EC2.204 Introduction to Processor Architecture

Y86-64 Processor

Team:

Meghana Tedla 2021102002

Anushka Agrawal 2021102023

Overview

The main objective of this project is to implement a Y86-64 processor. The processor should be able to execute all the instructions in the Y86-64 Instruction set architecture. The aim of this project is to implement a sequential and 5-stage pipelined U86-64 processor.

Instructions

The y86-64 instruction set architecture is a simplified version of the x86-64 architecture used by modern intel and AMD processors.

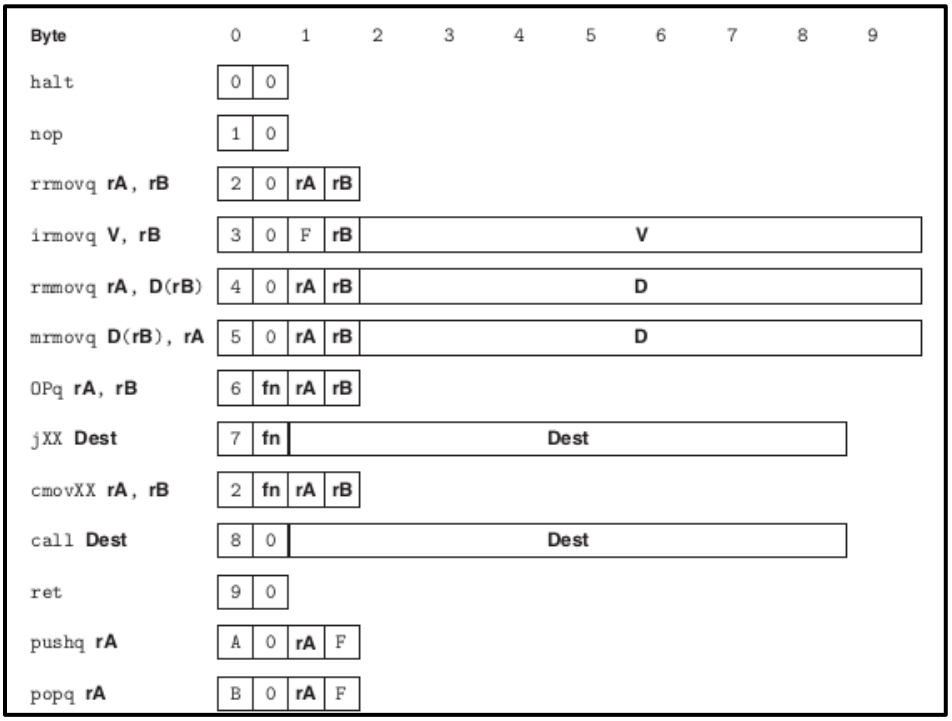
The y86-64 instruction set contains instructions for arithmetic and logical operations, comparison instructions, data movement instructions, stack instructions, jump instructions, and special instructions. The arithmetic and logic instructions include addition, subtraction, and bitwise operations like 'AND' and 'XOR'.

- The x86-64 movq instruction is split into four different instructions: irmovq, rrmovq, mrmovq, and rmmovq, explicitly indicating the form of the source and destination.
- There are four integer operation instructions (Opq), these are addq, subq, andq, and xorq.
- The seven jump instructions are jmp, jle, jl, je, jne, jge, and jg.
- There are six conditional move instructions cmovle, cmovl, cmove, cmovne, cmovge, and cmovg.
- The call instruction pushes the return address on the stack and jumps to the destination address.
- The pushq and popq instructions implement push and pop, just as they do in x86-64.
- The halt instruction stops instruction execution.

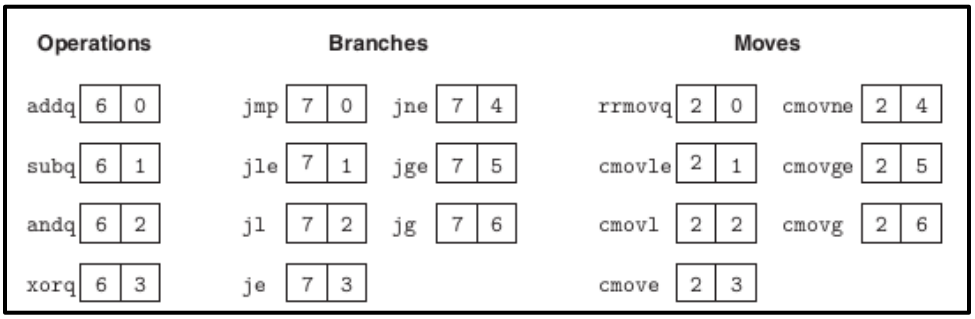
Instruction Encodings

The y86-64 instruction encoding range from of 1 and 10 bytes. An instruction consists of a 1-byte instruction specifier, possibly a 1-byte register specifier, and possibly an 8-byte constant word.

- The instruction specifier byte (initial byte) is split into two 4-bit parts: the high-order, or code part called *icode* , and the low-order, or function part called *ifun*. The function values are significant only for the cases where a group of related instructions share a common code.
- There can be an additional register specifier byte (if required), specifying either one or two registers. These register fields are called *rA* and *rB*.
- If the instruction requires a constant value (be it a memory address or a number) it is contained in the next 8 bytes.

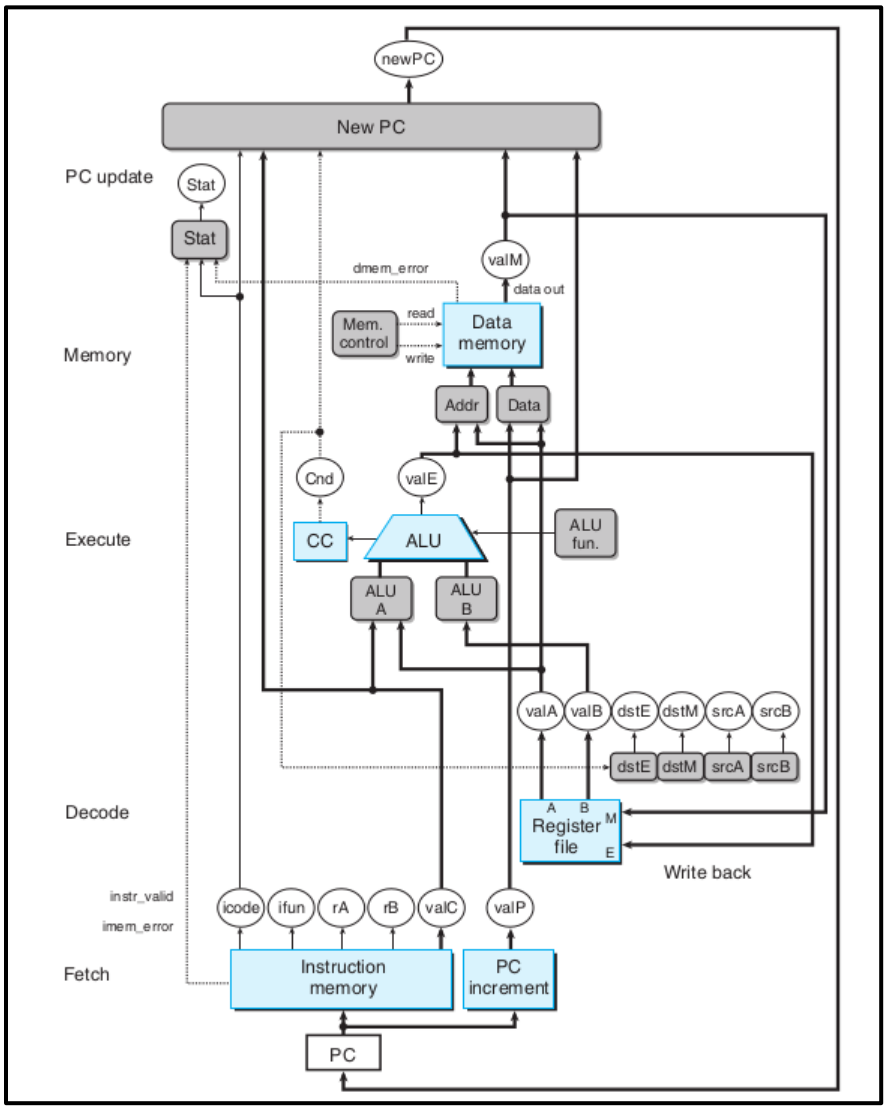


Y86-64 instruction set



Function codes for Y86-4 instruction set.

SEQ: Sequential Y86-64 Implementation



Hardware structure of SEQ, a sequential implementation

A sequential processor is a type of computer architecture that executes instructions sequentially, one after another.

The execution of a single instruction can be broken down into 6 stages:

- Fetch
- Decode
- Execute
- Memory
- Write Back
- PC Update

In a sequential processor, instructions are stored in memory, and the processor fetches instructions one by one from memory, decodes them, and executes them in order. The processor can perform only one instruction at a time, and instructions cannot overlap in their execution.

FETCH

The fetch stage is the first stage of the instruction execution cycle.it is responsible for fetching the next instruction from memory and storing it in the instruction register (IR). The length of the instruction fetched is 10 bytes long.

The instruction is fed into split and align.

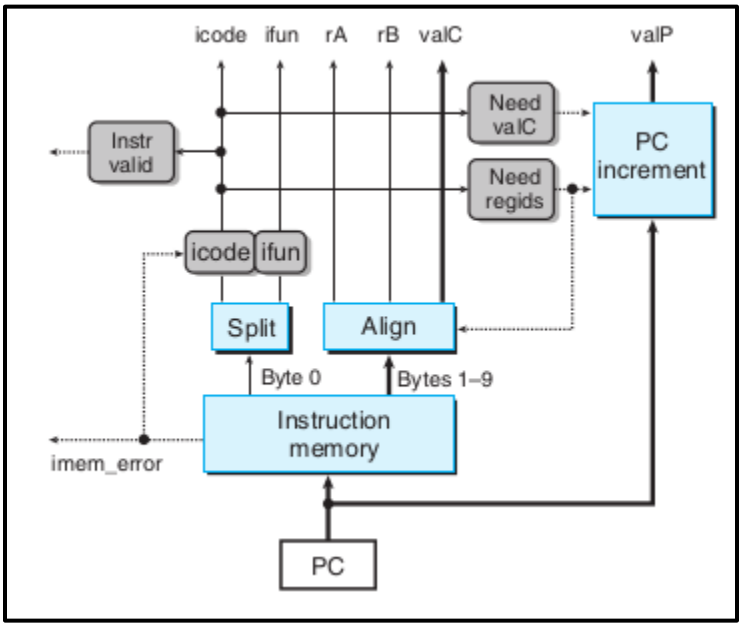
- Spilt contains the icode and ifun.
- Align contains the rest of the information. It can include rA, rB and valC.

From split, we can determine whether to read from align for rA and rB and valC. Internally,

1) need_regids tells whether rA and rB are needed.
2) need_valC tells whether valC needs to be read.

Depending on the instruction, the value of valP (the possible PC increment) is updated.

We set imem_error and instr_valid signals to 0 or 1 based on whether the memory address or instruction address is valid. The signals instr_valid and imem_error are used to generate the status code in the memory stage.



