PROJECT – REPORT (Team Noobs)

-Satla Shashank (2021122003)
-Shubham Priyadarshan(2020102027)

Y-86_64 sequential implementation

We have implemented 6 moules namely **fetch(fetch.v)**, **decode(decode.v)**, **execute(execute_new.v)**, **memory(memory.v), write back (wb.v)** and **pc_update(pc_update.v)**.

For keeping things compact we have combined the decode and write back into a single function named "**dec_wb.v**".

Let us go over one by one.

Fetch module (fetch.v)

```
module fetch(clk, PC, rA , rB , iCode , iFun ,valC,valP,instr_Validity, imem_error,HF);

// PORT DECLARATION
parameter N = 511;
input [63:0] PC;
input clk;
output reg[3:0] iCode,iFun,rA,rB; // instruction
output reg[63:0] valC,valP;  // values to be assigned for op and PC update
output reg instr_Validity,imem_error,HF; // Validity and memory overflow flag and halt flag 'HF' to distinguish between
                                //                                    HALT and NOop.

reg [7:0] InstructionMemory [0:N]; // Can take 511 single byte instructions at a time.

reg [0:79]Instruction;

initial begin
   // //   //mrmovq $0x10, %rdx
  InstructionMemory[10]=8'b01000000; //4 0   PC = 10
  InstructionMemory[11]=8'b00000010; //rA=0 rB=2
  InstructionMemory[12]=8'b00000000;
  InstructionMemory[13]=8'b00000000;
  InstructionMemory[14]=8'b00000000;
  InstructionMemory[15]=8'b00000000;
  InstructionMemory[16]=8'b00000000;
  InstructionMemory[17]=8'b00000000;
  InstructionMemory[18]=8'b00000000;
  InstructionMemory[19]=8'b00010000; //V=16


  // OPq addq
  InstructionMemory[29]=8'b01100000; // 6 0  (add)   // PC = 29
  InstructionMemory[30]=8'b00010010; // 1 2  %rcx %rdx



  //jxx
  InstructionMemory[20]=8'b01110000; //7 0   (jmp)  // PC  = 20;
  InstructionMemory[21]=8'b00000000;
  InstructionMemory[22]=8'b00000000;
  InstructionMemory[23]=8'b00000000;
  InstructionMemory[24]=8'b00000000;
  InstructionMemory[25]=8'b00000000;
  InstructionMemory[26]=8'b00000000;
  InstructionMemory[27]=8'b00000000;
  InstructionMemory[28]=8'b00010000; //dest = 16

  //call
  InstructionMemory[31]=8'b10000000; // 80 (call) // PC =31
  InstructionMemory[32]=8'b00000000;
  InstructionMemory[33]=8'b00000000;
  InstructionMemory[34]=8'b00000000;
  InstructionMemory[35]=8'b00000000;
  InstructionMemory[36]=8'b00000000;
  InstructionMemory[37]=8'b00000000;
  InstructionMemory[38]=8'b00000000;
  InstructionMemory[39]=8'b01000000;  // dest


end



always@(posedge clk)
 begin

                Instruction = {InstructionMemory[PC],InstructionMemory[PC+1],
                               InstructionMemory[PC+2],InstructionMemory[PC+3],
                               InstructionMemory[PC+4],InstructionMemory[PC+5],
                               InstructionMemory[PC+6],InstructionMemory[PC+7],
                               InstructionMemory[PC+8],InstructionMemory[PC+9]
                               }; // Concatenating the Instruction memory to make a single 10 byte instruction.

    instr_Validity = 1'b1;
    if(PC>N)
    begin
        imem_error =1;  // Checking for Memory Overflow; assign 1 if TRUE;
    end

    else
    begin
           imem_error =0;  // Assign 0 if FALSE
    end
    iCode = Instruction[0:3];
    iFun  = Instruction[4:7];
    HF = 1'b0;
    if((iCode==4'b0000) & (iFun==4'b0000) ) // Halt ;
    begin
        valP = PC+64'd1;
        HF = 1'b1;
    end

    else if ((iCode==4'b0001)&(iFun==4'b0000)) begin
        valP = PC+64'd1;
    end

    else if((iCode==4'b0010)&(iFun<4'b0111)) // ccmovxx
    begin
        rA = Instruction[8:11];
        rB = Instruction[12:15];
        valP = PC + 64'd2;
    end

    else if ((iCode==4'b0011)&(iFun==4'b0000))//irmovq
    begin
        rA = Instruction[8:11];
        rB = Instruction[12:15];
        valC = Instruction[16:79];          // Stores immediate value to be moved
        valP = PC+64'd10;

    end

    else if((iCode==4'b0100)&(iFun==4'b0000)) //rmmovq
      begin
        rA = Instruction[8:11];
        rB = Instruction[12:15];
        valC = Instruction[16:79];          // Stores Displacement value
        valP = PC+64'd10;
      end

    else if ((iCode==4'b0101)&(iFun==4'b0000)) //mrmovq
    begin
        rA = Instruction[8:11];
        rB = Instruction[12:15];
        valC = Instruction[16:79];          // Stores Displacement value
        valP = PC+ 64'd10;
    end

    else if ((iCode==4'b0110) & (iFun<4'b0100))        //OPq
    begin
        if ((iFun==4'b0000))
        begin                                   //addq
            rA = Instruction[8:11];
            rB = Instruction[12:15];
            valP = PC + 64'd2;
        end

        else if (iFun==4'b0001)
        begin                                   //subq
            rA = Instruction[8:11];
            rB = Instruction[12:15];
            valP = PC + 64'd2;
        end

        else if (iFun==4'b0010)
        begin                                   //andq
            rA = Instruction[8:11];
            rB = Instruction[12:15];
            valP = PC + 64'd2;
        end

        else if (iFun==4'b0011)
        begin                                   //xorq
```

```
          rA = Instruction[8:11];
          rB = Instruction[12:15];
          valP = PC + 64'd2;
       end
   end

 else if ((iCode==4'b0111) & (iFun<4'b0111))          // jxx  Instruction size is of 9 bytes for jump intsruction.
   begin

          valC = Instruction[8:71];  // stores destination
          valP = PC + 64'd9;
   end

   else if ((iCode==4'b1000)&(iFun==4'b0000))        // call
   begin

          valC = Instruction[8:71];
          valP = PC + 64'd9;
   end

   else if((iCode==4'b1001)&(iFun==4'b0000))         //return (ret)
   begin
          valP = PC + 64'd1;
   end

   else if((iCode==4'b1010)&(iFun==4'b0000))         //pushq
   begin
          rA = Instruction[8:11];
          rB = Instruction[12:15];
          valP = PC + 64'd2;
   end

   else if((iCode==4'b1011)&(iFun==4'b0000))         //popq
   begin
          rA = Instruction[8:11];
          rB = Instruction[12:15];
          valP = PC + 64'd2;
   end

   else
   begin
          instr_Validity = 1'b0;                        // If any other combination is entered its invalid;
   end
 end

endmodule
```

