CPU DESIGN LAB REPORT

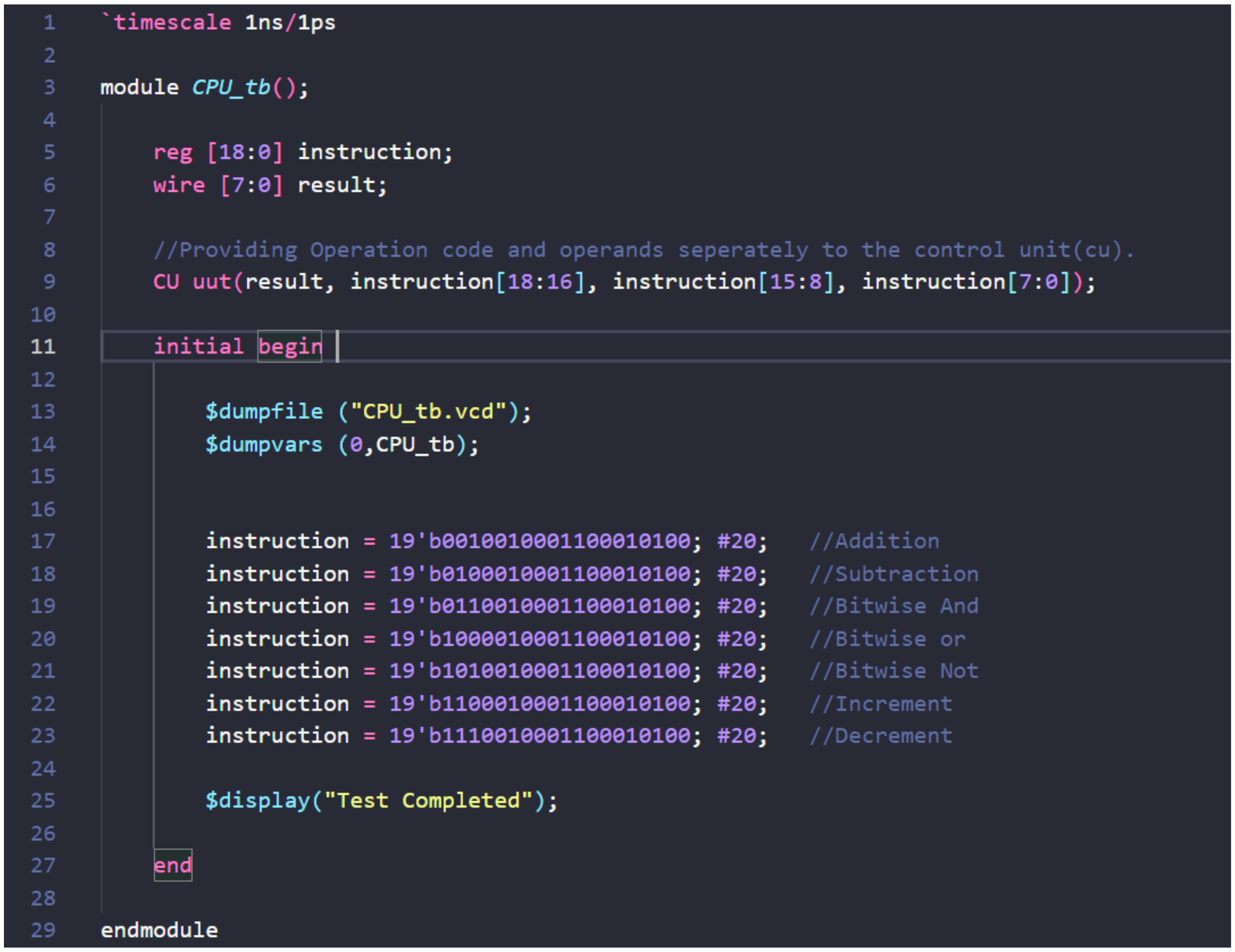
CS2310 VERILOG LAB 4

Name	:	G ASISH SAI
Roll No	:	CS21B031
Batch No	:	11

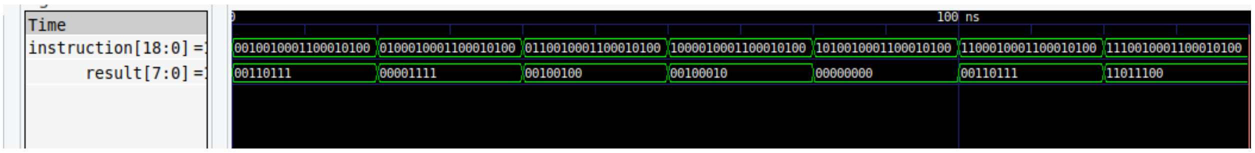
All modules are explained in detail along with the test bench and relevant waveform snapshots/outputs below.

