



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

1. Introduction	1
2. Description	1
3. BONUS Features.....	1
4. Extensions	2
5. Datapath	3
6. Truth Table.....	4
7. User Guide	5
8. Work Load	10
9. Appendix A	11
10. Appendix B	17

Table of Figures

Figure 1 datapath used by simulator	3
Table 1 the truth table of the control unit.....	4
Figure 2 program main buttons	5
Figure 3 manually inserting data into data memory	5
Figure 4 memory panel layout.....	6
Figure 5 text area	6
Figure 6 wire values label.....	7
Figure 7 the different values of data during each cycle.....	7
Figure 8 the next button function.....	8
Figure 9 the memory update mechanism.....	8
Figure 10 indicates how loads from memory works.....	9
Figure 11 different number format of wire values	9
Figure 12 the first 12 bits of the logic circuit of the control unit.....	17
Figure 13 the last 4 bits of the logic circuit of the control unit.....	17

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

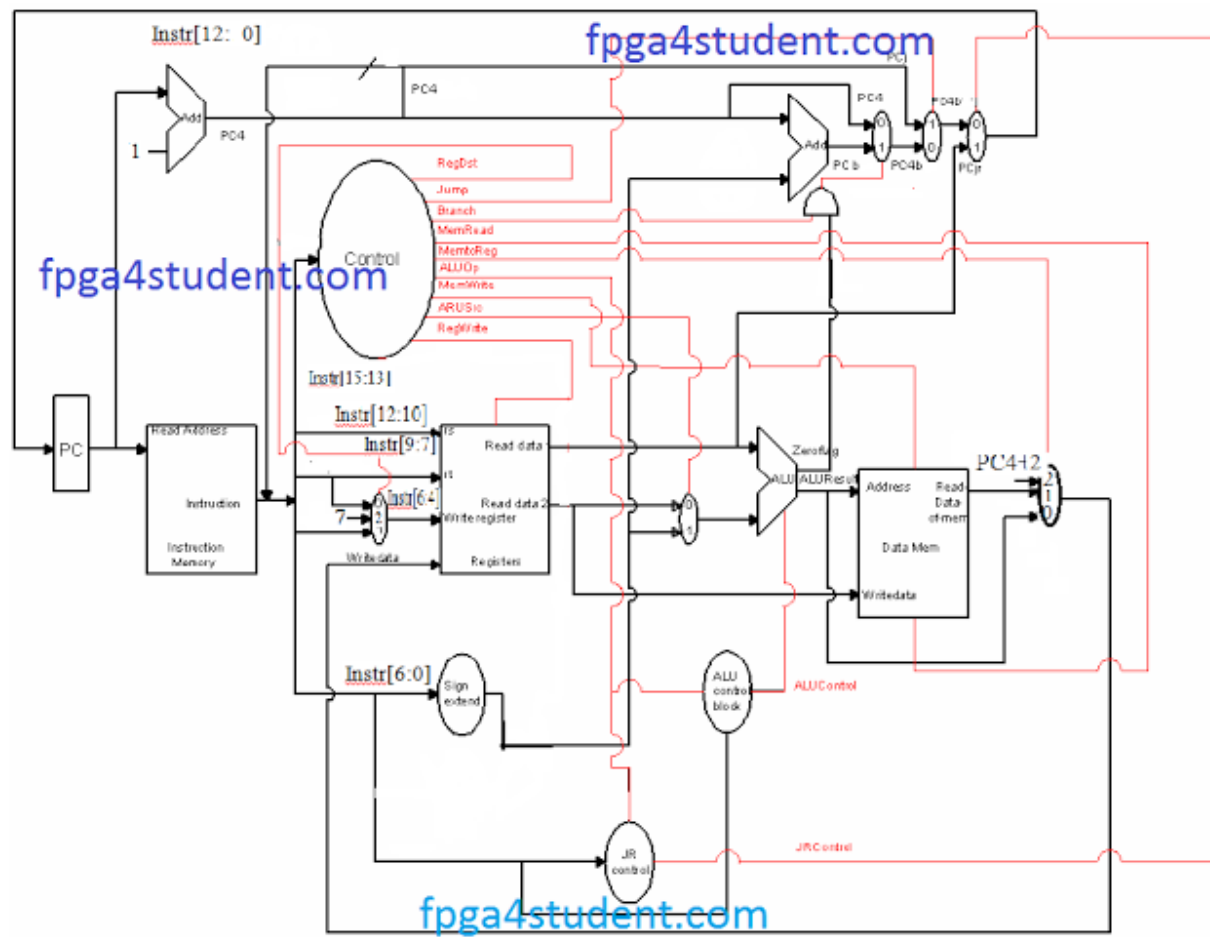


Figure 1 datapath used by simulator

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	0010	0	0	1	X	0
001111	0	0	0	0	0	0100	0	1	1	X	0
100011	0	0	0	1	1	0000	0	1	1	000	0
101011	X	0	0	0	X	0000	1	1	0	000	0
000100	X	0	1	0	X	0001	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	0000	0	1	1	X	0
100000	0	0	0	1	1	0000	0	1	1	011	0
100100	0	0	0	1	1	0000	0	1	1	100	0
101000	X	0	0	0	X	0000	1	1	0	100	0
100001	0	0	0	1	1	0000	0	1	1	001	0
100101	0	0	0	1	1	0000	0	1	1	010	0
101001	X	0	0	0	X	0000	1	1	0	010	0
001010	0	0	0	0	0	0011	0	1	1	X	0
001001	0	0	0	0	0	0101	0	1	1	X	0
001100	0	0	0	0	0	0110	0	1	1	X	0
001101	0	0	0	0	0	0111	0	1	1	X	0
000101	X	0	1	0	X	1111	0	0	0	X	0