The fetch module takes in PC and clk as inputs and outputs iCode, iFun, valP, valC along with a bunch of flags.

The fetch testbench yielded the following results.

```
clk=0 PC=          0 icode=xxxx ifun=xxxx rA= x rB= x,valC=          x,valP=          x,HF=x,imem_error=x,instr_Validity=x
clk=1 PC=         10 icode=0100 ifun=0000 rA= 0 rB= 2,valC=         16,valP=         20,HF=0,imem_error=0,instr_Validity=1
clk=0 PC=         10 icode=0100 ifun=0000 rA= 0 rB= 2,valC=         16,valP=         20,HF=0,imem_error=0,instr_Validity=1
clk=1 PC=         20 icode=0111 ifun=0000 rA= 0 rB= 2,valC=         16,valP=         29,HF=0,imem_error=0,instr_Validity=1
clk=0 PC=         20 icode=0111 ifun=0000 rA= 0 rB= 2,valC=         16,valP=         29,HF=0,imem_error=0,instr_Validity=1
clk=1 PC=         29 icode=0110 ifun=0000 rA= 1 rB= 2,valC=         16,valP=         31,HF=0,imem_error=0,instr_Validity=1
clk=0 PC=         29 icode=0110 ifun=0000 rA= 1 rB= 2,valC=         16,valP=         31,HF=0,imem_error=0,instr_Validity=1
clk=1 PC=         31 icode=1000 ifun=0000 rA= 1 rB= 2,valC=         64,valP=         40,HF=0,imem_error=0,instr_Validity=1
clk=0 PC=         31 icode=1000 ifun=0000 rA= 1 rB= 2,valC=         64,valP=         40,HF=0,imem_error=0,instr_Validity=1
```

The above result confirms that the fetch module works fine.

Decode and Write back (dec_wb.v)

The decode and writeback ***(dec_wb.v)*** module takes clk, iCode,rA,rB, valE, valM as input and gives valA, valB and updated register values as the output.

The code and the testbench results are below.

```
module dec_wb(clk,iCode,rA,rB,valA,valB,cnd,valE,valM,memreg0,memreg1,memreg2,memreg3,memreg4,memreg5,memreg6,memreg7,memreg8,
        memreg9,memreg10,memreg11,memreg12,memreg13,memreg14);

input clk,cnd;
input[3:0] iCode,rA,rB;
input[63:0] valE,valM;
output reg[63:0] valA,valB;

output reg[63:0] memreg0,memreg1,memreg2,memreg3,memreg4,memreg5,memreg6,memreg7,memreg8,
          memreg9,memreg10,memreg11,memreg12,memreg13,memreg14;

reg[63:0] memreg[0:14];

integer i;

initial begin
memreg[0] = 64'd0;

for(i=1;i<15;i=i+1)
begin
memreg[i] = memreg[i-1]+64'd1;
end
end

always@(*)
begin
if(iCode==4'b0010)
begin
valA = memreg[rA];
end

else if(iCode==4'b0100) //rmmovq
begin
valA<=memreg[rA];
valB<=memreg[rB];
end

else if(iCode==4'b0101) //mrmovq
begin
valB<=memreg[rB];
end

else if(iCode==4'b0110) //OPq
begin
valA<=memreg[rA];
valB<=memreg[rB];
end

else if(iCode==4'b1000) //call
begin
valB<=memreg[4]; //rsp
end

else if((iCode==4'b1001) || (iCode==4'b1011) ) //ret or popq
begin
valA<=memreg[4]; //rsp
valB<=memreg[4]; //rsp
end

else if(iCode==4'b1010) //pushq
begin
valA<=memreg[rA];
valB<=memreg[4]; //rsp
end



memreg0 <= memreg[0];
memreg1 <= memreg[1];
memreg2 <= memreg[2];
memreg3 <= memreg[3];
memreg4 <= memreg[4];
memreg5 <= memreg[5];
memreg6 <= memreg[6];
memreg7 <= memreg[7];
memreg8 <= memreg[8];
memreg9 <= memreg[9];
memreg10 <= memreg[10];
memreg11 <= memreg[11];
memreg12 <= memreg[12];
memreg13 <= memreg[13];
memreg14 <= memreg[14];
```

```
end


always @(negedge clk)
begin
if((iCode == 4'b0010) & (cnd==1'b1))
begin
memreg[rB] <= valE;
end


else if((iCode == 4'b0011) ||(iCode==4'b0110)) // for  irmovq and OPq
begin
memreg[rB] <= valE;
end

else if (iCode ==4'b0101)  // for mrmovq
begin
memreg[rA] <= valM;
end

else if((iCode>4'b0111) & (iCode < 4'b1011))  // for call ret and pushq
begin
memreg[4] <= valE;
end

else if(iCode == 4'b1011)  // for popq
begin
memreg[4] <= valE;
memreg[rA] <=valM;
end

memreg0 <= memreg[0];
memreg1 <= memreg[1];
memreg2 <= memreg[2];
memreg3 <= memreg[3];
memreg4 <= memreg[4];
memreg5 <= memreg[5];
memreg6 <= memreg[6];
memreg7 <= memreg[7];
memreg8 <= memreg[8];
memreg9 <= memreg[9];
memreg10 <= memreg[10];
memreg11 <= memreg[11];
memreg12 <= memreg[12];
memreg13 <= memreg[13];
memreg14 <= memreg[14];

end
endmodule
```

