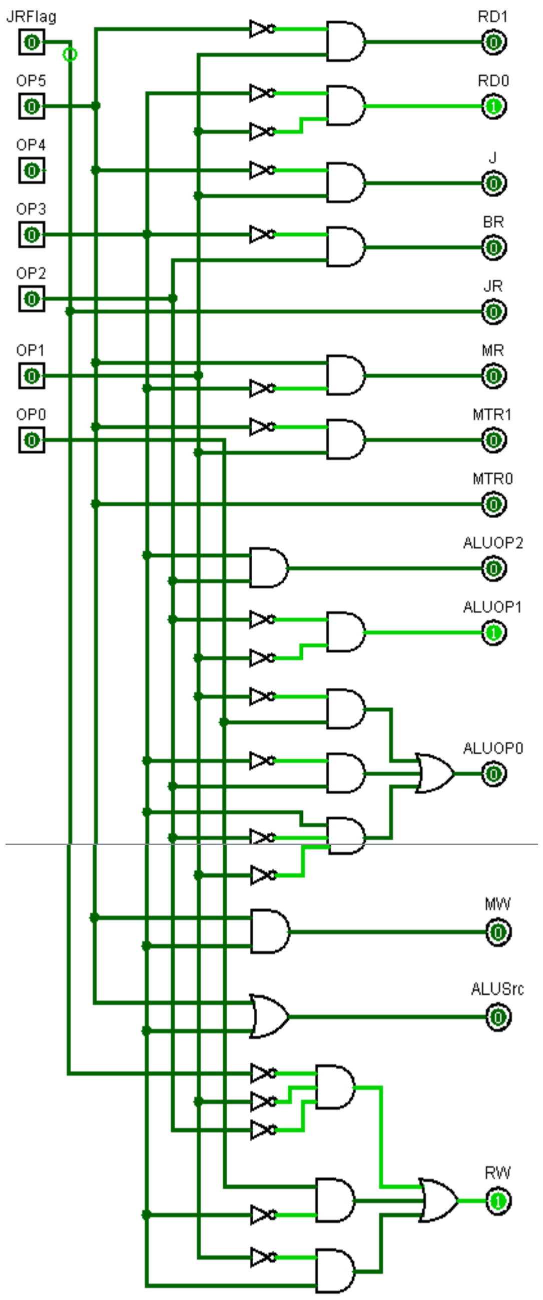
JR Flag	OP5	OP4	OP3	OP2	OP1	OP0	RD1	RD0	J	BR	JR	MR	MTR1	MTR0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	X	X	1	0	0	0	X	X
0	0	0	0	0	1	1	1	0	1	0	0	0	1	0
0	0	0	0	1	0	0	X	X	0	1	0	0	X	X
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	0	0	0	1	0	1
0	1	0	1	0	1	1	X	X	0	0	0	0	X	X
1	0	0	0	0	0	0	X	X	0	0	1	0	X	X

JR Flag	OP5	OP4	OP3	OP2	OP1	OP0	ALU2	ALU1	ALU0	MW	ALUSrc	RW
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	X	X	X	0	X	0
0	0	0	0	0	1	1	X	X	X	0	X	1
0	0	0	0	1	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	1	1	0	1	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	0	1	1	0	1	1	0	1	0	1	1
0	1	0	0	0	1	1	0	0	0	0	1	1
0	1	0	1	0	1	1	0	0	0	1	1	0
1	0	0	0	0	0	0	X	X	X	0	X	0

The ALU Control Unit:

ALU2	ALU1	ALU0	FN5	FN4	FN3	FN2	FN1	FN0	ALUC3	ALUC2	ALUC1	ALUC0
0	0	0	X	X	X	X	X	X	0	0	1	0
0	0	1	X	X	X	X	X	X	0	0	0	1
0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	0	0	1	0	0	0	0	1	1
0	1	0	1	0	0	1	0	1	0	1	0	0
0	1	0	1	0	0	1	1	1	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	1	0
0	1	1	X	X	X	X	X	X	0	0	1	0
1	0	0	X	X	X	X	X	X	0	0	1	1
1	0	1	X	X	X	X	X	X	0	1	0	0

The Logic Diagram:



