MIPS SIMULATOR

Submitted by
Nour El din Osama 16p6043
Youssef Khaled Hussein 16p6040
Mohamed Sameh Abd el hakim 16p3061

MAY 9, 2018

Submitted to
Dr. Cherif Salama
Engineer Ahmed Fathy

Table of Contents

Table of Figures

1. Introduction

This report will discuss a software program which is used to implement low-level MIPS data path (simple data path) but it also has some additional features and some changes had to be made to the path for all implemented functions to work but they will be discussed later on, the program simulates the whole data path.

2. Description

This simulator was implemented by doing a one to one representation of the hardware components into **Java** classes in such a way that every component in the datapath (**register file, ALU, etc..**) has a class that does the same function as in the circuit.

There is a class named **Wire** that represent the values propagated through wires, and another class named Processor that is responsible for integrating all those components into a functional module.

Another class **StringToInt** is responsible for translating assembly language into machine code is added to the **processor** module

Processor module is then embedded into the GUI class to create the interface that the user sees.

3. BONUS Features

The Bonus features implemented are:

1. Graphical User Interface
2. The extra instructions requested (all instructions except pseudo-instructions are supported).
3. The Assembler
4. Including 6 C functional programs and their mips equivalent (Appendix A)

4. Extensions

- New wires were added:
    - One to know what load from the memory, word, half-word, or byte, this wire comes from the control unit to the memory unit
    - Another one was added to the alu control in case of the jr instruction as it is an R-type instruction so the control can't determine it
    - One last wire was added to the control unit in case of jal instruction to make saving of address possible in the $ra register.
- New muxes were added to the data path:
    - jr mux
    - 2 jal muxes
- ALUOP had to become 4 bits to support the extra instructions
- ALU control added a new value for bne

5. Datapath

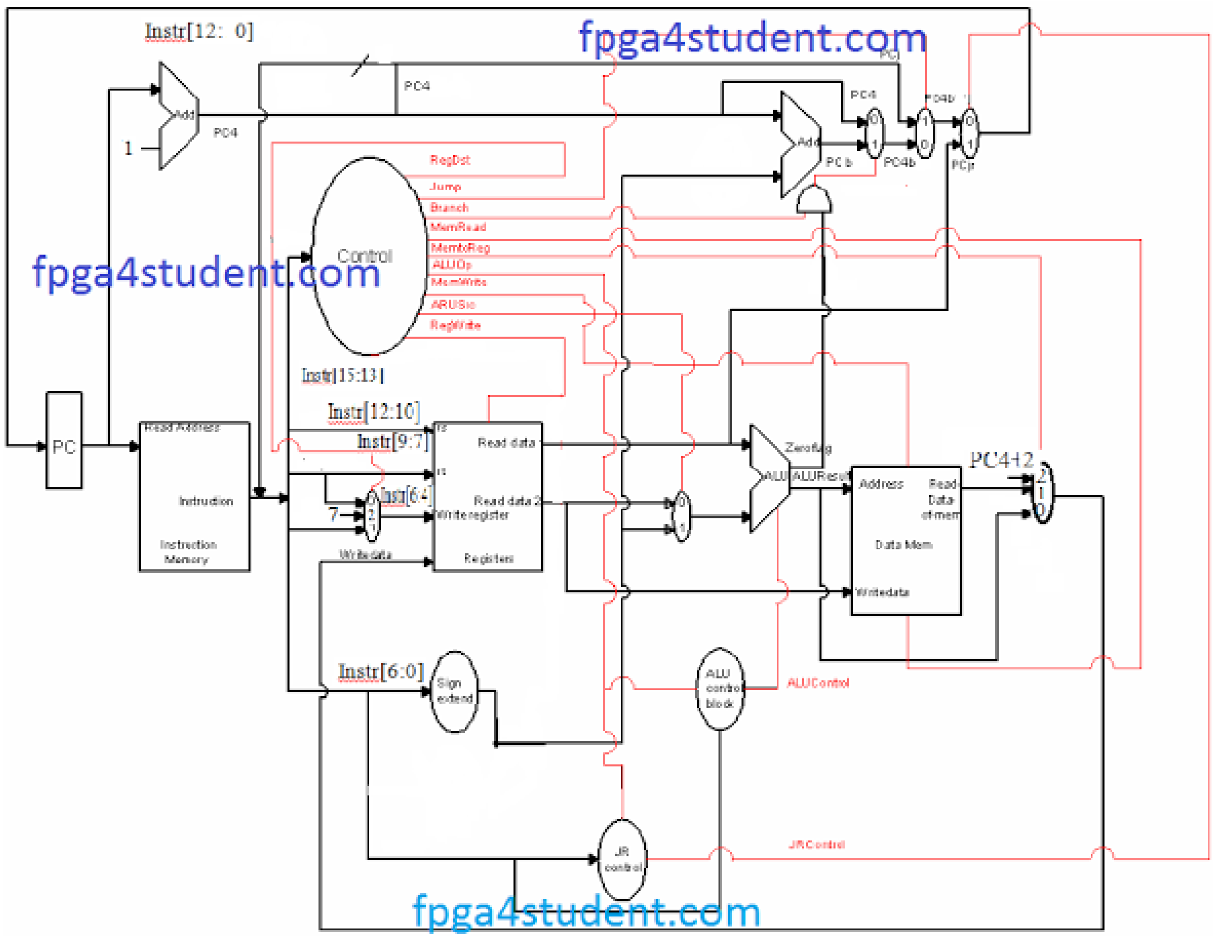


Figure 1 datapath used by simulator

6. Truth Table

The logic diagram is in (Appendix B)

Table 1 the truth table of the control unit

Opcode	*RegDst*	*Jump*	*Branch*	*MemRead*	*MemToReg*	*AluOp*	*MemWrite*	*AluSrc*	*RegWrite*	*LoadType*	*JAL*
000000	*1*	*0*	*0*	*0*	*0*	*0 0 1 0*	*0*	*0*	*1*	*X*	*0*
001111	*0*	*0*	*0*	*0*	*0*	*0 1 0 0*	*0*	*1*	*1*	*X*	*0*
100011	*0*	*0*	*0*	*1*	*1*	*0 0 0 0*	*0*	*1*	*1*	*0 0 0*	*0*
101011	*X*	*0*	*0*	*0*	*X*	*0 0 0 0*	*1*	*1*	*0*	*0 0 0*	*0*
000100	*X*	*0*	*1*	*0*	*X*	*0 0 0 1*	*0*	*0*	*0*	*X*	*0*
000010	*X*	*1*	*0*	*0*	*X*	*X*	*0*	*X*	*0*	*X*	*0*
000011	*X*	*1*	*0*	*0*	*X*	*X*	*0*	*X*	*1*	*X*	*1*
001000	*0*	*0*	*0*	*0*	*0*	*0 0 0 0*	*0*	*1*	*1*	*X*	*0*
100000	*0*	*0*	*0*	*1*	*1*	*0 0 0 0*	*0*	*1*	*1*	*0 1 1*	*0*
100100	*0*	*0*	*0*	*1*	*1*	*0 0 0 0*	*0*	*1*	*1*	*1 0 0*	*0*
101000	*X*	*0*	*0*	*0*	*X*	*0 0 0 0*	*1*	*1*	*0*	*1 0 0*	*0*
100001	*0*	*0*	*0*	*1*	*1*	*0 0 0 0*	*0*	*1*	*1*	*0 0 1*	*0*
100101	*0*	*0*	*0*	*1*	*1*	*0 0 0 0*	*0*	*1*	*1*	*0 1 0*	*0*
101001	*X*	*0*	*0*	*0*	*X*	*0 0 0 0*	*1*	*1*	*0*	*0 1 0*	*0*
001010	*0*	*0*	*0*	*0*	*0*	*0 0 1 1*	*0*	*1*	*1*	*X*	*0*
001001	*0*	*0*	*0*	*0*	*0*	*0 1 0 1*	*0*	*1*	*1*	*X*	*0*
001100	*0*	*0*	*0*	*0*	*0*	*0 1 1 0*	*0*	*1*	*1*	*X*	*0*
001101	*0*	*0*	*0*	*0*	*0*	*0 1 1 1*	*0*	*1*	*1*	*X*	*0*
000101	*X*	*0*	*1*	*0*	*X*	*1 1 1 1*	*0*	*0*	*0*	*X*	*0*

