Assignment on 4-bit computer

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Course Number: EEE 415

Course Title: Microprocessor and Embedded Systems

Prepared For:

Dr. Sajid Muhaimin Chowdhury

Assistant Professor, Department of EEE, BUET

Prepared By:

Md Adnan Faisal Hossain

Student ID **:** 1606063

Level **:** 4 **Term:** 1

Department **:** EEE

Section: A2

Date of Submission: 01/04/2021

Problem Statement:

Implement a 4-bit computer in Verilog HDL with the given instruction set.

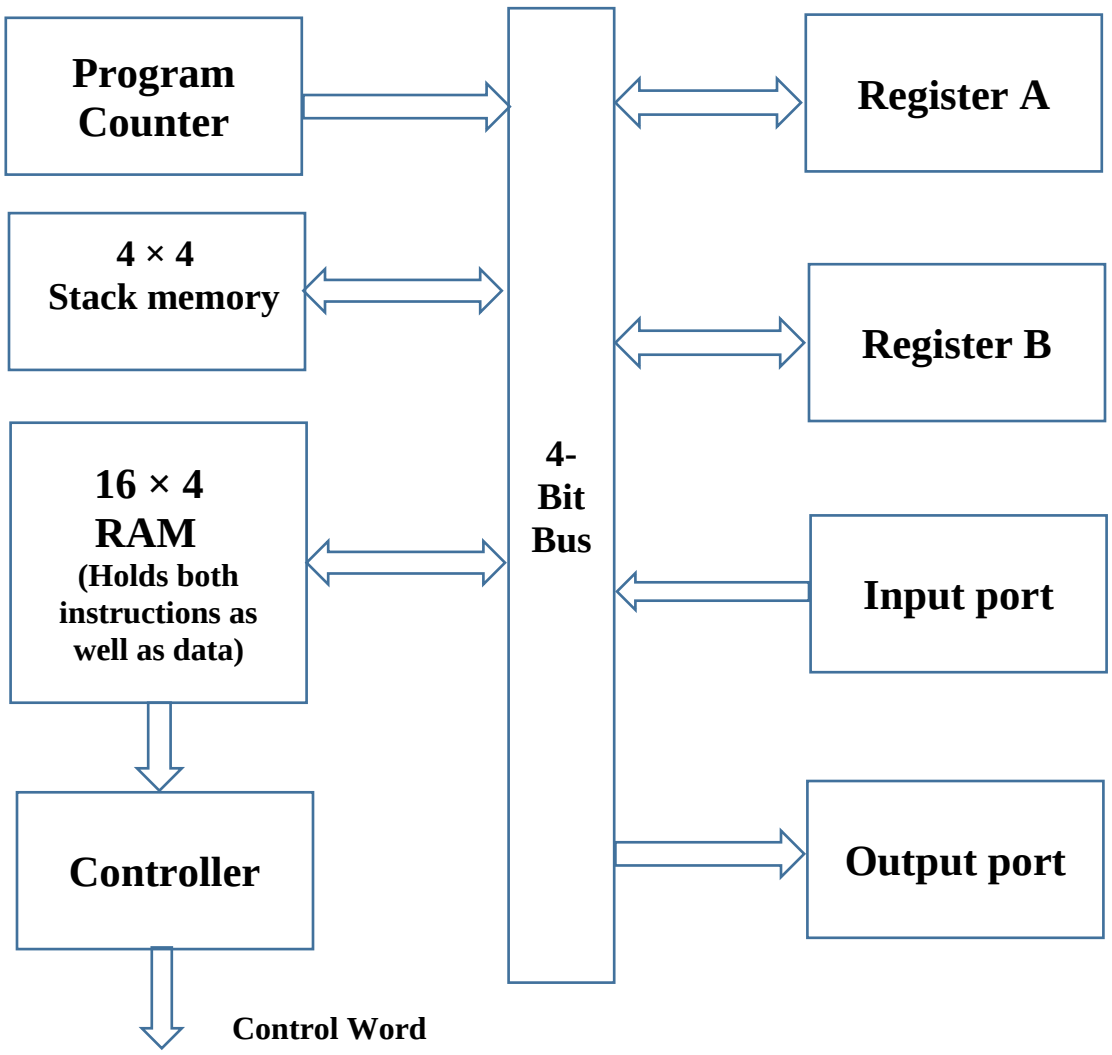
ID = 1606063

Therefore, assigned instruction set with explanation is given below:

Roll XX3		
1	ADD A, B	The values in registers A and B are added and stored in A. If the value of summation is greater than 15 (maximum possible 4-bit number) overflow flag OF is set to 1. If the result of summation is zero, zero flag ZF is set to 1.
2	SUB A, B	The value in register B is subtracted from the value of register A and stored in A. If the result of subtraction is a negative number (B>A) then sign flag SF is set to 1. If the result of subtraction is zero, ZF is set to 1.
3	XCHG A, B	The value in register A is transferred to register B, and the value in register B is transferred to register A.
4	IN A	The value at the input port is transferred to register A.
5	OUT A	The value at register A is transferred to the output port.
6	INC A	The value of register A is increased by 1 and the result is stored in A. If the new value is greater than 15, then OF is set to 1.
7	MOV A, [ADDRESS]	The value at the given ADDRESS location is transferred to register A.
8	MOV A, BYTE	Register A is assigned the value which BYTE is set to.
9	JZ ADDRESS	If ZF is set to 1 then set the program counter (instruction pointer) to the value ADDRESS.
10	PUSH B	Transfer the value of register B to the Stack at the location where the current stack pointer is pointed. Simultaneously increment the stack pointer by 1.
11	POP B	Transfer the value at the location which the stack pointer is currently pointing to, to register B. Simultaneously decrement the stack pointer by 1. If B is equal to zero, set ZF to 1.
12	RCL B	Rotate with carry left by 1 unit the value stored in register B, and then store the result in register B.
13	CALL ADDRESS	Store the current value of the program counter in the stack, and set the program counter to the value ADDRESS.

14	RET	Set the program counter to the current value in the stack at which stack pointer is pointed.
15	AND A, [ADDRESS]	Perform bitwise AND operation between the value in A register and the value at the location address, and then store the result in register A. If the result is equal to zero, then set ZF to 1.
16	HLT	Terminate the program and set halt flag to 1. Set the instruction pointer and stack pointer to zero. Clear all the memory in registers A and B, and in RAM and stack memory.

Block Diagram of the computer:



Verilog Code:

```
module Computer(clock, port_in, port_out, A, B, temp, PC, PC_temp, IP, SP, OF, ZF, SF, HLT);

    input clock;                            //system clock
    input [3:0]port_in;                     //input port 1
    output reg [3:0]port_out;               //output port 1
    output reg [3:0]A, B, temp;             //Registers A and B
    output reg [3:0] PC, PC_temp, IP;       //Program counter PC and instruction IP
    output reg [1:0]SP;                     //stack pointer SP
    reg [3:0]RAM [0:15];                    //(16x4) RAM. First 12 words contain instructions. Last 4 words contain data.
    reg [3:0]Stack [0:3];                   //(4x4) Stack memory to contain data
    reg [3:0]address, const_byte;           //register to hold ADDRESS and BYTE
    output reg OF, ZF, SF, HLT;             //overflow flag OF, zero flag ZF, sign flag SF, halt HLT
    integer i;                              //loop variable

    initial
    begin
        //Initialize counters, pointers, registers and flags to zero
        PC = 0;
        PC_temp = 0;
        IP = 0;
        SP = 0;
        A = 4'b0000;
        B = 4'b0000;
        OF = 0;
        ZF = 0;
        SF = 0;
        HLT = 0;

        //Instruction memory (Currently no instructions have been uploaded to the program)
        RAM[0][3:0] = 0;
        RAM[1][3:0] = 0;
        RAM[2][3:0] = 0;
        RAM[3][3:0] = 0;
        RAM[4][3:0] = 0;
        RAM[5][3:0] = 0;
        RAM[6][3:0] = 0;
        RAM[7][3:0] = 0;
        RAM[8][3:0] = 0;
        RAM[9][3:0] = 0;
        RAM[10][3:0] = 0;
        RAM[11][3:0] = 0;
        //Data memory (Currently no data is stored in the memory)
        RAM[12][3:0] = 0;
        RAM[13][3:0] = 0;
        RAM[14][3:0] = 0;
        RAM[15][3:0] = 0;

        //Initializing stack memory to zero
        for (i=0; i<4; i=i+1)
        begin
            Stack[i][3:0] = 4'b0000;
        end

        address = 4'bxxxx;          //assumed value of ADDRESS (different value assumed for each program)
        const_byte = 4'bxxxx;       //assumed value of BYTE (different value assumed for each program)

    end


    always @(posedge clock)                     //The instructions of the program are performed at positive clock edge
    begin
        if (HLT == 0)                           //Perform the program instructions only if Halt Flag is equal to zero.
        begin
            if (IP == 4'b000)                   //Line 1 executed if instruction has machine code = 0000
                begin                           //ADD A, B
                ADD(A, B, temp, OF);
                A = temp;
                if (A==0)   ZF = 1;             //Set ZF to 1 if A is equal to zero.
                end

            else if (IP == 4'b0001)             //Line 2 executed if instruction has machine code = 0001
                begin                           //SUM A, B
                SUB(A, B, temp, SF);
                A = temp;
                if (A==0)   ZF = 1;
                end

            else if (IP == 4'b0010)             //Line 3 executed if instruction has machine code = 0010
                begin                           //XCHG A, B
                temp = A;
                A = B;
                B = temp;
                end

            else if (IP == 4'b0011)             //Line 4 executed if instruction has machine code = 0011
                begin                           //IN A
                A = port_in;
                end

            else if (IP == 4'b0100)             //Line 5 executed if instruction has machine code = 0100
                begin                           //OUT A
                port_out = A;
```

```
93                  end
94
95              else if (IP == 4'b0101)             //Line 6 executed if instruction has machine code = 0101
96                  begin                           //INC A
97                  A = A + 1;
98                  if (A>15)
99                      OF =1;
100                 end
101
102             else if (IP == 4'b0110)             //Line 7 executed if instruction has machine code = 0110
103                 begin                           //MOV A, [ADDRESS]
104                 A = RAM[address];
105                 if (RAM[address]>15)
106                     OF = 1;
107                 end
108
109             else if (IP == 4'b0111)             //Line 8 executed if instruction has machine code = 0111
110                 begin                           //MOV A, BYTE
111                 A = const_byte;
112                 if (const_byte>15)
113                     OF = 1;
114                 end
115
116             else if (IP == 4'b1000)             //Line 9 executed if instruction has machine code = 1000
117                 begin                           //JZ ADDRESS
118                 if (ZF==1)
119                     PC_temp = address;
120                 end
121
122             else if (IP == 4'b1001)             //Line 10 executed if instruction has machine code = 1001
123                 begin                           //PUSH B
124                 Stack[SP] = B;
125                 SP = SP + 1;
126                 end
127
128             else if (IP == 4'b1010)             //Line 11 executed if instruction has machine code = 1010
129                 begin                           //POP B
130                 SP = SP - 1;
131                 B = Stack[SP];
132                 if (B==0)   ZF = 1;
133                 end
134
135             else if (IP == 4'b1011)             //Line 12 executed if instruction has machine code = 1011
136                 begin                           //RCL B
137                 temp = {B[2:0], B[3]};
138                 B = temp;
139                 if (B==0)   ZF = 1;
140                 end
141
142             else if (IP == 4'b1100)             //Line 13 executed if instruction has machine code = 1100
143                 begin                           //CALL ADDRESS
144                 Stack[SP] = PC;
145                 SP = SP + 1;
146                 PC_temp = address;
147                 end
148
149             else if (IP == 4'b1101)             //Line 14 executed if instruction has machine code = 1101
150                 begin                           //RET
151                 SP = SP - 1;
152                 PC_temp = Stack[SP];
153                 end
154
155             else if (IP == 4'b1110)             //Line 15 executed if instruction has machine code = 1110
156                 begin                           //AND A, [ADDRESS]
157                 temp = A & RAM[address];
158                 A = temp;
159                 if (A==0)   ZF = 1;
160                 end
161
162             else                                //Line 16 executed if instruction has machine code = 1111
163                 HLT = 1;                        //HLT
164         end
165
166         else if (HLT == 1)                      //If Halt Flag is equal to 1, clear registers and memory
167         begin                                   //and set stack pointer to zero
168             SP = 0;
169             A = 4'b0000;
170             B = 4'b0000;
171             temp = 4'b0000;
172             for (i=0; i<16; i=i+1)
173             begin
174                 RAM[i][3:0] = 4'b0000;
175             end
176             for (i=0; i<4; i=i+1)
177             begin
178                 Stack[i][3:0] = 4'b0000;
179             end
180         end
181
182     end
183
184
```

```
185     always @(negedge clock)                    //Increment program counter and and obtain instruction from memory
186     begin                                      //on each negative clock edge
187         if (HLT==0)
188         begin
189             if ((IP == 4'b1000 && ZF == 1) || IP == 4'b1100 || IP == 4'b1101)
190                 begin
191                 PC = PC_temp;
192                 IP = RAM[PC];
193                 PC = PC + 1;
194                 end
195             else
196                 begin
197                 IP = RAM[PC];
198                 PC = PC + 1;
199                 end
200         end
201         else                                   //if HLT is equal to 1, set program counter and instruction to zero
202         begin
203             PC = 0;
204             IP = 0;
205         end
206     end
207
208
209     task ADD;                                  //task to perform addition operation including OF flag
210
211         input [3:0]A,B;
212         output reg [3:0]SUM;
213         output reg OF;
214
215         {OF,SUM} = A + B;
216
217     endtask
218
219
220     task SUB;                                  //task to perform subtraction operation including SF flag
221
222         input [3:0]A;
223         input [3:0]B;
224         output reg [3:0]DIFF;
225         output reg SF;
226
227         if (A>=B)
228             begin
229             DIFF = A - B;
230             SF = 0;
231             end
232         else
233             begin
234             DIFF = B - A;
235             SF = 1;
236             end
237
238     endtask
239
240
241 endmodule
242
243
244
```