The test bench yielded the following results:

```
clk=0 icode=xxxx  rA= x rB= x,valA=          x,valB=          x,valE=          x,valM=          x
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          11,memreg12=          12,memreg13=          13,memreg14=          14

clk=1 icode=xxxx  rA= x rB= x,valA=          x,valB=          x,valE=          x,valM=          x
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          11,memreg12=          12,memreg13=          13,memreg14=          14

clk=0 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=1 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=0 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=1 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=0 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=1 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
,memreg10=          10,memreg11=          0,memreg12=          12,memreg13=          13,memreg14=          14

clk=0 icode=0110  rA= 2 rB=11,valA=          2,valB=          0,valE=          0,valM=          0
,memreg0=          0,memreg1=          1,memreg2=          2,memreg3=          3
,memreg4=          4,memreg5=          5,memreg6=          6,memreg7=          7,memreg8=          8,memreg9=          9
```

```
,memreg10=            10,memreg11=            0,memreg12=           12,memreg13=           13,memreg14=           14

clk=1 icode=0110  rA= 2 rB=11,valA=           2,valB=            0,valE=            0,valM=            0
,memreg0=            0,memreg1=            1,memreg2=            2,memreg3=            3
,memreg4=            4,memreg5=            5,memreg6=            6,memreg7=            7,memreg8=            8,memreg9=            9
,memreg10=            10,memreg11=            0,memreg12=           12,memreg13=           13,memreg14=           14

clk=0 icode=0110  rA= 2 rB=11,valA=           2,valB=            0,valE=            0,valM=            0
,memreg0=            0,memreg1=            1,memreg2=            2,memreg3=            3
,memreg4=            4,memreg5=            5,memreg6=            6,memreg7=            7,memreg8=            8,memreg9=            9
,memreg10=            10,memreg11=            0,memreg12=           12,memreg13=           13,memreg14=           14

clk=1 icode=0101  rA= 2 rB=11,valA=           2,valB=            0,valE=            0,valM=            0
,memreg0=            0,memreg1=            1,memreg2=            2,memreg3=            3
,memreg4=            4,memreg5=            5,memreg6=            6,memreg7=            7,memreg8=            8,memreg9=            9
,memreg10=            10,memreg11=            0,memreg12=           12,memreg13=           13,memreg14=           14

clk=0 icode=0101  rA= 2 rB=11,valA=           2,valB=            0,valE=            0,valM=            0
,memreg0=            0,memreg1=            1,memreg2=            0,memreg3=            3
,memreg4=            4,memreg5=            5,memreg6=            6,memreg7=            7,memreg8=            8,memreg9=            9
,memreg10=            10,memreg11=            0,memreg12=           12,memreg13=           13,memreg14=           14
```

The above result confirms that our dec_wb modules works fine.
Note: We tested our module for iCode = 6 and 5 only. We set the valE and valM to be 0.
The memregrA and memregrB values should be 0 accordingly.

Execute stage(execute_new.v)

The Inputs to this module are clk,icode,ifun,valA,valB,valC.
This module outputs cnd signal,valE,cc.
In this module we are performing ALU operations and also setting condition codes depending on instruction i.e icode and ifun.

Cnd signal-This signal is set for both cmove and jump instructions depending upon the condition codes set i.e zero flag,sign flag,overflow flag during Opq instruction execution.
This signal will be high whenever the codition(involving zero flag,sign flag,overflow flag) corresponding to the instruction is high.
cc-It is a condition code register,It stores the values of condition codes of most recent arithmetic instruction.
valE signal is generated depending on the type of instruction i.e icode and ifun and it is generally the output of ALU.

