MIPS Simulator Report

CSE116: Computer Architecture

Made By: Medhat Ashraf Abdo Elhadad,11p1109

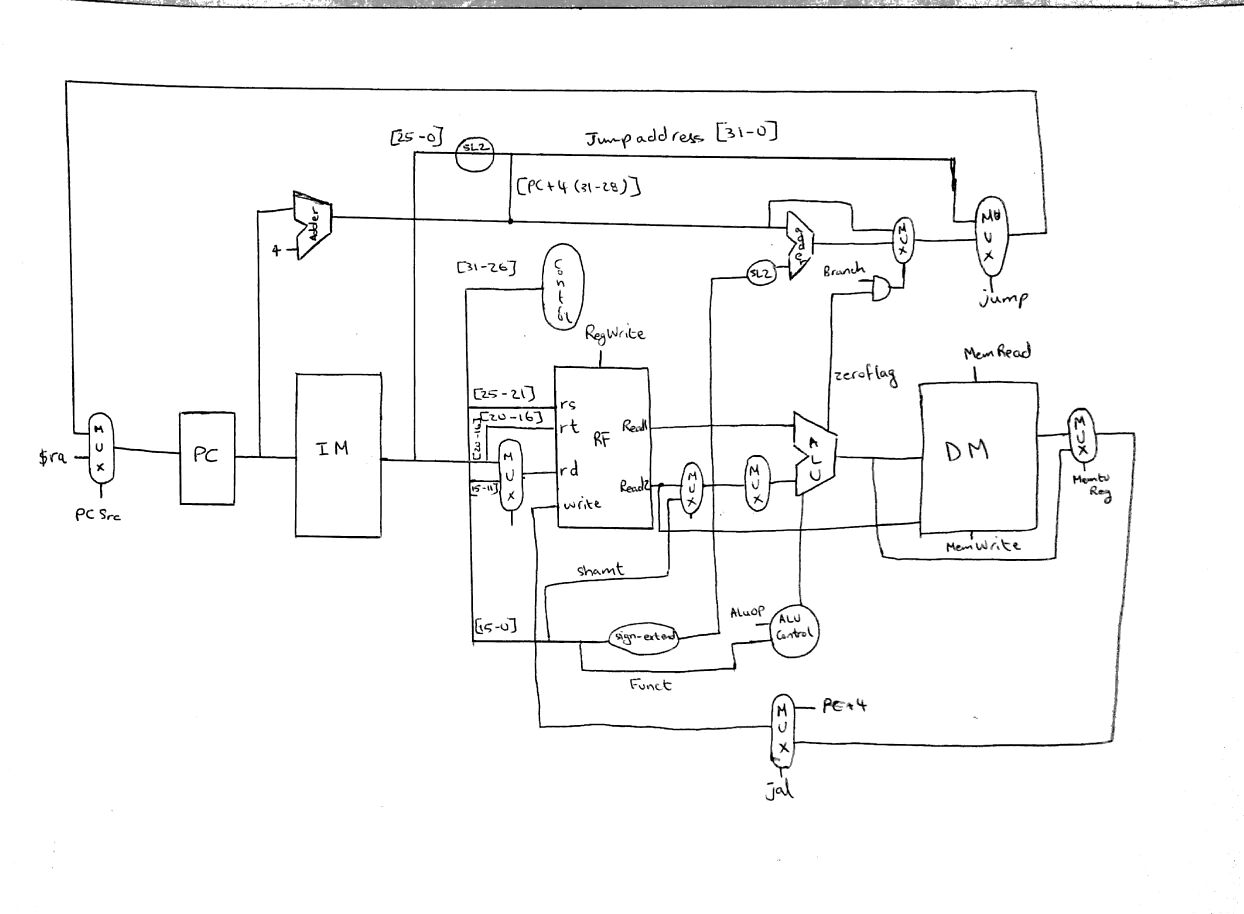
Ahmad Abdalla Medany,16p6072

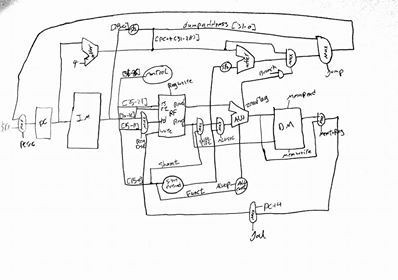
Hazem Hamada Abdellatif Mohamed,16p3100

# 1-Description:

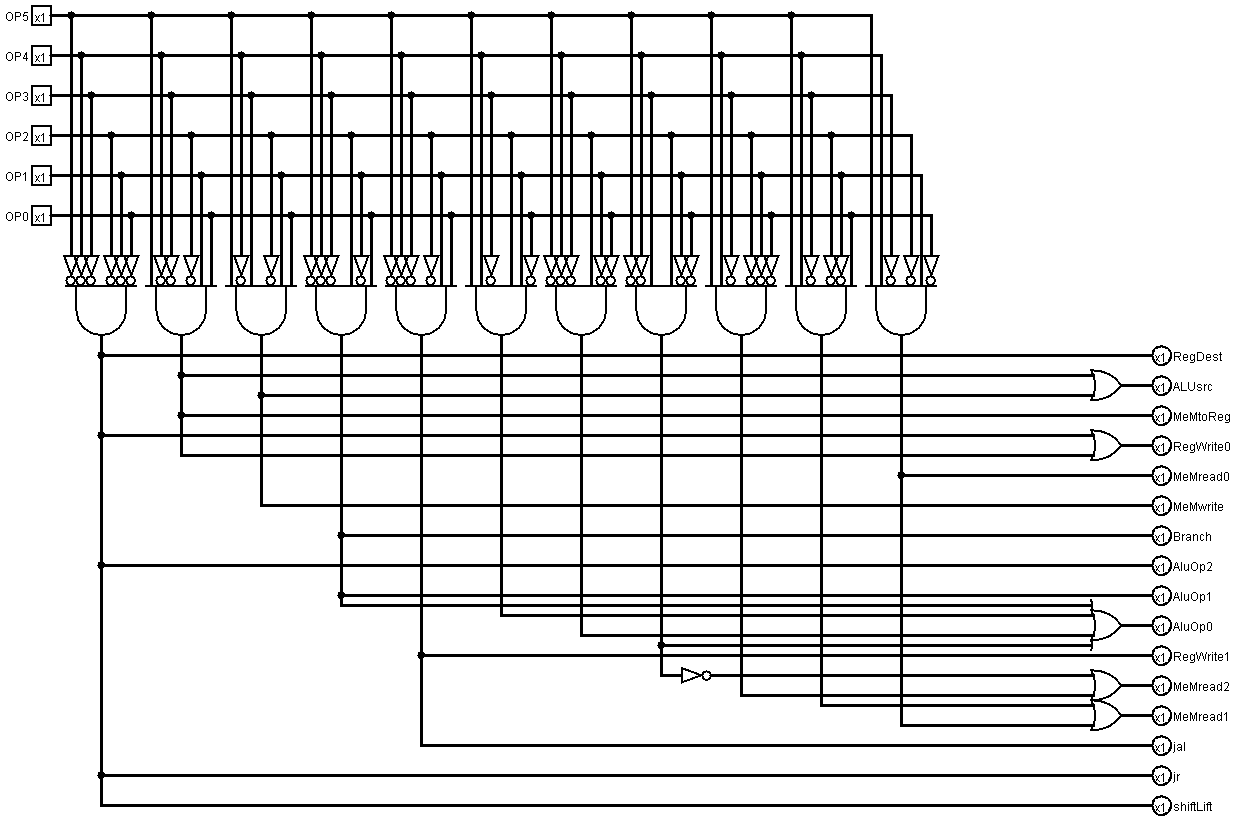
Mips Datapath and Control Unit Simulator is a software designed to simulate the flow of data inside a mips processor. This involves the decoding of assembly language programs into binary instructions, showing the exact path of these instructions through the different components that make up the processor, to get the desired program output. The software was written in Java. It supports all the primary instructions supported by mips, in addition to more instructions as a bonus feature. The software also has a GUI with an assembler. The instruction format is partially like the format originally supported by Mips , plus we added an GUI with an assembler plus we added 6 test programs with their c code .

