NYU-6463 Processor Design

Members:

1. Yash Gandhi

2. Venus Sherathiya

3. Rugved Sawant

4. Bhagyashri Soman

5. Harshada Patankar

Table of Content:

1 PROJECT DESCRIPTION

We implemented a 32-bit processor in VHDL, called NYU-6463 Processor, which can execute programs. The instruction set used, the block diagram and description of its components is as follows:

1.1 Instruction Set:

The processor was designed using the following instruction set.

Mnemonic	Description	Type	Opcode (Hex)	Func (Hex)	Operation
ADD	Add Registers	R	00	10	Rd = Rs + Rt
ADDI	Add Immediate	I	01		Rt = Rs + SignExt(Imm)
SUB	Subtract Registers	R	00	11	Rd = Rs - Rt
SUBI	Subtract Immediate	I	02		Rt = Rs - SignExt(Imm)
AND	Register bitwise And	R	00	12	Rd = Rs & Rt
ANDI	And Immediate	I	03		Rt = Rs & SignExt(Imm)
OR	Register bitwise OR	R	00	13	Rd = Rs | Rt
NOR	Register bitwise NOR	R	00	14	Rd = !(Rs | Rt)
ORI	OR Immediate	I	04		Rt = Rs | SignExt(Imm)
SHL	Shift Left by immediate bits	I	05		Rt = Rs << Imm
SHR	Shift Right by immediate bits	I	06		Rt = Rs >> Imm
LW	Load Word	I	07		Rt ← Mem[SignExt(Imm) + Rs]
SW	Store Word	I	08		Mem[SignExt(Imm) + Rs] ← Rt
BLT	Branch if less than	I	09		If (Rs < Rt) then PC = PC + 4 + Imm
BEQ	Branch if equal	I	0A		If (Rs = = Rt) then PC = PC + 4 + Imm
BNE	Branch if not equal	I	0B		If (Rs != Rt) then PC = PC + 4 + Imm
JMP	Jump	J	0C		PC = {(PC + 4)[31:28], address, 00}
HAL	Halt	J	3F		

Every instruction specified above is a 32 bit instruction. All the operations are performed assuming 2's complement notation for operands and the result. These instructions are divided into three parts:

a. R-Type Instructions: These handle all arithmetic instructions. The fields used for this instruction are:

Opcode (6-bits)	Rs (5-bits)	Rt (5-bits)	Rd (5-bits)	Shamt (5-bits)	Funct (6-bits)

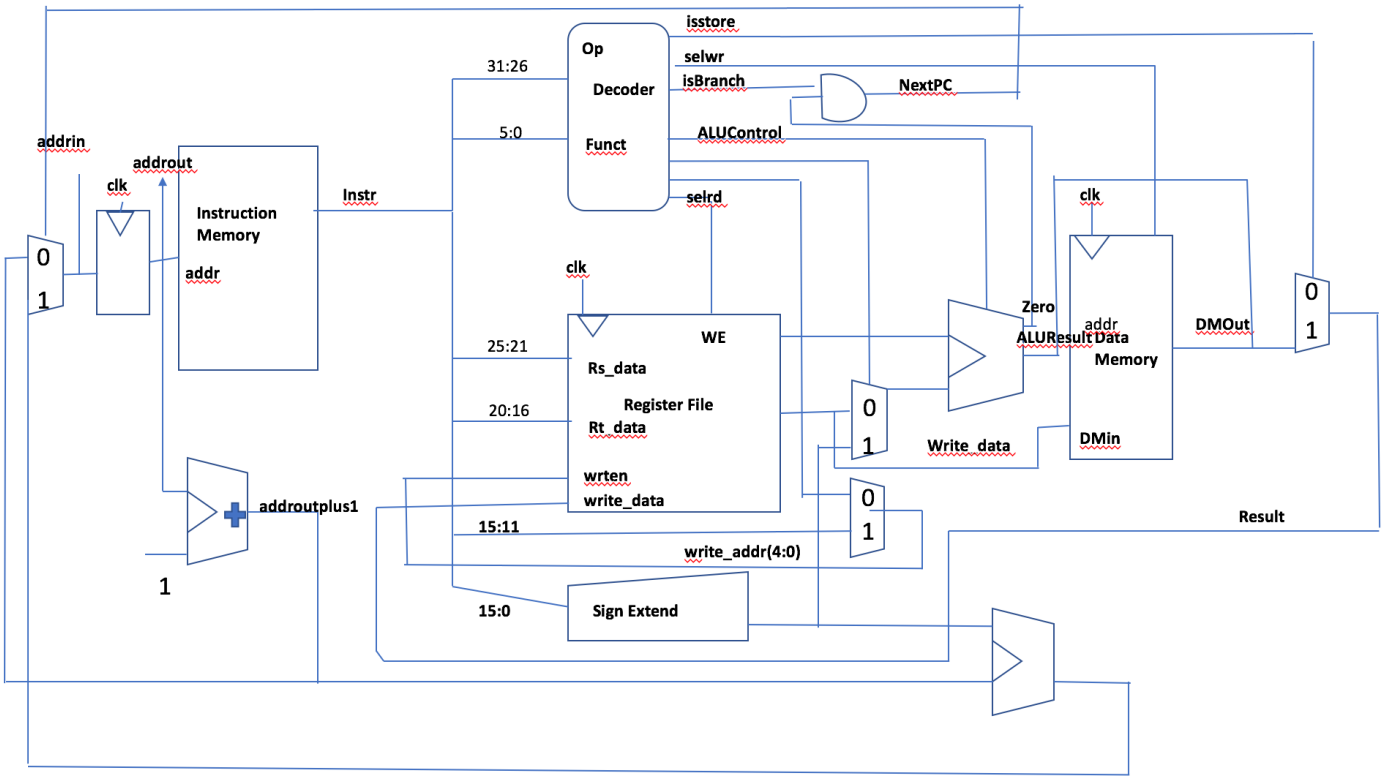
b. I-Type Instructions: These handle all the immediate value operations, load and store instructions. The fields used for these instructions are:

Opcode (6-bits)	Rs (5-bits)	Rt (5-bits)	Address/Immediate (16-bits)

c. J-Type Instructions: These handle all jump and branch instructions. The fields used are:

Opcode (6-bits)	Address (26-bits)

1.2 Processor Design (Block Diagram):



1.3 Processor Components:

a. ALU: This block performs basic arithmetic and logic operations like addition, subtraction, comparison etc. It uses the control signals from the Decoder, and also uses the immediate value of the I-type instructions. It comptes data that is to be written to any of the register files, or even the Program Counter itself.

b. Decoder: This block basically takes in some or all parts of the 32-bit instruction, depending on the opcode and funct bits – computes proper signals to be utilized for other modules in the processor. These are basically the control signals like isLess, isEqual, or whether to read or write tp/from memory etc.

c. Data Memory: This block is basically used to store the data used by the processor. It is accessed by the load and store functions. Based on which function is to be implemented, i.e if load is used, the processor reads the required data from a memory location(address) specified and in case of store instructions, it writes to the memory at the location specified in the instruction.

d. Instruction Memory: This block is primarily used to load the instruction based on the address(pointed by the program counter). It contains all the instructions that are to be executed for the program to run on our processor.
e. Register File: This module contains the 32 32-bit registers. The functionality of this block can handle reading to a register and writing to another register in the same clock cycle. It receives the control signals – reg read and reg write from the decoder unit. It is addressed using 5 bits.
f. MUX: This unit handles the functionality of sign extension, and is used to calculate the next address of the program counter. It does this based on the type of instruction that is being accessed. It also determines the data that is to be written to the register file i.e if the data is ALU Result or the output of Data Memory.
g. Program Counter: The only function of this block it to point to the next instruction. It does this using a 32-bit register which basically contains the address of the next instruction that is to be executed by the processor.

1.4 Processor Operation:

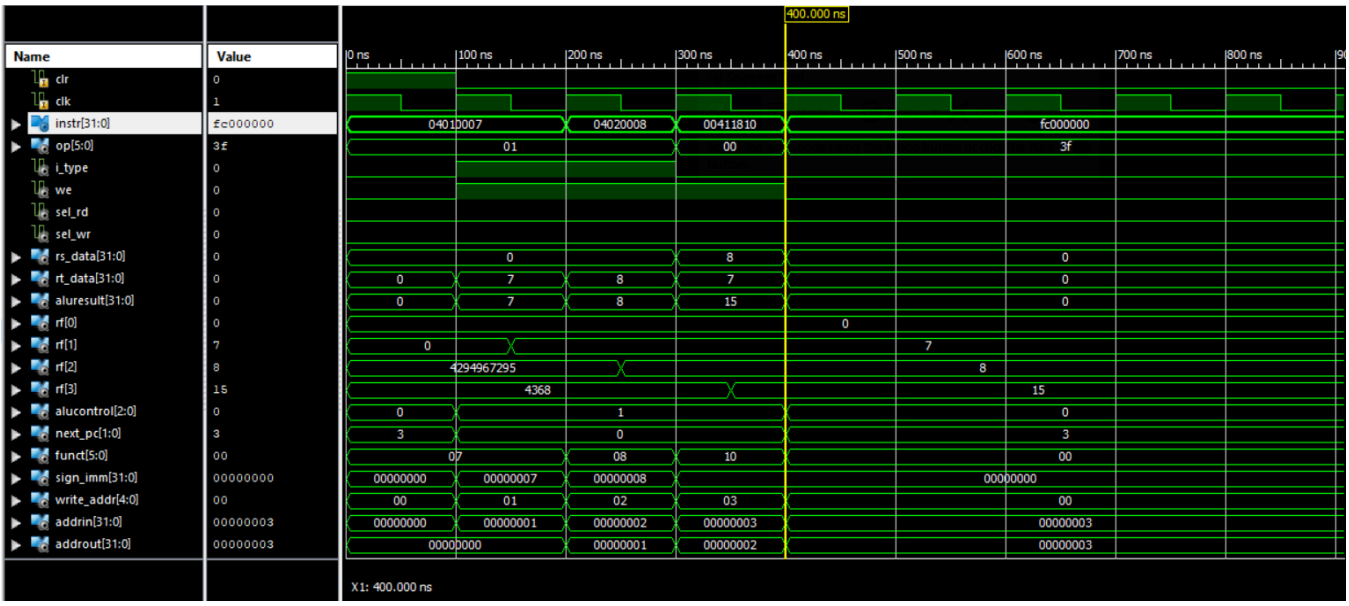
Our processor performs the task of instruction fetch, instruction decode, execution, memory access and write-back all in one cycle. Initially, the Program Counter value is used as an address to index the Instruction Memory which gives it 32-bit instruction to be executed next. This instruction is then divided into different fields as described above in section 1.1. The Decoder block takes in the opcode as input and based on that determines the type of instruction to be executed (R-Type, I-Type, J-Type). This type of input then determines which control signals are to be asserted, and what function the ALU is supposed to perform, thus, decoding the instruction. The Rs, Rt, Rd bits are used to address the register file. The register file reads in the requested addresses and outputs the data values contained in the registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g branch). If the instruction decoded is arithmetic, the ALU result must be written to the register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

