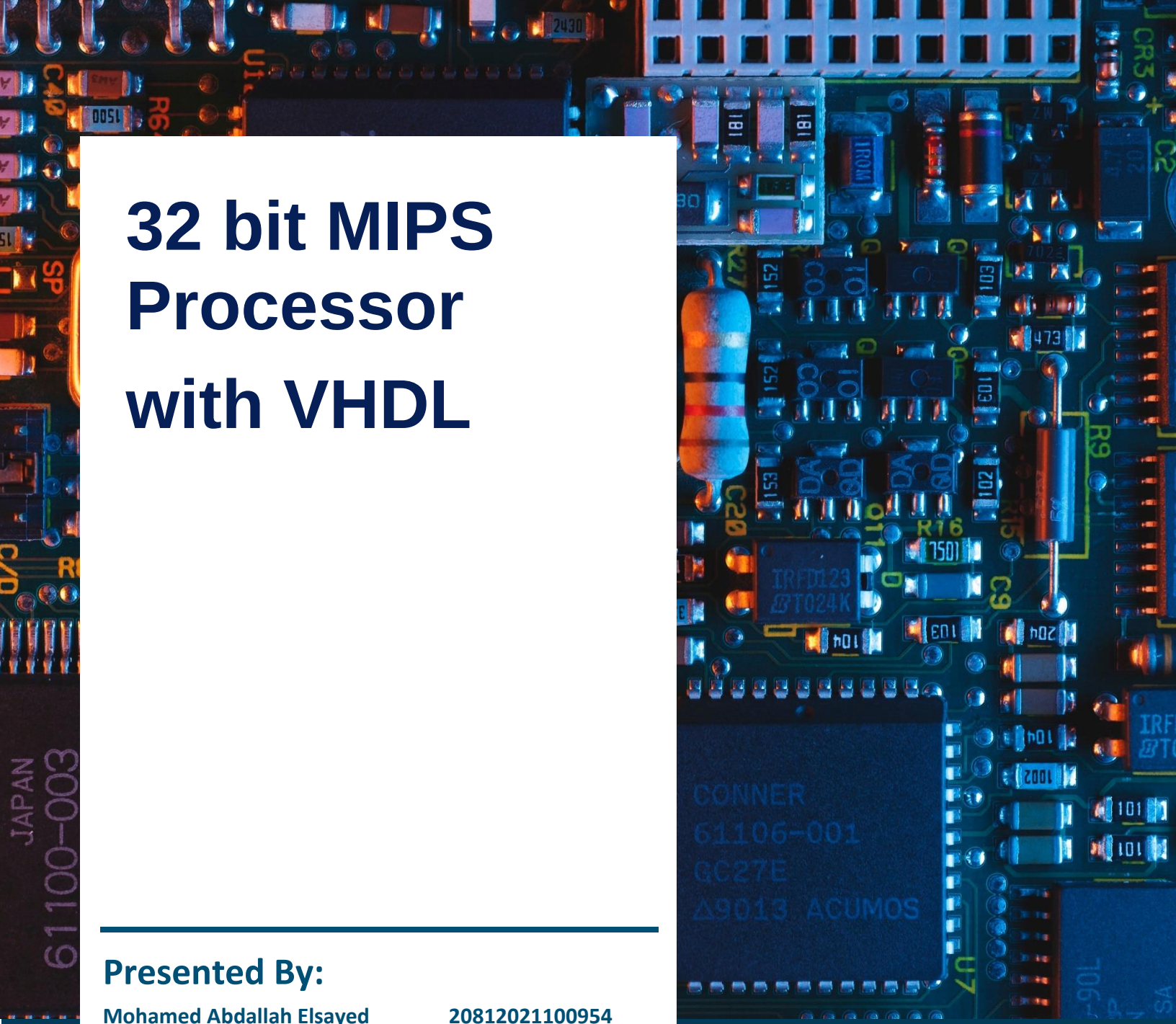


32 bit MIPS Processor with VHDL

Presented By:

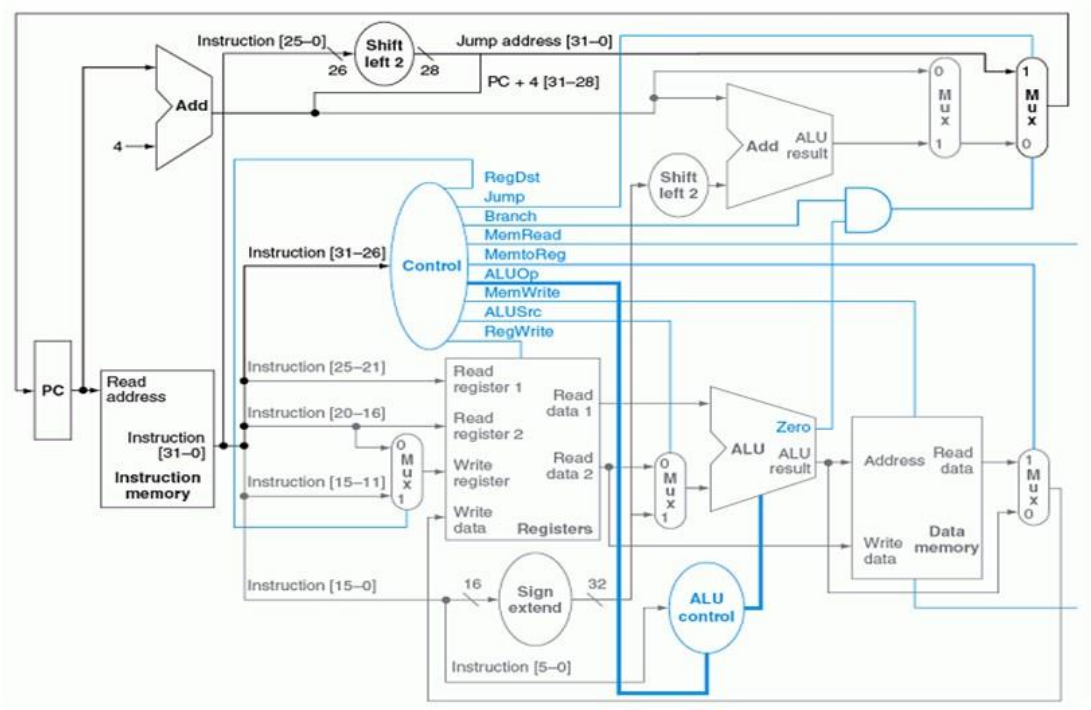
Mohamed Abdallah Elsayed	20812021100954
Ahmed Mohamed Fayad	20812021100138
Reham Hamdy Mohamed Ibrahim	20812021201109
Mahmoud Elsayed Hussein	20812021101038
Shahd Elsayed Ahmed Ahmed	20812021200543
Abdelrahman Ibrahim Mahmoud	20812021100007
Nancy Ayman Nabil Mohamed	20812021200506

Table Of Content

1. The Assignment Aim

1.1 Introduction

This project aims to implement a 32 bit microprocessor based on the MIPS architecture to get a grasp of how processors work and how to design real-world hardware using VHDL.



1.2 Scope

For this project we implemented a single cycle processor.

1.3 Methodology

Every one of us was assigned a certain block of the processor, test it thoroughly and deliver a functioning block that is ready to be integrated with the whole processor.

2. Implementation

2.1 MUX

2.1.1 Objective

The multiplexer takes two 32bit inputs and selects one of them based on the select signal.