## 2-Data path:

****



3-Control Unit Logic Diagram & Truth Table:

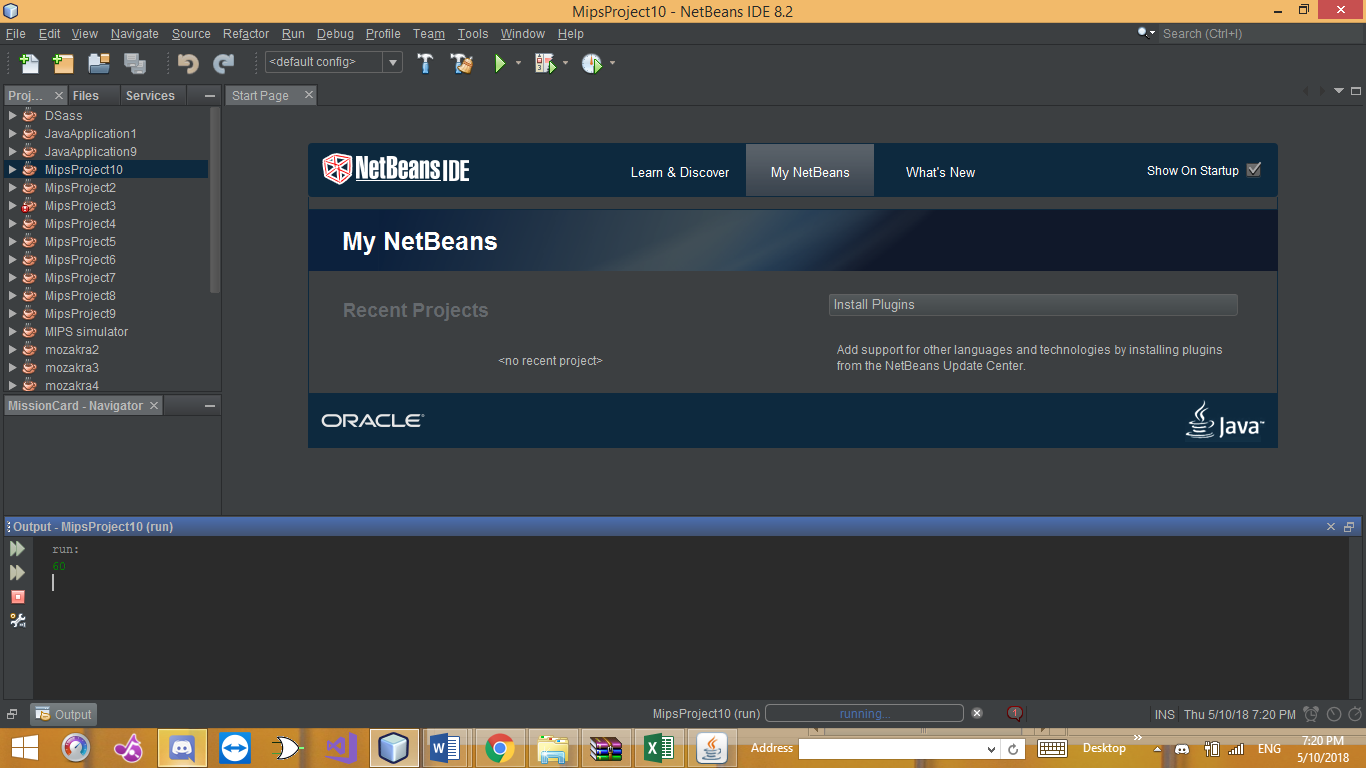


|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| opcode | 1-RegDst | 2-RegWrite | 3-AluSrc | 4-Aluop | 5-Branch | 6-MemRead | 7-MemWrite | 8-MemtoReg | 9-jump | 10- Jal | 11- jr | 12-Shiftlogic |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 001000 | 0 | 1 | 1 | 000 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 001100 | 0 | 1 | 1 | 011 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000100 | 0 | 0 | 0 | 001 | 1 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000101 | 0 | 0 | 0 | 001 | 1 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000010 | 0 | 0 | 0 | 000 | 0 | 000 | 00 | 0 | 1 | 0 | 0 | 0 |
| 000011 | 0 | 1 | 0 | 000 | 0 | 000 | 00 | 0 | 1 | 1 | 0 | 0 |
| 000000 | 0 | 1 | 0 | 000 | 0 | 000 | 00 | 0 | 0 | 0 | 1 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 100011 | 0 | 1 | 1 | 000 | 0 | 000 | 00 | 1 | 0 | 0 | 0 | 0 |
| 101011 | 0 | 0 | 1 | 000 | 0 | 000 | 10 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 1 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 1 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 110000 | 0 | 1 | 1 | 000 | 0 | 100 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110001 | 0 | 1 | 1 | 000 | 0 | 011 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110010 | 0 | 1 | 1 | 000 | 0 | 010 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110011 | 0 | 1 | 1 | 000 | 0 | 101 | 00 | 1 | 0 | 0 | 0 | 0 |
| 101100 | 0 | 0 | 1 | 000 | 0 | 000 | 01 | 0 | 0 | 0 | 0 | 0 |
| 101101 | 0 | 0 | 1 | 000 | 0 | 000 | 11 | 0 | 0 | 0 | 0 | 0 |
| 001101 | 0 | 1 | 1 | 010 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 110110 | 0 | 1 | 1 | 101 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |

