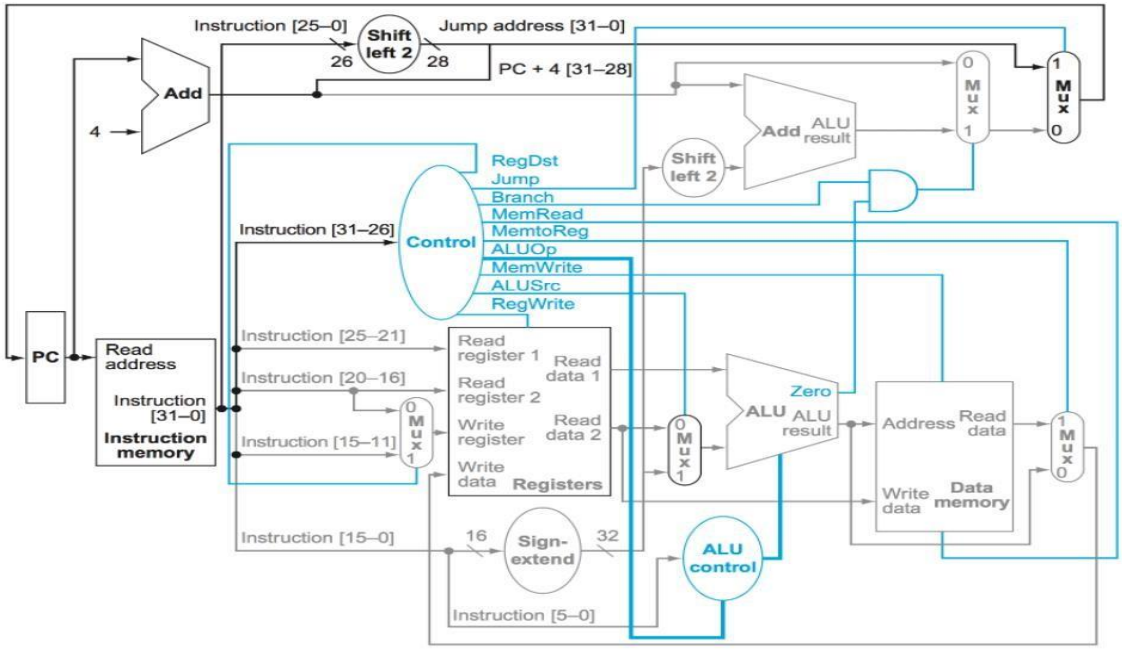
MIPS SINGLE CYCLE Processor
Hodgson Tetteh



The Final PC connection was made based on the figure above

addu	R-format instruction
addi/addiu	I-format instruction
beq/bne	I-format instruction
lw/sw	I-format instruction
j	J-format instruction

The instructions above were implemented on the single cycle mips processor And tested

Additional Components were added to the previous implementation to enable smooth operation.

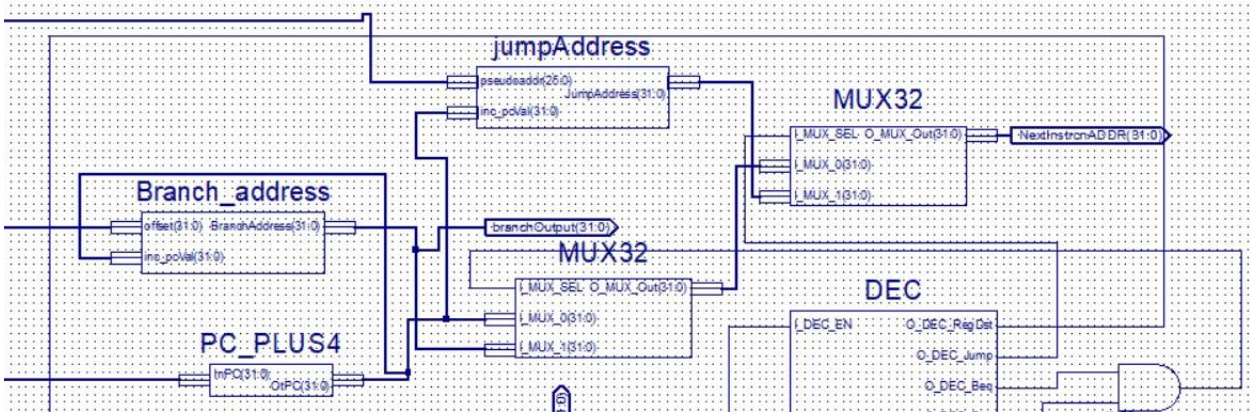
5 bit MUX

```
library ieee;
use ieee.std_logic_1164.all;


entity MUX5 is
    port(
         I_MUX_SEL : in std_logic;
        I_MUX_0, I_MUX_1: in std_logic_vector(4 downto 0);
        O_MUX_Out : out std_logic_vector(4 downto 0)
    );
end MUX5;

architecture Behavioral of MUX5 is
    begin
        process(I_MUX_0, I_MUX_1, I_MUX_SEL )
            begin
                if I_MUX_SEL = '0' then
                    O_MUX_Out<= I_MUX_0;
                elsif I_MUX_SEL = '1' then
                    O_MUX_Out <= I_MUX_1;
                end if;
        end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MUX32 is
port(
         I_MUX_SEL : in std_logic;
        I_MUX_0, I_MUX_1: in std_logic_vector(31 downto 0);
        O_MUX_Out : out std_logic_vector(31 downto 0)
    );
end MUX32;

architecture Behavioral of MUX32 is

begin
 process(I_MUX_0, I_MUX_1, I_MUX_SEL )
            begin
                if I_MUX_SEL = '0' then
                    O_MUX_Out<= I_MUX_0;
                elsif I_MUX_SEL = '1' then
                    O_MUX_Out <= I_MUX_1;
                end if;
        end process;
```

