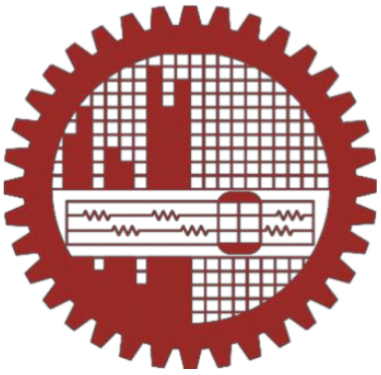
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING



EEE – 415
Microprocessor And Embedded System

Continuous Assessment 1
4 bit Computer Using VerilogHDL

Submitted By:
Name : Prasun Datta
Level 4/Term 1
Student's ID : 1606148
Section: C1

Submitted To:
Dr.Sajid Muhaimin Choudhury
Asst. Professor, Department of EEE, BUET

Videolink:
https://web.microsoftstream.com/video/44503f2d-303b-4f06-b3a4-282c5807a93e

Introduction:

4 bit pc is a very basic model of a microprocessor which is motivated by SAP(simple-as-possible-1) architecture explained by Malvino. 4 bit pc accomplishes all its instruction using 4 bit computing method.In computer architecture, 4-bit integers, memory addresses, or other data units are those that are 4 bits wide. Also, 4-bit CPU and ALU architectures are those that are based on registers, address buses, or data buses of that size. A group of four bits is also called a nibble and has $2^4 = 16$ possible values. Its primary purpose is to develop a basic understanding of how a microprocessor works, interacts with memory and other parts of the system like input and output. The instruction set is very limited and is simple.

Explanation of Instruction Set:

Here goes the name of the instruction set that is implemented in this design.

Number of Instruction	Instruction Name
1	ADD A,B
2	SUB A,B
3	XCHG B,A
4	RCL B
5	SHR A
6	MOV A, [ADDRESS]
7	XOR A,[ADDRESS]
8	AND A,B
9	OR B,[ADDRESS]
10	OUT A
11	JZ ADDRESS
12	PUSH B
13	POP B
14	CALL ADDRESS
15	RET
16	HLT

1. **ADD A,B:**

   This instruction indicates that A,B are summed and their summation is stored in the A register. In this task, addition is performed accordingly and to show the summation the result is assigned to an output register. The carry out is directly assigned to the carry flag. Hence status of the carry flag gets updated with the summation task. Moreover if the summation becomes zero, status of the zero flag becomes 1.

2. **SUB A,B**

   This instruction indicates that B is subtracted from A and the result is stored in A. For executing this block, subtraction operation is performed accordingly. If there is any carry, it is assigned to carry flag variable.Here also if output of subtraction becomes zero, zero flag updates its value.

3. **XCHG B,A**

   It's basically a swap operation.In this case values of A & B will be exchanged between each other. For executing this , one temp variable is called and value of A is assigned to that variable. And B is placed in the address of A. Finally temp is placed in the address of B. For verilog execution , we have simply taken two variables called mypc_outA & mypc_outB and accordingly placed the values of B & A in them.

4. **RCL B**

   Here RCL stands for rotate carry left. This instruction basically changes the bit pattern of B. Initially carry flag is 0. Since this is rotation towards left, the MSB will replace the carry flag(CF) . Eventually whole bit pattern will be shifted towards left leaving the LSB position empty.And finally, the value stored in CF(which was MSB bit) will go in the place of LSB.

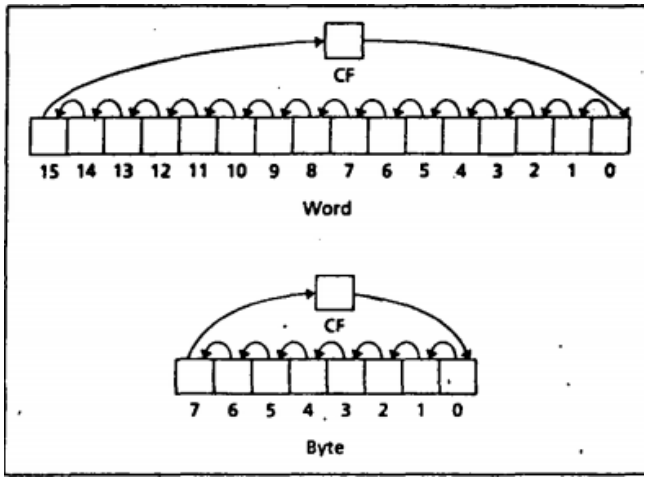


Fig : RCL instruction

5. **SHR A:**

   SHR stands for shift right. Here the whole bit batten of A will be shifted towards right direction. Hence the LSB bit will go in the place of CF. Hence position of MSB bit, being empty, will be filled by zero.