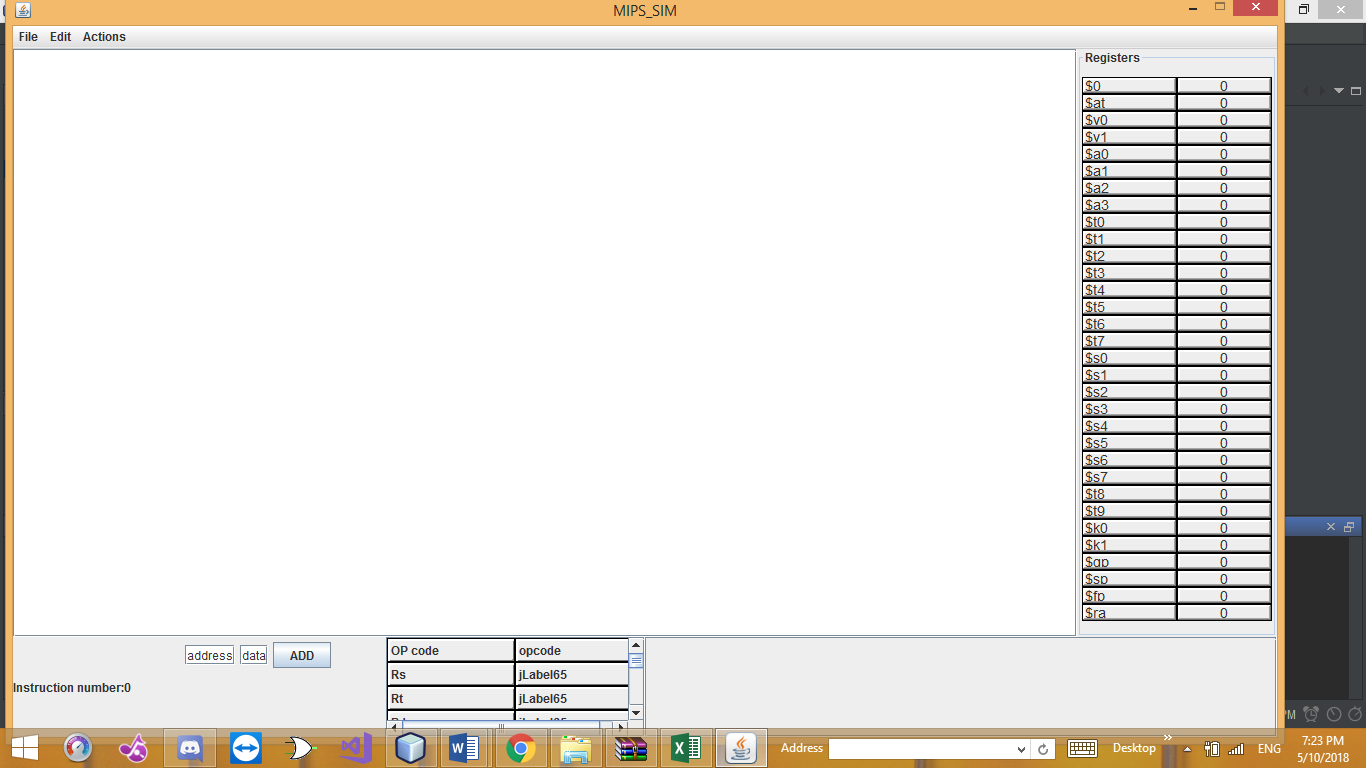
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| opcode | funct | alucont | type | no | inst | RegDst | RegWrite | AluSrc | Aluop | Branch | MemRead | MemWrite | MemtoReg | jump | jal | jr | Shiftlogic |
| 000000 | 100000 | 0010 | r | 0 | Add | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 001000 |  | 0010 | i | 1 | Addi | 0 | 1 | 1 | 000 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 100100 | 0000 | r | 2 | And | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 001100 |  | 0000 | i | 3 | Andi | 0 | 1 | 1 | 011 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000100 |  | 0110 | i | 4 | beq | 0 | 0 | 0 | 001 | 1 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000101 |  | 0101 | i | 5 | Bne | 0 | 0 | 0 | 001 | 1 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000010 |  | xxx | j | 6 | J | 0 | 0 | 0 | 000 | 0 | 000 | 00 | 0 | 1 | 0 | 0 | 0 |
| 000011 |  | xxx | j | 7 | Jal | 0 | 1 | 0 | 000 | 0 | 000 | 00 | 0 | 1 | 1 | 0 | 0 |
| 000000 | 001000 | xxx | r | 8 | Jr | 0 | 1 | 0 | 000 | 0 | 000 | 00 | 0 | 0 | 0 | 1 | 0 |
| 000000 | 100111 | 1100 | r | 9 | Nor | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 100101 | 0001 | r | 10 | or | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 100011 |  | 0010 | i | 11 | Lw | 0 | 1 | 1 | 000 | 0 | 000 | 00 | 1 | 0 | 0 | 0 | 0 |
| 101011 |  | 0010 | i | 12 | Sw | 0 | 0 | 1 | 000 | 0 | 000 | 10 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 111000 | 0011 | r | 13 | Sll | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 1 |
| 000000 | 100010 | 0110 | r | 14 | Sub | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 111001 | 1010 | r | 15 | Srl | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 1 |
| 000000 | 101010 | 0111 | r | 16 | Slt | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 101011 | 0111 | r | 17 | Sltu | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 110000 |  | 0010 | i | 18 | lb | 0 | 1 | 1 | 000 | 0 | 100 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110001 |  | 0010 | i | 19 | lbu | 0 | 1 | 1 | 000 | 0 | 011 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110010 |  | 0010 | i | 20 | lh | 0 | 1 | 1 | 000 | 0 | 010 | 00 | 1 | 0 | 0 | 0 | 0 |
| 110011 |  | 0010 | i | 21 | lhu | 0 | 1 | 1 | 000 | 0 | 101 | 00 | 1 | 0 | 0 | 0 | 0 |
| 101100 |  | 0010 | i | 22 | sb | 0 | 0 | 1 | 000 | 0 | 000 | 01 | 0 | 0 | 0 | 0 | 0 |
| 101101 |  | 0010 | i | 23 | sh | 0 | 0 | 1 | 000 | 0 | 000 | 11 | 0 | 0 | 0 | 0 | 0 |
| 001101 |  | 0001 | i | 24 | ori | 0 | 1 | 1 | 010 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 011000 | 1111 | r | 25 | mul | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 000000 | 011010 | 1110 | r | 26 | div | 1 | 1 | 0 | 100 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 110110 |  | 0111 | r | 27 | Slti | 0 | 1 | 1 | 101 | 0 | 000 | 00 | 0 | 0 | 0 | 0 | 0 |

4-User Guide:

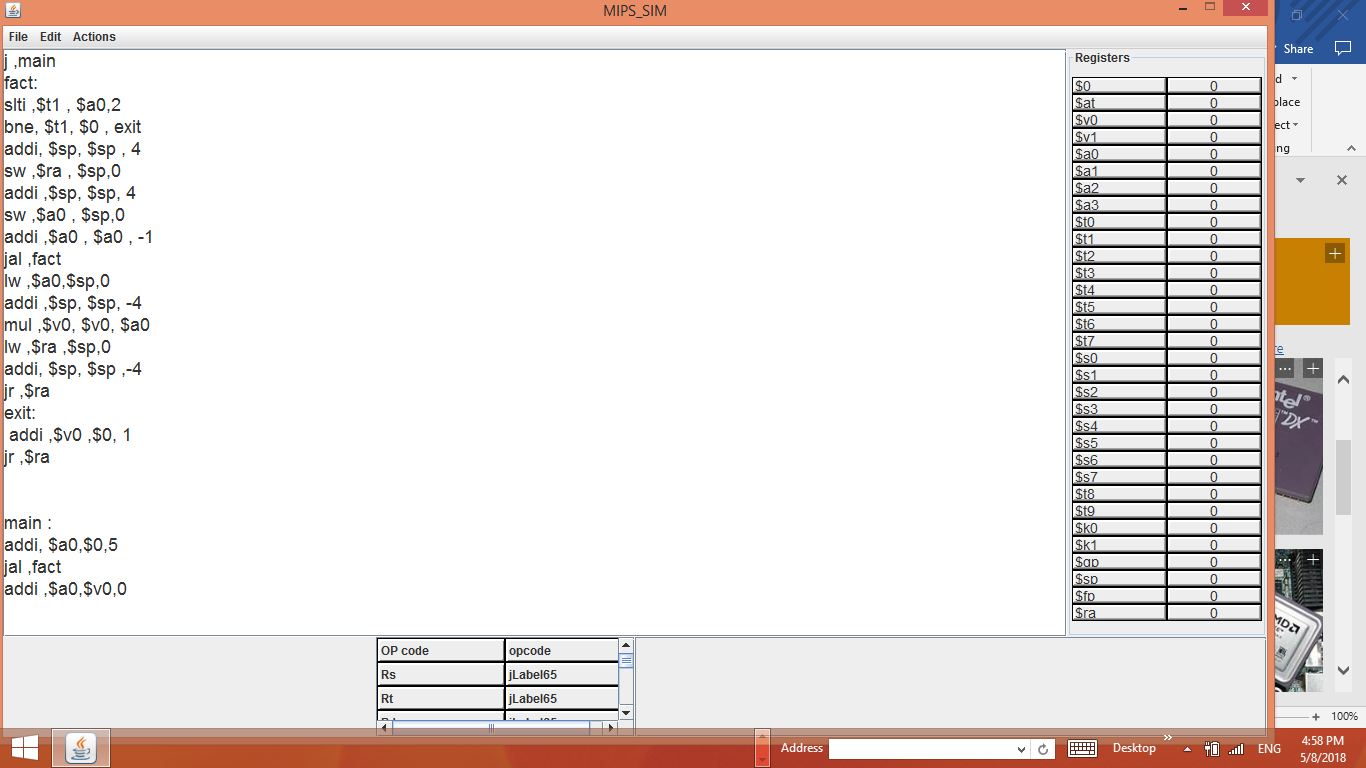
Step1: enter the instruction starting address from the console .



