INTRO TO PROCESSOR ARCHITECTURE
Y-86 64 ARCHITECTURE
(PROJECT REPORT)

TEAM-24
Prakhar Raj (2022102066)
Himanshu Gupta (2022102002)

Project Overview:-

The main objective of this project is to develop a processor architecture design. It involves the implementation of Y-86 64 processor architecture design using hardware coding language VERILOG.
The implementation should execute all instructions in the Y-86 64 ISA.
The goal of the project is to produce a 5 stage pipelined Y-86 64 implementation. The design approach must be modular.

To build the project we used two steps:-

1.Sequential implementation
2.Pipeline implementation

My report includes the sequential and pipeline implementation of the processor. The stages for sequential and pipeline implementation are Fetch logic, Decode logic, Execute logic, Memory Logic, Writeback stage. Also the pc-update is included in sequential part but not in pipeline implementation.

We have build the testcase for all the instruction to run all the stages and i have explained each and every stages of both steps which is used in during the building the Y-86 64 processor architecture. Then explained the all the testcases with their given instruction during implementaion.

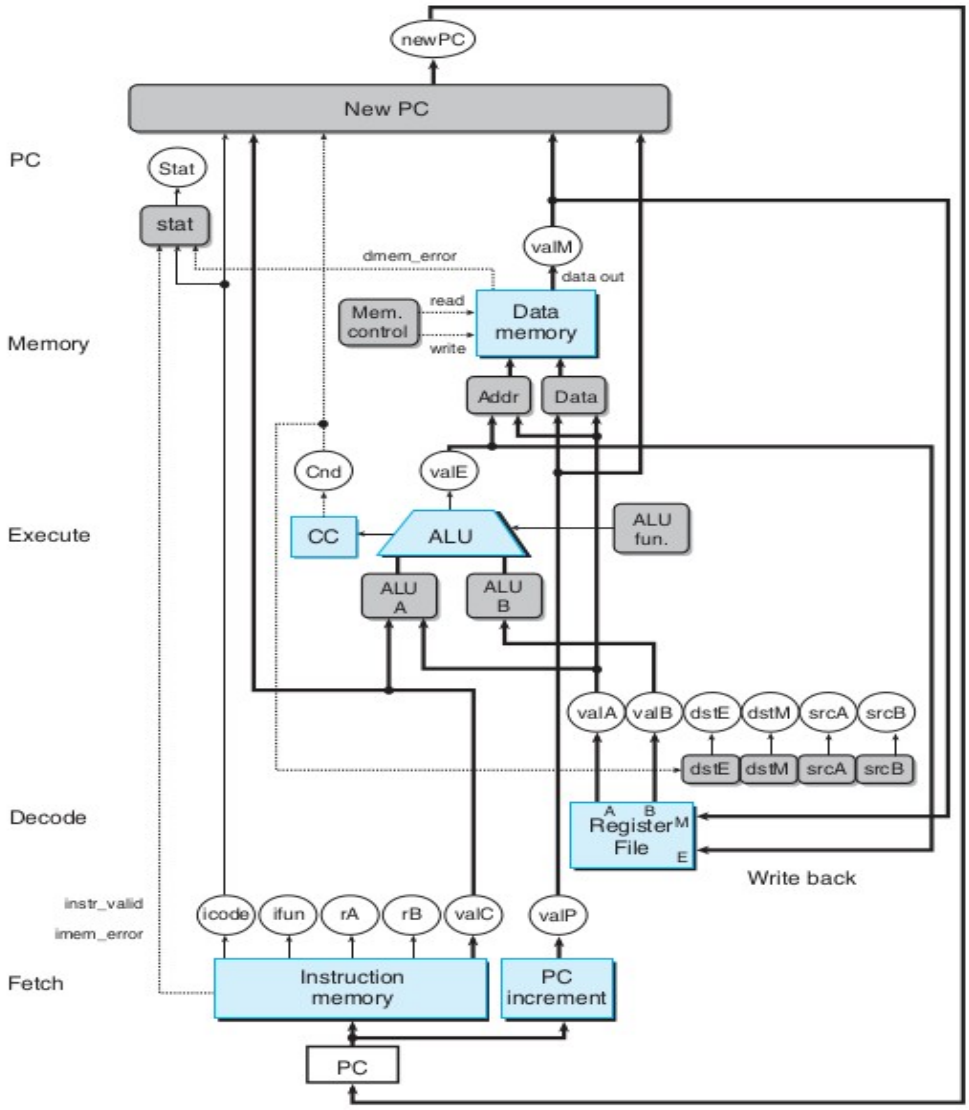
Sequential implementation:-

As a first step, we describe a processor called SEQ (sequential processor). On each clock cycle, SEQ performs all the steps required to process a complete instruction. This would require a very long cycle time, however, and so the clock rate would be unacceptably low. Our purpose in developing SEQ is to provide a first step toward our ultimate goal of implementing an efficient, pipelined processor.

We organize them in a particular sequence of stages, attempting to make all instructions follow a uniform sequence. The detailed processing at each step depends on the particular instruction being executed.

The computations and an informal description to implement all of the Y86 instructions can be organized as a series of six basic stages as follows:-

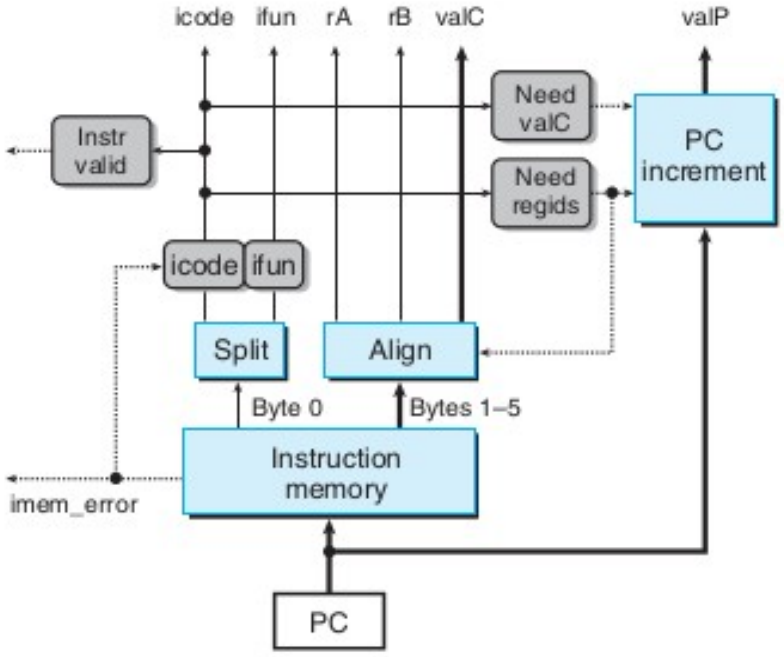
1.Fetch
2.Decode
3.Execute
4.Memory
5.Write back
6.PC update



1.Fetch:-

The fetch stage reads the bytes of an instruction from the istruction memory, using the program counter (PC) as the memory address. It fetches a register rA and rB as per given instruction. It also possibly fetches a 4-byte constant word valC. Using PC incrementer it computes valP to be the next address of the instruction, which is equals the value of the PC plus the length of the fetched instruction and then incremented program counter.

By this we will find the values of icode, ifun, rA, rB and valC as per the instruction, where icode and ifun is instruction code and instruction fun respectively.



Processes in the fetch block:-

i) On positive edge of clock signal the program counter (PC) stores current instruction, which will be executed and it goes to instruction memory.

ii) In our implementation we have used maximum number of instructions to 1024 in instruction memory, i.e. PC is from 0 to 1023. So, if the PC value is greater than the maximum instruction 1024 then the instruction memory gives an imem_error. So , if imem_error case happened then we use nop operation and if PC is valid then memory passes 10 bytes of instruction further.

iii) Now from the given instruction we will get icode elements equal to instr_mem[7:4] and ifun elements as instr_mem[3:0].

iv) Using the value of icode we get the values rA as instr_mem[PC+1][7:4] and rB as instr_mem[PC+1][3:0] per given instructions.

v) Using icode and according to given instruction and their rules we will get the value of valC as instr_mem[PC+i]. Where, i ranges from 1 to 10 and i is decided as per the provided instruction.

vi) Similarly, using the icode and as per given instruction and their rules we will get the value of valP.

vii) If the icode or ifun are invalid then instr_valid is set to 0. The Range of icode and ifun is defined according to the current icode instruction.

viii) If halt (icode = 0) is encountered the HLT flag is set to 1 and the processor stops running due to a $finish statement.

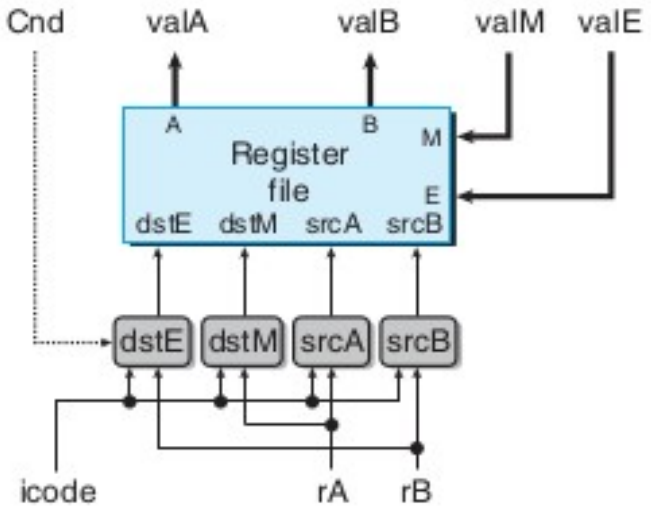
Fetch module:-

```
module fetch (clk,icode,ifun,rA,rB,valC,valP,imem_error,instr_valid,PC);
    input clk;
    input [63:0] PC;
    reg [7:0] instr_mem [0 : 1023];
    output reg [3:0] icode, ifun;
    output reg [3:0] rA, rB;
    output reg [63:0] valC, valP;
    output reg imem_error = 0, instr_valid = 1;

initial
begin

   $readmemb("test9.txt",instr_mem,0,836);

end

always @(posedge clk)
begin
    if (PC > 1023)
        assign imem_error = 1;
    else
        assign imem_error = 0;

    icode = instr_mem[PC][7:4];
    ifun  = instr_mem[PC][3:0];

    if (icode == 4'b0000 || icode == 4'b0001)
    begin
        valP = PC + 1;
    end
```

```
    else if (icode == 4'b0010 || icode == 4'b0011 || icode == 4'b0100 || icode == 4'b0101 || icode == 4'b0110 || icode == 4'hA || icode == 4'hB)
    begin  // irmovq
        rA = instr_mem[PC+1][7:4];
        rB = instr_mem[PC+1][3:0];
        if (icode == 4'b0011 || icode == 4'b0100 || icode == 4'b0101)  // irmovq mrmovq mrmovq
        begin
          valC = {instr_mem[PC+9],instr_mem[PC+8],instr_mem[PC+7],instr_mem[PC+6],instr_mem[PC+5],instr_mem[PC+4],instr_mem[PC+3],instr_mem[PC+2]};
          valP = PC + 10;
        end
        if (icode == 4'b0010 || icode == 4'b0110 || icode == 4'hA || icode == 4'hB)  // cmovq // Opq // pushq
        begin
          valP = PC + 2;
        end
    end

    else if (icode == 4'b0111 || icode == 4'b1000)
    begin // jxx
        valC = {instr_mem[PC+8],instr_mem[PC+7],instr_mem[PC+6],instr_mem[PC+5],instr_mem[PC+4],instr_mem[PC+3],instr_mem[PC+2],instr_mem[PC+1]};
        valP = PC + 9;
    end

    else if (icode == 4'b1001)
    begin // ret
        valP = PC + 1;
    end

    else
        instr_valid = 1'b0;
end

endmodule
```

2.Decode and write back:-

The register file has four ports. It supports up to two simultaneous reads (on ports A and B) and two simultaneous writes (on ports E and M). Each port has both an address connection and a data connection, where the address connection is a register ID, and the data connection is a set of 32 wires serving as either an output word (for a read port) or an input word (for a write port) of the register file. The two read ports have address inputs srcA and srcB, while the two write orts have address inputs dstE and dstM.



Processes in the decode and write back block:-

i) The decode stage reads up to two registers designated by instruction fields rA and rB, but for some instructions it also read register %esp.

ii) After reading operands from register file it assign the values of valA and valB based on icode, rA and rB. For this we need a register file, conaining 15 registers which updates throughout the program's runtime.

iii) The register file has two write ports. So, the write-back stage writes up to two results to the register file. Port E is used to write values computed by the ALU, while port M is used to write values read from the data memory.

iv) The register file gets updated corresponding to the current icode and ifun values, the values of valE, valM are passed to the register file and the value of valE and valM are the final values of the updated register.

Decode and write back module:-

```
module decode(clk, icode, rA, rB, valA, valB, valM, valE, cnd, rax,
 rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14);
```

```
  input clk;
  input [3:0] rA;
  input [3:0] rB;
  input [3:0] icode;
  input [63:0] valM;
  input [63:0] valE;
  input cnd;
  output reg [63:0] valA;
  output reg [63:0] valB;
  output reg [63:0] rax;
  output reg [63:0] rcx;
  output reg [63:0] rdx;
  output reg [63:0] rbx;
  output reg [63:0] rsp;
  output reg [63:0] rbp;
  output reg [63:0] rsi;
  output reg [63:0] rdi;
  output reg [63:0] r8;
  output reg [63:0] r9;
  output reg [63:0] r10;
  output reg [63:0] r11;
  output reg [63:0] r12;
  output reg [63:0] r13;
  output reg [63:0] r14;
  reg [63:0] reg_memory [0:14];

  initial
  begin
       reg_memory[0] = 0;
       reg_memory[1] = 0;
       reg_memory[2] = 0;
       reg_memory[3] = 0;
       reg_memory[4] = 0;
       reg_memory[5] = 0;
       reg_memory[6] = 0;
       reg_memory[7] = 0;
       reg_memory[8] = 0;
       reg_memory[9] = 0;
       reg_memory[10] = 0;
       reg_memory[11] = 0;
       reg_memory[12] = 0;
       reg_memory[13] = 0;
       reg_memory[14] = 0;

  end
```

```
always @(*)
begin
    // For nop and halt conditions there are no registers involved
    if(icode == 4'b0010 || icode == 4'b0100 || icode == 4'b0101 ||icode == 4'b0110) //cmovxx
    begin
        valA = reg_memory[rA];
        if(icode == 4'b0100 || icode == 4'b0101 ||icode == 4'b0110) //rmmovq //mrmovq //opq
        begin
            valB = reg_memory[rB];
        end
    end
    if(icode == 4'b1000 || icode == 4'b1001 || icode == 4'b1010 || icode == 4'b1011) //call
    begin
        if(icode == 4'b1001 || icode == 4'b1011) //ret
        begin
            valA = reg_memory[4];
        end
        if(icode == 4'b1010) //pushq
        begin
            valA = reg_memory[rA];
        end
        valB = reg_memory[4]; // As we have to push address o fnext instruction onto stack
    end
```

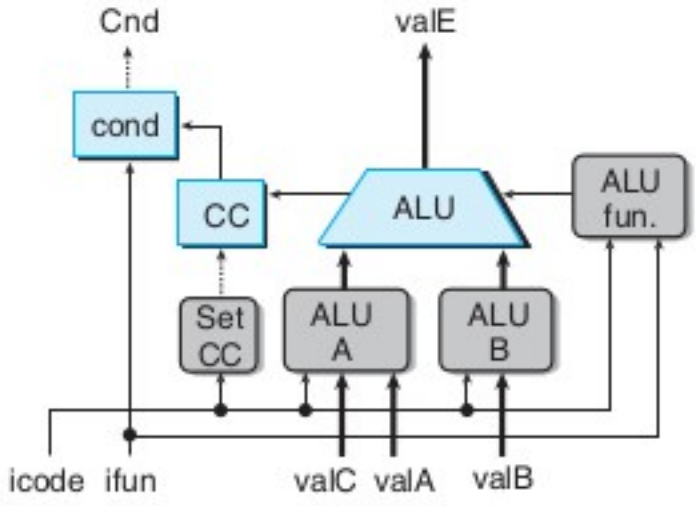
```
rax =reg_memory[0];
rcx =reg_memory[1];
rdx =reg_memory[2];
rbx =reg_memory[3];
rsp =reg_memory[4];
rbp =reg_memory[5];
rsi =reg_memory[6];
rdi =reg_memory[7];
r8  =reg_memory[8];
r9  =reg_memory[9];
r10 =reg_memory[10];
r11 =reg_memory[11];
r12 =reg_memory[12];
r13 =reg_memory[13];
r14 =reg_memory[14];

end
```

```
//write_back
always@(negedge clk)
begin

if(icode==4'b0010) //cmovxx
begin
if(cnd==1'b1)
begin
reg_memory[rB]=valE;
end
end
else if(icode==4'b0011 || icode==4'b0110) //irmovq  OPq
begin
reg_memory[rB]=valE;
end

else if(icode==4'b0101) //mrmovq
begin
reg_memory[rA]=valM;
end
else if(icode==4'b1000 || icode==4'b1001 || icode==4'b1010  ) //call  ret    pushq
begin
reg_memory[4]=valE;
end
else if(icode==4'b1011) //popq
begin
reg_memory[4]=valE;
reg_memory[rA]=valM;
end

rax =reg_memory[0];
rcx =reg_memory[1];
rdx =reg_memory[2];
rbx =reg_memory[3];
rsp =reg_memory[4];
rbp =reg_memory[5];
rsi =reg_memory[6];
rdi =reg_memory[7];
r8  =reg_memory[8];
r9  =reg_memory[9];
r10 =reg_memory[10];
r11 =reg_memory[11];
r12 =reg_memory[12];
r13 =reg_memory[13];
r14 =reg_memory[14];

end
endmodule
```

3. Execute:-

The execute stage uses the arithmetic logic (ALU) unit for different purposes according to the instruction type and performs the operation specified by the instruction, computes the effective address of a memory reference, or increments or decrements the stack pointer.

The execute stage has valA, valB, ifun, icode, valC as input's which are passed through 3 ALU blocks to compute valE, this usually refers to the effective address. Also, an important function of execute stage is to set Cnd. The three condition codes are also set for the instructions jXX and CMOV and valE is computed using ALU.



Processes in the execute block:-

i) The ALU computation for each instruction is shown as the first step in the execute stage. The operands are listed with aluB first and then aluA to make sure the subl instruction subtracts valA from valB.

ii) Now after the operations performed by the ALU in the execute stage, we can see that it is mostly used as an adder. For the OPl instructions, however, we use the operation encoded in the ifun field of the instruction.

iii) As this stage includes the condition code register. So now our ALU generates the three signals on which the condition codes are based on flags as zero, sign, and overflow flag every time it operates and then we set the condition codes when an OPl instruction is executed.

iv) Depending on “cond” and the function code it determine whether a conditional branch or data transfer should take place.

v) Now it generates the Cnd signal used both for the setting of dstE with CMMOV, and in the next PC logic for conditional branches.

vi) The Cnd signal may be set to either 1 or 0, depending on the instruction’s function code and the setting of the condition codes, but it will be ignored by the control logic.