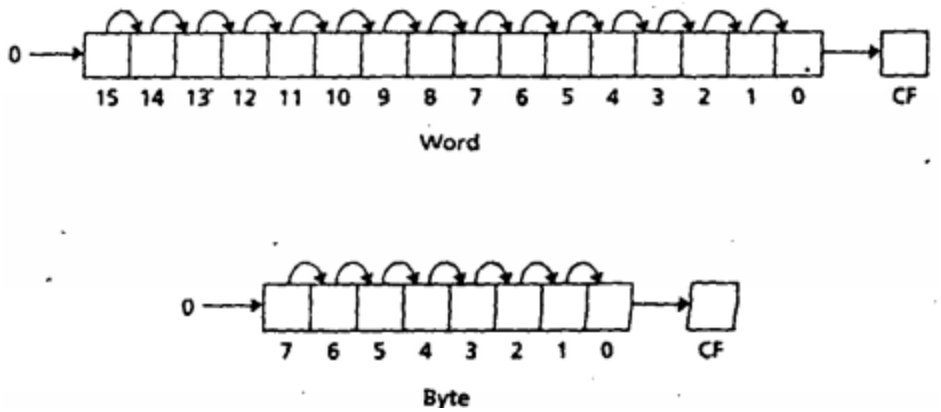


Fig : Illustration of SHR

6. MOV A, [ADDRESS]

This assembly instruction basically moves the content of the ADDRESS to the A register.For this purpose, a 2D RAM is created which is basically 16*4 in size. More precisely, RAM has 16 unique addresses where each address can hold 4 bit content.Eventually, some values are stored in the RAM in the sequential order from 0000 to 1111. Now whenever the clock gives positive edge, value of address counter is increased one.Eventually, this address counter is used as an index to access the corresponding value stored in RAM.Hence during execution of thic block, value of address counter is observed and it is used to access the corresponding value which eventually is moved to mypc_outA variable.

7. XOR A ,[ADDRESS]

This instruction executes the bitwise xor operation between input A and the content of the ADDRESS. To accomplish this, value of address counter is observed and this counter is used to read corresponding value from RAM. After that, XOR operation is performed between A and the value achieved from memory.

8. AND A, B

This instruction simply indicates the logical bitwise AND operation between A & B.and their result is saved in A. In this task, to observe the value of A, the output is assigned to mypc_outA.

9. OR B,[ADDRESS]

This instruction executes the bitwise or operation between input B and the content of the ADDRESS. To accomplish this, value of address counter is observed and this counter is used to read corresponding value from RAM. After that, OR operation is performed between B and the value achieved from memory.

10. OUT A

This operation simply transfers the value of A to the output register. For accomplishing this, value of A is transferred to mypc_outA variable.

11. JZ ADDRESS

JZ stands for jump if zero. Here if the zero flag(ZF) becomes 1, the instruction pointer will jump to the corresponding value of address counter.

12. PUSH B

This instruction basically performs the push operation. It stores the value of B to the

stack memory so that we can use it later if we want.For implementing this, a 2D stack memory was declared having size of 16*4 which means that there will be 16 memory address each having bit depth of 4.So value of B is stored in the stack memory and eventually after the push operation the value of stack pointer(SP) is decreased by one.Moreover to show the status of the stack, another output variable called stack_output is declared which basically indicates the current value pushed or popped in the stack memory.Again if B is zero, ZF becomes 1.

13. POP B

This basically takes out the value from the stack memory that was previously pushed.Now to pop B , value of SP should be increased by one so that it indicates the memory where B was pushed. SP is used as an index to read the value of B from the stack memory.After the pop operation, stack memory becomes void if we previously pushed just one value.

14. CALL ADDRESS

This instruction basically calls a specific address to perform some operation. We can track the value of instruction pointer (IP) to understand this operation. Here,at first value of IP which it was pointing before executing this block will be pushed in the stack memory. In this way we can restore this address whenever we need it.Eventually value of SP will be decreased by one. After that IP jumps to the address it was assigned in this block. In this way we can understand the operation of call by observing the updated value of IP.

15. RET

This operation is the exact reverse of the CALL operation. In this case, IP should point to the address before execution of CALL statement. For achieving this, address that was pushed to stack memory needs to be popped. Hence SP should increase by one. And now it will indicate to the address that was pushed. The value of this address is read through SP and finally IP is assigned that value. In this way, IP will regain the value it was pointing before tha CALL statement.

16. HLT

This operation basically stops the execution of the program. To achieve this, simply the stop_flag is assigned the value of one whereas in other cases it was zero. In this way, observing the value of zero flag, HLT instruction can be realized.

Architecture of 4 bit PC:

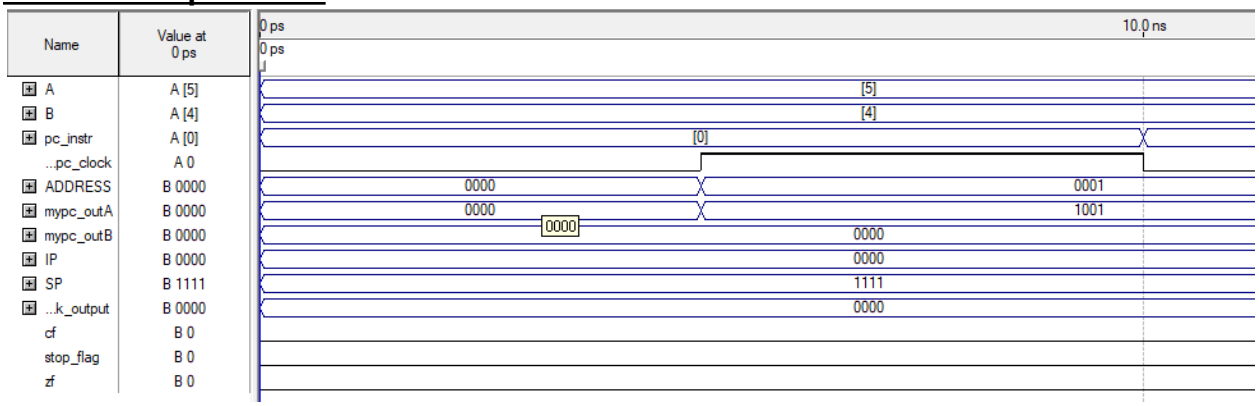
Here the architecture is pretty straightforward. Two inputs which basically mimic some input peripheral device were used as input port in this architecture. Eventually, two corresponding output ports were also used to observe the output state of A & B. In respect of memory,RAM and stack memory were made. These are basically 16*4 bit array. This means that 16 rows/ address will be available each with 4 bit depth. In the RAM some values of address are stored. To be precise, values ranging from 0000 to 1111 are stored in the corresponding address. Consequently, instruction pointer and stack pointer is also used to access values stored in RAM & stack memory. Now the execution block or the “Central Processing Block” consists of case modules. These modules perform specific task assigned to it. Infact “*pc_instr”* is the variable which decides the instruction to be performed by the computer. Since this pc was designed to perform 16 specific tasks,hence 16 separate case modules were built.