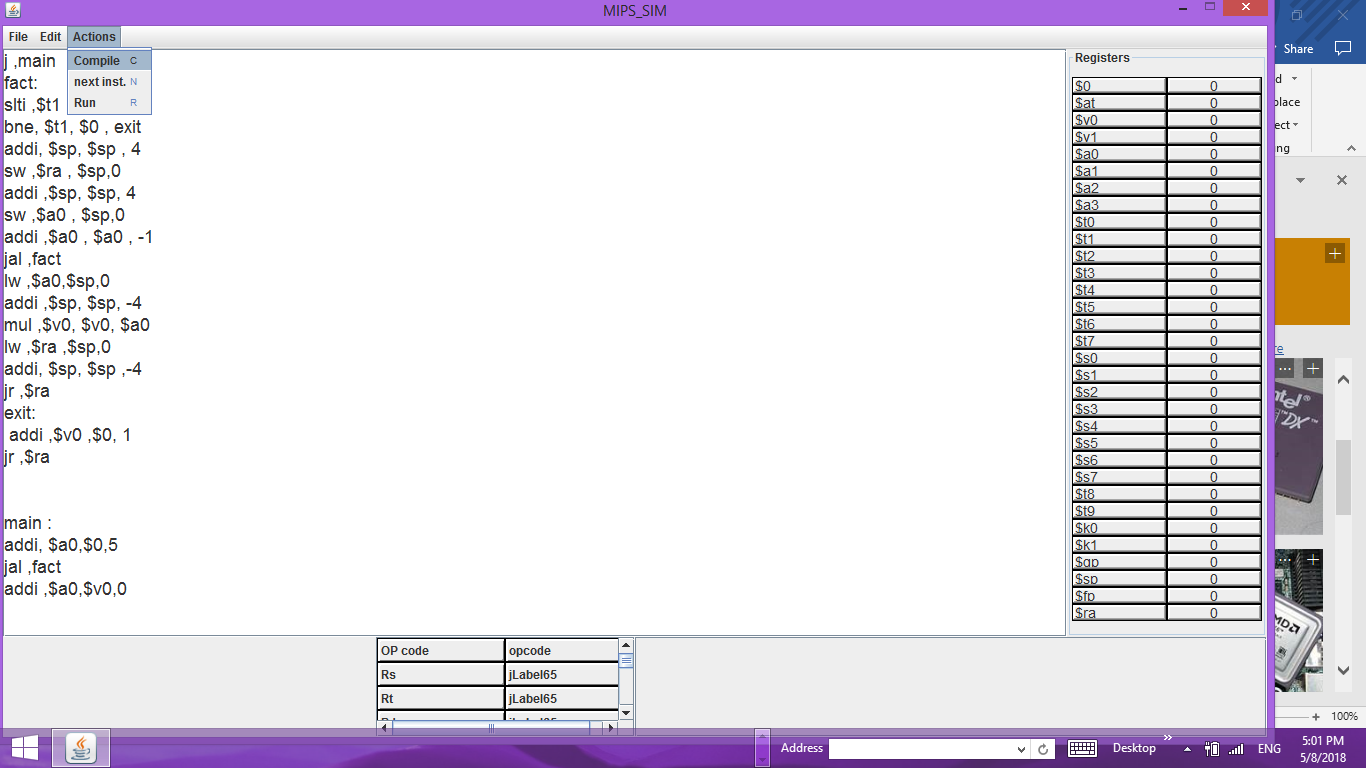
Step2:enter the data in the memory.



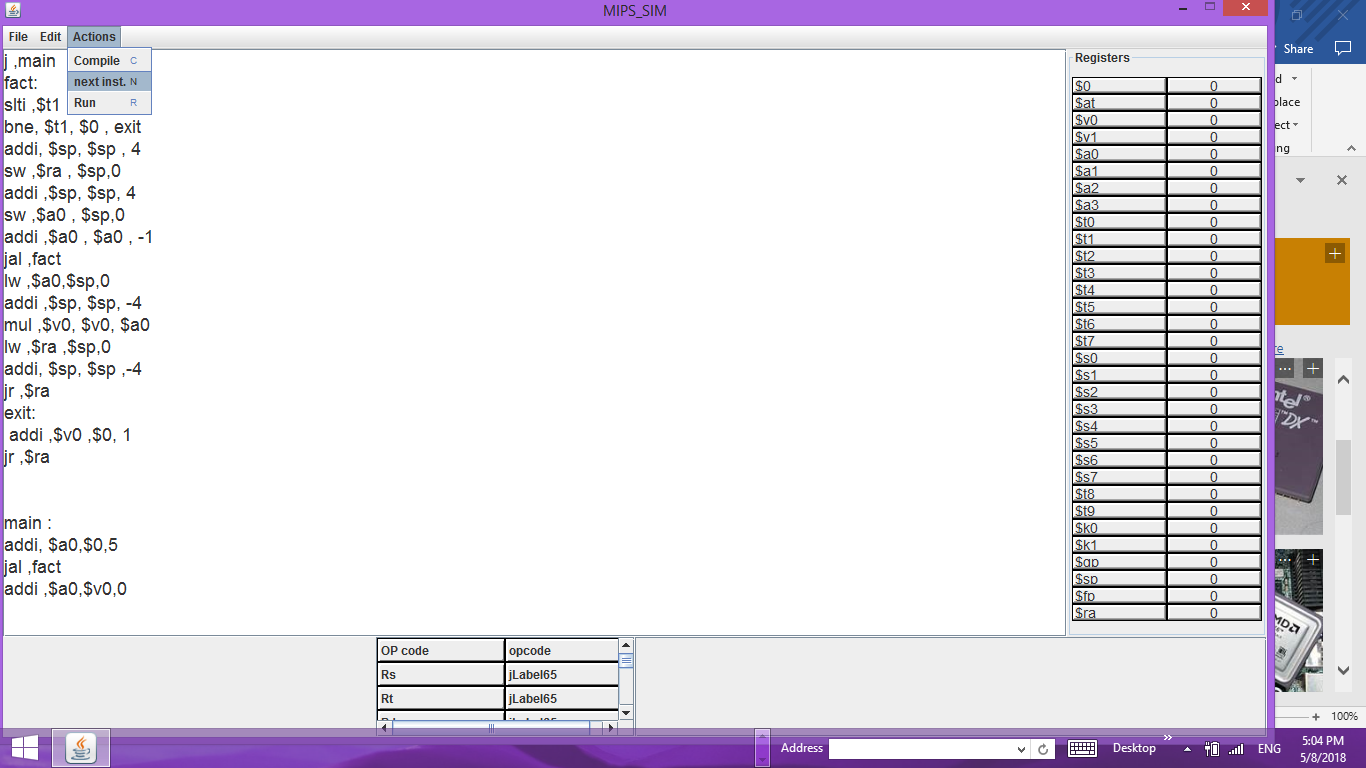
Step 3:enter the assembly code.



