Introduction:

In this project, the objective was to implement a 4-bit computer in Verilog HDL. As a starter, a set of instructions had been issued to different student IDs. My student ID is: **1606099.** Since the last digit of my ID is **'9',** the instruction set that I had to implement is as follows:

1	ADD A,B	9	IN B
2	SUB A,B	10	OUT A
3	XCHG B,A	11	JMP ADDRESS
4	MOV A,[ADDRESS]	12	PUSH A
5	MOV [ADDRESS],B	13	POP A
6	JNZ ADDRESS	14	CALL ADDRESS
7	XOR A,[ADDRESS]	15	RET
8	PUSHF	16	HLT

Defining the instruction set:

ADD A,B: Add value stored in B to value stored in A and store the result in A.

SUB A,B: Subtract and store the result in A.

XCHG B,A: Swap/exchange the value stored in B and value stored in A.

MOV A,[ADDRESS]: Store the value stored in data memory at the address given by the instruction to A.

MOV [ADDRESS],B: Store the value in B to the address given by instruction in the data memory.

JNZ ADDRESS: If zero flag not activated, jump to the instruction given by address stored in the program memory.

XOR A,[ADDRESS]: XOR the value stored in A and value stored in data memory at the address given by the instruction, and store the result in A.

PUSHF: Push the flags to the stack.

IN B: Store the input port data to B.

OUT A: Send the value in A to the output port.

JMP ADDRESS: Jump to the instruction given by address stored in the program memory.

PUSH A: Push the value in A to the stack.

POP A: Store the latest value stored in the stack to A.

CALL ADDRESS: Go to the instruction given by address stored in the program memory.

RET: Return to the instruction from where it was called.

HLT: Halt the program execution.

No.	Instruction	Assembly Key	Machine Code(Decimal)
1	ADD A,B	ADD_A_B	0
2	SUB A,B	SUB_A_B	1
3	XCHG B,A	XCHG_B_A	2
4	MOV A,[ADDRESS]	MOV_A_ADD	3
5	MOV [ADDRESS],B	MOV_ADD_B	4
6	JNZ ADDRESS	JNZ_ADD	5
7	XOR A,[ADDRESS]	XOR_A_ADD	6
8	PUSHF	PUSHF	7
9	IN B	IN_B	8
10	OUT A	OUT_A	9
11	JMP ADDRESS	JMP_ADD	10
12	PUSH A	PUSH_A	11
13	POP A	POP_A	12
14	CALL ADDRESS	CALL_ADD	13
15	RET	RET	14
16	HLT	HLT	15

Example Assembly Code:

Code:

```
MOV A,[0000]
XCHG B,A
MOV A,[0001]
ADD A,B
SUB A,B
OUT A
HLT
```

Expected result:

The data memory has decimal 3 stored in address 0000 and decimal 5 stored in address 0001. So, the result should be 5+3 = 8 then, 8-3 = 5. So, after the execution, 5 should be seen at the output port.

Design Code:

```
module four_bit_comp(clock,reset,data_in,prog_inst,prog_data,prog_count,data_out);
 input clock;
 input reset;
 input [3:0] data_in;                    //data input port
 input [3:0] prog_inst;                  //program intrsuction input port
 input [3:0] prog_data;                  //program data input port
 input [3:0] prog_count;                 //program counter

 output reg [3:0] data_out;              //output port
 reg [3:0] A,B,C;                        //arithmetic registers
 reg [3:0] IP,SP;                        //instruction pointer and stack pointer
 reg [3:0] data_mem[15:0];               //data memory
 reg [3:0] prog_inst_mem[15:0];          //program intsruction memory, stores inst code
 reg [3:0] prog_data_mem[15:0];          //program data memory, stores data corresponding to inst
 reg [3:0] stack[15:0];                  //stack memory
 reg HF,ZF,CF = 1'd0;                    //halt flag,zero flag,carry flag
 integer count = 0;

 always @(negedge clock)                 //this block loads the program and data
  begin                                  //before executing them
   data_mem[prog_count] = data_in;
   prog_inst_mem[prog_count] = prog_inst;
   prog_data_mem[prog_count] = prog_data;
  end

 always @(posedge clock)                 //this block executes the program
  begin

   if (reset == 1)                       //resets everything
    begin
     HF = 0;ZF = 0;CF = 0;
     A = 0;B = 0;C = 0;
     IP = 0;SP = 15;
     data_out = 4'bzzzz;
    end
   else if (reset == 0 && HF == 0)
    begin
     if (prog_inst_mem[IP] == 0)
       begin                                        //ADD A, B
         {CF, A} = A + B;
       if (A == 0) begin
                   ZF = 1;
                   end
         end
else if (prog_inst_mem[IP] == 1)
         begin                                      //SUB A, B
         {CF, A} = A - B;
       if (A == 0) begin
                   ZF = 1;
                   end
         end
```

```
 else if (prog_inst_mem[IP] == 2)
    begin                                   //XCHG B, A
    C = A;
    A = B;
    B = C;
  if (B == 0) begin
                ZF = 1;
                end
    end
else if (prog_inst_mem[IP] == 3)
begin                                       //MOV A,[ADDRESS]
  A = data_mem[prog_data_mem[IP]];
  if (A == 0) begin
                ZF = 1;
                end
end
else if (prog_inst_mem[IP] == 4)
begin                                       //MOV [ADDRESS],B
  data_mem[prog_data_mem[IP]] = B;
  if (B == 0) begin
                ZF = 1;
                end
end
else if (prog_inst_mem[IP] == 5)
begin                                       //JNZ ADDRESS
  if (ZF == 0) IP = prog_data_mem[IP];
end
else if (prog_inst_mem[IP] == 6)
begin                                       //XOR A,[ADDRESS]
  A = A ^ data_mem[prog_data_mem[IP]];
  if (A == 0) begin
                ZF = 1;
                end
end
else if (prog_inst_mem[IP] == 7)
begin                                       //PUSHF
  stack[SP] = {1'b0,ZF,CF,HF};
  SP = SP - 1;
end
else if (prog_inst_mem[IP] == 8)
begin
  B = data_in;                              //IN B
  if (B == 0) begin
                ZF = 1;
                end
end
else if (prog_inst_mem[IP] == 9)
begin                                       //OUT A
  data_out = A;
  if (A == 0) begin
                ZF = 1;
                end
end
```