2.1.2 Inputs and Outputs

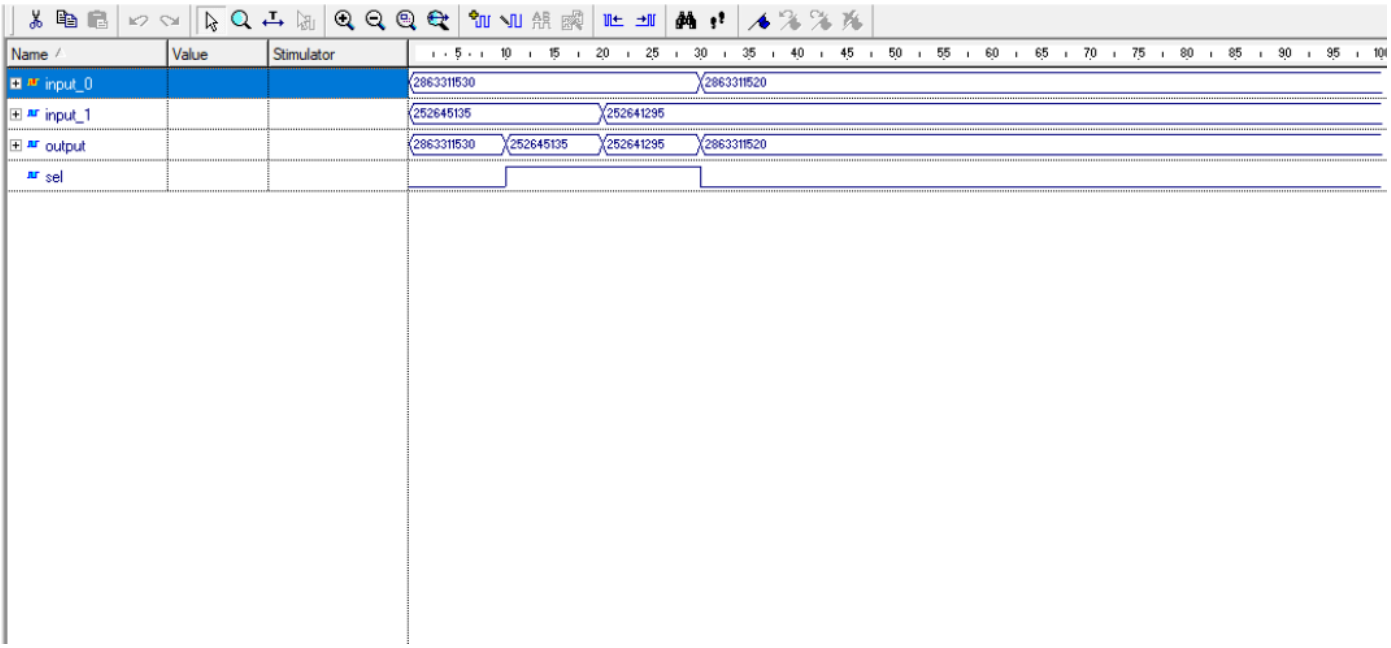
input_0 : 32-bit input

input_1 : 32-bit input

sel : 1-bit input selector

output : 32-bit output

2.1.3 Test



2.2 Left Shifter

2.2.1 Objective

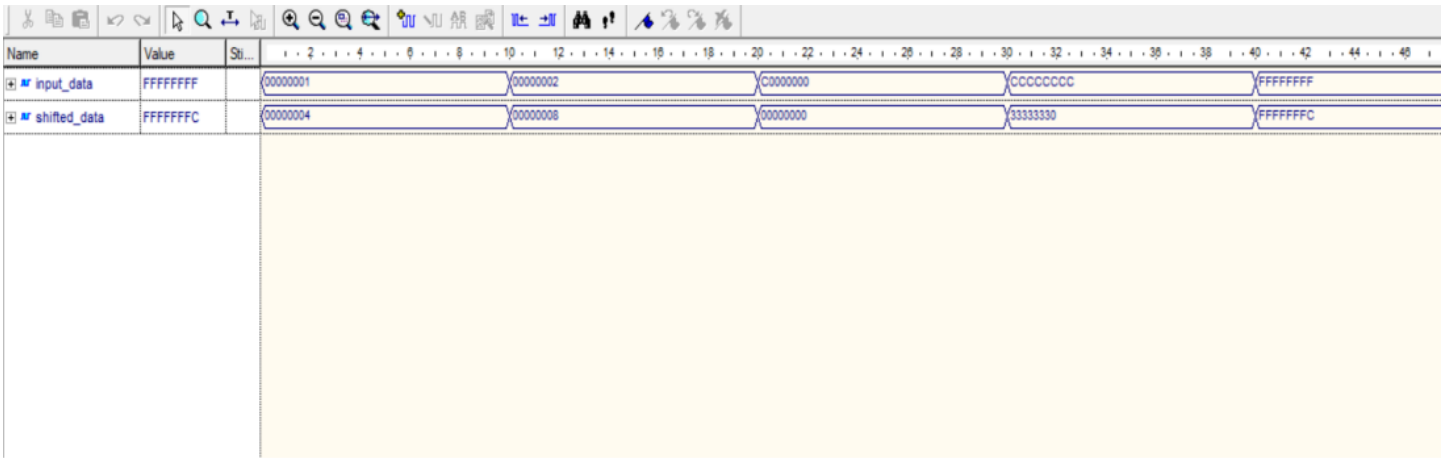
The "shift left logical by 2" operation provides a simple and efficient means of multiplying a number by 4 in MIPS architecture, as it effectively shifts the binary representation of the number two positions to the left.

2.2.2 Inputs and Outputs

input_data : 32-bit input

output_data: 32-bit output

2.2.3 Test



2.3 Sign Extender

2.3.1 Objective

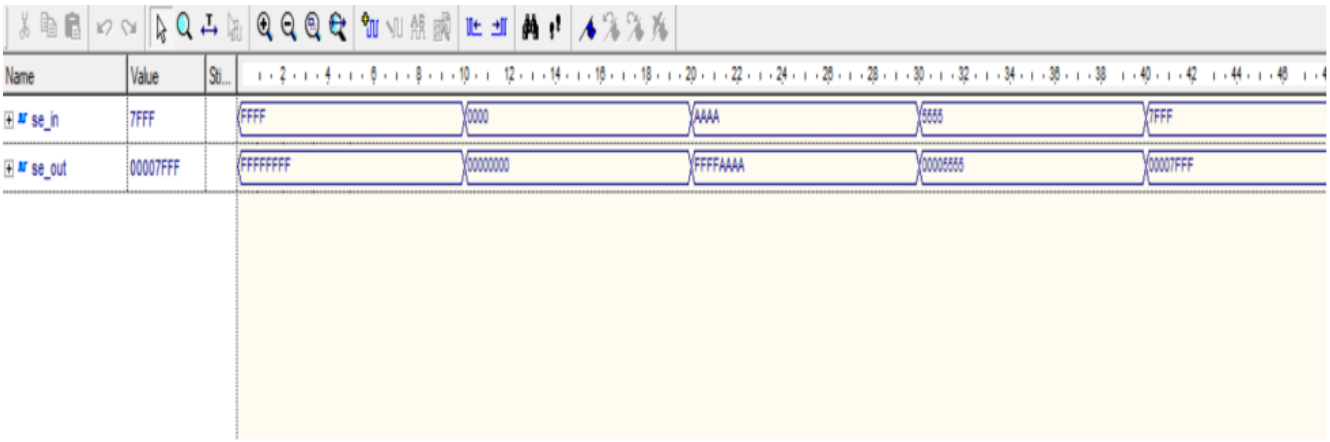
This operation copies the sign bit of a shorter binary number into additional bits, effectively maintaining the number's positive or negative interpretation. It's particularly important when handling signed data in instructions or arithmetic operations, ensuring accurate representation and computation of signed values.

2.3.2 Inputs and Outputs

se_in : 16-bit input

se_out: 32-bit output

2.3.3 Test



2.4 Data Memory

2.4.1 Objective

Reading and writing data to and from the memory of the MIPS.