The 32 bit MUX figure 3.0

We also implemented a system of component determing branch and jumps



to test it I created an entity called CPUtest and tested the labsimulation of the CPU

Connections

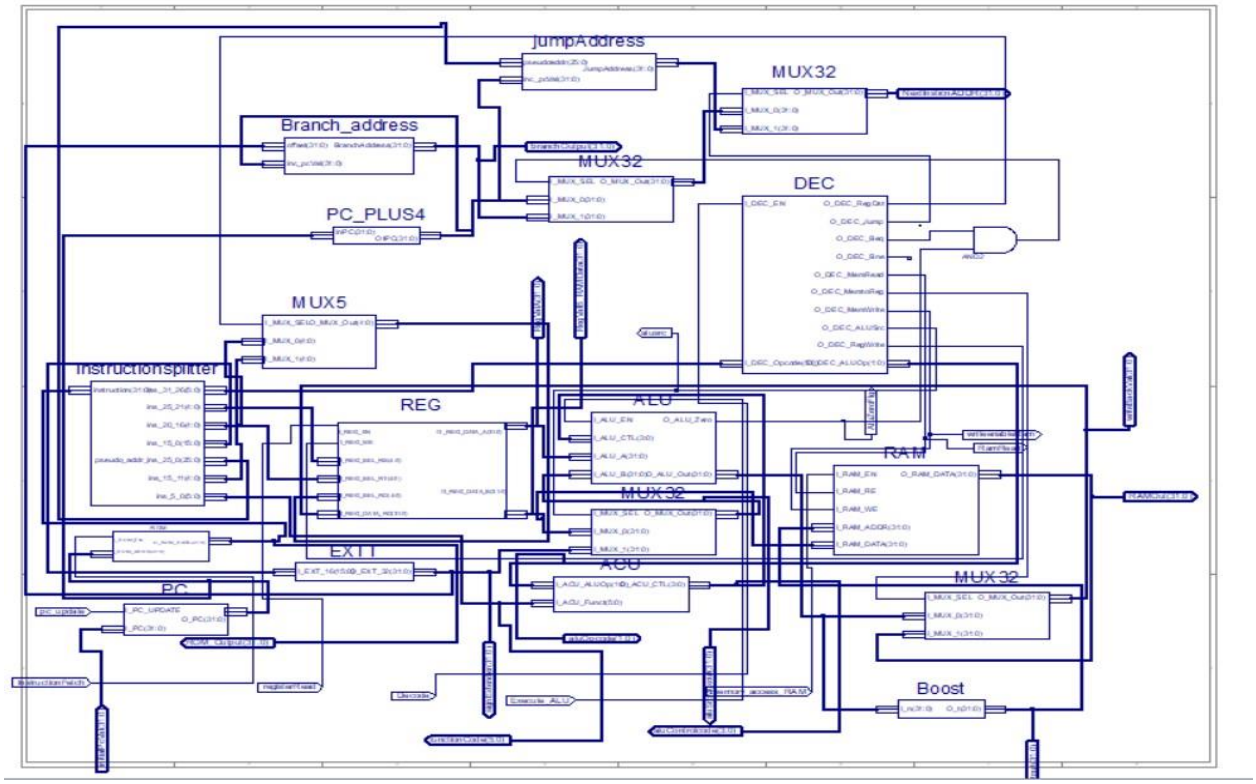


Figure 1

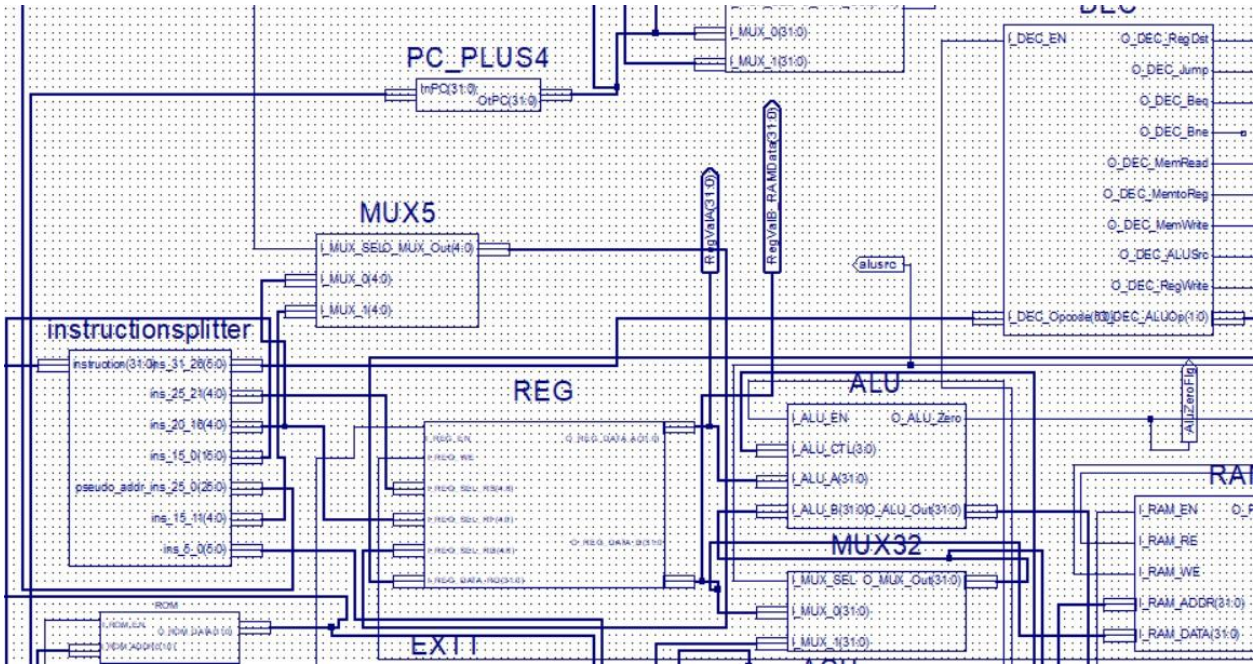


Figure 2 connections scoped in.

We implemented the finite state machine of sorts in our test bench

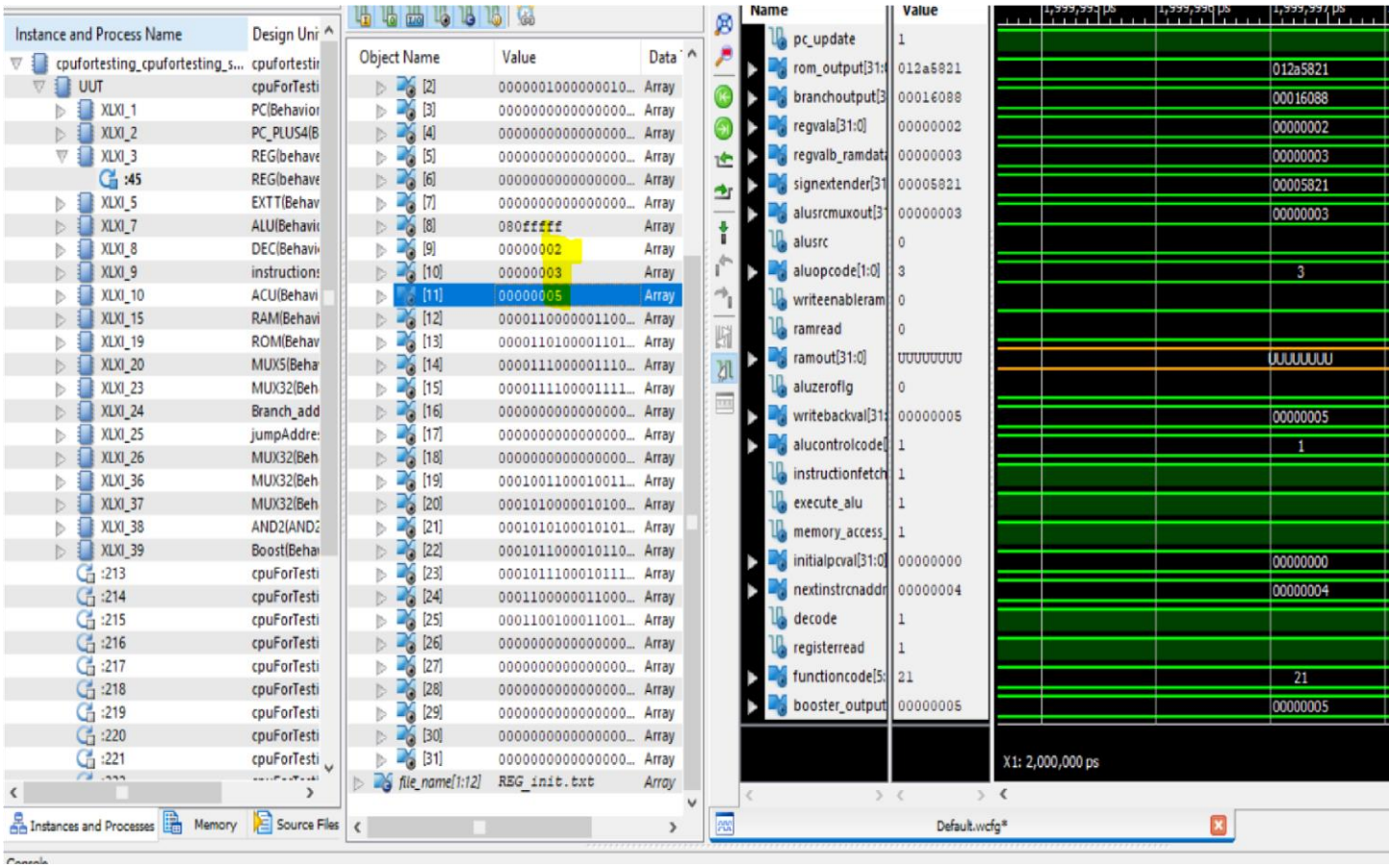
```
-- *** Test Bench - User Defined Section ***
 tb : PROCESS
  BEGIN
    WAIT for 10 ns; -- will wait forever
      initialPcVal <= X"00000008";
     pc_update <='1' ;
     WAIT for 10 ns;
       InstructionFetch<='1';
       WAIT for 10 ns;
       Decode<='1';
       registerRead<='1';
        WAIT for 10 ns;
        Execute_ALU<='1';

        memory_access_RAM<='1';

        WAIT for 30 ns;-- will wait forever -- will
        end process;
-- *** End Test Bench - User Defined Section ***
```

We initialize the IF part wait for a while then initialize the DC module part , then the EXECUTE module , then the MEMORY module and finally the write back.