Fetch block

Implementation and Working:

- On the positive edge of clock, the fetch stage fetches the instruction from the instruction memory based on the updated PC value. We have declared the instruction memory as a register array of 1024 bytes. This way of declaration makes the fetching of an instruction easy.

```
reg [7:0] memory[0:1023]; //memory has 256-32 bits words => 1024-8 bits
```

- If PC is greater than 1023 then it gives rise to an 'imem_error'.
- If PC is valid then 10 consecutive bytes are read as 'instruction' including the PC byte. This is because the maximum length of an instruction is 10 bytes.
- From the instruction register we found the icode and ifun values which corresponds to the first 4 bits and last 4 bits of the first byte respectively.
- If the icode is not a valid code for an instruction, then it sets the 'instr_valid' to 1.
- Depending on the value of icode, the other output values are obtained from the instruction register as follows:
  - rA is instruction[8:11]
  - rB is instruction[12:15]
  - valC is instruction[8:71] when icode = IJXX and ICALL
  - valC is instruction[16:79] when icode is in {IIRMOVQ, IRMMOVQ, IMRMOVQ, IOPQ}
  - valP = PC + 64'd1 for icode equal IHALT, INOP or IRET
  - valP = PC + 64'd2 for icode equal to IRRMOVQ, IOPQ, IPUSHQ or IPOPQ
  - valP = PC + 64'd9 for icode equal to IJXX or ICALL
  - valP = PC +64'd10 for all other instructions

Source code

```
// FETCH BLOCK

module fetch(clk, PC, icode, ifun, rA, rB, valC, valP, imem_error, instr_valid, hlt);

    input clk;
    input [63:0] PC;

    output reg [3:0] icode;
    output reg [3:0] ifun;

    output reg [3:0] rA;
    output reg [3:0] rB;

    output reg [63:0] valC;
    output reg [63:0] valP;

    output reg instr_valid;
    output reg imem_error;
    output reg hlt;

    reg [0:79] instruction; //Instruction encodings range between 1 and 10 bytes


    reg [7:0] memory[0:1023]; //memory has 256-32 bits words => 1024-8 bits



    reg [0:7] opcode; //operation codes
    reg [0:7] regids; //register IDs

    initial
    begin
        rA = 4'hf;
        rB = 4'hf;
        valC = 64'd0;
        valP = 64'd0;
        hlt = 0;
        instr_valid = 0;
        imem_error = 0;
    end
```

```
    // instruction codes
    // Constant values used in HCL descriptions.
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;

    initial
    begin

    // irmovq $0x0, %rax
    memory[0]=8'b00110000; //3 0
    memory[1]=8'b00000000; //F rB=0
    memory[2]=8'b00000000;
    memory[3]=8'b00000000;
    memory[4]=8'b00000000;
    memory[5]=8'b00000000;
    memory[6]=8'b00000000;
    memory[7]=8'b00000000;
    memory[8]=8'b00000000;
    memory[9]=8'b00000000; //V=0

    // irmovq $0x10, %rdx
    memory[10]=8'b00110000; //3 0
    memory[11]=8'b00000010; //F rB=2
    memory[12]=8'b00000000;
    memory[13]=8'b00000000;
    memory[14]=8'b00000000;
    memory[15]=8'b00000000;
    memory[16]=8'b00000000;
    memory[17]=8'b00000000;
    memory[18]=8'b00000000;
    memory[19]=8'b00010000; //V=16
```

```
always@(posedge clk)
begin

    if(PC > 64'd1023)
    begin

        imem_error = 1; //invalid address
    end
    else
    begin

        imem_error = 0;

        //Instruction encodings range between 1 and 10 bytes
        instruction = {memory[PC], memory[PC+64'd1], memory[PC+64'd2], memory[PC+64'd3],
                       memory[PC+64'd4], memory[PC+64'd5], memory[PC+64'd6], memory[PC+64'd7], memory[PC+64'd8], memory[PC+64'd9]};

        //An instruction consists of a 1-byte instruction specifier, 1-byte register specifier, and an 8-byte constant word.

        // icode:ifun = M1[PC]
        opcode = instruction[0:7];
        icode = opcode[0:3];
        ifun = opcode[4:7];


        if(icode < 4'd0 || icode > 4'd11)
        begin

            instr_valid = 1; //invalid instruction
        end
        else
        begin

            instr_valid = 0;

            //halt
            if(icode == IHALT)
            begin

                hlt = 1;

                //valP = PC+1
                valP = PC + 64'd1;

            end
```

```
//nop
else if (icode == INOP)
begin

    //valP = PC+1
    valP = PC + 64'd1;

end
//rrmovq and  cmovXX
else if (icode == IRRMOVQ)
begin

    // rA:rB = M1[PC+1]


    regids = instruction[8:15];
    rA = regids[0:3];
    rB = regids[4:7];

    //valP <- PC+2
    valP = PC + 64'd2;

end
//irmovq
else if (icode == IIRMOVQ)
begin

    // rA:rB = M1[PC+1]
    regids = instruction[8:15];
    rA = regids[0:3];
    rB = regids[4:7];

    //valC <- M8[PC+2]
    valC = instruction[16:79];

    //valP <- PC+10
    valP = PC + 64'd10;

end
// rmmovq
else if (icode == IRMMOVQ)
begin

    // rA:rB = M1[PC+1]
    regids = instruction[8:15];
    rA = regids[0:3];
    rB = regids[4:7];

    //valC <- M8[PC+2]
    valC = instruction[16:79];

    //valP <- PC+10
    valP = PC + 64'd10;

end
// mrmovq
else if (icode == IMRMOVQ)
begin

    // rA:rB = M1[PC+1]
    regids = instruction[8:15];
    rA = regids[0:3];
    rB = regids[4:7];

    //valC <- M8[PC+2]
    valC = instruction[16:79];

    //valP <- PC+10
    valP = PC + 64'd10;

end
//Opq
else if (icode == IOPQ)
begin

    // rA:rB = M1[PC+1]
    regids = instruction[8:15];
    rA = regids[0:3];
    rB = regids[4:7];

    //valP <- PC+2
    valP = PC + 64'd2;

end
```

```
            //jXX
            else if (icode == IJXX)
            begin

                //valC <- M8[PC+1]
                valC = instruction[8:71];

                //valP <- PC+9
                valP = PC + 64'd9;

            end
            //call
            else if (icode == ICALL)
            begin

                //valC <- M8[PC+1]
                valC = instruction[8:71];

                //valP <- PC+9
                valP = PC + 64'd9;

            end
            //ret
            else if (icode == IRET)
            begin

                //valP <- PC+1
                valP = PC + 64'd1;

            end
            //pushq
            else if (icode == IPUSHQ)
            begin

                // rA:rB = M1[PC+1]
                regids = instruction[8:15];
                rA = regids[0:3];
                rB = regids[4:7];

                //valP <- PC+2
                valP = PC + 64'd2;

            end
                //popq
                else if (icode == IPOPQ)
                begin

                    // rA:rB = M1[PC+1]
                    regids = instruction[8:15];
                    rA = regids[0:3];
                    rB = regids[4:7];

                    //valP <- PC+2
                    valP = PC + 64'd2;

                end

            end

        end

    endmodule
```

The instruction given to the fetch module is:

irmovq $0x0, %rax

irmovq $0x10, %rdx

rmovq $0xc, %rbx

jmp check

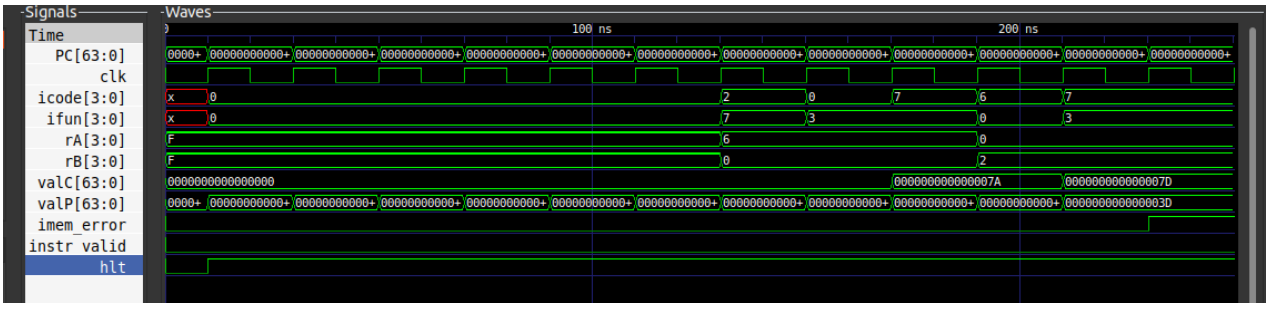
check:

```
addq %rax, %rbx

je rbxres

halt
```

Output of test bench

```
10    clk = 1 PC = 32
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 33 imem_error = 0, instr_valid = 0, hlt = 1

20    clk = 0 PC = 32
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 33 imem_error = 0, instr_valid = 0, hlt = 1

30    clk = 1 PC = 33
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 34 imem_error = 0, instr_valid = 0, hlt = 1

40    clk = 0 PC = 33
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 34 imem_error = 0, instr_valid = 0, hlt = 1

50    clk = 1 PC = 34
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 35 imem_error = 0, instr_valid = 0, hlt = 1

60    clk = 0 PC = 34
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 35 imem_error = 0, instr_valid = 0, hlt = 1

70    clk = 1 PC = 35
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 36 imem_error = 0, instr_valid = 0, hlt = 1

80    clk = 0 PC = 35
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 36 imem_error = 0, instr_valid = 0, hlt = 1

90    clk = 1 PC = 36
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 37 imem_error = 0, instr_valid = 0, hlt = 1

100   clk = 0 PC = 36
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 37 imem_error = 0, instr_valid = 0, hlt = 1

110   clk = 1 PC = 37
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 38 imem_error = 0, instr_valid = 0, hlt = 1

120   clk = 0 PC = 37
      icode = 0000 ifun = 0000 rA = 1111 rB = 1111 valC = 0 valP = 38 imem_error = 0, instr_valid = 0, hlt = 1

130   clk = 1 PC = 38
      icode = 0010 ifun = 0111 rA = 0110 rB = 0000 valC = 0 valP = 40 imem_error = 0, instr_valid = 0, hlt = 1

140   clk = 0 PC = 38
      icode = 0010 ifun = 0111 rA = 0110 rB = 0000 valC = 0 valP = 40 imem_error = 0, instr_valid = 0, hlt = 1

150   clk = 1 PC = 40
      icode = 0000 ifun = 0011 rA = 0110 rB = 0000 valC = 0 valP = 41 imem_error = 0, instr_valid = 0, hlt = 1

160   clk = 0 PC = 40
      icode = 0000 ifun = 0011 rA = 0110 rB = 0000 valC = 0 valP = 41 imem_error = 0, instr_valid = 0, hlt = 1

170   clk = 1 PC = 41
      icode = 0111 ifun = 0011 rA = 0110 rB = 0000 valC = 122 valP = 50 imem_error = 0, instr_valid = 0, hlt = 1

180   clk = 0 PC = 41
      icode = 0111 ifun = 0011 rA = 0110 rB = 0000 valC = 122 valP = 50 imem_error = 0, instr_valid = 0, hlt = 1

190   clk = 1 PC = 50
      icode = 0000 ifun = 0000 rA = 0110 rB = 0000 valC = 122 valP = 51 imem_error = 0, instr_valid = 0, hlt = 1
```

Gtkwave output:



Command to run the code:

iverilog –o fetch fetch_tb.v fetch.v

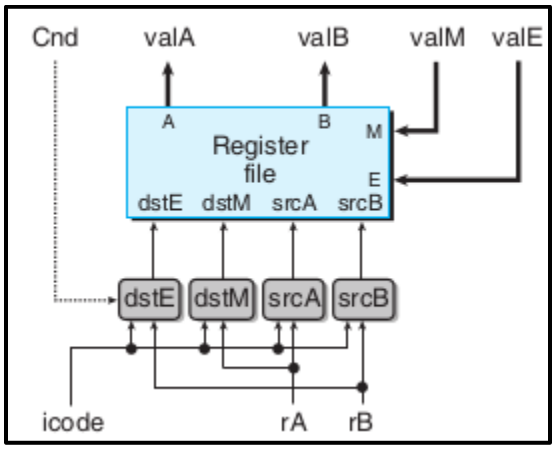
vvp fetch

DECODE

In a sequential processor, the decode stage is the second stage of the instruction execution cycle. It is responsible for decoding the instruction fetched in the previous fetch stage and preparing the processor for the execution stage.