2.4.2 Inputs and Outputs

clk : clock signal input

addr : destination address

wr_data : 32-bit input data to be written

memory_read : input signal to read from memory

memory_write: input signal to write memory

data_out : 32-bit data output read from memory

2.4.3 Writing Data To Memory

In Case of Store Word(SW) operation , the ALU First calculate the memory address where the fetched data coming from the register file(rt) will be stored.

Example:

IF the Address is equal(x"10010000") ,the data coming from the register file say it equal (x"11112222) will be stored in this address.

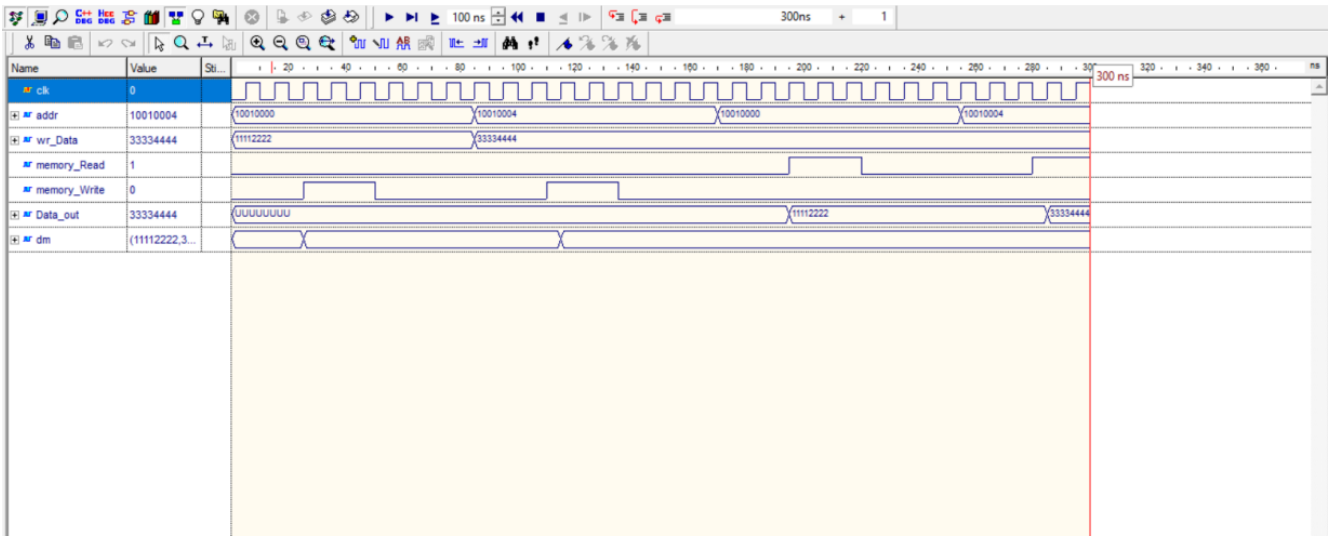
2.4.4 Reading Data From Memory

In Case of Load Word (LW) operation, the ALU first calculate the memory address where the data is located then transfer this data to register file where it will be stored.

Examples:

The data located at this address (x"10010000") is transferred from data memory to register file where it will be stored.

2.4.5 Test



2.5 Instruction Memory

2.5.1 Objective

this memory contains 16 instruction, every instruction has an address in the memory. the memory checks if the address that came from PC in range (`0x00400000` the first address in memory to `0x0040003C` the last address in memory) the memory output will be the instruction of this address otherwise the output will be Zero.

2.5.2 Instruction Array

```
-  x"02119001"   -- 0x00400000  :  and $s2, $s0, $s1
-  x"0253a002"   -- 0x00400004  :  or   $s4, $s2, $s3
-  x"02b6b804"   -- 0x00400008  :  nor  $s7, $s5, $s6
-  x"01095008"   -- 0x0040000C  :  xor  $t2, $t0, $t1
-  x"8e320030"   -- 0x00400010  :  lw   $s2, 48($s1)
-  x"AEB40012"   -- 0x00400014  :  sw  $s4, 18($s5)
-  x"8EAC0012"   -- 0x00400018  :  lw   $t4, 18($s5)
-  x"12760019"   -- 0x0040001C  :  beq  $s3, $s6, 25
-  x"02929801"   -- 0x00400020  :  and $s3, $s4, $s2
-  x"12720004"   -- 0x00400024  :  beq  $s3, $s2, 4
-  x"adaf0018"   -- 0x00400028  :  sw    $t7, 24($t5)
-  x"0810000c"   -- 0x0040002C  :  J (02b6b804)
-  x"016c6810"   -- 0x00400030  :  add  $t5, $t3, $t4
-  x"02515020"   -- 0x00400034  :  sub $t2, $s2, $s1
-  x"01a85810"   -- 0x00400038  :  add  $t3, $t5, $t0
-  x"018E7820"   -- 0x0040003C  :  sub $t6, $t4, $t6
```

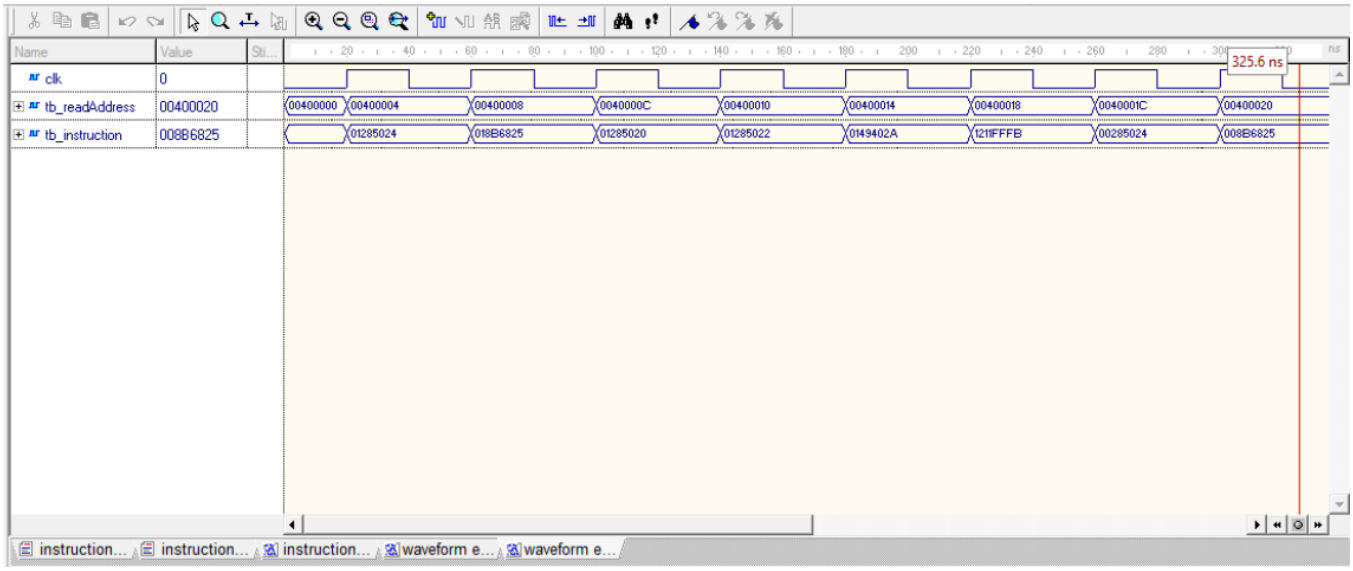
2.5.3 Inputs and Outputs

Inst_addr : 32-bit input

Clk : clock signal input

instruction : 32-bit output instruction

2.5.4 Test



2.6 Program Counter

2.6.1 Objective

Fetches instructions and stores the address where the program begins.