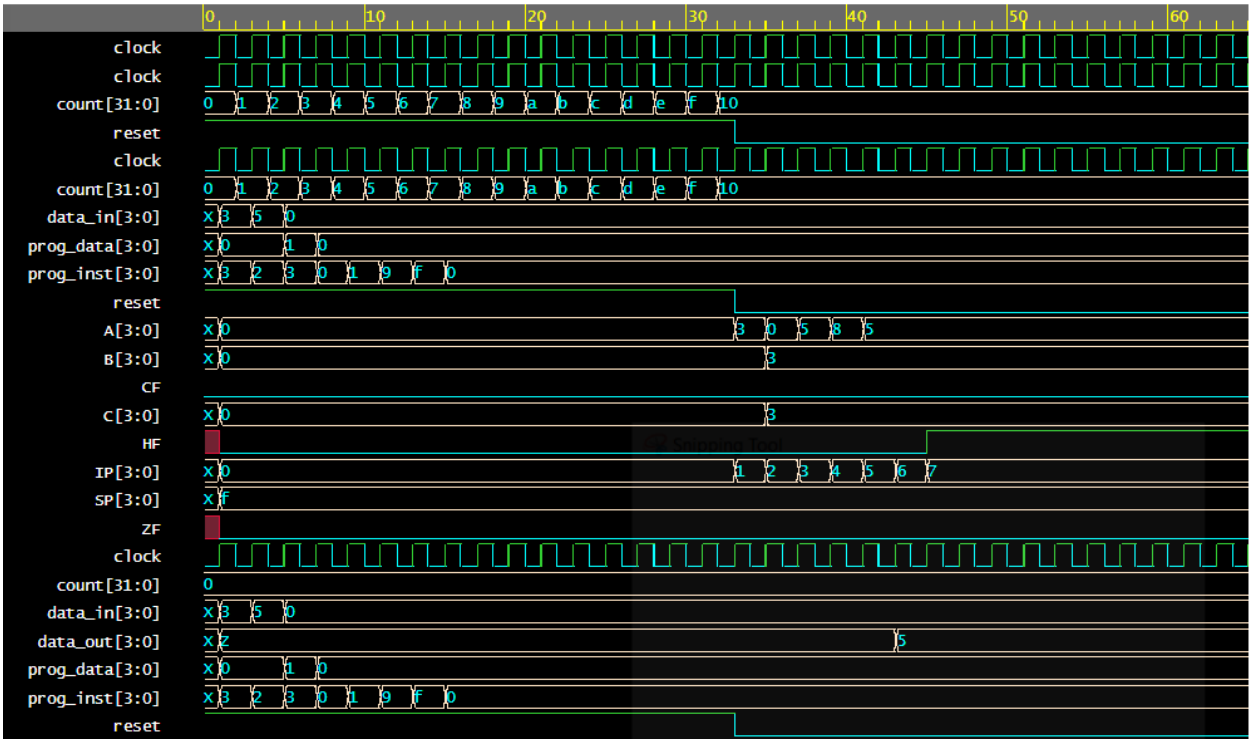
```
        else if (prog_inst_mem[IP] == 10)
        begin                                   //JMP ADDRESS
          IP = prog_data_mem[IP];
        end
        else if (prog_inst_mem[IP] == 11)
        begin                                   // PUSH A
            stack[SP] = A;
            SP = SP - 1;
            if (A == 0) begin
                        ZF = 1;
                        end
            end
        else if (prog_inst_mem[IP] == 12)
        begin                                   //POP A
          SP = SP + 1;
          A = stack[SP];
          stack[SP] = 4'd0;
          if (A == 0) begin
                        ZF = 1;
                        end
        end
        else if (prog_inst_mem[IP] == 13)
        begin                                   //CALL ADDRESS
          stack[SP] = IP;
          SP = SP - 1;
          IP = prog_data_mem[IP];
        end
        else if (prog_inst_mem[IP] == 14)
        begin
          SP = SP + 1;                          //RET
          IP = stack[SP];
        end
        else if (prog_inst_mem[IP] == 15)
        begin                                   //HLT
            HF = 1;
        end
        IP = IP + 1;                            //updating instruction pointer
      end
    end

endmodule
```