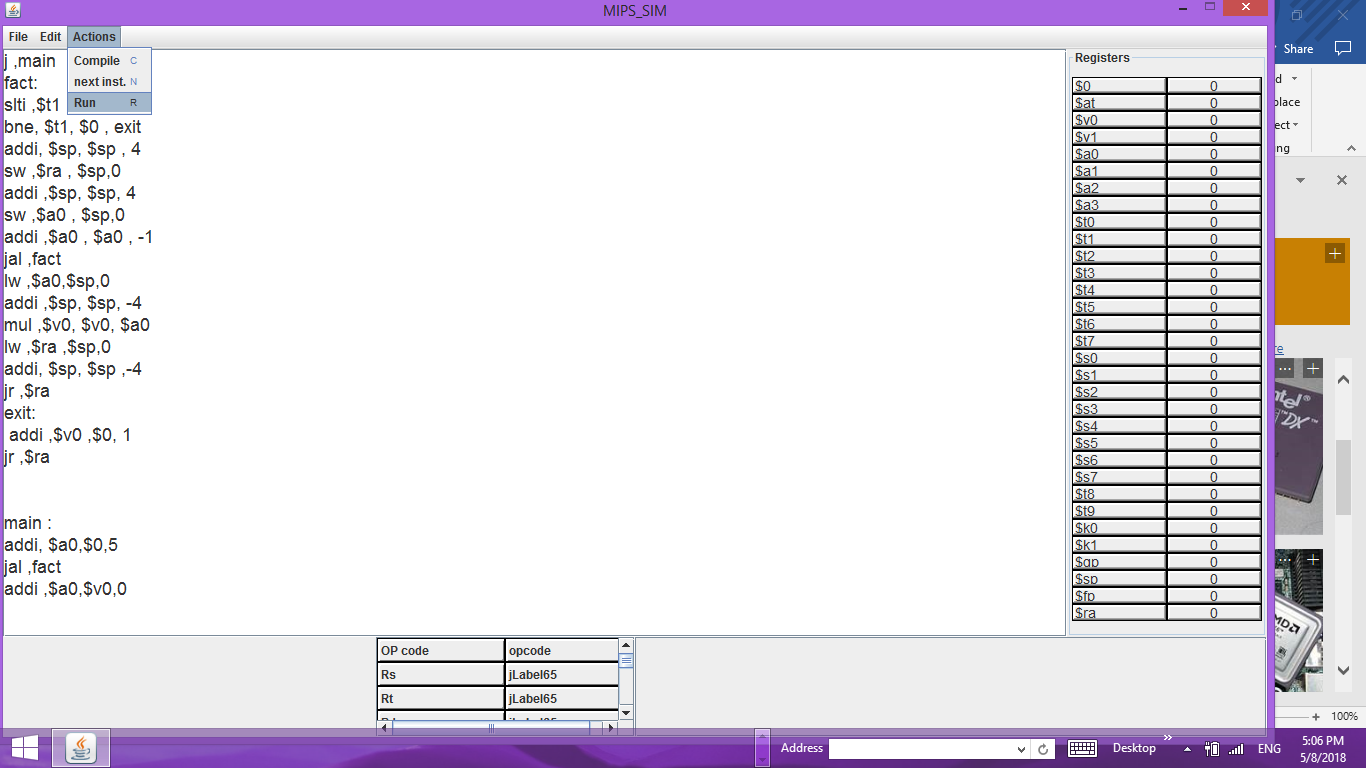
Step 4: Press the “compile” button from “Actions” from the menu bar.

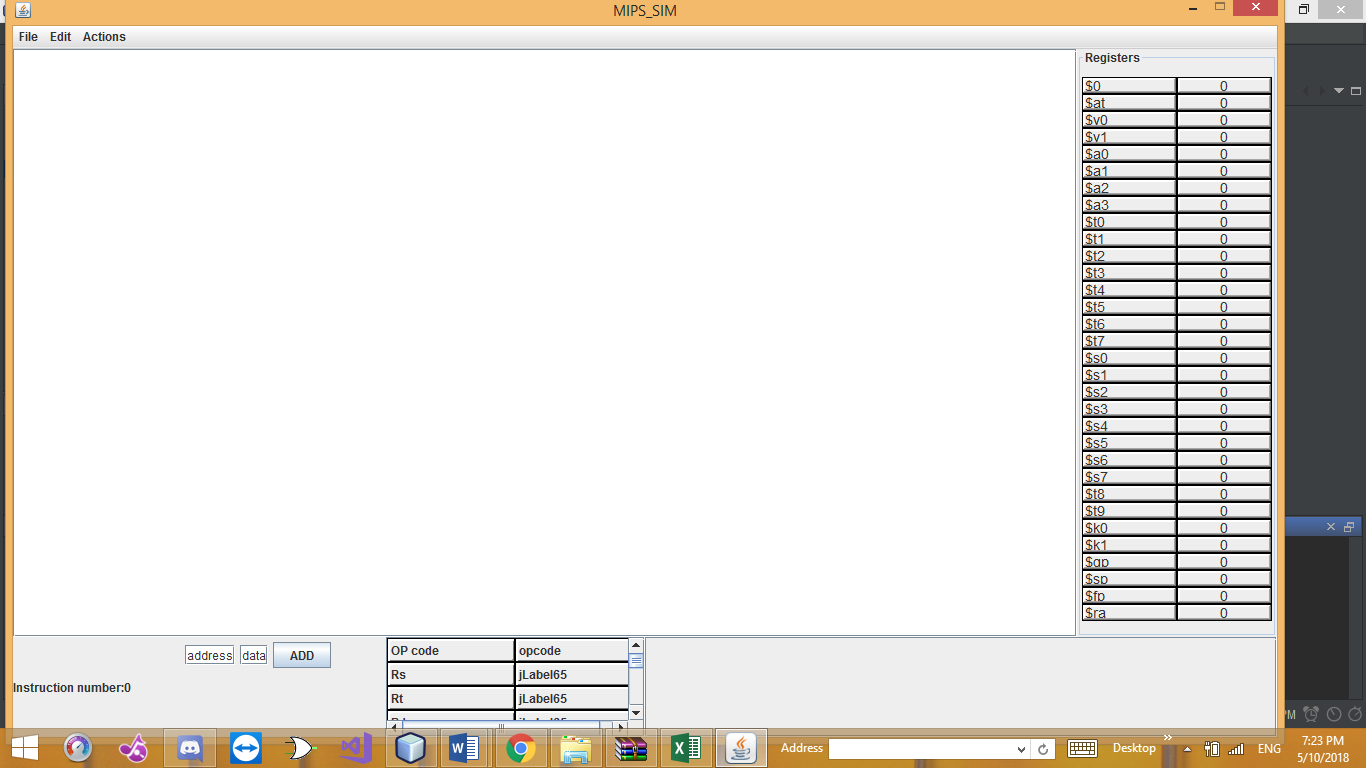


Step 5: Press the “Next Step” button from “Actions” from the menu bar to run the program step by step.



You can also run all the program at once by pressing the “Next Step” button from “Actions” from the menu bar.