Unique Features: To give some uniqueness to this architecture, flag operation is included in this pc. One can observe the change of flag with the execution of various

instructions.In this case **carry flag and zero flag** are included. Carry flag basically handles the carry bit. If during some operation carry becomes one, then CF changes to 1 otherwise it remains zero. On the other hand, zero flag changes to 1 if somehow the value of some input data or the result becomes zero.

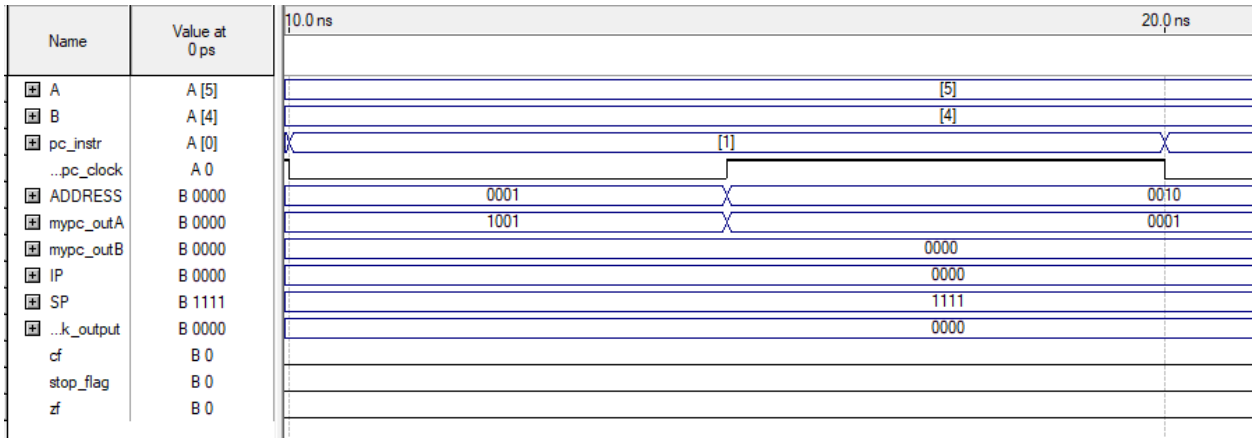
Waveforms & Corresponding Explanation:

1.Addition operation:



Here A = 5 (0101) in binary & B = 4 (0100) in binary.
During this operation,
ADDRESS = 0001 is the pointer basically.
Pc_instr = 0
Summation is stored in mypc_outA (9 / 1001)
Since there is no zero involved ZF = 0 .
Also no carry is generated from the summation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111.

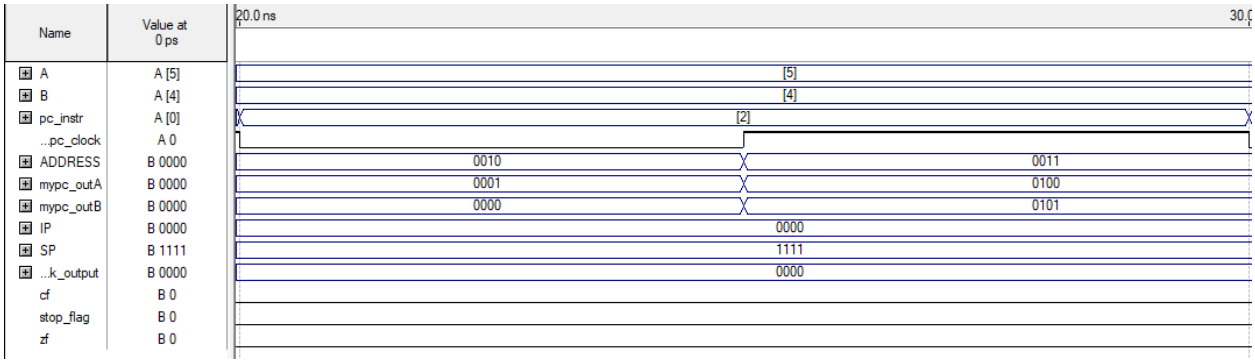
2. Subtraction operation:



Here A = 5 (0101) in binary & B = 4 (0100) in binary.
During this operation,
ADDRESS = 0010 is the pointer basically.
Pc_instr = 1
Subtarction is stored in mypc_outA (1 / 0001)

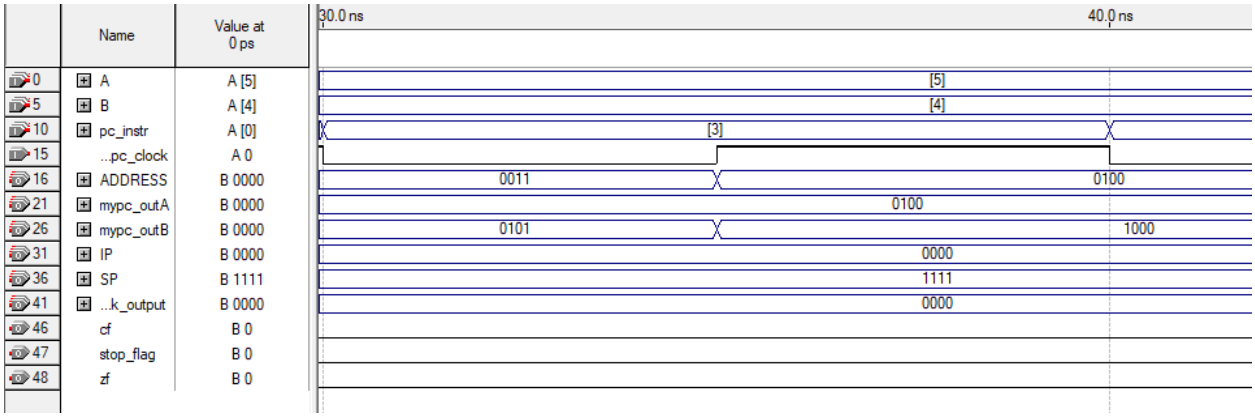
Since there is no zero involved ZF = 0 .
Also no carry is generated from the operation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111.