Execute module:-

```
`include "ALU.v"

module execute(clk,icode,ifun,valC,valA,valB,cnd,valE,ZF,SF,OF);

input clk;
input [3:0] icode,ifun;
input [63:0] valC,valA,valB;
input [2:0] CC_in; // 3 conditional code
output reg cnd;
output reg signed [63:0] valE;
// output reg [2:0]; // Changed values of CC

output reg ZF,SF,OF;          // flag bits
initial begin
    ZF =0;
    SF =0;
    OF =0;
end

reg [1:0] control;
reg signed [63:0]A,B;
wire signed [63:0]add_out,sub_out,and_out,xor_out;

sixty_four_bit_ALU alu(control,A,B,add_out,sub_out,and_out,xor_out,add_Cout,sub_Cout);
```

```
always@(*)
begin
if(icode==4'b0110 && clk==1)

begin
    if(control == 2'b00)
    begin
  ZF =(add_out == 1'b0);
    end
    else if(control == 2'b01)
    begin
  ZF=(sub_out == 1'b0);
    end
    else if(control == 2'b10)
    begin
  ZF=(and_out == 1'b0);
    end
    else if(control == 2'b11)
    begin
  ZF=(xor_out == 1'b0);
    end
```

```
if(control == 2'b00)
begin
SF=(add_out[63]==1'b1);
end
else if(control == 2'b01)
begin
SF=(sub_out[63]==1'b1);
end
else if(control == 2'b10)
begin
SF=(and_out[63]==1'b1);
end
else if(control == 2'b11)
begin
SF=(xor_out[63]==1'b1);
end
```

```
if(control == 2'b00)
begin
OF=(((A[63] == 1'b1) == (B[63] == 1'b1))&&((add_out[63] == 1'b1) != (A[63] == 1'b1)));
end
else if(control == 2'b01)
begin
OF=(((A[63] == 1'b1) == (B[63] == 1'b1))&&((sub_out[63] == 1'b1) != (A[63] == 1'b1)));
end

end

    if(icode == 4'b0010 || icode == 7)
    begin
        if(ifun == 4'h0) cnd = 1;             // unconditional
        else if(ifun == 4'h1) cnd = (SF^OF)|ZF; // le
        else if(ifun == 4'h2) cnd = OF^SF;    // l
        else if(ifun == 4'h3) cnd = ZF;       // e
        else if(ifun == 4'h4) cnd = ~ZF;      // ne
        else if(ifun == 4'h5) cnd = ~(SF^OF); // ge
        else if(ifun == 4'h6) cnd = ~(SF^OF)&~ZF; // g
    end

end
```

```
always @(*)
begin

    if(icode == 4'b0011) //irmovq
    begin
        B=valC;
        valE= B;
    end
    if(icode==4'b0100) //rmmovq
    begin
         A =valC;
         B=valB;
         control=2'b00;
         valE=add_out;
    end
    if(icode == 4'b0101) //mrmovq
    begin
        A=valC;
        B=valB;
        control=2'b00;
        valE=add_out;

    end
```

```
    if (icode == 6)   // opq
    begin
        A = valA;
        B = valB;
        if (ifun == 2'b00)
        begin
            control = 2'b00;
            valE=add_out;
        end
        else if (ifun == 2'b01)
        begin
            control = 2'b01;
            valE= sub_out;
        end
        else if (ifun == 2'b10)
        begin
            control = 2'b10;
            valE=and_out;
        end
        else if (ifun == 2'b11)
        begin
            control = 2'b11;
        valE= xor_out;
        end
    end
```

```
    else if(icode==4'b1000) //call
    begin
        A=valB;
        B=64'd8;
        control=2'b01;
        valE=sub_out;
    end
    else if(icode==4'b1001)//ret
    begin
        A=64'd8;
        B=valB;
        control=2'b00;
        valE= add_out;
    end
    else if(icode==4'hA)//pushq
    begin
        A=valB;
        B=64'd8;
        control=2'b01;
        valE= sub_out;
    end
    else if(icode==4'hB)//popq
    begin
        A=64'd8;
        B=valB;
        control=2'b00;
        valE= add_out;
    end


    else if(icode == 4'b0010) // cmovxx
    begin
        valE= valA;
    end

end

endmodule
```

ALU module:-

```
module FullAdder(X,Y,Cin,Sum,Cout);

input X,Y,Cin;
output Sum,Cout;
wire a,b,c;

xor X1(a,X,Y);
and X2(b,X,Y);
xor X3(Sum,a,Cin);
and X4(c,a,Cin);
or  X5(Cout,b,c);

endmodule

module sixty_four_bit_adder(A,B,Sum,Cout);

input [63:0]A,B;
output [63:0]Sum;
output Cout;
wire [64:0]Cin;

assign Cin[0] = 0;
genvar i;
generate for(i=0;i<64;i=i+1)

begin

FullAdder alu(A[i],B[i],Cin[i],Sum[i],Cin[i+1]);
xor (Cout,Cin[64],Cin[63]);

end
endgenerate

endmodule
```

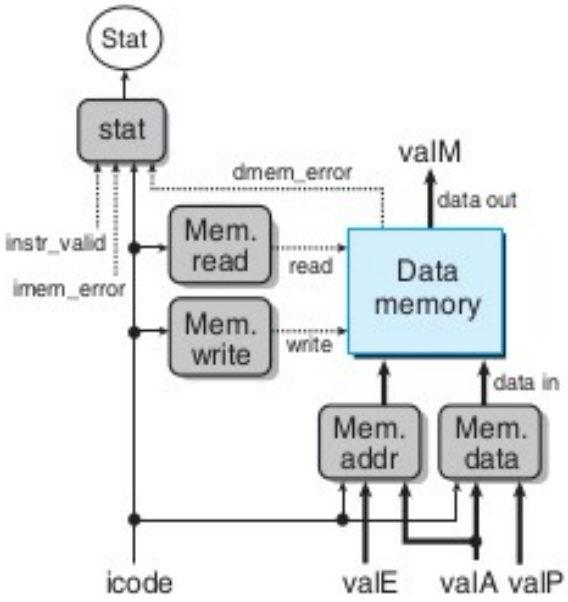
```
module sixty_four_bit_subtractor(A,B,Sum,Cout);

input [63:0]A,B;
output [63:0]Sum;
output Cout;
wire [64:0]Cin;
wire [63:0]new;

assign Cin[0] = 1;

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

begin

xor (new[i],B[i],Cin[0]);
FullAdder alu(A[i],new[i],Cin[i],Sum[i],Cin[i+1]);
xor (Cout,Cin[64],Cin[63]);

end
endgenerate

endmodule

module sixty_four_bit_and(A,B,Out);

input [63:0]A,B;
output [63:0]Out;

genvar i;
generate for(i=0;i<64;i=i+1)
and x1(Out[i],A[i],B[i]);
endgenerate

endmodule
```

```
module sixty_four_bit_xor(A,B,Out);

input [63:0]A,B;
output [63:0]Out;

genvar i;
generate for(i=0;i<64;i=i+1)
xor x1(Out[i],A[i],B[i]);
endgenerate

endmodule

module sixty_four_bit_ALU(control,A,B,add_out,sub_out,and_out,xor_out,add_Cout,sub_Cout);

input [1:0]control;
input [63:0]A,B;
output [63:0]add_out,sub_out,and_out,xor_out;
output add_Cout,sub_Cout;
output add_overflow,sub_overflow;

wire [63:0]add_out_temp,sub_out_temp,and_out_temp,xor_out_temp;
wire add_Cout_temp,sub_Cout_temp;
wire control_0,control_1,control_2,control_3;
wire k1,k2;

sixty_four_bit_adder      ALU_0(A,B,add_out_temp,add_Cout_temp);
sixty_four_bit_subtractor ALU_1(A,B,sub_out_temp,sub_Cout_temp);
sixty_four_bit_and        ALU_2(A,B,and_out_temp);
sixty_four_bit_xor        ALU_3(A,B,xor_out_temp);

and x0(control_0,!control[1],!control[0]);
and x1(control_1,!control[1],control[0]);
and x2(control_2,control[1],!control[0]);
and x3(control_3,control[1],control[0]);
```

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

begin

and x4(add_out[i],add_out_temp[i],control_0);
and x5(sub_out[i],sub_out_temp[i],control_1);
and x6(and_out[i],and_out_temp[i],control_2);
and x7(xor_out[i],xor_out_temp[i],control_3);
and x8(add_Cout,add_Cout_temp,control_0);
and x9(sub_Cout,sub_Cout_temp,control_1);

end
endgenerate

endmodule
```

4.Memory:-

The memory stage is responsible for reading from and writing to memory, when executing a memory instruction. The instruction and data memories access the same memory locations, but for different purposes. It retrieves the memory address from the previous stage and sends a request to the memory system to read or write data from or to that address. In reading the memory it passes data to the next stage and in writing it stores the data to memory at the specified address.



Processes in the memory block:-

i) I have declared the memory array as a register array in the memory stage as access to the data memory is only required here.

ii) Based on the icode we can decide whether we are required to read from or write to memory.

iii) The data address is calculated using icode, valE and valA and the data input is calculated using icode, valA and valP.

iv) Now it also checks for memory-related errors, such as invalid memory accesses and sets appropriate flags in the status register.

v) The status register is then passed on to the next stage, where it can be used to make decisions or trigger interruptions.

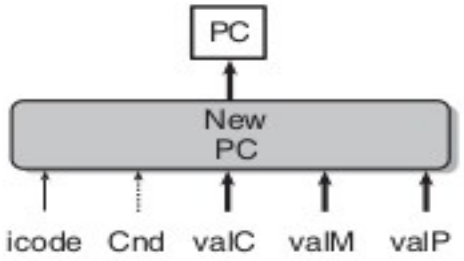
Memory Module:-

```
module memory (clk, icode, valA, valB, valP, valE, valM, stored_in_mem,index);

input clk;
input wire [3:0] icode;
input wire [63:0] valA, valB, valP, valE;

output reg [63:0] valM;
reg [63:0] memory_arr [0:1023];
output reg memory_error = 0;
//This is just a flag to ensure that we are accessing the correct memory_arr
output reg signed [63:0] stored_in_mem,index;

always @(*)
begin

    if((valE>1023 )| (valA >1023 ))
    begin
        memory_error=1;
    end

    else if(icode == 4'b0101) //mrmovq
    begin
        valM=memory_arr[valE];
    end
    else if(icode == 4'b1001) //ret
    begin
        valM=memory_arr[valA];
    end
    else if( icode== 4'b1011) //popq
    begin
        valM=memory_arr[valA];
    end

end
```

```
always @(*)
begin
// we are checking only valE because we are accessing memory_arr
// with the value of valE in the following cases->
    if(valE>1023)
       begin
        memory_error=1;
       end

    if(icode == 4'b0100) //rmmovq
    begin
        memory_arr[valE] <= valA;
    end
    else if(icode == 4'b1000) //call
    begin
        memory_arr[valE] <= valP;
    end
    else if(icode == 4'b1010) // pushq
    begin
        memory_arr[valE] <= valA;
    end

    stored_in_mem = memory_arr[valE];

end
endmodule
```

5.PC update:-

The main role of this stage is to update the value of PC to the address of the next instruction after a given instruction has finished executing.
The new PC will be valC, valM, or valP, depending on the instruction type and according to icode, Cnd, valC, valM and valP as inputs, whether or not a branch should be taken.



	OPq rA, rB	
PC update	PC valP	Update PC
	rmmovq rA, D(rB)	
PC update	PC valP	Update PC
	popq rA	
PC update	PC valP	Update PC
	jXX Dest	
PC update	PC Cnd ? valC : valP	Update PC
	call Dest	
PC update	PC valC	Set PC to destination
	ret	
PC update	PC valM	Set PC to return address

PC update module:-

```
module pc_update(cnd,clk,icode,new_PC,valM,valC,valP);

input [63:0] valC,valM,valP;
input clk,cnd;
input [3:0] icode;

output reg [63:0] new_PC; // updates pc value

always@(*)
begin
case(icode)
4'b0111:         ///jump
begin
    if(cnd)
        new_PC=valC;

    else
        new_PC=valP;
end

4'b1001:      ///return
begin
    new_PC=valM;
end

4'b1000:   ///call
begin
    new_PC=valC;
end

default:        ///default
begin
    new_PC=valP;
end

endcase

end
endmodule
```

Processor module:-

```
`include "fetch.v"
`include "decode.v"
`include "execute.v"
`include "memory.v"
`include "pc_update.v"

module processor();

reg clk;
reg [63:0] PC;
reg [0:79] instr;
wire [63:0] new_PC;
wire [3:0] icode, ifun, rA, rB;
wire signed [63:0] valA, valB, valC, valP, valE, valM;

wire ZF, SF, OF;
wire [2:0] CC_in;
wire [2:0] CC_out;
assign ZF=CC_in[0];
assign SF=CC_in[1];
assign OF=CC_in[2];

wire halt,cnd;
wire instr_valid;
wire memory_error;
reg [7:0] instr_mem [0 : 14];
wire signed [63:0] rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14;
wire signed [63:0] stored_in_mem,index;

fetch UUT_f(clk,icode,ifun,rA,rB,valC,valP,memory_error,instr_valid,PC);

decode UUT_d(clk, icode, rA, rB, valA, valB, valM, valE, cnd, rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14);

execute ext(clk,icode,ifun,valC,valA,valB,cnd,valE,ZF,SF,OF);

memory memo(clk, icode, valA, valB, valP, valE, valM, stored_in_mem,index);

pc_update P1(cnd,clk,icode,new_PC,valM,valC,valP);
```

```
initial begin
    PC = 64'd0;
    clk =0;
end
always @(*)
begin
    if(icode== 4'b0000 && ifun== 4'b0000 || !instr_valid ) // halt
      $finish;
end
always #20 clk = ~clk;
always@(*)
begin
    PC = new_PC;
end

initial
begin
    $monitor("  clk=%0d  PC=%0d icode=%0h  ifun=%0d  rA=%0d  rB=%0d
    valC=%0d  valP=%0d valA=%0d valB=%0d valM=%0d valE=%0d memory_error=%0d,
    halt=%0d\n\n,  rax =%d\n, rcx=%d\n, rdx=%d\n, rbx=%d\n, rsp=%d\n, rbp=%d\n,
    rsi=%d\n, rdi=%d\n, r8=%d\n, r9=%d\n, r10=%d\n, r11=%d\n, r12=%d\n, r13=%d\n,
    r14=%d\n\n  ZF=%d\n,SF=%d\n,OF=%d\n\n stored_in_mem =%d\n\n ", clk, PC, icode,
    ifun, rA, rB, valC, valP, valA, valB, valM, valE, memory_error, halt, rax, rcx,
    rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14,ZF,SF,OF,stored_in_mem);
end
endmodule
```

Instructions:-

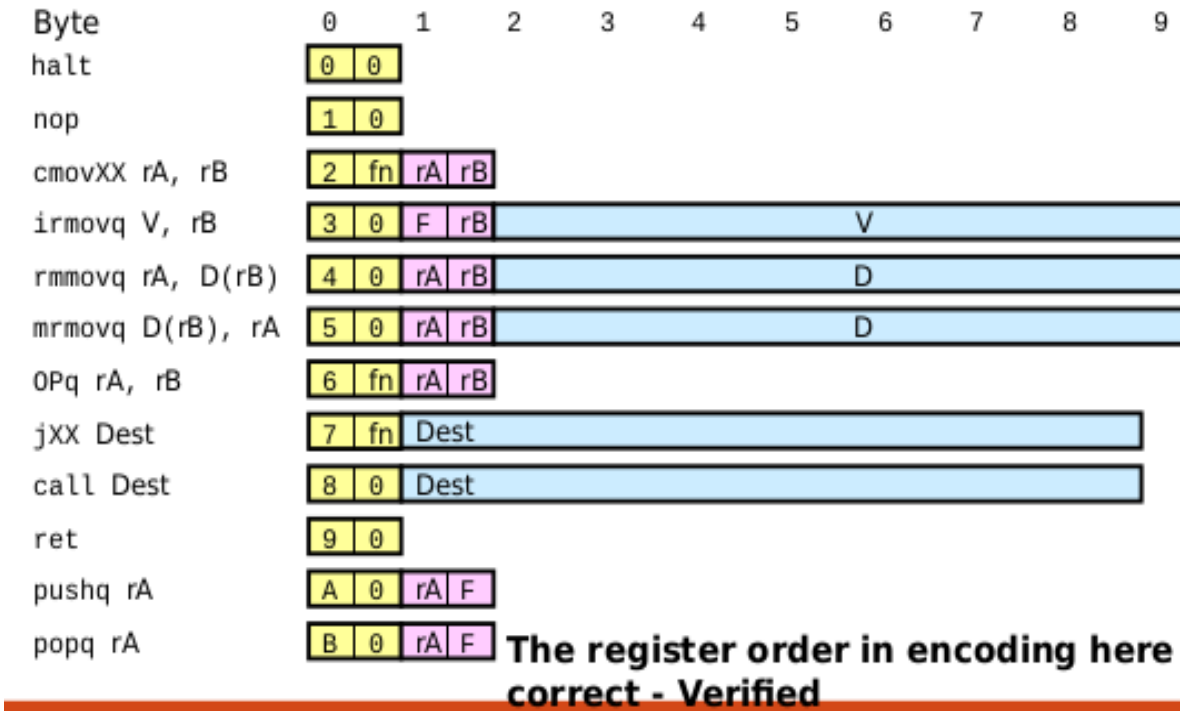
Here we have used 12 instructions are as follows:-

1. halt
2. nop
3. cmovXX
4. irmovq
5. rmmovq
6. mrmovq
7. OPq
8. jXX
9. call
10. ret
11. pushq
12. popq

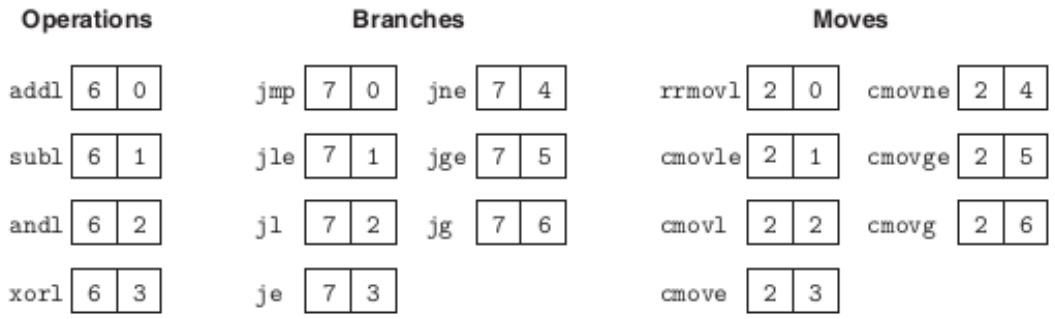
We have used 15 resgister as follows:-

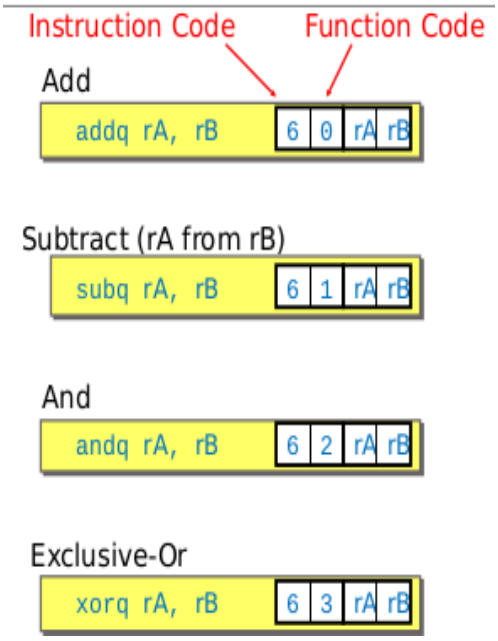
%rax	0
%rcx	1
%rdx	2
%rbx	3
%rsp	4
%rbp	5
%rsi	6
%rdi	7

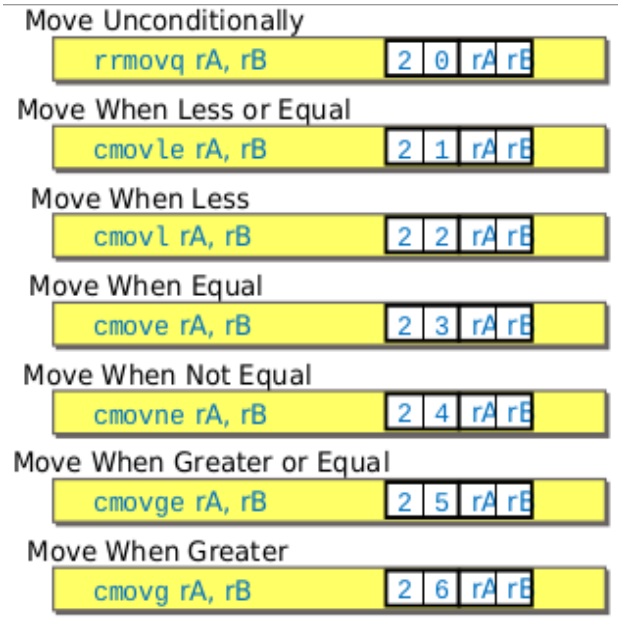
%r8	8
%r9	9
%r10	A
%r11	B
%r12	C
%r13	D
%r14	E
No Register	F

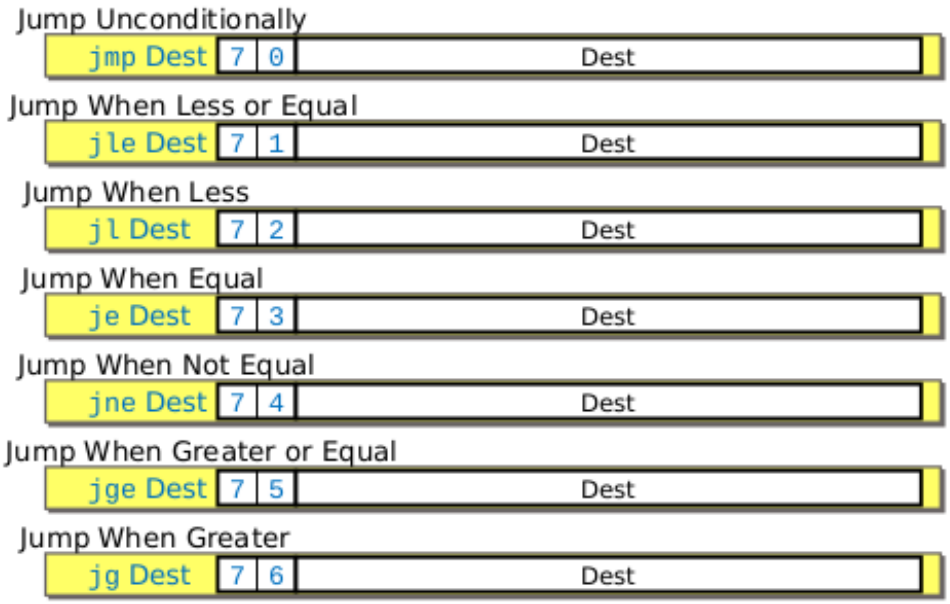


These are logic we have used for the operation, branches and for conditional moves for sequential implement.









Testcases:- I have build 5-testcases for showing each instruction.

i) test.txt:-