Instructions done

Double tap

Insert data in memory

The value on each wire in the dataPath

The values inside the registers

The values of each used memory location

Area to write the assembly code

Menu bar for control and edit

5-Programs:

1-void main(){

Fact(5);}

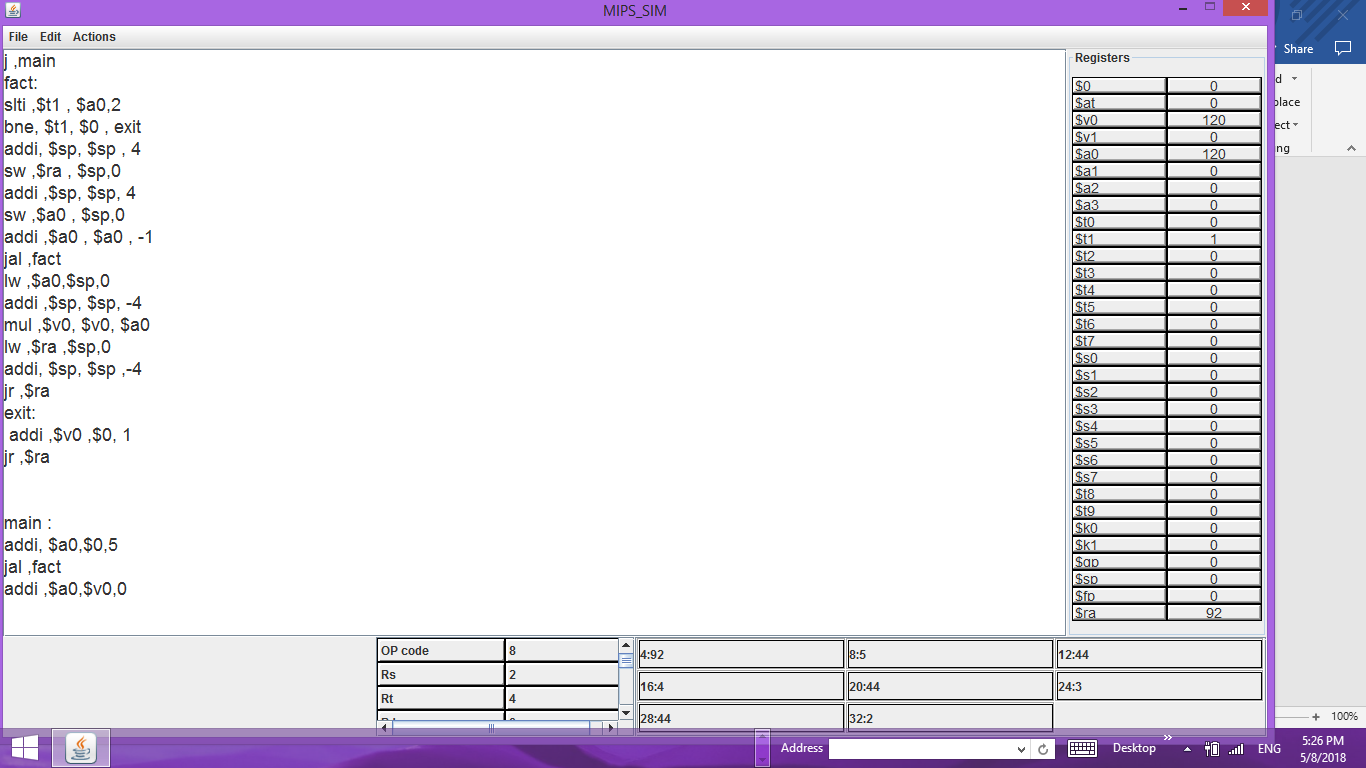
int fact(int n){

If(n==1)

Return 1;

Else

Return n\*fact(n-1);}



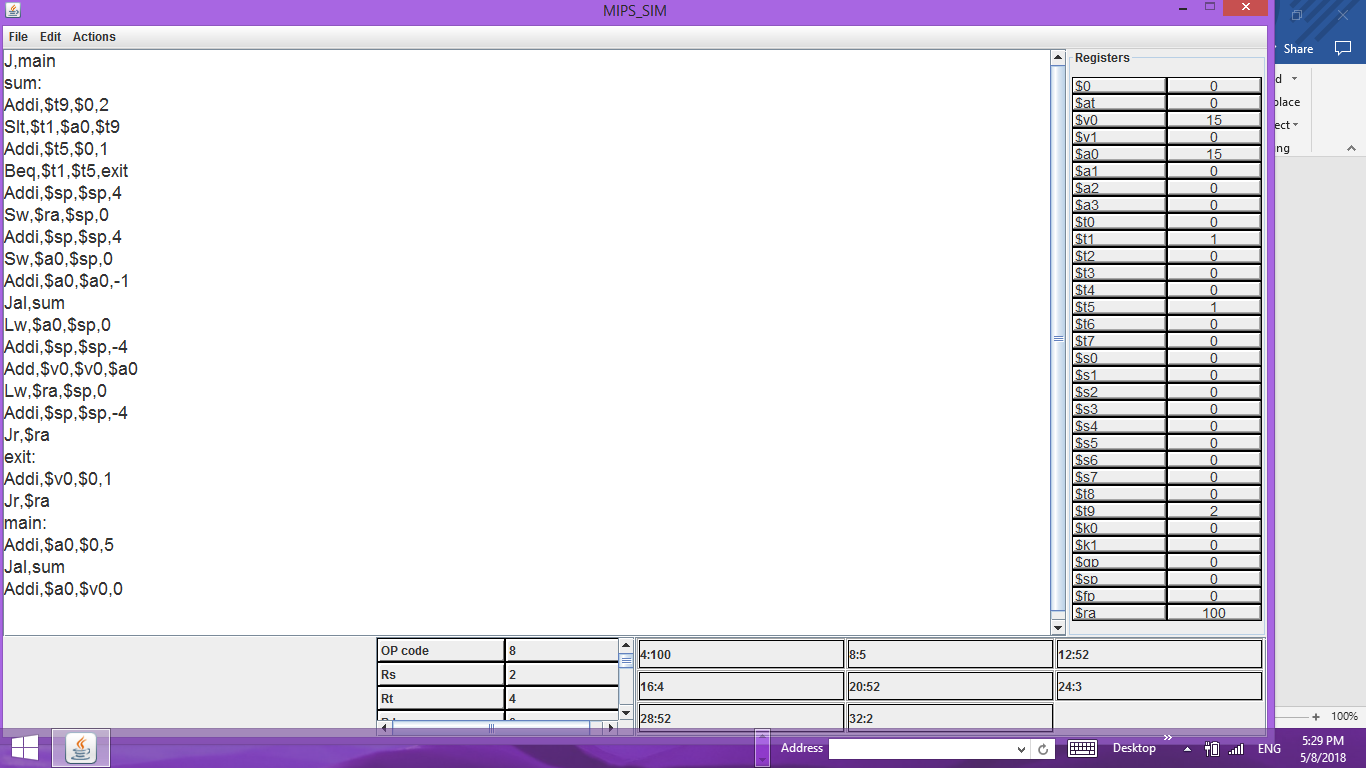
2-void main(){

Sum(5);}

Int sum(int n){

If(n==1)return 1;

