Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Advanced digital design

ENCS 3310

Project

Prepared by:

Mohammad Khdour

1212517

Instructor:

Elias Khalil

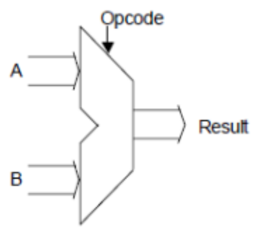
Section: 1

Data: 20/1/2023

1. Part one:

1.1 ALU

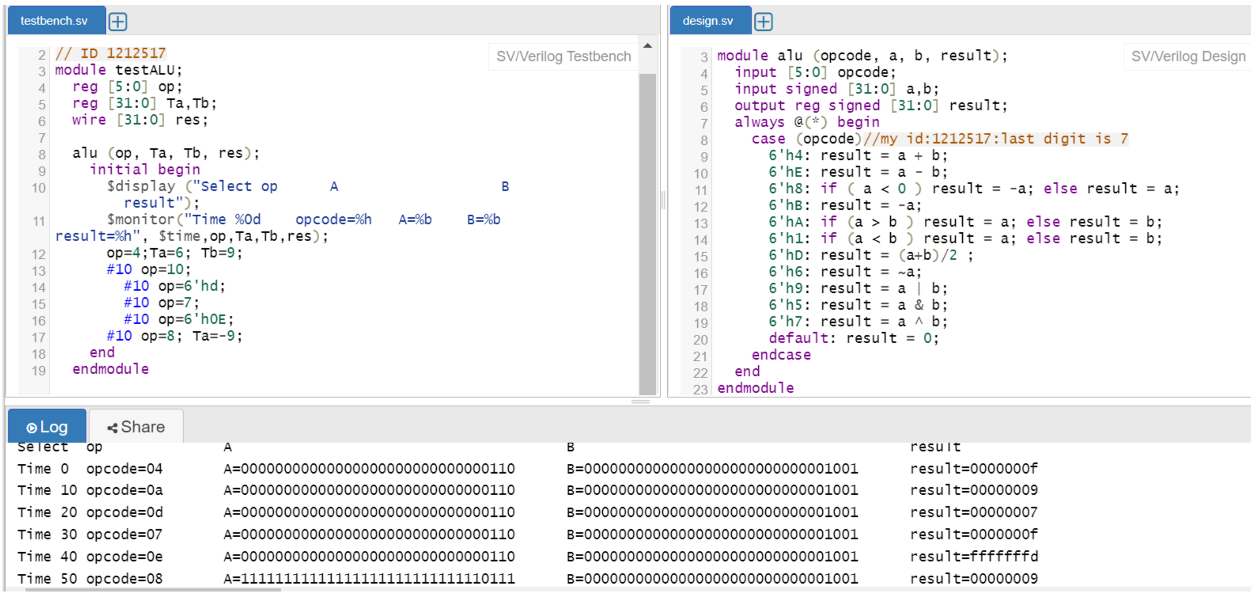
We will design ALU with two 32-bit input for a, and b, 6-bit input for opcode, and one 32-bit output. The opcode that will be used to represent each operation is determined by the last digit of students ID number.



My id is 1212517 and the last digit is 7, so we will use opcodes from the table below.

Digit of ID no.	7
a + b	4
a – b	14
|a|	8
-a	11
max(a,b)	10
min (a,b)	1
avg(a,b)	13
not a	6
a or b	9
a and b	5
a xor b	7

1.1.1 code & Test



testbench.sv — SV/Verilog Testbench

```
// ID 1212517
module testALU;
  reg [5:0] op;
  reg [31:0] Ta,Tb;
  wire [31:0] res;

  alu (op, Ta, Tb, res);
    initial begin
      $display ("Select op       A                   B
        result");
      $monitor("Time %0d    opcode=%h    A=%b     B=%b
result=%h", $time,op,Ta,Tb,res);
      op=4;Ta=6; Tb=9;
      #10 op=10;
        #10 op=6'hd;
        #10 op=7;
        #10 op=6'h0E;
      #10 op=8; Ta=-9;
    end
  endmodule
```

design.sv — SV/Verilog Design

```
module alu (opcode, a, b, result);
  input [5:0] opcode;
  input signed [31:0] a,b;
  output reg signed [31:0] result;
  always @(*) begin
    case (opcode)//my id:1212517:last digit is 7
      6'h4: result = a + b;
      6'hE: result = a - b;
      6'h8: if ( a < 0 ) result = -a; else result = a;
      6'hB: result = -a;
      6'hA: if (a > b ) result = a; else result = b;
      6'h1: if (a < b ) result = a; else result = b;
      6'hD: result = (a+b)/2 ;
      6'h6: result = ~a;
      6'h9: result = a | b;
      6'h5: result = a & b;
      6'h7: result = a ^ b;
      default: result = 0;
    endcase
  end
endmodule
```

Log

```
Select  op           A                                   B                                   result
Time 0  opcode=04    A=00000000000000000000000000000110  B=00000000000000000000000000001001  result=0000000f
Time 10 opcode=0a    A=00000000000000000000000000000110  B=00000000000000000000000000001001  result=00000009
Time 20 opcode=0d    A=00000000000000000000000000000110  B=00000000000000000000000000001001  result=00000007
Time 30 opcode=07    A=00000000000000000000000000000110  B=00000000000000000000000000001001  result=0000000f
Time 40 opcode=0e    A=00000000000000000000000000000110  B=00000000000000000000000000001001  result=fffffffd
Time 50 opcode=08    A=11111111111111111111111111110111  B=00000000000000000000000000001001  result=00000009
```

For alu design we use behavioral description, and followed by case statement. Using signed input and output to dial with both number positive and negative. If opcode not exist than result assigned by zero.

