LAB 1: To find the 1’s complement and 2’s complement using C or C++.

(Source Code from Gopal Sir’s Lab Manual)

LAB 2: SIngned Magnitude multiplication

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

int q=0,b=0,c=0,e=0,q1=0,b1=0;

int qnum[4] = {0},bnum[4] = {0};

int acc[4] = {0}, res[4] = {0};

int sc=0,bs=0,qs=0,asize=0;

void binary()

{

  b1 = fabs(b);

  q1 = fabs(q);

  int r1, r2, i, temp;

  for (i = 0; i < 4; i++)

        {

    r1 = b1 % 2;

    b1 = b1 / 2;

    r2 = q1 % 2;

    q1 = q1 / 2;

    bnum[i] = r1;

    qnum[i] = r2;

  }

  sc=sizeof(qnum)/sizeof(int);
```

```
    asize=sc;
}

void add(int num[])
{
    int i;
    c = 0;
    for (i = 0; i < 4; i++){
           res[i] = acc[i] + num[i] + c;
           if (res[i] >= 2){
                c = 1;
           }
           else{
                c = 0;
           }
           e=c;
           res[i] = res[i]%2;
    }

    for (i = 3; i >= 0; i--){
        acc[i] = res[i];
        printf("%d",acc[i]);
     }
             printf(" : ");
```

```
            for (i = 3; i >= 0; i--)

            {

          printf("%d", qnum[i]);

    }

}

void rshift(){//for shift right

    int temp2 = acc[0], i;

    for (i = 1; i < 4 ; i++){//shift the MSB of product

       acc[i-1] = acc[i];

    }

    acc[3] = e;

    e=0;

    for (i = 1; i < 4 ; i++){//shift the LSB of product

        qnum[i-1] = qnum[i];

    }

    qnum[3] = temp2;


            printf("\n R-SHIFT: ");//display together

    for (i = 3; i >= 0; i--){

        printf("%d",acc[i]);

    }

    printf(":");

    for(i = 3; i >= 0; i--){

        printf("%d", qnum[i]);

    }
```

```
}
int main()
{
  int i;
  int p=0,n=1;
  printf("\t\tSIGNED MAGNITUDE MULTIPLICATION ALGORITHM");
  printf("\nEnter two numbers to multiply: ");
  printf("\nBoth must be less than 16");
  //simulating for two numbers each below 16
  do{
      printf("\nEnter b: ");
      scanf("%d",&b);
      printf("Enter Q: ");
      scanf("%d",&q);
    }while(b >=16 || q >=16);
   printf("\n Expected product = %d", b * q);
   binary();
   printf("\nS.C. = %d",sc);


          printf("\n\n Signed Binary Equivalents are: ");
   printf("\n b = ");
   for (i = 3; i >= 0; i--){
      printf("%d", bnum[i]);
   }
```

```
printf("\n q = ");
for (i = 3; i >= 0; i--){
   printf("%d", qnum[i]);
}
printf("\n\n");
                        while(sc!=0)
                        {
                                      printf("\nS.C. = %d",sc);
                                      if(qnum[0]==0){
          printf("\n-->");
          rshift();
          }
          else
                                      {
                        printf("\n-->");
          printf("\n ADD B: ");
          add(bnum);
          rshift();
          }
          sc--;
          }

printf("\nproduct is = ");

          if((b<0 && q>0)||(b>0 && q<0))
```

```
                        printf("%d",n);
            else
                        printf("%d",p);
    for (i = 3; i >= 0; i--){
          printf("%d", acc[i]);
    }
     for (i = 3; i >= 0; i--){
          printf("%d",qnum[i]);
     }
    return 0;
}
```

Output:

LAB 3: Booths Algorithm

```
#include <stdio.h>
#include <math.h>
int a = 0,b = 0, c = 0, a1 = 0, b1 = 0, com[5] = { 1, 0, 0, 0, 0};
int anum[5] = {0}, anumcp[5] = {0}, bnum[5] = {0};
int acomp[5] = {0}, bcomp[5] = {0}, pro[5] = {0}, res[5] = {0};
void binary(){
     a1 = fabs(a);
     b1 = fabs(b);
     int r, r2, i, temp;
     for (i = 0; i < 5; i++){
          r = a1 % 2;
```