In this module we are using a Arithmatic and Logic Unit(ALU) to perform Arithmatic and Logic operations.
The inputs to the ALU module are icode,x,y,control.
This module outputs result,zf,sf,OF
Depending on icode it will check to wheather to calculate zf,sf,OF or not.Only for Opq instruction we set zf,sf,of.

```
module execute(clk,icode,ifun,valA,valB,valC,cnd,valE,cc);
//inputs Declaration
input  clk;
input [3:0] icode;
input [3:0] ifun;
input  signed [63:0] valA;
input  signed [63:0] valB;
input signed [63:0] valC;

//Outputs Declaration
output reg signed  [63:0] valE;
output reg cnd;
output reg [2:0] cc;

reg [3:0] icd;
reg signed [63:0] x;
reg signed [63:0] y;
reg [1:0]  control;
wire signed [63:0] z;
wire zf;
wire sf;
wire of;

reg not_input;
reg or_input1;
reg or_input2;
reg xor_input1;
reg xor_input2;

wire not_output;
wire or_output;
wire xor_output;


ALU_final alu(icd,x,y,control,z,zf,sf,of);
always @(z)
    valE=z;
always @(zf,sf,of)
    begin
        cc[0]=zf;
        cc[1]=sf;
        cc[2]=of;
    end




not notgate(not_output,not_input);
or  orgate(or_output,or_input1,or_input2);
xor xor_gate(xor_output,xor_input1,xor_input2);




always @(*)
    begin
        if(clk==1)
            begin
            cnd=0;
            //cmovxx
            if(icode==4'b0010)
                begin
                    x=valA;
                    y=valB;
                    icd=icode;
                    control=2'b00;
                    //rrmovq
                    if(ifun==4'b0000)
                    begin
                        cnd=1;
                    end
                    //cmovle
                    else if(ifun==4'b0001)
                        begin
                            // (sf^of)||zf
                            xor_input1=cc[1];
                            xor_input2=cc[2];
                            or_input1=xor_output;
                            or_input2=cc[0];
        if(or_output)
                                begin
                                    cnd=1;
                                end
                        end
```

```
//cmovl
                    else if(ifun==4'b0010)
                        begin
                            // sf^of
                            xor_input1=cc[1];
                            xor_input2=cc[2];
                            if(xor_output)
                                begin
                                    cnd=1;
                                end
                        end
                    //cmove
                    else if(ifun==4'b0011)
                        begin
                            // zf
                            if(cc[0])
                                begin
                                    cnd=1;
                                end
                        end
                    //cmovne
                    else if(ifun==4'b0100)
                        begin
                            // !zf
                            not_input=cc[0];
                            if(not_output)
                                begin
                                    cnd=1;
                                end
                        end
                     //cmovge
                    else if(ifun==4'b0101)
                        begin
                            // !(sf^of)
                            xor_input1=cc[1];
                            xor_input2=cc[2];
                            not_input=xor_output;
                            if(not_output)
                                begin
                                    cnd=1;
                                end
                        end
                    //cmovg
                    else if(ifun==4'b0110)
                        begin
                            //!(sf^of)) && (!zf)
                            xor_input1=cc[1];
                            xor_input2=cc[2];
                            not_input=xor_output;
                            if(not_output)
                                begin
                                    not_input=cc[0];
                                    if(not_output)
                                        begin
                                            cnd=1;
                                        end
                                end
                        end
                end
            //irmovq
            else if(icode==4'b0011)
                begin
                    x=valC;
                    y=64'b0;
                    icd=icode;
                    control=2'b00;

                end
            //rmmovq
            else if(icode==4'b0100)
                begin
                icd=icode;
                    x=valC;
                    y=valB;
                    control=2'b00;

                end
            //mrmovq
            else if(icode==4'b0101)
                begin
                    icd=icode;
                    x=valC;
                    y=valB;

                end
            //Arithmatic and logic Operations
            else if(icode==4'b0110)
                begin
                    x=valA;
                    y=valB;
                    icd=icode;
                    control=2'b00;
                    //add
                    if(ifun==4'b0000)
                        begin
                            control=2'b00;
                        end
                    //sub
                    else if(ifun==4'b0001)
                        begin
                            control=2'b01;
                        end
                    //and
                    else if(ifun==4'b0010)
                        begin
                            control=2'b10;
 end
                     end
```

```
//xor
          else if(ifun==4'b0011)
              begin
                  control=2'b11;
              end

        end
       //call
       else if(icode==4'b1000)
           begin
              x=-64'b1;
              y=valB;
              icd=icode;
              control=2'b00;

           end
      //ret
      else if(icode==4'b1001)
          begin
              x=64'b1;
              y=valB;
              icd=icode;
              control=2'b00;

          end
      //pushq
      else if(icode==4'b1010)
          begin
              x=-64'b1;
              y=valB;
              icd=icode;
              control=2'b00;

          end
      //popq
      else if(icode==4'b1011)
          begin
              x=64'b1;
              y=valB;
              icd=icode;
              control=2'b00;

          end

      //jxx
      else if(icode==4'b0111)
          begin
              //jmp
              if(ifun==4'b0000)
              begin
                  cnd=1;
              end
              //jle
              else if(ifun==4'b0001)
                  begin
                      // (sf^of)||zf
                      xor_input1=cc[1];
                      xor_input2=cc[2];
                      or_input1=xor_output;
                      or_input2=cc[0];
                      if(or_output)
                          begin
                              cnd=1;
                          end
                  end
              //jl
              else if(ifun==4'b0010)
                  begin
                      // sf^of
                      xor_input1=cc[1];
                      xor_input2=cc[2];
                      if(xor_output)
                          begin
                              cnd=1;
                          end
                  end
              //je
              else if(ifun==4'b0011)
                  begin
                      // zf
                      if(cc[0])
                          begin
                              cnd=1;
                          end
                  end
              //jne
              else if(ifun==4'b0100)
                  begin
         // !zf
                      not_input=cc[0];
                      if(not_output)
                          begin
                              cnd=1;
                          end
                  end
//jge
              else if(ifun==4'b0101)
                  begin
                      // !(sf^of)
                      xor_input1=cc[0];
                      xor_input2=cc[1];
                      not_input=xor_output;
                      if(not_output)
                          begin
                              cnd=1;
                          end
                  end
```

```
//jg
          else if(ifun==4'b0110)
              begin
                  //!(sf^of)) && (!zf)
                  xor_input1=cc[1];
                  xor_input2=cc[2];
                  not_input=xor_output;
                  if(not_output)
                      begin
                          not_input=cc[0];
                          if(not_output)
                              begin
                                  cnd=1;
                              end
                      end
              end
          end
      end

  end
endmodule
```