In the decode stage, the values of valA and valB are read from the registers with addresses rA, rB, or %rsp, depending on the icode.

The decode and write back stages can be combined as they both access the register file.



Implementation and Working:

- We have implemented the register memory in a text file which contains the values stored in the registers. This way of implementation helps us access the values stored in the registers in both the decode and write back stage. The values in the text file are read into a register array in the decode stage to obtain valA and valB.

```
// 0x00000000
0000000000000000
0000000000000001
0000000000000010
000000000000000c
0000000000000004
0000000000000005
0000000000000006
0000000000000007
0000000000000008
0000000000000009
000000000000000a
000000000000000b
000000000000000c
000000000000000d
000000000000000e
```

Text file

```
reg [63:0] Reg_File[0:14];

always@(posedge clk)
begin

    $readmemh("reg_file.txt", Reg_File);

end
```

Declaration of the register array and reading the text file

- The values of valA and valB are present in the registers given by the address rA and rB. We use rA and rB and index and reference the 'Reg_File' register array to obtain valA and valB.
- The icode of the instruction decides the values of valA and valB as follows:
  - valA = Reg_File[rA] for icode in {IRRMOVQ, IRMOVQ, IOPQ, IPUSHQ}
  - valA = Reg_File[RRSP] for icode in {IPOPQ, IRET}
  - valB = Reg_File[rB] for icode in {IRRMOVQ, IRMMOVQ, IMROVQ, IOPQ}
  - valB = Reg_File[RRSP] for icode in {ICALL, IPUSHQ, IPOPQ, IRET}

Here RRSP represents the index to the %rsp register.

Source code:

```
// DECODE BLOCK

module decode(clk, icode, rA, rB, valA, valB);

    input clk;
    input [3:0] icode;
    input [3:0] rA;
    input [3:0] rB;

    output reg [63:0] valA;
    output reg [63:0] valB;

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

    integer i;

    initial
    begin
        valA = 64'd0;
        valB = 64'd0;
    end

    // instruction codes
    // Constant values used in HCL descriptions.
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
```

```
// registers
parameter Register_rax = 4'd0;
parameter Register_rcx = 4'd1;
parameter Register_rdx = 4'd2;
parameter Register_rbx = 4'd3;
parameter RRSP         = 4'd4;
parameter Register_rbp = 4'd5;
parameter Register_rsi = 4'd6;
parameter Register_rdi = 4'd7;
parameter Register_r8  = 4'd8;
parameter Register_r9  = 4'd9;
parameter Register_r10 = 4'd10;
parameter Register_r11 = 4'd11;
parameter Register_r12 = 4'd12;
parameter Register_r13 = 4'd13;
parameter Register_r14 = 4'd14;
parameter RNONE        = 4'd15;


always@(posedge clk)
begin

    $readmemh("reg_file.txt", Reg_File);

end

always@(*)
begin

    if(clk == 1)
    begin

        if (icode == IRRMOVQ)
        begin

            // valA ← R[rA]
            valA = Reg_File[rA];

        end
```

```
        // rmmovq
        else if (icode == IRMMOVQ)
        begin

            // valA ← R[rA]
            valA = Reg_File[rA];

            // valB ← R[rB]
            valB = Reg_File[rB];

        end
        // mrmovq
        else if (icode == IMRMOVQ)
        begin

            // valB ← R[rB]
            valB = Reg_File[rB];

        end
        //Opq
        else if (icode == IOPQ)
        begin

            // valA ← R[rA]
            valA = Reg_File[rA];

            // valB ← R[rB]
            valB = Reg_File[rB];

        end
        else if (icode == ICALL)
        begin

            //valB ← R[ %rsp ]
            valB = Reg_File[RRSP];

        end
        //ret
        else if (icode == IRET)
        begin

            // valA ← R[ %rsp ]
            valA = Reg_File[RRSP];

            // valB ← R[ %rsp ]
            valB = Reg_File[RRSP];

        end
```

```
            //pushq
            else if (icode == IPUSHQ)
            begin

                // valA ← R[rA]
                valA = Reg_File[rA];

                // valB ← R[ %rsp ]
                valB = Reg_File[RRSP];

            end
            //popq
            else if (icode == IPOPQ)
            begin

                // valA ← R[ %rsp ]
                valA = Reg_File[RRSP];

                // valB ← R[ %rsp ]
                valB = Reg_File[RRSP];

            end
        end
    end

endmodule
```

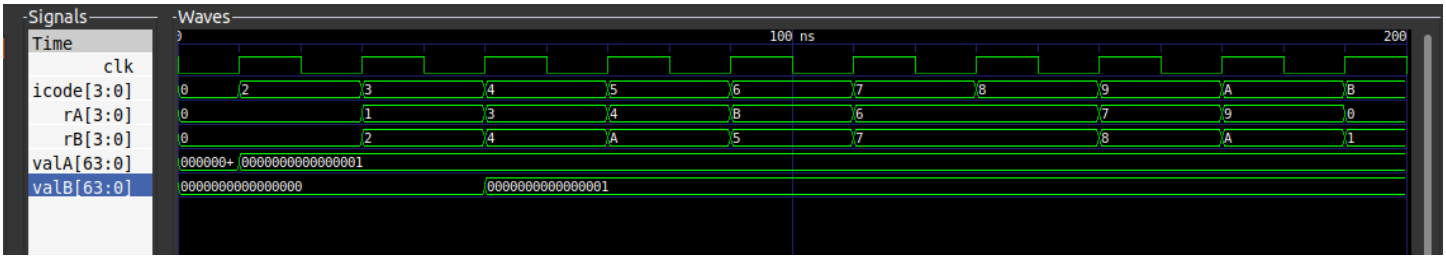
Test Bench:

```
`timescale 1ns / 10ps

module decode_tb();

    reg clk;
    reg [3:0] icode;
    reg [3:0] rA;
    reg [3:0] rB;

    wire [63:0] valA;
    wire [63:0] valB;

    decode UUT(clk, icode, rA, rB, valA, valB);

    //creating clk
    initial
    begin
        clk = 0;
        repeat (20)  #10 clk = ~clk;
    end

    initial
    begin

        $dumpfile("decode_tb.vcd");
        $dumpvars(0, decode_tb);

        clk=0;
        icode=4'd0;
        rA=4'd0;
        rB=4'd0;

    end
    initial
    begin

        #10
        icode=4'd2;rA=4'd0;rB=4'd0;

        #20
        icode=4'd3;rA=4'd1;rB=4'd2;

        #20
        icode=4'd4;rA=4'd3;rB=4'd4;

        #20
        icode=4'd5;rA=4'd4;rB=4'd10;

        #20
        icode=4'd6;rA=4'd11;rB=4'd5;

        #20
        icode=4'd7;rA=4'd6;rB=4'd7;

        #20
        icode=4'd8;rA=4'd6;rB=4'd7;

        #20
        icode=4'd9;rA=4'd7;rB=4'd8;

        #20
        icode=4'd10;rA=4'd9;rB=4'd10;

        #20
        icode=4'd11;rA=4'd0;rB=4'd1;
    end

    initial
    begin
        $monitor($time, "\tclk = %d icode = %b rA = %b rB = %b\n\t\t\tvalA = %g valB = %g\n", clk,icode, rA, rB, valA, valB);
    end

endmodule
```

Output of Test Bench:

```
 10    clk = 1 icode = 0010 rA = 0000 rB = 0000
       valA = 0 valB = 0

 20    clk = 0 icode = 0010 rA = 0000 rB = 0000
       valA = 0 valB = 0

 30    clk = 1 icode = 0011 rA = 0001 rB = 0010
       valA = 0 valB = 0

 40    clk = 0 icode = 0011 rA = 0001 rB = 0010
       valA = 0 valB = 0

 50    clk = 1 icode = 0100 rA = 0011 rB = 0100
       valA = 12 valB = 4

 60    clk = 0 icode = 0100 rA = 0011 rB = 0100
       valA = 12 valB = 4

 70    clk = 1 icode = 0101 rA = 0100 rB = 1010
       valA = 12 valB = 10

 80    clk = 0 icode = 0101 rA = 0100 rB = 1010
       valA = 12 valB = 10

 90    clk = 1 icode = 0110 rA = 1011 rB = 0101
       valA = 11 valB = 5

100    clk = 0 icode = 0110 rA = 1011 rB = 0101
       valA = 11 valB = 5

110    clk = 1 icode = 0111 rA = 0110 rB = 0111
       valA = 11 valB = 5

120    clk = 0 icode = 0111 rA = 0110 rB = 0111
       valA = 11 valB = 5

130    clk = 1 icode = 1000 rA = 0110 rB = 0111
       valA = 11 valB = 4

140    clk = 0 icode = 1000 rA = 0110 rB = 0111
       valA = 11 valB = 4

150    clk = 1 icode = 1001 rA = 0111 rB = 1000
       valA = 4 valB = 4

160    clk = 0 icode = 1001 rA = 0111 rB = 1000
       valA = 4 valB = 4

170    clk = 1 icode = 1010 rA = 1001 rB = 1010
       valA = 9 valB = 4

180    clk = 0 icode = 1010 rA = 1001 rB = 1010
       valA = 9 valB = 4

190    clk = 1 icode = 1011 rA = 0000 rB = 0001
       valA = 4 valB = 4
```

Gtkwave output:



Command to run the code:

iverilog –o decode decode_tb.v decode.v

vvp decode

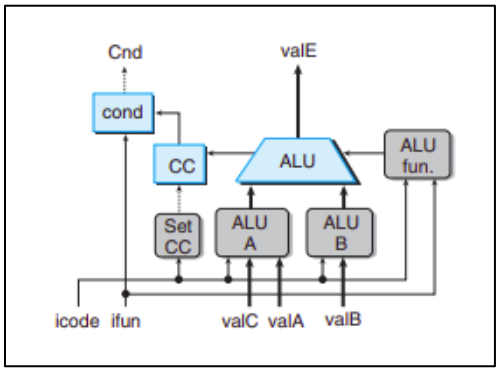
EXECUTE

The execute stage includes the arithmetic/logic unit (ALU). In this stage, the operations ADD, SUBTRACT, AND or XOR are performed on aluA and aluB, depending on the values of icode and ifun.

ifun sets the value of alufun

Condition codes are set during this set depending on the instruction. The output of this stage is valE and cnd. The value of cnd is assigned depending on the values of ifun, ZF, OF and SF as shown below:

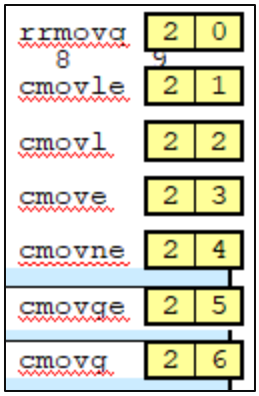
```
if(ifun == 4'd0) // unconditional
begin
    cnd = 1;
end
else if(ifun == 4'd1) // less than equal to
begin
    cnd = ((sf^of)| zf);
end
else if(ifun == 4'd2) // less than
begin
    cnd = (sf^of);
end
else if(ifun == 4'd3) // equal to
begin
    cnd = zf;
end
else if(ifun == 4'd4) // not equal to
begin
    cnd = ~zf;
end
else if(ifun == 4'd5) // greater than equal to
begin
    cnd = ~(sf^of);
end
else if(ifun == 4'd6) // greater than
begin
    cnd = ((~(sf^of))&~zf);
end
```



Execute Block

Implementation and Working:

- When the clock is high, we perform the following operations according to icode of the instruction
    - For ‘cmov’ instruction we set the cnd output according to the condition codes and ifun as specified above. We also set valE = valA.