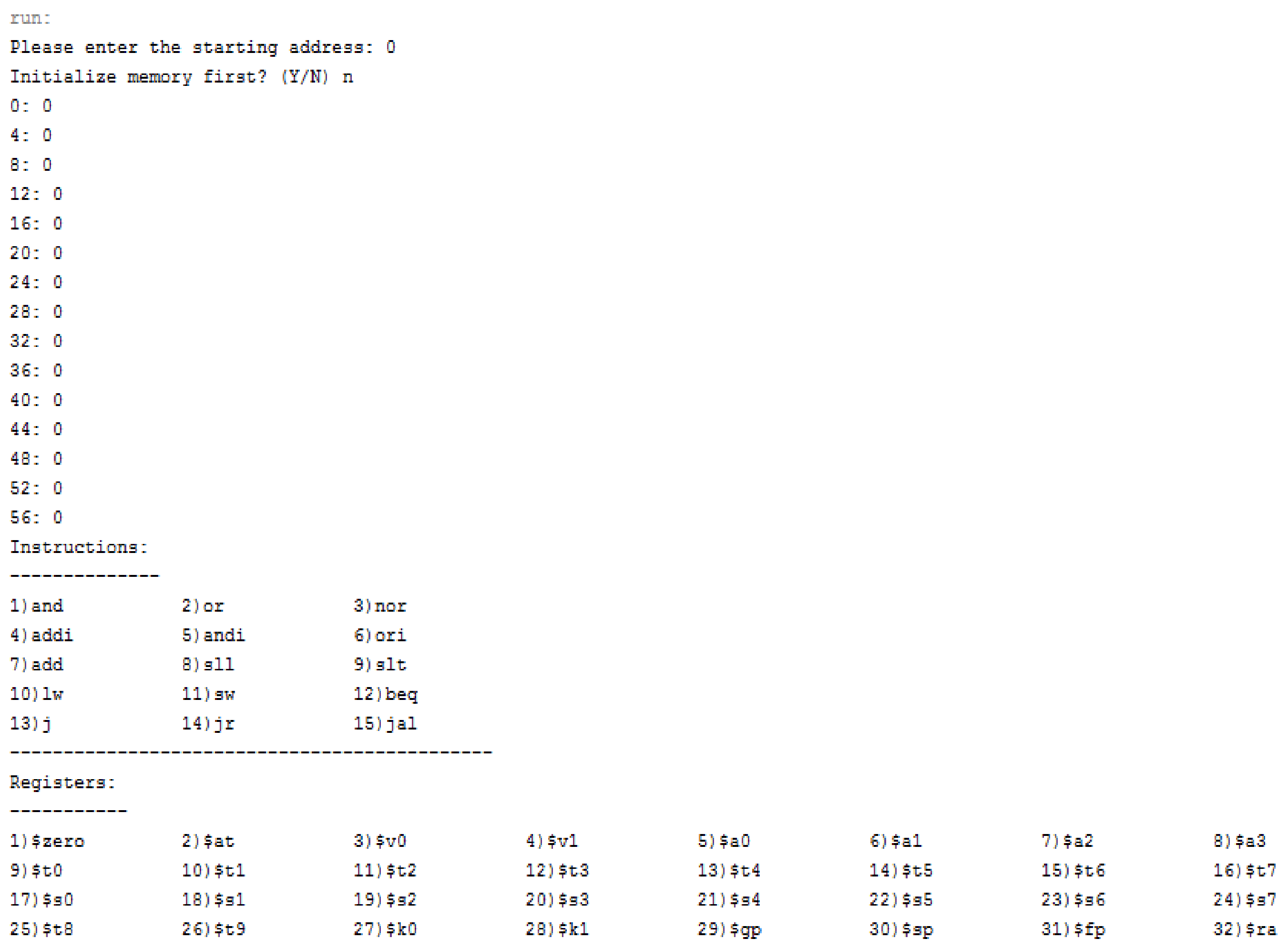
Assumptions:

- Two separate memories, instruction memory and data memory.
- A don’t care is treated as a -1 in the code.
- To simulate the stack pointer ($sp) in the data memory we assumed that the first 14 elements in the data memory are reserved for the data memory and initialized to zero, however they can be overwritten.
- The ALU op code signal was extended to a 3 bit signal to allow the ALU control unit to treat addi, andi, ori since they don’t have a function code and a 2 bit signal cannot cover these operations because each one requires a different ALU operation.