7. User Guide

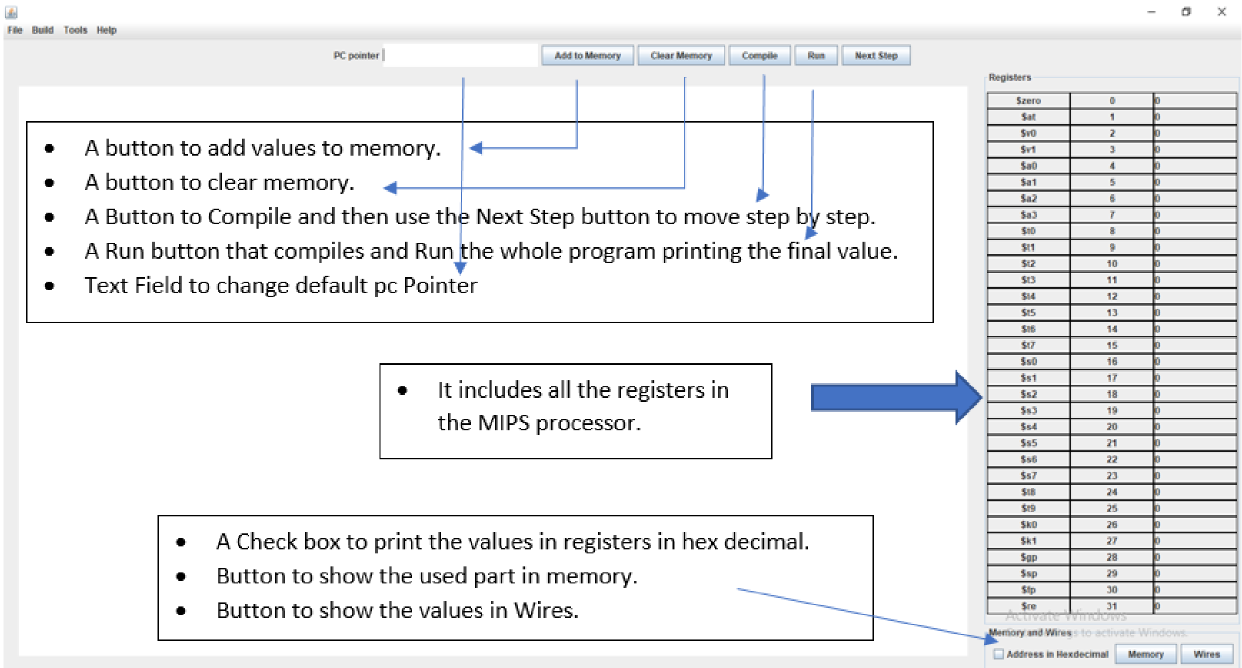


Figure 2 program main buttons

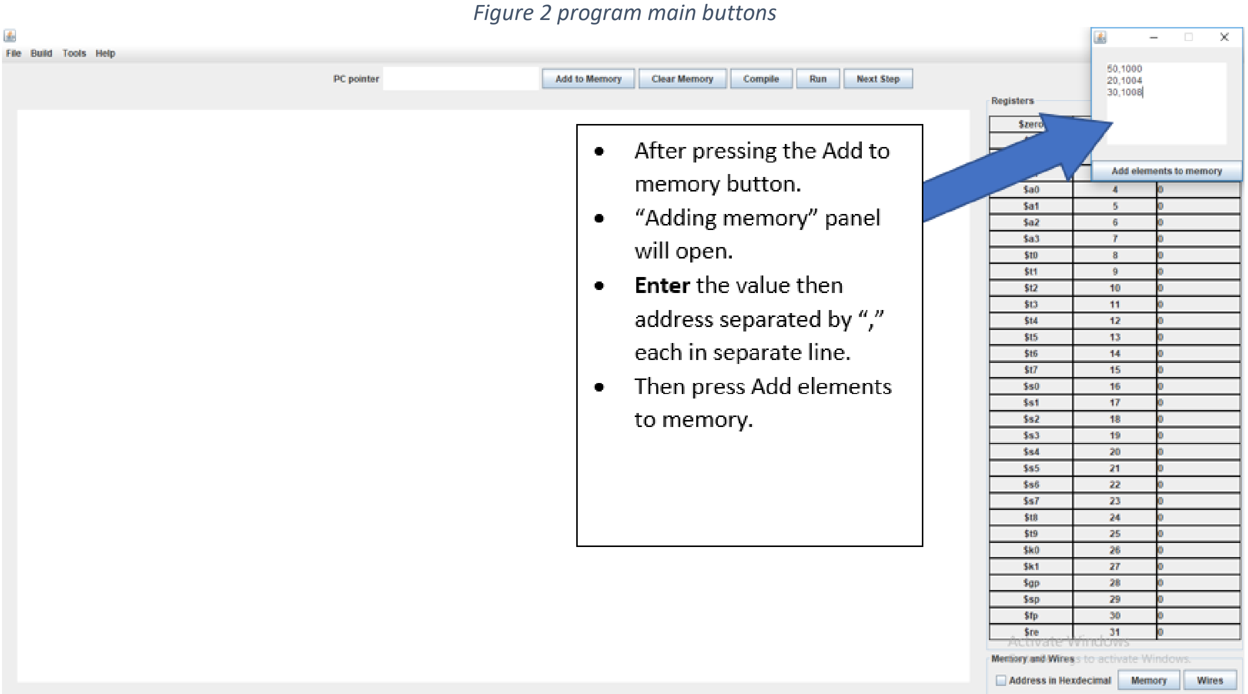


Figure 3 manually inserting data into data memory

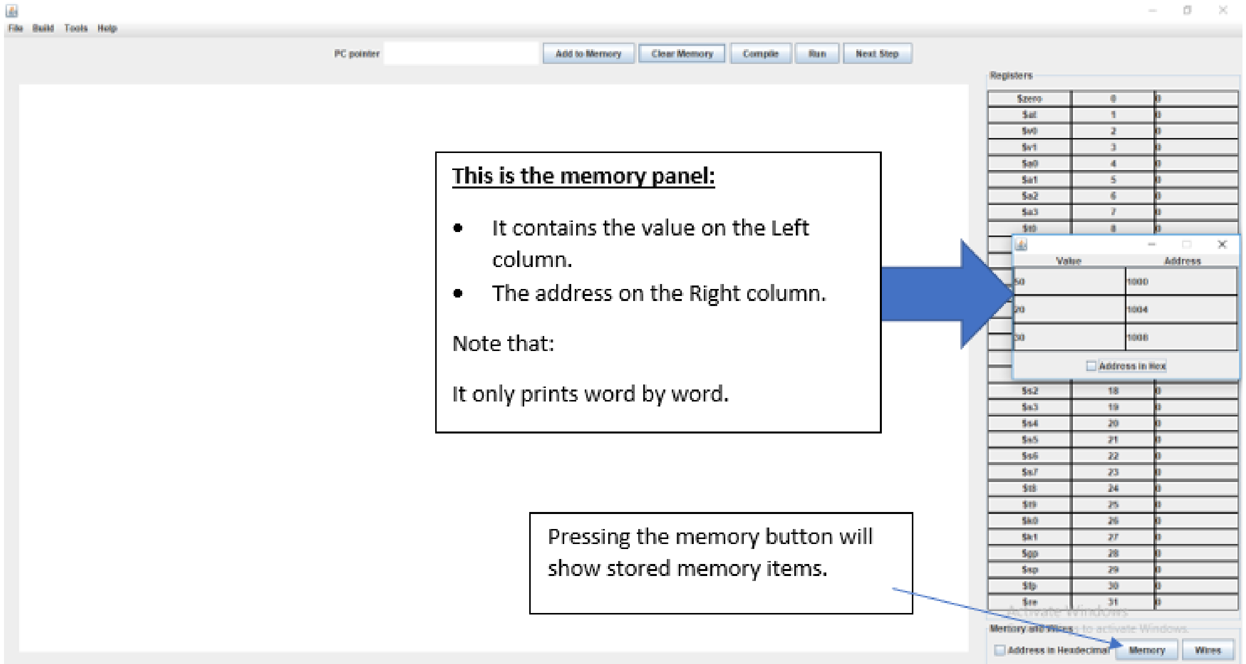


Figure 4 memory panel layout

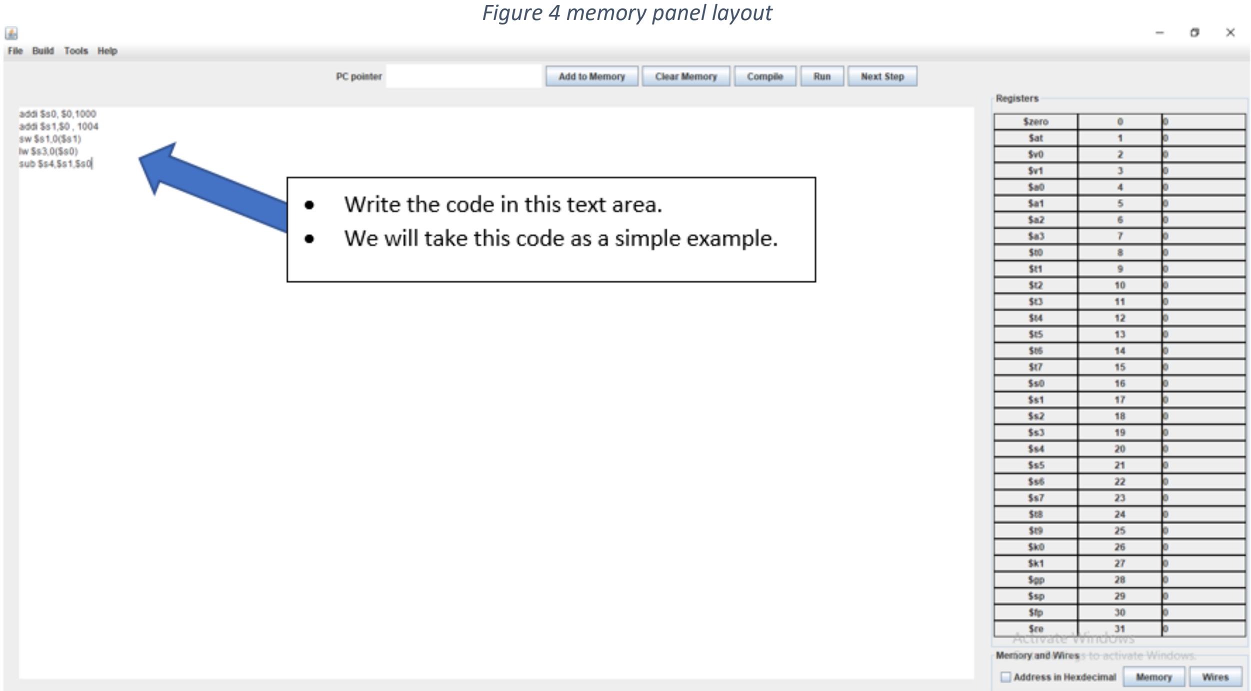


Figure 5 text area