ALU Code:

```
module ALU_final(icode,x,y,control,result,zf,sf,OF);
  input [3:0] icode;
  input signed[63:0] x,y;
  input [1:0] control;
  output reg signed[63:0] result;
  output reg zf;
  output reg sf;
  output reg OF;

  wire signed [63:0]ans1;
  wire signed [63:0]ans2;
  wire signed [63:0]ans3;
  wire signed [63:0]ans4;

  reg signed [63:0]ansfinal;
  reg overflowfinal;
  wire OF1,OF2;
  ADD g1(x,y,ans1,OF1);
  SUB g2(x,y,ans2,OF2);
  AND g3(x,y,ans3);
  XOR g4(x,y,ans4);

  always @(*)
  begin
    case(control)
      2'b00:begin
          ansfinal=ans1;
          overflowfinal=OF1;
        end
      2'b01:begin
          ansfinal=ans2;
          overflowfinal=OF2;
        end
      2'b10:begin
          ansfinal=ans3;
          overflowfinal=1'b0;
        end
      2'b11:begin
          ansfinal=ans4;
          overflowfinal=1'b0;
        end
    endcase
  end


always @(*)
  result= ansfinal;

always @(*)
    begin
        if(icode==4'b0110)
            begin
                if(ansfinal==64'b0)
                    begin
                        zf=1;
                    end
                else
                    begin
                        zf=0;
                    end

                if(ansfinal[63]==1'b1)
                    begin
                        sf=1;
                    end
                else
                    begin
                        sf=0;
                    end
                OF= overflowfinal;
            end
    end
endmodule
```

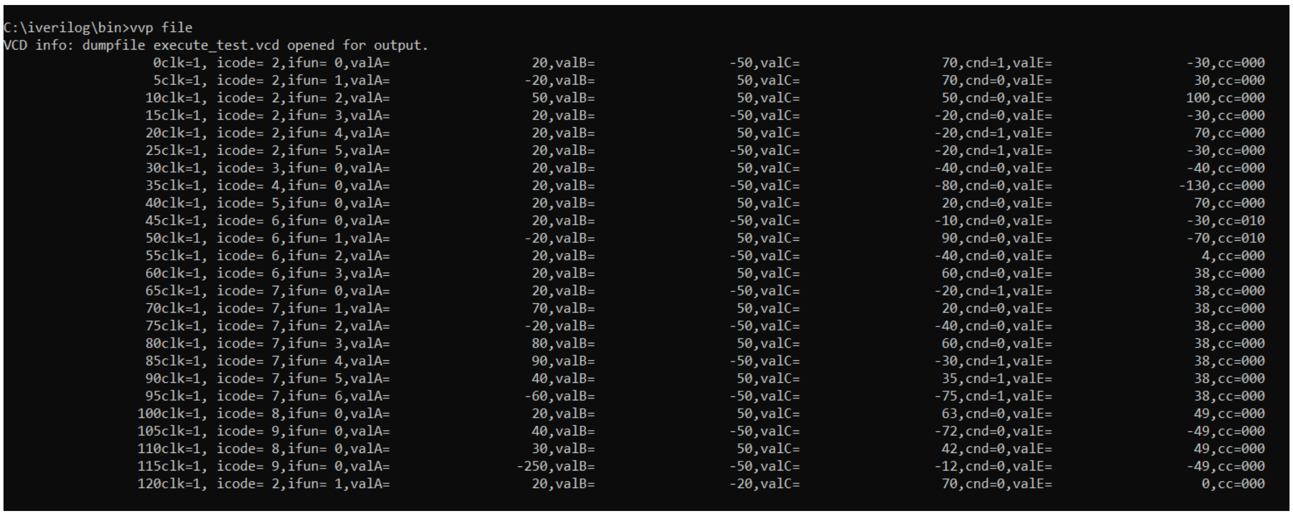
ALU intrinsic Functions:

```
module XOR(x,y,z);
input [63:0] x,y;
output [63:0] z;

genvar i;

generate
for(i=0;i<64;i = i+1) begin

xor(z[i] , x[i] , y[i]);

end

endgenerate

endmodule

module AND(x,y,z);
input[0:63] x,y;
output[0:63]z;

genvar i;

generate

for(i=0;i<64;i = i+1) begin

and(z[i] , x[i] , y[i]);

end

endgenerate

endmodule

module ADD(x,y,sum,OF);
input signed[63:0] x,y;
output signed[63:0] sum;
output signed OF;
wire signed[63:0] ci;

wire carry_in;
assign carry_in = 1'b0;


full_adder FA1(x[0],y[0],carry_in,sum[0],ci[0]);
genvar i;
generate
for(i=1;i<64;i=i+1)
begin
full_adder FA0(x[i] , y[i] , ci[i-1] , sum[i] , ci[i]);
end
endgenerate

xor(OF,ci[62],ci[63]);

endmodule

module full_adder(x,y,carry_in,sum,carry);
   input signed x,y,carry_in;
   output signed sum,carry;
   wire signed w1,w2,w3,w4,w5;
   xor(w1,x,y);
   xor(sum,w1,carry_in);
   and(w2,x,y);
   and(w3,x,carry_in);
   and(w4,y,carry_in);
   or(w5,w2,w3);
   or(carry,w5,w4);
endmodule

module  SUB(x,y,z,OF);
input signed[63:0] x,y;
output signed[63:0] z;
output OF;

wire signed[63:0] u,v,w;
wire signed[63:0] k;


twocomp comp(y , u);



ADD sub_add(x , u , w , OF);

assign z=w;

endmodule



module twocomp(x ,y);
input signed[63:0] x;
output signed[63:0] y;
wire signed[63:0] w;
wire c_out;
genvar i;
generate
for(i=0 ; i<64 ; i = i+1)
begin
not (w[i] , x[i]);
end
endgenerate
ADD twocomp1(w,64'b1,y,c_out);
endmodule
```