2.6.2 Inputs and Outputs

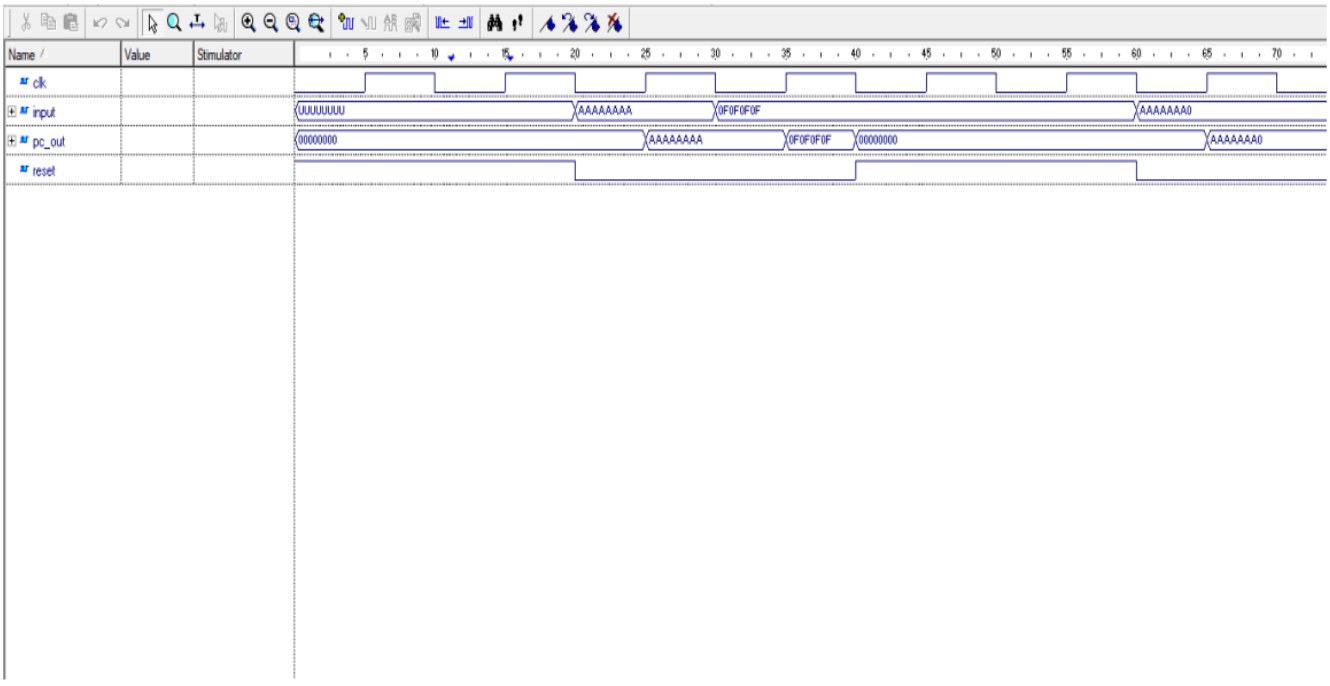
clk : clock signal input

reset : reset signal input

intput : 32bit input signal

output : 32bit output signal

2.6.3 Test



2.7 Register File

2.7.1 Objective

Provides a set of registers to storing data and intermediate results in program execution.

2.7.2 Functionality

- Any value provided on 5-line Read register 1 port results in the content of the corresponding register being provided on the 32-line Read data 1 port.
- Any value provided on 5-line Read register 2 port results in the content of the corresponding register being provided on the 32-line Read data 2 port.
- In writing operation, values that appear on 32-bit Write data port are written into the register with the number specified on the 5-line Write register port

2.7.3 Inputs and Outputs

clk : clock signal input

RD1 : 5-bit input for selecting the 1st register to read from

RD2 : 5-bit input for selecting the 2nd register to read from

WData_add : 5-bit input for selecting the address of the register to write on

WData : 32-bit data input to be written into the selected register

RegWrit : input control signal for writing

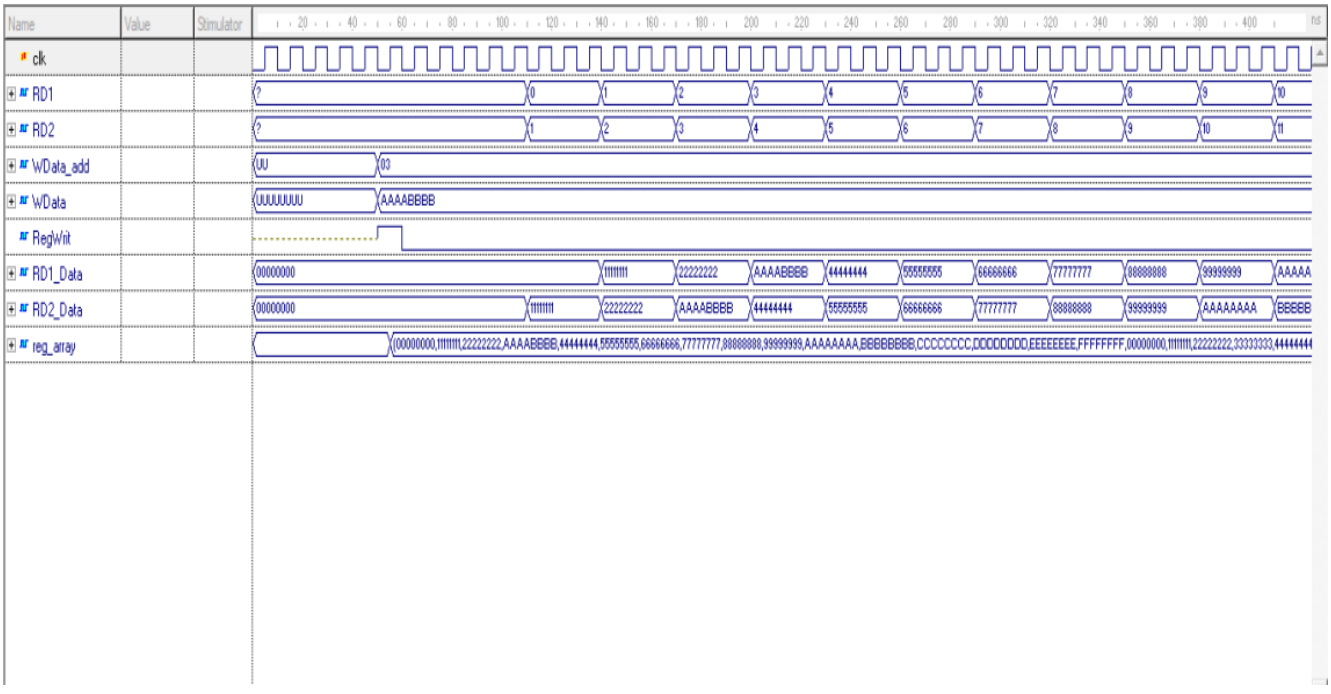
RD1_Data : 32-bit output data from the first selected register

RD2_Data : 32-bit output data from the second selected register

2.7.4 Register Array

```
-  $zero     0=>00000          -  $s0     16=>10000
-  $at       1=>00001          -  $s1     17=>10001
-  $v0       2=>00010          -  $s2     18=>10010
-  $v1       3=>00011          -  $s3     19=>10011
-  $a0       4=>00100          -  $s4     20=>10100
-  $a1       5=>00101          -  $s5     21=>10101
-  $a2       6=>00110          -  $s6     22=>10110
-  $a3       7=>00111          -  $s7     23=>10111
-  $t0       8=>01000          -  $t8     24=>11000
-  $t1       9=>01001          -  $t9     25=>11001
-  $t2       10=>01010         -  $k0     26=>11010
-  $t3       11=>01011         -  $k1     27=>11011
-  $t4       12=>01100         -  $gp     28=>11100
-  $t5       13=>01101         -  $sp     29=>11101
-  $t6       14=>01110         -  $fp     30=>11110
-  $t7       15=>01111         -  $ra     31=>11111
```

2.7.5 Test



2.8 ALU

2.8.1 Objective

Provides logical and mathematical operations for the MIPS.