Else return n+sum(n-1);}



3- void main(){

Fact(10);}

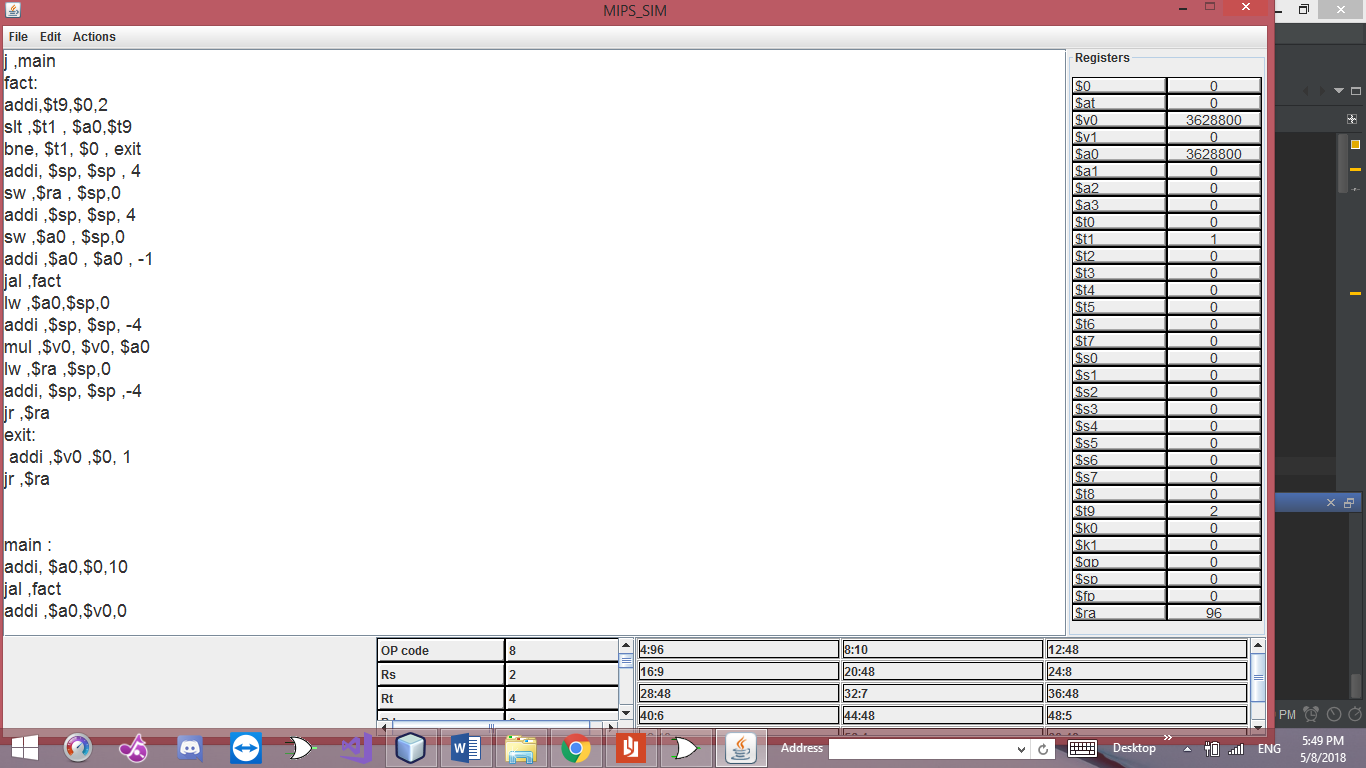
int fact(int n){

If(n==1)

Return 1;

Else

Return n\*fact(n-1);}



4-

void f(){

Int x=0,i=0;

Int y=12;

Int[3] n;

While(x<y){

N[i]=4\*x;

I++;

X+=4;}

Int m=0;

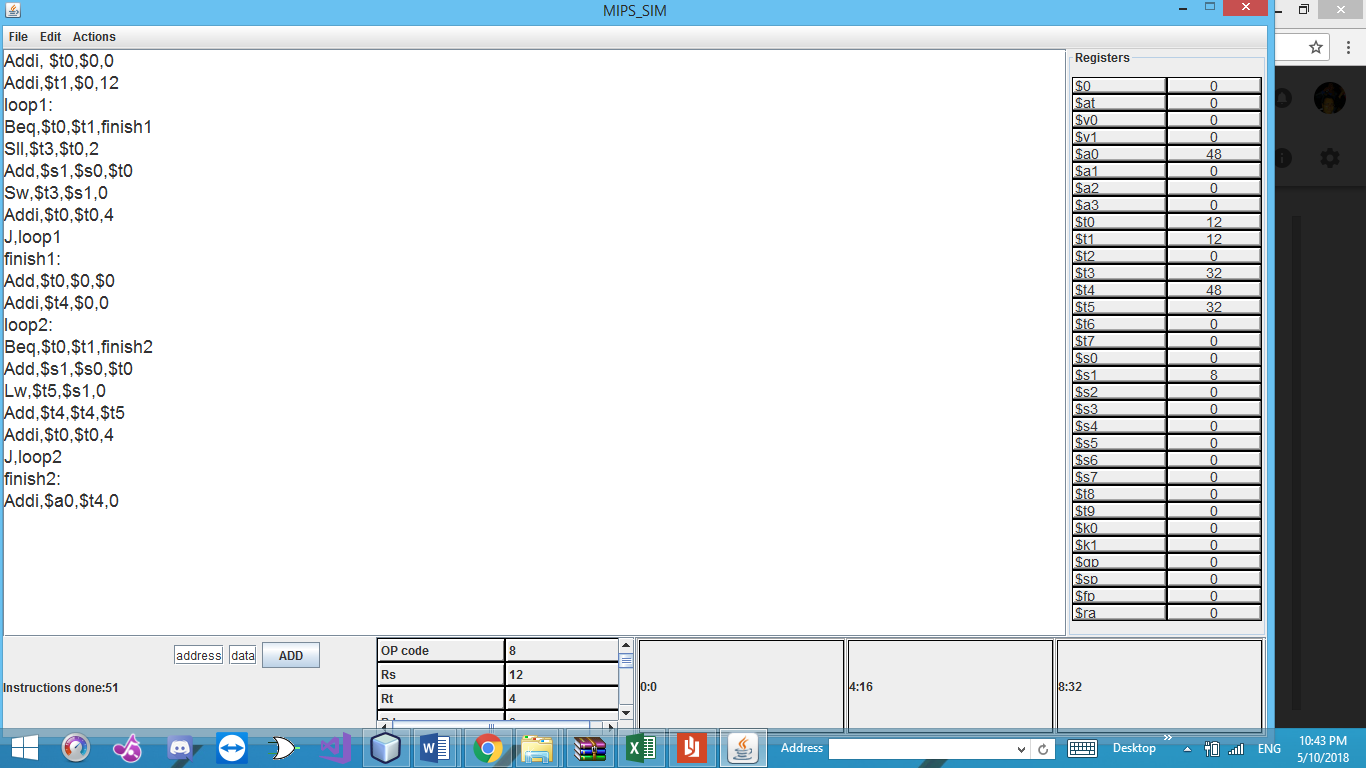
X=0;i=0;

While(x<y){

M+=n[i];

X+=4;

I++;}}



5- void f(){

Int x=0,i=0;

Int y=12;

Int[3] n;

While(x<y){

N[i]=4\*x;

I++;

X+=4;}

Int m=0;