User guide:

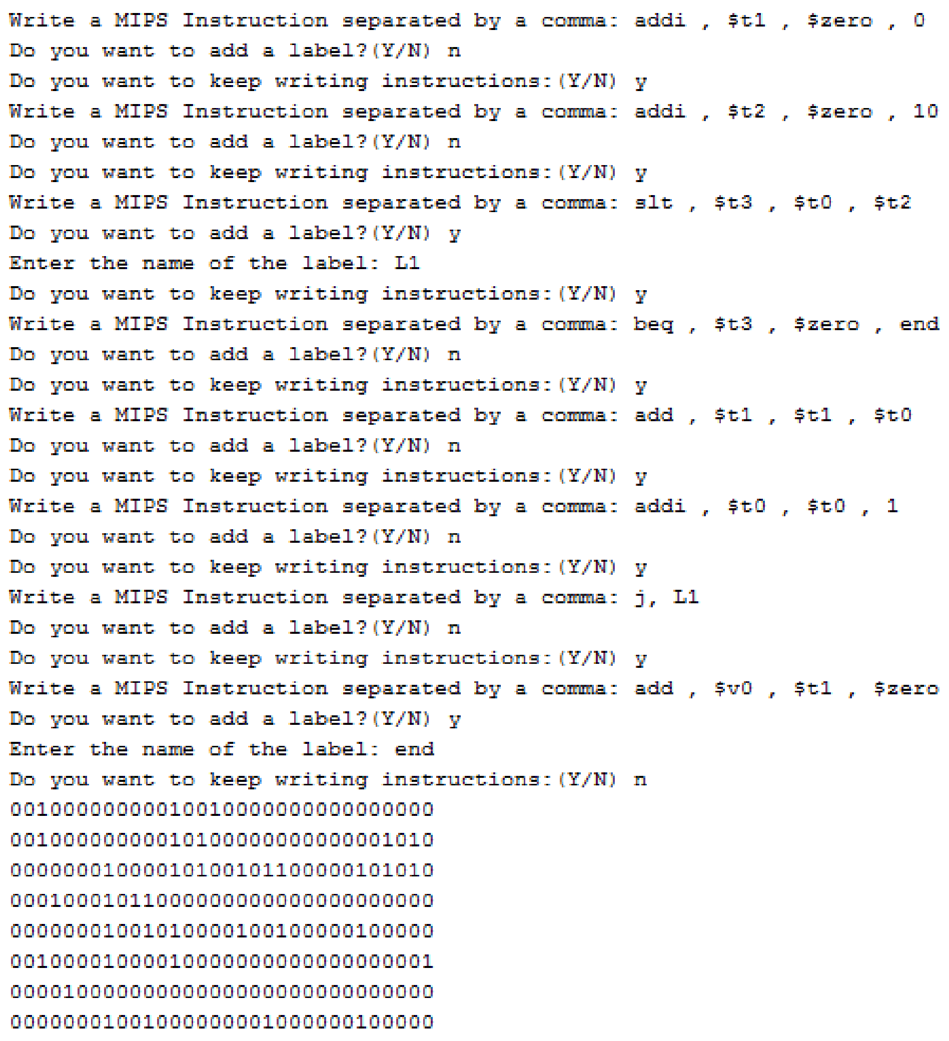
The user is allowed to choose a starting address and is given the option to initialize the data memory first, if the user inputs a starting address which is not a multiple of 4 the programs asks the user to re-enter a correct address, same goes for the memory initialization.

After finishing a list of supported instructions and registers are printed to guide the user while entering the program.



```
run:
Please enter the starting address: 0
Initialize memory first? (Y/N) n
0: 0
4: 0
8: 0
12: 0
16: 0
20: 0
24: 0
28: 0
32: 0
36: 0
40: 0
44: 0
48: 0
52: 0
56: 0
Instructions:
--------------
1)and          2)or           3)nor
4)addi         5)andi         6)ori
7)add          8)sll          9)slt
10)lw          11)sw          12)beq
13)j           14)jr          15)jal
---------------------------------------------
Registers:
-----------
1)$zero        2)$at          3)$v0          4)$v1          5)$a0          6)$a1          7)$a2          8)$a3
9)$t0          10)$t1         11)$t2         12)$t3         13)$t4         14)$t5         15)$t6         16)$t7
17)$s0         18)$s1         19)$s2         20)$s3         21)$s4         22)$s5         23)$s6         24)$s7
25)$t8         26)$t9         27)$k0         28)$k1         29)$gp         30)$sp         31)$fp         32)$ra
```