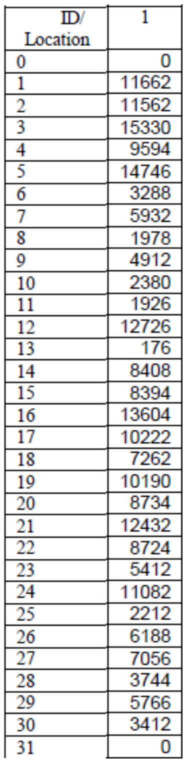
Testbench have one output which is res, and three input defined as reg, first one for opcode, and the other for a, and b, after that we use **$monitor** to display opcode, a, b, and result when one of them change.

1.1.2 Results

- At time zero, the opcode equal 0x04 which make alu do addition operation, between a=6 and b=9 where their sum equal 15 in decimal and (f) in hexadecimal.
- At time 10, opcode equal 0x0a which make alu return the greater number between a=6 and b=9, The result should equal 9.
- At time 50, opcode equal 0x08 which make alu return the absolute value for a, which a equal to (-9), the result should equal 9.

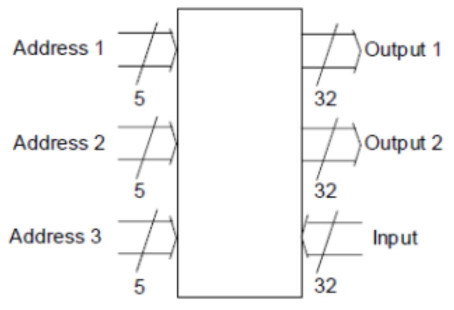
1.2 Register file

We initialize the register by the values given in the table below using values in location 1, that depend on the second digit of last from my id which equal 12125**1**7.

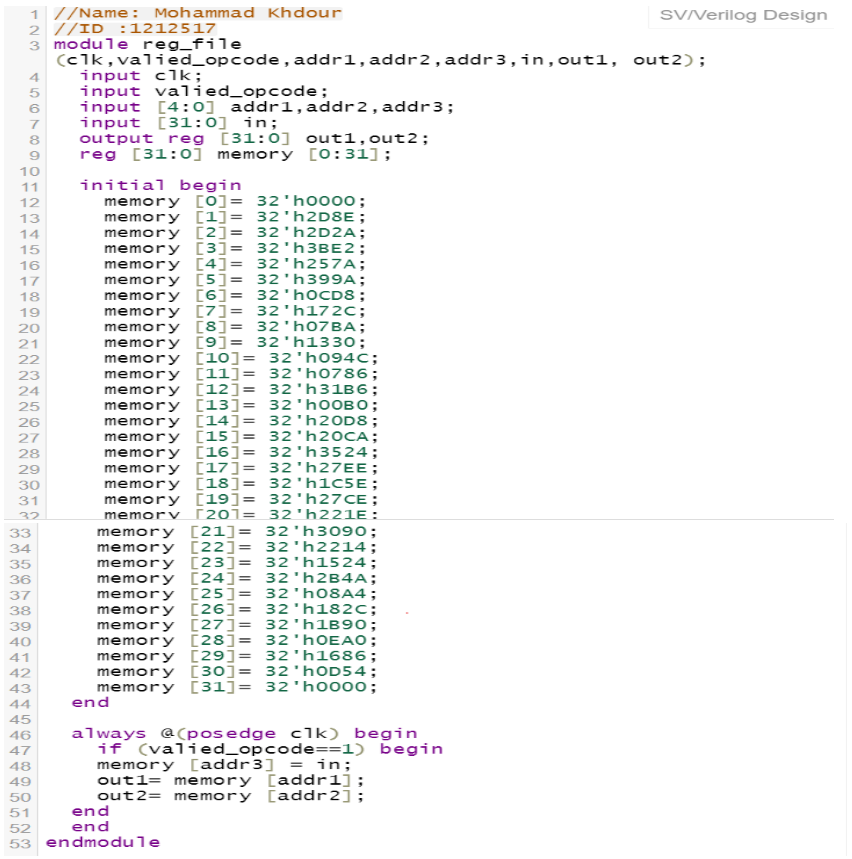


ID/ Location	1
0	0
1	11662
2	11562
3	15330
4	9594
5	14746
6	3288
7	5932
8	1978
9	4912
10	2380
11	1926
12	12726
13	176
14	8408
15	8394
16	13604
17	10222
18	7262
19	10190
20	8734
21	12432
22	8724
23	5412
24	11082
25	2212
26	6188
27	7056
28	3744
29	5766
30	3412
31	0

Reg_file module:



1.2.1 Code:

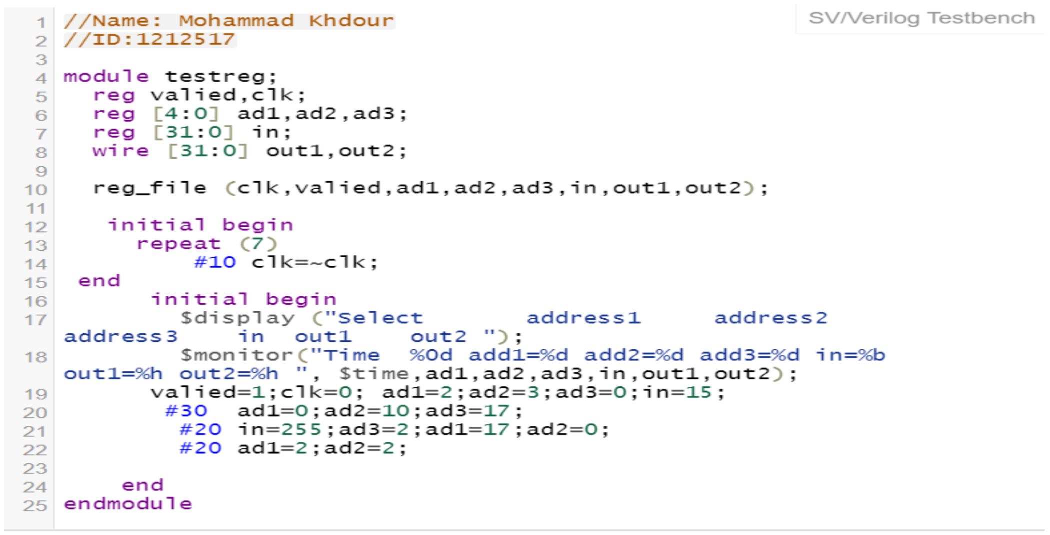


```
//Name: Mohammad Khdour
//ID :1212517
module reg_file
(clk,valied_opcode,addr1,addr2,addr3,in,out1, out2);
  input clk;
  input valied_opcode;
  input [4:0] addr1,addr2,addr3;
  input [31:0] in;
  output reg [31:0] out1,out2;
  reg [31:0] memory [0:31];

  initial begin
    memory [0]= 32'h0000;
    memory [1]= 32'h2D8E;
    memory [2]= 32'h2D2A;
    memory [3]= 32'h3BE2;
    memory [4]= 32'h257A;
    memory [5]= 32'h399A;
    memory [6]= 32'h0CD8;
    memory [7]= 32'h172C;
    memory [8]= 32'h07BA;
    memory [9]= 32'h1330;
    memory [10]= 32'h094C;
    memory [11]= 32'h0786;
    memory [12]= 32'h31B6;
    memory [13]= 32'h00B0;
    memory [14]= 32'h20D8;
    memory [15]= 32'h20CA;
    memory [16]= 32'h3524;
    memory [17]= 32'h27EE;
    memory [18]= 32'h1C5E;
    memory [19]= 32'h27CE;
    memory [20]= 32'h221E;
    memory [21]= 32'h3090;
    memory [22]= 32'h2214;
    memory [23]= 32'h1524;
    memory [24]= 32'h2B4A;
    memory [25]= 32'h08A4;
    memory [26]= 32'h182C;
    memory [27]= 32'h1B90;
    memory [28]= 32'h0EA0;
    memory [29]= 32'h1686;
    memory [30]= 32'h0D54;
    memory [31]= 32'h0000;
  end

  always @(posedge clk) begin
    if (valied_opcode==1) begin
    memory [addr3] = in;
    out1= memory [addr1];
    out2= memory [addr2];
  end
  end
endmodule
```

We defined array from 32-bit and its size is 5-bit named memory, it works as a register, that content all the values we initialize, and three 5-bit addresses. Output 1 produces the item within the register file that is address by Address 1, and output 2 produces the item from register file that is address by address 2, and a value is supplied through input and written to the address by Address 3. Address1, and address2 for read, address3 for write to.

Reg_file is synchronous module which make it sensitive for positive edge clock, when positive edge coming and the opcode is valid then the register will work normally, otherwise the register file will ignore its input, and will not update its output.

1.2.2 Testbench: for reg_file

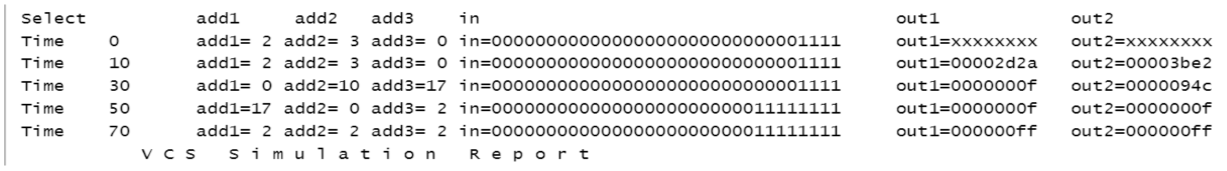


SV/Verilog Testbench

```verilog
//Name: Mohammad Khdour
//ID:1212517

module testreg;
  reg valied,clk;
  reg [4:0] ad1,ad2,ad3;
  reg [31:0] in;
  wire [31:0] out1,out2;

  reg_file (clk,valied,ad1,ad2,ad3,in,out1,out2);

   initial begin
     repeat (7)
         #10 clk=~clk;
 end
       initial begin
         $display ("Select       address1     address2
address3    in   out1    out2 ");
         $monitor("Time  %0d add1=%d add2=%d add3=%d in=%b
out1=%h out2=%h ", $time,ad1,ad2,ad3,in,out1,out2);
       valied=1;clk=0; ad1=2;ad2=3;ad3=0;in=15;
        #30  ad1=0;ad2=10;ad3=17;
         #20 in=255;ad3=2;ad1=17;ad2=0;
         #20 ad1=2;ad2=2;

     end
endmodule
```

This is small testbench to check reg_file is work correctly or not, so we defined all the argument and make two initial block to work parallel in same time, one for clock and the other to assign value for input parameter. We used **$display** to print the addresses, and **$monitor** Verilog system task to print both input and output when on of them has changed.

1.2.3 Results



```
Select          add1     add2   add3    in                                    out1          out2
Time     0      add1= 2 add2= 3 add3= 0 in=00000000000000000000000000001111   out1=xxxxxxxx out2=xxxxxxxx
Time     10     add1= 2 add2= 3 add3= 0 in=00000000000000000000000000001111   out1=00002d2a out2=00003be2
Time     30     add1= 0 add2=10 add3=17 in=00000000000000000000000000001111   out1=0000000f out2=0000094c
Time     50     add1=17 add2= 0 add3= 2 in=00000000000000000000000011111111   out1=0000000f out2=0000000f
Time     70     add1= 2 add2= 2 add3= 2 in=00000000000000000000000011111111   out1=000000ff out2=000000ff
           V C S   S i m u l a t i o n   R e p o r t
```