CPU_tb.v

- The instruction is separated into operation code, operand 1, operand2 and given as inputs to module **CU(control unit).**
- The output is stored in result.



```verilog
`timescale 1ns/1ps

module CPU_tb();

    reg [18:0] instruction;
    wire [7:0] result;

    //Providing Operation code and operands seperately to the control unit(cu).
    CU uut(result, instruction[18:16], instruction[15:8], instruction[7:0]);

    initial begin

        $dumpfile ("CPU_tb.vcd");
        $dumpvars (0,CPU_tb);


        instruction = 19'b0010010001100010100; #20;   //Addition
        instruction = 19'b0100010001100010100; #20;   //Subtraction
        instruction = 19'b0110010001100010100; #20;   //Bitwise And
        instruction = 19'b1000010001100010100; #20;   //Bitwise or
        instruction = 19'b1010010001100010100; #20;   //Bitwise Not
        instruction = 19'b1100010001100010100; #20;   //Increment
        instruction = 19'b1110010001100010100; #20;   //Decrement

        $display("Test Completed");

    end

endmodule
```



```
Time                 0                                                                                                   100 ns
instruction[18:0] =  0010010001100010100  0100010001100010100  0110010001100010100  1000010001100010100  1010010001100010100  1100010001100010100  1110010001100010100
result[7:0] =        00110111             00001111             00100100             00100010             00000000             00110111             11011100
```

CU.v

- In this module op indicates operation code, operand1 and operand2 are the operands.
- Operand1 and Operand2 are sent as inputs to **ALU(Arithmetic and logic unit)**. The 7 outputs(x1 to x7) returned are the respective results in the 7 operations. (Operations are taken in the order given with addition being given index 1)
- Now the idea of Decoder is used to find which operation the operation code corresponds to. opcode[i] in the module is 1 if it represents the ith operation. (Operations are numbered in the same way as the outputs)
- Now the idea of Multiplexer is used to copy the output of the corresponding operation into the result.

(1) An AND operation is implemented on each output (xi) and its opcode (opcode[i]) and stored in temporary variables (temp1 to temp7) each of 8 bits.

(2) An OR operation is implemented on each of the 8 bits in the 7 temporary variables and the value is stored in the result. The result is returned to the **CPU_tb.**

ALU.v

- In this module operand1 and operand2 are received as inputs.
- Each of the 7 operations are computed as follows.

(1) To compute ADDITION , operand1 and operand2 are passed as inputs to **fulladder_tb** and output is stored in x1.

(2) To compute SUBTRACTION , operand1 and 2's complement of operand2 are passed as input to **fulladder_tb** and output is stored in x2.

2's complement of operand2 is computed by applying a NOT over each bit of operand2 and adding 1 by using a full adder **(fulladder_tb)**.

(3) To compute INCREMENT , operand 1 and the literal 8'b00000001 as operand 2 are given as inputs to **fulladder_tb** and output is stored in x3.