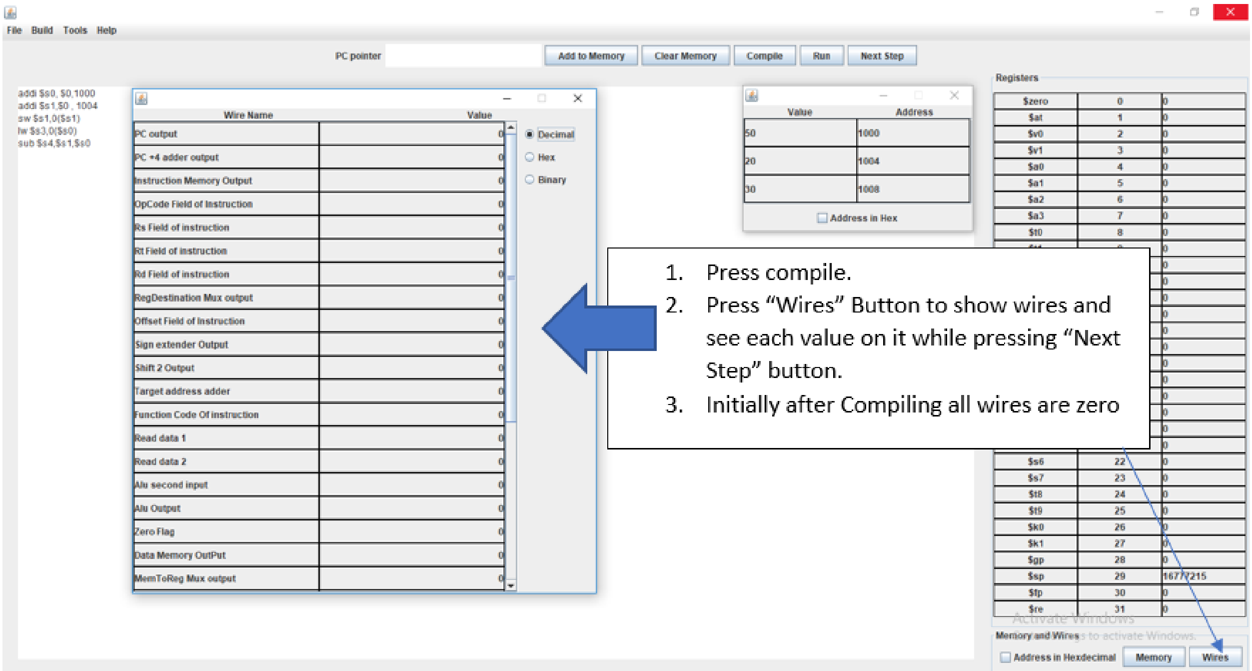


Figure 6 wire values label

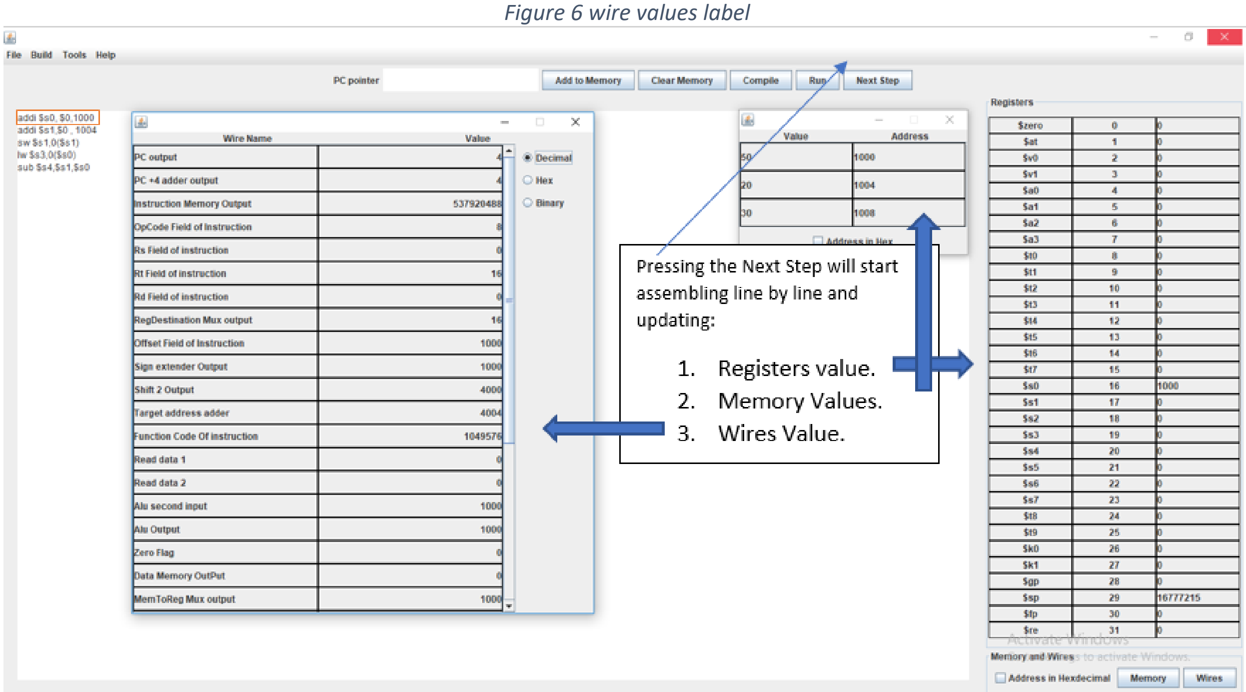


Figure 7 the different values of data during each cycle

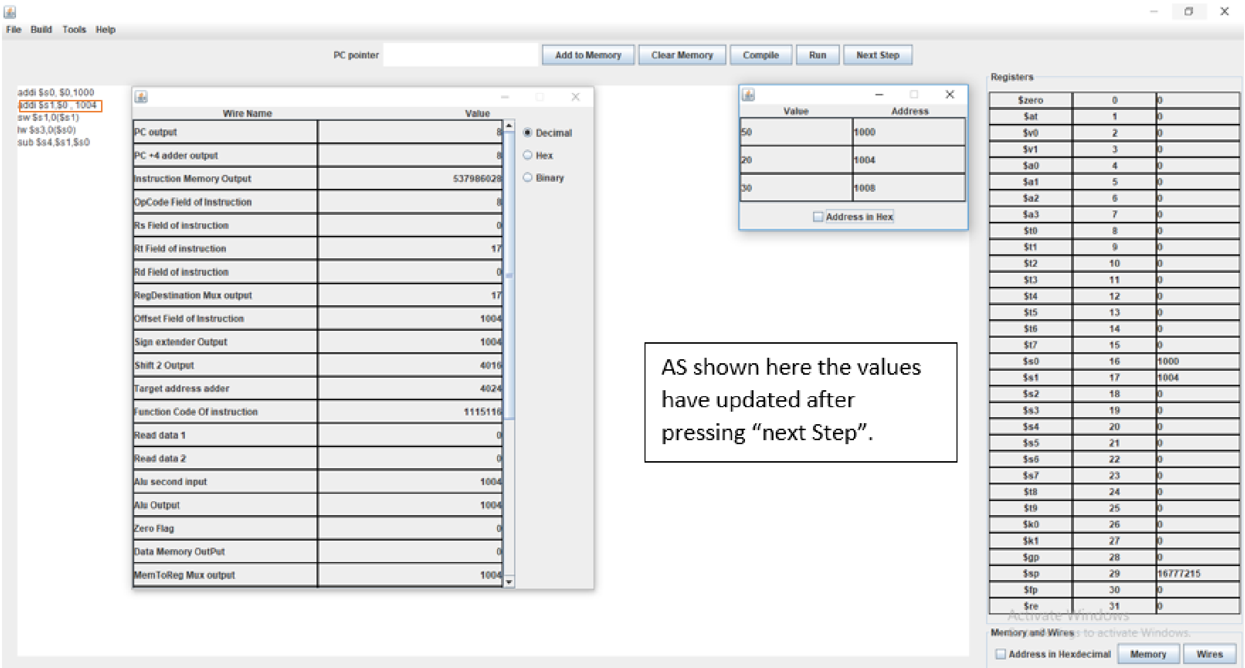


Figure 8 the next button function

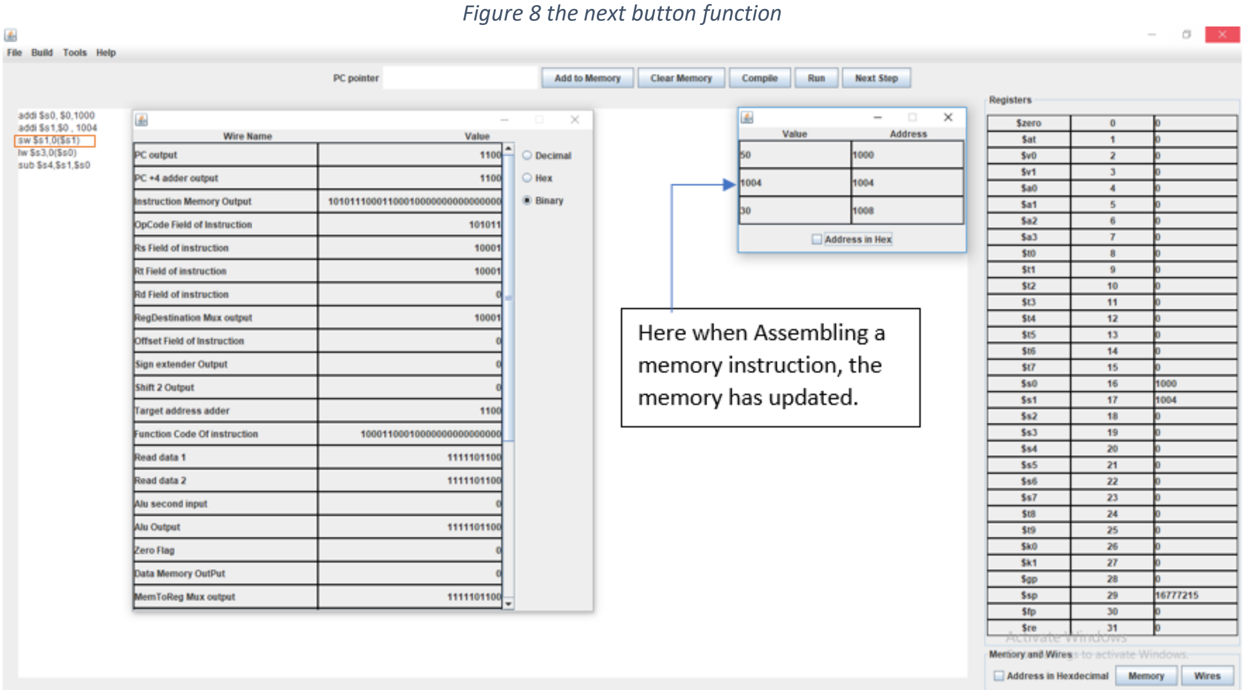


Figure 9 the memory update mechanism