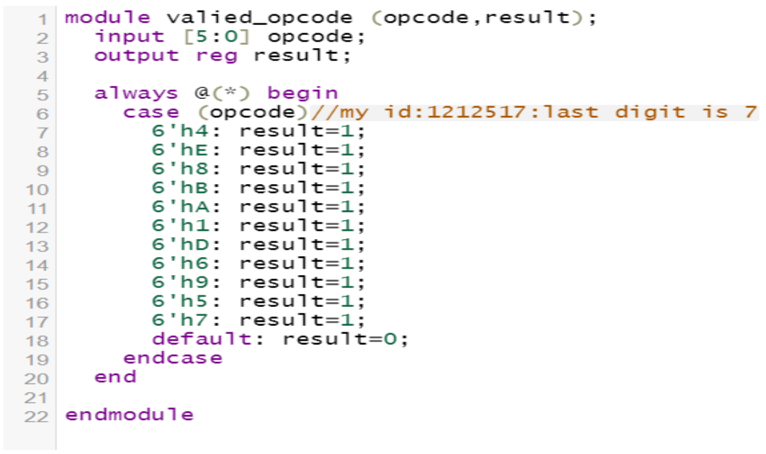
- At time 0: there is no positive edge clock, so there is no output.
- At time 10: the positive edge has coming, so output1 will have the value exist in address1 in memory location that equal to 2, and output2 will have the value exist in address2 from memory that equal to 3, so the output1 will be equal to 0x2d2a in hexadecimal and 11562 in decimal, and output2 will be equal to 0x3be2 in hexadecimal and 15330 in decimal, after that the input (**in**) will be written to address3 in memory location, we can say address 0 equal 15.
- At time 30: address1 equal to the address which we assigned to it number 15, so the output1 must equal 15 that locate at memory address zero, and output2 will equal the value in address 10 from memory that equal to 0x094c in hexadecimal.

1.3 Another module

1.3.1 Valid Opcode

We add this part to return one if opcode exist, or zero if opcode does not exist. We use this module for register file to give output for valid opcode, otherwise will not access the memory.

1.3.1.1 Code

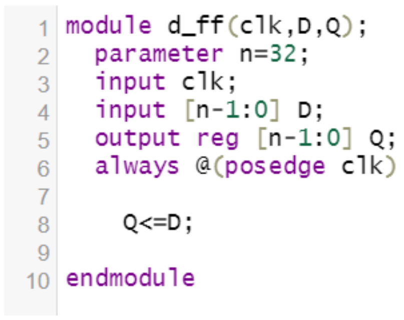


```
module valied_opcode (opcode,result);
  input [5:0] opcode;
  output reg result;

  always @(*) begin
    case (opcode)//my id:1212517:last digit is 7
      6'h4: result=1;
      6'hE: result=1;
      6'h8: result=1;
      6'hB: result=1;
      6'hA: result=1;
      6'h1: result=1;
      6'hD: result=1;
      6'h6: result=1;
      6'h9: result=1;
      6'h5: result=1;
      6'h7: result=1;
      default: result=0;
    endcase
  end

endmodule
```

1.3.2 D-flipflop

We need this part to make delay one cycle for the opcode when entered ALU.

1.3.2.1 Code



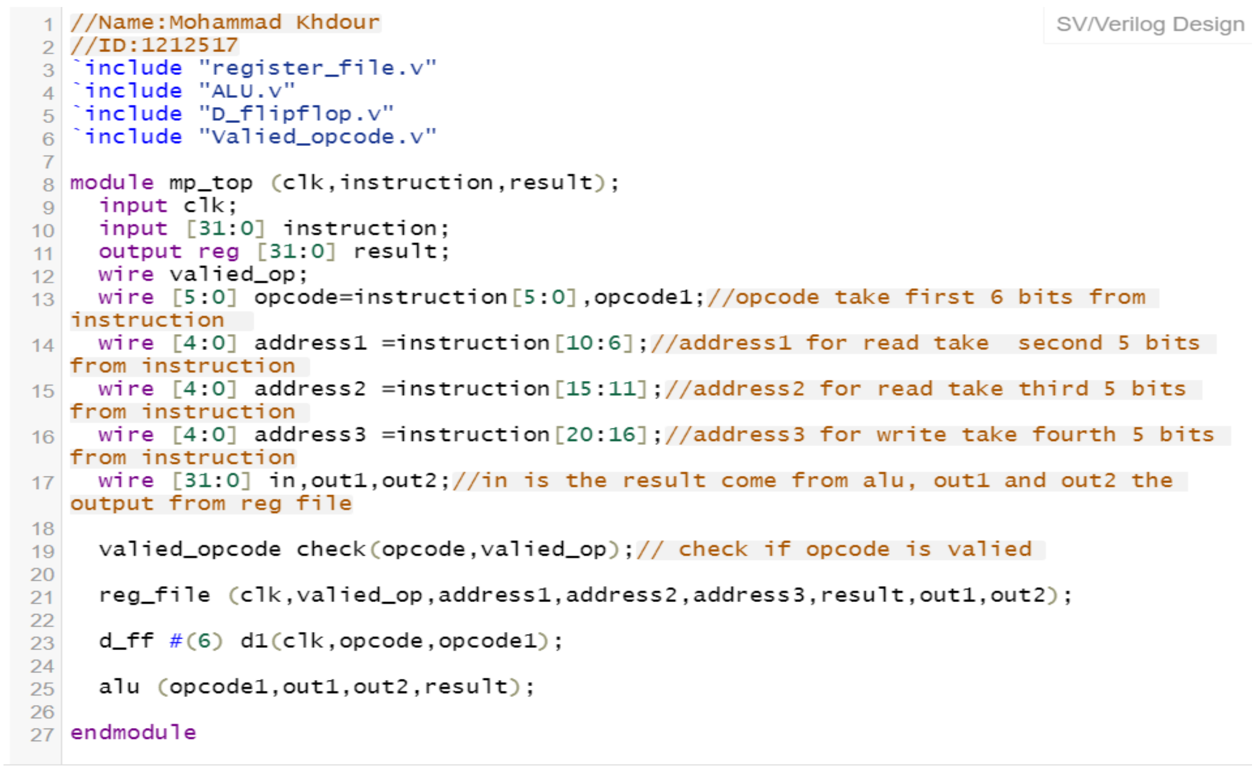
```
module d_ff(clk,D,Q);
  parameter n=32;
  input clk;
  input [n-1:0] D;
  output reg [n-1:0] Q;
  always @(posedge clk)

    Q<=D;

endmodule
```

2. Part two:

2.1 TOP module

In this part we will connect all the modules from part one in one module called mp_top module.