7. User Guide

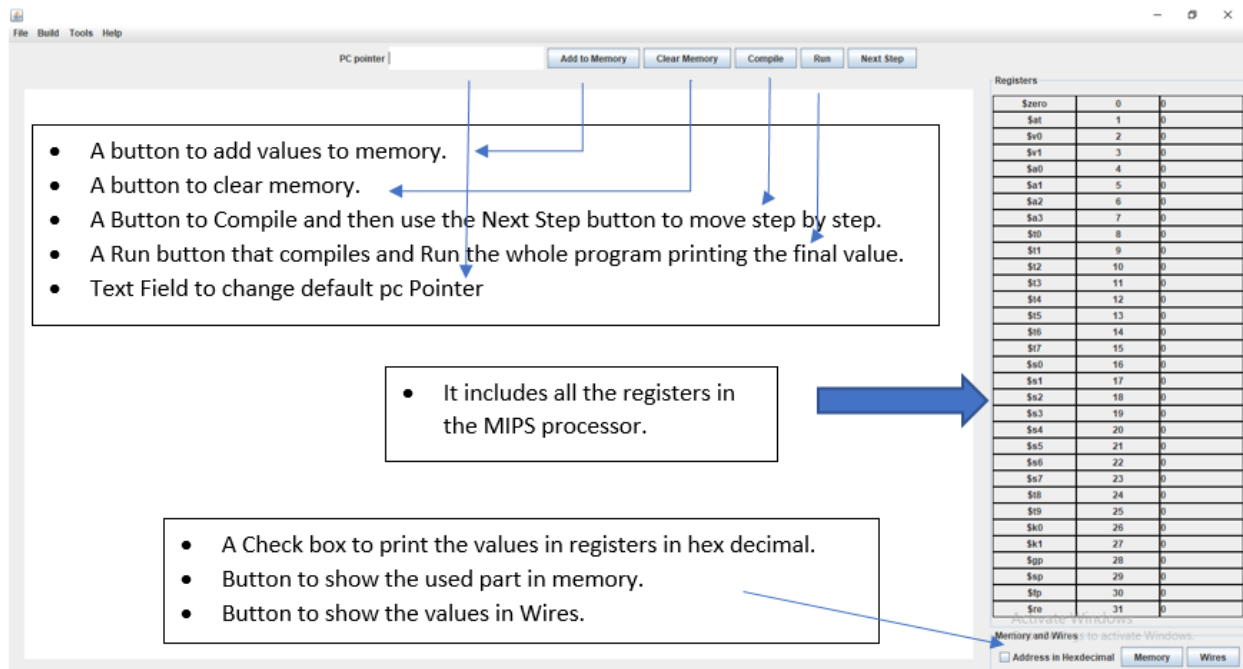


Figure 2 program main buttons

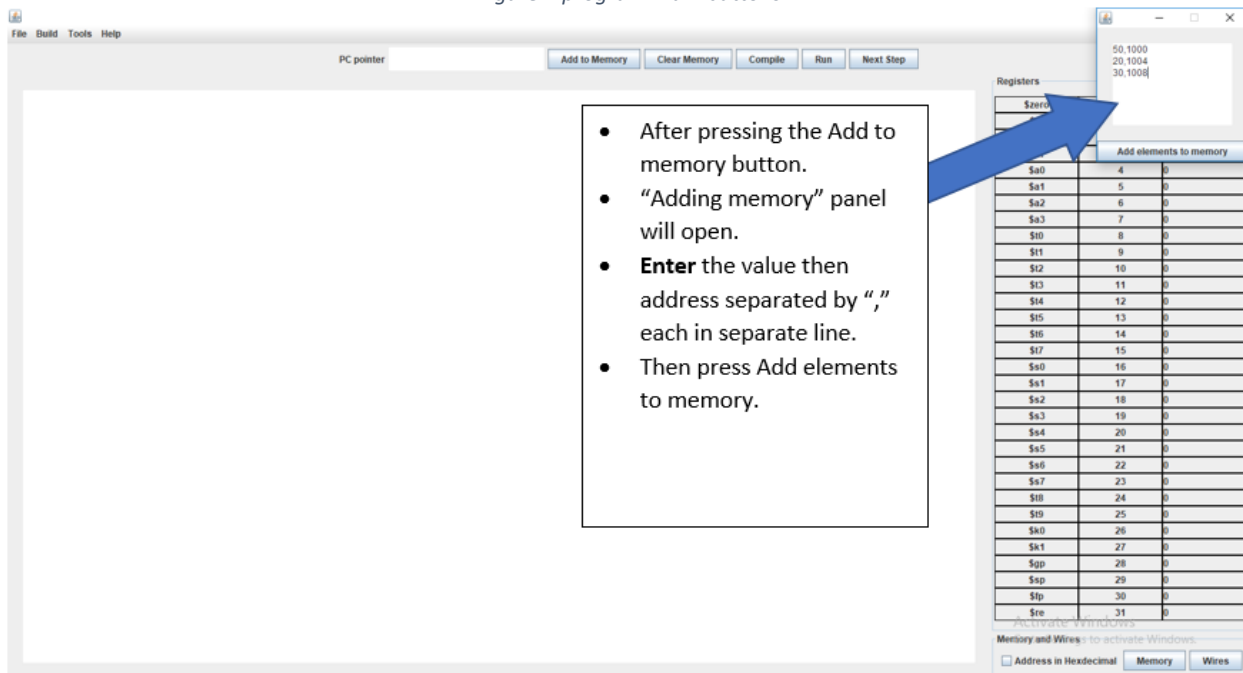


Figure 3 manually inserting data into data memory

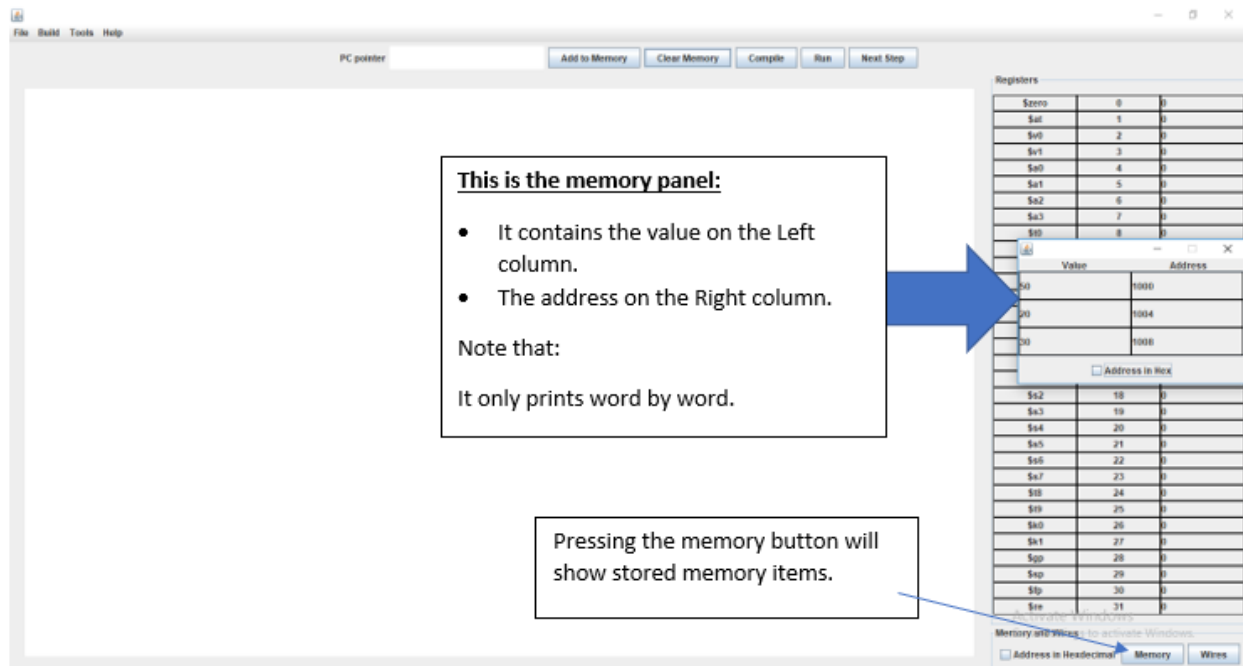


Figure 4 memory panel layout

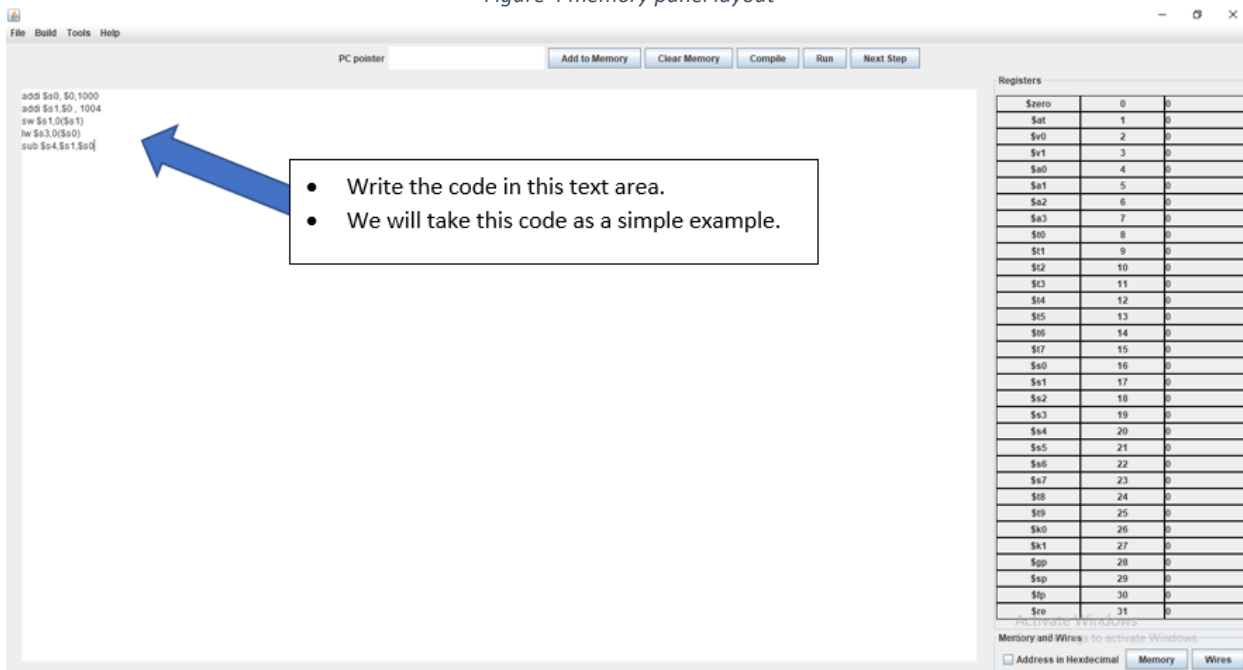


Figure 5 text area

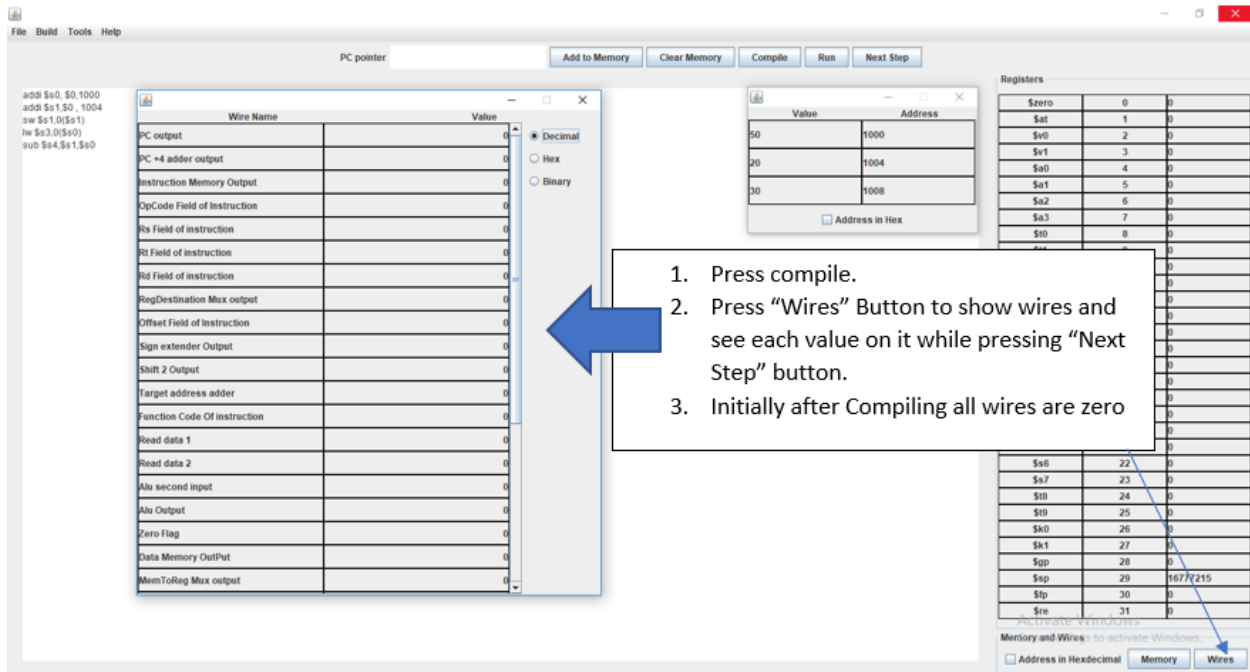


Figure 6 wire values label

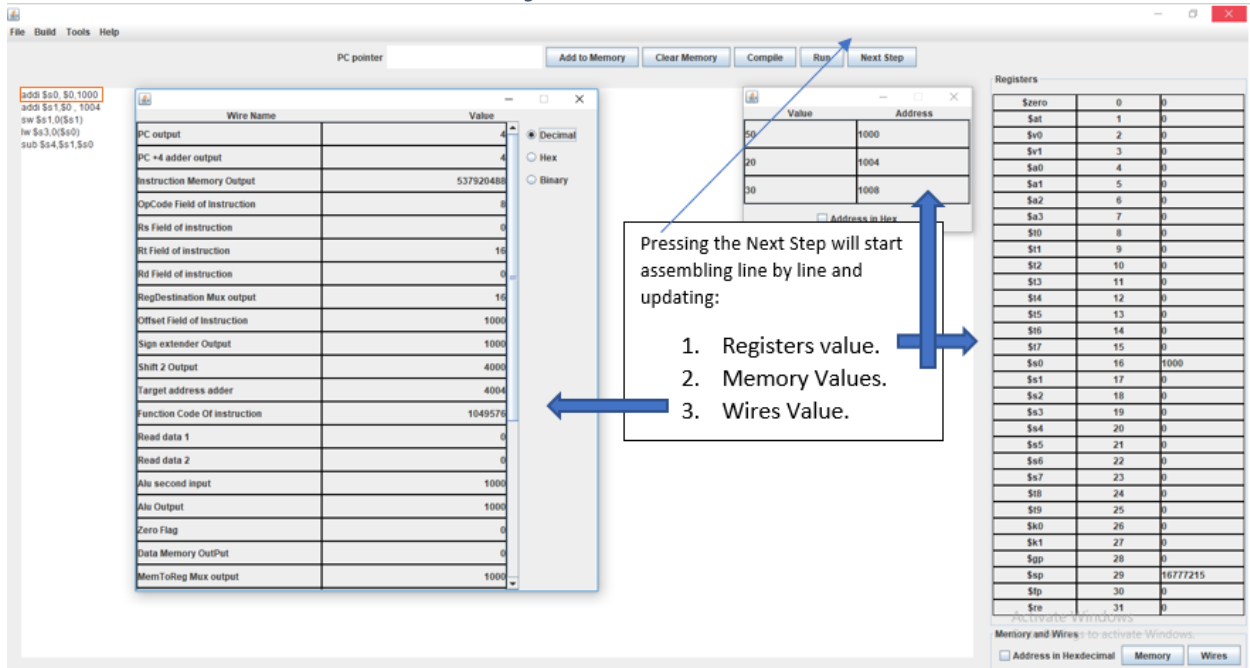


Figure 7 the different values of data during each cycle

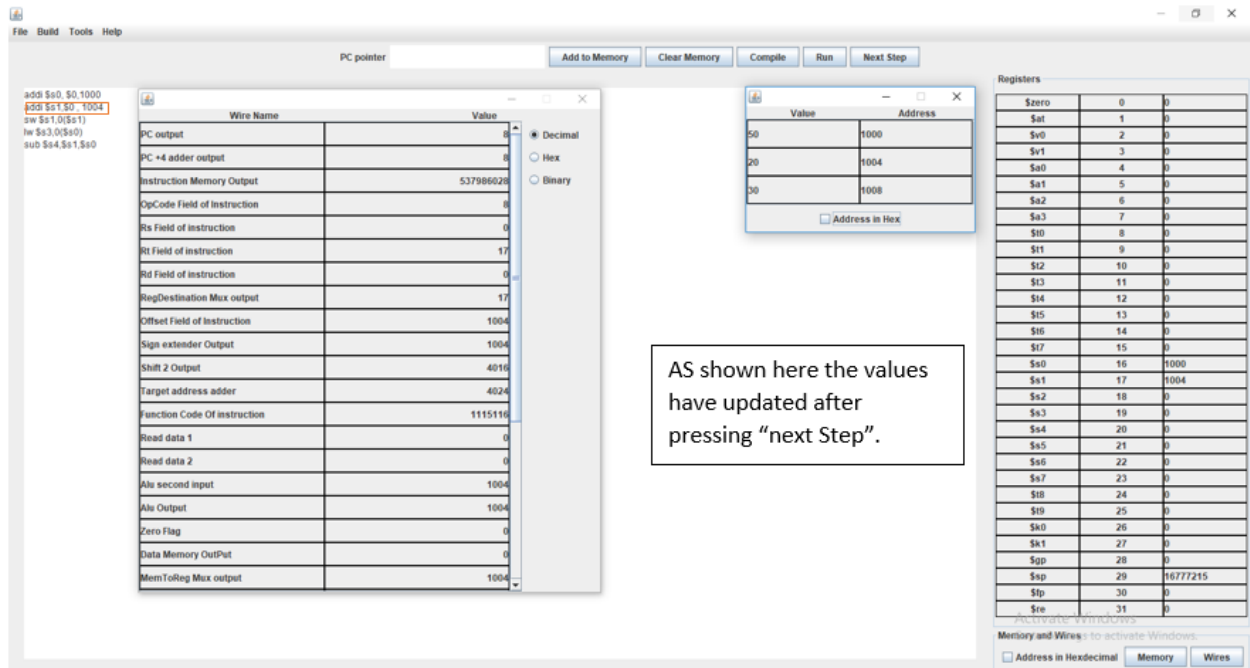


Figure 8 the next button function

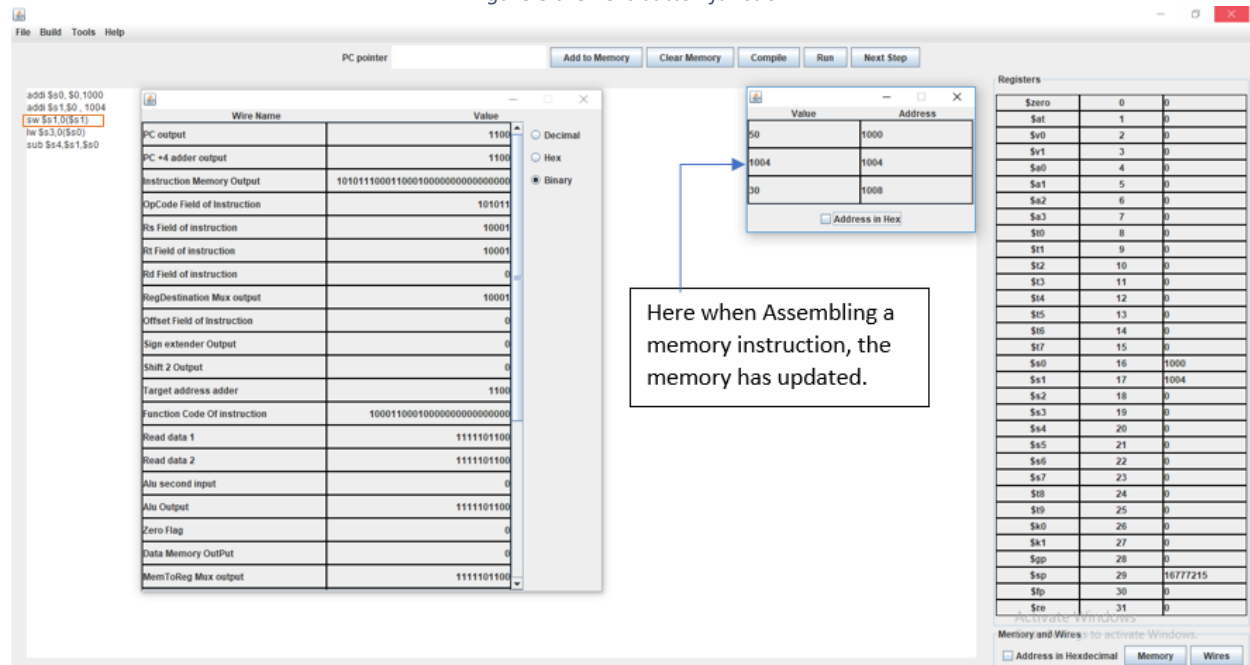


Figure 9 the memory update mechanism