3. Exchange operation:



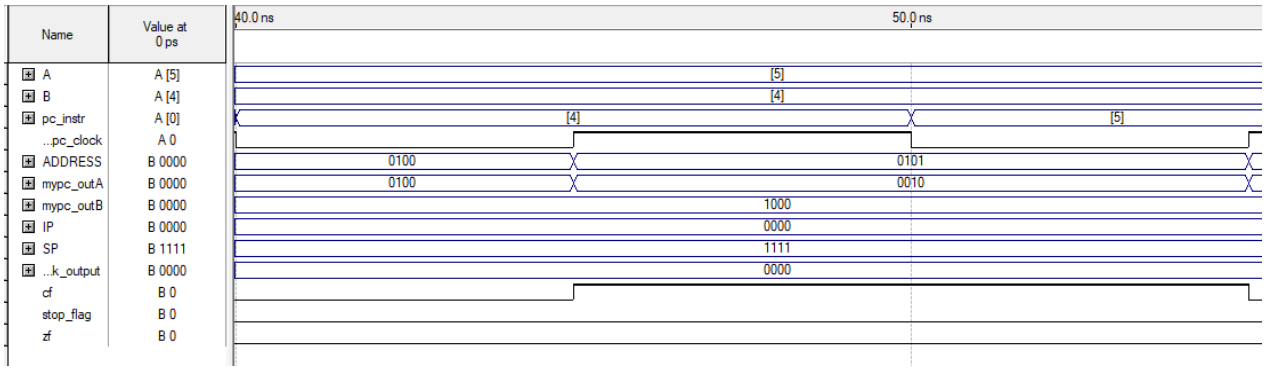
From this operation we can see that value of A(0101) is stored in mypc_outB and vice versa. Thus A and B swapped .

4. Rotate Carry Left Operation:



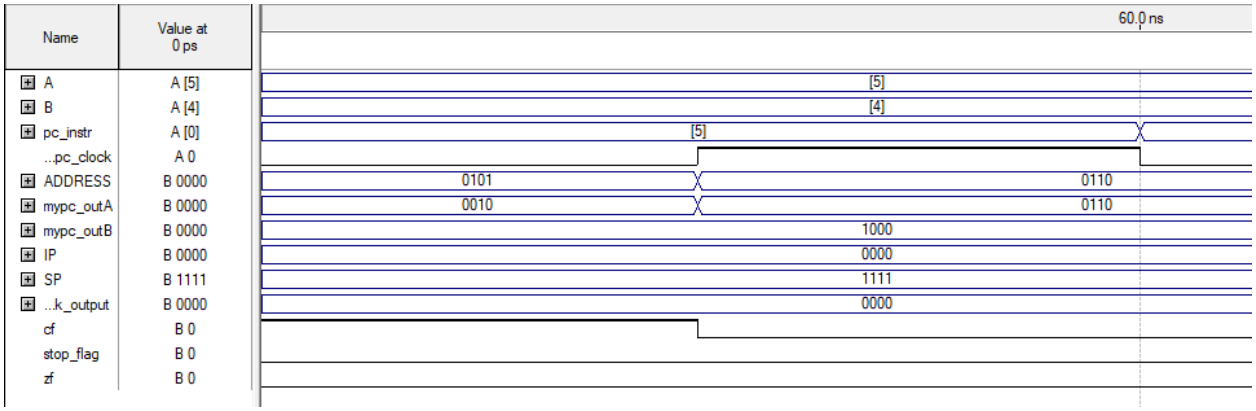
Here RCL operation is performed over B one time and the result is stored in mypc_outB(1000). Since there is no zero involved ZF = 0 .
Also no carry is generated from the operation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111.

5. Shift Right A:



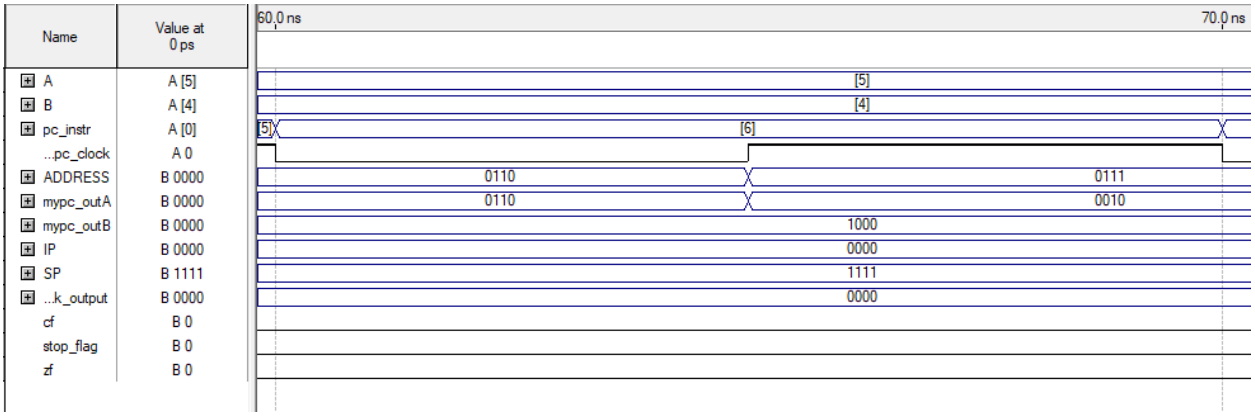
Here SHR operation is performed over A one time. Hence from the bit pattern of A we can see that LSB is 1. As such CF will be 1 now which is evident in the waveform picture. And the shifted bit pattern which is 0010, will be stored in mypc_outA.Since there is no zero involved ZF = 0.