2 Simulations:

2.1 Functional Simulation:

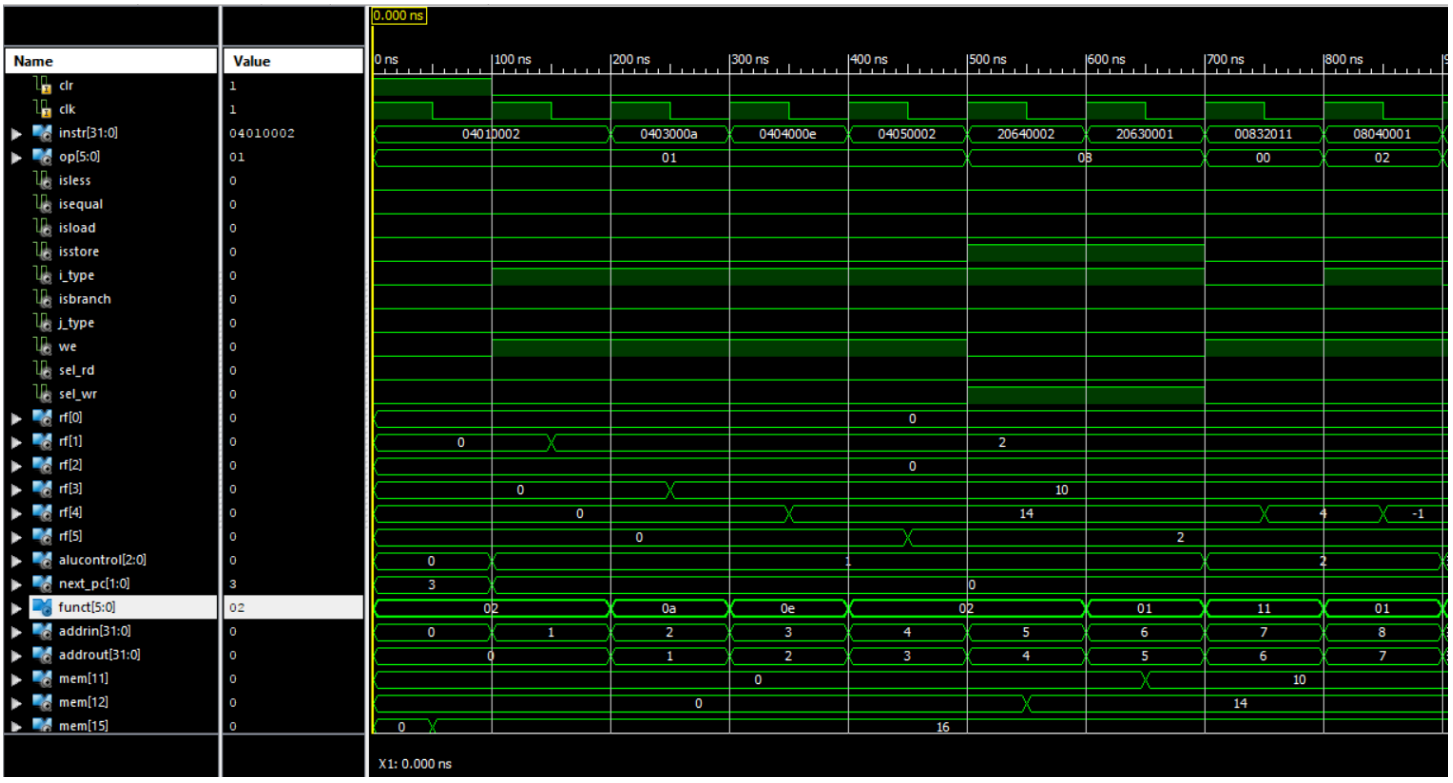
2.1.1 Sample Code 1:

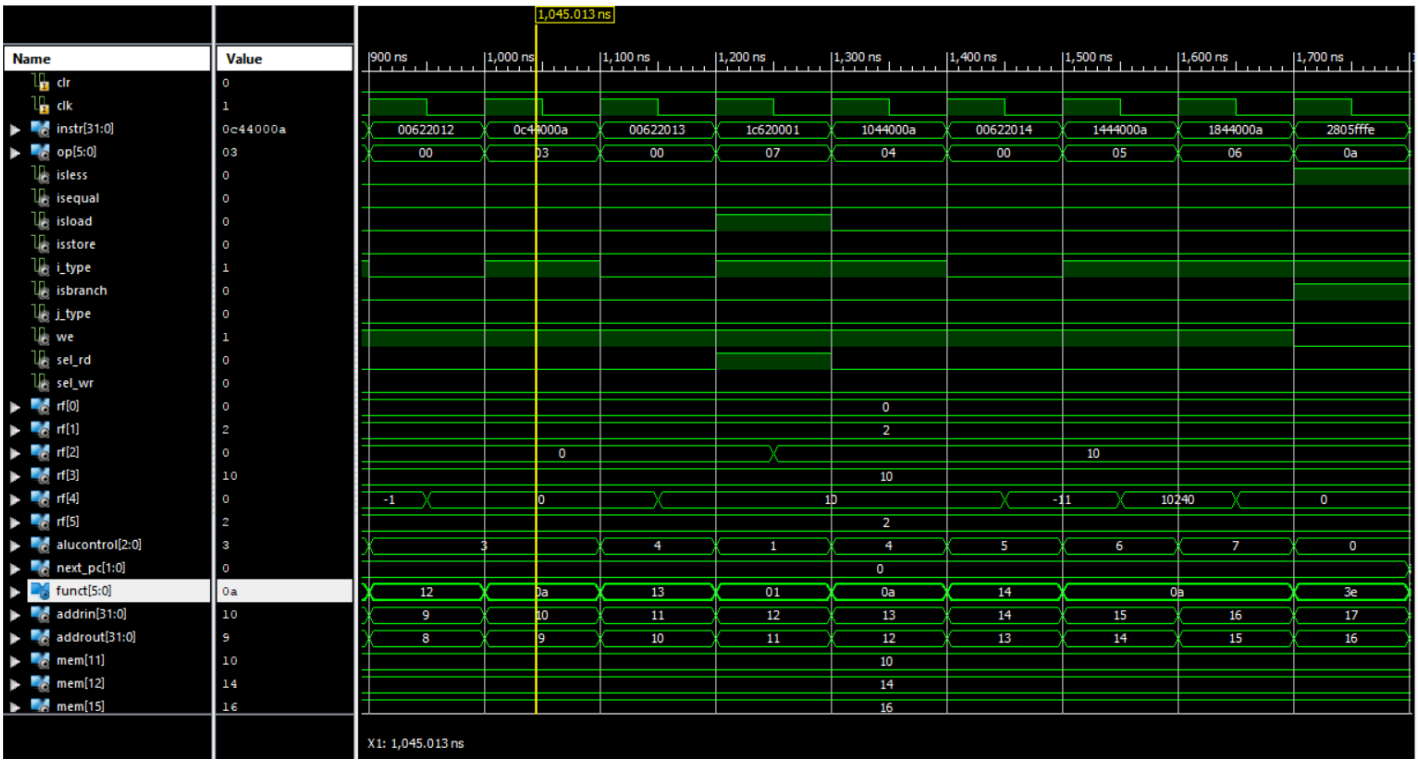
00000100000000010000000000000111 --ADDI R1, R0, 7 // R1 = 7
00000100000000010000000000001000 --ADDI R2, R0, 8 // R2 = 8
00000000010000010001100000010000 --ADD R3, R1, R2 // R3 = R1 + R2 =15
11111100000000000000000000000000 --HAL // HALT

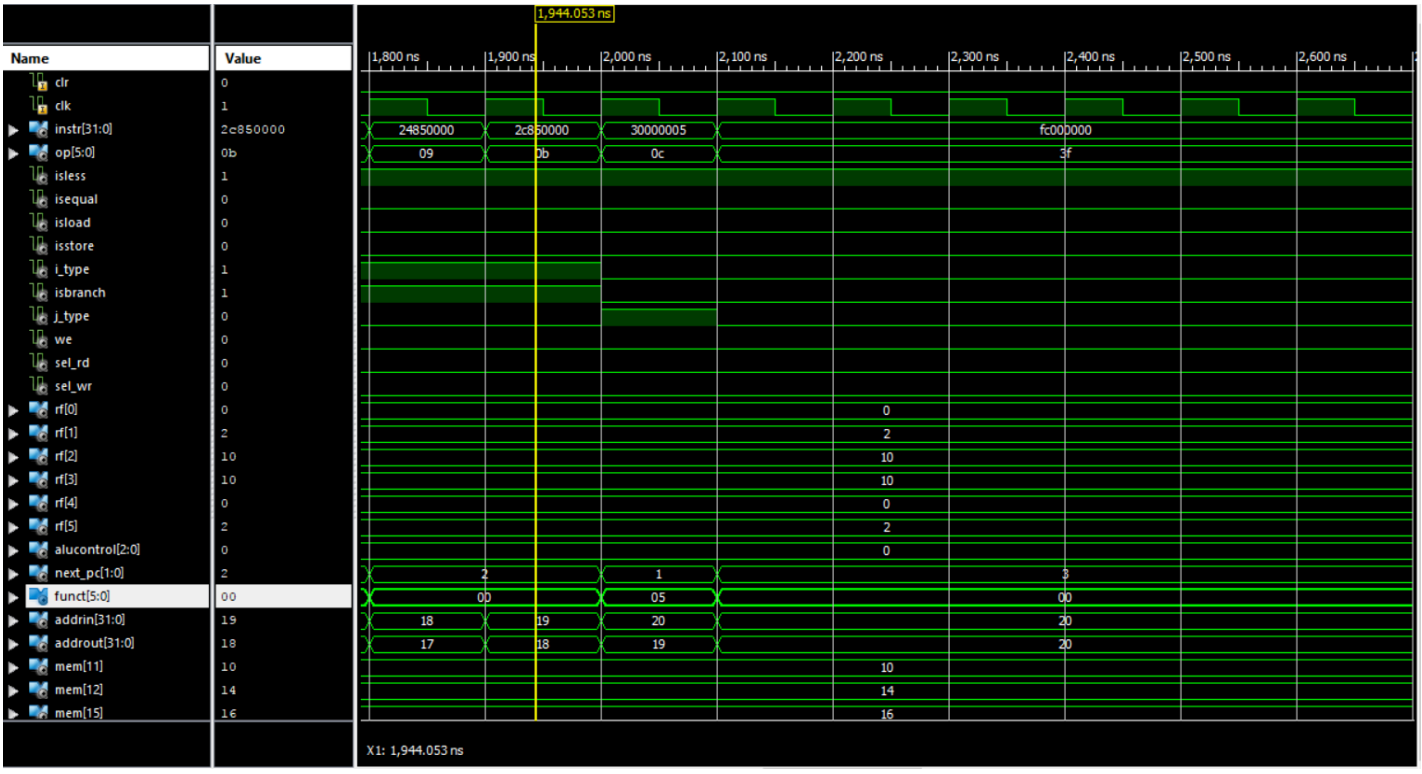


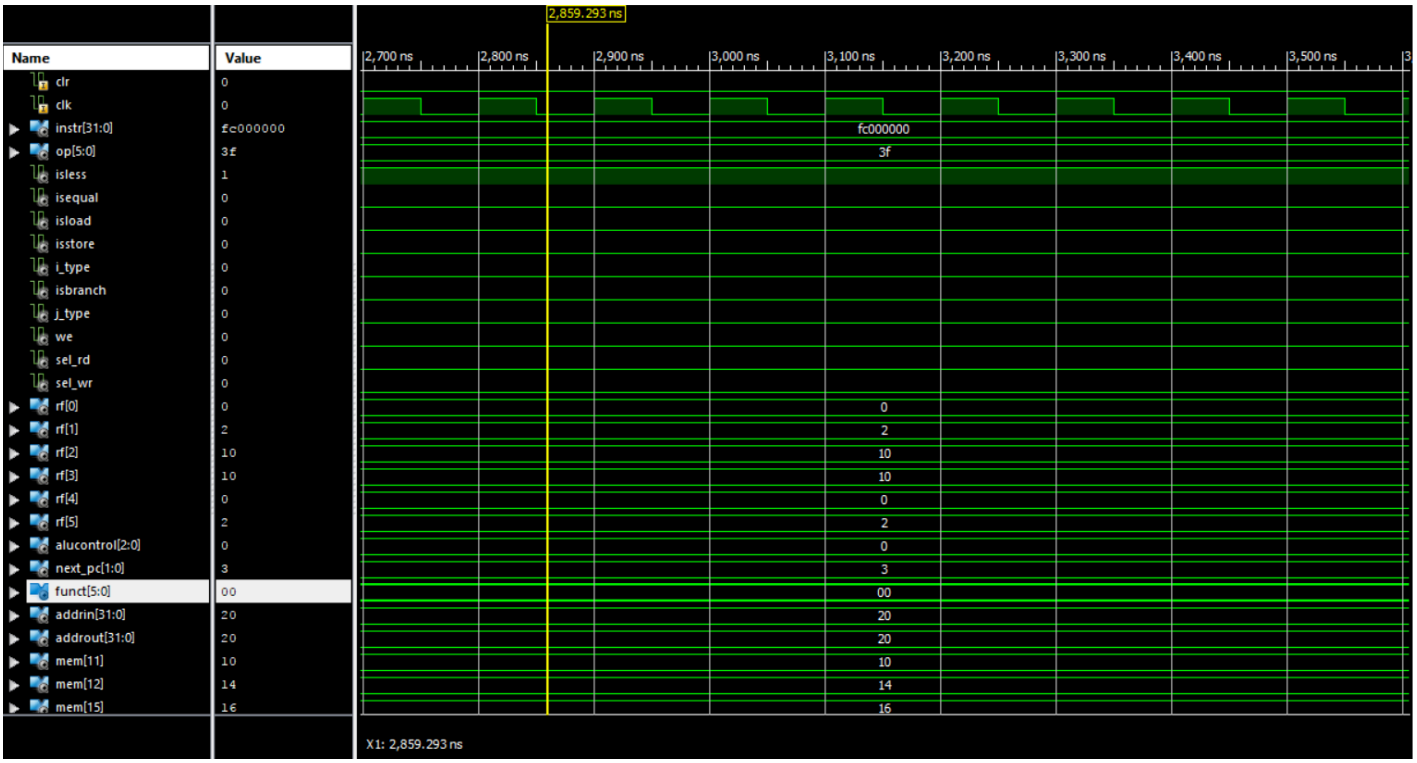
2.1.2 Sample Code 2:

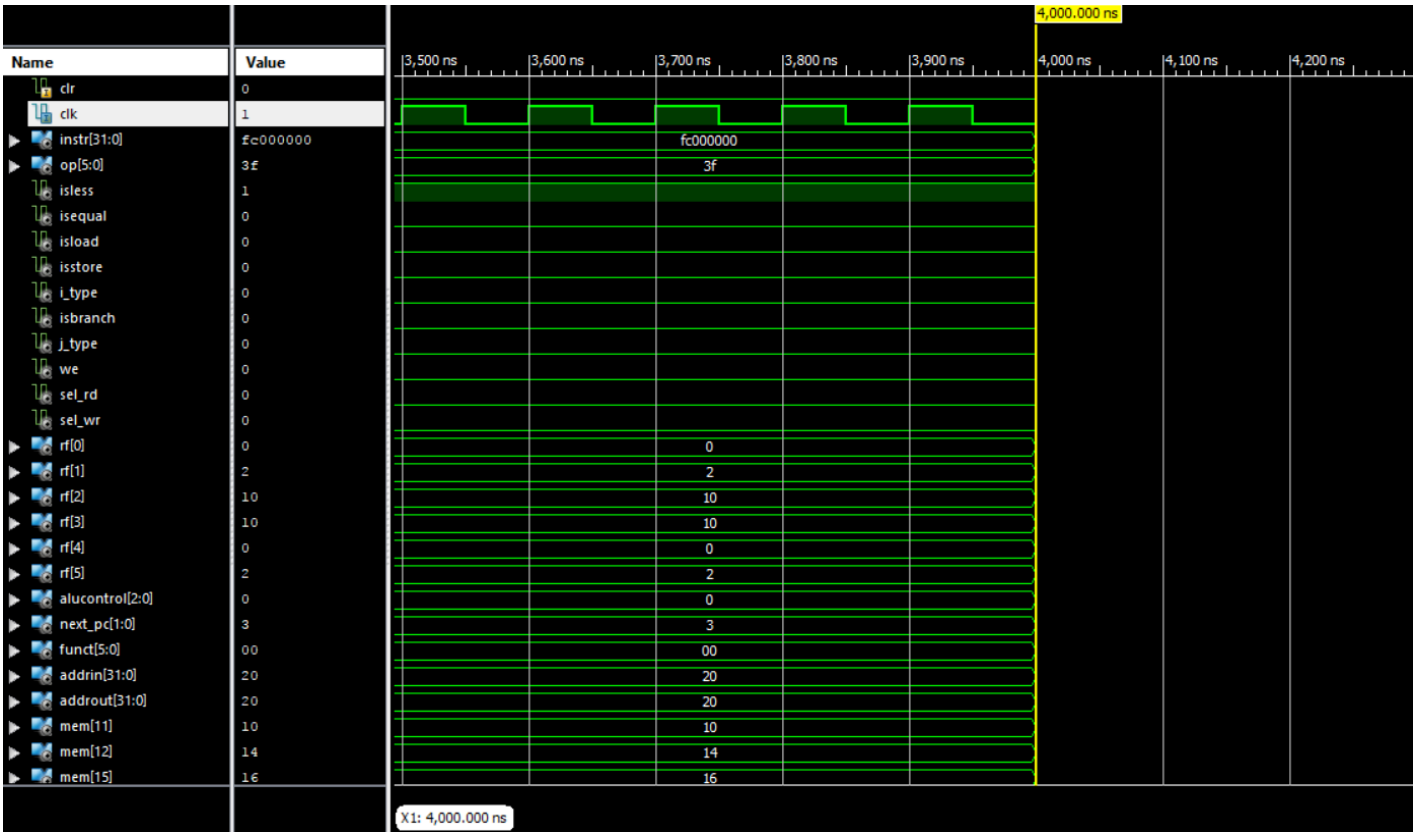
000001 00000 00001 0000000000000010 --ADDI R1, R0, 2 //R1=R0+2(decimal)
000001 00000 00011 0000000000001010 --ADDI R3, R0, 10 //R3=R0+10(decimal)
000001 00000 00100 0000000000001110 --ADDI R4, R0, 14 //R4=R0+14(decimal)
000001 00000 00101 0000000000000010 --ADDI R5, R0, 2 //R5=R0+2
001000 00011 00100 0000000000000010 --SW R4, 2(R3) //Mem[R3+2]=R4
001000 00011 00011 0000000000000001 --SW R3, 1(R3) //Mem[R3+1]=R3
000000 00100 00011 00100 00000 010001 --SUB R4, R4, R3 //R4=R4-R3
000010 00000 00100 0000000000000001 --SUBI R4, R0, 1 //R4=R0-1(decimal)
000000 00011 00010 00100 00000 010010 --AND R4, R2, R3 //R4=R2 and R3
000011 00010 00100 0000000000001010 --ANDI R4, R2, 10 //R4=R2 and 10(decimal)
000000 00011 00010 00100 00000 010011 --OR R4, R2, R3 //R4= R2 or R3
000111 00011 00010 0000000000000001 --LW R2, 1(R3) //R2=Mem[1+R3]
000100 00010 00100 0000000000001010 --ORI R4, R2, 10 //R4=R2 or 10(decimal)
000000 00011 00010 00100 00000 010100 --NOR R4, R2, R3 //R4= R2 nor R3
000101 00010 00100 0000000000001010 --SHL R4, R2, 10 //R4= R2 << 10(decimal)
000110 00010 00100 0000000000001010 --SHR R4, R2, 10 //R4=R2 >> 10(decimal)
001010 00000 00101 1111111111111110 --BEQ R5, R0, -2
001001 00100 00101 0000000000000000 --BLT R5, R4, 0
001011 00100 00101 0000000000000000 --BNE R5, R4, 0
001100 00000000000000000000010100 --JMP 20
111111 00000000000000000000000000 –HAL




1,045.013 ns
Name
Value
clr 0
clk 1
instr[31:0] 0c44000a
op[5:0] 03
isless 0
isequal 0
isload 0
isstore 0
i_type 1
isbranch 0
j_type 0
we 1
sel_rd 0
sel_wr 0
rf[0] 0
rf[1] 2
rf[2] 0
rf[3] 10
rf[4] 0
rf[5] 2
alucontrol[2:0] 3
next_pc[1:0] 0
funct[5:0] 0a
addrin[31:0] 10
addrout[31:0] 9
mem[11] 10
mem[12] 14
mem[15] 16
900 ns
1,000 ns
1,100 ns
1,200 ns
1,300 ns
1,400 ns
1,500 ns
1,600 ns
1,700 ns
00622012
0c44000a
00622013
1c620001
1044000a
00622014
1444000a
1844000a
2805fffe
X1: 1,045.013 ns


1,944.053 ns
Name
Value
clr 0
clk 1
instr[31:0] 2c850000
op[5:0] 0b
isless 1
isequal 0
isload 0
isstore 0
i_type 1
isbranch 1
j_type 0
we 0
sel_rd 0
sel_wr 0
rf[0] 0
rf[1] 2
rf[2] 10
rf[3] 10
rf[4] 0
rf[5] 2
alucontrol[2:0] 0
next_pc[1:0] 2
funct[5:0] 00
addrin[31:0] 19
addrout[31:0] 18
mem[11] 10
mem[12] 14
mem[15] 16
1,800 ns
1,900 ns
2,000 ns
2,100 ns
2,200 ns
2,300 ns
2,400 ns
2,500 ns
2,600 ns
24850000
2c850000
30000005
fc000000
X1: 1,944.053 ns

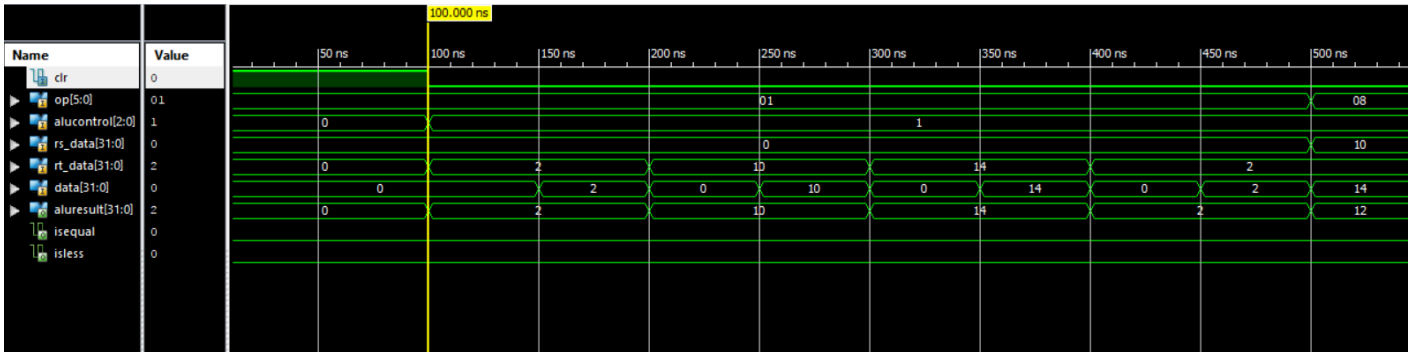

2,859.293 ns
Name
Value
clr 0
clk 0
instr[31:0] fc000000
op[5:0] 3f
isless 1
isequal 0
isload 0
isstore 0
i_type 0
isbranch 0
j_type 0
we 0
sel_rd 0
sel_wr 0
rf[0] 0
rf[1] 2
rf[2] 10
rf[3] 10
rf[4] 0
rf[5] 2
alucontrol[2:0] 0
next_pc[1:0] 3
funct[5:0] 00
addrin[31:0] 20
addrout[31:0] 20
mem[11] 10
mem[12] 14
mem[15] 16
2,700 ns
2,800 ns
2,900 ns
3,000 ns
3,100 ns
3,200 ns
3,300 ns
3,400 ns
3,500 ns
fc000000
3f
X1: 2,859.293 ns


4,000.000 ns
Name
Value
clr 0
clk 1
instr[31:0] fc000000
op[5:0] 3f
isless 1
isequal 0
isload 0
isstore 0
i_type 0
isbranch 0
j_type 0
we 0
sel_rd 0
sel_wr 0
rf[0] 0
rf[1] 2
rf[2] 10
rf[3] 10
rf[4] 0
rf[5] 2
alucontrol[2:0] 0
next_pc[1:0] 3
funct[5:0] 00
addrin[31:0] 20
addrout[31:0] 20
mem[11] 10
mem[12] 14
mem[15] 16
3,500 ns
3,600 ns
3,700 ns
3,800 ns
3,900 ns
4,000 ns
4,100 ns
4,200 ns
fc000000
3f
X1: 4,000.000 ns