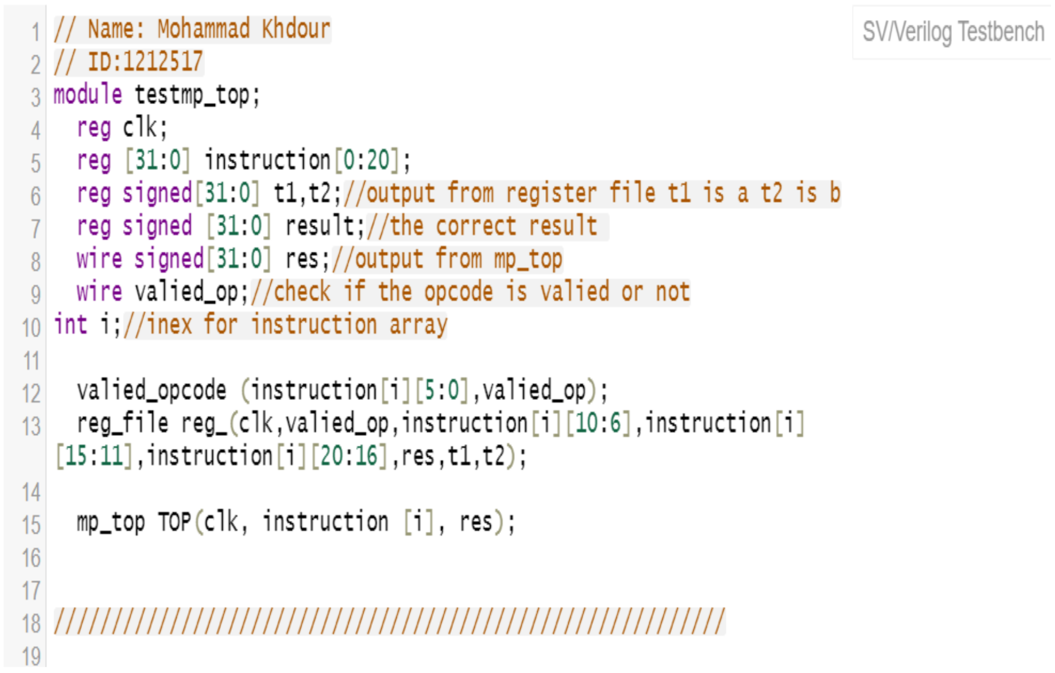
2.2 Code



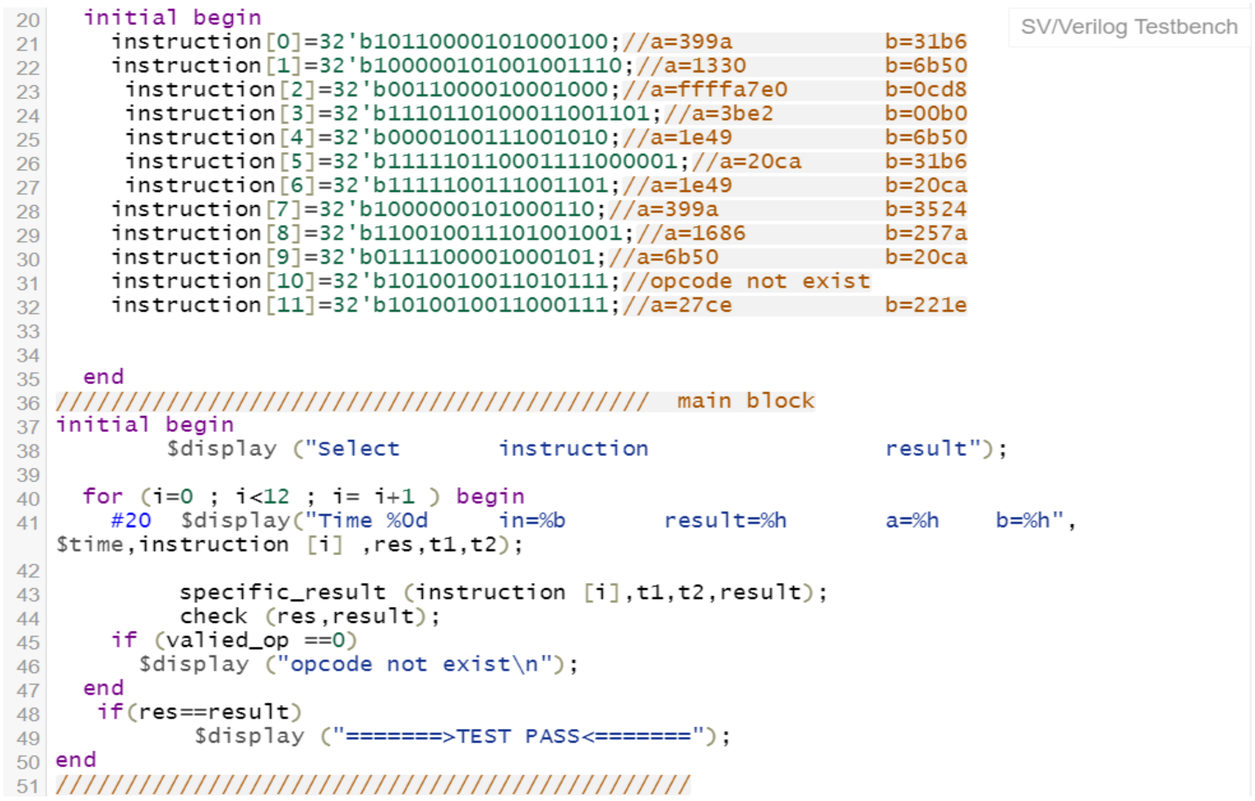
```verilog
//Name:Mohammad Khdour
//ID:1212517
`include "register_file.v"
`include "ALU.v"
`include "D_flipflop.v"
`include "valied_opcode.v"

module mp_top (clk,instruction,result);
  input clk;
  input [31:0] instruction;
  output reg [31:0] result;
  wire valied_op;
  wire [5:0] opcode=instruction[5:0],opcode1;//opcode take first 6 bits from instruction
  wire [4:0] address1 =instruction[10:6];//address1 for read take  second 5 bits from instruction
  wire [4:0] address2 =instruction[15:11];//address2 for read take third 5 bits from instruction
  wire [4:0] address3 =instruction[20:16];//address3 for write take fourth 5 bits from instruction
  wire [31:0] in,out1,out2;//in is the result come from alu, out1 and out2 the output from reg file

  valied_opcode check(opcode,valied_op);// check if opcode is valied

  reg_file (clk,valied_op,address1,address2,address3,result,out1,out2);

  d_ff #(6) d1(clk,opcode,opcode1);

  alu (opcode1,out1,out2,result);

endmodule
```