6. Mov operation:



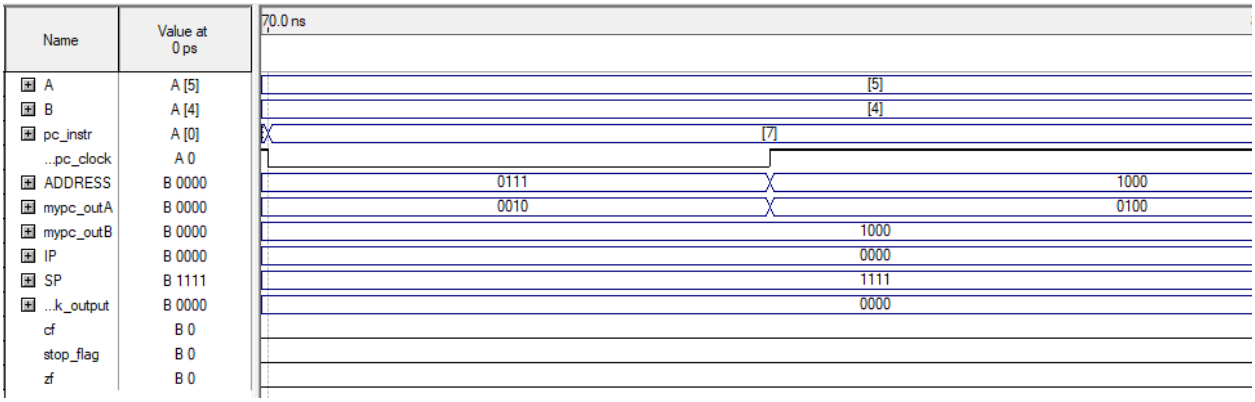
During this operation value of address counter is 0110. And the value stored in RAM is read through this address counter using it as an index. And that value is also 0110. Finally this 0110 is moved to the mypc_outA.
Since there is no zero involved ZF = 0 .
Also no carry is generated from the operation hence CF again becomes zero.
IP was initialized to zero and stack was initialized to its maximum value 1111

7. **XOR operation:**



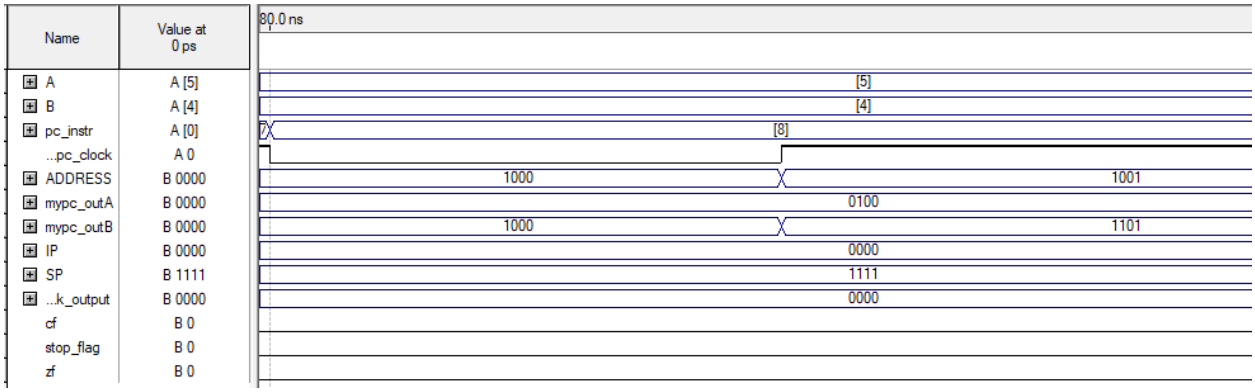
In this case, XOR operation is performed between A and the value stored in RAM at the index position of 0111. Here the value stored in RAM in this position in 0111.
Now 0111 xor 0101 returns 0010 bit pattern and it is stored in mypc_outA.Since there is no zero involved ZF = 0 . Also no carry is generated from the operation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111.

8. **AND Operation**



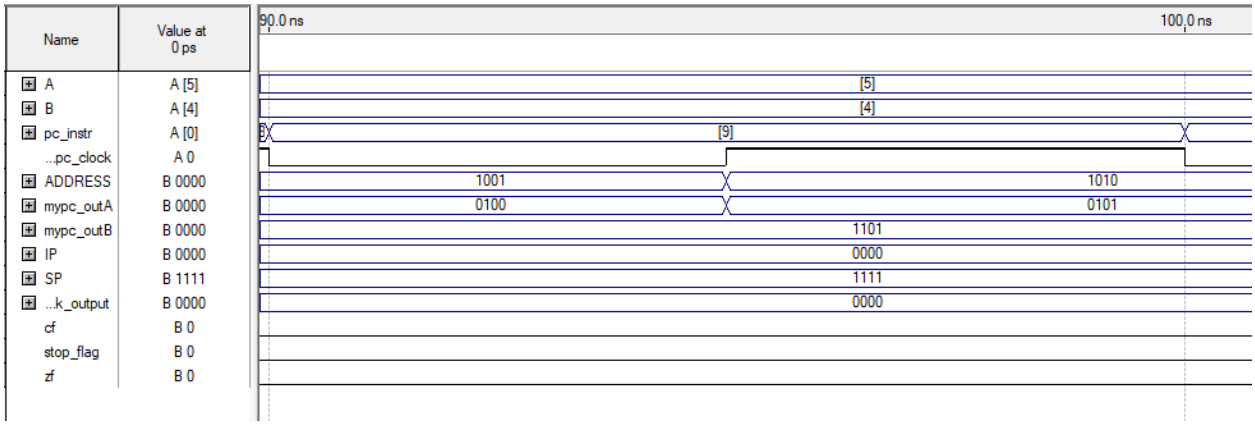
Here bitwise and is performed.
A = 0101
B = 0100 .Now A and B = 0100 and this is stored in mypc_outA.Since there is no zero involved ZF = 0 . Also no carry is generated from the operation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111