- For ‘irmovq’ instruction we set the output valE = valC
- For ‘mrmovq’ and ‘rmmovq’ instructions we set the output valE = valB + valC.
- For ‘opx’ instruction we check ifun code and perform the arthmetic and logical function accordingly. We set valE = ans which is the output from the alu. We also set the condition codes in this instruction.

- For ‘jXX’ instruction we again set the cnd bit according to the condition codes and ifun as specified above.

- For ‘call’ and ‘pushq’ we set the output valE = -64’d1+valB.
- For ‘ret’ and ‘popq’ instructions we set the output valE = 64’d1+valB.

PS: here we increment and decrement the value of pointer in stack by 1 in popq, ret, popq and call instructions because we have considered data memory as a 2-dimensional array containing 1024 words for 64bits.

Source code:

```
//EXECUTE BLOCK

`include "alu.v"

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

    input clk;
    input [3:0] icode;
    input [3:0] ifun;
    input [63:0] valA;
    input [63:0] valB;
    input [63:0] valC;

    output reg [63:0] valE;
    output reg cnd;

    // Condition Code [ZF, SF, OF]
    reg [2:0] CC;
    reg zf;
    reg sf;
    reg of;

    // reg [63:0] aluA, aluB;
    wire signed [63:0]ans;
    reg signed [63:0] ans_;
    wire [2:0] conCode;
    reg [2:0] CC_;
    reg signed [63:0] aluA, aluB;
    reg [1:0] op;


     // Instruction codes
     parameter IHALT   = 4'd0;
     parameter INOP    = 4'd1;
     parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
     parameter IIRMOVQ = 4'd3;
     parameter IRMMOVQ = 4'd4;
     parameter IMRMOVQ = 4'd5;
     parameter IOPQ    = 4'd6;
     parameter IJXX    = 4'd7;
     parameter ICALL   = 4'd8;
     parameter IRET    = 4'd9;
     parameter IPUSHQ  = 4'd10;
     parameter IPOPQ   = 4'd11;

     //Intialisation
     initial
     begin
         valE = 64'd0;
         cnd = 1'b0;
         CC = 3'd0;
         zf = 0;
         sf = 0;
         of = 0;
     end

     always@(*)
     begin
         if(clk == 1)
         begin
             zf = conCode[2];
             sf = conCode[1];
             of = conCode[0];
         end
     end

     alu_ uut(aluA, aluB, op, ans, conCode);
```

```
always @(*)
begin
    if(clk ==1)
    begin
        if(icode == IRRMOVQ)
        begin
            valE = valA;
            if(ifun == 4'd0) // unconditional
            begin
                cnd = 1;
            end
            else if(ifun == 4'd1) // less than equal to
            begin
                cnd = ((sf^of)| zf);
            end
            else if(ifun == 4'd2) // less than
            begin
                cnd = (sf^of);
            end
            else if(ifun == 4'd3) // equal to
            begin
                cnd = zf;
            end
            else if(ifun == 4'd4) // not equal to
            begin
                cnd = ~zf;
            end
            else if(ifun == 4'd5) // greater than equal to
            begin
                cnd = ~(sf^of);
            end
            else if(ifun == 4'd6) // greater than
            begin
                cnd = ((~(sf^of))&~zf);
            end
        end
        else if(icode == IIRMOVQ)
        begin
            valE = valC;
        end
        else if(icode == IRMMOVQ)
        begin
            valE = valC+valB;
        end
        else if(icode == IMRMOVQ)
        begin
            valE = valC+valB;
        end
        else if(icode == IOPQ)
        begin
            aluA = valB;
            aluB = valA;
            if(ifun==4'b0000)
                op = 2'b00;
            else if(ifun==4'b0001)
                op = 2'b01;
            else if(ifun==4'b0010)
                op = 2'b10;
            else
                op = 2'b11;
            // assign ans_ = ans;
            valE = ans;
            // assign CC_ = conCode;
            CC = conCode;
        end
```

```
        else if(icode== IJXX)
        begin
            if(ifun == 4'd0)
            begin
                cnd = 1;
            end
            else if(ifun == 4'd1)
            begin
                cnd = ((sf^of)| zf);
            end
            else if(ifun == 4'd2)
            begin
                cnd = (sf^of);
            end
            else if(ifun == 4'd3)
            begin
                cnd = zf;
            end
            else if(ifun == 4'd4)
            begin
                cnd = ~zf;
            end
            else if(ifun == 4'd5)
            begin
                cnd = ~(sf^of);
            end
            else if(ifun == 4'd6)
            begin
                cnd = ((~(sf^of))&~zf);
            end
        end
        else if(icode == ICALL)
        begin
            valE = -64'd1+valB;
        end
```

```
        else if(icode == IRET)
        begin
            valE = 64'd1+valB;
        end
        else if(icode == IPUSHQ)
        begin
            valE = -64'd1+valB;
        end
        else if(icode == IPOPQ)
        begin
            valE = 64'd1+valB;
        end

    end
end

endmodule
```

Test Bench:

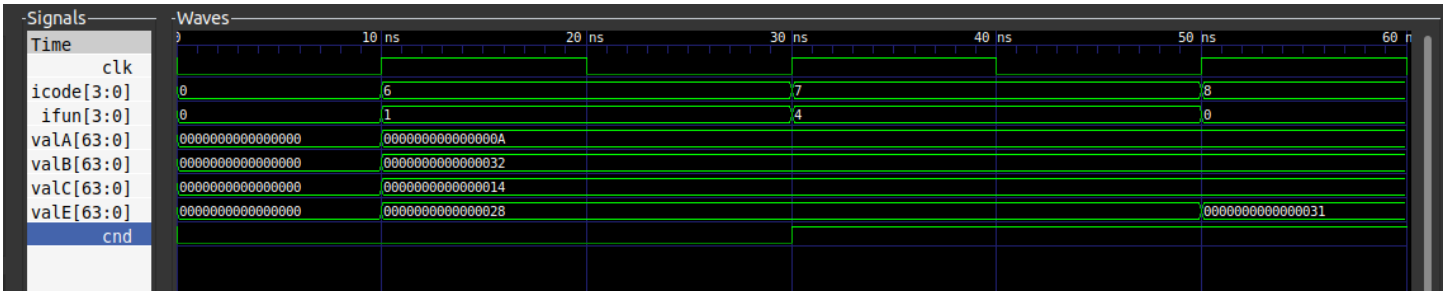
```
`timescale 1ns / 10ps

module execute_tb;
  reg clk;

  reg [3:0] icode;
  reg [3:0] ifun;
  reg [63:0] valC;
  reg [63:0] valA;
  reg [63:0] valB;
  output signed [63:0] valE;
  output cnd;

  execute execute(
    .clk(clk),
    .icode(icode),
    .ifun(ifun),
    .valA(valA),
    .valB(valB),
    .valC(valC),
    .valE(valE),
    .cnd(cnd)
  );

  initial
  begin

    $dumpfile("execute_tb.vcd");
    $dumpvars(0, execute_tb);

    clk=0;
    icode=4'd0;
    ifun=4'd0;
    valC=64'd0;
    valA=64'd0;
    valB=64'd0;

  end
```

```
  initial begin
    clk=0;

    #10 clk=~clk;
    icode=4'b0110;
    ifun=4'b0001;
    valA=64'd10;
    valB=64'd50;
    valC=64'd20;
    #10 clk=~clk;
    #10 clk=~clk;
    icode=4'b0111;
    ifun=4'b0100;
    valA=64'd10;
    valB=64'd50;
    valC=64'd20;
    #10 clk=~clk;
    #10 clk=~clk;
    icode=4'b1000;
    ifun=4'b0000;
    valA=64'd10;
    valB=64'd50;
    valC=64'd20;
    #10 clk=~clk;

  end

  initial
    $monitor("clk=%d icode=%b ifun=%b valA=%d valB=%d valC=%d valE=%d cnd=%d\n",clk,icode,ifun,valA,valB,valC,valE, cnd);
endmodule
```

Output of Test Bench:

```
clk=0 icode=0000 ifun=0000 valA=                   0 valB=                   0 valC=                   0 valE=                   0 cnd=0

clk=1 icode=0110 ifun=0001 valA=                  10 valB=                  50 valC=                  20 valE=                  40 cnd=0

clk=0 icode=0110 ifun=0001 valA=                  10 valB=                  50 valC=                  20 valE=                  40 cnd=0

clk=1 icode=0111 ifun=0100 valA=                  10 valB=                  50 valC=                  20 valE=                  40 cnd=1

clk=0 icode=0111 ifun=0100 valA=                  10 valB=                  50 valC=                  20 valE=                  40 cnd=1

clk=1 icode=1000 ifun=0000 valA=                  10 valB=                  50 valC=                  20 valE=                  49 cnd=1

clk=0 icode=1000 ifun=0000 valA=                  10 valB=                  50 valC=                  20 valE=                  49 cnd=1
```

Gtkwave output:



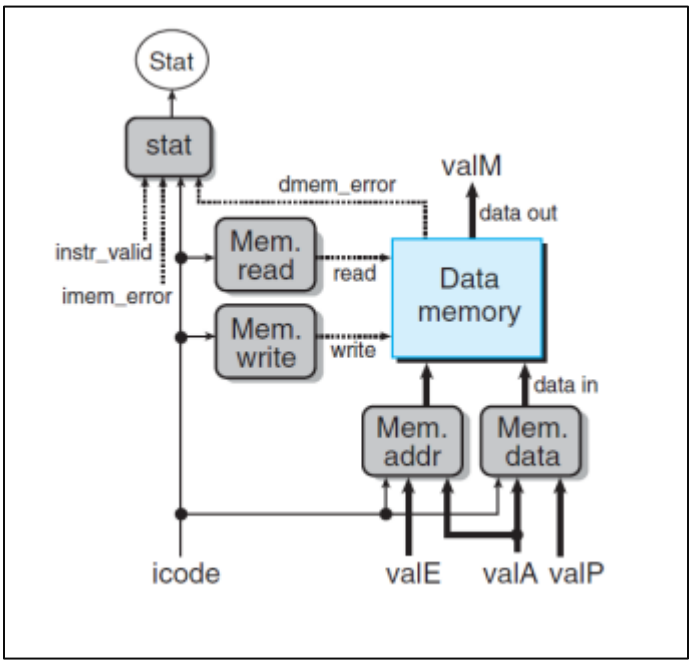
Command to run the code:

iverilog –o execute execute_tb.v execute.v

vvp execute

MEMORY

The memory stage is responsible for reading from and writing data into the main memory. Reading and writing depends in the instruction specified by icode. The values read from the main memory are outputted as valM. If values are to be written, the icode decides whether value of valP or valA is written and destination is given by valE.