ALU test result:

```
 0control=00 icode= 0 a=                 -90 b=                   4 ans=                 -86 zf=x sf=x overflow=x
                20control=01 icode= 0 a=-6067004223159161910 b=-6067004223159161667 ans=                -243
zf=x sf=x overflow=x
                40control=10 icode=0000
a=0000000000000000000000000000000000000000000000000001000000010001
b=0000000000000000000000000000000000000000000000000000000000000100
ans=1111111111111111111111111111111111111111111111111111111100001101 zf=x sf=x overflow=x
                40control=10 icode= 0 a=                4113 b=                   4 ans=                   0 zf=x sf=x overflow=x
                60control=01 a=0000000000000000000000000000000000000000000000000000000000001011
b=0000000000000000000000000000000000000000000000000000000000001011
ans=0000000000000000000000000000000000000000000000000000000000000000 zf=x sf=x overflow=x
                60control=01 icode= 6 a=                  11 b=                  11 ans=                   0 zf=1 sf=0 overflow=0
                80control=00 icode= 2 a=                 -40 b=                 -50 ans=                 -90 zf=1 sf=0 overflow=0
               100control=01 icode= 6 a=                  30 b=                 -80 ans=                 110 zf=0 sf=0 overflow=0
```

Output of Execute stage:



```
C:\iverilog\bin>vvp file
VCD info: dumpfile execute_test.vcd opened for output.
                   0clk=1, icode= 2,ifun= 0,valA=                   20,valB=                  -50,valC=                   70,cnd=1,valE=                  -30,cc=000
                   5clk=1, icode= 2,ifun= 1,valA=                  -20,valB=                   50,valC=                   70,cnd=0,valE=                   30,cc=000
                  10clk=1, icode= 2,ifun= 2,valA=                   50,valB=                   50,valC=                   50,cnd=0,valE=                  100,cc=000
                  15clk=1, icode= 2,ifun= 3,valA=                   20,valB=                  -50,valC=                  -20,cnd=0,valE=                  -30,cc=000
                  20clk=1, icode= 2,ifun= 4,valA=                   20,valB=                   50,valC=                  -20,cnd=1,valE=                   70,cc=000
                  25clk=1, icode= 2,ifun= 5,valA=                   20,valB=                  -50,valC=                  -20,cnd=1,valE=                  -30,cc=000
                  30clk=1, icode= 3,ifun= 0,valA=                   20,valB=                   50,valC=                  -40,cnd=0,valE=                  -40,cc=000
                  35clk=1, icode= 4,ifun= 0,valA=                   20,valB=                  -50,valC=                  -80,cnd=0,valE=                 -130,cc=000
                  40clk=1, icode= 5,ifun= 0,valA=                   20,valB=                   50,valC=                   20,cnd=0,valE=                   70,cc=000
                  45clk=1, icode= 6,ifun= 0,valA=                   20,valB=                  -50,valC=                  -10,cnd=0,valE=                  -30,cc=010
                  50clk=1, icode= 6,ifun= 1,valA=                  -20,valB=                   50,valC=                   90,cnd=0,valE=                  -70,cc=010
                  55clk=1, icode= 6,ifun= 2,valA=                   20,valB=                  -50,valC=                  -40,cnd=0,valE=                    4,cc=000
                  60clk=1, icode= 6,ifun= 3,valA=                   20,valB=                   50,valC=                   60,cnd=0,valE=                   38,cc=000
                  65clk=1, icode= 7,ifun= 0,valA=                   20,valB=                  -50,valC=                  -20,cnd=1,valE=                   38,cc=000
                  70clk=1, icode= 7,ifun= 1,valA=                   70,valB=                   50,valC=                   20,cnd=0,valE=                   38,cc=000
                  75clk=1, icode= 7,ifun= 2,valA=                  -20,valB=                  -50,valC=                  -40,cnd=0,valE=                   38,cc=000
                  80clk=1, icode= 7,ifun= 3,valA=                   80,valB=                   50,valC=                   60,cnd=0,valE=                   38,cc=000
                  85clk=1, icode= 7,ifun= 4,valA=                   90,valB=                  -50,valC=                  -30,cnd=1,valE=                   38,cc=000
                  90clk=1, icode= 7,ifun= 5,valA=                   40,valB=                   50,valC=                   35,cnd=1,valE=                   38,cc=000
                  95clk=1, icode= 7,ifun= 6,valA=                  -60,valB=                  -50,valC=                  -75,cnd=1,valE=                   38,cc=000
                 100clk=1, icode= 8,ifun= 0,valA=                   20,valB=                   50,valC=                   63,cnd=0,valE=                   49,cc=000
                 105clk=1, icode= 9,ifun= 0,valA=                   40,valB=                  -50,valC=                  -72,cnd=0,valE=                  -49,cc=000
                 110clk=1, icode= 8,ifun= 0,valA=                   30,valB=                   50,valC=                   42,cnd=0,valE=                   49,cc=000
                 115clk=1, icode= 9,ifun= 0,valA=                 -250,valB=                  -50,valC=                  -12,cnd=0,valE=                  -49,cc=000
                 120clk=1, icode= 2,ifun= 1,valA=                   20,valB=                  -20,valC=                   70,cnd=0,valE=                    0,cc=000
```

Memory Stage:(memory.v)

The Inputs to this module are clk,icode,valA,valE,valP.
This module outputs valM, data_memory_error.
Mainly two operations are performed in this stage i.e read from memory or write to the memory.
These two operations are done depending on the type of instruction.
So we have created a memory which is a array of 64 bit registers to store,read and write data.
If there is a invalid memory address given as input then it will generate a data_memory_error.