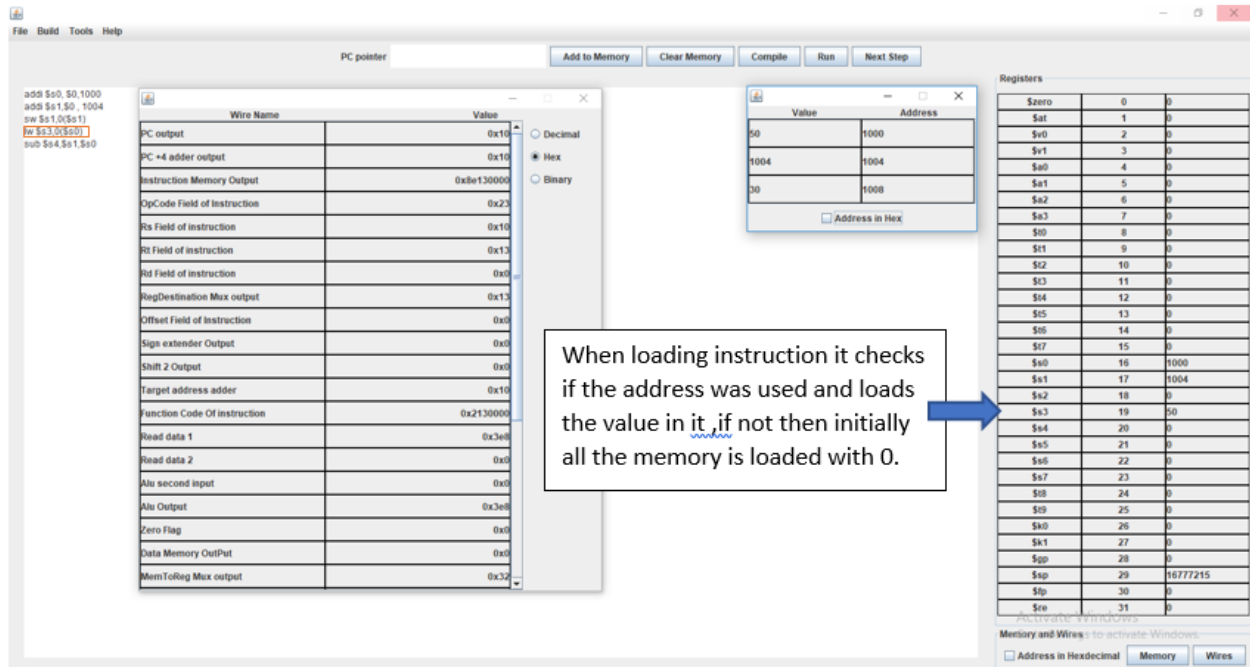


Figure 10 indicates how loads from memory works

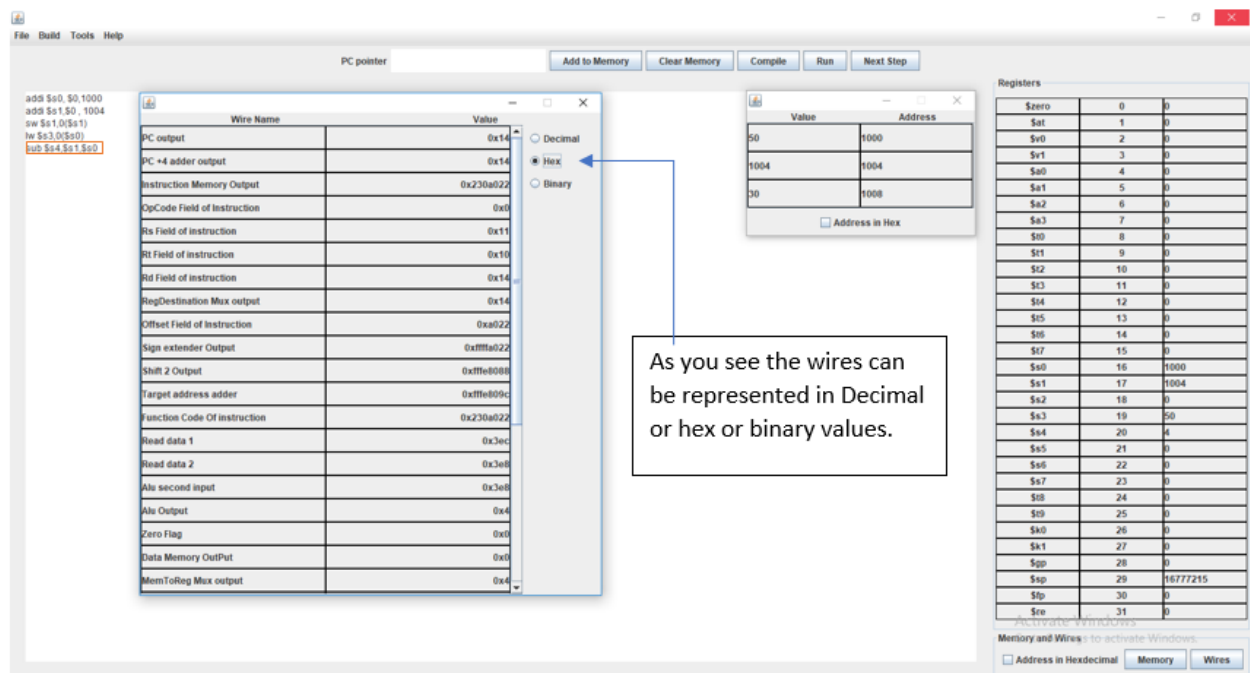


Figure 11 different number format of wire values

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

```
1  #include<iostream>
2  using namespace std;
3  int fact(int n){
4      if (n < 2)
5          return 1;
6      else
7          return n * fact( n - 1);
8  }
9  void main(){
10     int m = fact(5);
11 }
```

The screenshot displays a MIPS simulator interface with the following components:

- Assembly Code:** The left pane shows the assembly translation of the C program. It includes instructions