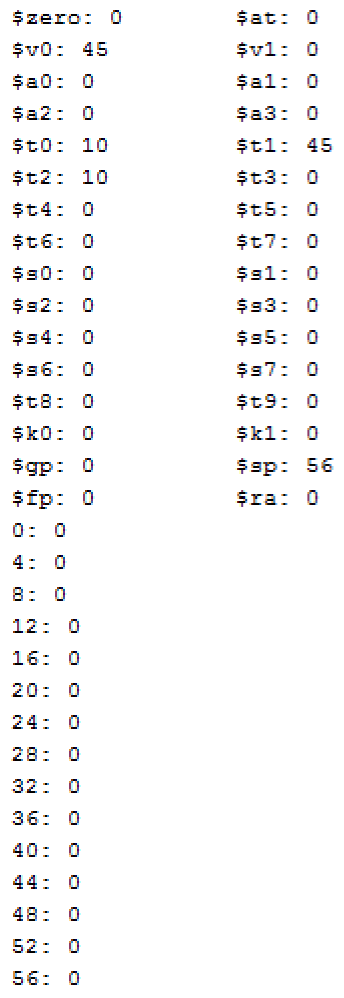
The user is asked to enter an instruction, however the instruction is slightly different from a regular MIPS instruction, a comma is added after the operation name to help us decode the string into separate parts. The decoding code covers all possible miss-inputs or errors in the instruction and inform the user of what the error is and allows him to re-enter. To support labels: the user is given the option to add a label name for each instruction. After finishing writing the program, the instructions in their binary form are printed.



```
Write a MIPS Instruction separated by a comma: addi , $t1 , $zero , 0
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: addi , $t2 , $zero , 10
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: slt , $t3 , $t0 , $t2
Do you want to add a label?(Y/N) y
Enter the name of the label: L1
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: beq , $t3 , $zero , end
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: add , $t1 , $t1 , $t0
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: addi , $t0 , $t0 , 1
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: j, L1
Do you want to add a label?(Y/N) n
Do you want to keep writing instructions:(Y/N) y
Write a MIPS Instruction separated by a comma: add , $v0 , $t1 , $zero
Do you want to add a label?(Y/N) y
Enter the name of the label: end
Do you want to keep writing instructions:(Y/N) n
00100000000010010000000000000000
00100000000010100000000000001010
00000001000010100101100000101010
00010001011000000000000000000000
00000001001010000100100000100000
00100001000010000000000000000001
00001000000000000000000000000000
00000001001000000001000000100000
```

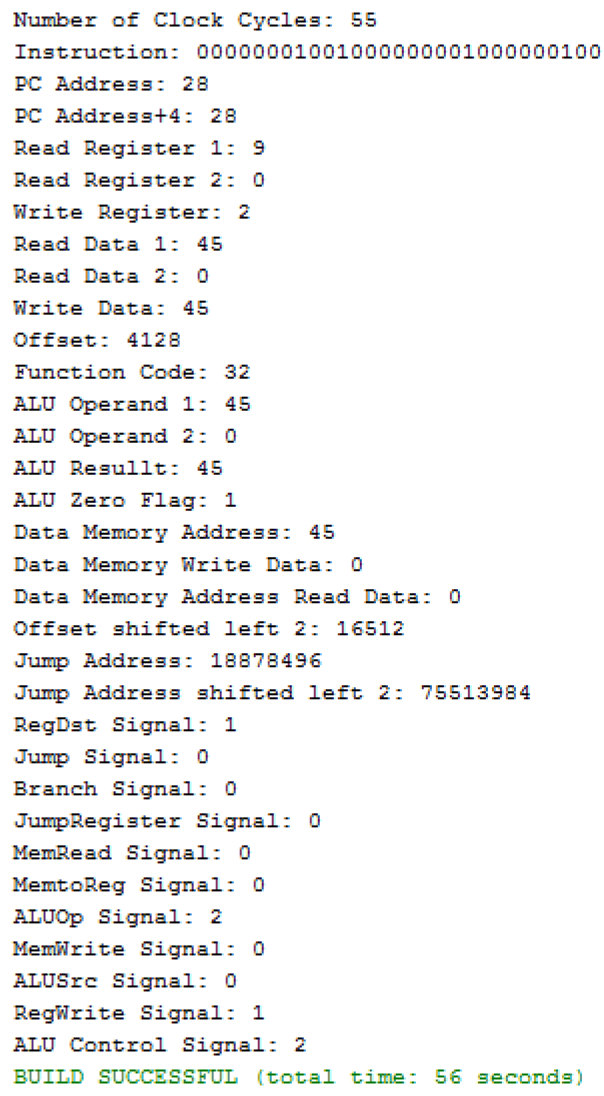
After executing the MIPS program final values of all registers and all memory used are printed.