To build mp_top module which is the top module we must include all part that we want it. After that we know the top module has two input, and one output. The first input is clock, we need it in asynchronies register file and D-flipflop, and the second input is the instruction which has the addresses for numbers in memory. The output result is the output from ALU which make mathematical or logical operation defined by the opcode between out1, and out2 from register file.

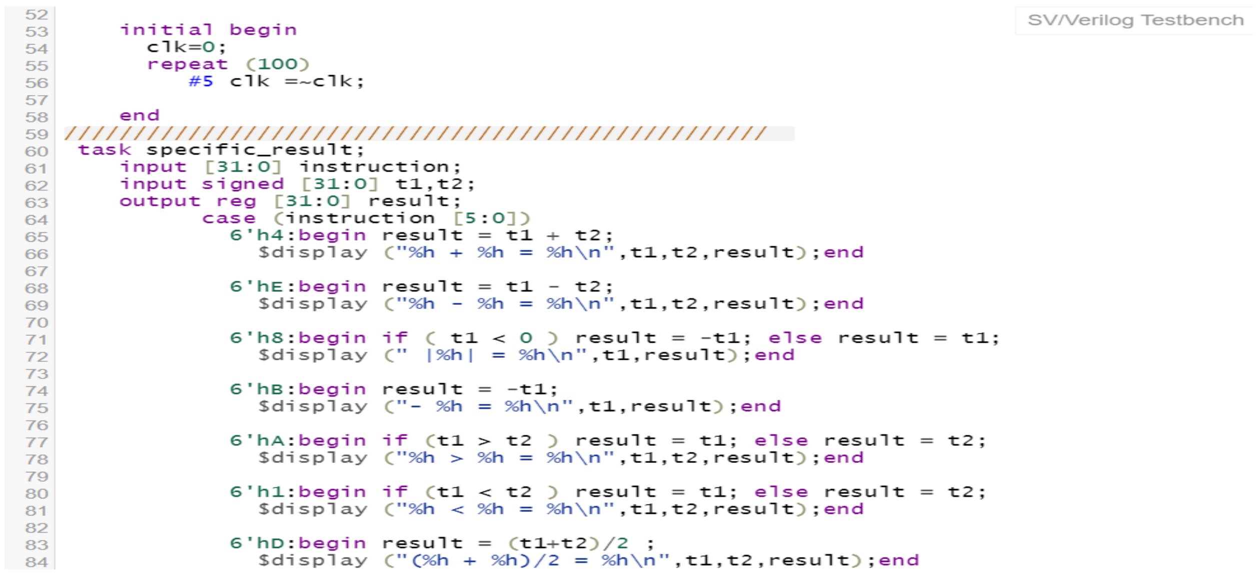
2.3 Testbench



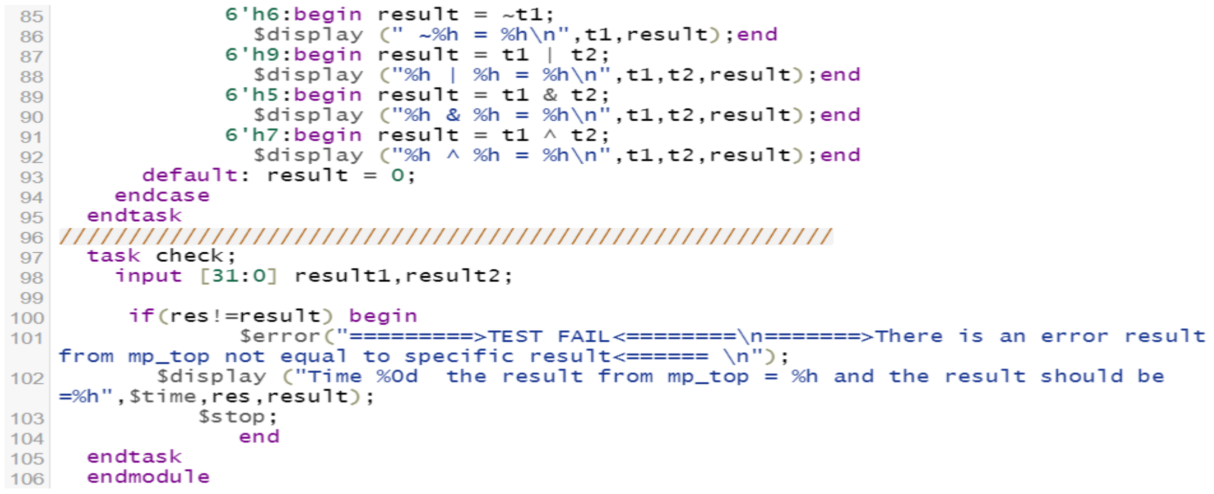
```
// Name: Mohammad Khdour
// ID:1212517
module testmp_top;
  reg clk;
  reg [31:0] instruction[0:20];
  reg signed[31:0] t1,t2;//output from register file t1 is a t2 is b
  reg signed [31:0] result;//the correct result
  wire signed[31:0] res;//output from mp_top
  wire valied_op;//check if the opcode is valied or not
int i;//inex for instruction array

  valied_opcode (instruction[i][5:0],valied_op);
  reg_file reg_(clk,valied_op,instruction[i][10:6],instruction[i]
[15:11],instruction[i][20:16],res,t1,t2);

  mp_top TOP(clk, instruction [i], res);


//////////////////////////////////////////////////////////