for setting up the stack frame, calling the recursive function `fact`, and returning the result to `main`.
- Registers:** The central pane lists MIPS registers (\$zero through \$s31) and their current values. For example, `$s0` holds the value 120, which is the result of `fact(5)`.
- Wire Outputs:** The right pane shows the values of various internal wires, such as the PC output (80), Instruction Memory Output (541327360), and the ALU output (120).
- Memory and Wires:** At the bottom, there are controls for viewing memory and wires, including checkboxes for "Address in Hexadecimal" and "Memory" vs "Wires".

Program 2: recursive Fibonacci calculator

```

1  #include<iostream>
2  using namespace std;
3  int fib(int n){
4      if(n > 1)
5          return fib(n-1)+fib(n-2);
6      else if (n == 0)
7          return 0;
8      else if(n == 1)
9          return 1;
10 }
11 void main(){
12     int m = fib (5);
13 }

```

The screenshot displays a MIPS simulator interface with the following components:

- PC pointer:** A button labeled "PC pointer" with sub-buttons "Add to Memory", "Clear Memory", "Compile", "Run", and "Next Step".
- Assembly Code:**

```

j main
fib:
    addi $t0,$0,2
    sllt $t1,$4,$8
    bne $t1,$0,exit
    jr $ra
    rec:
    addi $sp,$sp,-12
    sw $a0,0($sp)
    sw $a1,4($sp)
    addi $a1,$a0,-2
    jal fib
    addi $a0,$v0,0
    addi $a0,$a1,0
    jal fib
    addi $v0,$v0,$a0
    lw $a0,0($sp)
    lw $a1,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
main:
    addi $a0,$0,5
    jal fib

```
- Registers:** A table showing the state of MIPS registers.

Register	Value
\$zero	0
\$at	1
\$v0	2
\$v1	3
\$a0	4
\$a1	5
\$a2	6
\$a3	7
\$t0	8
\$t1	9
\$t2	10
\$t3	11
\$t4	12
\$t5	13
\$t6	14
\$t7	15
\$a0	16
\$a1	17
\$a2	18
\$a3	19
\$a4	20
\$a5	21
\$a6	22
\$a7	23
\$t8	24
\$t9	25
\$k0	26
\$k1	27
\$gp	28
\$sp	29
\$fp	30
\$re	31
- Wire Name and Value:** A table showing the values of various wires.