In this example we did not use the memory, however the $sp reserved space, from address 0 to 56 are initialized to 0 as mentioned above.



```
$zero: 0        $at: 0
$v0: 45         $v1: 0
$a0: 0          $a1: 0
$a2: 0          $a3: 0
$t0: 10         $t1: 45
$t2: 10         $t3: 0
$t4: 0          $t5: 0
$t6: 0          $t7: 0
$s0: 0          $s1: 0
$s2: 0          $s3: 0
$s4: 0          $s5: 0
$s6: 0          $s7: 0
$t8: 0          $t9: 0
$k0: 0          $k1: 0
$gp: 0          $sp: 56
$fp: 0          $ra: 0
0: 0
4: 0
8: 0
12: 0
16: 0
20: 0
24: 0
28: 0
32: 0
36: 0
40: 0
44: 0
48: 0
52: 0
56: 0
```

Finally, values of all wires are printed.



```
Number of Clock Cycles: 55
Instruction: 00000001001000000001000000100000
PC Address: 28
PC Address+4: 28
Read Register 1: 9
Read Register 2: 0
Write Register: 2
Read Data 1: 45
Read Data 2: 0
Write Data: 45
Offset: 4128
Function Code: 32
ALU Operand 1: 45
ALU Operand 2: 0
ALU Resullt: 45
ALU Zero Flag: 1
Data Memory Address: 45
Data Memory Write Data: 0
Data Memory Address Read Data: 0
Offset shifted left 2: 16512
Jump Address: 18878496
Jump Address shifted left 2: 75513984
RegDst Signal: 1
Jump Signal: 0
Branch Signal: 0
JumpRegister Signal: 0
MemRead Signal: 0
MemtoReg Signal: 0
ALUOp Signal: 2
MemWrite Signal: 0
ALUSrc Signal: 0
RegWrite Signal: 1
ALU Control Signal: 2
BUILD SUCCESSFUL (total time: 56 seconds)
```

<u>Note:</u>

To make things easier, in the UserInterface class there are 2 methods InputProgram and InputProgram_FileReader included in the project's folder a ready made text file which can be directly read using the file reader method to avoid manually entering each instruction especially in large programs.

List of simulated programs:

1- \

```
int sum = 0;

for(int i=0 ; i < 10 ; i++)
    sum = sum+i;
```

```
addi , $t1 , $zero , 0
addi , $t2 , $zero , 10
L1: slt , $t3 , $t0 , $t2
beq , $t3 , $zero , end
add , $t1 , $t1 , $t0
addi , $t0 , $t0 , 1
j, L1
end: add , $v0 , $t1 , $zero
```

2-

```
void swap(int[] v, int k)
{
    int temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

```
void sort (int[] v , int n)
{
    int i, j;
    for (i=0 ; i<n ; i++)
    {
        for (j=i-1 ; j>=0 && v[j] > v[j+1] ; j--)
        {
            swap(v,j);
        }
    }
}
```

```
addi , $a0 , $zero , 100
addi , $a1 , $zero , 10
addi , $t9 , $zero , 1
jal , sort
j , exit
swap:
sll , $t1 , $a1 , 2
add , $t1 , $a0 , $t1
lw , $t0 , 0($t1)
lw , $t2 , 4($t1)
sw , $t2 , 0($t1)
sw , $t0 , 4($t1)
jr , $ra
sort:
addi , $sp ,$sp , -20
sw , $ra , 16($sp)
sw , $s3 , 12($sp)
sw , $s2 , 8($sp)
sw , $s1 , 4($sp)
sw , $s0 , 0($sp)
add ,$s2 , $zero ,$a0
add ,$s3 , $zero , $a1
add ,$s0 , $zero ,$zero
```