9. **OR operation :**



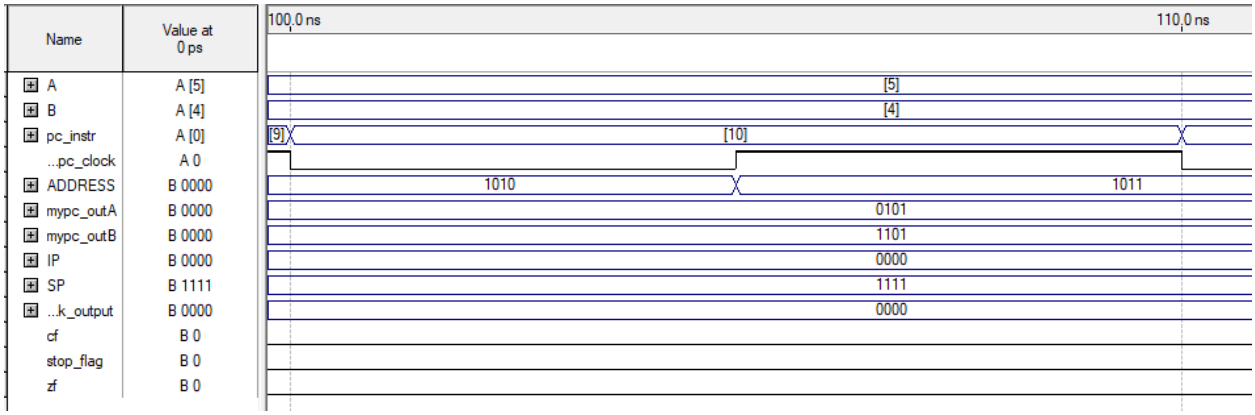
Here content of ADDRESS(1001) is 1001. B = 0100
So 1001 OR 0100 = 1101 and it is stored in mypc_outB.Since there is no zero involved ZF = 0 . Also no carry is generated from the operation hence CF =0
IP was initialized to zero and stack was initialized to its maximum value 1111

10 .**Output operation:**



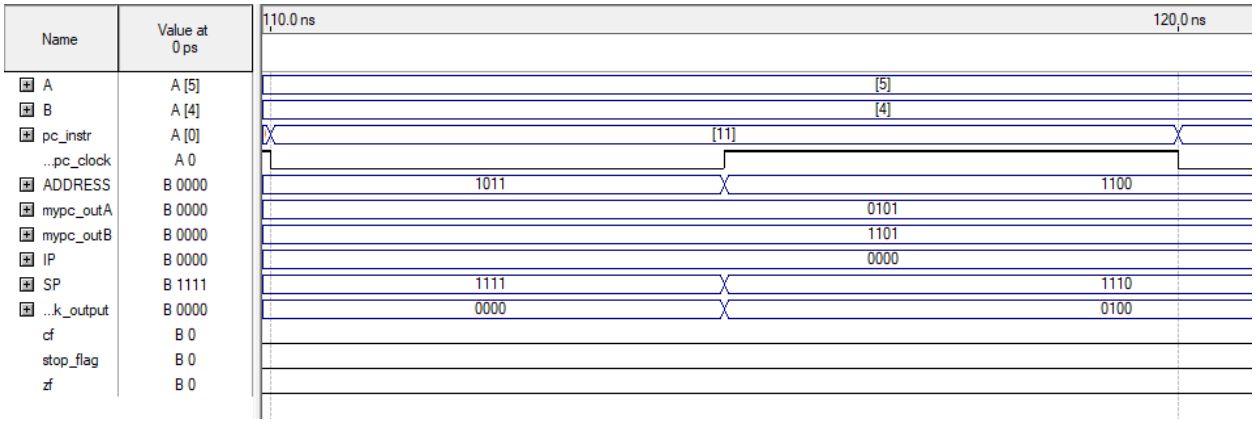
Here simply content of A is transferred to the output variable called mypc_outA(0101). Since there is no zero involved ZF = 0

11. Jump if zero



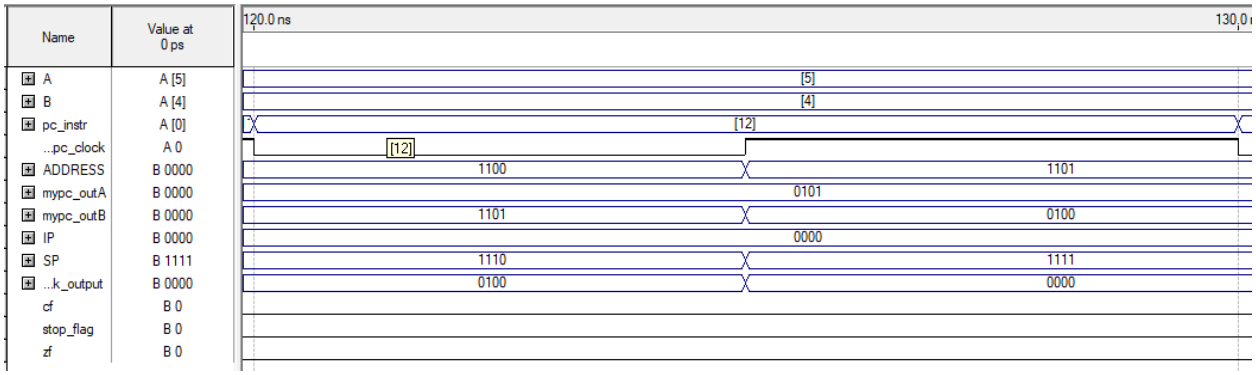
Here we didn't get any zero and hence the zero flag remains zero. Hence the value of IP was not updated. If we got zero somehow, the value of IP would load the value of ADDRESS (1011).