Implementation and Working:

- We have declared the data memory as a register array in the memory stage as access to the memory stage is only required here.

```
//Data Memory
reg [63:0] data_memory [1023:0];
```

- We read and write data to the data memory as follows:
  - We write to the data memory for 'rmmovq', 'call' and 'pushq' instructions
    - We do data_memory[valE] = valA for rmmovq and pushq
    - We do data_memory[valE] = valP for call
  - We read from the data memory for 'mrmovq', 'ret' and 'popq' instructions
    - We do valM = data_memory[valE] for mrmovq and popq
    - We do valM = data_memory[valE] for ret

➢ If the memory address accessed by the memory block is out of bounds (here greater than 1023) then we set the dmem_error =1.

Source code

```
//MEMORY BLOCK

module memory(clk, icode, valA, valE, valP, valM, dmem_error, datamem, memory_address);

    input clk;
    input[3:0] icode;
    input [63:0] valA;
    input [63:0] valE;
    input [63:0] valP;
    output reg [63:0] valM;
    output reg dmem_error;
    output reg [63:0] datamem;

    //Data Memory
    reg [63:0] data_memory [1023:0];

    output reg [63:0] memory_address;

    // Initialization
    initial
    begin
        valM = 64'd0;
        dmem_error = 1'b0;
        memory_address = 1'b0;
    end

    // instruction codes
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
    // Hardcoding Data Memory
    initial
    begin
        data_memory[0] = 64'd0;
        data_memory[1] = 64'd1;
        data_memory[2] = 64'd2;
        data_memory[3] = 64'd3;
        data_memory[4] = 64'd4;
        data_memory[5] = 64'd5;
        data_memory[6] = 64'd6;
        data_memory[7] = 64'd7;
        data_memory[8] = 64'd8;
        data_memory[9] = 64'd9;
        data_memory[10] = 64'd10;
        data_memory[11] = 64'd11;
        data_memory[12] = 64'd12;
        data_memory[13] = 64'd13;
        data_memory[14] = 64'd14;
        data_memory[15] = 64'd15;
        data_memory[16] = 64'd16;
        data_memory[17] = 64'd17;
        data_memory[18] = 64'd18;
        data_memory[19] = 64'd19;
        data_memory[20] = 64'd20;
    end
```

```
    always@(*)
    begin
        if(clk==1)
        begin
            if(icode == IRMMOVQ)
            begin
                memory_address = valE;
                data_memory[valE] = valA;
            end
            else if (icode == IMRMOVQ)
            begin
                memory_address = valE;
                valM = data_memory[valE];
            end
            else if (icode == ICALL)
            begin
                memory_address = valE;
                data_memory[valE] = valP;
            end
            else if (icode == IRET)
            begin
                memory_address = valA;
                valM = data_memory[valA];
            end
            else if (icode == IPUSHQ)
            begin
                memory_address = valE;
                data_memory[valE] = valA;
            end
            else if (icode == IPOPQ)
            begin
                memory_address = valA;
                valM = data_memory[valA];
            end
            datamem = data_memory[valE];
        end
    end
```

```
    always@(*)
    begin
        if(clk==0)
        begin
            if(memory_address>64'd1023)
            begin
                dmem_error = 1'd1;
            end
        end
    end

endmodule
```

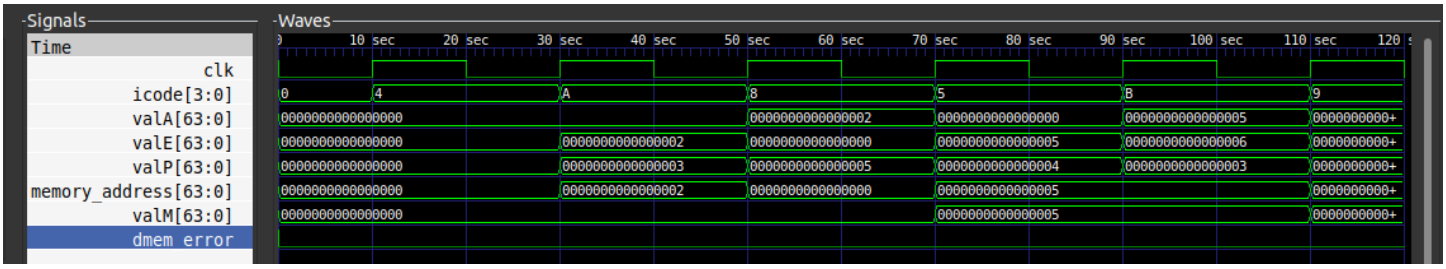
Test Bench:

```
module memory_tb;

    reg clk;
    reg [3:0] icode;
    reg [63:0] valA;
    reg [63:0] valE;
    reg [63:0] valP;
    output [63:0] valM;
    output dmem_error;
    output [63:0] datamem;
    output [63:0] memory_address;

    memory uut(
        .clk(clk),
        .icode(icode),
        .valA(valA),
        .valE(valE),
        .valP(valP),
        .valM(valM),
        .dmem_error(dmem_error),
        .datamem(datamem),
        .memory_address(memory_address)
    );

    initial
    begin

        $dumpfile("memory_tb.vcd");
        $dumpvars(0, memory_tb);

        clk=0;
        icode=4'd0;
        valA=64'd0;
        valE=64'd0;
        valP=64'd0;

    end

     initial
     begin

        #10 clk = ~clk;
        icode = 4'b0100; valA = 64'd0; valE = 64'd0; valP = 64'd0;
        #10 clk = ~clk;
        #10 clk = ~clk;
        icode = 4'b1010; valA = 64'd0; valE = 64'd2; valP = 64'd3;
        #10 clk = ~clk;
        #10 clk = ~clk;
        icode = 4'b1000; valA = 64'd2; valE = 64'd0; valP = 64'd5;
        #10 clk = ~clk;
        #10 clk = ~clk;
        icode = 4'b0101; valA = 64'd0; valE = 64'd5; valP = 64'd4;
        #10 clk = ~clk;
        #10 clk = ~clk;
        icode = 4'b1011; valA = 64'd5; valE = 64'd6; valP = 64'd3;
        #10 clk = ~clk;
        #10 clk = ~clk;
        icode = 4'b1001; valA = 64'd4; valE = 64'd1; valP = 64'd7;
        #10 clk = ~clk;
     end

     initial
        $monitor("clk = %d icode = %b valA = %g valE = %g valP = %g valM = %g dmem_error = %d datamem = %g memory_address = %g\n"
        ,clk,icode,valA,valE,valP,valM, dmem_error, datamem,memory_address);

endmodule
```

Output of Test Bench:

```
clk = 0 icode = 0000 valA = 0 valE = 0 valP = 0 valM = 0 dmem_error = 0 datamem = 0 memory_address = 0

clk = 1 icode = 0100 valA = 0 valE = 0 valP = 0 valM = 0 dmem_error = 0 datamem = 0 memory_address = 0

clk = 0 icode = 0100 valA = 0 valE = 0 valP = 0 valM = 0 dmem_error = 0 datamem = 0 memory_address = 0

clk = 1 icode = 1010 valA = 0 valE = 2 valP = 3 valM = 0 dmem_error = 0 datamem = 0 memory_address = 2

clk = 0 icode = 1010 valA = 0 valE = 2 valP = 3 valM = 0 dmem_error = 0 datamem = 0 memory_address = 2

clk = 1 icode = 1000 valA = 2 valE = 0 valP = 5 valM = 0 dmem_error = 0 datamem = 5 memory_address = 0

clk = 0 icode = 1000 valA = 2 valE = 0 valP = 5 valM = 0 dmem_error = 0 datamem = 5 memory_address = 0

clk = 1 icode = 0101 valA = 0 valE = 5 valP = 4 valM = 5 dmem_error = 0 datamem = 5 memory_address = 5

clk = 0 icode = 0101 valA = 0 valE = 5 valP = 4 valM = 5 dmem_error = 0 datamem = 5 memory_address = 5

clk = 1 icode = 1011 valA = 5 valE = 6 valP = 3 valM = 5 dmem_error = 0 datamem = 6 memory_address = 5

clk = 0 icode = 1011 valA = 5 valE = 6 valP = 3 valM = 5 dmem_error = 0 datamem = 6 memory_address = 5

clk = 1 icode = 1001 valA = 4 valE = 1 valP = 7 valM = 4 dmem_error = 0 datamem = 1 memory_address = 4

clk = 0 icode = 1001 valA = 4 valE = 1 valP = 7 valM = 4 dmem_error = 0 datamem = 1 memory_address = 4
```

Gtkwave output:



Command to run the code:

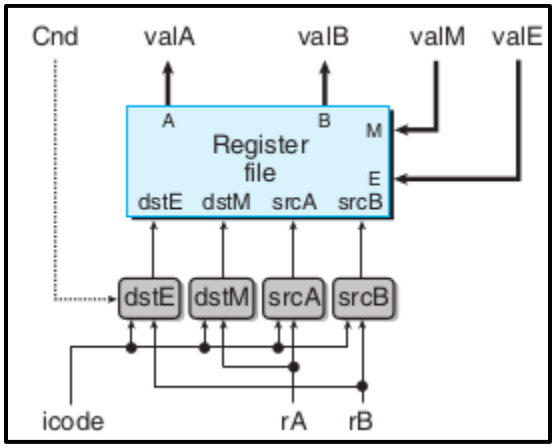
iverilog –o memory memory_tb.v memory.v

vvp memory

WRITE BACK

In the write back stage, values are written back into either of the main 15 registers, that is the values valE, or valM are written into the registers with addresses rA, rB or RRSP depending on the icode.

The decode and write back stages can be combined as they both access the register file.



Implementation and Working:

- We have implemented the register memory in a text file which contains the values stored in the registers. This way of implementation helps us access the values stored in the registers in both the decode and write back stage. In the write back stage the values in the text file are read into the 'Reg_File' register array and at the end of the stage they are written back to the text file.

```
always@(*)
begin
    if(clk==1)
        $readmemh("reg_file.txt", Reg_File);

end

always@(*)
begin
    if(clk == 0)
    begin
        $writememh("reg_file.txt", Reg_File);
    end
end
```

- The values valE or valM are written into register with register code rA, Rb or RRSP depending on the icode as follows:
  - Reg_File[rA] = valM for icode in {IMRMOVQ, IPOPQ}
  - Reg_File[rB] = valE for icode in {IRRMOVQ & cnd = 1, IIRMOVQ, IOPQ}
  - Reg_File[RRSP] = valE for icode in {ICALL, IRET, IPUSHQ, IPOPQ}