Testbench Code:

```
module testbench;
 reg clock = 0;
 reg reset = 1;
 reg [3:0] data_in,prog_inst,prog_data,prog_count;
 wire [3:0] data_out;
 reg [3:0] prog_inst_mem[15:0];
 reg [3:0] prog_data_mem[15:0];
 reg [3:0] data_mem[15:0];

 four_bit_comp test(clock,reset,data_in,prog_inst,prog_data,prog_count,data_out);

 integer count;

        parameter NONE = 4'b0000;

        parameter ADD_A_B      = 4'd0;        //defining assembly keys
        parameter SUB_A_B      = 4'd1;        //with machine code
        parameter XCHG_B_A     = 4'd2;
        parameter MOV_A_ADD    = 4'd3;
        parameter MOV_ADD_B    = 4'd4;
        parameter JNZ_ADD      = 4'd5;
        parameter XOR_A_ADD    = 4'd6;
        parameter PUSHF        = 4'd7;
        parameter IN_B         = 4'd8;
        parameter OUT_A        = 4'd9;
        parameter JMP_ADD      = 4'd10;
        parameter PUSH_A       = 4'd11;
        parameter POP_A        = 4'd12;
        parameter CALL_ADD     = 4'd13;
        parameter RET          = 4'd14;
        parameter HLT          = 4'd15;

        initial begin
        data_mem[0] = 4'd3;                   //data for storing in data memory
        data_mem[1] = 4'd5;
        data_mem[2] = 4'd0;
        data_mem[3] = 4'd0;
        data_mem[4] = 4'd0;
        data_mem[5] = 4'd0;
        data_mem[6] = 4'd0;
        data_mem[7] = 4'd0;
        data_mem[8] = 4'd0;
        data_mem[9] = 4'd0;
        data_mem[10] = 4'd0;
        data_mem[11] = 4'd0;
         data_mem[12] = 4'd0;
        data_mem[13] = 4'd0;
        data_mem[14] = 4'd0;
        data_mem[15] = 4'd0;
```

```
        prog_inst_mem[0] = MOV_A_ADD; prog_data_mem[0] = 4'd0;          //assebly code
        prog_inst_mem[1] = XCHG_B_A;  prog_data_mem[1] = NONE;
        prog_inst_mem[2] = MOV_A_ADD; prog_data_mem[2] = 4'd1;
        prog_inst_mem[3] = ADD_A_B;   prog_data_mem[3] = NONE;
        prog_inst_mem[4] = SUB_A_B;   prog_data_mem[4] = NONE;
        prog_inst_mem[5] = OUT_A;     prog_data_mem[5] = NONE;
        prog_inst_mem[6] = HLT;       prog_data_mem[6] = NONE;
        prog_inst_mem[7] = NONE;      prog_data_mem[7] = NONE;
        prog_inst_mem[8] = NONE;      prog_data_mem[8] = NONE;
        prog_inst_mem[9] = NONE;      prog_data_mem[9] = NONE;
        prog_inst_mem[10] = NONE;     prog_data_mem[10] = NONE;
        prog_inst_mem[11] = NONE;     prog_data_mem[11] = NONE;
        prog_inst_mem[12] = NONE;     prog_data_mem[12] = NONE;
        prog_inst_mem[13] = NONE;     prog_data_mem[13] = NONE;
        prog_inst_mem[14] = NONE;     prog_data_mem[14] = NONE;
        prog_inst_mem[15] = NONE;     prog_data_mem[15] = NONE;

  end


 initial begin
  $dumpfile("dump.vcd");
  $dumpvars;

  for (count = 0;count < 16; count = count+1) begin
   # 1 clock = 0;
   data_in = data_mem[count];
   prog_inst = prog_inst_mem[count];
   prog_data = prog_data_mem[count];
   prog_count = count;
   # 1 clock = 1;

  end

  # 1 reset = 0;
  # 32 $finish;
 end
 always #1 clock = !clock;
endmodule
```

Waveform:



It can be observed from the waveform that, at the end of the program, output port "data_out[3:0]" gave 5 as output which matches with the expected output.