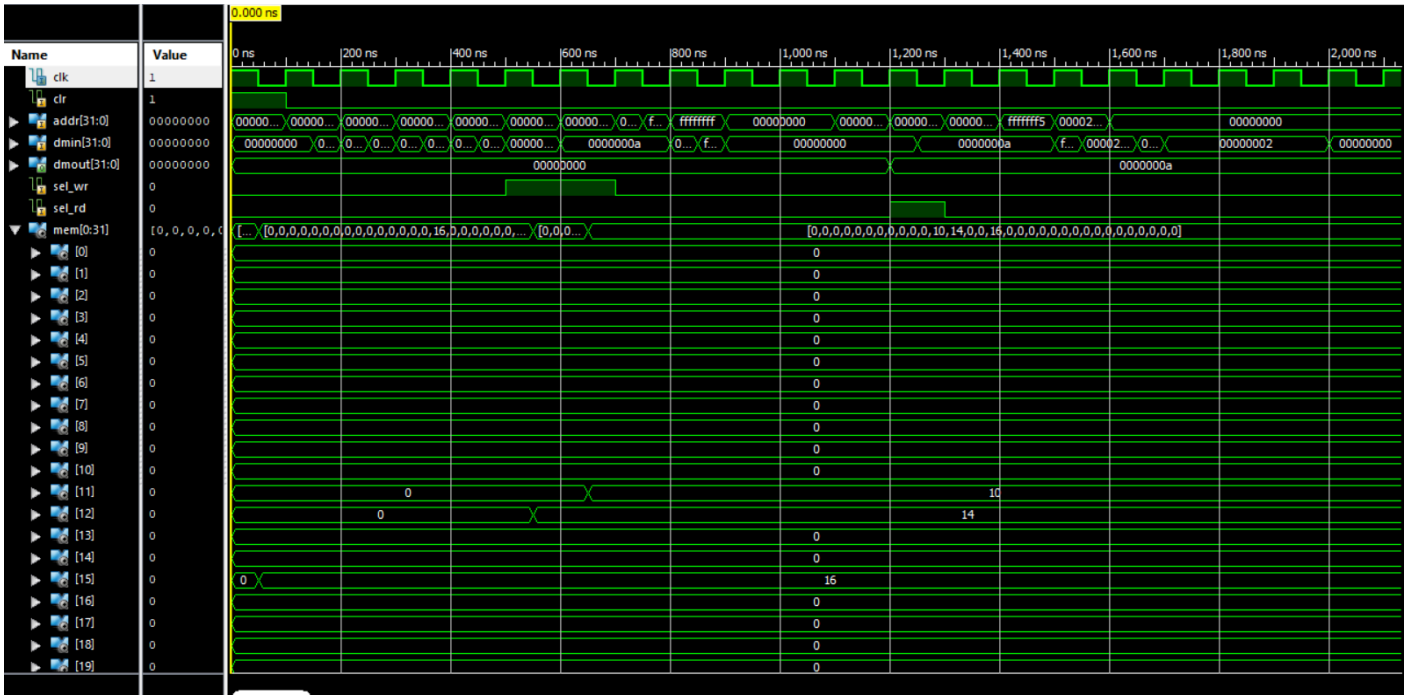
2.2 Functional Simulation for Individual Components:

We used Sample Code 2 for demonstration of working of the individual components.

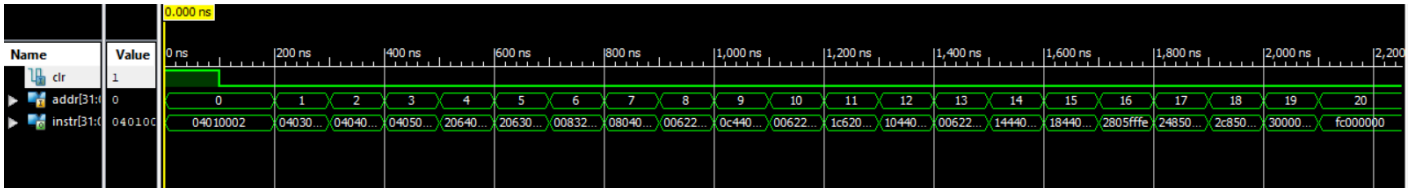
a. ALU:



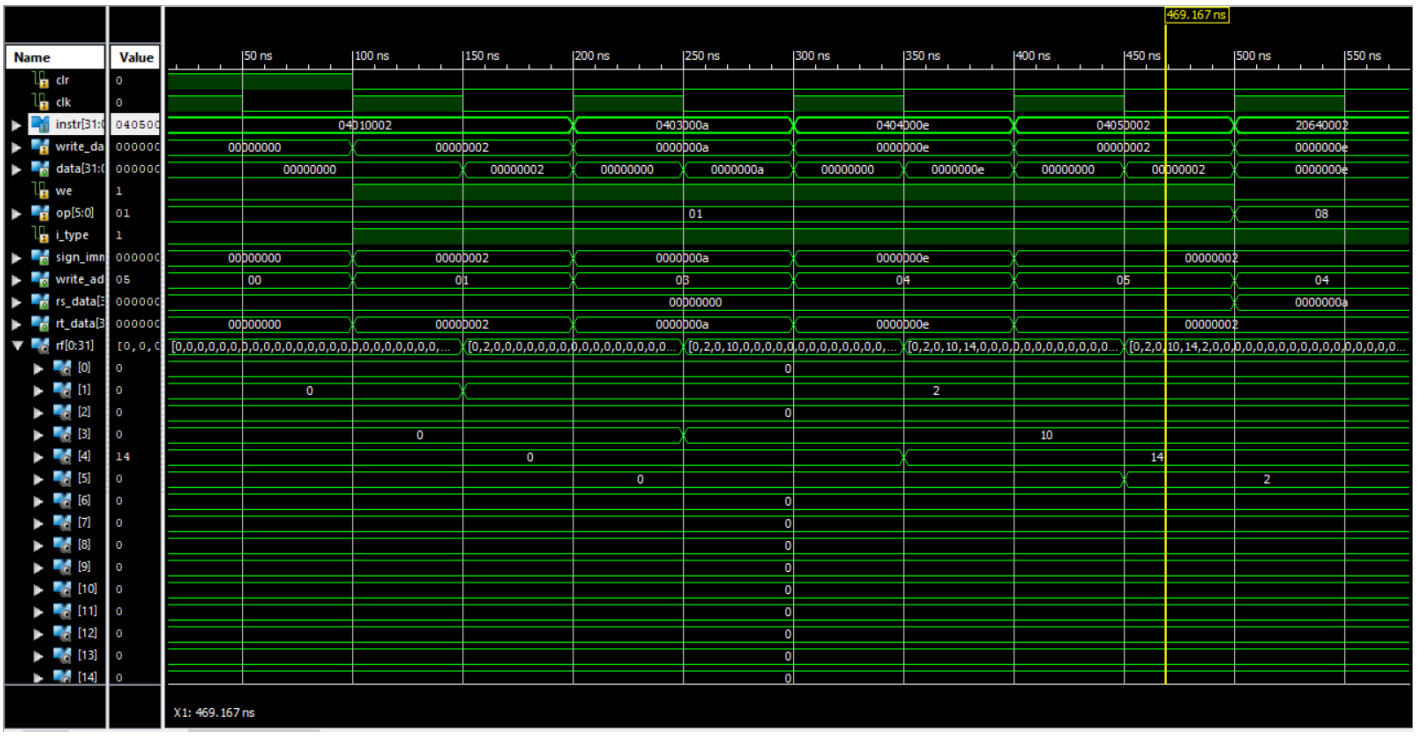
b. Data Mem:



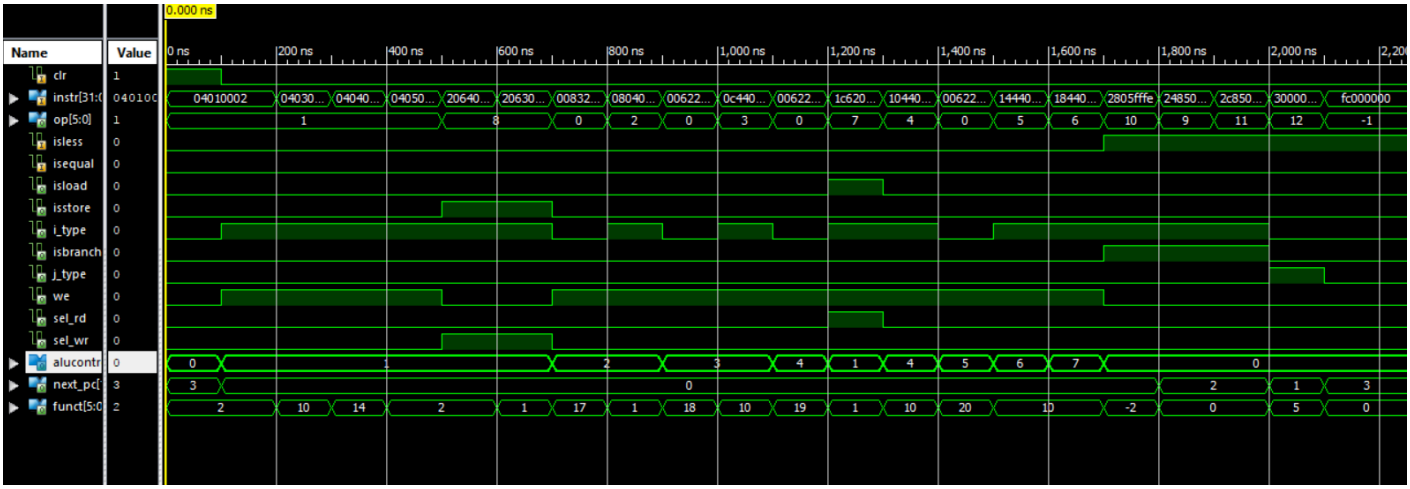
c. IMem:



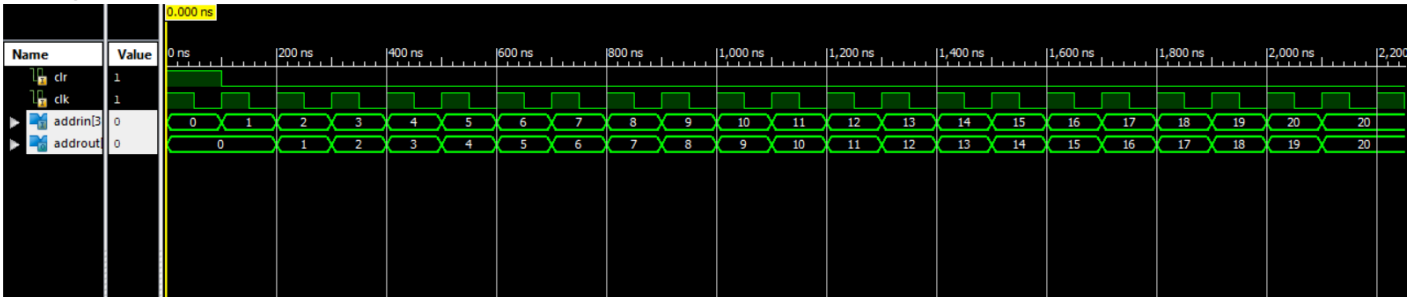
d. RF:



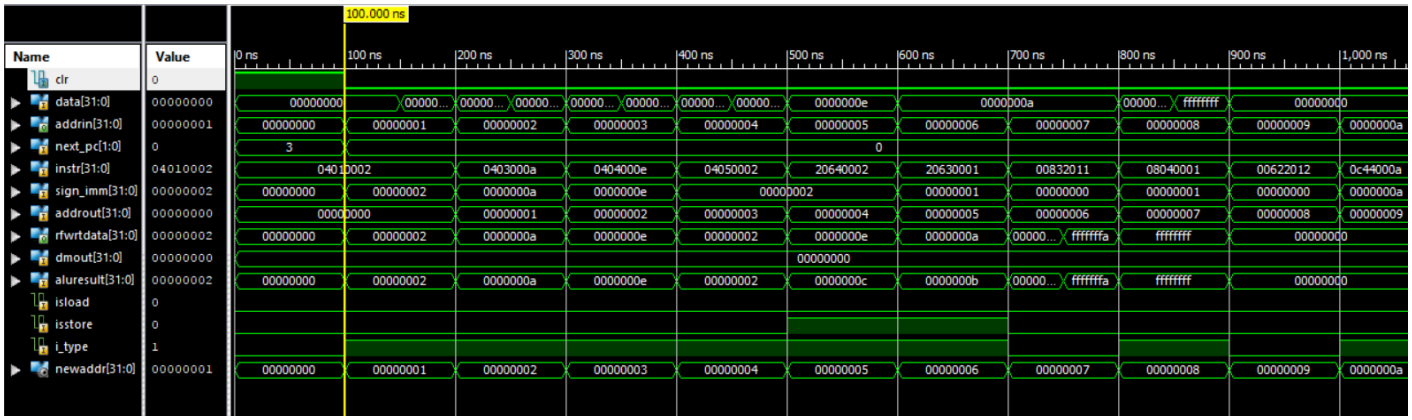
e. Decoder:



f. Program Counter:



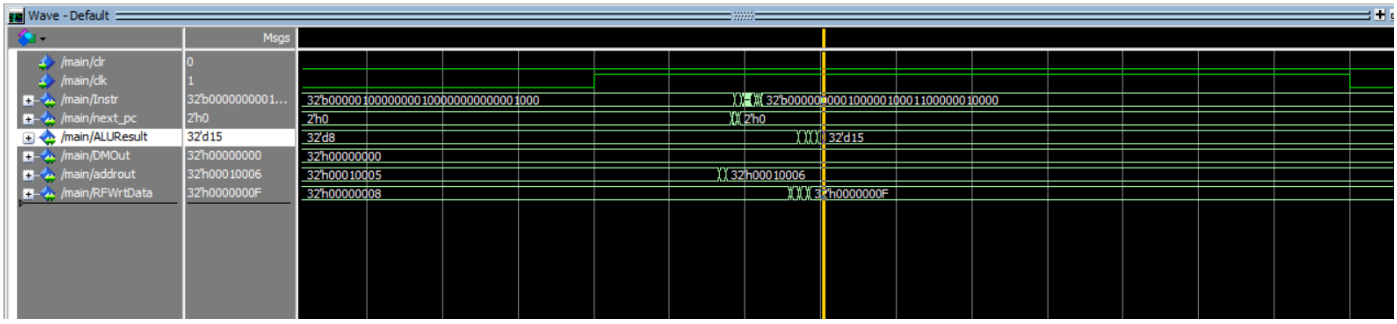
g. MUX:



2.3 Timing Simulation:

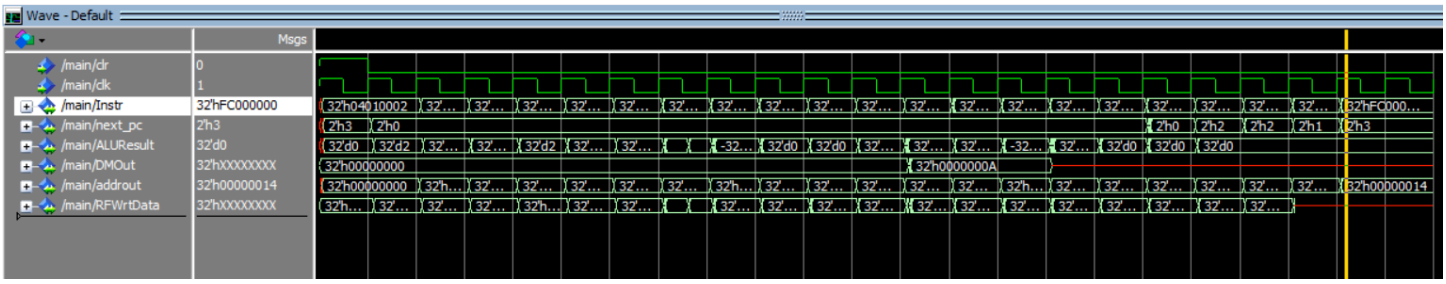
2.3.1 Sample Code 1:

00000100000000010000000000000111 --ADDI R1, R0, 7 // R1 = 7
00000100000000010000000000001000 --ADDI R2, R0, 8 // R2 = 8
00000000010000010001100000010000 --ADD R3, R1, R2 // R3 = R1 + R2 =15
11111100000000000000000000000000 --HAL // HALT



2.3.2 Sample Code 2:

000001 00000 00001 0000000000000010 --ADDI R1, R0, 2 //R1=R0+2(decimal)
000001 00000 00011 0000000000001010 --ADDI R3, R0, 10 //R3=R0+10(decimal)
000001 00000 00100 0000000000001110 --ADDI R4, R0, 14 //R4=R0+14(decimal)
000001 00000 00101 0000000000000010 --ADDI R5, R0, 2 //R5=R0+2
001000 00011 00100 0000000000000010 --SW R4, 2(R3) //Mem[R3+2]=R4
001000 00011 00011 0000000000000001 --SW R3, 1(R3) //Mem[R3+1]=R3
000000 00100 00011 00100 00000 010001 --SUB R4, R4, R3 //R4=R4-R3
000010 00000 00100 0000000000000001 --SUBI R4, R0, 1 //R4=R0-1(decimal)
000000 00011 00010 00100 00000 010010 --AND R4, R2, R3 //R4=R2 and R3
000011 00010 00100 0000000000001010 --ANDI R4, R2, 10 //R4=R2 and 10(decimal)
000000 00011 00010 00100 00000 010011 --OR R4, R2, R3 //R4= R2 or R3
000111 00011 00010 0000000000000001 --LW R2, 1(R3) //R2=Mem[1+R3]
000100 00010 00100 0000000000001010 --ORI R4, R2, 10 //R4=R2 or 10(decimal)
000000 00011 00010 00100 00000 010100 --NOR R4, R2, R3 //R4= R2 nor R3
000101 00010 00100 0000000000001010 --SHL R4, R2, 10 //R4= R2 << 10(decimal)
000110 00010 00100 0000000000001010 --SHR R4, R2, 10 //R4=R2 >> 10(decimal)
001010 00000 00101 1111111111111110 --BEQ R5, R0, -2
001001 00100 00101 0000000000000000 --BLT R5, R4, 0
001011 00100 00101 0000000000000000 --BNE R5, R4, 0
001100 00000000000000000000010100 --JMP 20
111111 00000000000000000000000000 –HAL



3 Design Analysis:

3.1 Area Analysis:

3.1.1 Synthesis Report:

```
Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:
 Number of Slice Registers:             4248  out of  126800     3%
 Number of Slice LUTs:                  6213  out of  63400      9%
    Number used as Logic:               6165  out of  63400      9%
    Number used as Memory:                48  out of  19000      0%
       Number used as RAM:                48

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    6239
   Number with an unused Flip Flop:     1991  out of   6239     31%
   Number with an unused LUT:             26  out of   6239      0%
   Number of fully used LUT-FF pairs:   4222  out of   6239     67%
   Number of unique control sets:         16

IO Utilization:
 Number of IOs:                          164
 Number of bonded IOBs:                  164  out of    210     78%
    IOB Flip Flops/Latches:               64

Specific Feature Utilization:
 Number of Block RAM/FIFO:                1  out of    135      0%
    Number using Block RAM only:          1
 Number of BUFG/BUFGCTRLs:                5  out of     32     15%
```

3.1.2 Place and Route Report:

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                 4,247 out of 126,800    3%
    Number used as Flip Flops:               4,128
    Number used as Latches:                    119
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                      6,083 out of  63,400    9%
    Number used as logic:                    6,037 out of  63,400    9%
      Number using O6 output only:           5,879
      Number using O5 output only:              30
      Number using O5 and O6:                  128
      Number used as ROM:                        0
    Number used as Memory:                      44 out of  19,000    1%
      Number used as Dual Port RAM:             44
        Number using O6 output only:             0
        Number using O5 output only:             0
        Number using O5 and O6:                 44
      Number used as Single Port RAM:            0
      Number used as Shift Register:             0
    Number used exclusively as route-thrus:      2
      Number with same-slice register load:      0
      Number with same-slice carry load:         2
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                 1,866 out of  15,850   11%
  Number of LUT Flip Flop pairs used:        6,135
    Number with an unused Flip Flop:         1,888 out of   6,135   30%
    Number with an unused LUT:                  52 out of   6,135    1%
    Number of fully used LUT-FF pairs:       4,195 out of   6,135   68%
    Number of slice register sites lost
      to control set restrictions:               0 out of 126,800    0%
```

3.2 Performance Analysis:

3.2.1 Synthesis Report:

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.312ns (Maximum Frequency: 188.245MHz)
   Minimum input arrival time before clock: 8.358ns
   Maximum output required time after clock: 5.606ns
   Maximum combinational path delay: 6.262ns
```

3.2.2 Place and Route Report:

```
------------------------------------------------------------------------------------------------------
 Constraint                                |    Check    | Worst Case |  Best Case | Timing |   Timing
                                           |             |    Slack   | Achievable | Errors |    Score
------------------------------------------------------------------------------------------------------
 Autotimespec constraint for clock net clk | SETUP       |         N/A|    13.567ns|     N/A|
 _BUFGP                                    | HOLD        |     0.203ns|            |       0|
------------------------------------------------------------------------------------------------------
```

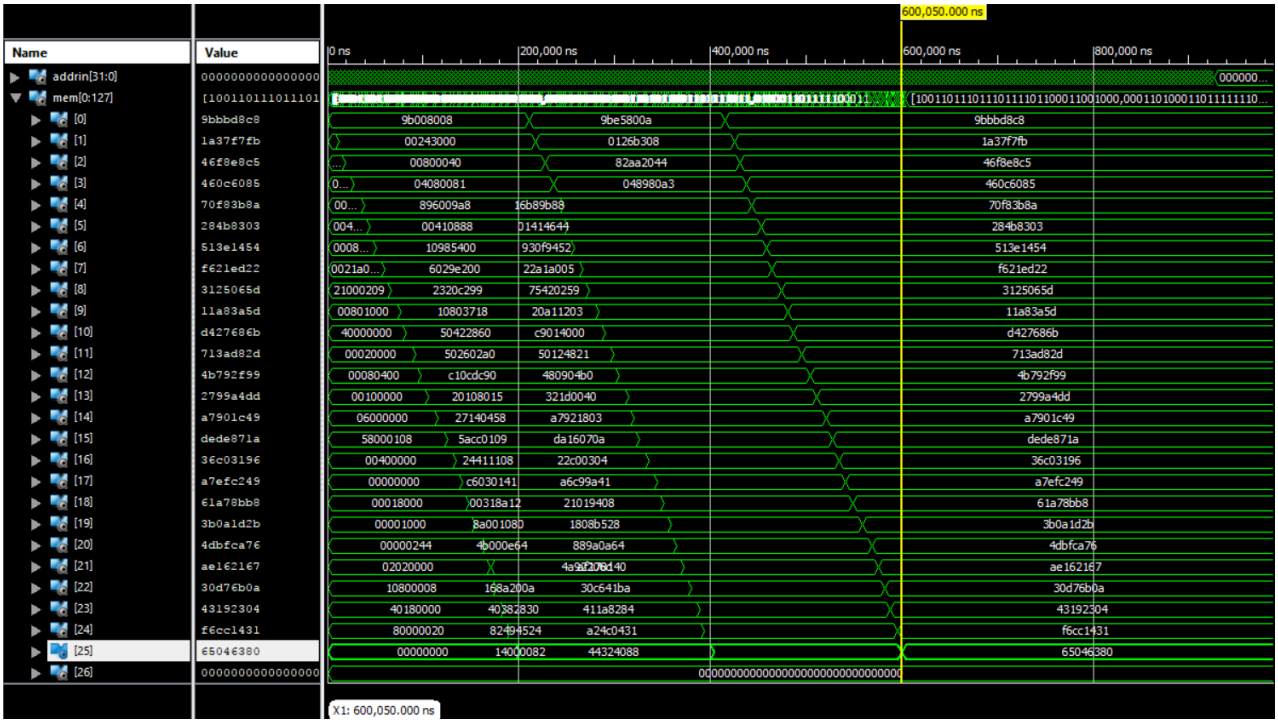
Critical Path Delay(from the report): 13.567ns
Maximum Frequency: 1/13.567ns = 73.70 Mhz