```
        a1 = a1 / 2;

        r2 = b1 % 2;

        b1 = b1 / 2;

        anum[i] = r;

        anumcp[i] = r;

        bnum[i] = r2;

        if(r2 == 0){

            bcomp[i] = 1;

        }

        if(r == 0){

            acomp[i] =1;

        }

  }

//part for two's complementing

c = 0;

for ( i = 0; i < 5; i++){

        res[i] = com[i]+ bcomp[i] + c;

        if(res[i] >= 2){

            c = 1;

        }

        else

            c = 0;

        res[i] = res[i] % 2;

  }

for (i = 4; i >= 0; i--){
  bcomp[i] = res[i];
}
//in case of negative inputs
if (a < 0){
   c = 0;
  for (i = 4; i >= 0; i--){
        res[i] = 0;
  }
  for ( i = 0; i < 5; i++){
        res[i] = com[i] + acomp[i] + c;
        if (res[i] >= 2){
             c = 1;
        }
        else
             c = 0;
        res[i] = res[i]%2;
  }
  for (i = 4; i >= 0; i--){
        anum[i] = res[i];
        anumcp[i] = res[i];
  }


}
if(b < 0){
  for (i = 0; i < 5; i++){
```

```
            temp = bnum[i];

            bnum[i] = bcomp[i];

            bcomp[i] = temp;

      }

    }

}

void add(int num[]){

    int i;

    c = 0;

    for ( i = 0; i < 5; i++){

            res[i] = pro[i] + num[i] + c;

            if (res[i] >= 2){

                 c = 1;

            }

            else{

                 c = 0;

            }

            res[i] = res[i]%2;

      }

      for (i = 4; i >= 0; i--){

          pro[i] = res[i];

          printf("%d",pro[i]);

      }

    printf(":");

    for (i = 4; i >= 0; i--){
          printf("%d", anumcp[i]);
     }
}
void arshift(){//for arithmetic shift right
    int temp = pro[4], temp2 = pro[0], i;
    for (i = 1; i < 5  ; i++){//shift the MSB of product
       pro[i-1] = pro[i];
    }
    pro[4] = temp;
    for (i = 1; i < 5  ; i++){//shift the LSB of product
        anumcp[i-1] = anumcp[i];
    }
    anumcp[4] = temp2;
    printf("\nAR-SHIFT: ");//display together
    for (i = 4; i >= 0; i--){
        printf("%d",pro[i]);
    }
    printf(":");
    for(i = 4; i >= 0; i--){
        printf("%d", anumcp[i]);
    }
}

void main(){
   int i, q = 0;
```

```
printf("\t\tBOOTH'S MULTIPLICATION ALGORITHM");

printf("\nEnter two numbers to multiply: ");

printf("\nBoth must be less than 16");

//simulating for two numbers each below 16

do{

    printf("\nEnter A: ");

    scanf("%d",&a);

    printf("Enter B: ");

    scanf("%d", &b);

 }while(a >=16 || b >=16);


 printf("\nExpected product = %d", a * b);

 binary();

 printf("\n\nBinary Equivalents are: ");

 printf("\nA = ");

 for (i = 4; i >= 0; i--){

    printf("%d", anum[i]);

 }

 printf("\nB = ");

 for (i = 4; i >= 0; i--){

    printf("%d", bnum[i]);

 }

 printf("\nB'+ 1 = ");

 for (i = 4; i >= 0; i--){

    printf("%d", bcomp[i]);
}

printf("\n\n");

for (i = 0;i < 5; i++){

      if (anum[i] == q){//just shift for 00 or 11

          printf("\n-->");

          arshift();

          q = anum[i];

      }

      else if(anum[i] == 1 && q == 0){//subtract and shift for 10

         printf("\n-->");

         printf("\nSUB B: ");

         add(bcomp);//add two's complement to implement subtraction

         arshift();

         q = anum[i];