```
for1tst:
slt, $t0 , $s0 , $s3
beq , $t0 , $zero , exit1
addi , $s1 , $s0 , -1
for2tst:
slt , $t0 , $s1 , $zero
beq , $t0 , $t9 , exit2
sll , $t1 , $s1 , 2
add , $t2 , $s2 , $t1
lw , $t3 , 0($t2)
lw , $t4 , 4($t2)
slt ,$t0 , $t4 , $t3
beq ,$t0 , $zero , exit2
add ,$a0 , $zero , $s2
add ,$a1 , $zero , $s1
jal ,swap
addi ,$s1 , $s1 , -1
j ,for2tst
exit2:
addi , $s0 , $s0,1
j , for1tst
```

```
exit1:
lw , $s0 , 0($sp)
lw , $s1 , 4($sp)
lw , $s2 , 8($sp)
lw , $s3 , 12($sp)
lw , $ra , 16($sp)
addi ,$sp , $sp , 20
jr , $ra
exit: add , $zero , $zero , $zero
```

3-

```
int[] B = new int[10];
int[] A = new int[10];

A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;
A[5] = 6;
A[6] = 7;
A[7] = 8;
A[8] = 9;
A[9] = 10;

for(int i=0 ; i < 10 ; i++)
    B[i] = A[i];
```

```
addi,$s0,$zero,1000
addi,$s1,$zero,2000
add,$t0,$s0,$zero
addi,$t1,$zero,9
sll,$t1,$t1,2
add,$t1,$t1,$s1
L1:lw,$t2,0($t0)
sw,$t2,0($t1)
addi,$t0,$t0,4
addi,$t1,$t1,-4
slt,$t3,$t1,$s1
beq,$t3,$zero,L1
```

4-

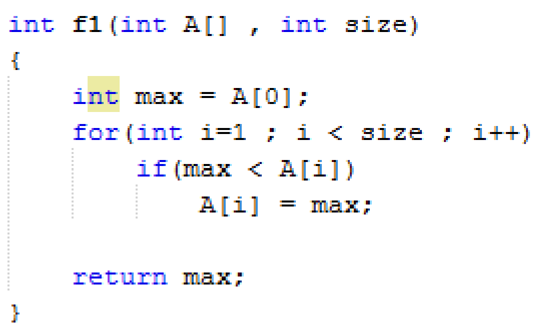
```
int f1(int A[] , int size)
{
    int max = A[0];
    for(int i=1 ; i < size ; i++)
        if(max < A[i])
            A[i] = max;

    return max;
}
```

```
addi ,$s0, $zero, 2000
addi ,$t0, $zero, 5
sw ,$t0, 0($s0)
addi ,$t0, $zero, 10
sw ,$t0, 4($s0)
addi ,$t0, $zero, 9
sw ,$t0, 8($s0)
addi ,$t0, $zero, 50
sw ,$t0, 12($s0)
addi ,$t0, $zero, 7
sw ,$t0, 16($s0)
addi ,$a0, $s0, 0
addi ,$a1, $zero, 5
jal ,f1
j , exit
f1: lw, $v0, 0($a0)
addi ,$t1, $zero, 1
L1: slt, $t2, $t1, $a1
beq ,$t2, $zero, L2
addi ,$a0, $a0, 4
lw ,$t3, 0($a0)
slt ,$t4, $v0, $t3
addi ,$t1, $t1, 1
```

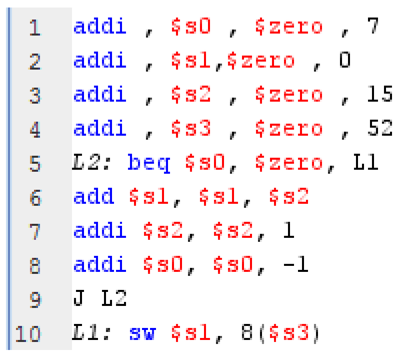
```
addi ,$v0, $t3, 0
j ,L1
L2: jr ,$ra
exit: add , $zero ,$zero,$zero
```

5-



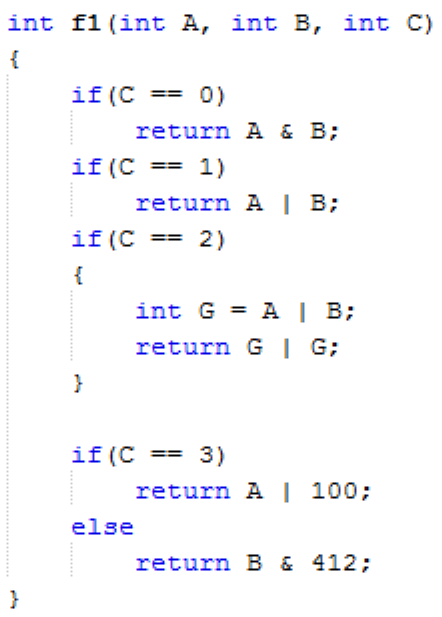
```c
int f1(int A[] , int size)
{
    int max = A[0];
    for(int i=1 ; i < size ; i++)
        if(max < A[i])
            A[i] = max;