Wire Name	Value
PC output	100
PC +4 adder output	92
Instruction Memory Output	65011720
OpCode Field of Instruction	0
Rs Field of Instruction	31
Rt Field of Instruction	0
Rd Field of Instruction	0
RegDestination Mux output	0
Offset Field of Instruction	0
Sign extender Output	0
Shift 2 Output	32
Target address adder	124
Function Code Of Instruction	65011720
Read data 1	100
Read data 2	0
Alu second input	0
Alu Output	100
Zero Flag	0
Data Memory OutPut	0
MemToReg Mux output	100
- Memory and Wires:** A section at the bottom with checkboxes for "Address in Hexadecimal", "Memory", and "Wires".

Program 3: recursive Euclidian HCF calculator

```

1  #include<iostream>
2  using namespace std;
3  int euclid(int a, int b){
4      if ( a == b )
5          return a;
6      else if ( a < b )
7          return euclid(a, b-a);
8      else
9          return euclid(a-b,b);
10 }
11 void main(){
12     int m = euclid(87,29);
13     cout<<m;
14     cin>>m;
15 }

```

The screenshot displays a MIPS simulator interface with three main panels:

- Assembly Code Panel (Left):** Shows the assembly translation of the C program. Key instructions include:
 - `main:` `addi $a0, $0, 87`, `addi $a1, $0, 29`, `jal euclid`
 - `euclid:` `beq $a0, $a1, notrec`, `addi $v0, $a0, 0`, `jr $ra`
 - `notrec:` `add $sp, $sp, -4`, `sw $ra, 0($sp)`, `sub $t0, $a1, $a0`, `sll $t0, $t0, $0`, `beq $t0, $0, notfirstcase`, `jal firstcase`
 - `notfirstcase:` `sub $a1, $a1, $a0`, `jal euclid`, `j exit`
 - `firstcase:` `sub $a0, $a0, $a1`, `jal euclid`
 - `exit:` `lw $ra, 0($sp)`, `addi $sp, $sp, 4`, `jr $ra`
- Registers Panel (Middle):** A table showing the state of MIPS registers.

Register	Value
\$zero	0
\$at	1
\$v0	2
\$v1	3
\$a0	4
\$a1	5
\$a2	6
\$a3	7
\$t0	8
\$t1	9
\$t2	10
\$t3	11
\$t4	12
\$t5	13
\$t6	14
\$t7	15
\$s0	16
\$s1	17
\$s2	18
\$s3	19
\$s4	20
\$s5	21
\$s6	22
\$s7	23
\$t8	24
\$t9	25
\$k0	26
\$k1	27
\$gp	28
\$sp	29
\$fp	30
\$re	31
- Wires Panel (Right):** A table showing the values of various MIPS components.

Wire Name	Value
PC output	88
PC + 4 adder output	76
Instruction Memory Output	65011720
OpCode Field of instruction	0
Rs Field of instruction	31
Rt Field of instruction	0
Rd Field of instruction	0
RegDestination Mux output	0
Offset Field of instruction	8
Sign extender Output	8
Shift 2 Output	32
Target address adder	108
Function Code Of instruction	65011720
Read data 1	88
Read data 2	0
Alu second input	0
Alu Output	88
Zero Flag	0
Data Memory OutPut	0
MemToReg Mux output	88

A small dialog box in the bottom right corner shows a table with two rows:

Value	Address
88	16777196
64	16777192

Below the table is a checkbox labeled "Address in Hex" which is currently unchecked.

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

```

1  #include<iostream>
2  using namespace std;
3  int rangeSum(int a, int b){
4      int sum = 0;
5      for (int i = a; i <1+b; i++)
6          sum = sum + i;
7      return sum;
8  }
9  void main(){
10     int m = rangeSum(5,10);
11     cout<<m;
12     cin>>m;
13 }

```

The screenshot displays a MIPS simulator interface with three main panels:

- PC pointer:** Shows the assembly code being executed.


```

j main
sum:
    addi $t0, $0, 0
    addi $t1, $a1, 1
loop:
    beq $a0, $t1, exit
    add $t0, $t0, $a0
    addi $a0, $a0, 1
    j loop
exit:
    add $v0, $t0, $0
    jr $ra
main:
    addi $a0, $0, 5
    addi $a1, $0, 10
    jal sum
      
```
- Registers:** A table showing the current values of MIPS registers.

Register	Value	Comment
\$zero	0	
\$at	1	
\$v0	2	45
\$v1	3	0
\$a0	4	11
\$a1	5	10
\$a2	6	0
\$a3	7	0
\$t0	8	45
\$t1	9	11
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$t8	16	0
\$t9	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$t0	26	0
\$t1	27	0
\$t2	28	0
\$t3	29	16777200
\$t4	30	0
\$t5	31	48
- Wire Name / Value:** A table showing the values of various internal wires.

Wire Name	Value
PC output	48
PC +4 adder output	36
Instruction Memory Output	65011720
OpCode Field of Instruction	0
Rs Field of instruction	31
Rt Field of instruction	0
Rd Field of instruction	0
RegDestination Mux output	0
Offset Field of Instruction	8
Sign extender Output	8
Shift 2 Output	32
Target address adder	68
Function Code Of Instruction	65011720
Read data 1	48
Read data 2	0
Alu second input	0
Alu Output	48
Zero Flag	0
Data Memory OutPut	0
MemToReg Mux output	48

Program 5: determines minimum element in an array

```

1  #include<iostream>
2  using namespace std;
3  int arrayMin(int* a, int n){
4      int min = *a;
5      for(int *i = a + 1; i < a+n; i++)
6          if (*i < min)
7              min = *i;
8      return min;
9  }
10 }
11 void main(){
12     int arr[] = {-5, 6, -9, 10, -20};
13     int m = arrayMin(arr,5);
14     cout<<m;
15     cin>>m;
16 }

```

The screenshot displays a MIPS simulator interface with the following components:

- PC pointer:** A text area showing assembly code for a program that finds the minimum element in an array. The code includes labels for `sum`, `loop`, `exit`, and `main`.
- Registers:** A table listing MIPS registers (\$zero to \$ra) and their current values. For example, \$zero is 0, \$a0 is 4, and \$ra is 68.
- Wires:** A table showing various internal processor signals and their values, such as PC output (68), Instruction Memory Output (65011720), and ALU Output (68).
- Memory and Wires:** A section at the bottom with tabs for "Memory" and "Wires". The "Memory" tab is active, showing a table of memory addresses and their corresponding values. The values are: 5 at address 1000, 6 at 1004, -9 at 1008, 10 at 1012, and -20 at 1016.