      }

      else{//add ans shift for 01

         printf("\n-->");

         printf("\nADD B: ");

         add(bnum);

         arshift();

         q = anum[i];

      }

 }


 printf("\nProduct is = ");
```

```
    for (i = 4; i >= 0; i--){
        printf("%d", pro[i]);
    }
    for (i = 4; i >= 0; i--){
        printf("%d", anumcp[i]);
    }
}
```

Output:

LAB 4: Restoring Division

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int a=0,b=0,c=0,com[5]={1,0,0,0,0},s=0;
int anum[5]={0},anumcp[5] ={0},bnum[5]={0};
int acomp[5]={0},bcomp[5]={0},rem[5]={0},quo[5]={0},res[5]={0};
void binary(){
    a = fabs(a);
    b = fabs(b);
    int r, r2, i, temp;
    for(i = 0; i < 5; i++){
        r = a % 2;
        a = a / 2;
        r2 = b % 2;
        b = b / 2;
```

```
            anum[i] = r;

            anumcp[i] = r;

            bnum[i] = r2;

            if(r2 == 0){

                bcomp[i] = 1;

            }

            if(r == 0){

                acomp[i] =1;

            }

       }

    //part for two's complementing

    c = 0;

    for( i = 0; i < 5; i++){

            res[i] = com[i]+ bcomp[i] + c;

            if(res[i]>=2){

                c = 1;

            }

            else

                c = 0;

            res[i] = res[i]%2;

       }

    for(i = 4; i>= 0; i--){

      bcomp[i] = res[i];

    }

}
```

```
void add(int num[]){
    int i;
    c = 0;
    for( i = 0; i < 5; i++){
         res[i] = rem[i]+ num[i] + c;
         if(res[i]>=2){
             c = 1;
         }
         else
             c = 0;
         res[i] = res[i]%2;
    }
    for(i = 4; i>= 0; i--){
         rem[i] = res[i];
         printf("%d",rem[i]);
    }
    printf(":");
    for(i = 4; i>= 0; i--){
         printf("%d",anumcp[i]);
    }
}
void shl(){//for shift left
    int i;
    for(i = 4; i > 0  ; i--){//shift the remainder
         rem[i] = rem[i-1];
    }
    rem[0] = anumcp[4];
    for(i = 4; i > 0  ; i--){//shift the remtient
        anumcp[i] = anumcp[i-1];
    }
    anumcp[0] = 0;
    printf("\nSHIFT LEFT: ");//display together
    for(i = 4; i>= 0; i--){
        printf("%d",rem[i]);
    }
    printf(":");
    for(i = 4; i>= 0; i--){
        printf("%d",anumcp[i]);
    }
}

void main(){
    int i;
    printf("\t\tRESTORING DIVISION ALGORITHM");
    printf("\nEnter two numbers to multiply: ");
    printf("\nBoth must be less than 16");
    //simulating for two numbers each below 16
    do{
        printf("\nEnter A: ");
        scanf("%d",&a);
        printf("Enter B: ");

        scanf("%d",&b);

    }while(a>=16 || b>=16);

    printf("\nExpected Quotient = %d", a/b);

    printf("\nExpected Remainder = %d", a%b);

    if(a*b <0){

        s = 1;

    }

    binary();

    printf("\n\nUnsigned Binary Equivalents are: ");

    printf("\nA = ");

    for(i = 4; i>= 0; i--){

        printf("%d",anum[i]);

    }

    printf("\nB = ");

    for(i = 4; i>= 0; i--){

        printf("%d",bnum[i]);

    }

    printf("\nB'+ 1 = ");

    for(i = 4; i>= 0; i--){

        printf("%d",bcomp[i]);

    }

    printf("\n\n-->");

    //division part

    shl();
```

```
for(i=0;i<5;i++){
    printf("\n-->"); //start with subtraction
    printf("\nSUB B: ");
    add(bcomp);
    if(rem[4]==1){//simply add for restoring
        printf("\n-->RESTORE");
        printf("\nADD B: ");
        anumcp[0] = 0;
        add(bnum);
    }
    else{
        anumcp[0] = 1;
    }
    if(i<4)
        shl();

}
printf("\n----------------------------");
printf("\nSign of the result = %d",s);
printf("\nRemainder is = ");
for(i = 4; i>= 0; i--){
    printf("%d",rem[i]);
}
printf("\nQuotient is = ");
for(i = 4; i>= 0; i--){
        printf("%d",anumcp[i]);
    }
getch();
}
```