Testing

TEST1;

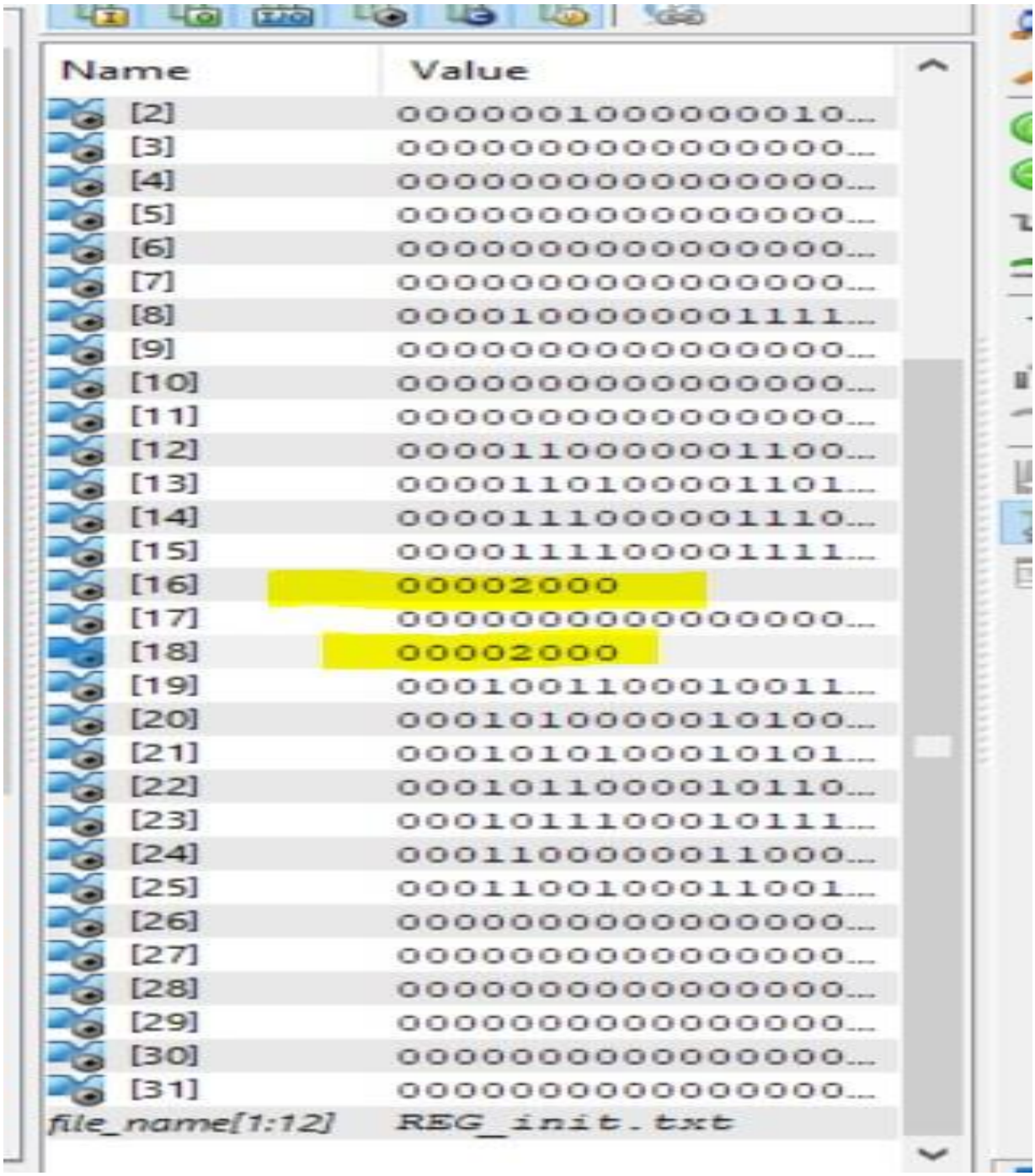
- **ADDU $11, $9,$10**
- **We put $9=2 , $10 = 3**
- **Opcode X”012a5821”**
- **Placed in instruction memory location X”00000000”**
- **……**



We can see register 11 being updated just as coded.