Source code:

```
// WRITE BACK BLOCK

module write_back(clk, icode, rA, rB, cnd, valE, valM);

    input clk;
    input [3:0] icode;
    input [3:0] rA;
    input [3:0] rB;
    input cnd;
    input [63:0] valE;
    input [63:0] valM;

    reg [3:0] dstE;
    reg [3:0] dstM;

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

    integer i;

    initial
    begin
        dstE = 4'd15;
        dstM = 4'd15;
    end

    // instruction codes
    // Constant values used in HCL descriptions.
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
```

```
    // registers
    parameter Register_rax = 4'd0;
    parameter Register_rcx = 4'd1;
    parameter Register_rdx = 4'd2;
    parameter Register_rbx = 4'd3;
    parameter RRSP         = 4'd4;
    parameter Register_rbp = 4'd5;
    parameter Register_rsi = 4'd6;
    parameter Register_rdi = 4'd7;
    parameter Register_r8  = 4'd8;
    parameter Register_r9  = 4'd9;
    parameter Register_r10 = 4'd10;
    parameter Register_r11 = 4'd11;
    parameter Register_r12 = 4'd12;
    parameter Register_r13 = 4'd13;
    parameter Register_r14 = 4'd14;
    parameter RNONE        = 4'd15;


    always@(*)
    begin
        if(clk==1)
            $readmemh("reg_file.txt", Reg_File);

    end

    always@(*)
    begin
        if(clk == 0)
        begin
            $writememh("reg_file.txt", Reg_File);
        end
    end

    always@(negedge clk)
    begin

        //rrmovq and  cmovXX
        if (icode == IRRMOVQ && cnd == 1)
        begin

            // R[rB] ← valE
            dstE = rB;
            Reg_File[dstE] = valE;

        end
```

```
        //irmovq
        else if (icode == IIRMOVQ)
        begin

            // R[rB] ← valE
            dstE = rB;
            Reg_File[dstE] = valE;

        end
        // mrmovq
        else if (icode == IMRMOVQ)
        begin

            // R[rA] ← valM
            dstM = rA;
            Reg_File[dstM] = valM;

        end
        //Opq
        else if (icode == IOPQ)
        begin

            // R[rB] ← valE
            dstE = rB;
            Reg_File[dstE] = valE;

        end
        //call
        else if (icode == ICALL)
        begin

            // R[ %rsp ] ← valE
            dstE = RRSP;
            Reg_File[dstE] = valE;

        end
        //ret
        else if (icode == IRET)
        begin

            // R[ %rsp ] ← valE
            dstE = RRSP;
            Reg_File[dstE] = valE;

        end
          //pushq
          else if (icode == IPUSHQ)
          begin

              // R[ %rsp ] ← valE
              dstE = RRSP;
              Reg_File[dstE] = valE;

          end
          //popq
          else if (icode == IPOPQ)
          begin

              // R[ %rsp ] ← valE
              dstE = RRSP;
              Reg_File[dstE] = valE;

              // R[rA] ← valM
              dstM = rA;
              Reg_File[dstM] = valM;

          end

    end


endmodule
```

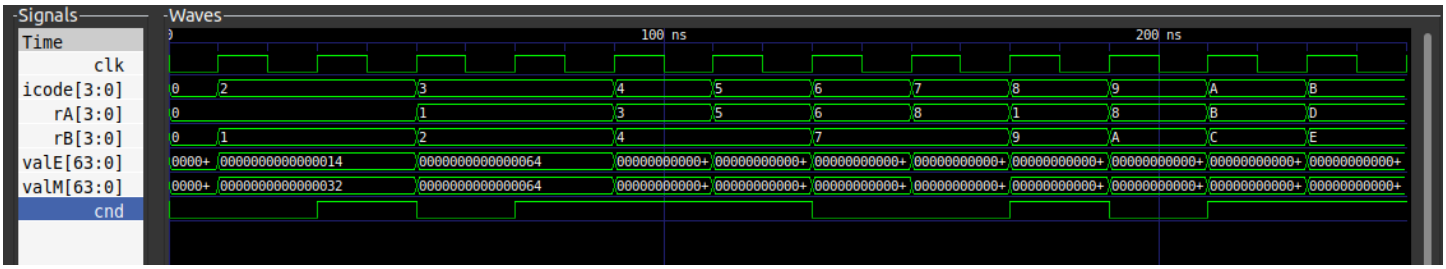
Test bench:

```
`timescale 1ns / 10ps

module write_back_tb();

    reg clk;
    reg [3:0] icode;
    reg [3:0] rA;
    reg [3:0] rB;
    reg cnd;
    reg [63:0] valE;
    reg [63:0] valM;

    write_back UUT(clk, icode, rA, rB, cnd, valE, valM);

    //creating clk
    initial
    begin
        clk = 0;
        repeat (25)  #10 clk = ~clk;
    end

    initial
    begin

        $dumpfile("write_back_tb.vcd");
        $dumpvars(0, write_back_tb);

        clk=0;
        icode=4'd0;
        rA=4'd0;
        rB=4'd0;
        cnd=0;
        valE=64'd0;
        valM=64'd0;

    end
        initial
        begin

            #10
            icode=4'd2;rA=4'd0;rB=4'd1;cnd=0;valE=64'd20;valM=64'd50;

            #20
            icode=4'd2;rA=4'd0;rB=4'd1;cnd=1;valE=64'd20;valM=64'd50;

            #20
            icode=4'd3;rA=4'd1;rB=4'd2;cnd=0;valE=64'd100;valM=64'd100;

            #20
            icode=4'd3;rA=4'd1;rB=4'd2;cnd=1;valE=64'd100;valM=64'd100;

            #20
            icode=4'd4;rA=4'd3;rB=4'd4;cnd=1;valE=64'd256;valM=64'd10;

            #20
            icode=4'd5;rA=4'd5;rB=4'd4;cnd=1;valE=64'd1024;valM=64'd51;

            #20
            icode=4'd6;rA=4'd6;rB=4'd7;cnd=0;valE=64'd91;valM=64'd1;

            #20
            icode=4'd7;rA=4'd8;rB=4'd7;cnd=0;valE=64'd34;valM=64'd0;

            #20
            icode=4'd8;rA=4'd1;rB=4'd9;cnd=1;valE=64'd720;valM=64'd3;

            #20
            icode=4'd9;rA=4'd8;rB=4'd10;cnd=0;valE=64'd2222;valM=64'd189302;

            #20
            icode=4'd10;rA=4'd11;rB=4'd12;cnd=1;valE=64'd77821;valM=64'd3321;

            #20
            icode=4'd11;rA=4'd13;rB=4'd14;cnd=1;valE=64'd728782;valM=64'd5849;

        end

        initial
        begin
            $monitor($time, "\tclk = %d icode = %b rA = %b rB = %b Cnd = %b valE = %g valM = %g\n",clk,icode,rA,rB,cnd,valE,valM);
        end
    endmodule
```

Output of Test Bench:

```
  0    clk = 0 icode = 0000 rA = 0000 rB = 0000 Cnd = 0 valE = 0 valM = 0
 10    clk = 1 icode = 0010 rA = 0000 rB = 0001 Cnd = 0 valE = 20 valM = 50
 20    clk = 0 icode = 0010 rA = 0000 rB = 0001 Cnd = 0 valE = 20 valM = 50
 30    clk = 1 icode = 0010 rA = 0000 rB = 0001 Cnd = 1 valE = 20 valM = 50
 40    clk = 0 icode = 0010 rA = 0000 rB = 0001 Cnd = 1 valE = 20 valM = 50
 50    clk = 1 icode = 0011 rA = 0001 rB = 0010 Cnd = 0 valE = 100 valM = 100
 60    clk = 0 icode = 0011 rA = 0001 rB = 0010 Cnd = 0 valE = 100 valM = 100
 70    clk = 1 icode = 0011 rA = 0001 rB = 0010 Cnd = 1 valE = 100 valM = 100
 80    clk = 0 icode = 0011 rA = 0001 rB = 0010 Cnd = 1 valE = 100 valM = 100
 90    clk = 1 icode = 0100 rA = 0011 rB = 0100 Cnd = 1 valE = 256 valM = 10
100    clk = 0 icode = 0100 rA = 0011 rB = 0100 Cnd = 1 valE = 256 valM = 10
110    clk = 1 icode = 0101 rA = 0101 rB = 0100 Cnd = 1 valE = 1024 valM = 51
120    clk = 0 icode = 0101 rA = 0101 rB = 0100 Cnd = 1 valE = 1024 valM = 51
130    clk = 1 icode = 0110 rA = 0110 rB = 0111 Cnd = 0 valE = 91 valM = 1
140    clk = 0 icode = 0110 rA = 0110 rB = 0111 Cnd = 0 valE = 91 valM = 1
150    clk = 1 icode = 0111 rA = 1000 rB = 0111 Cnd = 0 valE = 34 valM = 0
160    clk = 0 icode = 0111 rA = 1000 rB = 0111 Cnd = 0 valE = 34 valM = 0
170    clk = 1 icode = 1000 rA = 0001 rB = 1001 Cnd = 1 valE = 720 valM = 3
180    clk = 0 icode = 1000 rA = 0001 rB = 1001 Cnd = 1 valE = 720 valM = 3
190    clk = 1 icode = 1001 rA = 1000 rB = 1010 Cnd = 0 valE = 2222 valM = 189302
200    clk = 0 icode = 1001 rA = 1000 rB = 1010 Cnd = 0 valE = 2222 valM = 189302
210    clk = 1 icode = 1010 rA = 1011 rB = 1100 Cnd = 1 valE = 77821 valM = 3321
220    clk = 0 icode = 1010 rA = 1011 rB = 1100 Cnd = 1 valE = 77821 valM = 3321
230    clk = 1 icode = 1011 rA = 1101 rB = 1110 Cnd = 1 valE = 728782 valM = 5849
240    clk = 0 icode = 1011 rA = 1101 rB = 1110 Cnd = 1 valE = 728782 valM = 5849
```

Gtkwave output:



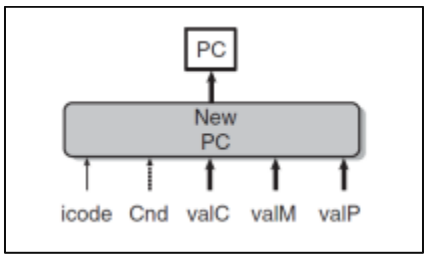
Command to run the code:

iverilog –o write_back write_back_tb.v write_back.v

vvp write_back

PC UPDATE

The PC update stage is responsible for calculating the value of updated_pc, the value of pc that contains the location of next instruction. The value of updated_pc depends on the values of icode, valC, valM, valP and cnd.



Implementation and Working:

- The value of the updated_pc is decided as follows
  - For 'jXX' instruction if the cnd = 1 then updated_pc = valC otherwise updated_pc = valP
  - For 'call' instruction updated_pc = valC
  - For 'ret' instruction updated_pc = valM
  - For all the other instructions updated_pc = valP

After all the 6 stages are over, the fetch stage of the next instruction occurs and the cycle continues the 'halt' instruction is encountered.