OUTPUT

LAB 5: Non-Restoring Algorithm

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10],m[10],q[10],i,j,c=0,b[10],mc[10],z=0,s[10],x[10];
clrscr();
printf("\nEnter Divisor [M] : ");
for(i=4;i>=1;i--)
{
scanf("%d",&m[i]);
mc[i]=!m[i];
a[i]=0, x[i]=0;
}
x[1]=1, x[5]=0, m[5]=0, mc[5]=1;
for(i=1;i<=5;i++)
{
s[i]=x[i]^mc[i]^z;
c=(x[i]&&mc[i])||(x[i]&&z)||(mc[i]&&z);
z=c;
mc[i]=s[i];
}
printf("\n\nEnter Divident [Q] : ");
for(i=4;i>=1;i--)
{
scanf("%d",&q[i]);
}
printf("\n\n\n Step \t\t Action Performed \t\t A \t\tQ\n");
printf("\n\n 0\t\t Initialization\t 0 0 0 0 0 \t ");
for(i=4;i>=1;i--)
{
printf(" %d",q[i]);
}
for(j=1;j<=4;j++)
{
```

```
printf("\n\n\n %d",j);
printf("\t\t Left Shift \t\t ");
for(i=5;i>=2;i--)
{
a[i]=a[i-1];
}
a[1]=q[4];
for(i=4;i>=2;i--)
{
q[i]=q[i-1];
}
for(i=5;i>=1;i--)
{
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=2;i--)
{
printf(" %d",q[i]);
}
if(a[5]==0)
{
z=0;
for(i=1;i<=5;i++)
{
s[i]=a[i]^mc[i]^z;
c=(a[i]&&mc[i])||(a[i]&&z)||(mc[i]&&z);
z=c;
a[i]=s[i];
}
if(a[5]==1)
{
q[1]=0;
}
else
{
q[1]=1;
}
printf("\n\n\t\t a = a-m\t\t ");
for(i=5;i>=1;i--)
{
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=1;i--)
{
printf(" %d",q[i]);
}
}
else
{
z=0;
for(i=1;i<=5;i++)
{
s[i]=a[i]^m[i]^z;
c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z);
z=c;
a[i]=s[i];
}
if(a[5]==1)
{
q[1]=0;
}
else
{
q[1]=1;
}
printf("\n\n\t\t a = a+m\t\t ");
for(i=5;i>=1;i--)
{
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=1;i--)
{
printf(" %d",q[i]);
}
}
}
if(a[5]==1)
{
printf("\n\n\n 5");
for(i=1;i<=5;i++)
{
s[i]=a[i]^m[i]^z;
c=(a[i]&&m[i])||(a[i]&&z)||(m[i]&&z);
z=c;
a[i]=s[i];
}
printf("\t\t a = a+m\t\t ");
for(i=5;i>=1;i--)
{
printf(" %d",a[i]);
}
printf("\t ");
for(i=4;i>=1;i--)
```

```
{
printf(" %d",q[i]);
}
}
printf("\n\n\n\n\nQuotient [Q] :");
for(i=4;i>=1;i--)
{
printf(" %d",q[i]);
}
printf("\n\n\n\nRemainder [A] :");
for(i=4;i>=1;i--)
{
printf(" %d",a[i]);
}
getch();
}
```

OUTPUT

LAB 6: Realization of AND Gate using VHDL

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

entity and_gate is
  port (
    input_1    : in  std_logic;
    input_2    : in  std_logic;
    and_result : out std_logic
    );
end and_gate;

architecture rtl of and_gate is
begin
  and_result <= input_1 and input_2;
end rtl;
```