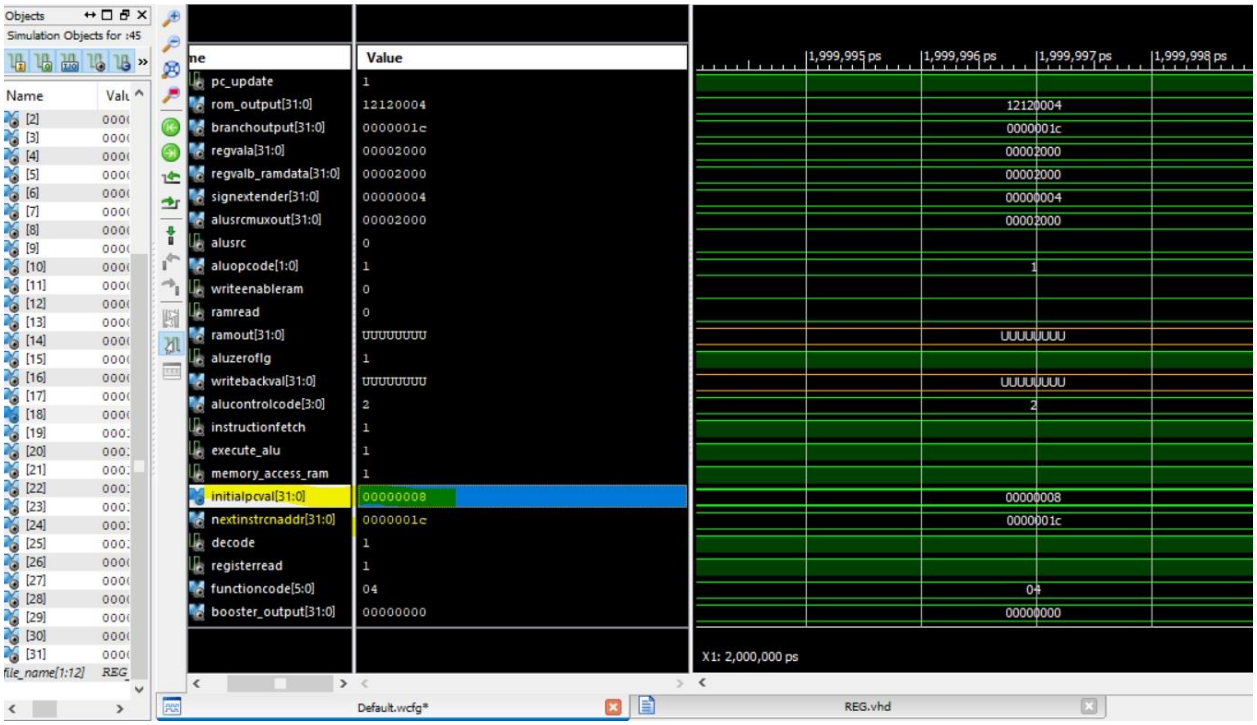
TEST 2

- **$16 has x2000**
- **$18 has x2000**
- **And we load instruction beq $16, $18 ,0x0004**
- **X”12120004”**
- **At location X”00000008”**



Name	Value
[2]	0000001000000010...
[3]	0000000000000000...
[4]	0000000000000000...
[5]	0000000000000000...
[6]	0000000000000000...
[7]	0000000000000000...
[8]	0000100000001111...
[9]	0000000000000000...
[10]	0000000000000000...
[11]	0000000000000000...
[12]	0000110000001100...
[13]	0000110100001101...
[14]	0000111000001110...
[15]	0000111100001111...
[16]	00002000
[17]	0000000000000000...
[18]	00002000
[19]	0001001100010011...
[20]	0001010000010100...
[21]	0001010100010101...
[22]	0001011000010110...
[23]	0001011100010111...
[24]	0001100000011000...
[25]	0001100100011001...
[26]	0000000000000000...
[27]	0000000000000000...
[28]	0000000000000000...
[29]	0000000000000000...
[30]	0000000000000000...
[31]	0000000000000000...
file_name[1:12]	REG_init.txt