Memory Stage Code:

```
 module memory(clk,icode,valA,valE,valP,valM,data_memory_error);
 input clk;
 input [3:0] icode;
 input [63:0] valA;
 input [63:0] valE;
 input [63:0] valP;
 output reg [63:0] valM;
 output reg data_memory_error;

 reg [63:0] data_memory[1023:0];

 initial valM=64'b0;

 always @(*)
   begin
     data_memory_error=0;
     //rmmovq
     if(icode==4'b0100)
       begin
         if((valE>1023)||(valE<0))
           data_memory_error=1;
         else
           data_memory[valE]=valA;
       end
     //mrmovq
     else if(icode==4'b0101)
       begin
         if((valE>1023)||(valE<0))
           data_memory_error=1;
         else
           valM=data_memory[valE];
       end
     //call
     else if(icode==4'b1000)
       begin
         if((valE>1023)||(valE<0))
           data_memory_error=1;
         else
           data_memory[valE]=valP;
       end
     //ret
     else if(icode==4'b1001)
       begin
         if((valA>1023)||(valA<0))
           data_memory_error=1;
         else
         valM=data_memory[valA];
       end
     //pushq
     else if(icode==4'b1010)
       begin
         if((valE>1023)||(valE<0))
           data_memory_error=1;
         else
         data_memory[valE]=valA;
       end
     //popq
     else if(icode==4'b1011)
       begin
         if((valA>1023)||(valA<0))
           data_memory_error=1;
         else
           valM=data_memory[valA];
       end
   end
endmodule
```

PC Update stage:

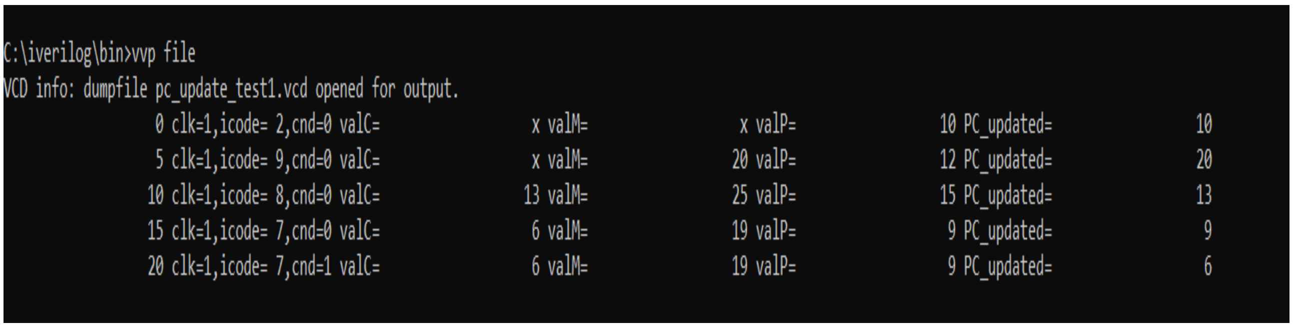
Inputs to this module are clk,icode,cnd,valC,valM,valP.
This module outputs PC_updated signal.

Depending on type of instruction the program counter value is updated using valC,valM,valP and it is shown at the output
For some instructions like jump it will check with cnd signal and then depending on the signal it update program counter.

PC Update Stage Code:

```
module pc_update(clk,icode,cnd,valC,valM,valP,PC_updated);
input clk;
input [3:0] icode;
input cnd;
input [63:0] valC;
input [63:0] valM;
input [63:0] valP;

output reg [63:0] PC_updated;

always @(*)
  begin
      if(icode==4'b0010||icode==4'b0011||icode==4'b0100||icode==4'b0101||icode==4'b0110||icode==4'b1010||icode==4'b1011)
         PC_updated=valP;
      else if(icode==4'b0111)
         PC_updated=cnd?valC:valP;
      else if(icode==4'b1000)
         PC_updated=valC;
      else if(icode==4'b1001)
         PC_updated=valM;
  end
endmodule
```

Output of PC Update Stage:



```
C:\iverilog\bin>vvp file
VCD info: dumpfile pc_update_test1.vcd opened for output.
                   0 clk=1,icode= 2,cnd=0 valC=                    x valM=                    x valP=                   10 PC_updated=                   10
                   5 clk=1,icode= 9,cnd=0 valC=                    x valM=                   20 valP=                   12 PC_updated=                   20
                  10 clk=1,icode= 8,cnd=0 valC=                   13 valM=                   25 valP=                   15 PC_updated=                   13
                  15 clk=1,icode= 7,cnd=0 valC=                    6 valM=                   19 valP=                    9 PC_updated=                    9
                  20 clk=1,icode= 7,cnd=1 valC=                    6 valM=                   19 valP=                    9 PC_updated=                    6
```