2.8.2 Operations

- 000 : AND
- 001 : OR
- 010 : NOR
- 011 : XOR
- 100 : ADD
- 101 : SUB

2.8.3 Inputs and Outputs

a : First 32bit input

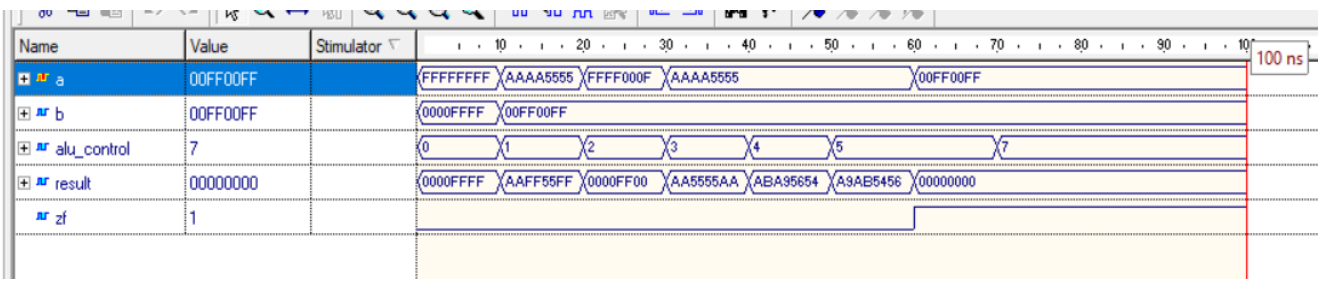
b : Second 32bit input

alu_control: control signal

zf : zero flag

result : 32-bit result

2.8.4 Test



2.9 Control Unit

2.9.1 Objective

Control the operations in the MIPS based on the instruction.

2.9.2 Inputs And Outputs

opcode : 6 bit code that determine type

funct : 6 bit code that determine functions

RegDst : select the destination register (**rt** or **rd**)

Jump : enables loading the jump target address into the **PC**

Branch : enable loading the branch target address into the **PC**

MemRead : enables a memory read for load instructions

MemWrite : enables a memory write for store instructions

MemtoReg : determines where the value to be written comes from **ALU** or **Memory**

RegWrite : enable writing data on **destinationReg** at Register Files

ALUSrc : select either a register operand or a constant operand

opSel : select **ALU** operation

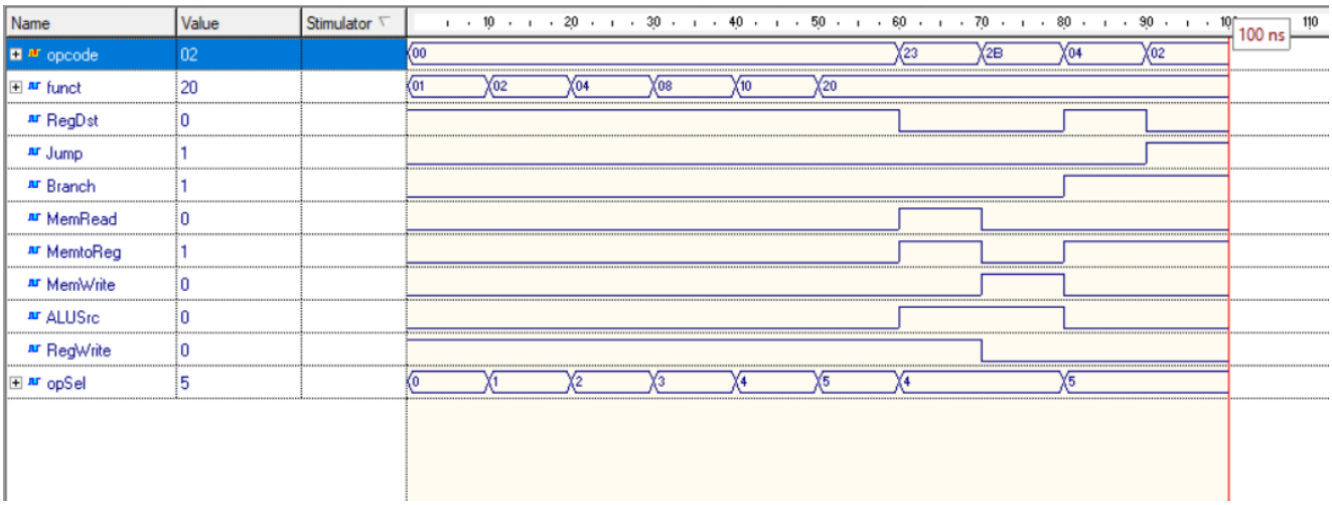
2.9.3 Instruction Types

- 000000 R-Type
- 100011 Lw-Type
- 101011 Sw-Type
- 000100 Beq-Type
- 000010 J-Type

2.9.4 Functions

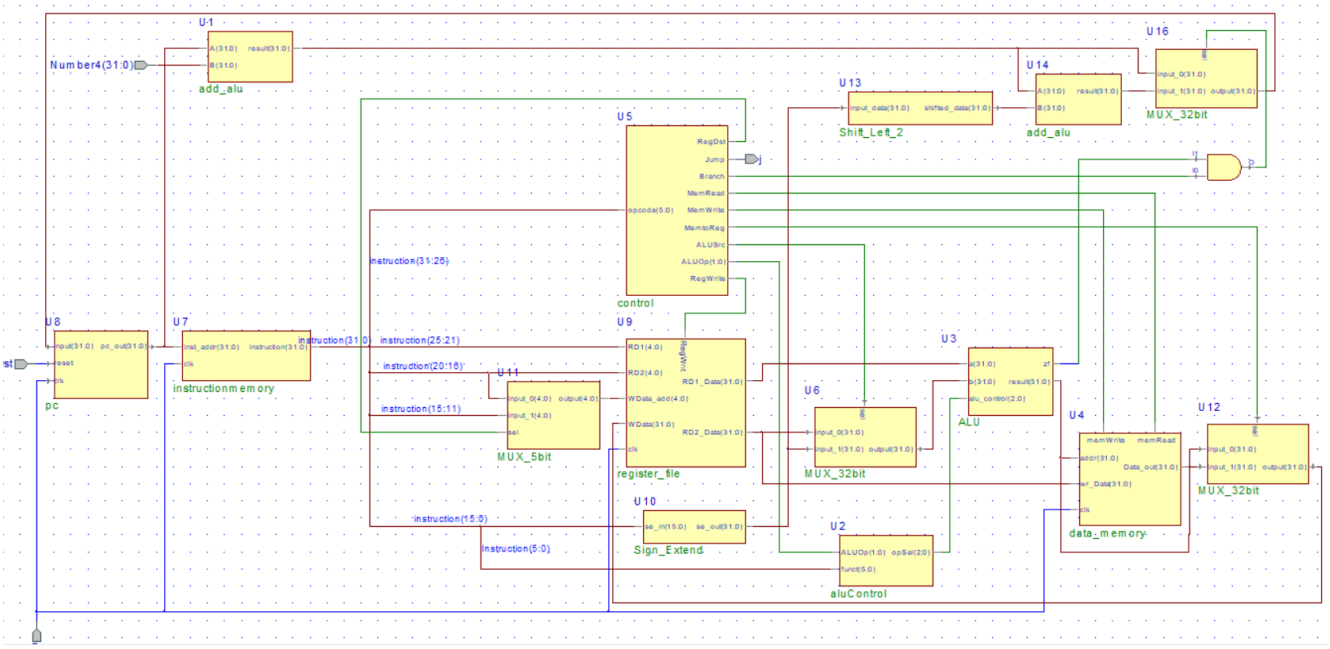
- 000001 : AND
- 000010 : OR
- 000100 : NOR
- 001000 : XOR
- 010000 : ADD
- 100000 : SUB