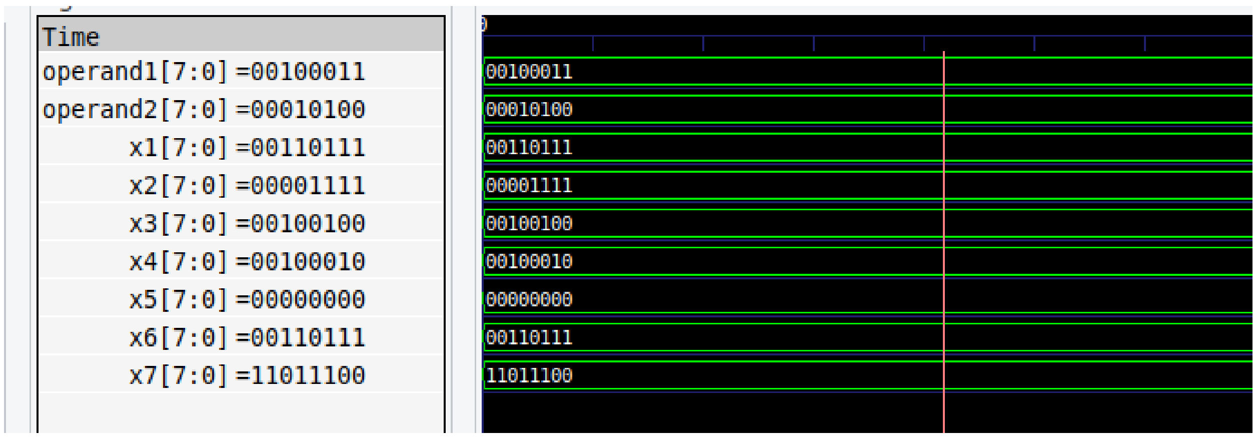
(4) To compute DECREMENT , operand 1 and the literal 8'b11111111 as operand 2 are given as inputs to **fulladder_tb** and output is stored in x4.

(5) To compute AND , operand 1 and operand 2 are given as inputs to **and12** and output is stored in x5.

(6) To compute OR , operand 1 and operand 2 are given as inputs to **or12** and output is stored in x6.

(7) To compute NOT , operand 1 is given as input to **not1** and output is stored in x7.

- The 7 outputs (x1 to x7) are returned to **CU.**



```
Time
operand1[7:0] =00100011
operand2[7:0] =00010100
      x1[7:0] =00110111
      x2[7:0] =00001111
      x3[7:0] =00100100
      x4[7:0] =00100010
      x5[7:0] =00000000
      x6[7:0] =00110111
      x7[7:0] =11011100
```

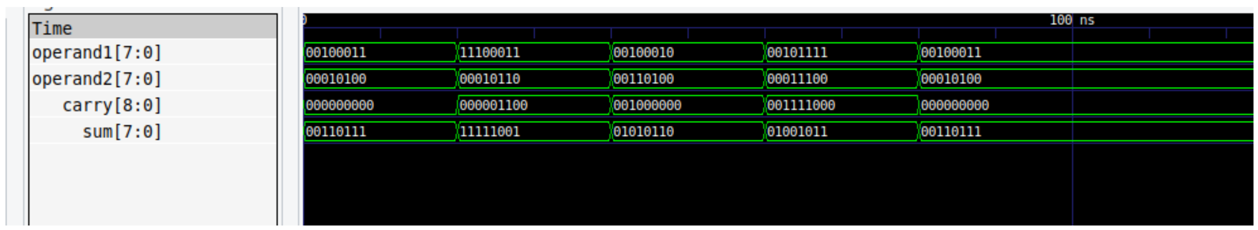
The outputs (x1 to x7) of the 7 given operations for two operands given in the test bench.

fulladder_tb.v

- This module is called multiple times in **ALU,** operand1 and operand2 Each consisting of 8 bits are given as inputs.
- An [8:0] carry array is declared as well to store the carry. carry[0] is initialized to 0 using an AND gate.
-
  (1) **fulladder** module is called 8 times, each time passing ith bit of operand1, operand2, carry as input.
  (2) A carry is returned which is stored in (i+1)th bit of carry array, and a sum is returned which is stored in the ith bit of sum array.
- The sum array is returned to the **ALU**.

fulladder.v

- In this module a, b, carry_in are received as inputs.
- To implement a full adder, I am implementing two half adders. Half adder is implemented by using an XOR and an AND gate.
- The inputs of the first half adder are a,b and outputs are stored in temp1 and temp2 (temporary variables).
- The inputs of the second half adder are temp1 and carry_in and outputs are stored in sum and temp3. An OR gate is implemented over temp2 and temp3 to obtain carry_out.
- carry_out and sum are returned as outputs to **fulladder_tb.**