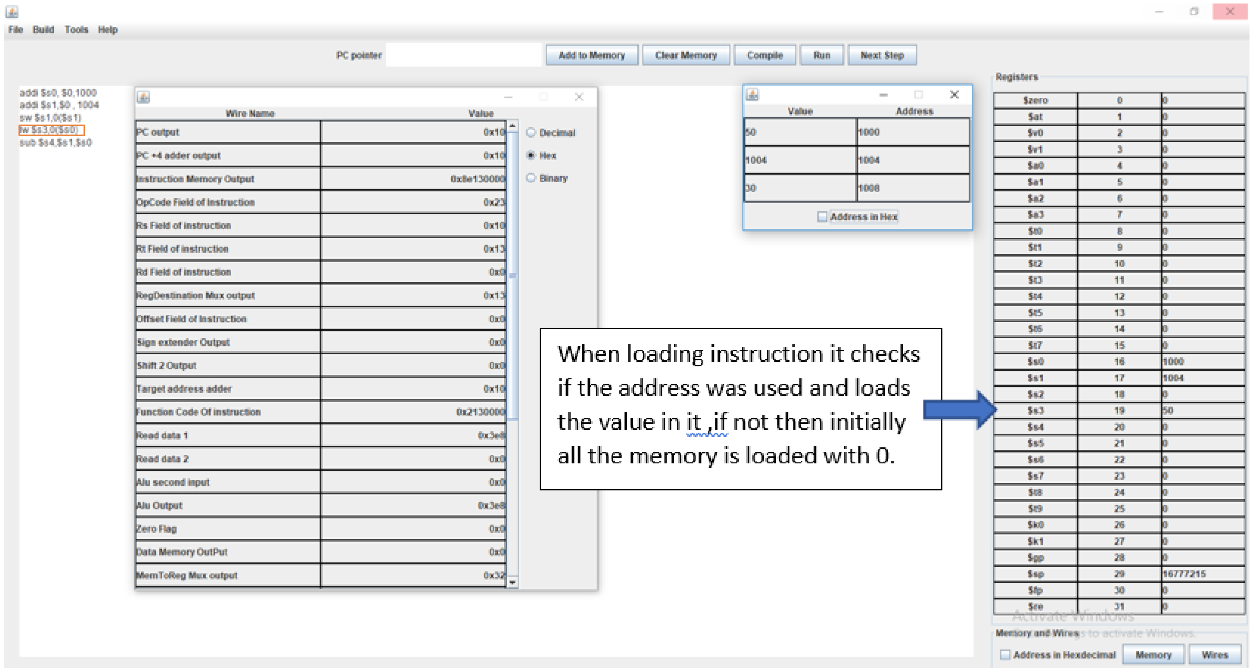


Figure 10 indicates how loads from memory works

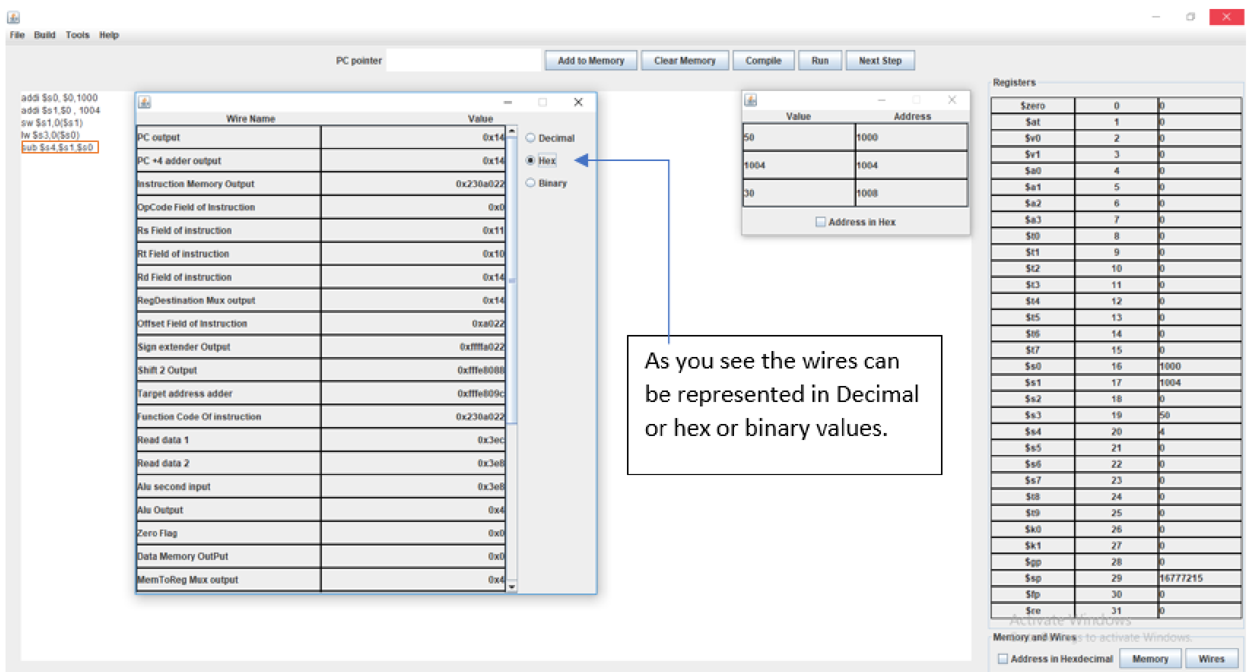


Figure 11 different number format of wire values

8. Work Load

the project had 4 main areas:

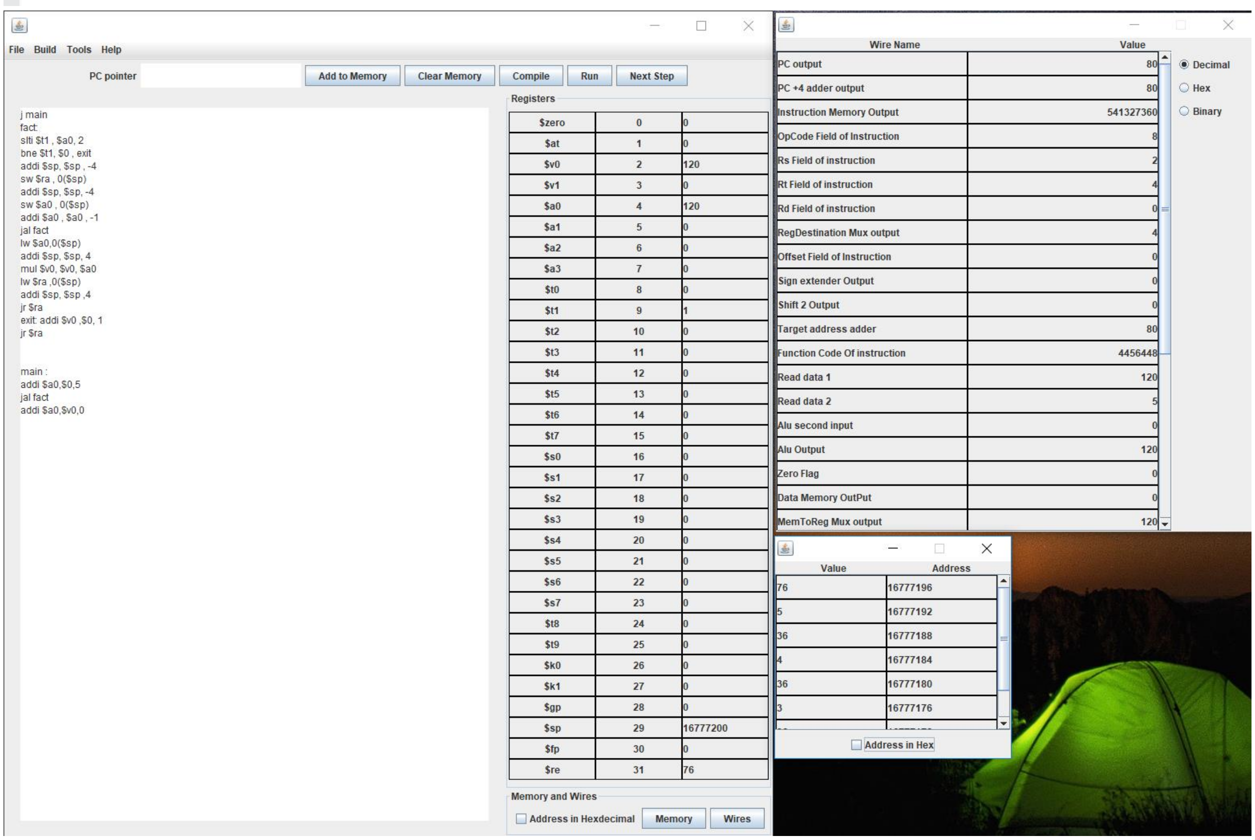
1. The GUI handled by Nour El din Osama
2. The Assembler handled by Mohamed Sameh
3. The control values and extra instructions handled by Youssef Khaled
4. The implementation of individual components and this report were fairly distributed on the three of us

9. Appendix A

This appendix has a list of functional c programs and their equivalent mips programs for reference.