```
00110000
11110011
00000000
00000001
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110010
00000000
00000010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100000
00100011
```

Output:-

```
himanshu@himanshu-HP-Laptop:~/Desktop/SEQ0$ ./a.out
  clk=0  PC=0 icode=x  ifun=x  rA=x  rB=x  valC=x  valP=x valA=x valB=x valM=x valE=x memory_error=0, halt=z

,  rax =                0
, rcx=                 0
, rdx=                 0
, rbx=                 0
, rsp=                 0
, rbp=                 0
, rsi=                 0
, rdi=                 0
, r8=                 0
, r9=                 0
, r10=                 0
, r11=                 0
, r12=                 0
, r13=                 0
, r14=                 0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                x
```

```
  clk=1  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=256  valP=10 valA=x valB=x valM=x valE=256 memory_error=0, halt=z

,  rax =                   0
, rcx=                   0
, rdx=                   0
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=0  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=256  valP=10 valA=x valB=x valM=x valE=256 memory_error=0, halt=z

,  rax =                   0
, rcx=                   0
, rdx=                   0
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
  clk=0  PC=20 icode=3  ifun=0  rA=15  rB=2  valC=512  valP=20 valA=x valB=x valM=x valE=512 memory_error=0, halt=z

,  rax =                   0
, rcx=                   0
, rdx=                 512
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=22 icode=6  ifun=0  rA=2  rB=3  valC=512  valP=22 valA=512 valB=256 valM=x valE=768 memory_error=0, halt=z

,  rax =                   0
, rcx=                   0
, rdx=                 512
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
 clk=0  PC=22 icode=6  ifun=0  rA=2  rB=3  valC=512  valP=22 valA=512 valB=768 valM=x valE=1280 memory_error=0, halt=z

  rax =                  0
 rcx=                   0
 rdx=                 512
 rbx=                 768
 rsp=                   0
 rbp=                   0
 rsi=                   0
 rdi=                   0
 r8=                   0
 r9=                   0
 r10=                   0
 r11=                   0
 r12=                   0
 r13=                   0
 r14=                   0

  ZF=0
 SF=0
 OF=0

 stored_in_mem =                x


 clk=1  PC=22 icode=x  ifun=x  rA=2  rB=3  valC=512  valP=22 valA=512 valB=768 valM=x valE=1280 memory_error=0, halt=z

  rax =                  0
 rcx=                   0
 rdx=                 512
 rbx=                 768
 rsp=                   0
 rbp=                   0
 rsi=                   0
 rdi=                   0
 r8=                   0
 r9=                   0
 r10=                   0
 r11=                   0
 r12=                   0
 r13=                   0
 r14=                   0

  ZF=0
 SF=0
 OF=0

 stored_in_mem =                x
```

ii) texst5.txt

test5.txt

```
00110000    //irmovq
11110011    // at F  %rbx i.e 3
00010100    // 20
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00100000    // rrmovq
00110100    //  rbx -> rsp
10000000    // call
00011000    // at 24
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100010    // opq   for  xor->3  AND->2   SUB->1  ADD->0
00111000    // rbx ^ r8
00000000    // halt
00110000    //irmovq
11111000    // at F  %r8 i.e 8
00001111    // 15
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10010000    //return
```

Output:-

```
,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=256  valP=10 valA=x valB=x valM=x valE=256 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
  clk=0  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=256  valP=10 valA=x valB=x valM=x valE=256 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=20 icode=3  ifun=0  rA=15  rB=2  valC=512  valP=20 valA=x valB=x valM=x valE=512 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
  clk=0  PC=20 icode=3  ifun=0  rA=15  rB=2  valC=512  valP=20 valA=x valB=x valM=x valE=512 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                 512
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=22 icode=6  ifun=0  rA=2  rB=3  valC=512  valP=22 valA=512 valB=256 valM=x valE=768 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                 512
, rbx=                 256
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
  clk=0  PC=22 icode=6  ifun=0  rA=2  rB=3  valC=512  valP=22 valA=512 valB=768 valM=x valE=1280 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                 512
, rbx=                 768
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=22 icode=x  ifun=x  rA=2  rB=3  valC=512  valP=22 valA=512 valB=768 valM=x valE=1280 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                 512
, rbx=                 768
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                   0
, r9=                   0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

iii) test7.txt:-

test7.txt

```
00110000    //irmovq
11110010    // F  %rdx
00001001
00000000
00000000
00000000     // 9
00000000
00000000
00000000
00000000
00110000
11110100       // irmovq  $128  ,  %rsp
10000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10100000       //  push
00101111        //  %rdx  9
10110000       // pop
00001111       // %rax
00000000       // halt
```

Output:-

```
  clk=0  PC=0 icode=x  ifun=x  rA=x  rB=x  valC=x  valP=x valA=x valB=x valM=x valE=x memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 x


  clk=1  PC=10 icode=3  ifun=0  rA=15  rB=2  valC=9  valP=10 valA=x valB=x valM=x valE=9 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   0
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 x
```

```
  clk=0  PC=10 icode=3  ifun=0  rA=15  rB=2  valC=9  valP=10 valA=x valB=x valM=x valE=9 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x


  clk=1  PC=20 icode=3  ifun=0  rA=15  rB=4  valC=128  valP=20 valA=x valB=x valM=x valE=128 memory_error=0, halt=

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                   0
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  x
```

```
   clk=0  PC=20 icode=3  ifun=0  rA=15  rB=4  valC=128  valP=20 valA=x valB=x valM=x valE=128 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 128
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
 ,SF=0
 ,OF=0

 stored_in_mem =                  x


   clk=1  PC=22 icode=a  ifun=0  rA=2  rB=15  valC=128  valP=22 valA=9 valB=128 valM=x valE=120 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 128
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                    0
, r9=                    0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                  9
```

```
  clk=0  PC=22 icode=a  ifun=0  rA=2  rB=15  valC=128  valP=22 valA=9 valB=120 valM=x valE=112 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 120
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                  0
, r9=                  0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 9


  clk=1  PC=24 icode=b  ifun=0  rA=0  rB=15  valC=128  valP=24 valA=120 valB=120 valM=9 valE=128 memory_error=0, halt=z

,  rax =                  0
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 120
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                  0
, r9=                  0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 9
```

```
  clk=0  PC=24 icode=b  ifun=0  rA=0  rB=15  valC=128  valP=24 valA=128 valB=128 valM=9 valE=136 memory_error=0, halt=z

,  rax =                  9
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 128
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                  0
, r9=                  0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 x


  clk=1  PC=25 icode=0  ifun=0  rA=0  rB=15  valC=128  valP=25 valA=128 valB=128 valM=9 valE=136 memory_error=0, halt=z

,  rax =                  9
, rcx=                   0
, rdx=                   9
, rbx=                   0
, rsp=                 128
, rbp=                   0
, rsi=                   0
, rdi=                   0
, r8=                  0
, r9=                  0
, r10=                   0
, r11=                   0
, r12=                   0
, r13=                   0
, r14=                   0

  ZF=0
,SF=0
,OF=0

 stored_in_mem =                 x
```

iv) test8.txt:-

test8.txt

```
00110000   //irmovq
11110010   // F  %rdx
00001001
00000000
00000000
00000000    // 9
00000000
00000000
00000000
00000000
00110000    // irmovq
11110011    // F %rbx
00010101
00000000
00000000
00000000   // 21
00000000
00000000
00000000
00000000
00110000      // irmovq  $128  ,  %rsp
11110100
10000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01000000      //  rmmovq
01000011       //  %rsp  ,  100(%rbx)
01100100
00000000
00000000       //  100
00000000
00000000
00000000
00000000
00000000
```

test8.txt

```
01100001      // opq subq          setting ZF=0
00001101
01110011       // je   7  3
00110100       // will jump to 53
00000000
00000000
00000000
00000000
00000000      //  if cnd ? halt pe jayega
00000000
00000000
00000000  // halt  if not take jump
01010000 //  mrmovq
10000000 // %r8  112(%rax)
01110000
00000000
00000000
00000000    // 112
00000000
00000000
00000000
00000000
00000000  // halt
```

Output:-

```
clk=0  PC=0 icode=x  ifun=x  rA=x  rB=x  valC=x  valP=x valA=x valB=x valM=x valE=x memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                0, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=10 icode=3  ifun=0  rA=15  rB=2  valC=9  valP=10 valA=x valB=x valM=x valE=9 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                0, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=10 icode=3  ifun=0  rA=15  rB=2  valC=9  valP=10 valA=x valB=x valM=x valE=9 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=20 icode=3  ifun=0  rA=15  rB=3  valC=21  valP=20 valA=x valB=x valM=x valE=21 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=20 icode=3  ifun=0  rA=15  rB=3  valC=21  valP=20 valA=x valB=x valM=x valE=21 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x
```

```
clk=1  PC=30 icode=3  ifun=0  rA=15  rB=4  valC=128  valP=30 valA=x valB=x valM=x valE=128 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=30 icode=3  ifun=0  rA=15  rB=4  valC=128  valP=30 valA=x valB=x valM=x valE=128 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=40 icode=4  ifun=0  rA=4  rB=3  valC=100  valP=40 valA=128 valB=21 valM=x valE=121 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =              128


clk=0  PC=40 icode=4  ifun=0  rA=4  rB=3  valC=100  valP=40 valA=128 valB=21 valM=x valE=121 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =              128


clk=1  PC=42 icode=6  ifun=1  rA=0  rB=13  valC=100  valP=42 valA=0 valB=0 valM=x valE=0 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=42 icode=6  ifun=1  rA=0  rB=13  valC=100  valP=42 valA=0 valB=0 valM=x valE=0 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x
```

```
clk=1  PC=52 icode=7  ifun=3  rA=0  rB=13  valC=52  valP=51 valA=0 valB=0 valM=x valE=0 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=52 icode=7  ifun=3  rA=0  rB=13  valC=52  valP=51 valA=0 valB=0 valM=x valE=0 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=62 icode=5  ifun=0  rA=8  rB=0  valC=112  valP=62 valA=0 valB=0 valM=x valE=112 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=62 icode=5  ifun=0  rA=8  rB=0  valC=112  valP=62 valA=x valB=0 valM=x valE=112 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      x, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=63 icode=0  ifun=0  rA=8  rB=0  valC=112  valP=63 valA=x valB=0 valM=x valE=112 memory_error=0, halt=z
rax =                0, rcx=                0, rdx=                9, rbx=               21, rsp=              128, rbp=                0, rsi=                0, rdi=                0, r8=
      x, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=1, SF=0, OF=0
stored_in_mem =                x
```

v) test9.txt:-

test9.txt

```
00110000    //irmovq
11110011    // at F  %rbx i.e 3
00010100    // 20
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10000000  // call
00011000  // at 24
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100000   // opq   for    ADD->0
00111000    // rbx + r8
00100101    // cmovge
10001101
00000000  // halt
00110000    //irmovq
11111000    // at F  %r8 i.e 8
00001111    // 15
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10010000   //return
```

Output:-

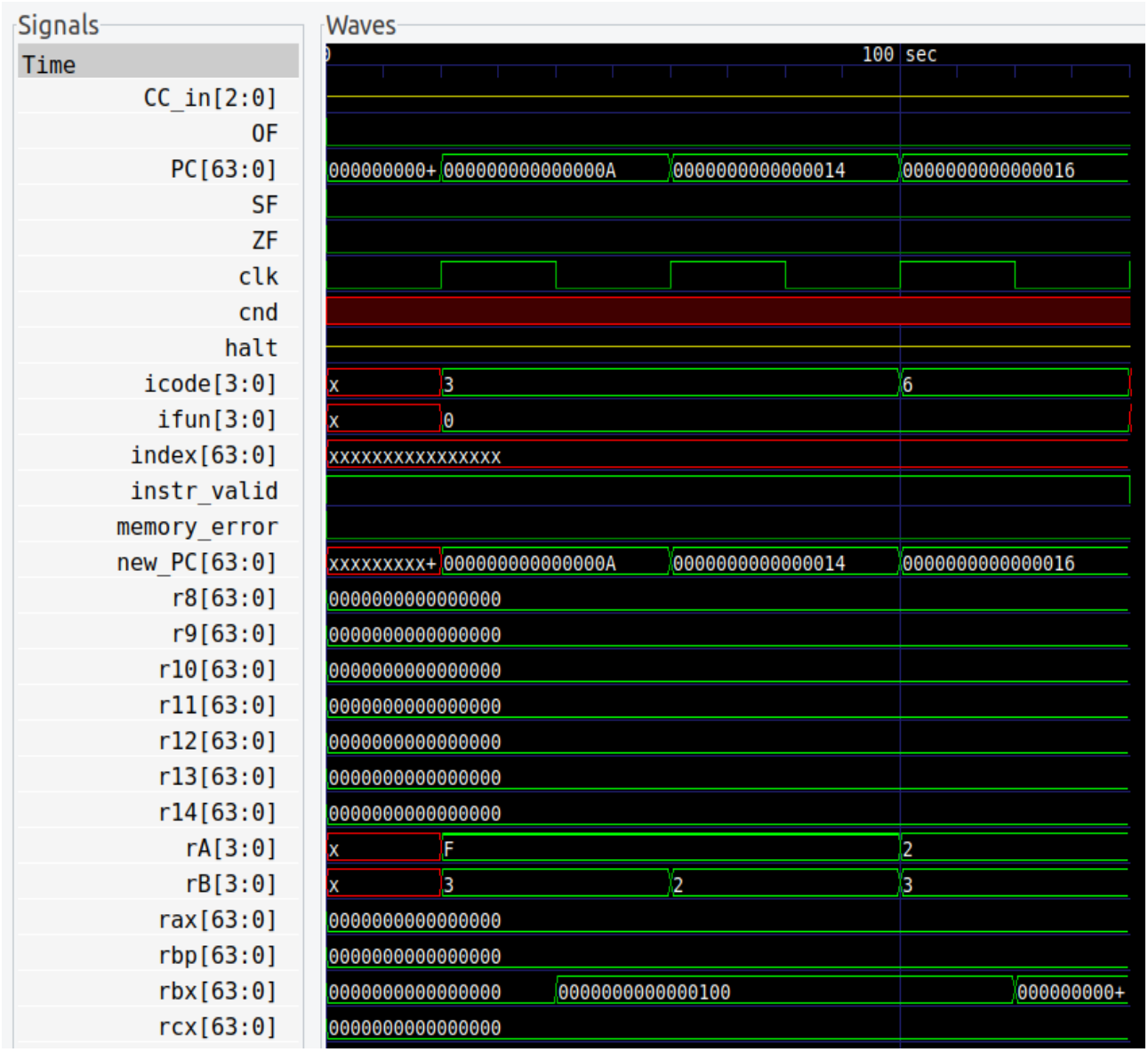
```
clk=0  PC=0 icode=x  ifun=x  rA=x  rB=x  valC=x  valP=x valA=x valB=x valM=x valE=x memory_error=0, halt=z
 rax =                0, rcx=                0, rdx=                0, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=1  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=20  valP=10 valA=x valB=x valM=x valE=20 memory_error=0, halt=z
 rax =                0, rcx=                0, rdx=                0, rbx=                0, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x


clk=0  PC=10 icode=3  ifun=0  rA=15  rB=3  valC=20  valP=10 valA=x valB=x valM=x valE=20 memory_error=0, halt=z
 rax =                0, rcx=                0, rdx=                0, rbx=               20, rsp=                0, rbp=                0, rsi=                0, rdi=                0, r8=
      0, r9=                0, r10=                0
r11=                0
r12=                0, r13=                0, r14=                0
ZF=0, SF=0, OF=0
stored_in_mem =                x
```