4 RC5 Implementation:

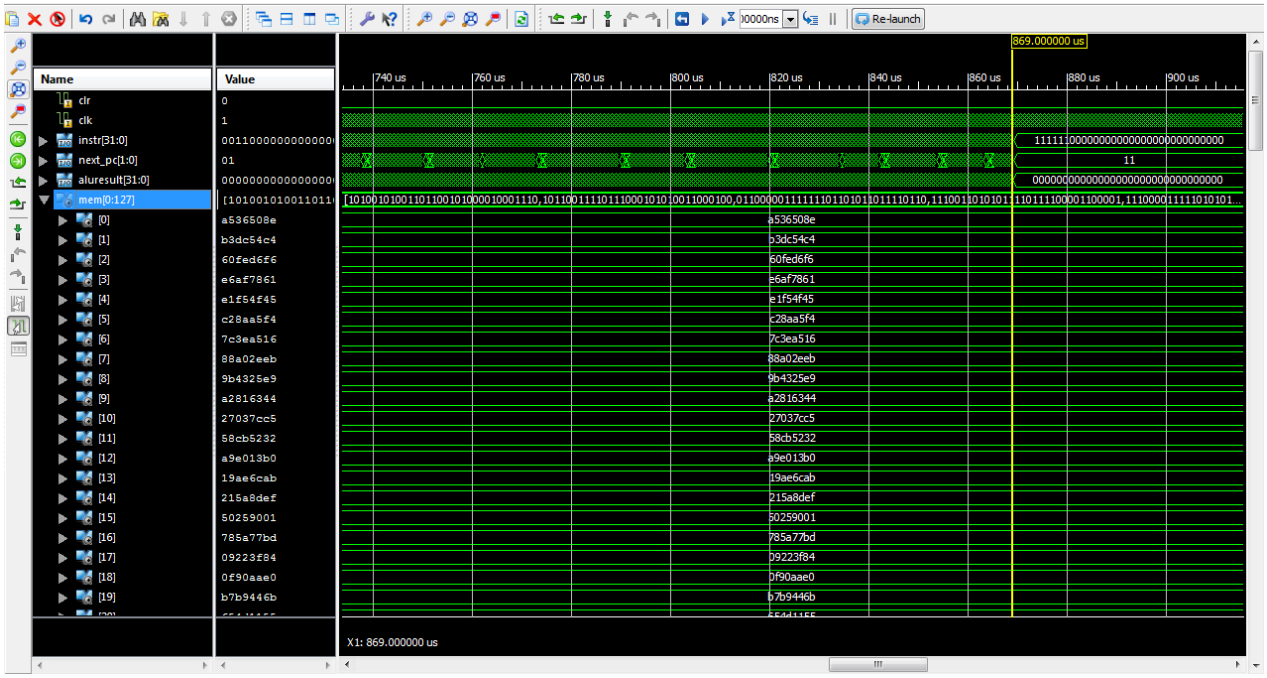
4.1 Functional Simulation:

4.1.1 Rc5 Key Generation:

a. The ukey given here is all 00s.



b. The ukey here is:
"10011100111110100110101010000111",
"11100000111110000001111000000000",
"00011110000111100000111100011111",
"10110110011011100111001111000111"

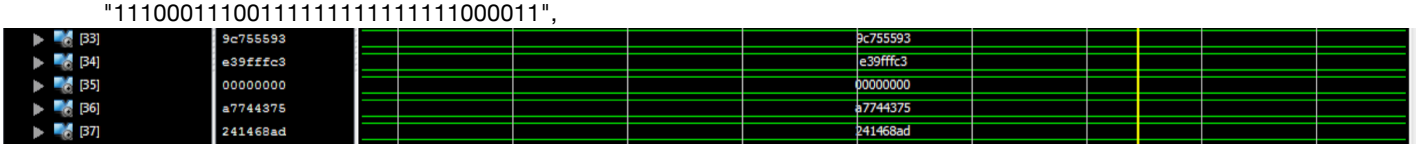


c. Ukey: (This is the one we were assigned from the excel file)
"10011100111110100110101010000111",
"11100000111110000000111100000000",
"00011110000111100000111100011111",
"10110110011011100111001111000111"

4.1.2 RC5 Encryption:

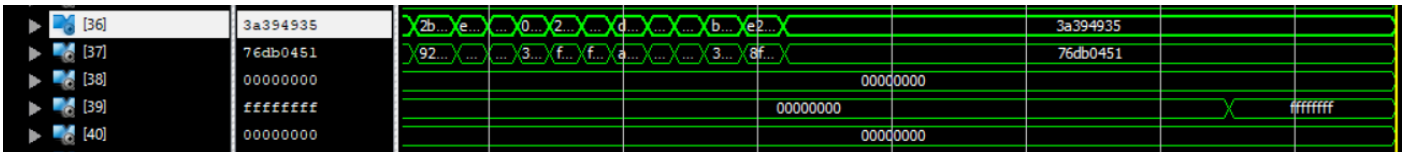
a. din = ffffffff00000000

b. din = "10011100011101010101010110010011",
"11100011100111111111111111111000011",

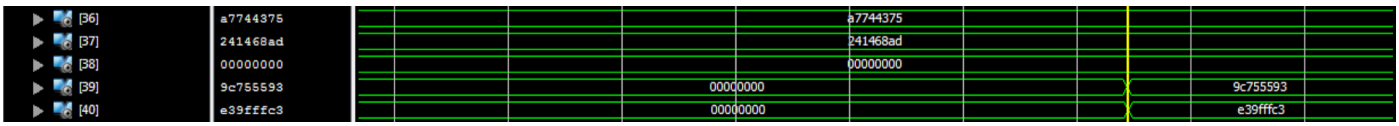


4.1.3 RC5 Decryption:

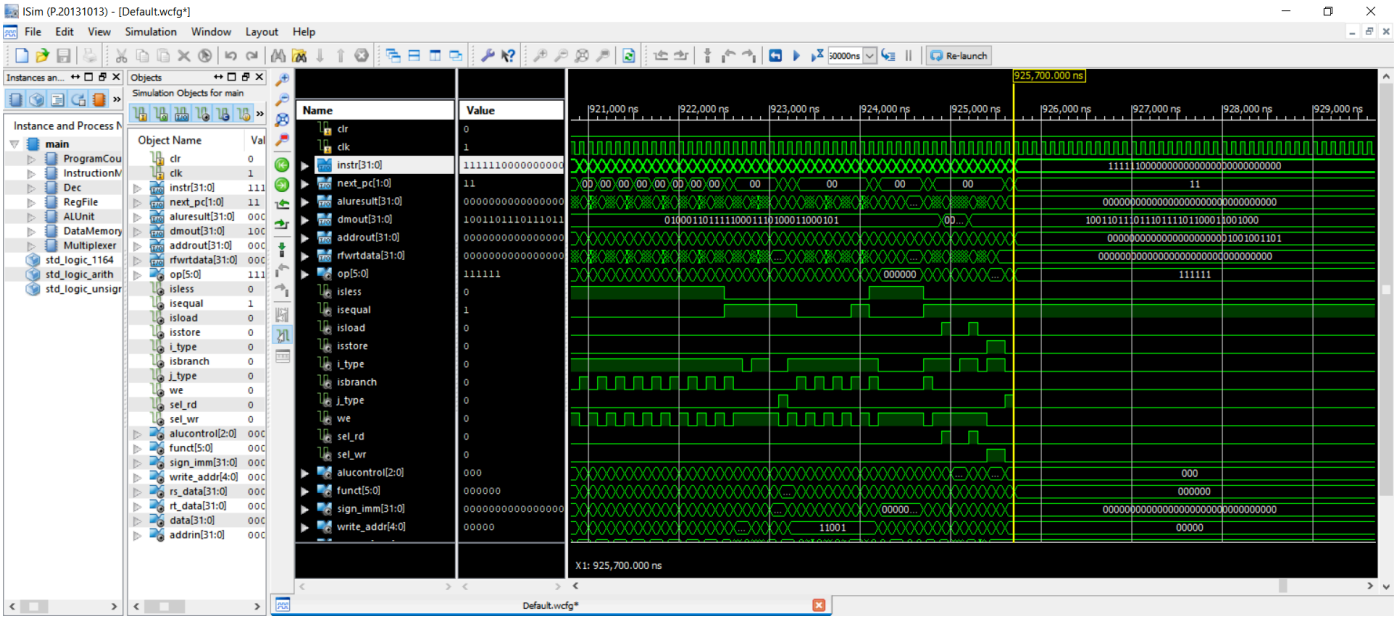
a. din = 3a39493576db0451

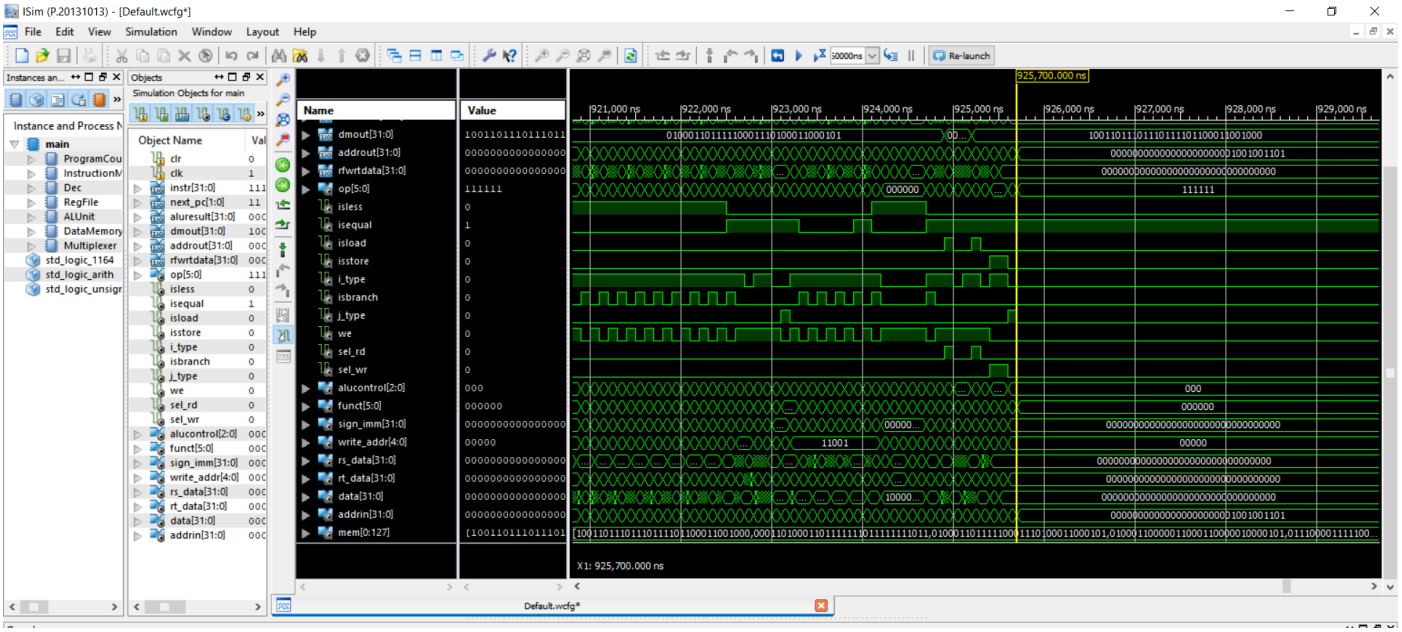


b. din = a774437s241468ad



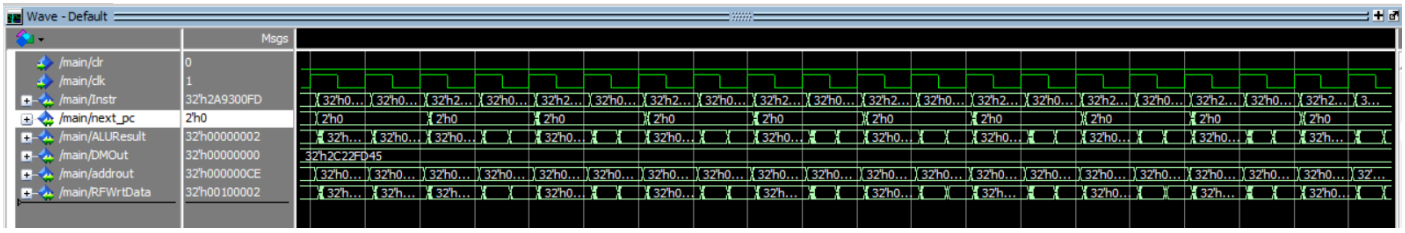
4.1.4 Complete RC5:





4.2 Timing Simulation:

4.2.1 Complete RC5:



4.3 Assembly Code:

4.3.1 Description:

The 32 32-bit register file was used to store the various values in the execution as follows

R0	Hard coded to all 0's
R1	A_reg
R2	B_reg
R3	A_tmp1
R4	A_tmp2
R5	AB_tmp
R5	B_tmp1
R6	B_tmp2
R7	B_tmp2
R8	L_ARR_tmp
R9	S_ARR_tmp
R10	L[i]
R11	S[j]
R12	I
R13	I_tmp

R14	J
R15	J_tmp
R16	K
R17	K_tmp
R18	B_tmp1_tmp (used for rotation of ab_tmp)
R19	AB_tmp_tmp (used for rotation of ab_tmp)
R20	How much to rotate by
R21	Rotation output
R22	AB_xor
R23	A_rot
R24	BA_xor
R25	B_rot
R26	AB_or
R27	AB_and
R28	Rotate amount after mod by31
R29	Hard coded to all 1's
R30	Address for jump back
R31	Temporary address for jump back

Initially R1, R2 (a_reg and b_reg) are initialized to 0 R12 is assigned the value of 25; R14 of 3 and R16 to 77 as values of I, j and k respectively. The temporary counters all initialized to 0. Initial value of ukey (1st location) is loaded in R10.

A loop now begins and the following actions happen sequentially
1. Load current S_ARR value into R9
2. Add R1 (a_reg) and R2 (b_reg) and store it in R3 (a_tmp1)
3. Add current value of s-array into R3
4. Left rotate R3 by 3 and save it in R4 (a_tmp2)
5. store value of R1 in current s-array location
6. add R1 and R2 into R5 (ab_tmp)
7. Load current L_ARR value into R8
8. Add R8 and R5 into R6 (b_tmp1)
9. Modulo R5 by 31 to get the last 5 bits to ascertain rotation amount
10. Transfer R6 to R18 and R5 to R19 for rotation purpose
11. Rotate R18 on the basis of R19
12. Transfer R21 (rotate_out) to R7 (b_tmp1) and R2
13. Store value of R2 into l-array
14. Check if all counters have reached their limit; if I and j have then reset them to 0 and when k has reached its limit exit out of loop else go back to start of loop.

This is how key expansion is executed. The final keys are stored in dmem locations 0-25. While the initial keys can be found from data location 28-31

Now for encryption all the registers are reinitialized to 0 with the exception of r12 which is initialized to 25. The inputs are read from line number 33 and 34. The encryption is performed step by step just like the vhdl code taught in class with the xor instruction being executed by a combination of ADD, OR and SUB instructions on the 2 registers, everything else works exactly like the code. Finally, the output is store in Data memory locations 36 and 37.

Similarly, decryptions work also like the vhdl code done in previous labs and uses the reverse logic of encryption. It picks up the input from the location of encryption output and decrypting it hence giving us back the original input back in location 39 and 40.

The main talking point for the assembly code are infact the rotation logic which takes a register R20 and increments it by 1, checking every time against the preset rotation amount saved in R19 and once they both match; the execution jumps to a corresponding rotation block which rotates R18 by the current value of R19. Using Shift lefts and Shift rights. The rotation block called is as follows

1. Shift an all 1 register R29 left by the required rotation amount
2. And R29 to R18 (the number to be rotated) and save the answer in R29
3. Shift R29 right by the required rotation amount
4. Shift R18 left by 32-rotation amount
5. Or R29 and R19 and store it in R21 (rotation output)

Now to reuse this rotation code up to 5 times. Once in expansion, twice in encryption and decryption respectively; we save the jump back address in R30 and then after rotation; R31 is loaded with all 5 possible jump back address one by one and as soon as it matches with R30 it sends the execution back to the required instruction.

4.4 Machine Code:

00000000000000000000100000010000	ADD R0 R0 R1
00000000000000000001000000010000	ADD R0 R0 R2
00000100000011000000000000011001	ADDI R0 R12 25
00000100000011010000000000000000	ADDI R0 R13 0
00000100000011100000000000000011	ADDI R0 R14 3
00000100000011110000000000000000	ADDI R0 R15 0
00000100000100000000000001001101	ADDI R0 R16 77
00000100000100010000000000000000	ADDI R0 R17 0
00000100000010110000000000000000	ADDI R0 R11 0
00000100000010100000000000011100	ADDI R0 R10 28
00000100000011110000000000011100	ADDI R0 R30 28

00011101011010010000000000000000	LW R11 R9 0	#key expansion
00000000001000100001100000010000	ADD R1 R2 R3	
00000000011010010001100000010000	ADD R3 R9 R3	
00110000000000000000000000110001	JMP 49	
00000000000001000000100000010000	ADD R0 R4 R1	
00100001011000010000000000000000	SW R11 R1 0	
00000000001000100010100000010000	ADD R1 R2 R5	
00011101010010000000000000000000	LW R10 R8 0	
00000001000001010011000000010000	ADD R8 R5 R6	
00001100101001010000000000011111	ANDI R5 R5 31	
00000000000001101001000000010000	ADD R0 R6 R18	
00000000000001011001100000010000	ADD R0 R5 R19	
00110000000000000000000010011000	JMP 152	

00000000000101010011100000010000	ADD R0 R21 R7
00000000000001110001000000010000	ADD R0 R7 R2
00100001010000100000000000000000	SW R10 R2 0
00101001100011010000000000001001	BEQ R12 R13 9
00000101101011010000000000000001	ADDI R13 R13 1
00000101011010110000000000000001	ADDI R11 R11 1
00101001110011110000000000001010	BEQ R14 R15 10
00000101111011110000000000000001	ADDI R15 R15 1
00000101010010100000000000000001	ADDI R10 R10 1
00101010001100000000000000010100	BEQ R17 R16 20
00000110001100010000000000000001	ADDI R17 R17 1
00110000000000000000000000001111	JMP 15
00000100000010110000000000000000	ADDI R0 R11 0
00000100000011010000000000000000	ADDI R0 R13 0
00110000000000000000000000100010	JMP 34
00000100000010100000000000011100	ADDI R0 R10 28
00000100000011110000000000000000	ADDI R0 R15 0
00110000000000000000000000100101	JMP 37
00000100000111011111111111111111	ADDI R0 R29 65535
00010111101111010000000000011101	SHL R29 R29 29
00000000011111011110100000010010	AND R3 R29 R29
00011011101111010000000000011101	SHR R29 R29 29
00010100011000110000000000000011	SHL R3 R3 3
00000000011111010010000000010011	OR R3 R29 R4
00110000000000000000000000010011	JMP 19
00000000000000000111000000010000	ADD R0 R0 R28 #encryption
00000100000011010000000000000000	ADDI R0 R13 0
00000100000011000000000000011001	ADDI R0 R12 25
00011100000000010000000000100001	LW R0 R1 33
00011100000000010000000000100010	LW R0 R2 34
00000100000010110000000000000000	ADDI R0 R11 0
00011101011010010000000000000000	LW R11 R9 0
00000000001010010000100000010000	ADD R1 R9 R1
00000101011010110000000000000001	ADDI R11 R11 1
00000101101011010000000000000001	ADDI R13 R13 1
00011101011010010000000000000000	LW R11 R9 0
00000000010010010001000000010000	ADD R2 R9 R2
00000100000111100000000001001111	ADDI R0 R30 79
00000000001000101101000000010011	OR R1 R2 R26
00000000001000101101100000010010	AND R1 R2 R27
00000011010110111011000000010001	SUB R26 R27 R22
00000000000101101001000000010000	ADD R0 R22 R18
00001100010111000000000000011111	ANDI R2 R28 31

```
00000000000111001001100000010000    ADD R0 R28 R19
00110000000000000000000010011000    JMP 152
00000000000101011011100000010000    ADD R0 R21 R23
00000101011010110000000000000001    ADDI R11 R11 1
00000101101011010000000000000001    ADDI R13 R13 1
00011101011010010000000000000000    LW R11 R9 0
00000101110100100000100000010000    ADD R23 R9 R1
00000000001000101101000000010011    OR R1 R2 R26
00000000001000101101100000010010    AND R1 R2 R27
00000110101101111100000000010001    SUB R26 R27 R24
00000000000110001001000000010000    ADD R0 R24 R18
00001100001111000000000000011111    ANDI R1 R28 31
00000000000111001001100000010000    ADD R0 R28 R19
00000100000111100000000001011100    ADDI R0 R30 92
00110000000000000000000010011000    JMP 152
00000000000101011100100000010000    ADD R0 R21 R25
00000101011010110000000000000001    ADDI R11 R11 1
00000101101011010000000000000001    ADDI R13 R13 1
00011101011010010000000000000000    LW R11 R9 0
00000011001010010001000000010000    ADD R25 R9 R2
00100000000000010000000000100100    SW R0 R1 36
00100000000000010000000000100101    SW R0 R2 37
00101001100011010000000000000011    BEQ R12 R13 3
00110000000000000000000001000111    JMP 71


00011100000000010000000000100100    LW R0 R1 36          #decryption
00011100000000010000000000100101    LW R0 R2 37
00000000000000000111000000010000    ADD R0 R0 R28
00000100000011000000000000000010    ADDI R0 R12 2
00000100000011010000000000011001    ADDI R0 R13 25
00000100000010110000000000011001    ADDI R0 R11 25

00011101011010010000000000000000    LW R11 R9 0
00000100000111100000000001110101    ADDI R0 R30 117
00000000010010010001000000010001    SUB R2 R9 R2
00000000000000010100100000010000    ADD R0 R2 R18
00001100001111000000000000011111    ANDI R1 R28 31
00000000000111001001100000010000    ADD R0 R28 R19
00110000000000000000000011011011    JMP 219
00000000000101011100100000010000    ADD R0 R21 R25
00000011001000011101000000010011    OR R25 R1 R26
00000011001000011101100000010010    AND R25 R1 R27
00000011010110111100000000010001    SUB R26 R27 R24
00000000000110000001000000010000    ADD R0 R24 R2
00001001101011010000000000000001    SUBI R13 R13 1
00001001011010110000000000000001    SUBI R11 R11 1
00011101011010010000000000000000    LW R11 R9 0
00000000001010010000100000010001    SUB R1 R9 R1
```

00000000000000011001000000010000	ADD R0 R1 R18
00001100010111000000000000011111	ANDI R2 R28 31
00000000000111001001100000010000	ADD R0 R28 R19
00000100000111100000000010000011	ADDI R0 R30 131
00110000000000000000000011011011	JMP 219
00000000000101011011100000010000	ADD R0 R21 R23
00000010111000101101000000010011	OR R23 R2 R26
00000010111000101101100000010010	AND R23 R2 R27
00000011010110111011000000010001	SUB R26 R27 R22
00000000000101100000100000010000	ADD R0 R22 R1
00101001100011010000000000000100	BEQ R12 R13 4
00001001101011010000000000000001	SUBI R13 R13 1
00001001011010110000000000000001	SUBI R11 R11 1
00110000000000000000000001101110	JMP 110

00001001011010110000000000000001	SUBI R11 R11 1
00011101011010010000000000000000	LW R11 R9 0
00000000010010010001000000010001	SUB R2 R9 R2
00001001011010110000000000000001	SUBI R11 R11 1
00011101011010010000000000000000	LW R11 R9 0
00000000001010010000100000010001	SUB R1 R9 R1
00100000000000010000000000100111	SW R0 R1 39
00100000000000010000000000101000	SW R0 R2 40
00110000000000000000001000000100	JMP 516