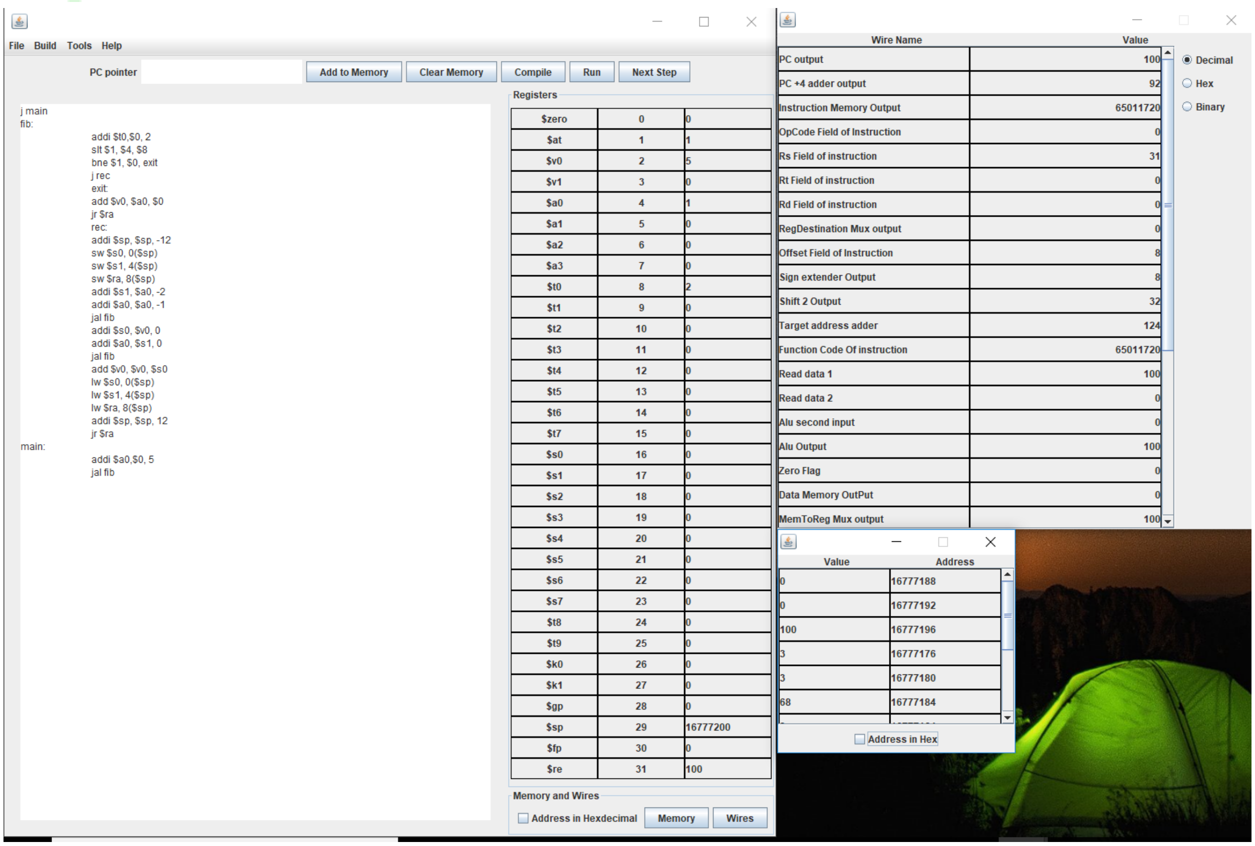
Program 1: recursive factorial function

```
#include<iostream>
using namespace std;
int fact(int n){
    if (n < 2)
        return 1;
    else
        return n * fact( n - 1);
}
void main(){
    int m = fact(5);
}
```



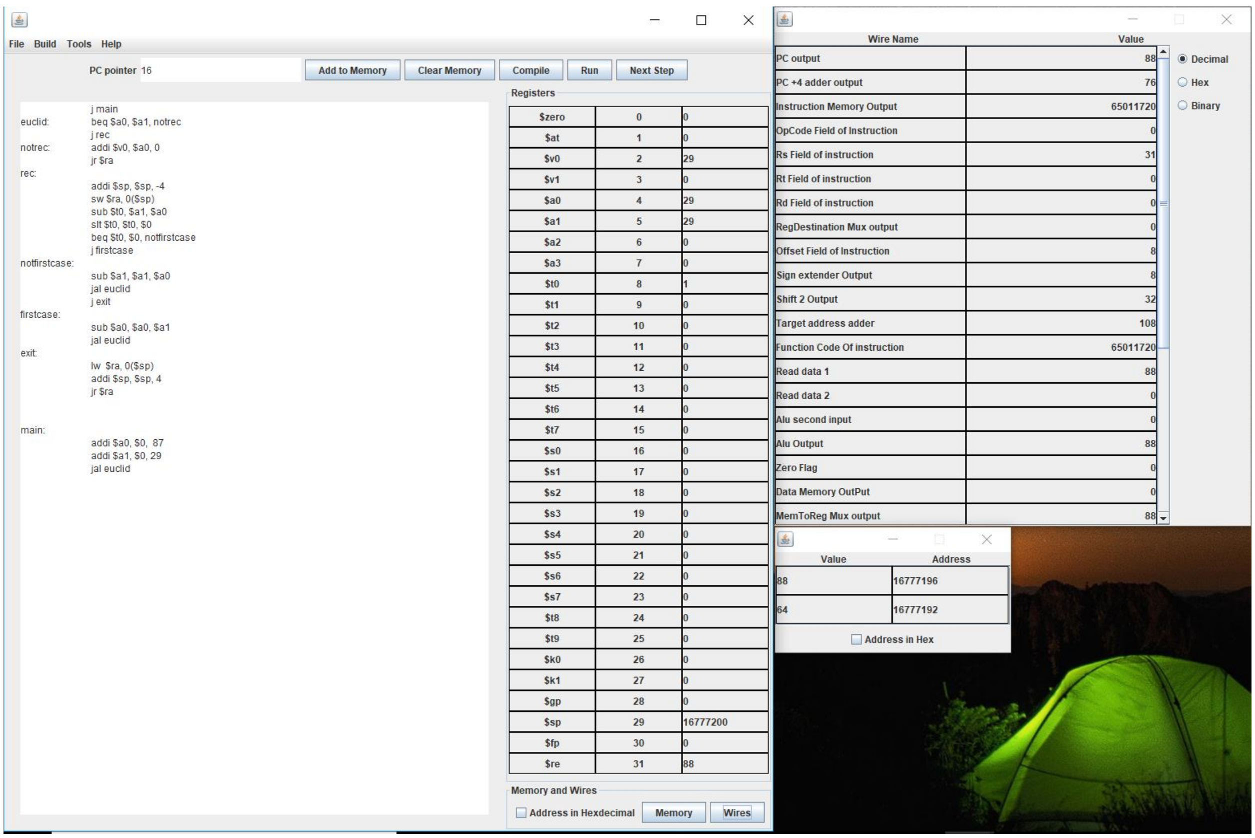
Program 2: recursive Fibonacci calculator

```cpp
#include<iostream>
using namespace std;
int fib(int n){
    if(n > 1)
        return fib(n-1)+fib(n-2);
    else if (n == 0)
        return 0;
    else if(n == 1)
        return 1;
}
void main(){
    int m = fib (5);
}
```