2.9.5 Test

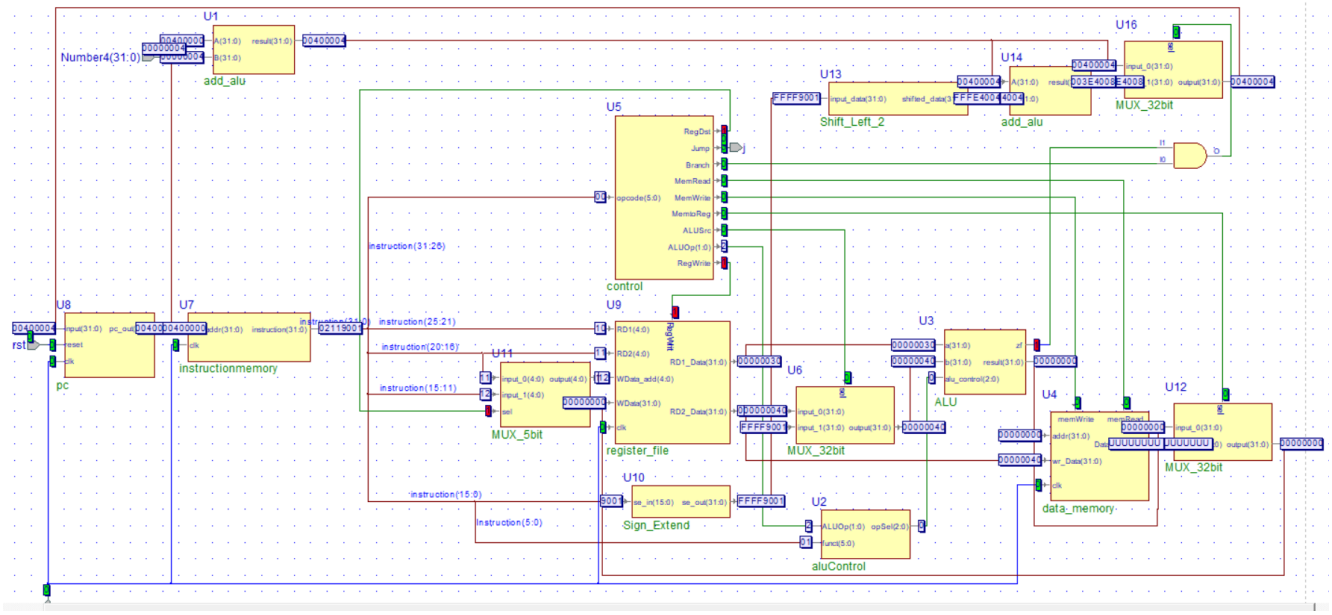


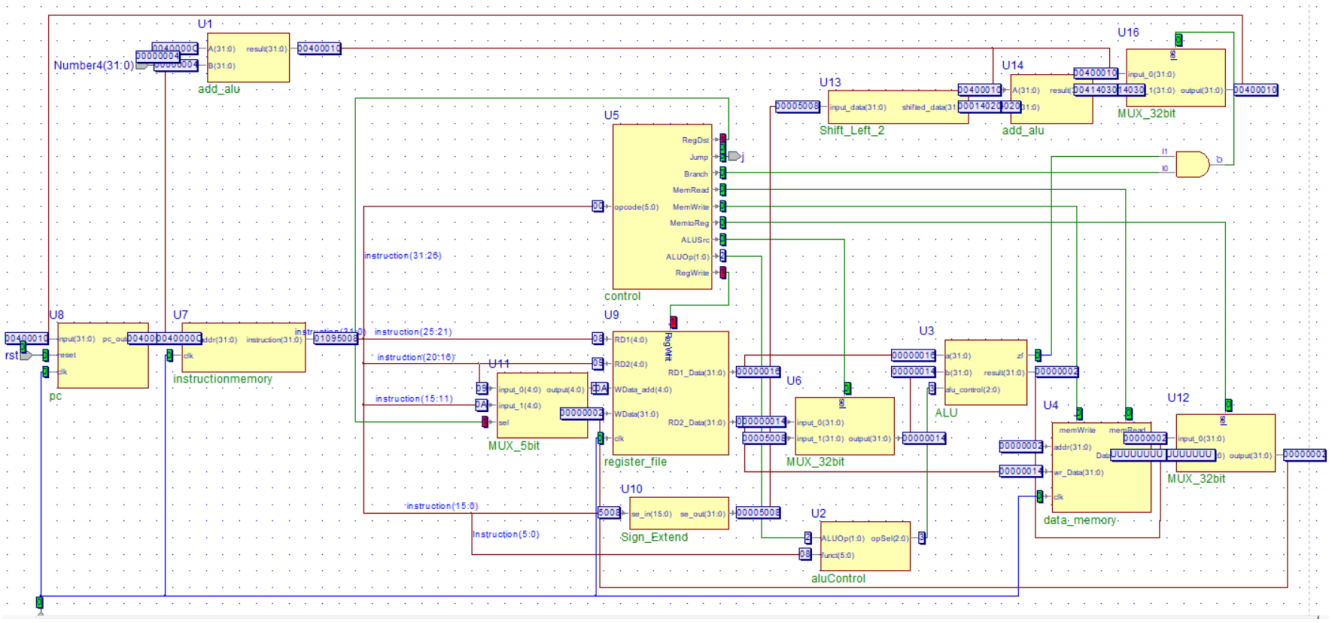
3. Final Result

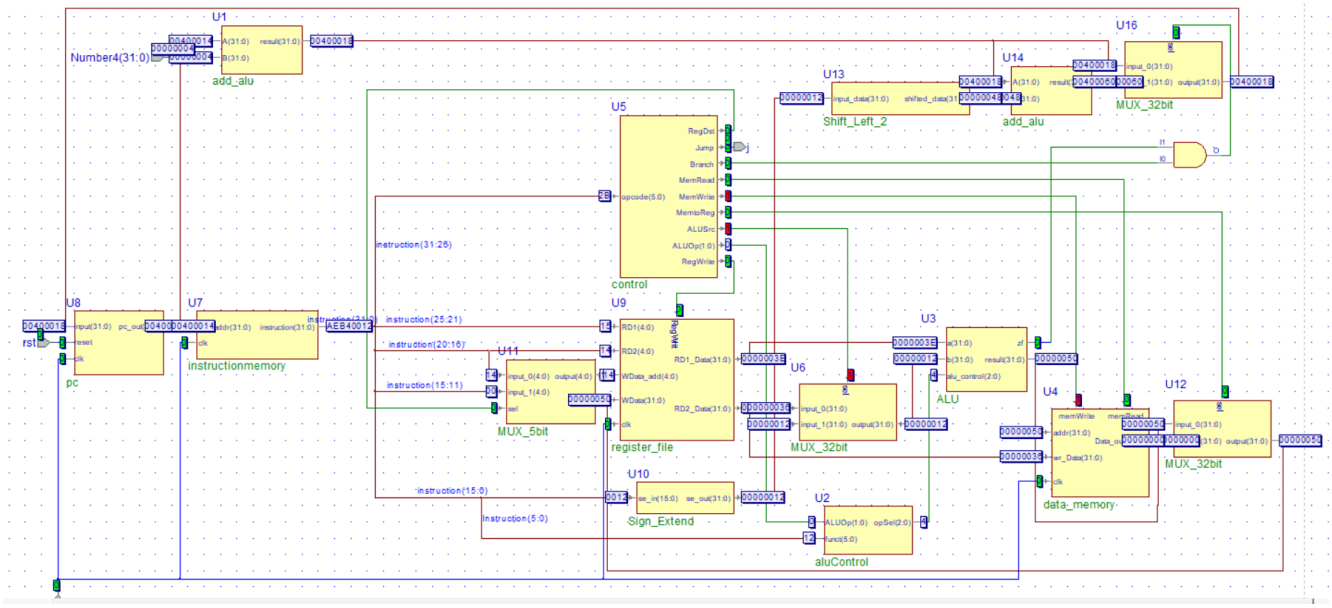
3.1 Block Diagram

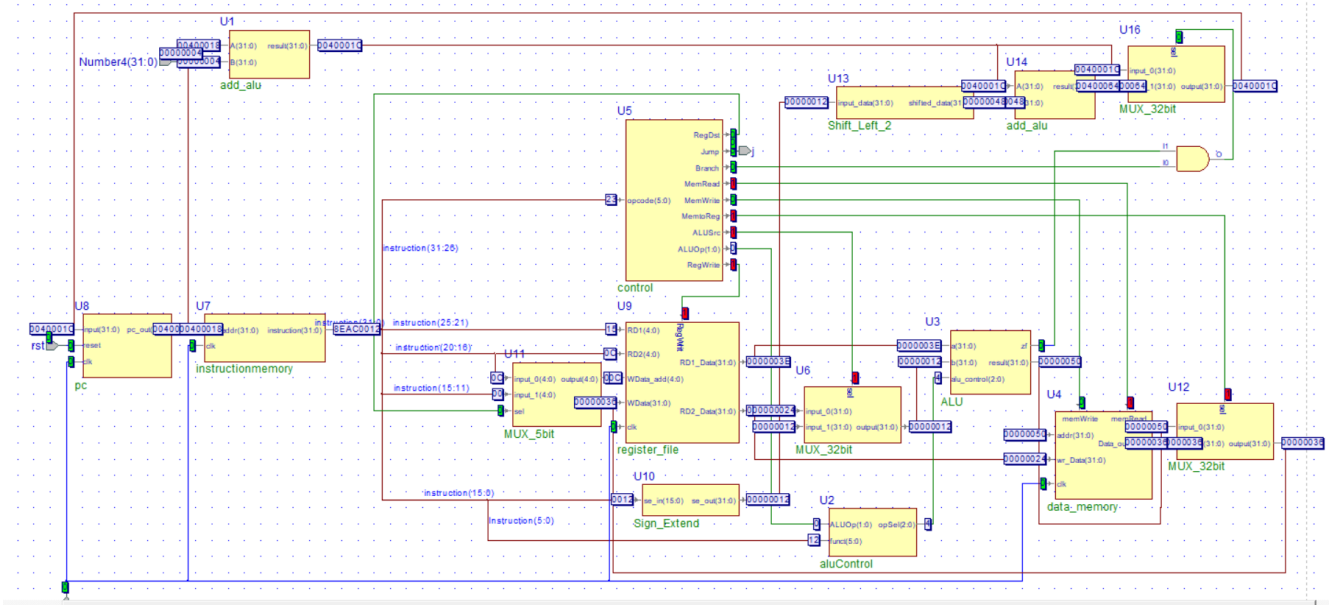


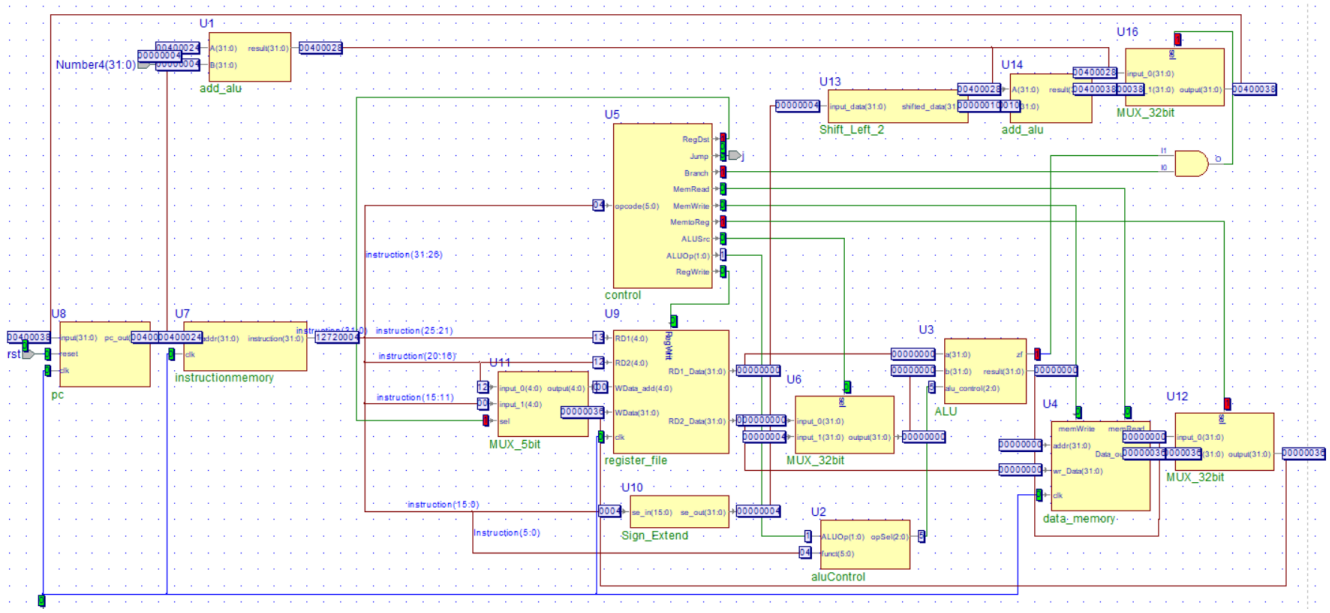
3.2 Test

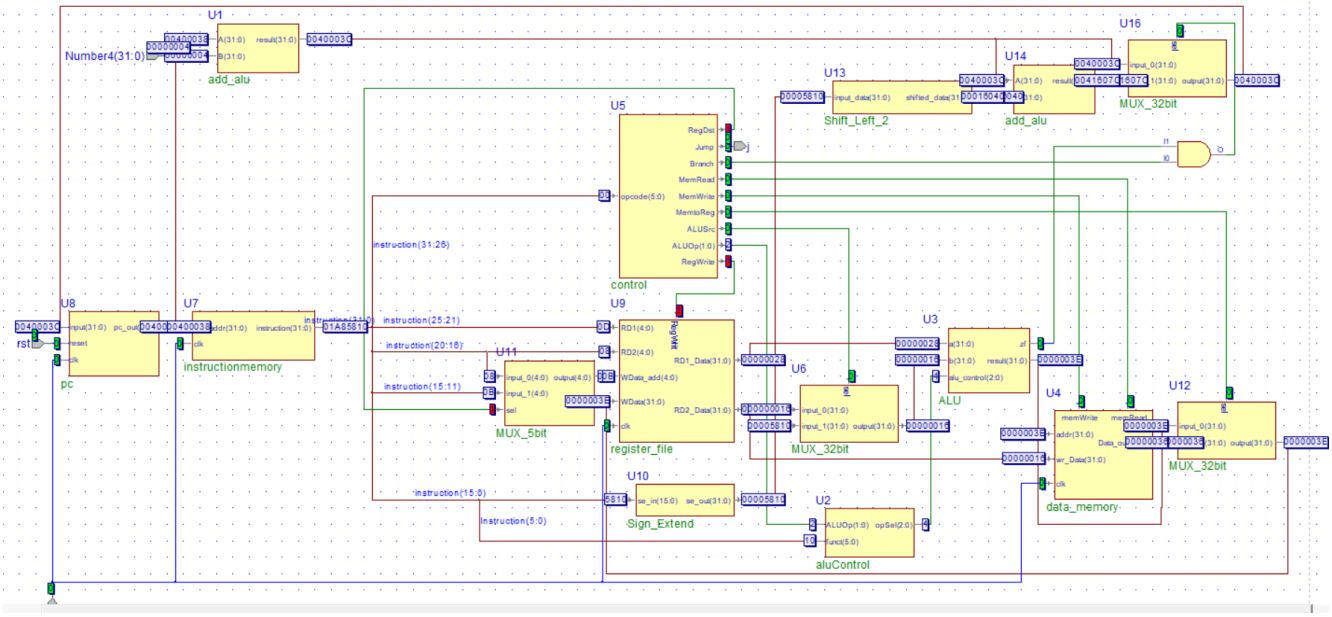


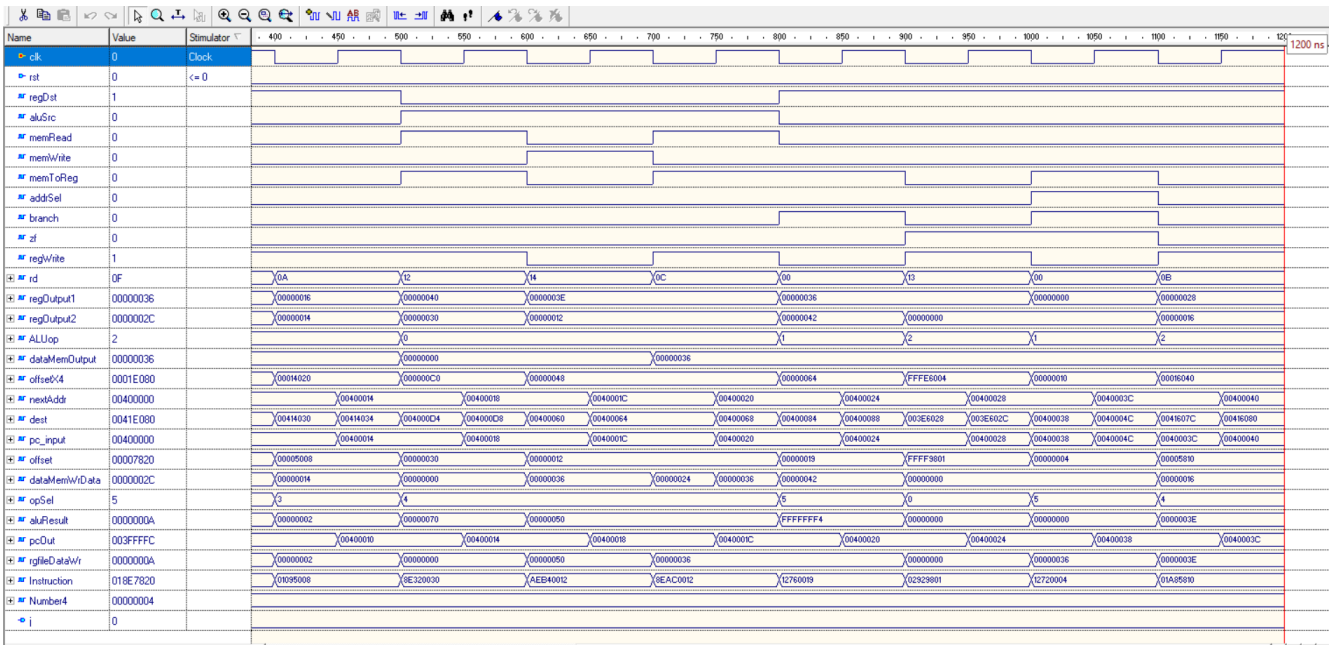

U1
Number4(31:0)
add_alu
U16
MUX_32bit
U14
add_alu
U13
Shift_Left_2
U5
control
RegDst
Jump
Branch
MemRead
MemWrite
MemtoReg
ALUSrc
ALUOp(1:0)
RegWrite
opcode(5:0)
instruction(31:26)
U8
pc
rst
U7
instructionmemory
instruction(25:21)
instruction(20:16)
instruction(15:11)
instruction(15:0)
Instruction(5:0)
U11
MUX_5bit
U9
register_file
RD1_Data(31:0)
RD2_Data(31:0)
WData_add(4:0)
WData(31:0)
U6
MUX_32bit
U3
ALU
U4
data_memory
U12
MUX_32bit
U10
Sign_Extend
U2
aluControl


U1
Number4(31:0)
add_alu
U16
MUX_32bit
U14
add_alu
U13
Shift_Left_2
U5
control
RegDst
Jump
Branch
MemRead