Source code

```
//PC UPDATE BLOCK

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

    input clk;
    input cnd;
    input [3:0] icode;
    input [63:0] valC;
    input [63:0] valM;
    input [63:0] valP;
    input [63:0] PC;
    output reg [63:0] updated_pc;

    // instruction codes
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
```

```
    always@(*)
    begin
        if(icode == IJXX)
        begin
            if(cnd == 1'b1)
                updated_pc = valC;
            else
                updated_pc = valP;
        end
        else if(icode == ICALL)
        begin
            updated_pc = valC;
        end
        else if(icode == IRET)
        begin
            updated_pc = valM;
        end
        else
        begin
            updated_pc  = valP;
        end
    end

endmodule
```

Test bench:

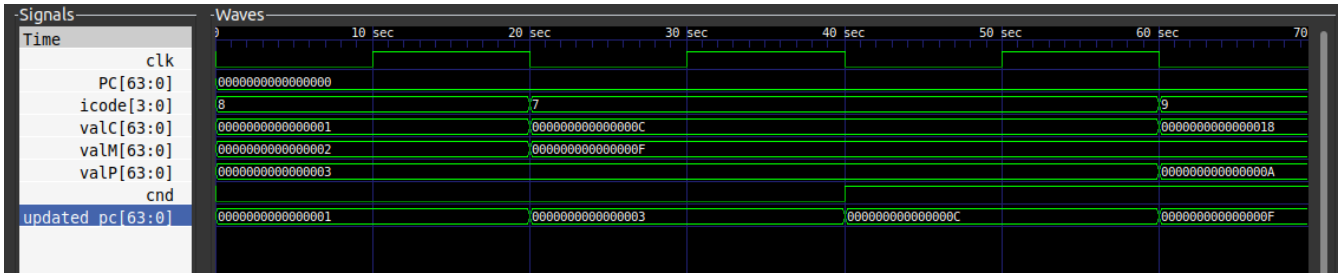
```
module pc_update_tb;

reg clk;
reg [3:0] icode;
reg cnd;
reg [63:0] PC;
reg [63:0] valC;
reg [63:0] valP;
reg [63:0] valM;
output [63:0] updated_pc;

pc_update uut(
    .clk(clk),
    .cnd(cnd),
    .icode(icode),
    .valC(valC),
    .valM(valM),
    .valP(valP),
    .updated_pc(updated_pc)
  );

initial
begin

    $dumpfile("pc_update_tb.vcd");
    $dumpvars(0, pc_update_tb);

    clk=0;
    icode=4'd0;
    cnd=0;
    valC=64'd0;
    valP=64'd0;
    valM=64'd0;

end
```

```
initial begin

    clk=1'b0;
    PC = 64'h0000;

    icode = 4'b1000; valC = 64'd1; valM = 64'd2; valP = 64'd3; cnd = 1'b0;
    #10 clk = ~clk;
    #10 clk = ~clk;
    icode = 4'b0111; valC = 64'd12; valM = 64'd15; valP = 64'd3; cnd = 1'b0;
    #10 clk = ~clk;
    #10 clk = ~clk;
    icode = 4'b0111; valC = 64'd12; valM = 64'd15; valP = 64'd3; cnd = 1'b1;
    #10 clk = ~clk;
    #10 clk = ~clk;
    icode = 4'b1001; valC = 64'd24; valM = 64'd15; valP = 64'd10; cnd = 1'b1;
    #10 clk = ~clk;

end

initial
    $monitor("clk = %d icode = %b valC = %g valP = %g valM = %g cnd = %g updated_pc = %g\n"
    ,clk,icode,valC,valP,valM,cnd,updated_pc);

endmodule
```

Output of Test Bench:

```
clk = 0 icode = 1000 valC = 1 valP = 3 valM = 2 cnd = 0 updated_pc = 1

clk = 1 icode = 1000 valC = 1 valP = 3 valM = 2 cnd = 0 updated_pc = 1

clk = 0 icode = 0111 valC = 12 valP = 3 valM = 15 cnd = 0 updated_pc = 3

clk = 1 icode = 0111 valC = 12 valP = 3 valM = 15 cnd = 0 updated_pc = 3

clk = 0 icode = 0111 valC = 12 valP = 3 valM = 15 cnd = 1 updated_pc = 12

clk = 1 icode = 0111 valC = 12 valP = 3 valM = 15 cnd = 1 updated_pc = 12

clk = 0 icode = 1001 valC = 24 valP = 10 valM = 15 cnd = 1 updated_pc = 15

clk = 1 icode = 1001 valC = 24 valP = 10 valM = 15 cnd = 1 updated_pc = 15
```

Gtkwave output:



Command to run the code:

iverilog –o pc_update pc_update_tb.v pc_update.v

vvp pc_update

SEQUENTIAL PROCESSOR

Source code:

```
`timescale 1ns/10ps

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

module seq();
    reg clk;
    reg [63:0] PC;

    reg stat_AOK;
    reg stat_INS;
    reg stat_HLT;
    reg stat_ADR;

    wire [3:0] icode;
    wire [3:0] ifun;
    wire [3:0] rA;
    wire [3:0] rB;
    wire [63:0] valC;
    wire [63:0] valP;
    wire instr_valid;
    wire imem_error;
    wire dmem_error;
    wire [63:0] valA;
    wire [63:0] valB;
    wire signed [63:0] valE;
    wire [63:0] valM;
    wire [63:0] datamem;
    wire cnd;
    output hlt;
    wire [63:0] updated_pc;
    wire [63:0] memory_address;

    fetch fetch_(clk, PC, icode, ifun, rA, rB, valC, valP, imem_error, instr_valid, hlt);
    decode decode_(clk, icode, rA, rB, valA, valB);
    execute execute_(clk, icode, ifun, valA, valB, valC, valE, cnd);
    memory memory_(clk, icode, valA, valE, valP, valM, dmem_error, datamem, memory_address);
    write_back write_back_(clk, icode, rA, rB, cnd, valE, valM);
    pc_update pc_update_(clk, cnd, icode, valC, valM, valP, updated_pc);

    initial
    begin
        $dumpfile("seq.vcd");
        $dumpvars(0, seq);
        clk = 1;
        PC = 64'd39;
        stat_AOK = 1;
        stat_INS = 0;
        stat_HLT = 0;
    stat_ADR = 0;
    end

    always #5 clk =~ clk;

    always@(*)
    begin
        if(hlt)
        begin
        stat_AOK=1'b0;
        stat_INS=1'b0;
        stat_HLT=1'b1;
        stat_ADR=1'b0;
        end
        else if(instr_valid)
        begin
        stat_AOK=1'b0;
        stat_INS=1'b1;
        stat_HLT=1'b0;
        stat_ADR=1'b0;
        end
        else if(imem_error == 1 || dmem_error == 1)
        begin
        stat_AOK=1'b0;
        stat_INS=1'b0;
        stat_HLT=1'b0;
        stat_ADR=1'b1;
        end
        else
        begin
        stat_AOK=1'b1;
        stat_INS=1'b0;
        stat_HLT=1'b0;
        stat_ADR=1'b0;
        end
    end

    always@(*)
    begin
        PC = updated_pc;
    end

    always@(*)
    begin
        if(stat_ADR == 1)
        begin
            $finish;
        end
        else if (stat_HLT == 1)
        begin
            $finish;
        end
    end

initial
    $monitor($time, "    clk = %d icode = %b ifun = %b rA = %b rB = %b\n\t\t\tvalA = %g valB = %g valC = %g valE = %g valM = %g updated_pc = %g\n
    \t\t\tdatamem = %g memory_address = %g dmem_error = %d instr_invalid = %d memory_error = %d cnd = %d HLT = %d AOK = %d INS = %d ADR = %d\n",
    clk,icode,ifun,rA,rB,valA,valB,valC,valE,valM,updated_pc,datamem,memory_address,dmem_error,instr_valid,imem_error,cnd,stat_HLT,stat_AOK,stat_INS,stat_ADR);

endmodule
```

Instructions given to the processor:

```
irmovq $0x0, %rax

irmovq $0x10, %rdx
```

```
irmovq $0xc, %rbx
jmp check
check:
    addq %rax, %rbx
    je rbxres
    addq %rax, %rdx
    je rdxres
    jmp loop2
rbxres:
rdxres:
Loop2:
    halt
```

Output of the Sequential processor

```
  0    clk = 1 icode = 0011 ifun = 0000 rA = 0000 rB = 0000
       valA = 0 valB = 0 valC = 0 valE = 0 valM = 0 updated_pc = 10
       datamem = 0 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

  5    clk = 0 icode = 0011 ifun = 0000 rA = 0000 rB = 0000
       valA = 0 valB = 0 valC = 0 valE = 0 valM = 0 updated_pc = 10
       datamem = 0 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 10    clk = 1 icode = 0011 ifun = 0000 rA = 0000 rB = 0010
       valA = 0 valB = 0 valC = 16 valE = 16 valM = 0 updated_pc = 20
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 15    clk = 0 icode = 0011 ifun = 0000 rA = 0000 rB = 0010
       valA = 0 valB = 0 valC = 16 valE = 16 valM = 0 updated_pc = 20
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 20    clk = 1 icode = 0011 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 0 valC = 12 valE = 12 valM = 0 updated_pc = 30
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 25    clk = 0 icode = 0011 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 0 valC = 12 valE = 12 valM = 0 updated_pc = 30
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 30    clk = 1 icode = 0111 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 0 valC = 39 valE = 12 valM = 0 updated_pc = 39
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 1 HLT = 0 AOK = 1 INS = 0 ADR = 0

 35    clk = 0 icode = 0111 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 0 valC = 39 valE = 12 valM = 0 updated_pc = 39
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 1 HLT = 0 AOK = 1 INS = 0 ADR = 0

 40    clk = 1 icode = 0110 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 12 valC = 39 valE = 12 valM = 0 updated_pc = 41
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 1 HLT = 0 AOK = 1 INS = 0 ADR = 0

 45    clk = 0 icode = 0110 ifun = 0000 rA = 0000 rB = 0011
       valA = 0 valB = 12 valC = 39 valE = 12 valM = 0 updated_pc = 41
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 1 HLT = 0 AOK = 1 INS = 0 ADR = 0

 50    clk = 1 icode = 0111 ifun = 0011 rA = 0000 rB = 0011
       valA = 0 valB = 12 valC = 122 valE = 12 valM = 0 updated_pc = 50
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 55    clk = 0 icode = 0111 ifun = 0011 rA = 0000 rB = 0011
       valA = 0 valB = 12 valC = 122 valE = 12 valM = 0 updated_pc = 50
       datamem = 12 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 60    clk = 1 icode = 0110 ifun = 0000 rA = 0000 rB = 0010
       valA = 0 valB = 16 valC = 122 valE = 16 valM = 0 updated_pc = 52
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 65    clk = 0 icode = 0110 ifun = 0000 rA = 0000 rB = 0010
       valA = 0 valB = 16 valC = 122 valE = 16 valM = 0 updated_pc = 52
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 70    clk = 1 icode = 0111 ifun = 0011 rA = 0000 rB = 0010
       valA = 0 valB = 16 valC = 125 valE = 16 valM = 0 updated_pc = 61
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 75    clk = 0 icode = 0111 ifun = 0011 rA = 0000 rB = 0010
       valA = 0 valB = 16 valC = 125 valE = 16 valM = 0 updated_pc = 61
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 0 AOK = 1 INS = 0 ADR = 0

 80    clk = 1 icode = 0000 ifun = 0000 rA = 0000 rB = 0010
       valA = 0 valB = 16 valC = 125 valE = 16 valM = 0 updated_pc = 62
       datamem = 16 memory_address = 0 dmem_error = 0 instr_invalid = 0 memory_error = 0 cnd = 0 HLT = 1 AOK = 0 INS = 0 ADR = 0
```