We set the values of $16 and $18 equal as seen in the figure above

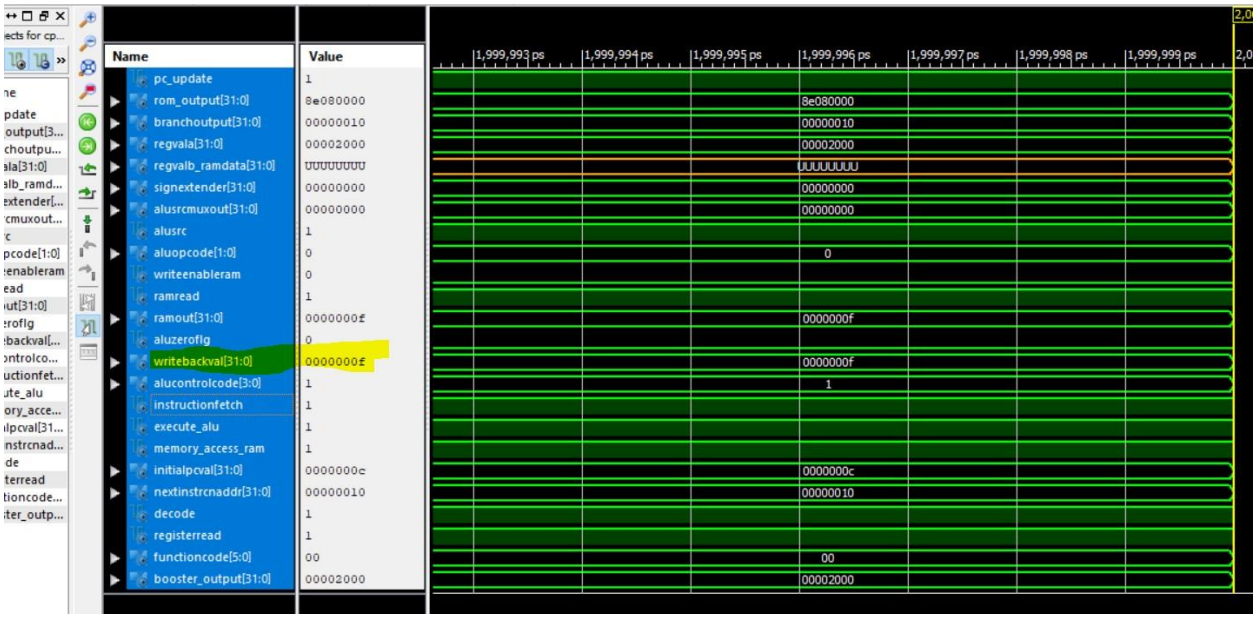


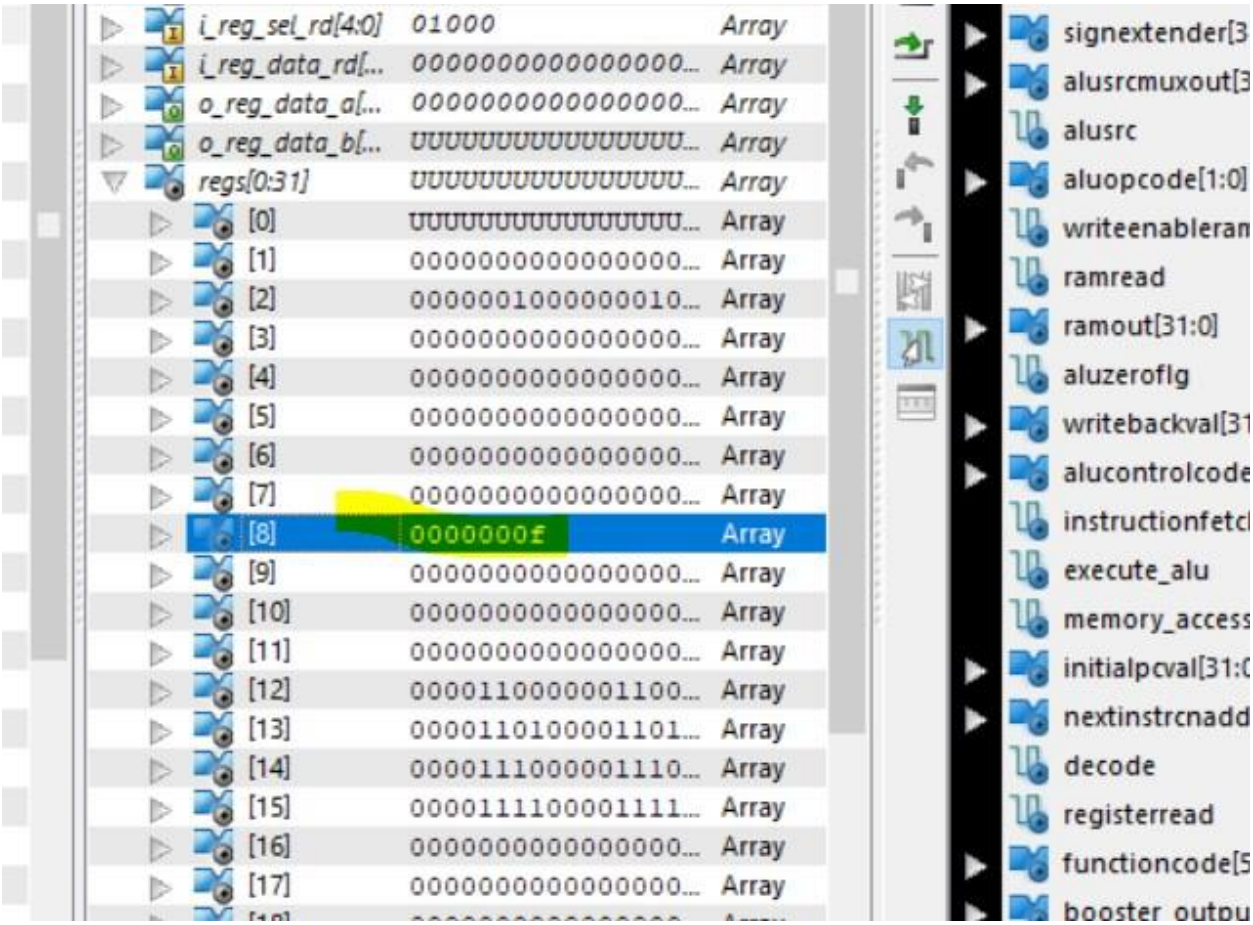
As we can see above the the next instruction changes from x00000008 to x0000001c in respect to the branch instruction

TEST 3

Now we will test lw and sw instructions

- **Lw $8 , 0x0000($16)**
- **$16 contains x”2000” which is a location in ram**
- **Mem[x2000] = b1111;**
- **Therefore we load b1111 into register $8;**
- **X”8e080000” placed in rom location x”0000000c”**


Name
Value
pc_update 1
rom_output[31:0] 8e080000
branchoutput[31:0] 00000010
regvala[31:0] 00002000
regvalb_ramdata[31:0] UUUUUUUU
signextender[31:0] 00000000
alusrcmuxout[31:0] 00000000
alusrc 1
aluopcode[1:0] 0
writeenableram 0
ramread 1
ramout[31:0] 0000000f
aluzeroflg 0
writebackval[31:0] 0000000f
alucontrolcode[3:0] 1
instructionfetch 1
execute_alu 1
memory_access_ram 1
initialpcval[31:0] 0000000c
nextinstrcnaddr[31:0] 00000010
decode 1
registerread 1
functioncode[5:0] 00
booster_output[31:0] 00002000
1,999,993 ps
1,999,994 ps
1,999,995 ps
1,999,996 ps
1,999,997 ps
1,999,998 ps
1,999,999 ps