We created a module with two registers A and B, a 16x4 RAM and a 4x4 Stack memory. The first 12 words of the RAM are meant to hold instructions while the last 4 words are meant to hold data. Since we were asked to build a 4-bit computer, we restrained ourselves to only 4-bit data. Therefore, the RAM only contains ($16 = 2^4$) memory locations, and each memory location can contain only 4-bit data. We needed to perform the 16 operations given in our instruction set. So all the possible 16 (2^4) binary numbers were assigned as Op-codes for the instructions. We restricted ourselves to 4 bits, and therefore we could not assign any Op-codes to extract data from memory locations, as that would require the use of 8 bit instructions. The code for the module is explained above with necessary comments.

Test Programs to test the operations of the computer:

Test Code – 1

Assume address = 14 and byte = 10.

```
MOV A, [address]
XCHG B, A
MOV A, byte
SUB A, B
ADD A, B
PUSH B
RCL B
POP B
HLT
```

Instruction and Data section of memory in the Verilog code:

```
//Instruction memory
RAM[0][3:0] = 6;
RAM[1][3:0] = 2;
RAM[2][3:0] = 7;
RAM[3][3:0] = 1;
RAM[4][3:0] = 0;
RAM[5][3:0] = 9;
RAM[6][3:0] = 11;
RAM[7][3:0] = 10;
RAM[8][3:0] = 15;
RAM[9][3:0] = 4'bxxxx;
RAM[10][3:0] = 4'bxxxx;
RAM[11][3:0] = 4'bxxxx;
//Data memory
RAM[12][3:0] = 0;
RAM[13][3:0] = 0;
RAM[14][3:0] = 6;
RAM[15][3:0] = 0;

//Initializing stack memory to zero
for (i=0; i<4; i=i+1)
begin
      Stack[i][3:0] = 4'b0000;
end

address = 14;
const_byte = 10;
```

Vector waveform file:

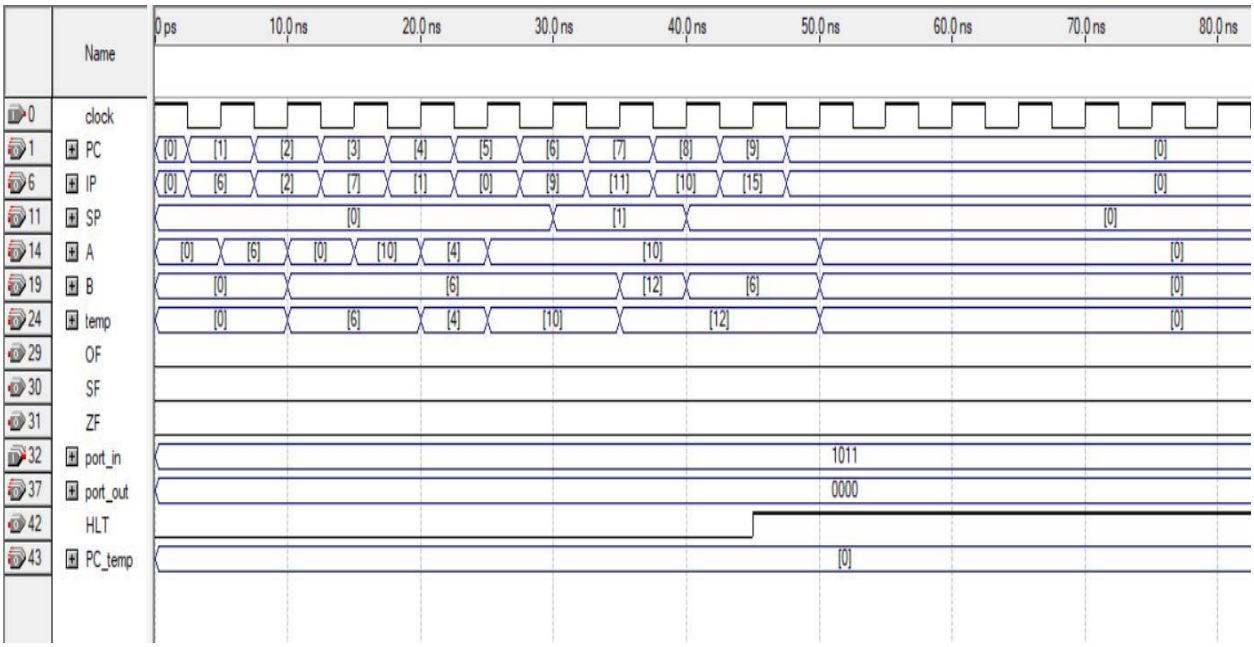


Figure: Output for Test Code 1

In this test code we tested the operations MOV, XCHG, ADD, SUB, PUSH, POP, RCL and HLT. We assumed the value of ADDRESS to be 14, the value stored at address location 14 be 6 and BYTE to be 10. We can see in the waveforms above that on each negative edge of clock program counter PC increments by 1 and the corresponding instruction that the instruction pointer is pointing at also changes. We can see the order of the instructions is 6, 2, 7, 1, 0, 9, 11, 10, 15 which corresponds to our Assembly Language.

MOV A, [ADDRESS] (instruction 6H) - When IP = 6, at the first positive edge of clock register A is set to the value of 6 which is the value stored in address location EH (14).

XCHG B, A (instruction 2H) – When IP = 2, at the first positive edge of clock register A is set to 0 (previous value of register B) and register B is set to 6 (previous value of register A).

MOV A, BYTE (instruction 7H) – When IP = 7, at the first positive edge of clock, register A is set to the value of BYTE (10).

SUB A, B (instruction 1H) – When IP = 1, at the first positive edge of clock, A is subtracted from B (10 – 6) and the result (4) is stored in A. B is unchanged.

ADD A, B (instruction 0H) – When IP = 0, at the first positive edge of clock, A is added to B (4 + 6) and the result (10) is stored in A. B is unchanged.

PUSH B (instruction 9H) – When IP = 9, at the first positive edge of clock, the value of register A is pushed to the stack. We can see SP changes to 1. So, stack pointer now points to memory location 1H of stack memory from 0H.

RCL B (instruction BH) – When IP = 11, at the first positive clock edge, the value of B rotates to the left by 1 unit and changes from 6 (0110B) to 12 (1100B).

POP B (instruction AH) – When IP = 10, at the first positive clock edge, the value at them stack memory at which the stack pointer is pointing (6) is passed to B. The stack pointer is decremented by 1 from 1 to 0.

HLT (instruction FH) – When IP = 15, at the first positive edge of clock, HLT is set to 1 indicating the end of program. At the next negative clock edge PC, IP and SP are set to 0, and at the following positive clock edge the RAM, stack and registers A and B are cleared.

Test Code – 2

Assume address = 6, input = 8 and byte = 10.

```
IN A
XCHG B, A
MOV A, 8H
SUB A, B
JZ 6H
RCL B
INC A
OUT A
HLT
```

Instruction and Data section of memory in the Verilog code:

```
//Instruction memory
RAM[0][3:0] = 3;
RAM[1][3:0] = 2;
RAM[2][3:0] = 7;
RAM[3][3:0] = 1;
RAM[4][3:0] = 8;
RAM[5][3:0] = 11;
RAM[6][3:0] = 5;
RAM[7][3:0] = 4;
RAM[8][3:0] = 15;
RAM[9][3:0] = 4'bxxxx;
RAM[10][3:0] = 4'bxxxx;
RAM[11][3:0] = 4'bxxxx;
```

```
//Data memory
RAM[12][3:0] = 0;
RAM[13][3:0] = 0;
RAM[14][3:0] = 0;
RAM[15][3:0] = 0;

//Initializing stack memory to zero
for (i=0; i<4; i=i+1)
begin
      Stack[i][3:0] = 4'b0000;
end

address = 6;
const_byte = 8;
```

Vector waveform file:

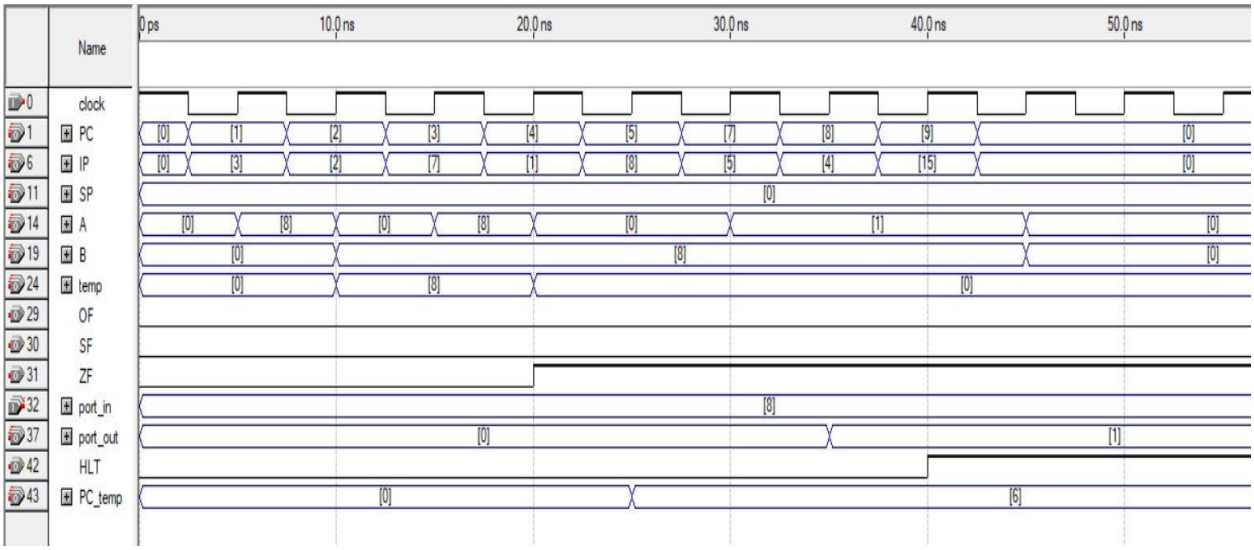


Figure: Output for Test Code 2

In this test code we tested the operations IN, OUT and INC. We also checked how the ZF flag changes. We assumed the value of ADDRESS to be 6, input to be 8 and BYTE to be 8. We can see in the waveforms above that on each negative edge of clock program counter PC increments by 1 and the corresponding instruction that the instruction pointer is pointing at also changes. We can see the order of the instructions is 3, 2, 7, 1, 8, 11, 5, 4, 15 which corresponds to our Assembly Language.

IN A (instruction 3H) – When IP = 3, at the first positive clock edge, the value at the input port (8) is passed to the register A.

ZF – When IP = 1, initially both A and B have the same value of 8. At the first positive clock edge SUB A, B is performed. Hence the value of A changes to 0. Simultaneously, ZF is updated to 1.

INC A (instruction 5H) – When IP = 5, at the first positive clock edge, the value of register A increments by 1 and changes from 0 to 1.

OUT A (instruction 4H) – When IP = 4, at the first positive clock edge, the value of register A (1) is passed to the output port.

Test Code – 3

Assume address = 8, input = 7 and byte = 11.

```
MOV A, BYTE
XCHG B, A
IN A
CALL ADDRESS
OUT A
HLT

ADD A, B
RET
```

Instruction and Data section of memory in the Verilog code:

```
//Instruction memory
RAM[0][3:0] = 7;
RAM[1][3:0] = 2;
RAM[2][3:0] = 3;
RAM[3][3:0] = 12;
RAM[4][3:0] = 4;
RAM[5][3:0] = 15;
RAM[6][3:0] = 4'bxxxx;
RAM[7][3:0] = 4'bxxxx;
RAM[8][3:0] = 0;
RAM[9][3:0] = 13;
RAM[10][3:0] = 4'bxxxx;
RAM[11][3:0] = 4'bxxxx;
```

```
//Data memory
RAM[12][3:0] = 0;
RAM[13][3:0] = 0;
RAM[14][3:0] = 0;
RAM[15][3:0] = 0;

//Initializing stack memory to zero
for (i=0; i<4; i=i+1)
begin
      Stack[i][3:0] = 4'b0000;
end

address = 8;
const_byte = 11;
```

Vector waveform file:

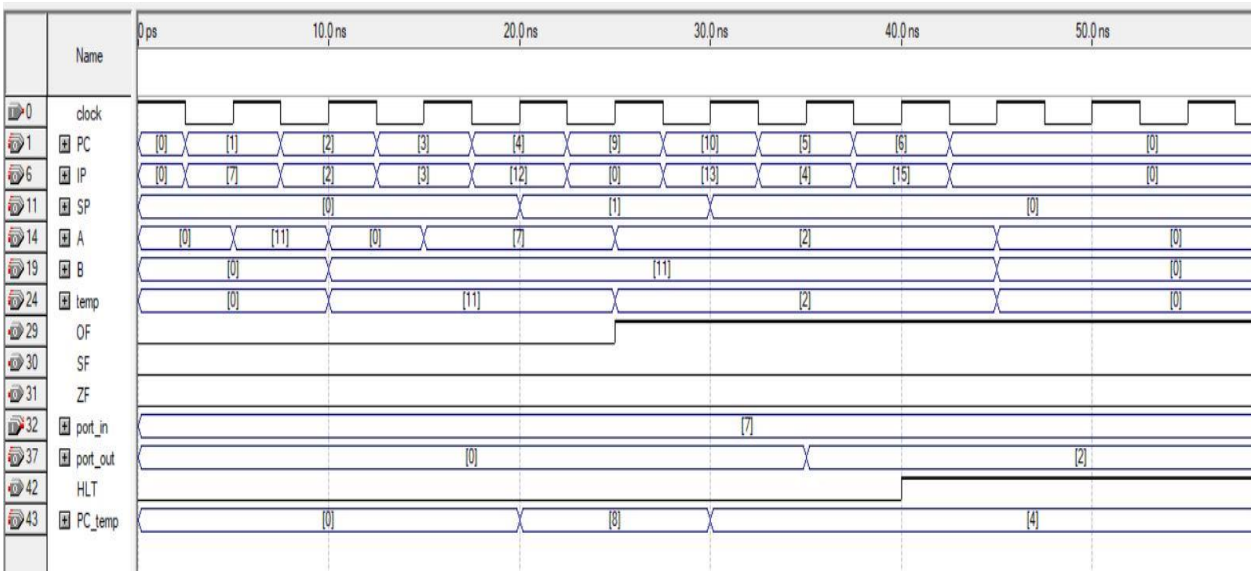


Figure: Output for Test Code 3

In this test code we tested the operations CALL and RET. We also checked how the OF flag changes. We assumed the value of ADDRESS to be 8, input to be 7 and BYTE to be 11. We can see in the waveforms above, that on each negative edge of clock, program counter PC increments by 1 and the corresponding instruction that the instruction pointer is pointing at also changes. We can see the order of the instructions is 7, 2, 3, 12, 0, 13, 4, 15 which corresponds to our Assembly Language.

CALL ADDRESS (instruction CH) – When IP = 12, at the first positive clock edge the current value of the PC (4) gets stored in the stack and the SP gets updated to 1. The PC value gets updated to the value ADDRESS. The instruction corresponding to that value (0) gets performed.

OF – When IP = 0, at the first positive clock edge ADD A, B (7+11) is performed. As the result is greater than 15, we cannot store it in a 4-bit number and hence the OF flag is updated to 1.

RET (instruction DH) – When IP =13, at the first positive edge of clock, the PC is set to the value at the stack memory at which the SP is currently pointed (4), and the SP is decremented to 0 form 1.

Test Code – 4

Assume address = 15, input = 11 and byte = 14.

```
MOV A, BYTE
XCHG B, A
MOV A, [ADDRESS]
SUB A, B
IN A
AND A, [ADDRESS]
HLT
```

Instruction and Data section of memory in the Verilog code:

```
//Instruction memory
RAM[0][3:0] = 7;
RAM[1][3:0] = 2;
RAM[2][3:0] = 6;
RAM[3][3:0] = 1;
RAM[4][3:0] = 3;
RAM[5][3:0] = 14;
RAM[6][3:0] = 15;
RAM[7][3:0] = 4'bxxxx;
RAM[8][3:0] = 4'bxxxx;
RAM[9][3:0] = 4'bxxxx;
RAM[10][3:0] = 4'bxxxx;
RAM[11][3:0] = 4'bxxxx;


//Data memory
RAM[12][3:0] = 0;
RAM[13][3:0] = 0;
RAM[14][3:0] = 0;
RAM[15][3:0] = 12;
```

```
//Initializing stack memory to zero
for (i=0; i<4; i=i+1)
begin
      Stack[i][3:0] = 4'b0000;
end

address =15;
const_byte = 14;
```

Vector waveform file:

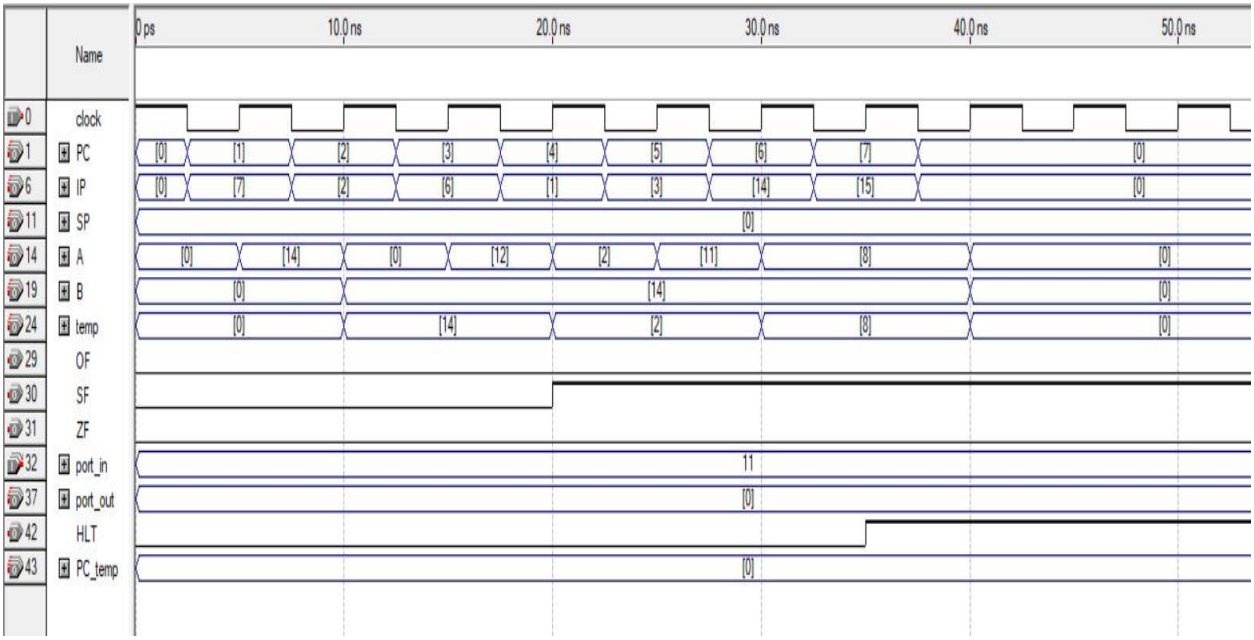


Figure: Output for Test Code 4

In this test code we tested the operation AND. We also checked how the SF flag changes. We assumed the value of ADDRESS to be 15, input to be 11 and BYTE to be 14. We can see in the waveforms above, that on each negative edge of clock, program counter PC increments by 1 and the corresponding instruction that the instruction pointer is pointing at also changes. We can see the order of the instructions is 7, 2, 6, 1, 3, 4, 15 which corresponds to our Assembly Language.

AND A, [ADDRESS] (instruction EH) – When IP = 14, at the first positive edge of clock, bitwise AND is performed between A (11 = 1011B) and the value at location ADDRESS (12 = 1100B). The result (1011B & 1100B = 1000B = 8) is stored in A.

SF – When IP = 11, at the first positive clock edge B is subtracted from A (12 – 14). As the result is negative, the SF flag is updated to 1.

Assembler:

An assembler was implemented to convert Assembly Language to Machine Language using Python.

Python Code:

```
assembly_file = open ('Assembly Test Code 2.txt', 'r')
machine_file = open ('Machine Test Code 2.txt', 'w')


assembly_str = assembly_file.read()
assembly_list = assembly_str.split('\n')
print(assembly_list)


for item in assembly_list:
    if 'ADD' in item:
        machine_file.write('0000\n')
    elif 'SUB' in item:
        machine_file.write('0001\n')
    elif 'XCHG' in item:
        machine_file.write('0010\n')
    elif 'INC' in item:
        machine_file.write('0101\n')
    elif 'IN' in item:
        machine_file.write('0011\n')
    elif 'OUT' in item:
        machine_file.write('0100\n')
    elif 'MOV' in item:
        if '[' in item:
            machine_file.write('0110\n')
        else:
            machine_file.write('0111\n')
    elif 'JZ' in item:
        machine_file.write('1000\n')
    elif 'PUSH' in item:
        machine_file.write('1001\n')
    elif 'POP' in item:
        machine_file.write('1010\n')
    elif 'RCL' in item:
        machine_file.write('1011\n')
    elif 'CALL' in item:
        machine_file.write('1100\n')
    elif 'RET' in item:
```

```
        machine_file.write('1101\n')
    elif 'AND' in item:
        machine_file.write('1110\n')
    elif 'HLT' in item:
        machine_file.write('1111\n')

assembly_file.close()
machine_file.close()
```

Examples of how the assembler works:

Test code 1 (Assembly Language)

```
MOV A, [14]
XCHG B, A

MOV A, 10
SUB A, B
ADD A, B

PUSH B
RCL B
POP B

HLT
```

Test code 1 (Machine Language generated using Assembler)

```
1010
0010
0111
0001
0000
1001
1011
1010
1111
```

Test code 2 (Assembly Language)

```
IN A
XCHG B, A
MOV A, 8H

SUB A, B
JZ 6H

RCL B
INC A

OUT A

HLT
```

Test code 2 (Machine Language generated using Assembler)

```
0011
0010
0111
0001
1000
1011
0101
0100
1111
```

Conclusion:

The 4-bit computer was successfully implemented using Verilog HDL. The code was written in Quartus II software. Vector waveform files to test the performance of the computer were also generated using Quartus II. Different test programs were used to test the performance of the code and it was seen that the computer is successfully able to perform all the tasks given in the instruction set. An assembler was also created using Python to convert Assembly code to Machine code.