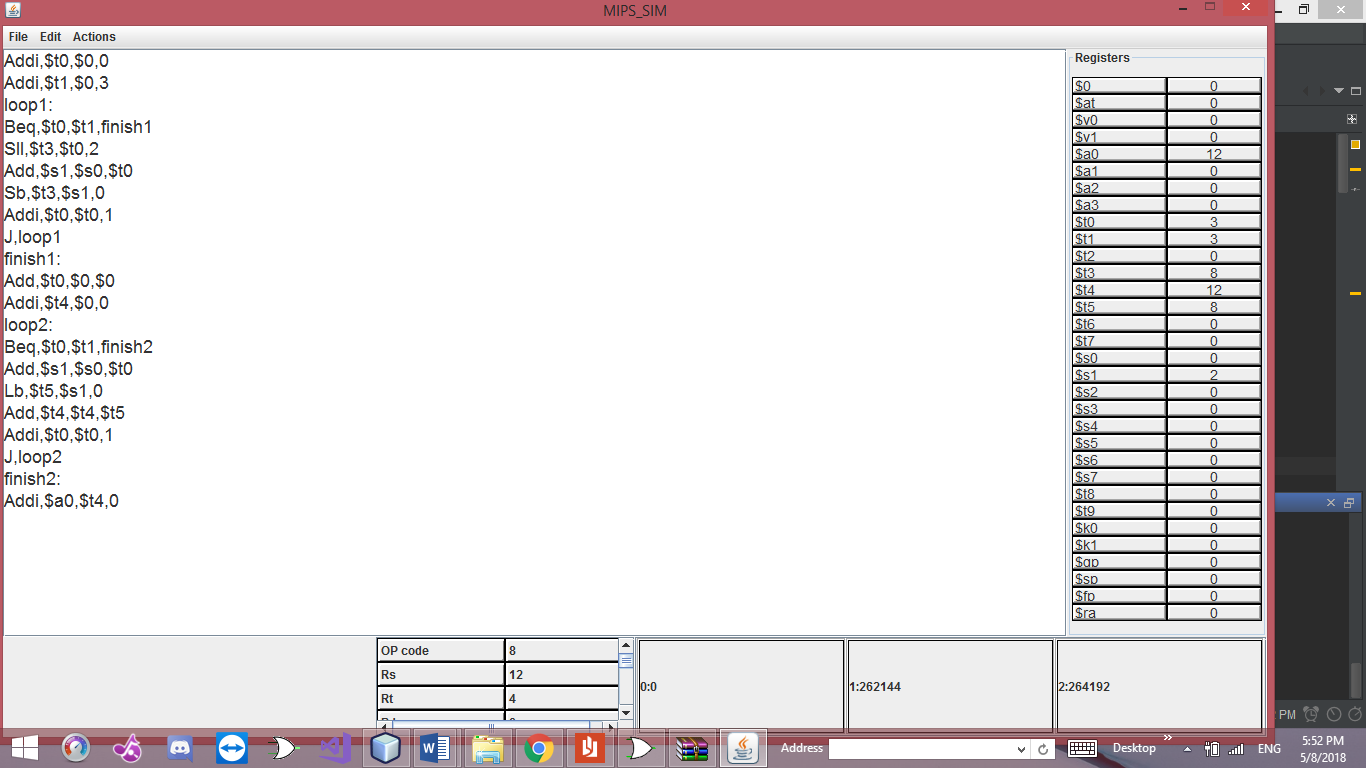
X=0;i=0;

While(x<y){

M+=n[i];

X+=4;

I++;}}



6-

Void f()

{

int x=15;

int y=5;

int n=x-y;

if(n!=10)

int m=28|80;

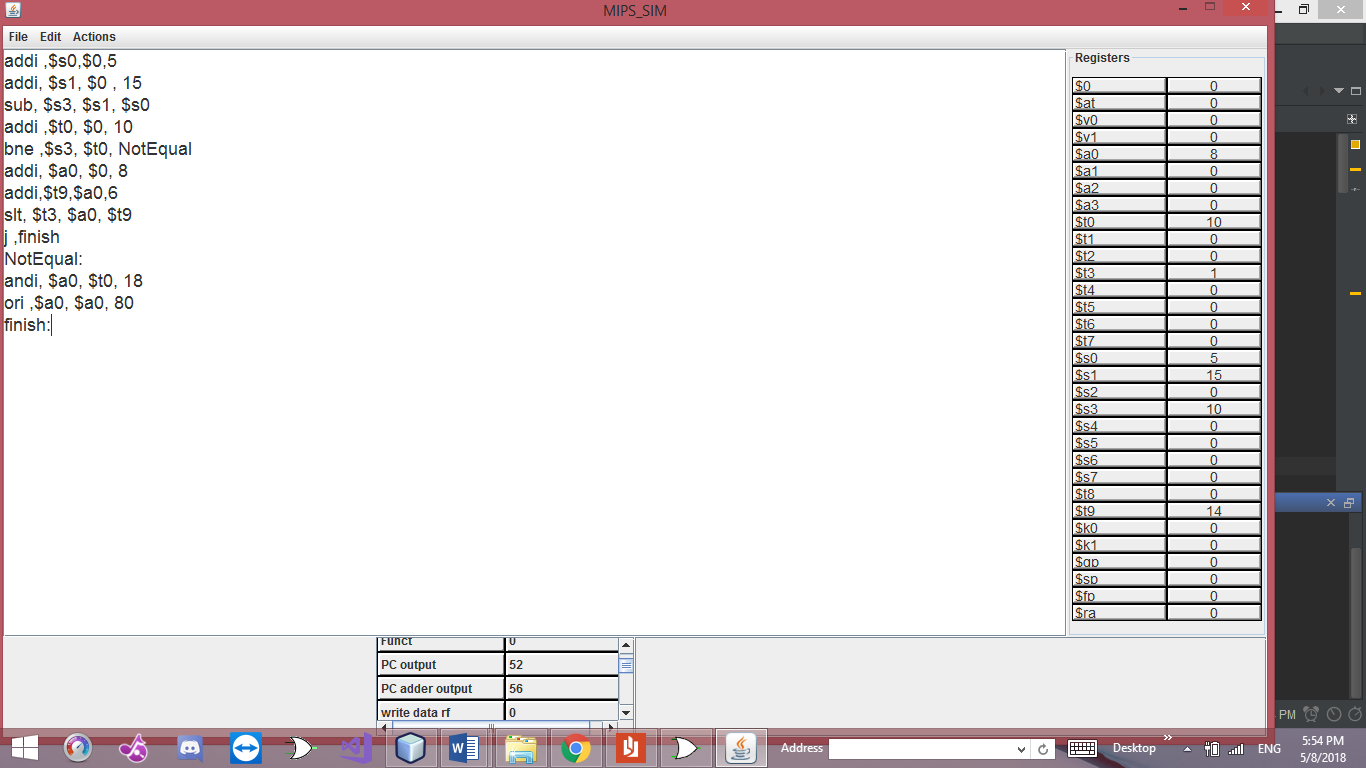
else { m=8;

if(m<14)

int p=1;

else int p=0;}

}



6-Assumptions

1. Operation name is separated from the rest of instructions by commas, not spaces.

**For example: add,rd,rs,rt**

**instead of: add rd, rs, rt**

2. The instructions format is somewhat different from the mars instructions format. The format for memory instructions is as follows:

**sw,rd,rt,offset**

3. The stack pointer’s increasing direction is reversed (For example, to add data to the memory, stack pointer should be incremented, not decremented).

4. Labels should not be followed by anything in the same line, instructions are written in a new line.

5. you cant run two programs after each other you must close the assembler first and reopen it.

7-Work:

This project was distributed among the team members in the following way:

**Medhat Ashraf Abdo:**

GUI,instruction memory,instructions,parse,sift lift,labels,testing ,report.

**Hazem Hamada Abdellatif:**

Compile, run, next ins. Action listener, Data memory ,register file, mux, PC, sign extend ,testing ,report.

**Ahmad Abdallah Mohammad:**

Control unit,Alu control, Alu, adder,testing,report.