```



```
  initial begin
    instruction[0]=32'b10110000101000100;//a=399a          b=31b6
    instruction[1]=32'b10000010100100110;//a=1330          b=6b50
     instruction[2]=32'b0011000010001000;//a=ffffa7e0      b=0cd8
     instruction[3]=32'b11101101000110011101;//a=3be2      b=00b0
     instruction[4]=32'b0000100111001010;//a=1e49          b=6b50
     instruction[5]=32'b111110110001111000001;//a=20ca     b=31b6
     instruction[6]=32'b1111100111001101;//a=1e49          b=20ca
    instruction[7]=32'b1000000101000110;//a=399a           b=3524
    instruction[8]=32'b11001001110100l001;//a=1686         b=257a
    instruction[9]=32'b0111100001000101;//a=6b50           b=20ca
    instruction[10]=32'b1010010011010111;//opcode not exist
    instruction[11]=32'b1010010011000111;//a=27ce          b=221e


  end
///////////////////////////////////////////  main block
initial begin
        $display ("Select       instruction                 result");

  for (i=0 ; i<12 ; i= i+1 ) begin
    #20  $display("Time %0d     in=%b        result=%h        a=%h     b=%h",
$time,instruction [i] ,res,t1,t2);

         specific_result (instruction [i],t1,t2,result);
         check (res,result);
    if (valied_op ==0)
      $display ("opcode not exist\n");
  end
   if(res==result)
          $display ("=======>TEST PASS<=======");
end
//////////////////////////////////////////////
```



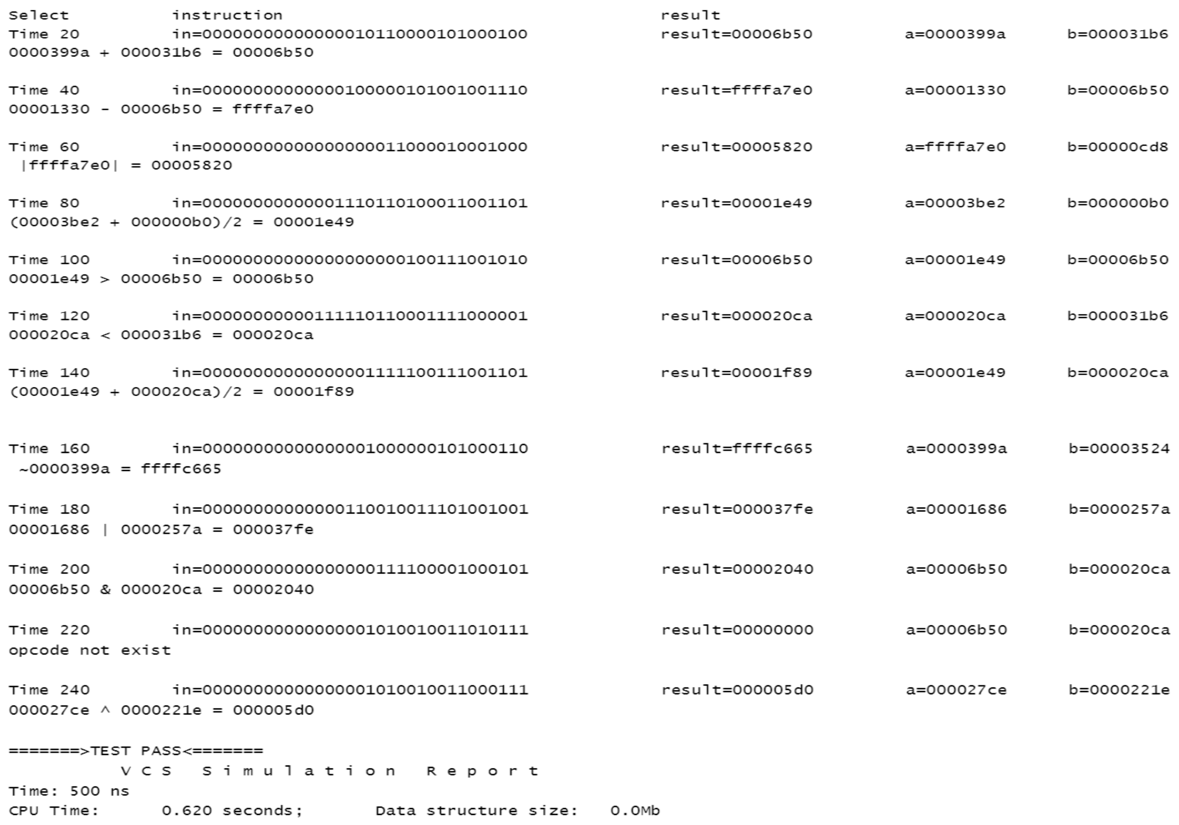
```verilog

    initial begin
      clk=0;
      repeat (100)
         #5 clk =~clk;

    end
///////////////////////////////////////////////////
 task specific_result;
    input [31:0] instruction;
    input signed [31:0] t1,t2;
    output reg [31:0] result;
          case (instruction [5:0])
            6'h4:begin result = t1 + t2;
              $display ("%h + %h = %h\n",t1,t2,result);end

            6'hE:begin result = t1 - t2;
              $display ("%h - %h = %h\n",t1,t2,result);end

            6'h8:begin if ( t1 < 0 ) result = -t1; else result = t1;
              $display ("  |%h| = %h\n",t1,result);end

            6'hB:begin result = -t1;
              $display ("- %h = %h\n",t1,result);end

            6'hA:begin if (t1 > t2 ) result = t1; else result = t2;
              $display ("%h > %h = %h\n",t1,t2,result);end

            6'h1:begin if (t1 < t2 ) result = t1; else result = t2;
              $display ("%h < %h = %h\n",t1,t2,result);end

            6'hD:begin result = (t1+t2)/2 ;
              $display ("(%h + %h)/2 = %h\n",t1,t2,result);end
```



```verilog
            6'h6:begin result = ~t1;
              $display (" ~%h = %h\n",t1,result);end
            6'h9:begin result = t1 | t2;
              $display ("%h | %h = %h\n",t1,t2,result);end
            6'h5:begin result = t1 & t2;
              $display ("%h & %h = %h\n",t1,t2,result);end
            6'h7:begin result = t1 ^ t2;
              $display ("%h ^ %h = %h\n",t1,t2,result);end
      default: result = 0;
    endcase
  endtask
///////////////////////////////////////////////////////
  task check;
    input [31:0] result1,result2;

     if(res!=result) begin
              $error("=========>TEST FAIL<=========\n=======>There is an error result
from mp_top not equal to specific result<====== \n");
         $display ("Time %0d  the result from mp_top = %h and the result should be
=%h",$time,res,result);
            $stop;
              end
  endtask
  endmodule
```