12**. PUSH Operation:**



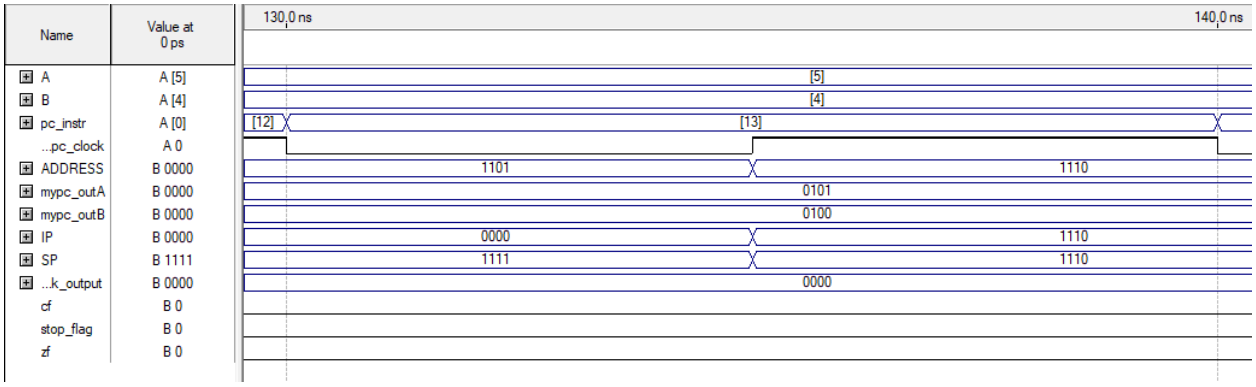
In this operation, value of B = 0100 is pushed in the stack memory. The variable named stack_output is highlighting the value of B . Also the value of SP has decreased from 1111 to 1110 as it should be. No zero is involved in this operation so ZF remains 0. CF is also 0.

1**3. POP Operation:**



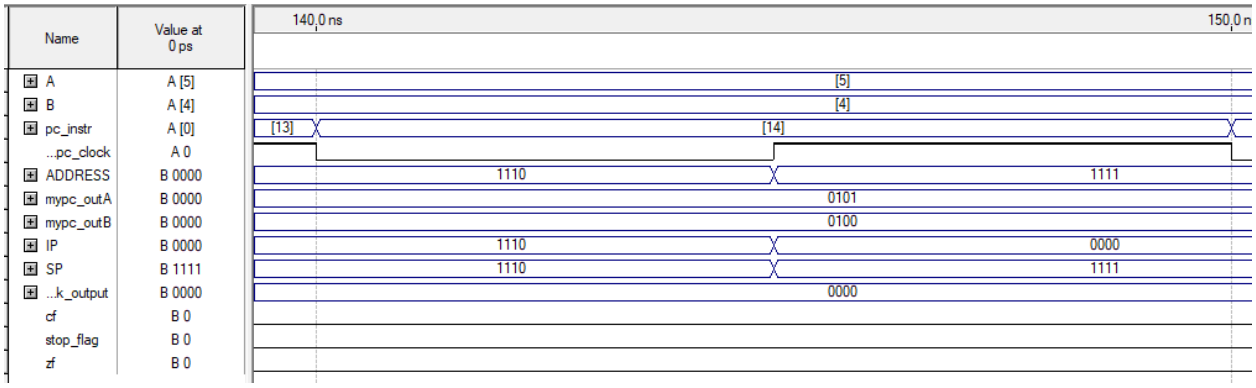
Here the value of B is popped out of the stack memory and the stack becomes zero again or void. This can be seen from the transition of stack_output variable. This variable becomes 0000 from 0100. Also after the pop operation SP should increase its value. And this is evident from values of SP as it has increased to 1111 from 1110.No zero is involved in this operation so ZF remains 0. CF is also 0.

1**4. CALL Address operation:**



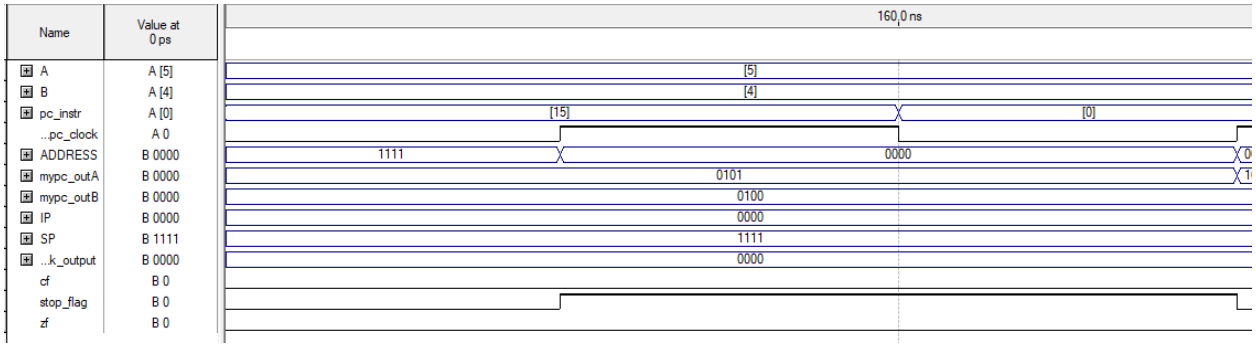
Here ADDRESS variable becomes 1110 during execution of this operation. To accomplish the task IP should now jump to the current ADDRESS location. From the picture, its clear that IP is now pointing at 1110 location instead of 0000 location. This previously indicated location of IP is stored in stack memory which can be realized from the value of stack_output.And finally SP has decreased also from 1111 to 1110.