We see the ram returning the x0000000f to the register 8 and after

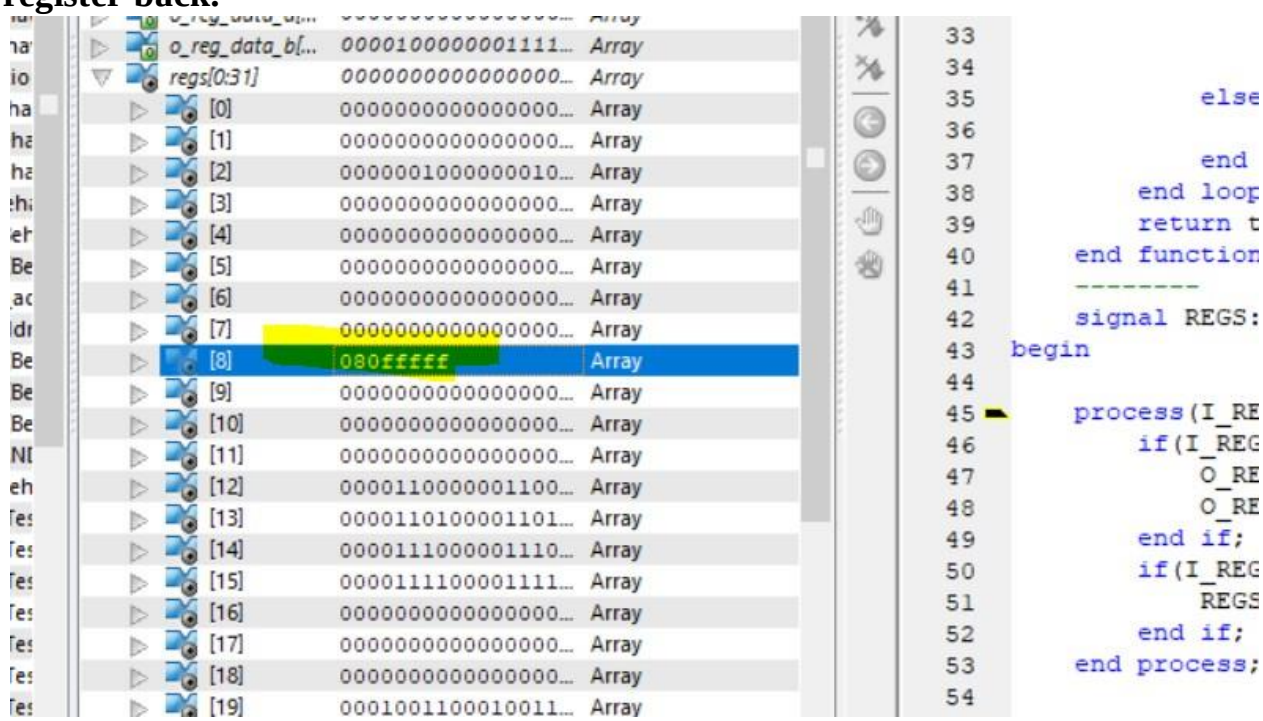
checking register $8 the value is correctly stored there hinting a

successful instruction executed.

TEST 4

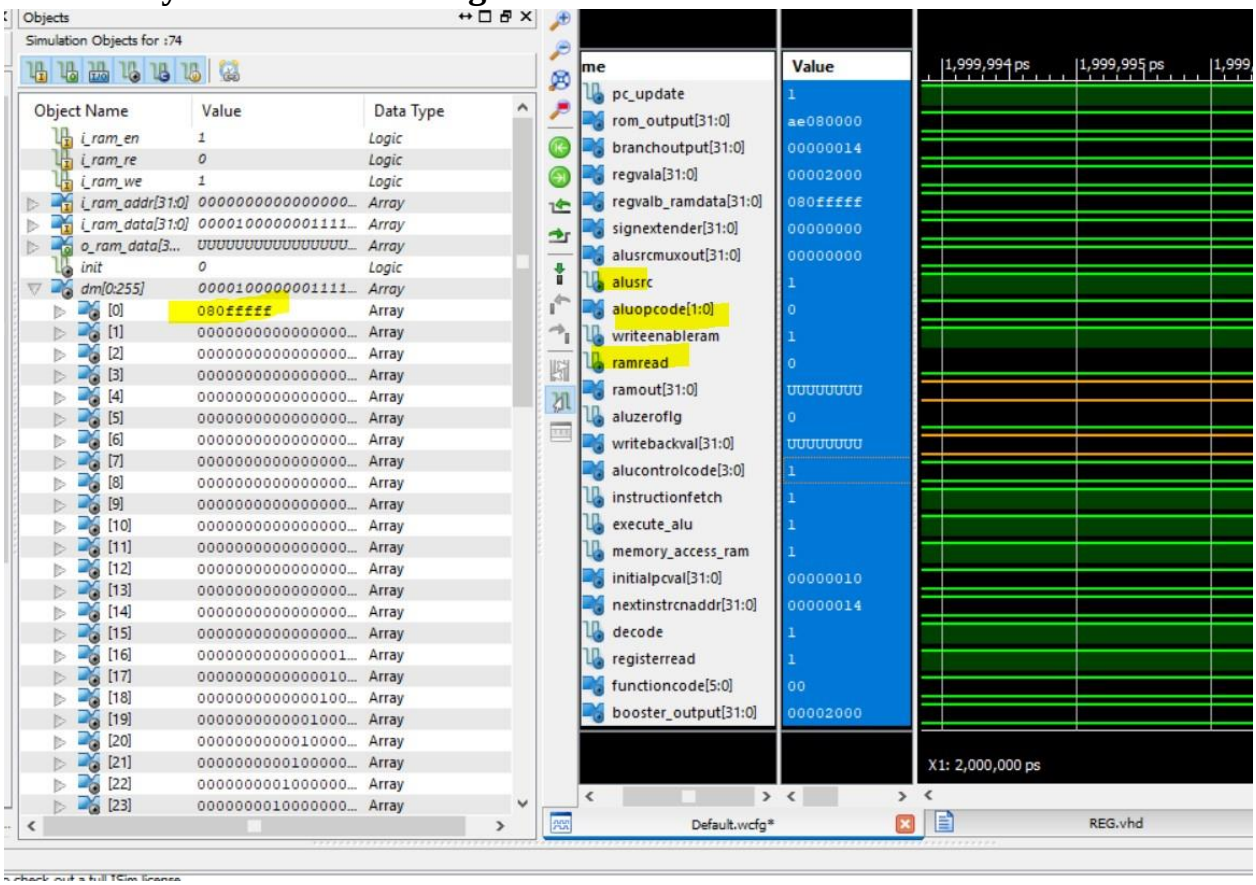
- **$8 has a value of x080fff**
- **We run the instruction in rom address x”00000010”**
- **Which corresponds to sw $8 ox0000($16)**