00000100000111011111111111111111	ADDI R0 R29 65535	#rotate
00000000000000001010000000010000	ADD R0 R0 R20	
00101010100100110000000101011011	BEQ R20 R19 347	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010000000	BEQ R20 R19 128	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010000101	BEQ R20 R19 133	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010001010	BEQ R20 R19 138	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010001111	BEQ R20 R19 143	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010010100	BEQ R20 R19 148	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010011001	BEQ R20 R19 153	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010011110	BEQ R20 R19 158	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010100011	BEQ R20 R19 163	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010101000	BEQ R20 R19 168	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010101101	BEQ R20 R19 173	

00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010110010	BEQ R20 R19 178	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010110111	BEQ R20 R19 183	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000010111100	BEQ R20 R19 188	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011000001	BEQ R20 R19 193	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011000110	BEQ R20 R19 198	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011001011	BEQ R20 R19 203	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011010000	BEQ R20 R19 208	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011010101	BEQ R20 R19 213	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011011010	BEQ R20 R19 218	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011011111	BEQ R20 R19 223	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011100100	BEQ R20 R19 228	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011101001	BEQ R20 R19 233	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011101110	BEQ R20 R19 238	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011110011	BEQ R20 R19 243	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011111000	BEQ R20 R19 248	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000011111101	BEQ R20 R19 253	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100000010	BEQ R20 R19 258	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100000111	BEQ R20 R19 263	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100001100	BEQ R20 R19 268	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100010001	BEQ R20 R19 273	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100010110	BEQ R20 R19 278	

00000100000111011111111111111111	ADDI R0 R29 65535	#rotate
00000000000000001010000000010000	ADD R0 R0 R20	
00101010100100110000000100011000	BEQ R20 R19 280	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000100001111	BEQ R20 R19 271	

00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000100000110	BEQ R20 R19 262
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011111101	BEQ R20 R19 253
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011110100	BEQ R20 R19 244
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011101011	BEQ R20 R19 235
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011100010	BEQ R20 R19 226
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011011001	BEQ R20 R19 217
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011010000	BEQ R20 R19 208
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000011000111	BEQ R20 R19 199
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010111110	BEQ R20 R19 190
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010110101	BEQ R20 R19 181
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010101100	BEQ R20 R19 172
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010100011	BEQ R20 R19 163
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010011010	BEQ R20 R19 154
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010010001	BEQ R20 R19 145
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000010001000	BEQ R20 R19 136
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001111111	BEQ R20 R19 127
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001110110	BEQ R20 R19 118
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001101101	BEQ R20 R19 109
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001100100	BEQ R20 R19 100
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001011011	BEQ R20 R19 91
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001010010	BEQ R20 R19 82
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001001001	BEQ R20 R19 73
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000001000000	BEQ R20 R19 64
00000110100101000000000000000001	ADDI R20 R20 1
00101010100100110000000000110111	BEQ R20 R19 55
00000110100101000000000000000001	ADDI R20 R20 1

00101010100100110000000000101110	BEQ R20 R19 46	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000000100101	BEQ R20 R19 37	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000000011100	BEQ R20 R19 28	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000000010011	BEQ R20 R19 19	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000000001010	BEQ R20 R19 10	
00000110100101000000000000000001	ADDI R20 R20 1	
00101010100100110000000000000001	BEQ R20 R19 1	
00010111101111010000000000011111	SHL R29 R29 31	#rotate amount
00000010010111011110100000010010	AND R18 R29 R29	
00011011101111010000000000011111	SHR R29 R29 31	
00010110010100100000000000000001	SHL R18 R18 1	
00000010010111011010100000010011	OR R18 R29 R21	
00110000000000000000000111111001	JMP 505	
00010111101111010000000000011110	SHL R29 R29 30	
00000010010111011110100000010010	AND R18 R29 R29	
00011011101111010000000000011110	SHR R29 R29 30	
00010110010100100000000000000010	SHL R18 R18 2	
00000010010111011010100000010011	OR R18 R29 R21	
00110000000000000000000111111001	JMP 505	
00010111101111010000000000011101	SHL R29 R29 29	
00000010010111011110100000010010	AND R18 R29 R29	
00011011101111010000000000011101	SHR R29 R29 29	
00010110010100100000000000000011	SHL R18 R18 3	
00000010010111011010100000010011	OR R18 R29 R21	
00110000000000000000000111111001	JMP 505	
00010111101111010000000000011100	SHL R29 R29 28	
00000010010111011110100000010010	AND R18 R29 R29	
00011011101111010000000000011100	SHR R29 R29 28	
00010110010100100000000000000100	SHL R18 R18 4	
00000010010111011010100000010011	OR R18 R29 R21	
00110000000000000000000111111001	JMP 505	
00010111101111010000000000011011	SHL R29 R29 27	
00000010010111011110100000010010	AND R18 R29 R29	
00011011101111010000000000011011	SHR R29 R29 27	
00010110010100100000000000000101	SHL R18 R18 5	
00000010010111011010100000010011	OR R18 R29 R21	
00110000000000000000000111111001	JMP 505	
00010111101111010000000000011010	SHL R29 R29 26	
00000010010111011110100000010010	AND R18 R29 R29	

00011011101111010000000000011010	SHR R29 R29 26
00010110010100100000000000000110	SHL R18 R18 6
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000011001	SHL R29 R29 25
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000011001	SHR R29 R29 25
00010110010100100000000000000111	SHL R18 R18 7
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000011000	SHL R29 R29 24
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000011000	SHR R29 R29 24
00010110010100100000000000001000	SHL R18 R18 8
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010111	SHL R29 R29 23
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000010111	SHR R29 R29 23
00010110010100100000000000001001	SHL R18 R18 9
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010110	SHL R29 R29 22
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000010110	SHR R29 R29 22
00010110010100100000000000001010	SHL R18 R18 10
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010101	SHL R29 R29 21
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000010101	SHR R29 R29 21
00010110010100100000000000001011	SHL R18 R18 11
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010100	SHL R29 R29 20
00000010010111011111010000010010	AND R18 R29 R29
00011011101111010000000000010100	SHR R29 R29 20
00010110010100100000000000001100	SHL R18 R18 12
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010011	SHL R29 R29 19
00000010010111011111010000010010	AND R18 R29 R29

00011011101111010000000000010011	SHR R29 R29 19
00010110010100100000000000001101	SHL R18 R18 13
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010010	SHL R29 R29 18
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000010010	SHR R29 R29 18
00010110010100100000000000001110	SHL R18 R18 14
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010001	SHL R29 R29 17
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000010001	SHR R29 R29 17
00010110010100100000000000001111	SHL R18 R18 15
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000010000	SHL R29 R29 16
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000010000	SHR R29 R29 16
00010110010100100000000000010000	SHL R18 R18 16
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001111	SHL R29 R29 15
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001111	SHR R29 R29 15
00010110010100100000000000010001	SHL R18 R18 17
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001110	SHL R29 R29 14
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001110	SHR R29 R29 14
00010110010100100000000000010010	SHL R18 R18 18
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001101	SHL R29 R29 13
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001101	SHR R29 R29 13
00010110010100100000000000010011	SHL R18 R18 19
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001100	SHL R29 R29 12
00000010010111011110100000010010	AND R18 R29 R29

00011011101111010000000000001100	SHR R29 R29 12
00010110010100100000000000010100	SHL R18 R18 20
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001011	SHL R29 R29 11
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001011	SHR R29 R29 11
00010110010100100000000000010101	SHL R18 R18 21
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001010	SHL R29 R29 10
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001010	SHR R29 R29 10
00010110010100100000000000010110	SHL R18 R18 22
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001001	SHL R29 R29 9
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001001	SHR R29 R29 9
00010110010100100000000000010111	SHL R18 R18 23
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000001000	SHL R29 R29 8
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000001000	SHR R29 R29 8
00010110010100100000000000011000	SHL R18 R18 24
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000000111	SHL R29 R29 7
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000000111	SHR R29 R29 7
00010110010100100000000000011001	SHL R18 R18 25
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000000110	SHL R29 R29 6
00000010010111011110100000010010	AND R18 R29 R29
00011011101111010000000000000110	SHR R29 R29 6
00010110010100100000000000011010	SHL R18 R18 26
00000010010111011010100000010011	OR R18 R29 R21
00110000000000000000000111111001	JMP 505
00010111101111010000000000000101	SHL R29 R29 5
00000010010111011110100000010010	AND R18 R29 R29

```
00011011101111010000000000000101    SHR R29 R29 5
00010110010100100000000000011011    SHL R18 R18 27
00000010010111011010100000010011    OR R18 R29 R21
00110000000000000000000111111001    JMP 505

00010111101111010000000000000100    SHL R29 R29 4
00000010010111011110100000010010    AND R18 R29 R29
00011011101111010000000000000100    SHR R29 R29 4
00010110010100100000000000011100    SHL R18 R18 28
00000010010111011010100000010011    OR R18 R29 R21
00110000000000000000000111111001    JMP 505

00010111101111010000000000000011    SHL R29 R29 3
00000010010111011110100000010010    AND R18 R29 R29
00011011101111010000000000000011    SHR R29 R29 3
00010110010100100000000000011101    SHL R18 R18 29
00000010010111011010100000010011    OR R18 R29 R21
00110000000000000000000111111001    JMP 505

00010111101111010000000000000010    SHL R29 R29 2
00000010010111011110100000010010    AND R18 R29 R29
00011011101111010000000000000010    SHR R29 R29 2
00010110010100100000000000011110    SHL R18 R18 30
00000010010111011010100000010011    OR R18 R29 R21
00110000000000000000000111111001    JMP 505

00010111101111010000000000000001    SHL R29 R29 1
00000010010111011110100000010010    AND R18 R29 R29
00011011101111010000000000000001    SHR R29 R29 1
00010110010100100000000000011111    SHL R18 R18 31
00000010010111011010100000010011    OR R18 R29 R21
00110000000000000000000111111001    JMP 505

00000000000100101010100000010000    ADD R0 R18 R21
00110000000000000000000111111001    JMP 505

00000100000111110000000000011100    ADDI R0 R31 28          #back
00101011110111111111111000100001    BEQ R30 R31 -479
00000100000111110000000001001111    ADDI R0 R31 79
00101011110111111111111001010010    BEQ R30 R31 -430
00000100000111110000000001011100    ADDI R0 R31 92
00101011110111111111111001011101    BEQ R30 R31 -419
00000100000111110000000001110101    ADDI R0 R31 117
00101011110111111111111001110100    BEQ R30 R31 -396
00000100000111110000000010000011    ADDI R0 R31 131
00101011110111111111111010000000    BEQ R30 R31 -384

11111100000000000000000000000000    HAL
```

5 FPGA Interface:

We have used all the 16 switches and 5 buttons of the FPGA board to implement RC5. Switches 0-7 are used as actual binary inputs for the machine. Switches 8-11 are used as multiplexing keys to provide 128 bits for ukey and 64 bits of din from the 8 input bits. These bits are saved to the data memory.
Clear signal is driven by 2 buttons where pressing btnc sets the clear and btnl resets the clock. Button btnl is also used to start the key expansion module. Button btnu starts of the encryption model while button btnd kicks off decryption.
The values of data memory can be viewed by setting switch 12 to 1 and providing the location using the lower 6 switches. The register file is checked by setting switch 13 to 1 and using the lower 6 switches as location pointer.

6 Design Verification:

We started out by building the decoder module first and generated multiple test instructions to check if the various modules operated as expected and are giving the correct results. We first checked the decoder module to see that it detects the correct opcode and generates the necessary control signals which would be used by rest of the modules to perform their operation. We then checked ALU module to see that it performs the correct operations based on the opcode and the functions bits and generates the desired output, after this we tested RF and Data Memory module to check that the correct locations are being accessed for reading and writing the data that are generated by the ALU module. We then checked the Multiplexer and Program Counter module to verify that the next PC is generated accurately depending upon the type of the instruction i.e. branch, jump or normal execution.
After this we executed both the sample codes provided to us and we saw that all the instructions were getting executed perfectly and also the required registers were updated accordingly and the final answer was as expected.

7 Demo Video: https://youtu.be/bmbj3KKWDpU

8 Github Link: https://github.com/yashhgandhi/NYU-6463_Processor_Design.git