LAB 7: Full Adder using VHDL

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

entity full_adder is
```

```
  port (
    input_1    : in  std_logic;
    input_2    : in  std_logic;
    input_3    : in  std_logic;
    sum        : out std_logic;
    carry      : out std_logic
    );
end full_adder;

architecture rtl of full_adder is
begin
  carry  <= (input_1 and input_2)or (input_2 and input_3) or (input_3 and input_1);
  sum  <= ((input_1 xor input_2) xor input_3);
end rtl;
```

LAB 8: ALU Implementation

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

use ieee.std_logic_unsigned.all;



entity alu_d is

 Port ( inp_a : in signed(3 downto 0);

 inp_b : in signed(3 downto 0);

 sel : in STD_LOGIC_VECTOR (2 downto 0);

 out_alu : out signed(3 downto 0));

end alu_d;



architecture Behavioral of alu_d is

begin

process(inp_a, inp_b, sel)
```

```
begin
case sel is
 when "000" =>
 out_alu<= inp_a + inp_b; --addition
 when "001" =>
 out_alu<= inp_a - inp_b; --subtraction
 when "010" =>
 out_alu<= inp_a - 1; --sub 1
 when "011" =>
 out_alu<= inp_a + 1; --add 1
 when "100" =>
 out_alu<= inp_a and inp_b; --AND gate
 when "101" =>
 out_alu<= inp_a or inp_b; --OR gate
 when "110" =>
 out_alu<= not inp_a ; --NOT gate
 when "111" =>
 out_alu<= inp_a xor inp_b; --XOR gate
 when others =>
 NULL;
end case;
end process;
end Behavioral;
```

LAB 9: MULTIPLEXER : 4 to 1

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity mux_4to1 is

 port(

     A,B,C,D : in STD_LOGIC;

     S0,S1: in STD_LOGIC;

     Z: out STD_LOGIC

  );

end mux_4to1;

architecture bhv of mux_4to1 is

begin

process (A,B,C,D,S0,S1) is

begin

  if (S0 ='0' and S1 = '0') then

      Z <= A;

  elsif (S0 ='1' and S1 = '0') then

      Z <= B;

  elsif (S0 ='0' and S1 = '1') then

      Z <= C;

  else

      Z <= D;

  end if;

end process;

end bhv;
```

LAB 10: ENCODER 8 TO 3

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity encoder8_3 is

    port(

        din : in STD_LOGIC_VECTOR(7 downto 0);

        dout : out STD_LOGIC_VECTOR(2 downto 0)

        );

end encoder8_3;

architecture encoder8_3_arc of encoder8_3 is

begin

    dout <= "000" when (din="10000000") else

            "001" when (din="01000000") else

            "010" when (din="00100000") else

            "011" when (din="00010000") else

            "100" when (din="00001000") else

            "101" when (din="00000100") else

            "110" when (din="00000010") else

            "111";

end encoder8_3_arc;
```

LAB 11: Partity Generator using VHDL

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity paritygen is

  port (a: in STD_LOGIC_VECTOR (2 downto 0);

        even : out STD_LOGIC;
        odd : out STD_LOGIC);
end paritygen;
architecture paritygen_arch of paritygen is
  signal i:std_logic;
begin
  i<= a(0) xor a(1) xor a(2);
  even <=i;
  odd <= not i;
end paritygen_arch;
```

LAB 12: Parity Checker using VHDL

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

entity Four_bit_even_parity_checker is
    Port ( x,y,z,p : in  STD_LOGIC;
           C : out  STD_LOGIC);
end Four_bit_even_parity_checker;

architecture Behavioral of Four_bit_even_parity_checker is

begin

C <= ((x xor y) xor (z xor p));

end Behavioral;
```

LAB 13: Shifter Implementation

The input/ output interface of the shifter is shown in the above figure. The shifter instruction set is as follows:

- SHIFT_Ctrl = "1000": SHIFTOUT <= Rotate SHIFTINPUT >>8
- SHIFT_Ctrl = "1001": SHIFTOUT <= Rotate SHIFTINPUT >>4