15. Return operation:



In this operation IP should retain its value(0000) before the CALL operation. In the picture, its clear that IP has retained its value of 0000 from the stack memory . It has basically retrieved its value from the stack. Hence IP is now pointing at the address it was previously pointing.Since some values are taken out of the stack memory, SP has increased from 1110 to 1111.

16. HLT Operation



This operation basically stops the execution of program.And it is accomplished by stop_flag. Since instruction 15 takes the flow of program in this module, our program should stop by raising the value of stop_flag to 1. And its clearly visible from the picture .

Conclusion:

This 4 bit pc is executed using the verilog programming language, This pc mainly performs some defined specific task. Therefore more functionality could be given to this pc. The architecture of this pc is inspired from SAP. Therefore, to increase its computational power, task handling capability, more sophisticated operation could be defined through verilogHDL language.

Appendix:

Code Snippet :

```
// ************4 Bit Computer Using Verilog******************//

module
My_computer4bit(A,B,pc_instr,mypc_clock,mypc_outA,mypc_outB,stop_flag,ADDRE
SS,stack_output,IP,SP,zf,cf);
        input [3:0]A; //data input A of size 4 bit
        input [3:0]B; // data input B of size 4 bit
        input [3:0]pc_instr; // 4bits are for instruction to be executed
        input mypc_clock; // clock pulse to control the overall functions of the pc

        reg [3:0]mypc_ram[15:0]; // RAM of size 16*4;16 rows each with bit depth of
4
        reg[3:0]mypcMEM_STACK[15:0]; // stack memory of size 16*4

        output reg[3:0]mypc_outA; //variable to show output states of A
        output reg[3:0]mypc_outB; // variable to show output states of B

        output reg stop_flag; // flog used to indicate the HLT instruction
        output reg[3:0]ADDRESS=0;
        output reg[3:0]stack_output=0; // variable to show the state of stack

        output reg [3:0]IP =0; //instruction pointer initialized to zero.
        output reg [3:0]SP =15; // stack pointer initialized to the last location

        output reg zf; // zero flag
        output reg cf; // carry flag

        integer k;
        parameter n = 16;
```

```
always @(posedge mypc_clock) // instruction execution block
begin

        stop_flag =0;
        zf = 0;
        cf  = 0;

        for(k=0;k<=n-1;k = k+1)
        begin
                mypc_ram[k] =k; // assigning values of addresses to the memory
block.
        end

        ADDRESS = ADDRESS +1;

   // instruction by instruction execution

         case(pc_instr)
                0:begin
                        {cf,mypc_outA}=A+B; //Add instruction
                        if (mypc_outA==0)
                                zf=1;
                  end

                1:begin
                        {cf,mypc_outA}=A-B; // subtract instruction
                        if (mypc_outA==0)
                                zf=1;
            end
                2:begin   // XCHG instruction here we have done it in this fashion to
see the output
                     // actually it should be temp=A; A=B;B=temp;
                        mypc_outA=B;
                        mypc_outB=A;
            end
                3:begin
                        mypc_outB={B[2:0],B[3]};//RCL  B; rotate carry left B
                        cf = B[3];
                        if (mypc_outB==0)
                                zf=1;
            end
                4:begin
                        cf=A[0];                //SHR A(shift right A)
                        mypc_outA[0]=A[1];
                        mypc_outA[1]=A[2];
                        mypc_outA[2]=A[3];
                        mypc_outA[3]=0;
                  end
```

```
5:begin
            mypc_outA=mypc_ram[ADDRESS]; //MOV A,[ADDRESS]
            if(mypc_ram[ADDRESS]==0)
                  zf = 1;
      end
      6:begin
            mypc_outA=A^mypc_ram[ADDRESS]; //XOR A,[ADDRESS]
            if(mypc_outA==0)
                  zf = 1;
end

      7:begin
            mypc_outA=A&B;       //AND A,B
            if (mypc_outA==0)
                  zf=1;
      end
      8:begin
            mypc_outB=B|mypc_ram[ADDRESS];       //OR B,[ADDRESS]
            if (mypc_outB==0)
                  zf=1;
      end

      9:begin
            mypc_outA = A;     //out A
            if (mypc_outA==0)
                  zf=1;
      end
      10:begin
            if(zf==1)       //JZ  ADDRESS
                  IP= ADDRESS;
       end
      11:begin
            mypcMEM_STACK[SP]=B;    //push B
            stack_output=mypcMEM_STACK[SP];
            SP=SP-1;
            if (B==0)
                  begin
                        zf=1;
                  end
            end
      12:begin
            SP=SP+1;
            mypc_outB=mypcMEM_STACK[SP];   //pop B
            stack_output=0;
            if (mypc_outB==0)
                  begin
                        zf=1;
                  end
       end
```

```
                13:begin
                        mypcMEM_STACK[SP]= IP; //call address
                        IP = ADDRESS;
                        SP = SP-1;
            end
                14:begin
                        SP=SP+1;
                        IP = mypcMEM_STACK[SP]; // return
        end

                15:begin
                        stop_flag=1;   //HLT
                  end

         endcase
end
endmodule
```