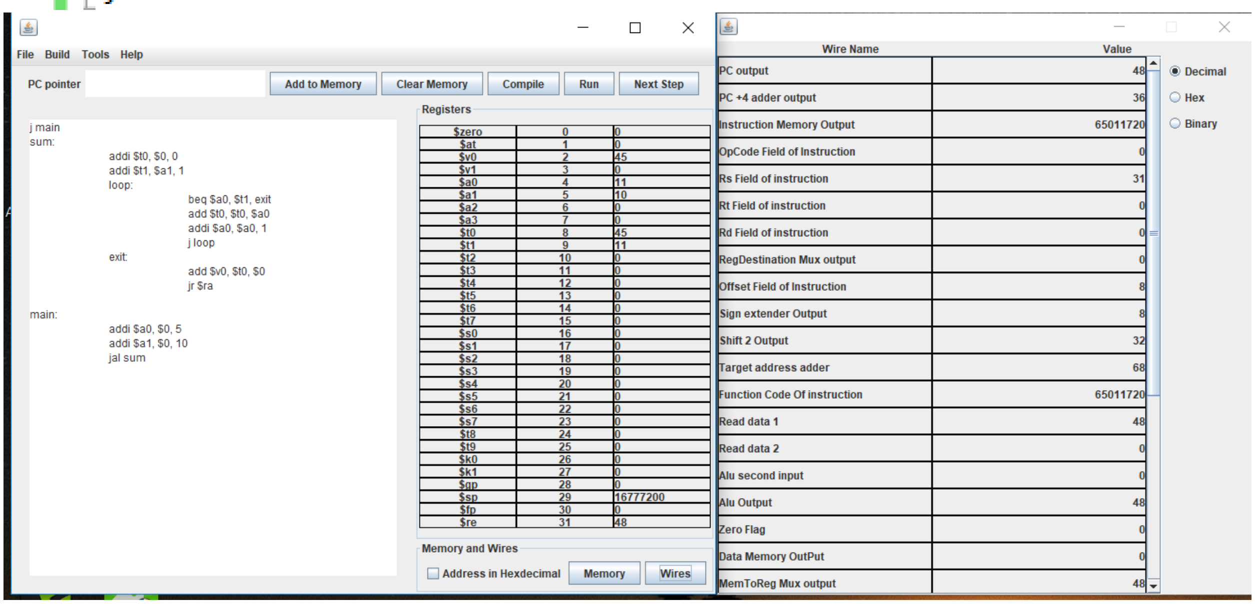
Program 3: recursive Euclidian HCF calculator

```
#include<iostream>
using namespace std;
int euclid(int a, int b){
    if ( a == b )
        return a;
    else if ( a < b )
        return euclid(a, b-a);
    else
        return euclid(a-b,b);
}
void main(){
    int m = euclid(87,29);
    cout<<m;
    cin>>m;
}
```



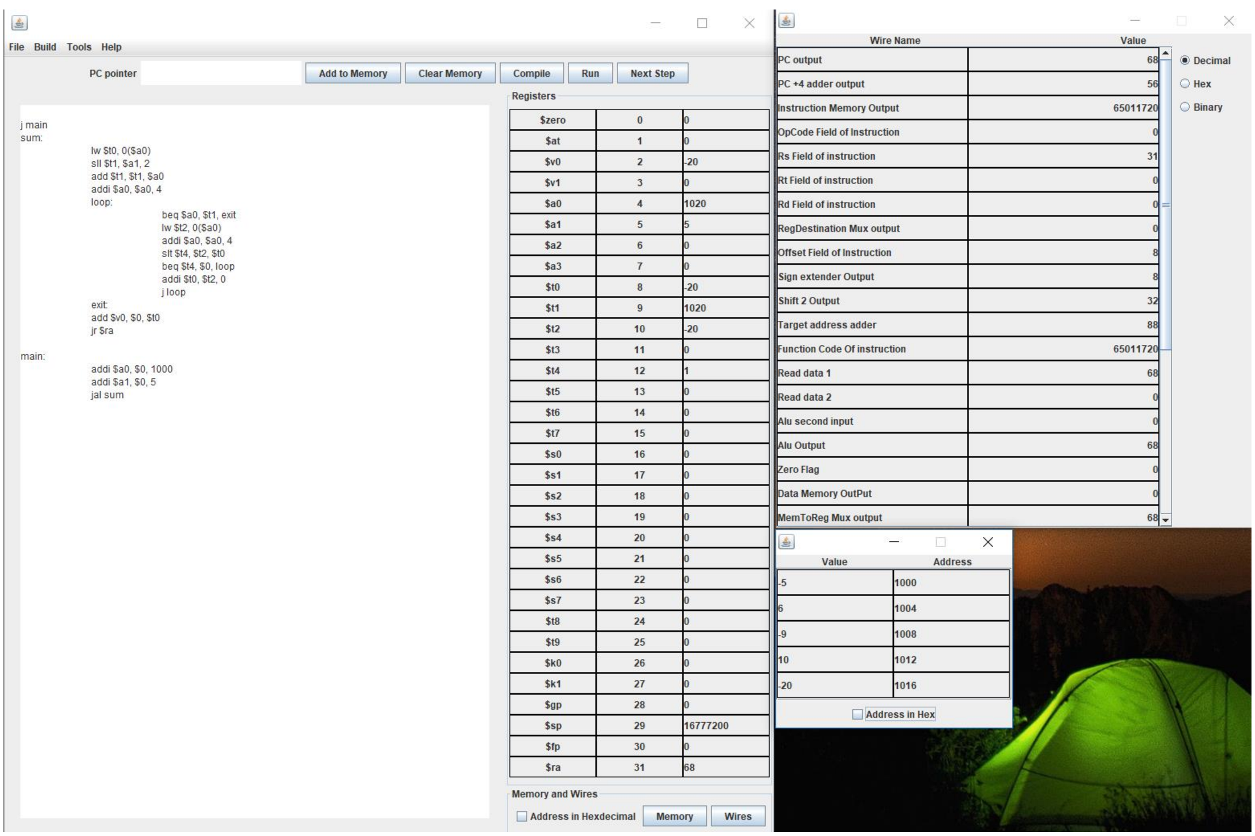
Program 4: sum of a range of numbers from a to b such that a < b

```cpp
#include<iostream>
using namespace std;
int rangeSum(int a, int b){
    int sum = 0;
    for (int i = a; i <1+b; i++)
        sum = sum + i;
    return sum;
}
void main(){
    int m = rangeSum(5,10);
    cout<<m;
    cin>>m;
}
```