Therefore location x2000 in ram is updated to x080fff
To test this instruction we will store the value we previously got in the $8 register back.



Value stored in register [8];

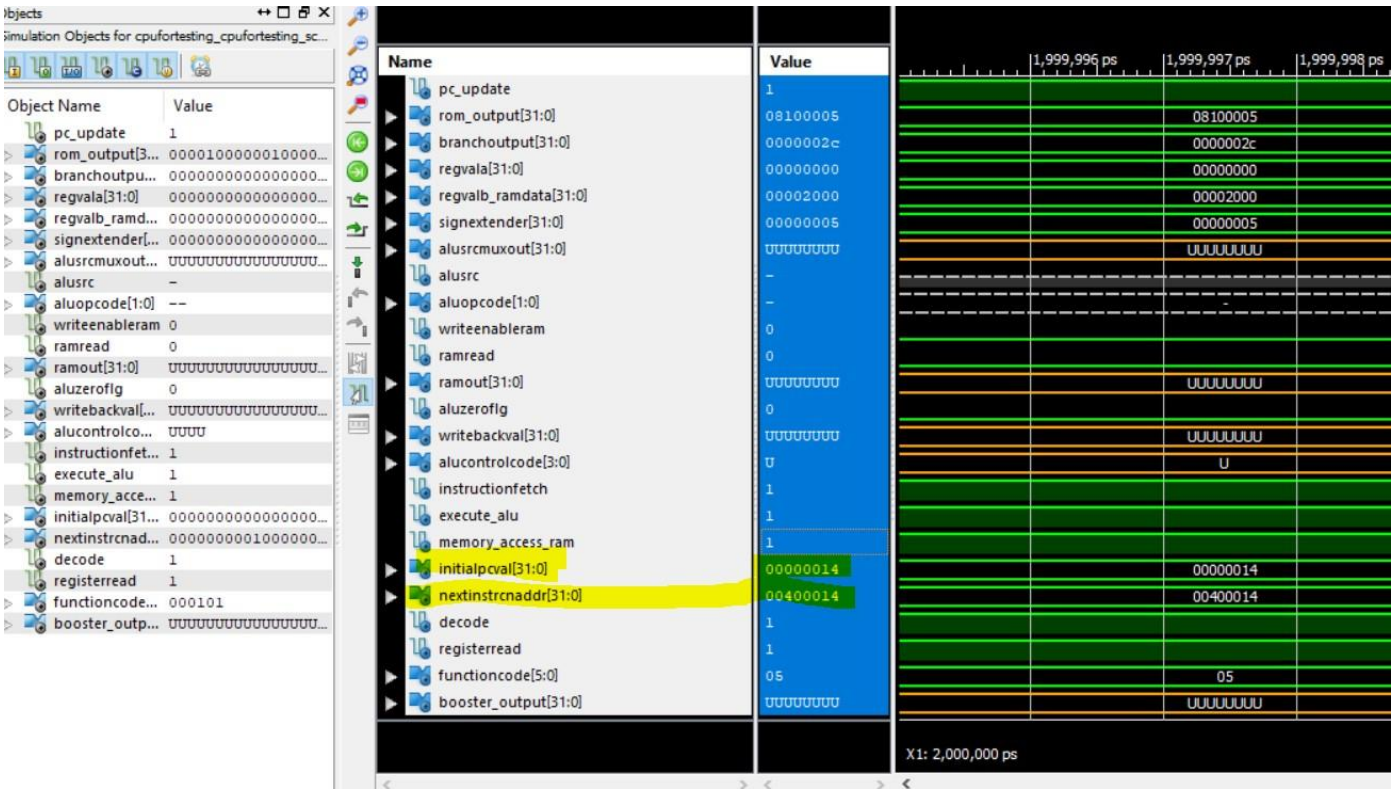
We also check the ram to make sure the instructions are stored it worked successfully as shown in the figure below.



And for our final test we tested the Jump instruction.

- **J 0x0040014**
- **Loaded in rom location 0x00000014**

Instruction x"08100005"



This instruction also runs smoothly and we can conclude that our processor can run all required instruction for this lab.

Conclusion, we faced a lot of hurdles connecting a lot if parts also considering that the final for the class was as the same day of the final presentation, we weren't able to add the FSM to control the modules hence the manual control with the test bench, nevertheless we got all the part fully functioning and runing