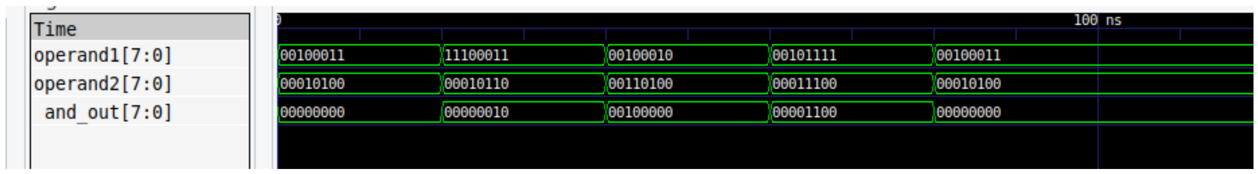


```
Time                                                                          100 ns
operand1[7:0]  00100011   11100011   00100010   00101111   00100011
operand2[7:0]  00010100   00010110   00110100   00011100   00010100
   carry[8:0]  000000000  000001100  001000000  001111000  000000000
     sum[7:0]  00110111   11111001   01010110   01001011   00110111
```

fulladder_tb.v output for 5 different examples(not the test bench one)

and12.v

- In this module operand1, operand2 are given as inputs.
- An AND operation is implemented over 8 bits of operand1, operand2 and the output is stored in and_out.
- and_out is returned as output to **ALU.**

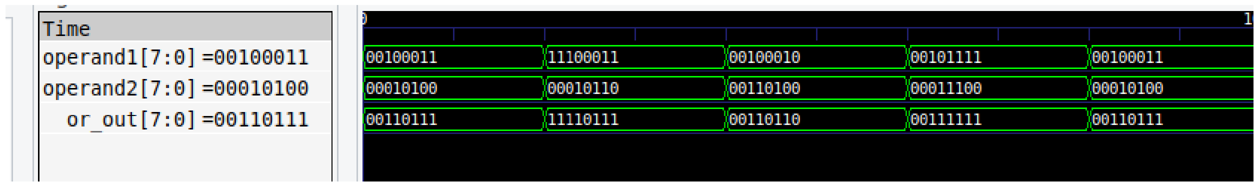


```
Time                                                                 100 ns
operand1[7:0]  00100011  11100011  00100010  00101111  00100011
operand2[7:0]  00010100  00010110  00110100  00011100  00010100
 and_out[7:0]  00000000  00000010  00100000  00001100  00000000
```

and12.v output for 5 different examples (not the test bench one)

or12.v

- In this module operand1, operand2 are given as inputs.
- An OR operation is implemented over 8 bits of operand1, operand2 and the output is stored in or_out.
- or_out is returned as output to **ALU.**

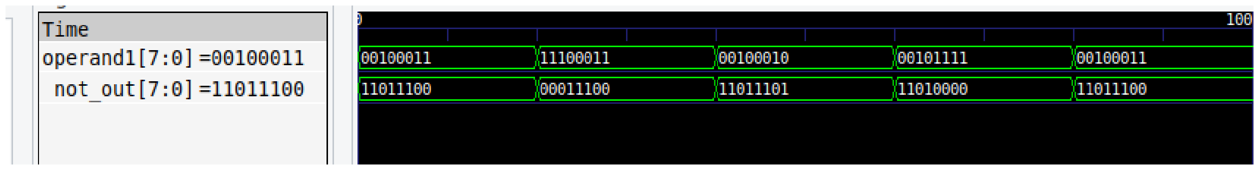


```
Time
operand1[7:0] =00100011    00100011  11100011  00100010  00101111  00100011
operand2[7:0] =00010100    00010100  00010110  00110100  00011100  00010100
   or_out[7:0] =00110111   00110111  11110111  00110110  00111111  00110111
```

or12.v output for 5 different examples (other than the one in the test bench).

not1.v

- In this module operand1 is given as inputs.
- A NOT operation is implemented over 8 bits of operand1 and the output is stored in not_out.
- not_out is returned as output to **ALU.**



```
Time                                                                        100
operand1[7:0] =00100011   00100011  11100011  00100010  00101111  00100011
  not_out[7:0] =11011100  11011100  00011100  11011101  11010000  11011100
```

not1.v output for 5 different examples (other than the one in the test bench).