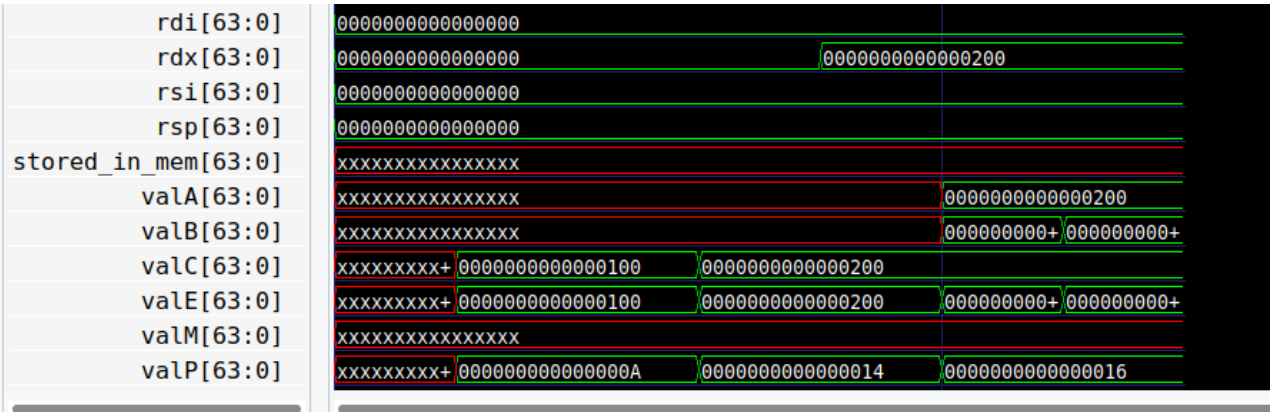
```
 clk=1  PC=24 icode=8  ifun=0  rA=15  rB=3  valC=24  valP=19 valA=x valB=0 valM=x valE=-8 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
       0, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=0  PC=24 icode=8  ifun=0  rA=15  rB=3  valC=24  valP=19 valA=x valB=-8 valM=x valE=-16 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                 -8, rbp=                  0, rsi=                  0, rdi=                  0, r8=
       0, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=1  PC=34 icode=3  ifun=0  rA=15  rB=8  valC=15  valP=34 valA=x valB=-8 valM=x valE=15 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                 -8, rbp=                  0, rsi=                  0, rdi=                  0, r8=
       0, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=0  PC=34 icode=3  ifun=0  rA=15  rB=8  valC=15  valP=34 valA=x valB=-8 valM=x valE=15 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                 -8, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      15, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=1  PC=x icode=9  ifun=0  rA=15  rB=8  valC=15  valP=35 valA=-8 valB=-8 valM=x valE=0 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                 -8, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      15, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                 19
```

```
 clk=0  PC=19 icode=9  ifun=0  rA=15  rB=8  valC=15  valP=35 valA=0 valB=0 valM=19 valE=8 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      15, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=1  PC=21 icode=6  ifun=0  rA=3  rB=8  valC=15  valP=21 valA=20 valB=15 valM=19 valE=35 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      15, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=0  PC=21 icode=6  ifun=0  rA=3  rB=8  valC=15  valP=21 valA=20 valB=35 valM=19 valE=55 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      35, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=1  PC=23 icode=2  ifun=5  rA=8  rB=13  valC=15  valP=23 valA=35 valB=35 valM=19 valE=35 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      35, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                  0, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=0  PC=23 icode=2  ifun=5  rA=8  rB=13  valC=15  valP=23 valA=35 valB=35 valM=19 valE=35 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      35, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                 35, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x


 clk=1  PC=24 icode=0  ifun=0  rA=8  rB=13  valC=15  valP=24 valA=35 valB=35 valM=19 valE=35 memory_error=0, halt=z
,  rax =                  0, rcx=                  0, rdx=                  0, rbx=                 20, rsp=                  0, rbp=                  0, rsi=                  0, rdi=                  0, r8=
      35, r9=                  0, r10=                  0
, r11=                 0
, r12=                 0, r13=                 35, r14=                 0
 ZF=0, SF=0, OF=0
 stored_in_mem =                  x
```

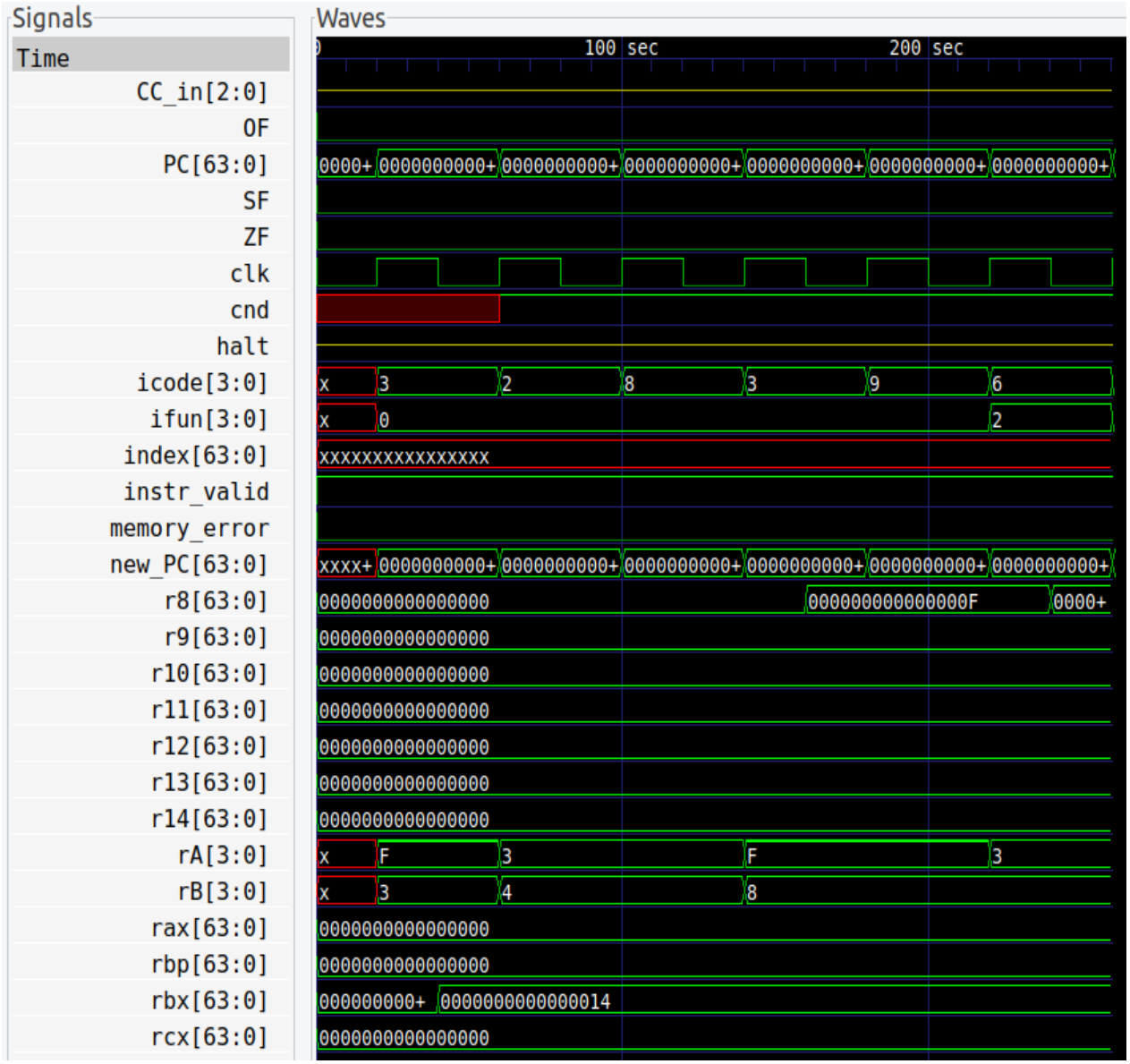
GTKwave Plots:-

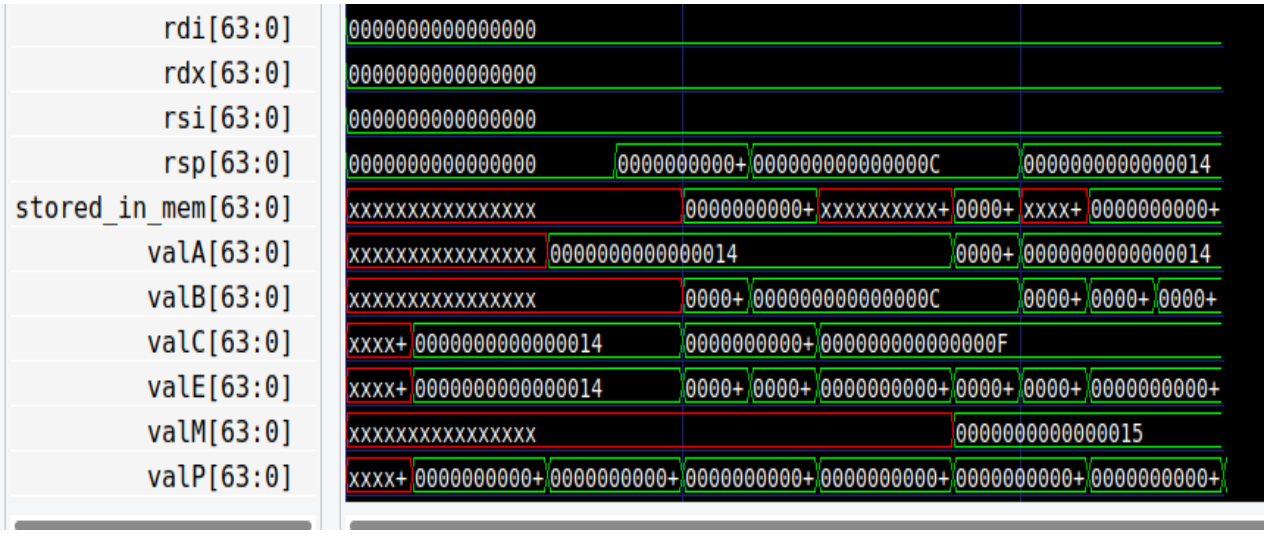
i) test.txt:-



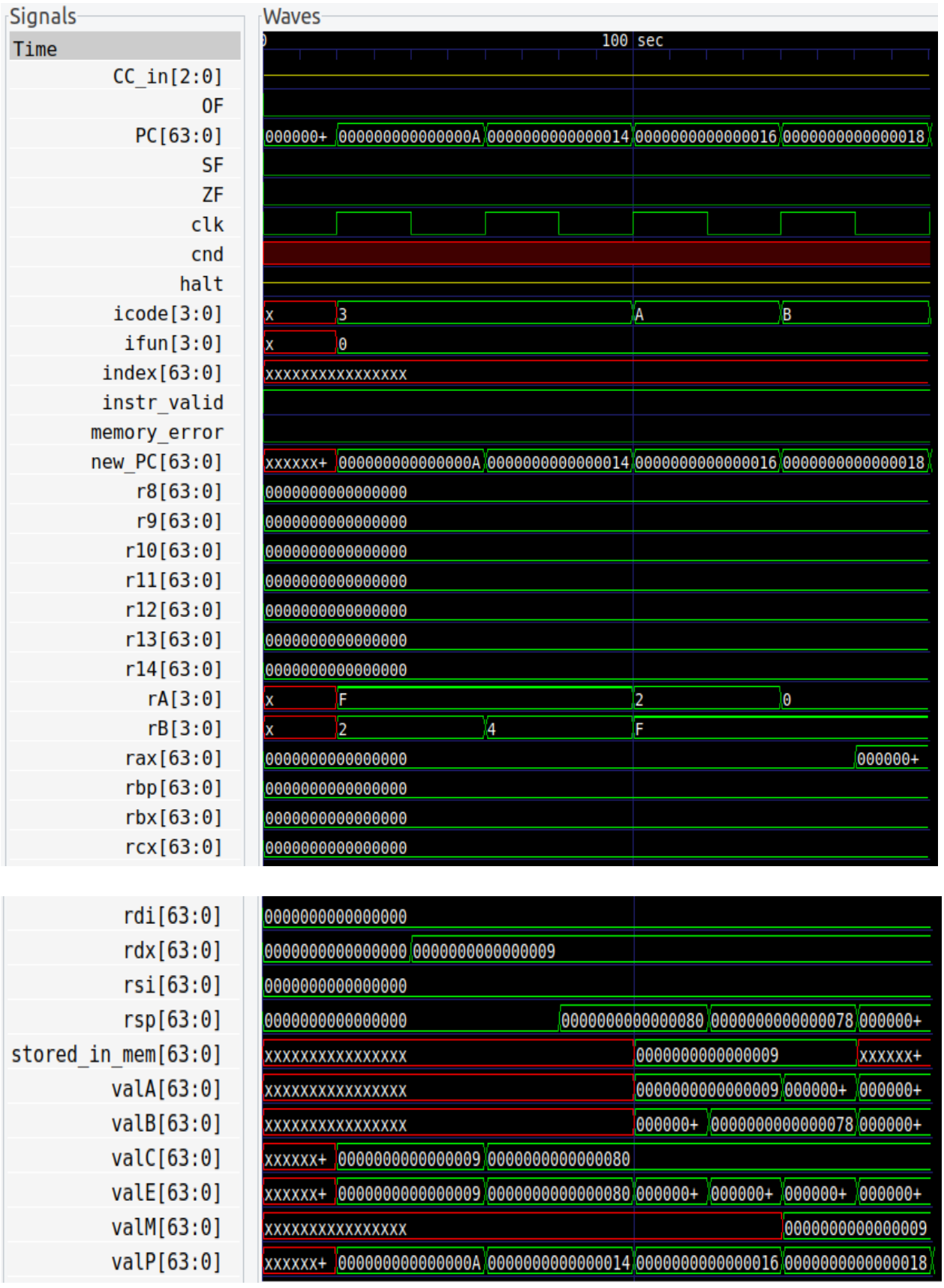


ii) test5.txt:-

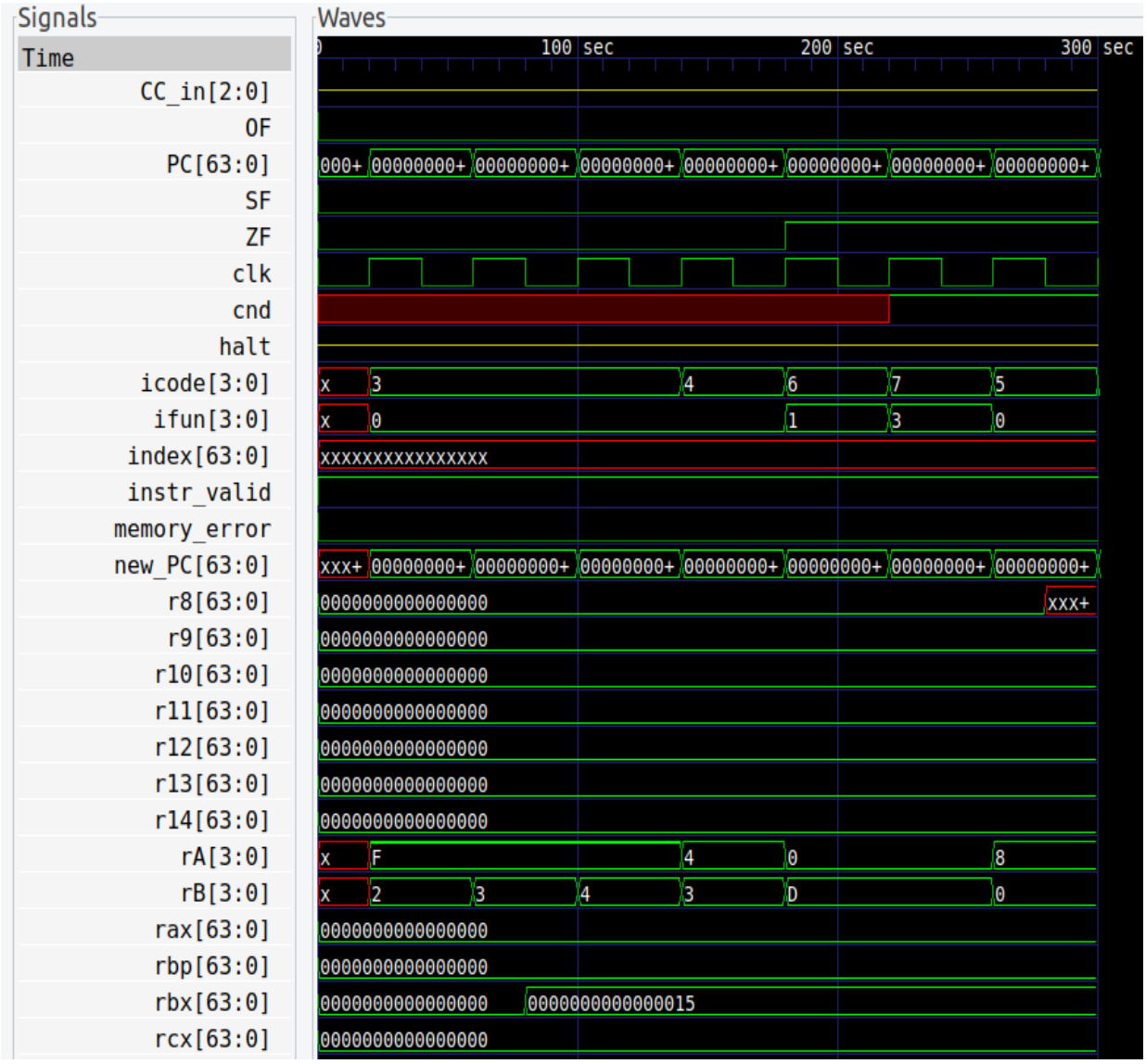

Signals
Waves
Time
CC_in[2:0]
OF
PC[63:0]
SF
ZF
clk
cnd
halt
icode[3:0]
ifun[3:0]
index[63:0]
instr_valid
memory_error
new_PC[63:0]
r8[63:0]
r9[63:0]
r10[63:0]
r11[63:0]
r12[63:0]
r13[63:0]
r14[63:0]
rA[3:0]
rB[3:0]
rax[63:0]
rbp[63:0]
rbx[63:0]
rcx[63:0]
100 sec
200 sec