MemWrite
MemtoReg
ALUSrc
ALUOp(1:0)
RegWrite
opcode(5:0)
instruction(31:26)
U8
pc
rst
U7
instructionmemory
instruction(25:21)
instruction(20:16)
instruction(15:11)
instruction(15:0)
Instruction(5:0)
U11
MUX_5bit
U9
register_file
RD1_Data(31:0)
RD2_Data(31:0)
WData_add(4:0)
WData(31:0)
U6
MUX_32bit
U3
ALU
U4
data_memory
U12
MUX_32bit
U10
Sign_Extend
U2
aluControl


U1
Number4(31:0)
add_alu
U16
MUX_32bit
U14
add_alu
U13
Shift_Left_2
U5
control
RegDst
Jump
Branch
MemRead
MemWrite
MemtoReg
ALUSrc
ALUOp(1:0)
RegWrite
opcode(5:0)
instruction(31:26)
U8
pc
rst
U7
instructionmemory
instruction(25:21)
instruction(20:16)
instruction(15:11)
instruction(15:0)
Instruction(5:0)
U11
MUX_5bit
U9
register_file
RD1_Data(31:0)
RD2_Data(31:0)
WData_add(4:0)
WData(31:0)
U6
MUX_32bit
U3
ALU
U4
data_memory
U12
MUX_32bit
U10
Sign_Extend
U2
aluControl


U1
Number4(31:0)
add_alu
U16
MUX_32bit
U14
add_alu
U13
Shift_Left_2
U5
control
instruction(31:26)