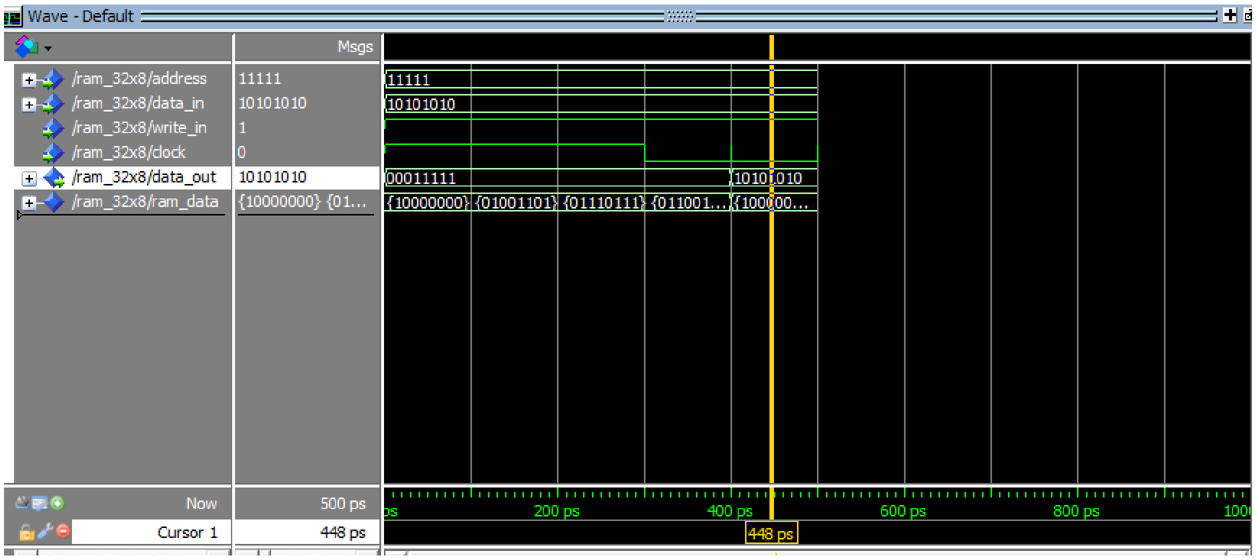
- SHIFT_Ctrl = "1010": SHIFTOUT <= Shift Left Logical SHIFTINPUT << 8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shifter is
    Port ( SHIFTINPUT : in  STD_LOGIC_VECTOR(15 downto 0);
   SHIFT_Ctrl : in  STD_LOGIC_VECTOR(3 downto 0);
   SHIFTOUT: out  STD_LOGIC_VECTOR(15 downto 0)
  );
end shifter;

architecture Behavioral of shifter is

begin
process(SHIFTINPUT,SHIFT_Ctrl)
begin
case(SHIFT_Ctrl) is
when "1000" => SHIFTOUT <= SHIFTINPUT(7 downto 0)&SHIFTINPUT(15 downto 8);
when "1001" => SHIFTOUT <= SHIFTINPUT(3 downto 0)&SHIFTINPUT(15 downto 4);
when "1010" => SHIFTOUT <= SHIFTINPUT(7 downto 0) & "00000000";
when others => SHIFTOUT <= "00000000";
end case;
end process;

end Behavioral;
```

LAB 14: RAM Implementation

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

USE ieee.numeric_std.ALL;

entity RAM_32X8 is

port(

 address: in std_logic_vector(4 downto 0);

 data_in: in std_logic_vector(7 downto 0);

 write_in: in std_logic;
 clock: in std_logic;

 data_out: out std_logic_vector(7 downto 0)

);

end RAM_32X8;

architecture Behav of RAM_32X8 is

type ram_array is array (0 to 31 ) of std_logic_vector (7 downto 0);

signal ram_data: ram_array :=(

   b"10000000",b"01001101",x"77",x"67",

   x"99",x"25",x"00",x"1A",

   x"00",x"00",x"00",x"00",

   x"00",x"00",x"00",x"00",

   x"00",x"0F",x"00",x"00",

   x"00",x"00",b"00111100",x"00",

   x"00",x"00",x"00",x"00",

   x"00",x"00",x"00",x"1F"

   );

begin

process(clock)

begin

 if(rising_edge(clock)) then

 if(write_in='1') then

 ram_data(to_integer(unsigned(address))) <= data_in;

  end if;

 end if;

end process;
```