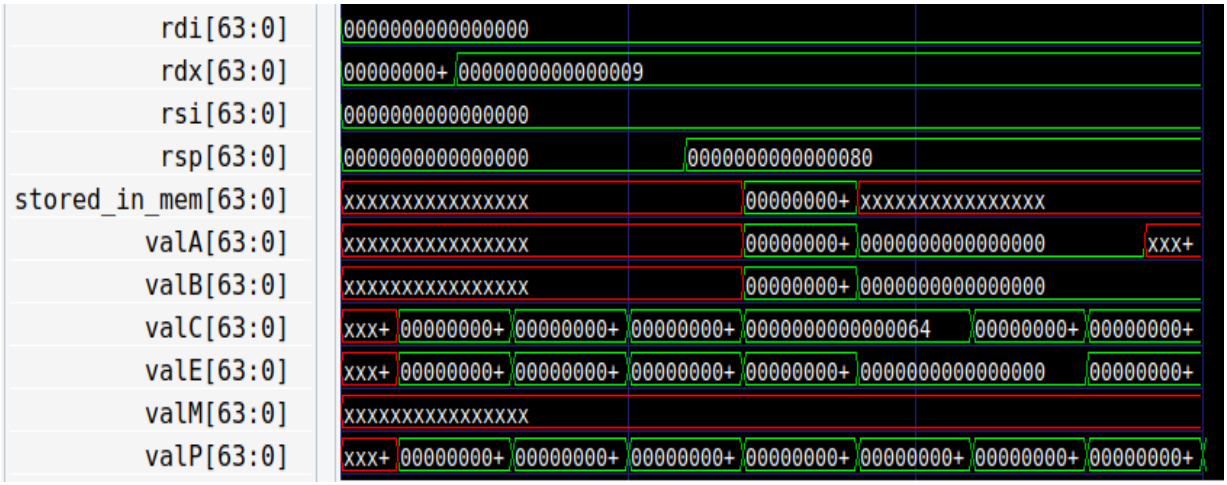

rdi[63:0]
rdx[63:0]
rsi[63:0]
rsp[63:0]
stored_in_mem[63:0]
valA[63:0]
valB[63:0]
valC[63:0]
valE[63:0]
valM[63:0]
valP[63:0]

iii) test7.txt:-

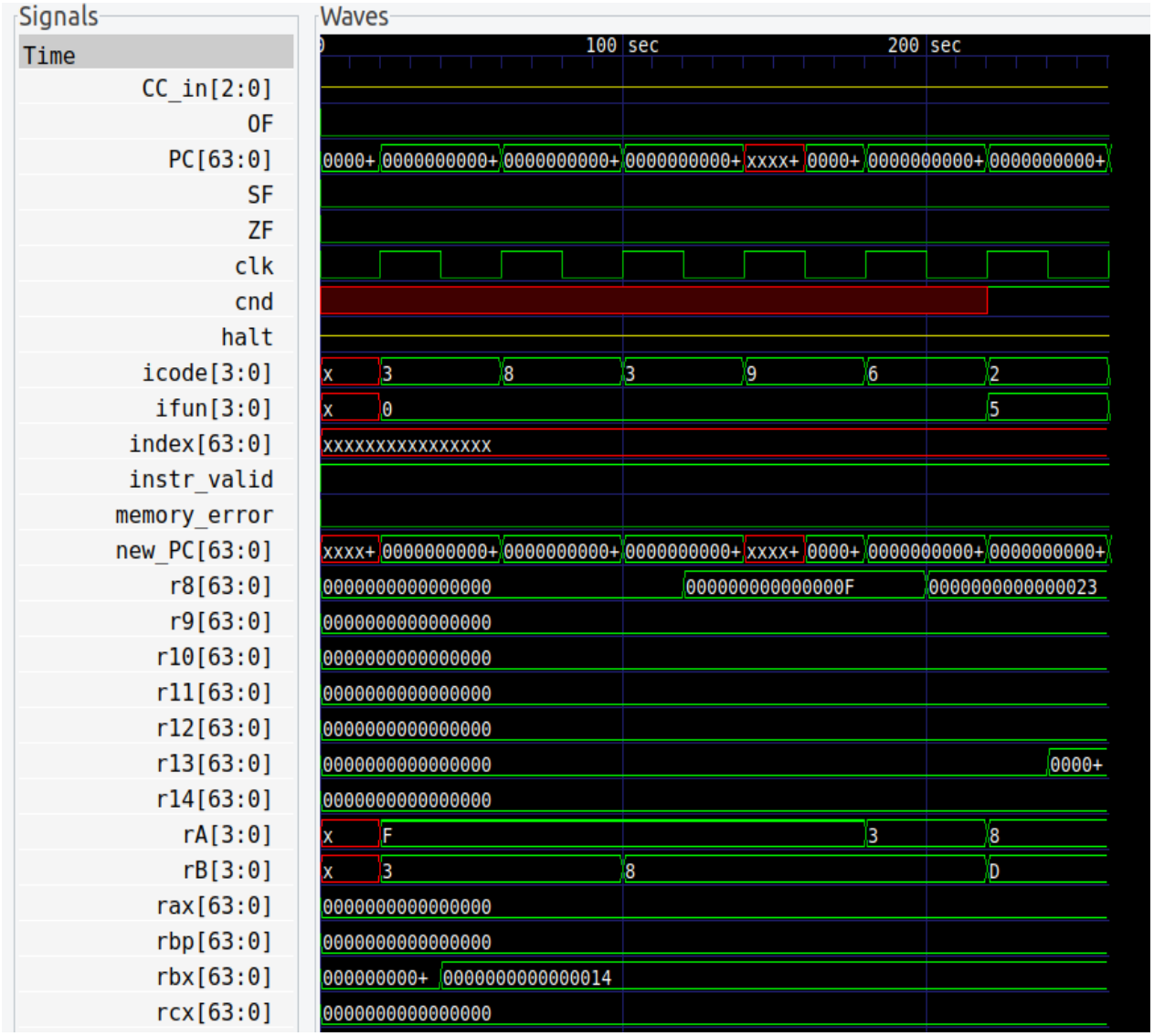


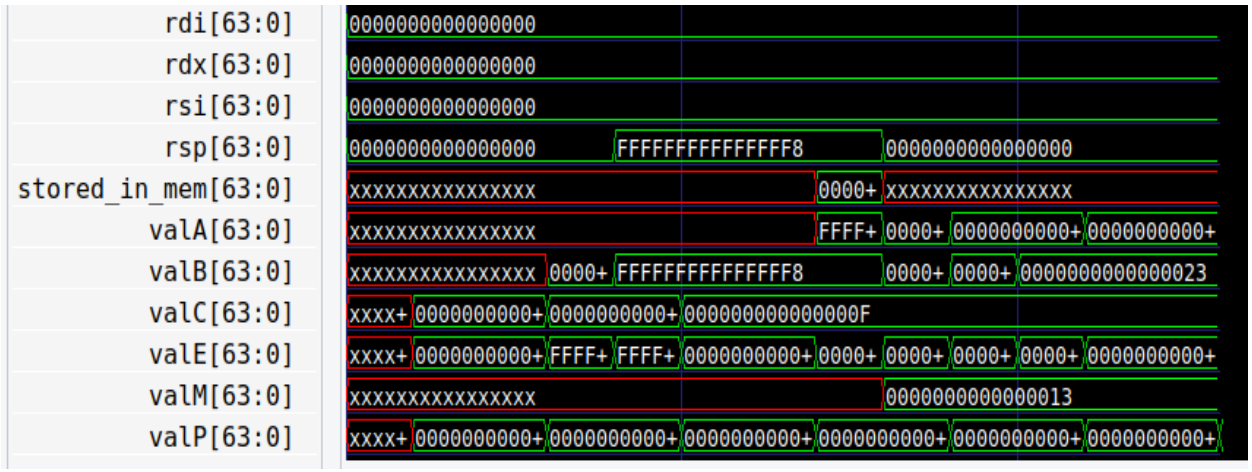
iv) test8.txt:-


Signals
Waves
Time
CC_in[2:0]
OF
PC[63:0]
SF
ZF
clk
cnd
halt
icode[3:0]
ifun[3:0]
index[63:0]
instr_valid
memory_error
new_PC[63:0]
r8[63:0]
r9[63:0]
r10[63:0]
r11[63:0]
r12[63:0]
r13[63:0]
r14[63:0]
rA[3:0]
rB[3:0]
rax[63:0]
rbp[63:0]
rbx[63:0]
rcx[63:0]
100 sec
200 sec
300 sec


rdi[63:0]
rdx[63:0]
rsi[63:0]
rsp[63:0]
stored_in_mem[63:0]
valA[63:0]
valB[63:0]
valC[63:0]
valE[63:0]
valM[63:0]
valP[63:0]

v) test9.txt:-


Signals
Waves
Time
CC_in[2:0]
OF
PC[63:0]
SF
ZF
clk
cnd
halt
icode[3:0]
ifun[3:0]
index[63:0]
instr_valid
memory_error
new_PC[63:0]
r8[63:0]
r9[63:0]
r10[63:0]
r11[63:0]
r12[63:0]
r13[63:0]
r14[63:0]
rA[3:0]
rB[3:0]
rax[63:0]
rbp[63:0]
rbx[63:0]
rcx[63:0]
100 sec
200 sec


rdi[63:0]
rdx[63:0]
rsi[63:0]
rsp[63:0]
stored_in_mem[63:0]
valA[63:0]
valB[63:0]
valC[63:0]
valE[63:0]
valM[63:0]
valP[63:0]

Pipeline implementation:-

Implementation of the pipeline involved the addition of the registers between each stage that can pass designated outputs from one stage register as inputs to the next stage register. The first step to pipelining is the rearrangement of computation stages and the inserting pipeline registers and rearranging and relabelling signals as per the given instruction.

We start by making a small adaptation of the sequential processor SEQ to shift the computation of the PC into the fetch stage.
We then add pipeline registers between the stages. Our first attempt at this does not handle the dif-ferent data and control dependencies properly. By making some modifications, however, we achieve our goal of an efficient pipelined processor that implements the Y86 ISA.

Another change is in the PC update. We do not have a separate PC update block, instead, we integrated PC update with the Fetch so that before every instruction executes, the Fetch stage will determine the current PC value and also the next predicted PC. In addition to these, there is a control logic too involved for Bubble, Stall and Forwarding implementation to eliminate pipeline hazards.

The Supported Instructions are as follows:-

1. halt
2. nop
3. cmovXX
4. irmovq
5. rmmovq
6. mrmovq
7. OPq
8. jXX
9. call
10. ret
11. pushq
12. popq

Pipeline stage's design approach:-

The Pipeline design is a better over sequential design in many ways.

1.Improved Performance:- Pipelining improves the performance of a processor by allowing it to execute multiple instructions at the same time. In a sequential design, the processor can only execute one instruction at a time. However, in a pipelined design, each stage of the pipeline can work on a separate instruction simultaneously, which leads to a significant increase in throughput.

2.Increased Efficiency:- Pipelining allows for more efficient use of the processor's resources. In a sequential design, many resources remain idle while waiting for a single instruction to be complete. However, in a pipelined design, the processor can start working on the next instruction as soon as the previous one enters the next stage of the pipeline. This reduces the amount of time that resources are idle, increasing efficiency.

3.Lower Latency:- Pipelining reduces the overall latency of the processor. In a sequential design, the time required to complete instruction is equal to the sum of the execution time of each individual stage. However, in a pipelined design, the time required to complete an instruction is equal to the time it takes to complete the slowest stage. This reduces the overall latency, resulting in faster execution times.

4.Scalability:- Pipelining allows for easy scalability of the processor. Adding additional stages to the pipeline can increase the performance of the processor further, making it possible to handle larger and more complex workloads.

Challenges Encountered:-

i) The major challenge we encountered is when we are trying to assign status code and trying to write the condition's for stall and bubble.

ii) Another major challenge was to figure out clock cycle and understand its working.

iii) We also faced challenge while implementing return operation.

iv) To figutre out control logic and implementing it correctly.

v) To go from the functional level design of the Y-86 64 processor to the hardware circuit level.

Problems faced in Pipeline Stages compare to Sequential Stages:-

1. Non-Uniform Delay in different Stages.
2. Register Overhead.
3. Data Dependencies in Processor.
4. Data Hazard.
5. Load/Use Hazard.
6. Mispredictions. (Memory-Register, Call, Return).

Methods we follow to overcome from above Problems-

1. Adding Pipeline Register.
2. PC Prediction and recovering from PC Misprediction.
3. Introducing NOP.
4. Stalling.
5. Data Forwarding.
6. Avoiding Load/Use Hazard.
7. Control for Mispredictions.
8. Signal naming convention.

Inserting pipeline registers:-

The next step to pipelining is inserting the pipeline registers. Here we rearrange some of the hardware and signals in the SEQ implmentation and insert pipeline register between each stage.

i) F register holds a predicted value of the PC.

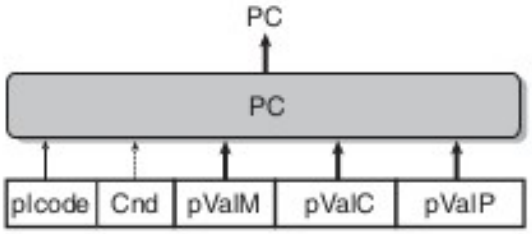
ii) D register holds information about the most recently fetched instruction for processing by the decode stage.

iii) E register holds information about the most recently decoded instruction and the values read from the register file.
iv) M register holds the results of the most recently executed instruction from memory stage and also information about branch conditions and branch targets for processing conditional jumps.

v) W register supply the computed results to the register file for writing and the return address to the PC selection logic.

Rearranging stages:-

For the pipelined implementation we should bring the PC update stage to the beginning of the cycle as we want to be able to continuously fetch the next instruction without having to wait for the PC update stage of the previous instruction to end had it been at the end of the cycle. This is known as circuit retiming.



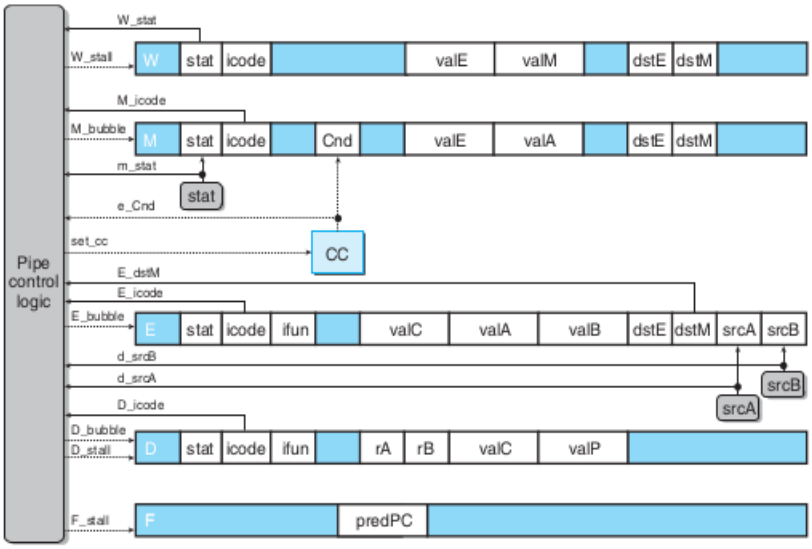
(b) SEQ+ PC selection

This changes the general sentation of the circuit without affecting its local behavior. Now the PC update stage at the beginning of the cycle can keep providing updated PC values to the fetch stage using the required values from different stages from instructions that have passed that stage.

Rearranging and relabelling signals:-

In the pipelined implementation the signals of an instruction pass through every stage one by one and these will have to be names with respect to the stage it is currently in as it is not possible to have one signal icode for all the 5 instructions running at the same time. So, to maintain the given signal at each stage we label them with respect to their stage as f_icode,d_icode,w_icode,etc.

Pipeline Control Design:-



Load/use hazards:- The pipeline must stall for one cycle between an instruction that reads a value from memory and an instruction that uses this value.

Processing 'ret':- The pipeline must stall until the ret instruction reaches the write-back stage.

Mispredicted branches:- By the time the branch logic detects that a jump should not have been taken, several instructions at the branch target will have started down the pipeline. These instructions must be canceled, and fetching should begin at the instruction following
the jump instruction.

Condition	Trigger
Processing ret	IRET in { D_icode, E_icode, M_icode }
Load/Use Hazard	E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB }
Mispredicted Branch	E_icode = IJXX & !e_Cnd

Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Load/Use Hazard	stall	stall	bubble	normal	normal
Mispredicted Branch	normal	bubble	bubble	normal	normal

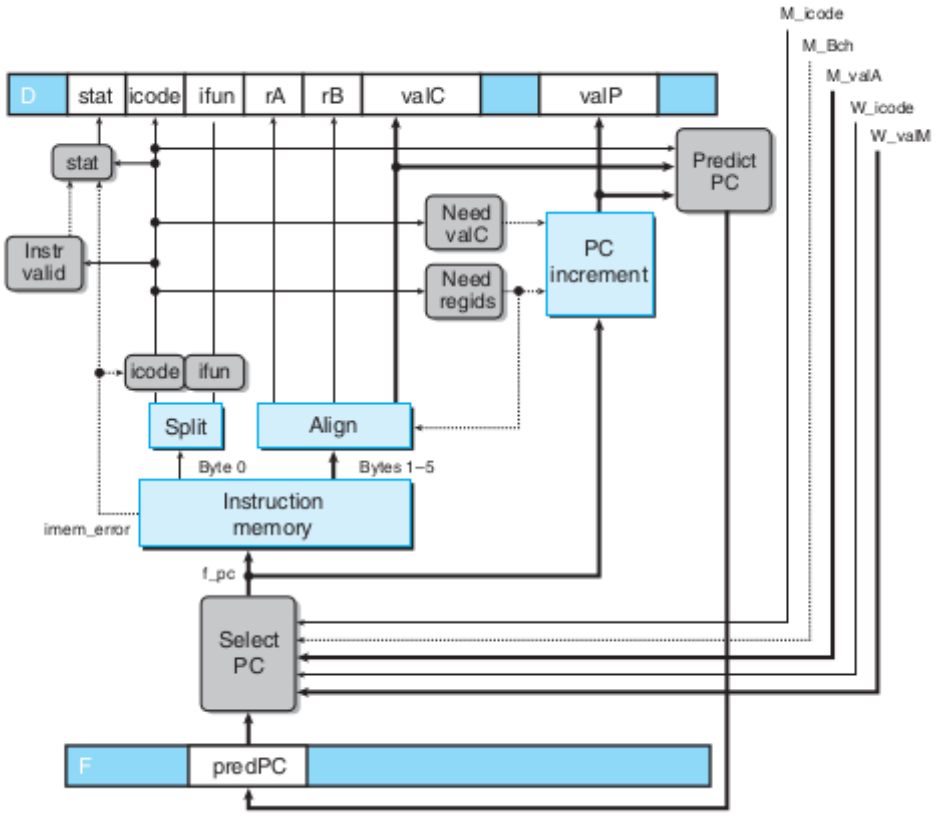
Exceptions:- When an instruction causes an exception, we want to disable the updating of the programmer-visible state by later instructions and halt execution once the excepting instruction reaches the write-back stage.

1.Fetch:-

In the fetch stage it takes the PC as input which is the predicted PC from the previous stage, and it selects the PC value based on the present Instructions and it outputs which the PC value we predicted for the next instruction. Now, this predicted PC we output will be assigned to PC which will serve as the input for the next instruction.

We are required to read instruction by instruction from the instruction memory and find the values of icode , ifun , rA , rB and valC according to the instruction.

The fetch stage retrieves the next instruction from the instruction memory using the PC, checks for boundary errors, and combines the 10-byte instruction into an 80-bit register for further processing in the pipeline. First Byte are the value of icode and ifun respectively, then second byte store the value rA and rB and rest is the value of the destination depends on the value of icode.