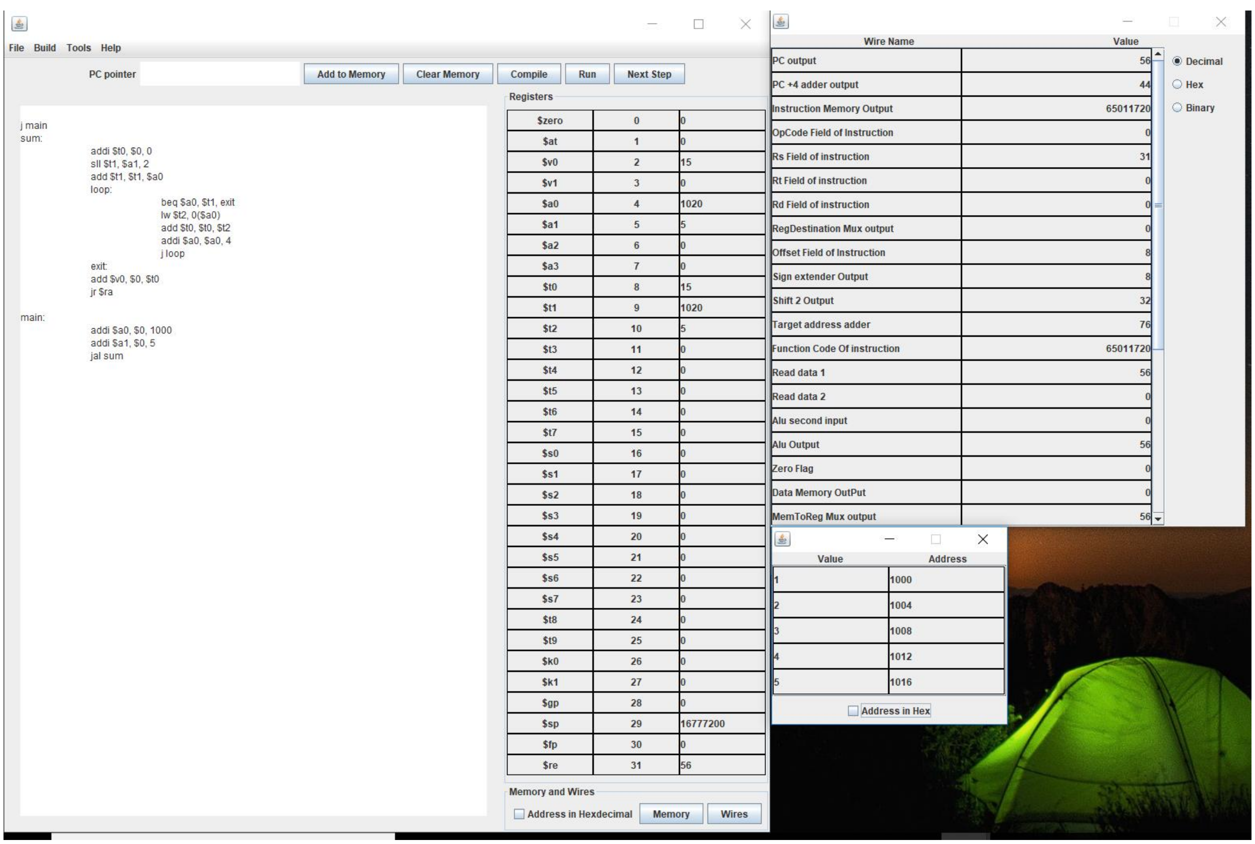
Program 5: determines minimum element in an array

```cpp
#include<iostream>
using namespace std;
int arrayMin(int* a, int n){
    int min = *a;
    for(int *i = a + 1; i < a+n; i++)
        if (*i < min)
            min = *i;
    return min;

}
void main(){
    int arr[] = {-5, 6, -9, 10, -20};
    int m = arrayMin(arr,5);
    cout<<m;
    cin>>m;
}
```



Program 6: determines sum of array elements

```cpp
#include<iostream>
using namespace std;
int arraySum(int* a, int n){
    int sum = 0;
    for(int *i = a;i < a+n;i++){
        sum = sum + *i;
    }
    return sum;

}
void main(){
    int arr[] = {1,2,3,4,5};
    int m = arraySum(arr,5);
    cout<<m;
    cin>>m;
}
```



10. Appendix B

This appendix has the logic diagram needed for the control unit

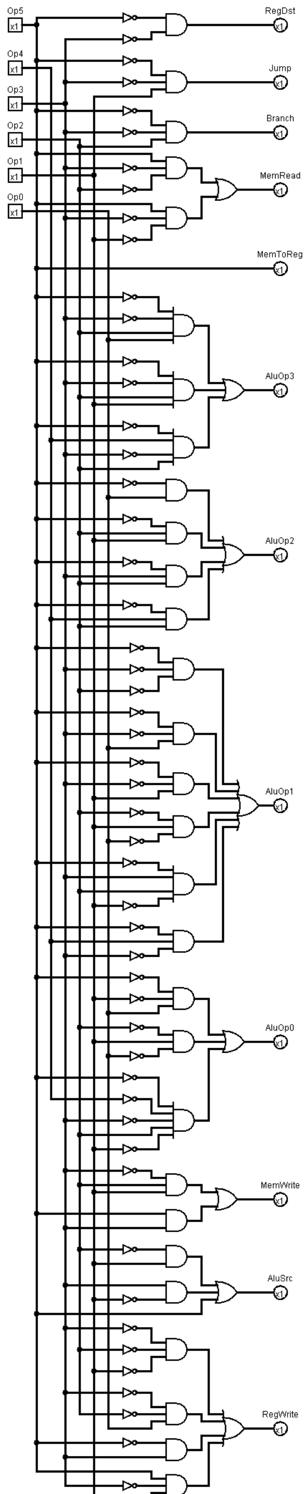


Figure 13 the first 12 bits of the logic circuit of the control unit

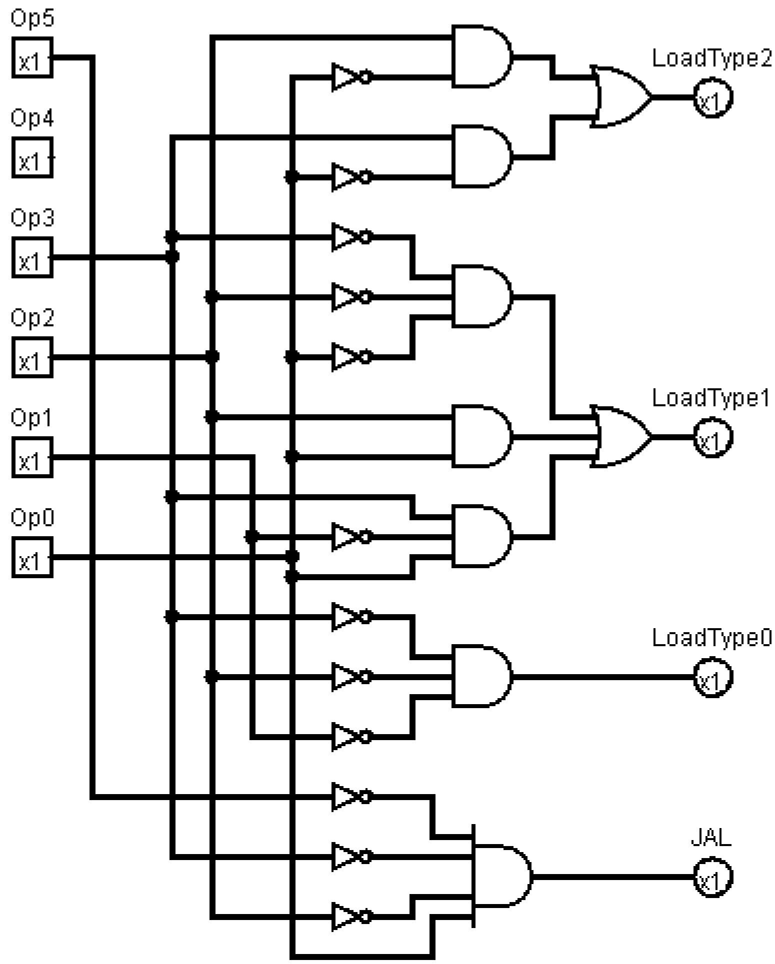


Figure 12 the last 4 bits of the logic circuit of the control unit