Sequential Processor stage:

```
`timescale 1ns / 1ps

`include "fetch.v"
`include "execute_new.v"
`include "dec_wb.v"
`include "memory.v"
`include "pc_update.v"

module processor_seq;
  reg clk;

  reg [63:0] PC;

  reg stat[0:2]; // for AOK INS HLT

  wire [3:0] iCode;
  wire [3:0] iFun;
  wire [3:0] rA;
  wire [3:0] rB;
  wire [63:0] valC;
  wire [63:0] valP;
  wire instr_Validity;
  wire imem_error;
  wire [63:0] valA;
  wire [63:0] valB;
  wire [63:0] valE;
  wire [63:0] valM;
  wire cnd;
  wire halt_flag;
  wire [63:0] PC_updated;

  wire[63:0]
memreg0,memreg1,memreg2,memreg3,memreg4,memreg5,memreg6,memreg7,memreg8,
            memreg9,memreg10,memreg11,memreg12,memreg13,memreg14;   // register file
  wire data_memory_error;


  wire[2:0] cc;  // for zf, sf and of
  integer i;

  fetch fetch(          // fetch
    .clk(clk),
    .PC(PC),
    .rA(rA),
    .rB(rB),
    .iCode(iCode),
    .iFun(iFun),

    .valC(valC),
    .valP(valP),
    .instr_Validity(instr_Validity),
    .imem_error(imem_error),
    .HF(halt_flag)
  );

  execute execute(      // execute
    .clk(clk),
    .icode(iCode),
    .ifun(iFun),
    .valA(valA),
    .valB(valB),
    .valC(valC),
    .cnd(cnd),
    .valE(valE),

    .cc(cc)

  );

  dec_wb dec_wb(       // decode write back
    .clk(clk),

    .iCode(iCode),
    .rA(rA),
    .rB(rB),
    .valA(valA),
    .valB(valB),
    .cnd(cnd),
    .valE(valE),
    .valM(valM),
    .memreg0(memreg0),
    .memreg1(memreg1),
    .memreg2(memreg2),
    .memreg3(memreg3),
    .memreg4(memreg4),
    .memreg5(memreg5),
    .memreg6(memreg6),
    .memreg7(memreg7),
    .memreg8(memreg8),
    .memreg9(memreg9),
    .memreg10(memreg10),
    .memreg11(memreg11),
    .memreg12(memreg12),
    .memreg13(memreg13),
    .memreg14(memreg14)
  );
```

```
memory mem(           // memory
    .clk(clk),
    .icode(iCode),
    .valA(valA),

    .valE(valE),
    .valP(valP),
    .valM(valM),
    .data_memory_error(data_memory_error)
  );

  pc_update pc_update(  // pc_Update
    .clk(clk),

    .icode(iCode),
    .cnd(cnd),
    .valC(valC),
    .valM(valM),
    .valP(valP),

    .PC_updated(PC_updated)

  );



  initial begin
    $dumpfile("processor_seq.vcd");
    $dumpvars(0,processor_seq);
    stat[0]=1;
    stat[1]=0;
    stat[2]=0;
    clk=0;

  end

  initial begin
  #5 clk = ~clk;PC=64'd10;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;

  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  #5 clk = ~clk;
  end

  always@(*)
  begin
    PC=PC_updated;
    if(halt_flag)  // if halt is encountered
    begin
      stat[2]=halt_flag;
      stat[1]=1'b0;
      stat[0]=1'b0;
    end
    else if(instr_Validity)
    begin
      stat[1]=instr_Validity;
      stat[2]=1'b0;
      stat[0]=1'b0;
    end
    else
    begin
      stat[0]=1'b1;
      stat[1]=1'b0;
      stat[2]=1'b0;
    end
  end



  always@(*)
  begin
    if(stat[2]==1'b1)  // if HALT is true
    begin
      $finish;
    end
  end

  initial

                                        $monitor("clk=%d icode=%b ifun=%b rA=%b rB=%b valA=%d valB=%d
valC=%d valE=%d valM=%d insval=%d memerr=%d cnd=%d halt=%d 0=%d 1=%d 2=%d 3=%d 4=%d
5=%d 6=%d 7=%d 8=%d 9=%d 10=%d 11=%d 12=%d 13=%d 14=%d data_memory_error=%d valP = %d
updated_PC=%d\
n",clk,iCode,iFun,rA,rB,valA,valB,valC,valE,valM,instr_Validity,imem_error,cnd,stat[2],memreg0,memreg
1,memreg2,memreg3,memreg4,memreg5,memreg6,memreg7,memreg8,

memreg9,memreg10,memreg11,memreg12,memreg13,memreg14,data_memory_error,valP,PC_updated);

endmodule
```

Sequential Processor testbench results:

```
clk=0 icode=xxxx ifun=xxxx rA=xxxx rB=xxxx valA=              x valB=              x valC=              x valE=              x valM=              0 insval=x
memerr=x cnd=x halt=0 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=x valP =              x
updated_PC=              x

clk=1 icode=0100 ifun=0000 rA=0000 rB=0010 valA=              0 valB=              2 valC=             16 valE=             18 valM=              0 insval=1
memerr=0 cnd=0 halt=0 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=0 valP =             20
updated_PC=             20

clk=0 icode=0100 ifun=0000 rA=0000 rB=0010 valA=              0 valB=              2 valC=             16 valE=             18 valM=              0 insval=1
memerr=0 cnd=0 halt=0 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=0 valP =             20
updated_PC=             20

clk=1 icode=0111 ifun=0000 rA=0000 rB=0010 valA=              0 valB=              2 valC=             16 valE=             18 valM=              0 insval=1
memerr=0 cnd=1 halt=0 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=0 valP =             29
updated_PC=             16

clk=0 icode=0111 ifun=0000 rA=0000 rB=0010 valA=              0 valB=              2 valC=             16 valE=             18 valM=              0 insval=1
memerr=0 cnd=1 halt=0 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=0 valP =             29
updated_PC=             16

clk=1 icode=0000 ifun=0000 rA=0000 rB=0010 valA=              0 valB=              2 valC=             16 valE=             18 valM=              0 insval=1
memerr=0 cnd=0 halt=1 0=              0 1=              1 2=              2 3=              3 4=              4 5=              5 6=              6 7=              7 8=
8 9=              9 10=             10 11=             11 12=             12 13=             13 14=             14 data_memory_error=0 valP =             17
updated
```

This concludes our sequential design.