Working of fetch stage:-

i) On the positive edge of the clock signal, the fetch stage retrieves the updated PC value and uses it to access the instruction memory to fetch the next instruction.

ii) If the PC is out of bounds of the instruction memory, an memory error occurs, and the fetch stage performs a nop operation, causing all stages of the processor to idle for that clock cycle.

iii) If the PC is valid, the fetch stage retrieves 10 consecutive bytes of instruction from the instruction memory, starting from the PC.

iv) The 10 bytes of instruction are combined into a single 80-bit register called instr. This is done because the maximum size of a single instruction is 10 bytes.

Fetch module:-

```
module fetch(D_stat,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,f_predPC,M_icode,M_cnd,M_valA,W_icode,W_valM,F_predPC,clk,F_stall,D_stall,D_bubble);

input clk;
input M_cnd;
input [3:0] M_icode,W_icode;
input [63:0] M_valA,W_valM;
input [63:0] F_predPC; // This is the predicted value of PC which is coming from previous instruction
input F_stall;  // in stall case we keep our PC value fixed
input D_stall; // Conditions for load/use hazard
input D_bubble; // Conditions for Mispredicted branch

// D pipeline register
output reg [3:0] D_icode,D_ifun;
output reg [3:0] D_rA,D_rB;
output reg [63:0] D_valC,D_valP;
output reg [0:3] D_stat =4'b1000; // AOK,HLT,ADR,INS
output reg [63:0] f_predPC;  // we predict next PC value

reg [3:0] icode,ifun;
reg [3:0] rA,rB;
reg [63:0] valC,valP;
reg memory_error=0, instr_valid=1;
reg [0:3] stat;
reg [0:79] instr;
reg [7:0] instr_mem[0:255];
reg [63:0] PC;

initial
    PC = F_predPC;

always @(*) begin
    if(W_icode==4'h9) // ret
        PC = W_valM;

    else if(M_icode==4'h7) // jxx
    begin
        if(!M_cnd)
            PC= M_valA;
    end

    else
        PC = F_predPC;
end
```

```
initial
begin

   $readmemb("test.txt",instr_mem,0,80);

end

always @(*)
begin

    instr_valid=1;
    if(PC>1023)
    begin
        memory_error=1;
    end

    icode = instr_mem[PC][7:4];
    ifun  = instr_mem[PC][3:0];

    if(icode == 4'b0000 || icode == 4'b0001)   //halt  //nop
    begin
        valP = PC + 1;
        f_predPC = valP;

    end

if(icode == 4'b0010 ||  icode == 4'b0011 || icode == 4'b0100 || icode == 4'b0101 ||  icode == 4'b0110)  //cmovq //irmovq  //rmmovq  //mrmovq  // OPq

    begin
        rA = instr_mem[PC+1][7:4];
        rB = instr_mem[PC+1][3:0];

        if(icode == 4'b0010 || icode == 4'b0110)  //cmovq  //Opq
        begin
        valP=PC+2;
        f_predPC=valP;
        end

        if(icode == 4'b0011 || icode == 4'b0100 || icode == 4'b0101) //irmovq  //rmmovq  //mrmovq
        begin
        valC = {instr_mem[PC+9],instr_mem[PC+8],instr_mem[PC+7],instr_mem[PC+6],instr_mem[PC+5],instr_mem[PC+4],instr_mem[PC+3],instr_mem[PC+2]};
        valP = PC+10;
        f_predPC=valP;
        end
    end
```

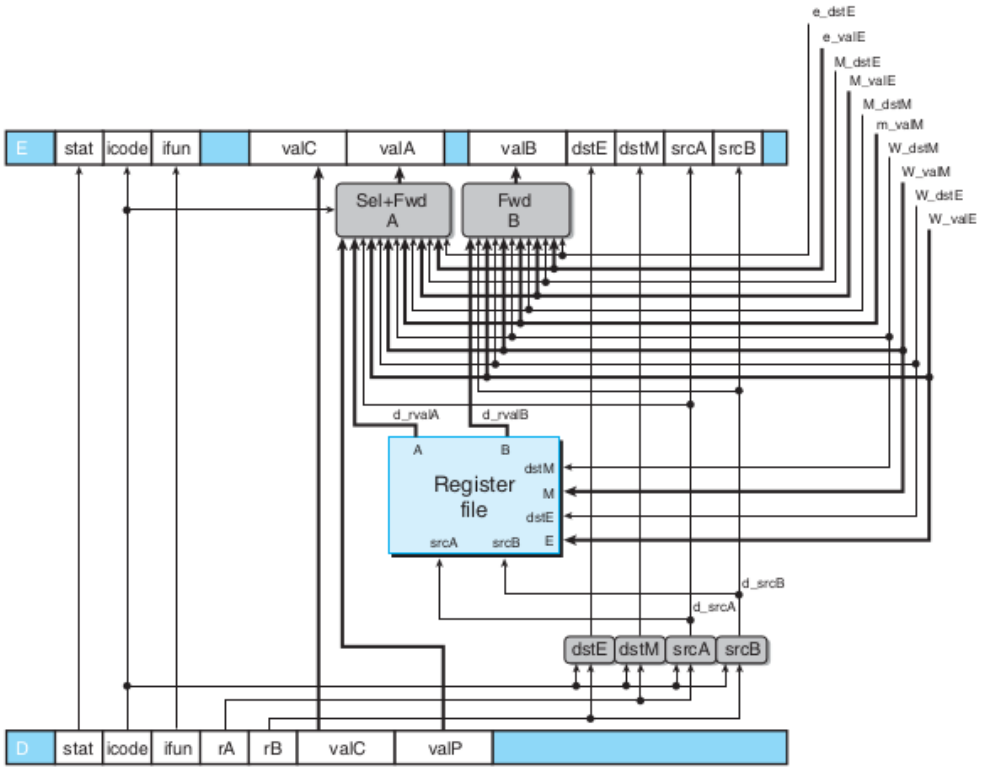
```
    if(icode == 4'b0111 || icode == 4'b1000 )  //call //jxx
    begin
        valC = {instr_mem[PC+8],instr_mem[PC+7],instr_mem[PC+6],instr_mem[PC+5],instr_mem[PC+4],instr_mem[PC+3],instr_mem[PC+2],instr_mem[PC+1]};

        valP=PC+9;
        f_predPC=valC; // In pipeline we assume initailly jump is taken
    end


  if(icode == 4'b1001) //ret
    begin
        valP=PC+1;
    end


   if(icode ==  4'hA || icode == 4'hB) //pushq   //popq
    begin
        rA = instr_mem[PC+1][7:4];
        rB = instr_mem[PC+1][3:0];
        valP=PC+2;
        f_predPC=valP;
    end


    if( icode > 12)
        instr_valid=1'b0;


    if(instr_valid==1'b0) // If the instruction is invalid
        stat = 4'b0001;

    else if(memory_error==1) // for memory address
    begin
        stat = 4'b0010;
    end

    else if(icode==4'b0000) // for halting
        stat = 4'b0100;
    else            // ALL OK (MSB is 1 and rest are 0)
        stat = 4'b1000;

end
```

```
always @(posedge clk)
begin
    if(F_stall)
    begin
        PC = F_predPC; // we keep our PC value same as previous
    end
    else if(D_bubble) // In this case we just execute of nop
    begin
        D_icode <= 4'b0001;
        D_ifun <= 4'b0000;
        D_rA <= 4'b0000;
        D_rB <= 4'b0000;
        D_valC <= 64'b0;
        D_valP <= 64'b0;
        D_stat <= 4'b1000; // ALL OK status
    end
    else
    begin
        D_icode <= icode;
        D_ifun <= ifun;
        D_rA <= rA;
        D_rB <= rB;
        D_valC <= valC;
        D_valP <= valP;
        D_stat <= stat;
    end
end

endmodule
```

2.Decode and Write-Back Stage:-

No instruction requires both valP and the value read from register port A, and so these two can be merged to form the signal valA for later stages. The block labeled “Sel+Fwd A” performs this task and also implements the forwarding logic for source operand valA.

The block labeled “Fwd B” implements the forwarding logic for source operand valB. The register write locations are specified by the dstE and dstM signals from the write back stage rather than from the decode stage, since it is writing the results of the instruction currently in the write-back stage.



Working of decode and write back stage:-

i) This block in a computing system that takes inputs from the decode register, including D_icode, D_ifun, D_rA, D_rB, D_valC, D_stat, D_valP, as well as inputs required for data forwarding.

ii)The block has two additional components, Sel+Fwd A and Fwd B, that facilitate data forwarding and allow for the direct retrieval of values for valA or valB from different stages.

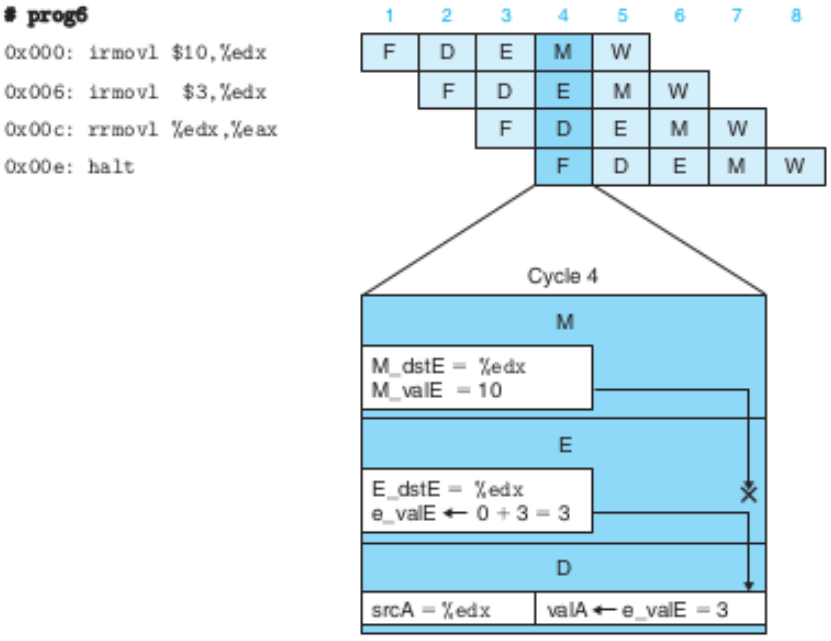
iii) The outputs of this block include E_stat, E_ifun, E_icode, E_valA, E_valB, E_valC, dstE, dstM, srcA, and srcB. The inputs function similarly to those in sequential computing, and the block also utilizes 14 registers that are updated during the writeback process.

iv) This block in a computing system takes inputs from the decode register and other sources to facilitate data forwarding and generate specific outputs for further processing.

There are five different forwarding sources, each with a data word and a destination register ID:-

Data word	Register ID	Source description
e_valE	e_dstE	ALU output
m_valM	M_dstM	Memory output
M_valE	M_dstE	Pending write to port E in memory stage
W_valM	W_dstM	Pending write to port M in write-back stage
W_valE	W_dstE	Pending write to port E in write-back stage

Forwarding Priority-



In cycle 4, values for %rdx are available from both the execute and memory stages. The forwarding logic should choose the one in the execute stage, since it represents the most recently generated value for this register.

Note:- The D_stat will check the status of the program and it a register of size 3 which has the following bits:

→ AOK: This is assigned to 1 if everything is okay in the program else 0

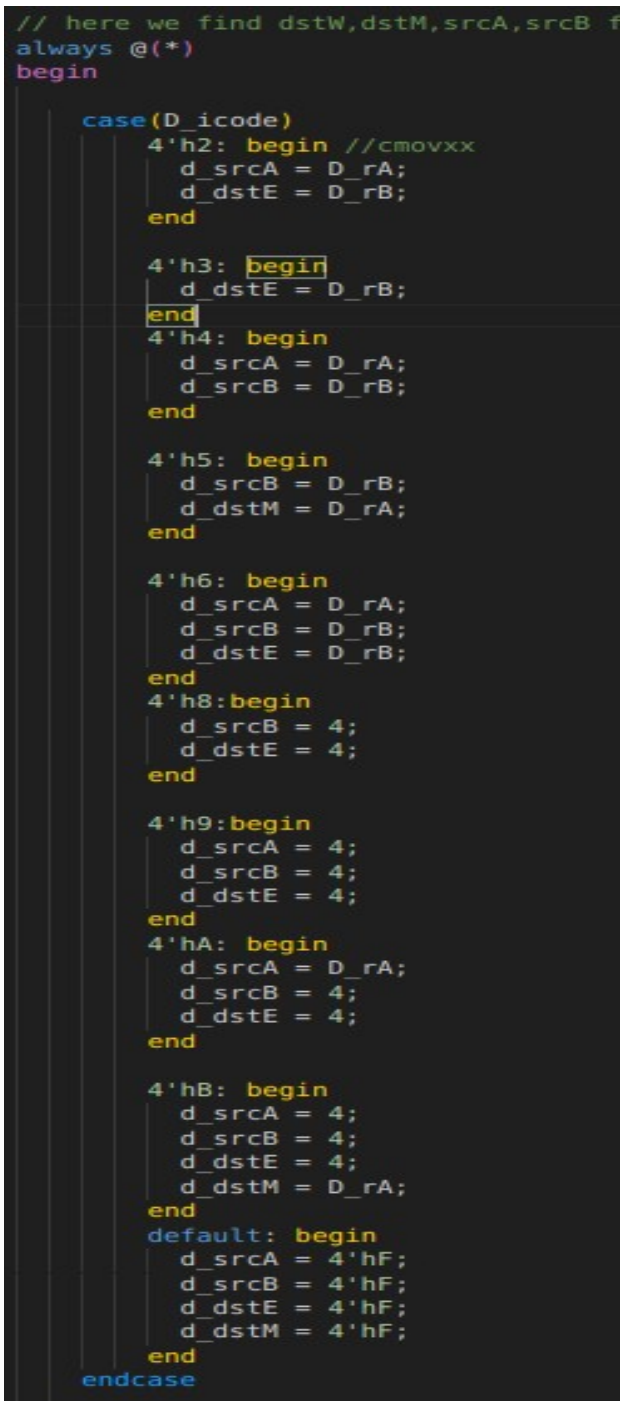
→ HLT: This is assigned to 1 if there is halt instruction else 0

→ INS/MEM: This is assigned to 1 if there is an invalid instruction or memory address else 0.

Decode and write back module:-

```
module decode (clk,D_icode,D_ifun,D_rA,D_rB,D_stat,D_valC,D_valP,E_bubble,W_icode,
e_dstE, M_dstE,M_dstM,W_dstE,W_dstM,e_valE,M_valE,m_valM,W_valE,W_valM,E_stat,E_icode,
E_ifun,E_valC, E_valA,E_valB,E_dstE,E_dstM,E_srcA,E_srcB, rax, rcx, rdx, rbx, rsp, rbp,
rsi, rdi, r8, r9, r10, r11, r12, r13, r14);

input clk;
input [3:0] D_icode,D_ifun,D_rA,D_rB;
input [63:0] D_valC,D_valP;
input E_bubble;
input [3:0] D_stat; // status coming from the decode pipeline register

input [3:0] W_icode;
input [3:0] e_dstE,M_dstE,M_dstM,W_dstE,W_dstM;
input [63:0] e_valE,M_valE,m_valM,W_valE,W_valM;

output reg [0:3] E_stat;
output reg [3:0] E_icode,E_ifun;
output reg [63:0] E_valC,E_valA,E_valB;
output reg [3:0] E_dstE,E_dstM,E_srcA,E_srcB;

output reg [63:0] rax;
output reg [63:0] rcx;
output reg [63:0] rdx;
output reg [63:0] rbx;
output reg [63:0] rsp;
output reg [63:0] rbp;
output reg [63:0] rsi;
output reg [63:0] rdi;
output reg [63:0] r8;
output reg [63:0] r9;
output reg [63:0] r10;
output reg [63:0] r11;
output reg [63:0] r12;
output reg [63:0] r13;
output reg [63:0] r14;

reg [63:0] register_memory[0:14];
reg [3:0] d_srcA,d_srcB;
reg [3:0] d_dstE,d_dstM;
reg [63:0] d_valA,d_valB;
```

```
initial begin
    register_memory[0]=0;
    register_memory[1]=0;
    register_memory[2]=0;
    register_memory[3]=0;
    register_memory[4]=0;
    register_memory[5]=0;
    register_memory[6]=0;
    register_memory[7]=0;
    register_memory[8]=0;
    register_memory[9]=0;
    register_memory[10]=0;
    register_memory[11]=0;
    register_memory[12]=0;
    register_memory[13]=0;
    register_memory[14]=0;
end
```



```
// here we find dstW,dstM,srcA,srcB f
always @(*)
begin

    case(D_icode)
        4'h2: begin //cmovxx
          d_srcA = D_rA;
          d_dstE = D_rB;
        end

        4'h3: begin
          d_dstE = D_rB;
        end
        4'h4: begin
          d_srcA = D_rA;
          d_srcB = D_rB;
        end

        4'h5: begin
          d_srcB = D_rB;
          d_dstM = D_rA;
        end

        4'h6: begin
          d_srcA = D_rA;
          d_srcB = D_rB;
          d_dstE = D_rB;
        end
        4'h8:begin
          d_srcB = 4;
          d_dstE = 4;
        end

        4'h9:begin
          d_srcA = 4;
          d_srcB = 4;
          d_dstE = 4;
        end
        4'hA: begin
          d_srcA = D_rA;
          d_srcB = 4;
          d_dstE = 4;
        end

        4'hB: begin
          d_srcA = 4;
          d_srcB = 4;
          d_dstE = 4;
          d_dstM = D_rA;
        end
        default: begin
          d_srcA = 4'hF;
          d_srcB = 4'hF;
          d_dstE = 4'hF;
          d_dstM = 4'hF;
        end
    endcase
```

This code block represents the decode stage of a pipeline processor. It determines the register operands used by the instruction being executed and sets the register destinations for the result of the operation. The specific values assigned depend on the instruction code (D_icode). If the code is not recognized, the default values of all operands and destinations are set to 4'hF. This block also sets up the registers used for forwarding in the subsequent stage of the pipeline.

```
if(D_icode == 4'b0010 || D_icode == 4'b0100 || D_icode == 4'b0110 || D_icode == 4'b1010)   //cmovxx //rmmovq  //opq  //pushq
begin
    d_valA = register_memory[D_rA];
    if(D_icode == 4'b0100 || D_icode == 4'b0110)   //rmmovq  //opq
    begin
    d_valB = register_memory[D_rB];
    end

    if(D_icode == 4'b1010) //pushq
    begin
    d_valB = register_memory[4];
    end
end

if(D_icode == 4'b0101) //mrmovq
begin
    d_valB = register_memory[D_rB];
end

if(D_icode == 4'b1000 || D_icode == 4'b1001 || D_icode == 4'b1011) //call  //ret  //popq
begin
    d_valB = register_memory[4];  // As we have to push address to fnext instruction onto stack
    if(D_icode == 4'b1001 || D_icode == 4'b1011)  //ret  //popq
    begin
      d_valA = register_memory[4];
    end
end
```

```
    // Forwarding A
    if(D_icode == 4'h7 | D_icode == 4'h8) //jxx or call
      d_valA = D_valP;

    else if(d_srcA == e_dstE & e_dstE != 4'hF)
      d_valA = e_valE;

    else if(d_srcA == M_dstM & M_dstM != 4'hF)
      d_valA = m_valM;

    else if(d_srcA == W_dstM & W_dstM != 4'hF)
      d_valA = W_valM;

    else if(d_srcA == M_dstE & M_dstE != 4'hF)
      d_valA = M_valE;

    else if(d_srcA == W_dstE & W_dstE != 4'hF)
      d_valA = W_valE;


    // Forwarding B
    if(d_srcB == e_dstE & e_dstE != 4'hF)
      d_valB = e_valE;

    else if(d_srcB == M_dstM & M_dstM != 4'hF)
      d_valB = m_valM;

    else if(d_srcB == W_dstM & W_dstM != 4'hF)
      d_valB = W_valM;

    else if(d_srcB == M_dstE & M_dstE != 4'hF)
      d_valB = M_valE;

    else if(d_srcB == W_dstE & W_dstE != 4'hF)
      d_valB = W_valE;

  end
```