In test bench we use three initial block first one to defined array of instructions which can make different operation, the second one is the main block, we use this block to bring all instruction one by one every 20ns, the delay put 20ns because the positive edge coming every 10ns, so we need two positive edge to access correctly the instruction, first positive edge to take the output t1, and t2 from register file, and the second one to write the result from alu in register file. And last initial block for clock. We use in main block two important tasks, **specific_result**, and **check**. **Specific_result** task used to write expected value for **mp_top** in argument called **result**, and **check** task used to compare between result from mp_top module and expected result from specific_result, if they are not equal print error massage and test fail massage and stop simulation, otherwise do nothing, we doing this for every instruction until we execute all instruction, and print test pass if there is no error in one of the instructions.

2.4 Results



```
Select          instruction                                        result
Time 20         in=00000000000000001011000010100100                result=00006b50        a=0000399a      b=000031b6
0000399a + 000031b6 = 00006b50

Time 40         in=00000000000000010000010100100111 0               result=ffffa7e0        a=00001330      b=00006b50
00001330 - 00006b50 = ffffa7e0

Time 60         in=00000000000000000011000010001000                result=00005820        a=ffffa7e0      b=00000cd8
 |ffffa7e0| = 00005820

Time 80         in=00000000000000111011010001100110 1               result=00001e49        a=00003be2      b=000000b0
(00003be2 + 000000b0)/2 = 00001e49

Time 100        in=00000000000000000000100111001010                result=00006b50        a=00001e49      b=00006b50
00001e49 > 00006b50 = 00006b50

Time 120        in=00000000000111110110001111000001                result=000020ca        a=000020ca      b=000031b6
000020ca < 000031b6 = 000020ca

Time 140        in=00000000000000001111100111001101                result=00001f89        a=00001e49      b=000020ca
(00001e49 + 000020ca)/2 = 00001f89


Time 160        in=00000000000000001000000101000110                result=ffffc665        a=0000399a      b=00003524
 ~0000399a = ffffc665

Time 180        in=00000000000000011001001110100100 1               result=000037fe        a=00001686      b=0000257a
00001686 | 0000257a = 000037fe

Time 200        in=00000000000000000111100001000101                result=00002040        a=00006b50      b=000020ca
00006b50 & 000020ca = 00002040

Time 220        in=00000000000000001010010011010111                result=00000000        a=00006b50      b=000020ca
opcode not exist

Time 240        in=00000000000000001010010011000111                result=000005d0        a=000027ce      b=0000221e
000027ce ^ 0000221e = 000005d0

=======>TEST PASS<=======
           V C S   S i m u l a t i o n   R e p o r t
Time: 500 ns
CPU Time:      0.620 seconds;       Data structure size:   0.0Mb
```

For all result and numbers, we use hexadecimal numbers.

A=memory[address1]

b=memory[address2].

- At time 20: first instruction will be executed, this instruction make addition operation between the number in address1 (5) from memory which equal to 399a and the number in address2 (11) from memory which equal to 31b6 and their addition is 6b50, so we store the result in address3 (1) in memory.
- At time 40: the instruction made subtraction operation between the number in address1 (9) and number in address2 (1). The number in address1 (9) equal 1330 from memory, and the second number from address2 (1) which we store in it the result from first instruction which equal 6b50. The subtraction from 1330 and 6b50 equal to ffffa7e0 this number is negative. We sore this result in address3 (2).

- At time 60: the instruction made absolute value for number from address1 (2), this number is the result in previous instruction which equal ffffa7e0, and after we execute the instruction the result equal 5820 in hexadecimal and sore it in address3 (0).
- At time 120: this instruction returns the min value between number from address1 (7) which equal to 20ca and the number from address2 (12) which equal to 31b6. The result should me 20ca because it is less than 31b6 and store it in address3 (31).
- At time 160: the instruction made not operation for number in address1 (5) which equal 399a. the result will be ffffc665 and store it in address3 (16).

  ~0000000000000000/0011100110011010 = 1111111111111111/1100011001100101

  In hexadecimal equal ffffc665

- At time 200: the instruction made and logical operation between number in address1 (1) which equal the result from first instruction 6b50 and the number in address2 (15) which equal 20ca. the result should be equal 2040.
  0110101101010000
  &
  0010000001000000
  0010000001000000
  In hexadecimal equal 2040
- At time 220: this instruction has not had valid opcode so the result will equal zero and a, b the output from register file not changed, and print massage to till us the opcode we entered are not exist.