    return max;
}
```



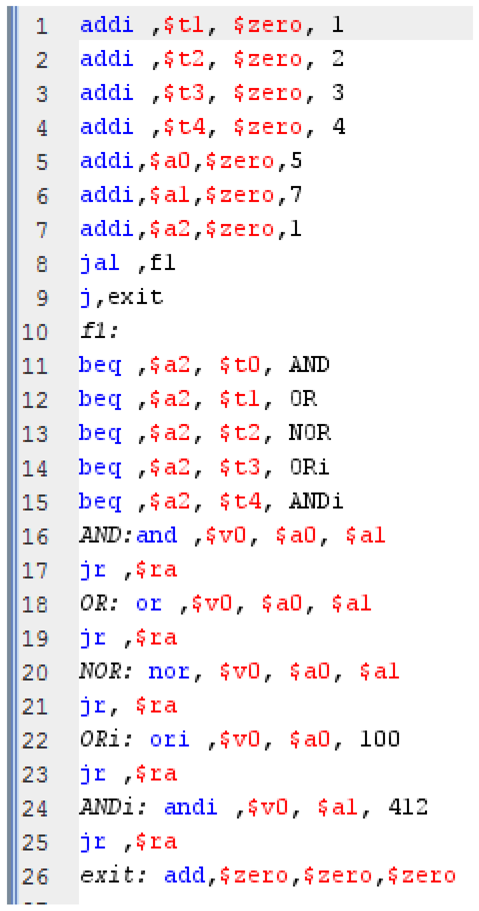
```
addi , $s0 , $zero , 7
addi , $s1,$zero , 0
addi , $s2 , $zero , 15
addi , $s3 , $zero , 52
L2: beq $s0, $zero, L1
add $s1, $s1, $s2
addi $s2, $s2, 1
addi $s0, $s0, -1
J L2
L1: sw $s1, 8($s3)
```

6-



```c
int f1(int A, int B, int C)
{
    if(C == 0)
        return A & B;
    if(C == 1)
        return A | B;
    if(C == 2)
    {
        int G = A | B;
        return G | G;
    }

    if(C == 3)
        return A | 100;
    else
        return B & 412;
}
```



```
addi ,$t1, $zero, 1
addi ,$t2, $zero, 2
addi ,$t3, $zero, 3
addi ,$t4, $zero, 4
addi,$a0,$zero,5
addi,$a1,$zero,7
addi,$a2,$zero,1
jal ,f1
j,exit
f1:
beq ,$a2, $t0, AND
beq ,$a2, $t1, OR
beq ,$a2, $t2, NOR
beq ,$a2, $t3, ORi
beq ,$a2, $t4, ANDi
AND:and ,$v0, $a0, $a1
jr ,$ra
OR: or ,$v0, $a0, $a1
jr ,$ra
NOR: nor, $v0, $a0, $a1
jr, $ra
ORi: ori ,$v0, $a0, 100
jr ,$ra
ANDi: andi ,$v0, $a1, 412
jr ,$ra
exit: add,$zero,$zero,$zero
```

Note: All the programs with their C code, MIPS code and screenshots are included in the project's folder.

How the work was split:

At the beginning, we divided the classes to each one of us.

Marina worked on the ALUControlUnit class and the InstructionMemory class and the MipsProcessor class.

Aly worked on the user interface and the Instruction class and the ControlUnit class.

Mohammed worked on the DataMemory class and the RegisterFile class and the Register class.

Moussa completed the user interface and the MipsProcessor class and worked on the ALU class and the JumpRegisterDetector class integrated the team’s work.