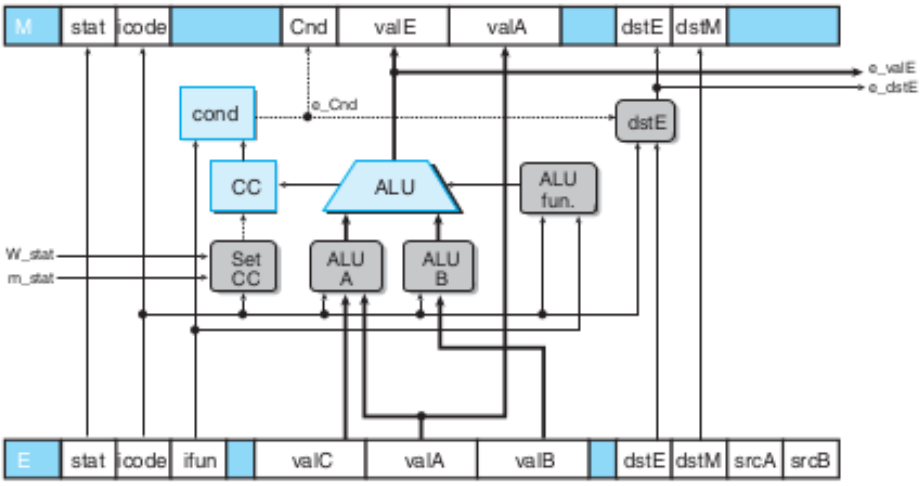
This code is for the decode stage of a pipelined processor. It determines the values of d_valA and d_valB based on the instruction being decoded and performs forwarding of values fromearlier pipeline stages if necessary.

```
//here we are updating the values in Execute pipeline register
always @(posedge clk)
begin

    if(E_bubble)
     begin
       E_stat <= 4'b1000;
       E_icode <= 4'b0001;
       E_ifun <= 4'b0000;
       E_valC <= 4'b0000;
       E_valA <= 4'b0000;
       E_valB <= 4'b0000;
       E_dstE <= 4'hF;
       E_dstM <= 4'hF;
       E_srcA <= 4'hF;
       E_srcB <= 4'hF;
     end

     else
     begin
       E_stat <= D_stat;
       E_icode <= D_icode;
       E_ifun <= D_ifun;
       E_valC <= D_valC;
       E_valA <= d_valA;
       E_valB <= d_valB;
       E_srcA <= d_srcA;
       E_srcB <= d_srcB;
       E_dstE <= d_dstE;
       E_dstM <= d_dstM;
     end

end
```

```
// Write back
always @(posedge clk)
begin

    if(W_icode == 4'b0010 || W_icode == 4'b0011 || W_icode == 4'b0110 || W_icode == 4'b1000 || W_icode == 4'b1001 || W_icode== 4'hA)   //cmovXX  //irmovq  //opq //Dest //ret //push
    begin
      register_memory[W_dstE] = W_valE; // updating %rsp
    end

    if(W_icode == 4'b0101) // mrrmovq D(rB),rA
    begin
      register_memory[W_dstM]= W_valE;
    end

    if(W_icode==4'b1011) // pop
    begin
        register_memory[W_dstE] =W_valE;
        register_memory[W_dstM]=W_valM;
    end

    rax <= register_memory[0];
    rcx <= register_memory[1];
    rdx <= register_memory[2];
    rbx <= register_memory[3];
    rsp <= register_memory[4];
    rbp <= register_memory[5];
    rsi <= register_memory[6];
    rdi <= register_memory[7];
    r8  <= register_memory[8];
    r9  <= register_memory[9];
    r10 <=register_memory[10];
    r11 <=register_memory[11];
    r12 <=register_memory[12];
    r13 <=register_memory[13];
    r14 <=register_memory[14];

end

endmodule
```

3.Execute Stage:-

The execute block serves the purpose of executing instructions and performing necessary calculations using the Arithmetic Logic Unit (ALU) to determine the effective address or value. It also sets the required condition codes by utilizing pre-built blocks. The execute block takes in icode, ifun, valC, valA, and valB as inputs, and produces Cnd and valE as outputs.



Working of execute stage:-

i) The execute block takes in various inputs, including E_stat, E_ifun, E_icode, E_valA, E_valB, E_valC, E_dstE, and E_dstM, which are outputs from the execute pipelined register.

ii) These inputs are processed to generate multiple outputs, such as M_stat, M_icode, Cnd, M_valE,M_valA, M_dstE, M_dstM, e_valE, and e_dstE.

iii) This process is similar to the sequential method of computing outputs. The value of e_dstE is determined based on the e_Cnd, which may result in either E_dstE or an empty register. The W_stat and m_stat ensure that the condition codes are not altered when the instruction is halted.

iv) The register is updated once the clock signal is received, and the M_stat, M_icode, M_valA, M_valE, M_dstE, and M_dstM outputs are available as an output of the memory register.

Execute module:-

```
`include "ALU.v"

module execute(clk,E_icode,E_ifun,E_valC,E_valA,E_valB,E_dstE,E_dstM,E_srcA,E_srcB,E_stat,W_stat,m_stat,M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,e_dstE,e_valE,e_cnd,ZF,SF,OF);

input clk;
input [3:0] E_icode,E_ifun;
input [63:0] E_valA,E_valB,E_valC;
input [3:0] E_dstE,E_dstM;
input [3:0] E_srcA,E_srcB;
input [0:3] E_stat;

output reg ZF,SF,OF;
initial begin
    ZF =0;
    SF =0;
    OF =0;
end

input [0:3] W_stat;
input [0:3] m_stat;


output reg [0:3] M_stat;
output reg [3:0] M_icode;
output reg M_cnd;
output reg [63:0] M_valE,M_valA;
output reg [3:0] M_dstE,M_dstM;


output reg [3:0] e_dstE;
output reg [63:0] e_valE;
output reg e_cnd=1;

reg [2:0] CC =3'b000;

reg [1:0] control;
reg signed [63:0]a,b;
wire signed [63:0]add_out,sub_out,and_out,xor_out;


sixty_four_bit_ALU alu(control,a,b,add_out,sub_out,and_out,xor_out,add_Cout,sub_Cout);
```

```
always @(*)
begin


begin
    if(control == 2'b00)
    begin
  ZF =(add_out == 1'b0);
    end
    else if(control == 2'b01)
    begin
  ZF=(sub_out == 1'b0);
    end
    else if(control == 2'b10)
    begin
  ZF=(and_out == 1'b0);
    end
    else if(control == 2'b11)
    begin
  ZF=(xor_out == 1'b0);
    end

if(control == 2'b00)
begin
SF=(add_out[63]==1'b1);
end
else if(control == 2'b01)
begin
SF=(sub_out[63]==1'b1);
end
else if(control == 2'b10)
begin
SF=(and_out[63]==1'b1);
end
else if(control == 2'b11)
begin
SF=(xor_out[63]==1'b1);
end

if(control == 2'b00)
begin
OF=(((a[63] == 1'b1) == (b[63] == 1'b1))&&((add_out[63] == 1'b1) != (a[63] == 1'b1)));
end
else if(control == 2'b01)
begin
OF=(((a[63] == 1'b1) == (b[63] == 1'b1))&&((sub_out[63] == 1'b1) != (a[63] == 1'b1)));
end

end
```

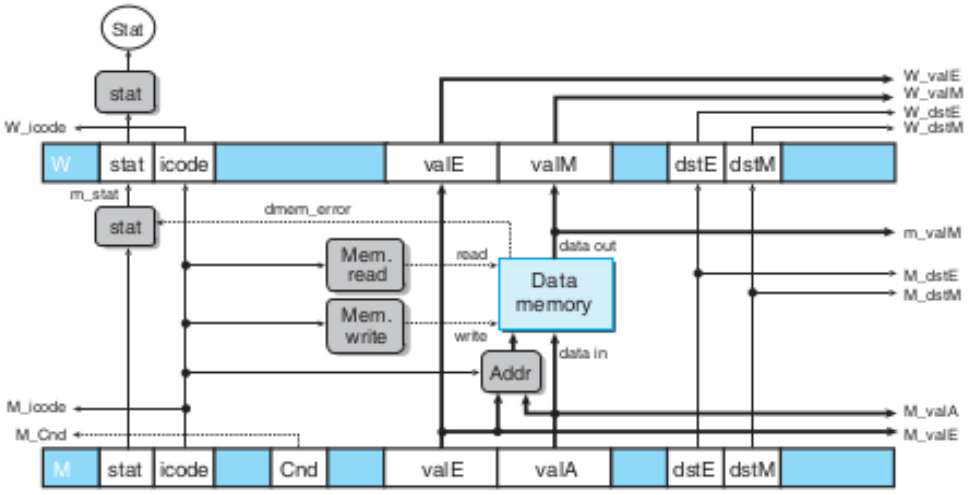
```
    if(E_icode == 4'b0010 || E_icode == 7)
    begin
        if(E_ifun == 4'h0) e_cnd = 1;             // unconditional
        else if(E_ifun == 4'h1) e_cnd = (SF^OF)|ZF; // le
        else if(E_ifun == 4'h2) e_cnd = OF^SF;     // l
        else if(E_ifun == 4'h3) e_cnd = ZF;        // e
        else if(E_ifun == 4'h4) e_cnd = ~ZF;       // ne
        else if(E_ifun == 4'h5) e_cnd = ~(SF^OF); // ge
        else if(E_ifun == 4'h6) e_cnd = ~(SF^OF)&~ZF; // g

    e_dstE = e_cnd ? E_dstE : 4'hF;
    end
    else
    begin
        e_dstE=E_dstE;
    end
end

always @(*)
begin

      if(E_icode == 4'b0011) //irmovq
      begin
           a=1'b0;
           b=E_valC;
              e_valE= b;

      end
      else if(E_icode==4'b0100) //rmmovq
      begin
            a=E_valC;
            b=E_valB;
            control=2'b00;
                e_valE=add_out;

      end
      else if(E_icode == 4'b0101) //mrmovq
      begin
           a=E_valC;
           b=E_valB;
           control=2'b00;
              e_valE= add_out;


      end
```

```
    else if(E_icode == 6) //opq
    begin
        a=E_valA;
        b=E_valB;
        if(E_ifun == 2'b00)
            begin
                control=2'b00;
                    e_valE= add_out;
            end

            else if(E_ifun == 2'b01)
            begin
                control=2'b01;
                    e_valE= sub_out;
            end
            else if(E_ifun == 2'b10)
            begin
                control=2'b10;
                    e_valE= and_out;
            end
            else if(E_ifun == 2'b11)
            begin
                control=2'b11;
                    e_valE=xor_out;

            end

    end

    else if(E_icode==4'b1000) //call
    begin
        a=E_valB;
        b=64'd8;
        control=2'b01;
            e_valE=sub_out;

    end
    else if(E_icode== 9)//ret
    begin
        a=64'd8;
        b=E_valB;
        control=2'b00;
            e_valE= add_out;

    end
    else if(E_icode==4'b1010)//pushq
    begin
        a=E_valB;
        b=64'd8;
        control=2'b01;
            e_valE= sub_out;

    end
```

```
    else if(E_icode==4'hB)//popq
    begin
        a=64'd8;
        b=E_valB;
        control=2'b00;
            e_valE= add_out;

    end
    else if(E_icode == 4'b0010) // cmovxx
    begin
        a=E_valA;
        b=64'd0;
            e_valE= a;

    end

    else
    begin
        a=64'd0;
        b=64'd0;
            e_valE= b;

    end

end

always @(posedge clk)
begin
    begin
        M_stat <= E_stat;
        M_icode <= E_icode;
        M_cnd <= e_cnd;
        M_valE <= e_valE;
        M_valA <= E_valA;
        M_dstE <= e_dstE;
        M_dstM <= E_dstM;
    end
end


endmodule
```

4.Memory Stage:-

Comparing this to the memory stage for SEQ, we see that, as noted before, the block labeled “Data” in SEQ is not present in PIPE. This block served to select between data sources valP (for call instructions) and valA, but this selection is now performed by the block labeled “Sel+Fwd A” in the decode stage.
Here the many of the values in pipeline registers and M and W are supplied to other parts of the circuit as part of the forwarding and pipeline control logic.



Working of execute stage:-

i) This takes the inputs as the outputs from the memory pipelined register which include M_stat, M_icode, M_Cnd, M_valE, M_valA, M_dstE and M_dstM. The outputs obtained from this block include W_stat, W_icode, W_valE, W_valM, W_dstE, W_dstM and m_valM.

ii) This block functions the same way as that of the sequential memory block which takes M_icode, M_Cnd, M_valE, M_valA, M_dstE and M_dstM as inputs and gives W_icode, W_valE, W_valM, W_dstE, W_dstM and m_valM as outputs.

iii) This also updates the m_stat to M_stat if dmem_error has not occurred. The register gets updated once the clk hits and the output for W_stat, W_icode, W_valE, W_valM, W_dstE and W_dstM is available as an output of the memory register.

Memory module:-

```
module memory (clk,M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,W_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,m_valM,m_stat);

input clk;
input [0:3] M_stat;
input [3:0] M_icode;
input M_cnd;
input [63:0] M_valE,M_valA;
input [3:0] M_dstE,M_dstM;

output reg [0:3] W_stat;
output reg [3:0] W_icode;
output reg [63:0] W_valE,W_valM;
output reg [3:0] W_dstE,W_dstM;
output reg [63:0] m_valM;
output reg [0:3] m_stat;

reg [63:0] memory [0:255];
reg memory_error = 0;
reg check_valE,check_valA;

always @(*)
begin
    if(memory_error)
        m_stat = 4'b0010;
    else
        m_stat = M_stat;
end

always @(*)
begin
    if(M_icode==4'b0100 | M_icode==4'b0101 | M_icode== 4'b1000 | M_icode==4'hB) // These are the values of icode where value of valE determines the location in memory
    begin
        check_valE=1;
     end

    else
    begin
        check_valE=0;
    end

    if(M_icode == 4'b1001 | M_icode == 4'hA)
    begin
        check_valA=1;
    end

    else
    begin
        check_valA=0;
    end

end
```

```
always @(*)
begin

    if((M_valE>255 & check_valE) || (M_valA > 255 & check_valA))
    begin
        memory_error=1 ;
    end

    if(M_icode == 4'b0101) //mrmovq
    begin
        m_valM=memory[M_valE];
    end

    if(M_icode == 4'b1001 || M_icode == 4'hB)   //ret   //popq
    begin
        m_valM=memory[M_valA];
    end

end

always @(posedge clk)
begin
    // checking only valE as we are accessing memory with the value of valE in the these cases->
    if(M_valE>255 & check_valE)
        memory_error=1;

    if(M_icode == 4'b0100 || M_icode == 4'b1000 || M_icode == 4'hA)  //rmmovq   //call    // pushq
    begin
        memory[M_valE] <= M_valA;
    end

end

// here i am updating the writeback pipeline register
always @(posedge clk)
begin
    W_stat <= m_stat;
    W_icode <= M_icode;
    W_valE <= M_valE;
    W_valM <= m_valM;
    W_dstE <= M_dstE;
    W_dstM <= M_dstM;
end

endmodule
```

NOTE:- In memory.v, when we have to read memory, we have to access the memory, but since memory can be quite large and is complex to implement due to storage issues, a small change we made here to make it simple is to already assign fixed values to the memory values of the addresses M_valE and M_valE, so that whenever we want to access memory, we use one of these addresses to access it and we get output as these predefined values.

Pipeline control logic:-

```
module pipe_control (m_stat,W_stat,D_icode,E_icode,M_icode,d_srcA,d_srcB,E_dstM,e_cnd,F_stall,D_stall,D_bubble,E_bubble,set_cc);

    input [0:3] m_stat,W_stat;
    input [3:0] D_icode,E_icode,M_icode;
    input [3:0] d_srcA,d_srcB,E_dstM;
    input e_cnd;

    output reg F_stall, D_stall, D_bubble, E_bubble,set_cc;

    always @(*)
    begin
        F_stall=0;
        D_stall=0;
        D_bubble=0;
        E_bubble=0;
        set_cc=1;

        if(E_icode== 4'h7 & e_cnd==0)
        begin
            D_bubble=1;
            E_bubble=1;
        end
        else if ((E_icode == 4'h5 | E_icode==4'hB) & (E_dstM==d_srcA | E_dstM==d_srcB))
        begin
            F_stall=1;
            D_stall=1;
            E_bubble=1;
        end
        else if(E_icode == 4'h9 | M_icode == 4'h9 | D_icode == 4'h9)
        begin
            F_stall = 1;
            D_bubble = 1;
        end

        else if(E_icode == 4'h0 | m_stat!=4'b1000 | W_stat!=4'b1000)
        begin
            set_cc = 0;
        end
        else
        begin
            F_stall=0;
            D_stall=0;
            D_bubble=0;
            E_bubble=0;
            set_cc=1;
        end

    end

endmodule
```