Program 6: determines sum of array elements

```

1  #include<iostream>
2  using namespace std;
3  int arraySum(int* a, int n){
4      int sum = 0;
5      for(int *i = a; i < a+n; i++){
6          sum = sum + *i;
7      }
8      return sum;
9  }
10 }
11 void main(){
12     int arr[] = {1,2,3,4,5};
13     int m = arraySum(arr,5);
14     cout<<m;
15     cin>>m;
16 }

```

The screenshot displays a MIPS simulator interface with the following components:

- PC pointer:** A tab at the top left.
- Registers:** A table listing registers \$Zero through \$s31 with their current values.
- Memory and Wires:** A section at the bottom left with checkboxes for "Address in Hexadecimal", "Memory", and "Wires".
- Wire Name and Value:** A table on the right showing various wire outputs and their decimal values.
- Value and Address:** A small table at the bottom right showing memory addresses and their corresponding values.

Wire Name	Value
PC output	56
PC +4 adder output	44
Instruction Memory Output	65011720
OpCode Field of Instruction	0
Rs Field of Instruction	31
Rt Field of Instruction	0
Rd Field of Instruction	0
RegDestination Mux output	0
Offset Field of Instruction	8
Sign extender Output	8
Shift 2 Output	32
Target address adder	76
Function Code Of Instruction	65011720
Read data 1	56
Read data 2	0
Alu second input	0
Alu Output	56
Zero Flag	0
Data Memory OutPut	0
MemToReg Mux output	56

Value	Address
1	1000
2	1004
3	1008
4	1012
5	1016

10. Appendix B

This appendix has the logic diagram needed for the control unit

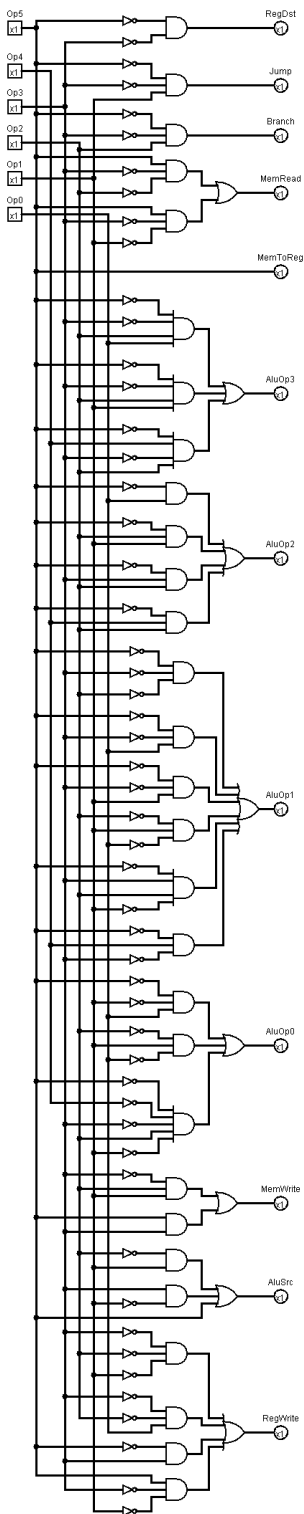


Figure 13 the first 12 bits of the logic circuit of the control unit

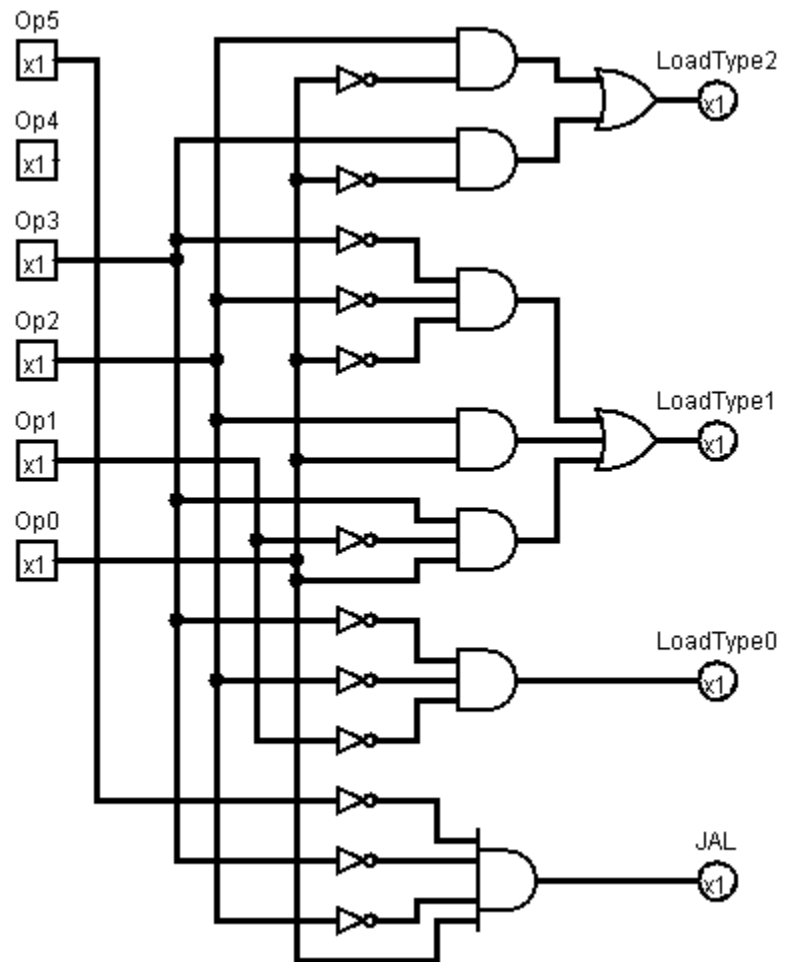


Figure 12 the last 4 bits of the logic circuit of the control unit