U8
pc
U7
instructionmemory
instruction(25:21)
instruction(20:16)
instruction(15:11)
U11
MUX_5bit
U9
register_file
U3
ALU
U6
MUX_32bit
U4
data_memory
U12
MUX_32bit
instruction(15:0)
Instruction(5:0)
U10
Sign_Extend
U2
aluControl


U1
Number4(31:0)
add_alu
U16
MUX_32bit
U14
add_alu
U13
Shift_Left_2
U5
control
instruction(31:26)
U8
pc
U7
instructionmemory
instruction(25:21)
instruction(20:16)
instruction(15:11)
U11
MUX_5bit
U9
register_file
U3
ALU
U6
MUX_32bit
U4
data_memory
U12
MUX_32bit
instruction(15:0)
Instruction(5:0)
U10
Sign_Extend
U2
aluControl


Name
Value
Stimulator
clk
rst
regDst
aluSrc
memRead
memWrite
memToReg
addrSel
branch
zf
regWrite
rd
regOutput1
regOutput2
ALUop
dataMemOutput
offset0X4
nextAddr
dest
pc_input
offset
dataMemWriData
opSel
aluResult
pcOut
rgfileDataWr
Instruction
Number4
j
1200 ns

4. References

4.1 Dr.Howida Hardware Design course 2nd CSE

4.2 Dr. Anirban Sengupta

4.3 twalsh123

4.4 ChatGPT