Processor:-

```
`include "fetch.v"
`include "decode.v"
`include "execute.v"
`include "memory.v"
`include "pipe_control.v"

module processor;
    reg clk;
reg [0:3] stat=4'b1000;
    reg [63:0] F_predPC;

    wire [63:0] f_predPC;
    wire [2:0] D_stat,E_stat,M_stat,W_stat,m_stat;
    wire [3:0] D_icode,E_icode,M_icode,W_icode;
    wire [3:0] D_ifun,E_ifun;
    wire [3:0] D_rA,D_rB;
    wire signed [63:0] D_valC,D_valP;
    wire [3:0] d_srcA,d_srcB;
    wire signed [63:0] E_valC,E_valA,E_valB,e_valE;
    wire signed [63:0] M_valE,M_valA,m_valM;
    wire signed [63:0] W_valE,W_valM;
    wire [3:0] E_dstE,E_dstM,E_srcA,E_srcB,e_dstE;
    wire [3:0] M_dstE,M_dstM;
    wire [3:0] W_dstE,W_dstM;
    // wire [63:0] e_valE;
    wire ZF,SF,OF;


 wire signed [63:0] M_valE_out;
 wire signed [63:0] M_valA_out;

    wire M_cnd;
    wire e_cnd;

     wire [63:0] rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14;
     wire F_stall,D_stall,D_bubble,E_bubble,set_cc;


    fetch fetch(D_stat,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,f_predPC,M_icode,M_cnd,M_valA,W_icode,W_valM,F_predPC,clk,F_stall,D_stall,D_bubble);

    decode decode (clk,D_icode,D_ifun,D_rA,D_rB,D_stat,D_valC,D_valP,E_bubble,W_icode,e_dstE,M_dstE,M_dstM,W_dstE,W_dstM,e_valE,M_valE,m_valM,W_valE,W_valM,E_stat,E_icode,
    E_ifun,E_valC,E_valA,E_valB,E_dstE,E_dstM,E_srcA,E_srcB, rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14);

    execute execute(clk,E_icode,E_ifun,E_valC,E_valA,E_valB,E_dstE,E_dstM,E_srcA,E_srcB,E_stat,W_stat,m_stat,M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,e_dstE,e_valE,e_cnd,ZF,SF,OF);

    memory memory (clk,M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,W_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,m_valM,m_stat);

    pipe_control pipe_control (m_stat,W_stat,D_icode,E_icode,M_icode,d_srcA,d_srcB,E_dstM,e_cnd,F_stall,D_stall,D_bubble,E_bubble,set_cc);
```

```
    always@(stat) begin
        if(stat==3'b100)
        begin
            $display("Instruction error");
            $finish;
        end
        else if (stat == 4'b011)
        begin
            $display("Memory_error");
            $finish;
        end
        else if(stat== 4'b010)
        begin
            $display("halting");
            $finish;
        end
    end


    always #10 clk = ~clk;


    always @(posedge clk)
    begin
        F_predPC <= f_predPC;
    end


    initial begin
        F_predPC=64'd0;
        clk=0;

        $monitor(" \n clk = %0d , M_valE =%0d , M_valA = %0d , m_valM = %0d \n.....................................\n\n W_stat=%d\n,f_p

    end


endmodule
```

Testbench:-

i) cmovq.txt:-

```
00110000    //irmovq
11110011    // at F  %rbx i.e 3
00010100    // 20
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000    //irmovq
11111000    // at F  %r8 i.e 8
00001111    // 15
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100000   // opq   for    ADD->0
00111000    // rbx + r8
00100101    // cmovge
10001101
00000000  // halt
```

Output:-

```
clk = 0 , M_valE =x , M_valA = x , m_valM = x
.....................................

W_stat=x
,f_predPC=                10
F_predPC=                  0

.....................................
D_icode= x
,E_icode= x
, M_icode= x
,W_icode= x
.....................................
F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                  x
  D_ifun= x
W_valE=                   x
E_valC=                   x

.....................................
,rax =                    x
, rcx=                    x
, rdx=                    x
, rbx=                    x
, rsp=                    x
, rbp=                    x
, rsi=                    x
, rdi=                    x
,  r8=                    x
,  r9=                    x
, r10=                    x
, r11=                    x
, r12=                    x
, r13=                    x
, r14=                    x


clk = 1 , M_valE =x , M_valA = x , m_valM = x
.....................................

W_stat=x
,f_predPC=                20
F_predPC=                 10

.....................................
D_icode= 3
,E_icode= x
, M_icode= x
,W_icode= x
.....................................
F_stall=0
D_stall=0
D_bubble=0
E_bubble=0
```

```
.....................................
  e_valE=                  x
  D_ifun= 0
 W_valE=                  x
 E_valC=                  x

.....................................
,rax =                    0
, rcx=                    0
, rdx=                    0
, rbx=                    0
, rsp=                 1023
, rbp=                    0
, rsi=                    0
, rdi=                    0
,  r8=                    0
,  r9=                    0
, r10=                    0
, r11=                    0
, r12=                    0
, r13=                    0
, r14=                    0



 clk = 0 , M_valE =x , M_valA = x , m_valM = x
.....................................

 W_stat=x
,f_predPC=                  20
 F_predPC=                  10

.....................................
 D_icode= 3
,E_icode= x
, M_icode= x
,W_icode= x
.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                  x
  D_ifun= 0
 W_valE=                  x
 E_valC=                  x

.....................................
,rax =                    0
, rcx=                    0
, rdx=                    0
, rbx=                    0
, rsp=                 1023
, rbp=                    0
, rsi=                    0
, rdi=                    0
,  r8=                    0
,  r9=                    0
, r10=                    0
, r11=                    0
, r12=                    0
, r13=                    0
, r14=                    0
```

```
 clk = 1 , M_valE =x , M_valA = x , m_valM = x
.....................................

 W_stat=x
,f_predPC=                  22
 F_predPC=                  20

.....................................
 D_icode= 3
,E_icode= 3
, M_icode= x
,W_icode= x
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                 20
  D_ifun= 0
 W_valE=                  x
 E_valC=                 20

.....................................
,rax =                    0
, rcx=                    0
, rdx=                    0
, rbx=                    0
, rsp=                 1023
, rbp=                    0
, rsi=                    0
, rdi=                    0
,  r8=                    0
,  r9=                    0
, r10=                    0
, r11=                    0
, r12=                    0
, r13=                    0
, r14=                    0



 clk = 0 , M_valE =x , M_valA = x , m_valM = x
.....................................

 W_stat=x
,f_predPC=                  22
 F_predPC=                  20

.....................................
 D_icode= 3
,E_icode= 3
, M_icode= x
,W_icode= x
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
```

```
.....................................
  e_valE=                  20
  D_ifun= 0
 W_valE=                    x
 E_valC=                  20

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                     0
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0



 clk = 1 , M_valE =20 , M_valA = x , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  24
 F_predPC=                  22

.....................................
 D_icode= 6
,E_icode= 3
, M_icode= 3
,W_icode= x
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                  15
  D_ifun= 0
 W_valE=                    x
 E_valC=                  15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                     0
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0
```

```
 clk = 0 , M_valE =20 , M_valA = x , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  24
 F_predPC=                  22

.....................................
 D_icode= 6
,E_icode= 3
, M_icode= 3
,W_icode= x
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                  15
  D_ifun= 0
 W_valE=                    x
 E_valC=                  15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                     0
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0



 clk = 1 , M_valE =15 , M_valA = x , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  25
 F_predPC=                  24

.....................................
 D_icode= 2
,E_icode= 6
, M_icode= 3
,W_icode= 3
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
```

```
.....................................
  e_valE=                   35
  D_ifun= 5
 W_valE=                   20
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                     0
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0



 clk = 0 , M_valE =15 , M_valA = x , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  25
 F_predPC=                  24

.....................................
 D_icode= 2
,E_icode= 6
, M_icode= 3
,W_icode= 3
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                   35
  D_ifun= 5
 W_valE=                   20
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                     0
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0
```

```
 clk = 1 , M_valE =35 , M_valA = 20 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  25
 F_predPC=                  25

.....................................
 D_icode= 0
,E_icode= 2
, M_icode= 6
,W_icode= 3
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                   35
  D_ifun= 0
 W_valE=                   15
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                    20
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                     0
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0



 clk = 0 , M_valE =35 , M_valA = 20 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  25
 F_predPC=                  25

.....................................
 D_icode= 0
,E_icode= 2
, M_icode= 6
,W_icode= 3
,....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                   35
  D_ifun= 0
 W_valE=                   15
 E_valC=                   15
```

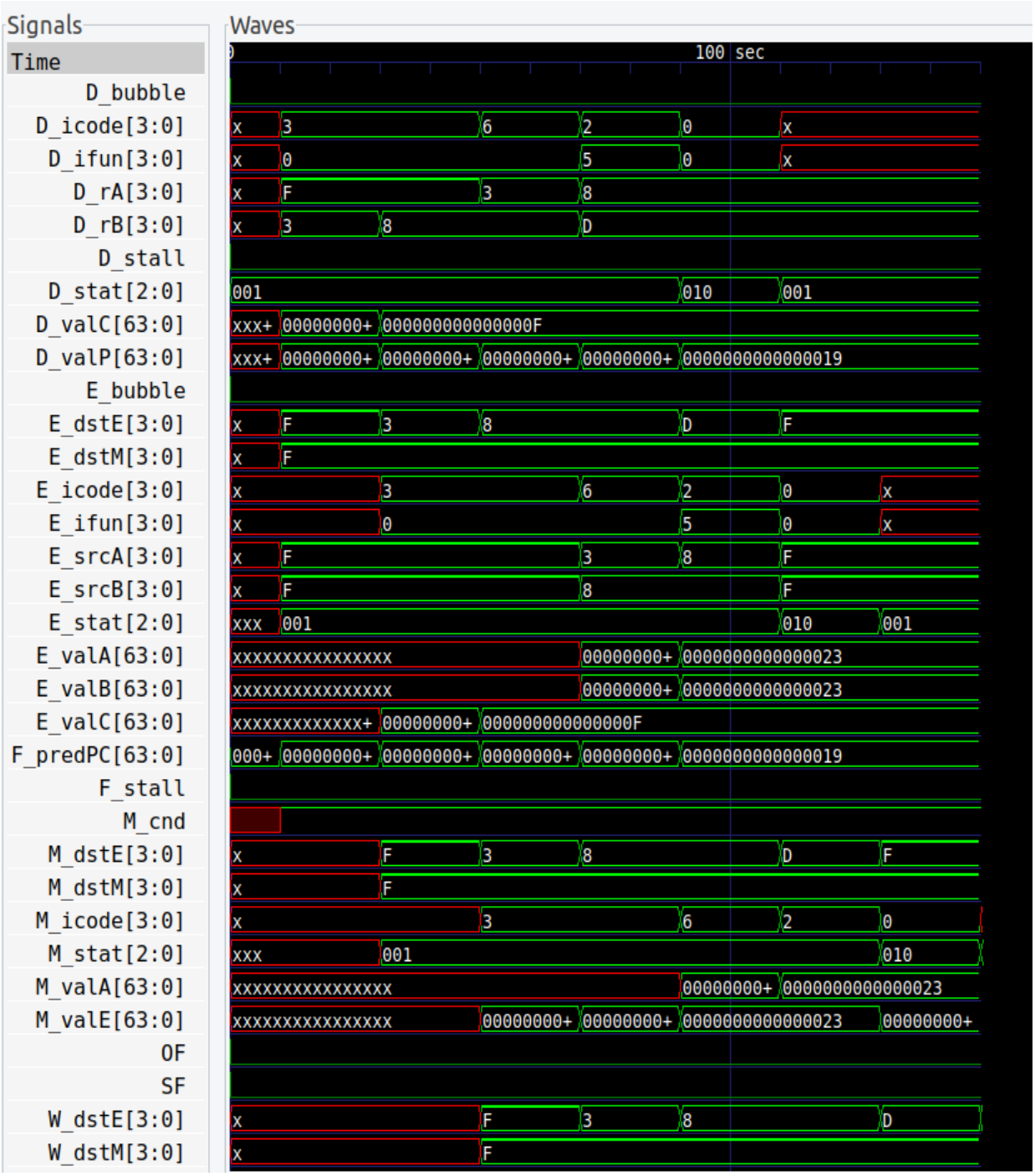
```
.....................................
,rax =                  0
, rcx=                  0
, rdx=                  0
, rbx=                 20
, rsp=               1023
, rbp=                  0
, rsi=                  0
, rdi=                  0
,  r8=                  0
,  r9=                  0
, r10=                  0
, r11=                  0
, r12=                  0
, r13=                  0
, r14=                  0



 clk = 1 , M_valE =35 , M_valA = 35 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                25
 F_predPC=                25

.................................
 D_icode= x
,E_icode= 0
, M_icode= 2
,W_icode= 6
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.................................
  e_valE=                 0
  D_ifun= x
 W_valE=                35
 E_valC=                15

.....................................
,rax =                  0
, rcx=                  0
, rdx=                  0
, rbx=                 20
, rsp=               1023
, rbp=                  0
, rsi=                  0
, rdi=                  0
,  r8=                 15
,  r9=                  0
, r10=                  0
, r11=                  0
, r12=                  0
, r13=                  0
, r14=                  0
```

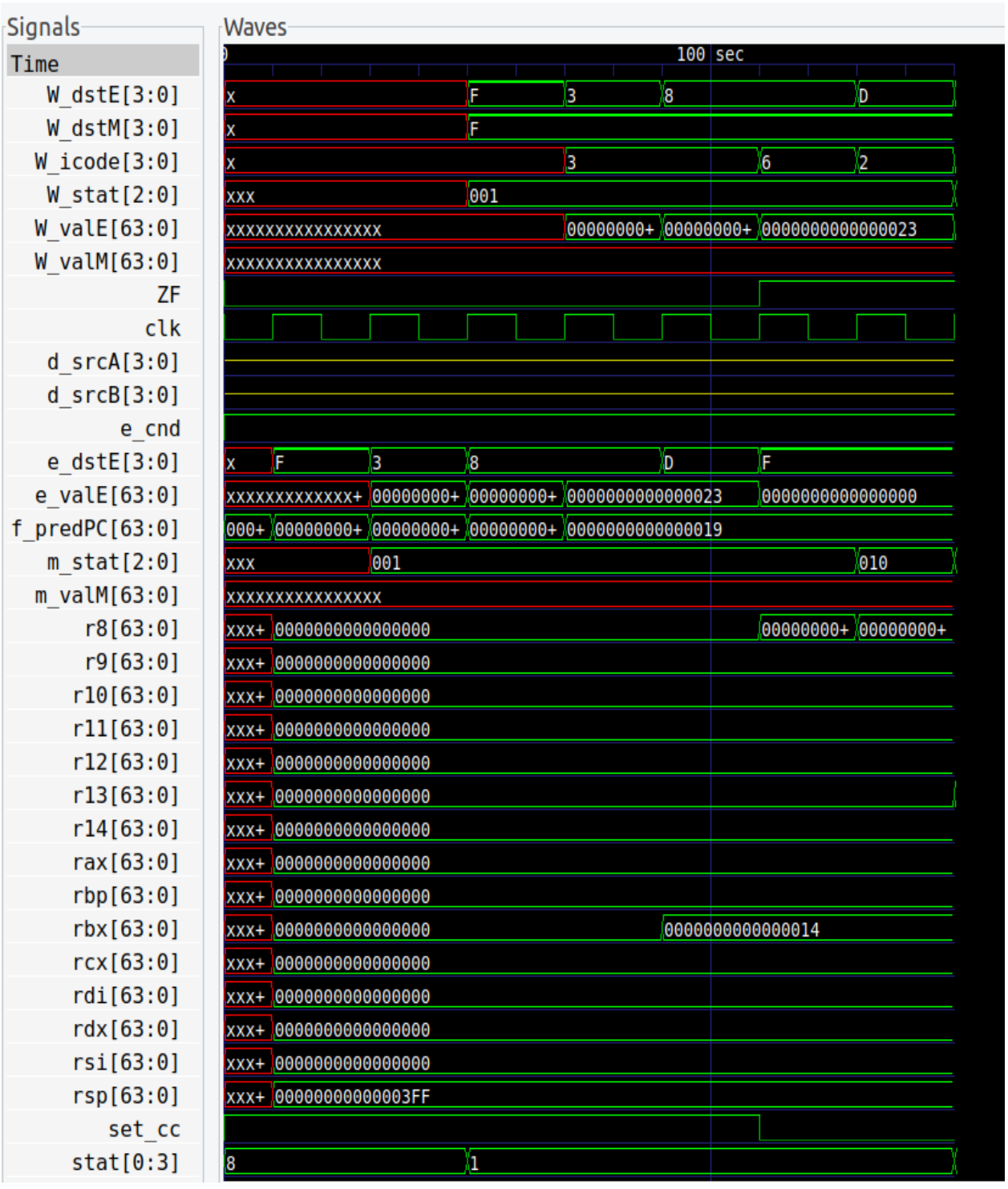
```
 clk = 0 , M_valE =35 , M_valA = 35 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                25
 F_predPC=                25

.................................
 D_icode= x
,E_icode= 0
, M_icode= 2
,W_icode= 6
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.................................
  e_valE=                 0
  D_ifun= x
 W_valE=                35
 E_valC=                15

.....................................
,rax =                  0
, rcx=                  0
, rdx=                  0
, rbx=                 20
, rsp=               1023
, rbp=                  0
, rsi=                  0
, rdi=                  0
,  r8=                 15
,  r9=                  0
, r10=                  0
, r11=                  0
, r12=                  0
, r13=                  0
, r14=                  0



 clk = 1 , M_valE =0 , M_valA = 35 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                25
 F_predPC=                25

.................................
 D_icode= x
,E_icode= x
, M_icode= 0
,W_icode= 2
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0
```

```
.....................................
  e_valE=                    0
  D_ifun= x
 W_valE=                   35
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                    20
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                    35
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0



 clk = 0 , M_valE =0 , M_valA = 35 , m_valM = x
.....................................

 W_stat=1
,f_predPC=                  25
 F_predPC=                  25

.....................................
 D_icode= x
,E_icode= x
, M_icode= 0
,W_icode= 2
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                    0
  D_ifun= x
 W_valE=                   35
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                    20
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                    35
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                     0
, r14=                     0
```

```
halting

 clk = 1 , M_valE =0 , M_valA = 35 , m_valM = x
.....................................

 W_stat=2
,f_predPC=                  25
 F_predPC=                  25

.....................................
 D_icode= x
,E_icode= x
, M_icode= x
,W_icode= 0
,.....................................
 F_stall=0
D_stall=0
D_bubble=0
E_bubble=0

.....................................
  e_valE=                    0
  D_ifun= x
 W_valE=                    0
 E_valC=                   15

.....................................
,rax =                     0
, rcx=                     0
, rdx=                     0
, rbx=                    20
, rsp=                  1023
, rbp=                     0
, rsi=                     0
, rdi=                     0
,  r8=                    35
,  r9=                     0
, r10=                     0
, r11=                     0
, r12=                     0
, r13=                    35
, r14=                     0
```

GTKwave Plot:-




Signals
Waves
Time
W_dstE[3:0]
W_dstM[3:0]
W_icode[3:0]
W_stat[2:0]
W_valE[63:0]
W_valM[63:0]
ZF
clk
d_srcA[3:0]
d_srcB[3:0]
e_cnd
e_dstE[3:0]
e_valE[63:0]
f_predPC[63:0]
m_stat[2:0]
m_valM[63:0]
r8[63:0]
r9[63:0]
r10[63:0]
r11[63:0]
r12[63:0]
r13[63:0]
r14[63:0]
rax[63:0]
rbp[63:0]
rbx[63:0]
rcx[63:0]
rdi[63:0]
rdx[63:0]
rsi[63:0]
rsp[63:0]
set_cc
stat[0:3]
100 sec

Call and return:-

As mentiones in project to write an assembly program for any algorithm using Y86 ISA and the corresponding encoded instructions and use the encoded instructions to test your integrated design.

ASSEMBLY CODE:-

```
  0x000:    irmovq Stack,%rsp  # Intialize stack pointer
0x00a:    nop                # Avoid hazard on %rsp
0x00b:    nop
0x00c:    nop
0x00d:    call p             # Procedure call
0x016:    irmovq $5,%rsi     # Return point
0x020:    halt
0x020: .pos 0x20
0x020: p: nop                 # procedure
0x021:    nop
0x022:    nop
0x023:    ret
0x024:    irmovq $1,%rax      # Should not be executed
0x02e:    irmovq $2,%rcx      # Should not be executed
0x038:    irmovq $3,%rdx      # Should not be executed
0x042:    irmovq $4,%rbx      # Should not be executed
0x100: .pos 0x100
0x100: Stack:                 # Initial stack pointer
```

THANK YOU!