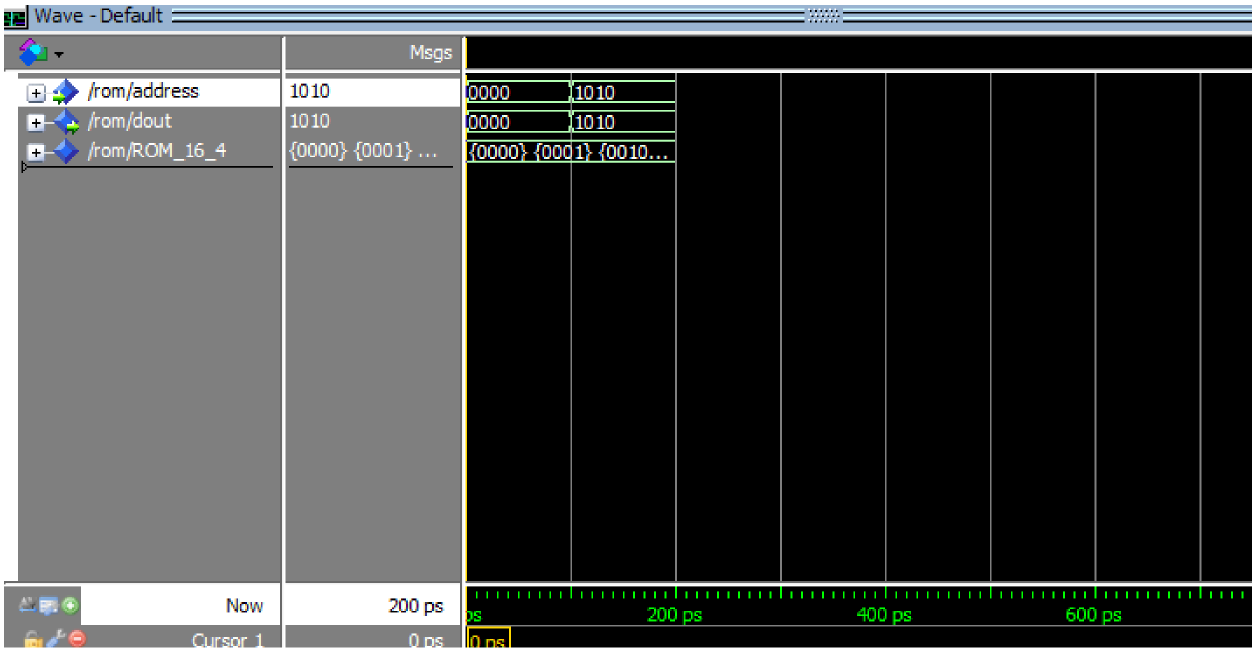
```
 data_out <= ram_data(to_integer(unsigned(address)));

end Behav;
```



LAB 15: ROM Implementation

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;


entity ROM is

   port(

      address : in  std_logic_vector(3 downto 0);

      dout    : out std_logic_vector(3 downto 0)

   );

end entity ROM;


architecture RTL of ROM is

   type MEMORY_16_4 is array (0 to 15) of std_logic_vector(3 downto 0);
```

```
    constant ROM_16_4 : MEMORY_16_4 := (
        x"0",
        x"1",
        x"2",
        x"3",
        x"4",
        x"5",
        x"6",
        x"7",
        x"8",
        x"9",
        x"a",
        x"b",
        x"c",
        x"d",
        x"e",
        x"f"
    );
begin
    main : process(address)
    begin
        dout <= ROM_16_4(to_integer(unsigned(address)));
    end process main;


end architecture RTL;
```



Signal	Msgs	Wave
/rom/address	1010	0000 / 1010
/rom/dout	1010	0000 / 1010
/rom/ROM_16_4	{0000} {0001} ...	{0000} {0001} {0010...

```
Now       200 ps
Cursor 1  0 ps
```