Register file before running the seq.v:

seq > reg_file.txt

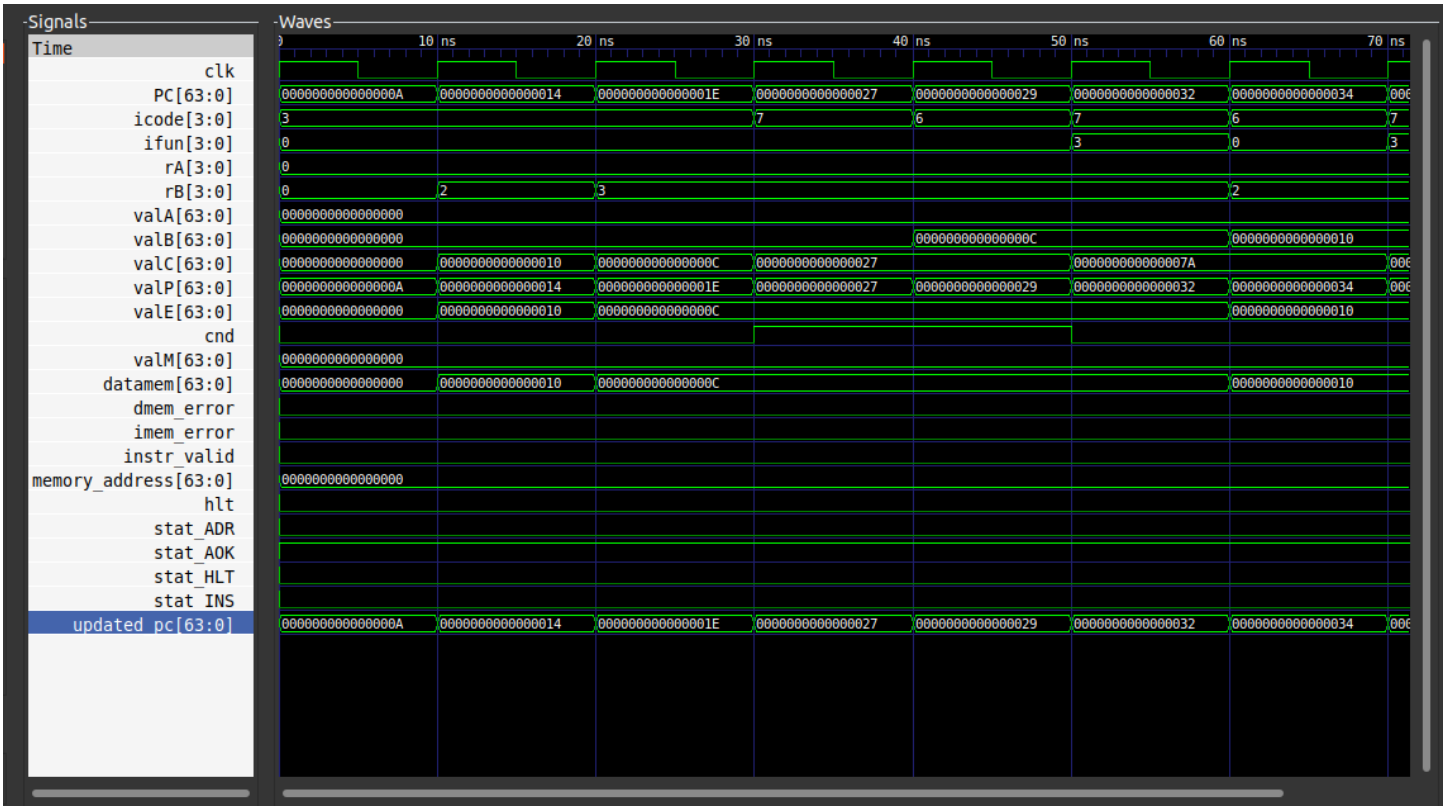
```
1   // 0x00000000
2   0000000000000000
3   0000000000000000
4   0000000000000000
5   0000000000000000
6   0000000000000000
7   0000000000000000
8   0000000000000000
9   0000000000000000
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

Register file after running the seq.v

seq > reg_file.txt

```
1   // 0x00000000
2   0000000000000000
3   0000000000000000
4   0000000000000010
5   000000000000000c
6   0000000000000000
7   0000000000000000
8   0000000000000000
9   0000000000000000
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

Gtkwave output:



Command to run the code for sequential processor:

iverilog –o seq seq.v

vvp seq

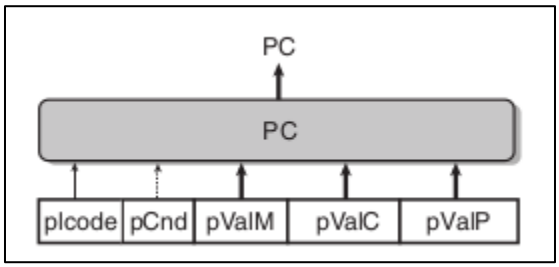
PIPE: 5-STAGE PIPELINED Y86-64 IMPLEMENTATION

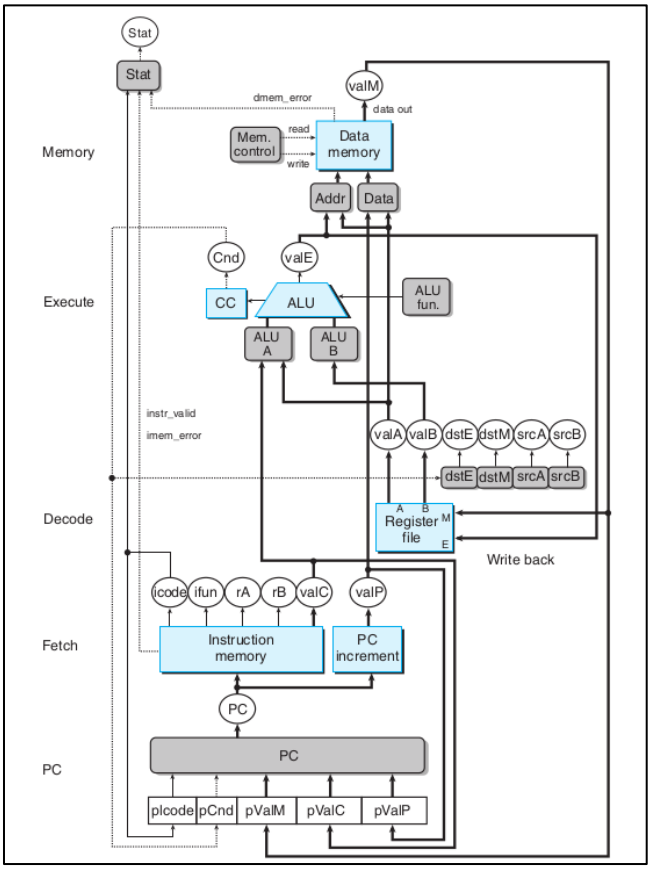
A pipelined processor is a type of computer processor architecture that allows multiple instructions to be executed simultaneously, by breaking the instruction execution cycle into smaller stages. As a result, multiple instructions can be in various stages of execution at the same time. This overlapping of instructions allows the processor to achieve higher performance and throughput.

Designing a pipelined Y86-64 processor from a sequential processor:

SEQ+

To start the implementation of the pipelined processor, we start by rearranging the stages of SEQ. The PC update stage is brought to the beginning of the clock cycle as this would allow us to continuously fetch the next instruction. This is known as circuit retiming. Retiming changes the state representation for a system without changing its logical behavior and allows us to balance the delays between stages in the pipelined processor. We refer to this design as SEQ+.





SEQ+ hardware structure.

PIPE-

Next, we insert pipeline registers between the stages of SEQ+ and rearrange signals somewhat, yielding the PIPE– processor, where the "–" in the name signifies that this processor has somewhat less performance than our ultimate processor design.

The PIPE– uses nearly the same set of hardware units as our sequential design SEQ but with the pipeline registers separating the stages. These registers stop the signals from flowing into the next stage and affect the processing happening there.

- F

  Holds a predicted value of the program counter, as will be discussed shortly.

- D

  Sits between the fetch and decode stages. It holds information about the most recently fetched instruction for processing by the decode stage.

- E

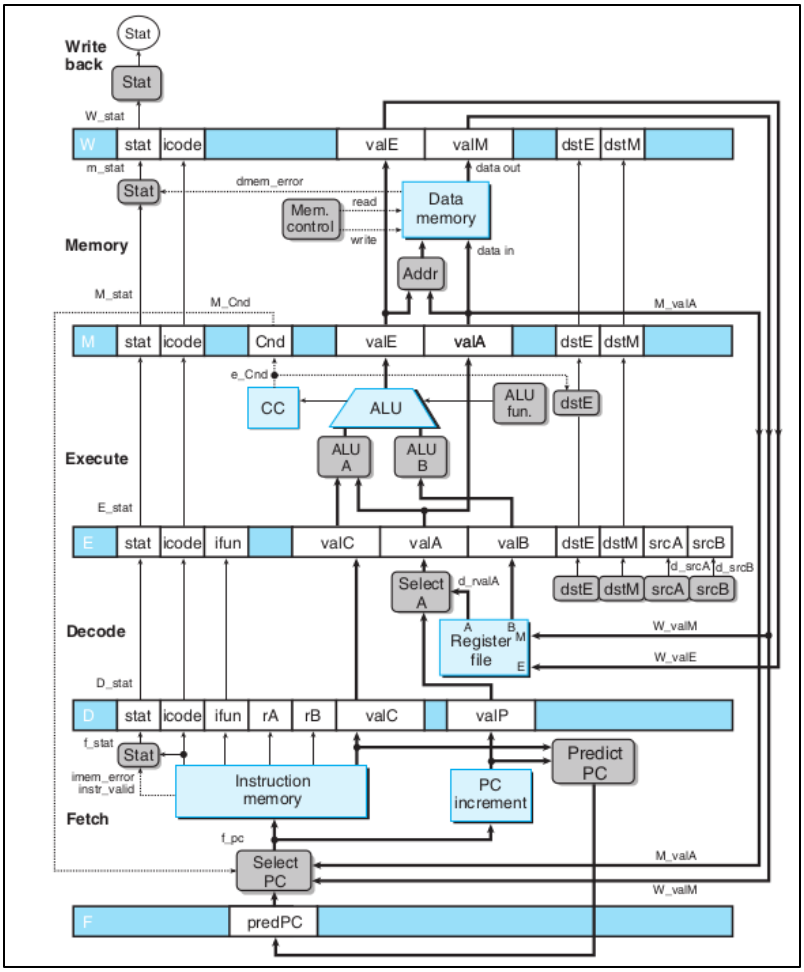
  Sits between the decode and execute stages. It holds information about the most recently decoded instruction and the values read from the register file for processing by the execute stage.

- M

  Sits between the execute and memory stages. It holds the results of the most recently executed instruction for processing by the memory stage. It also holds information about branch conditions and branch targets for processing conditional jumps.

- W

  Sits between the memory stage and the feedback paths that supply the computed results to the register file for writing and the return address to the PC selection logic when completing a ret instruction.



PIPE

In the pipelined design, there will be multiple versions of signals associated with the different instructions flowing through the system. We adopt a naming scheme where a signal stored in a pipeline register can be uniquely identified by prefixing its name with that of the pipe register written in uppercase. We also need to refer to some signals that have just been computed within a stage. These are labeled by prefixing the signal name with the first character of the stage name, written in lowercase.

Pipeline Hazards:

Introducing pipelining into a system with feedback can lead to problems when there are dependencies between successive instructions. These dependencies can take two forms: (1) data dependencies and (2) control hazards.

Avoiding Data Pipeline Hazards:

- *Avoiding Data Hazards by Stalling*

  One very general technique for avoiding hazards involves stalling, where the processor holds back one or more instructions in the pipeline until the hazard condition no longer holds. Stalling involves holding back one group of instructions in their stages while allowing other

instructions to continue flowing through the pipeline. The stages that should normally be processing would be injected with a bubble. A bubble is like a dynamically generated nop instruction—it does not cause any changes to the registers, the memory, the condition codes, or the program status.

- *Avoiding Data Hazards by Forwarding*

  The technique of passing a result value directly from one pipeline stage to an earlier one is commonly known as data forwarding. Data forwarding requires adding additional data connections and control logic to the basic hardware structure.

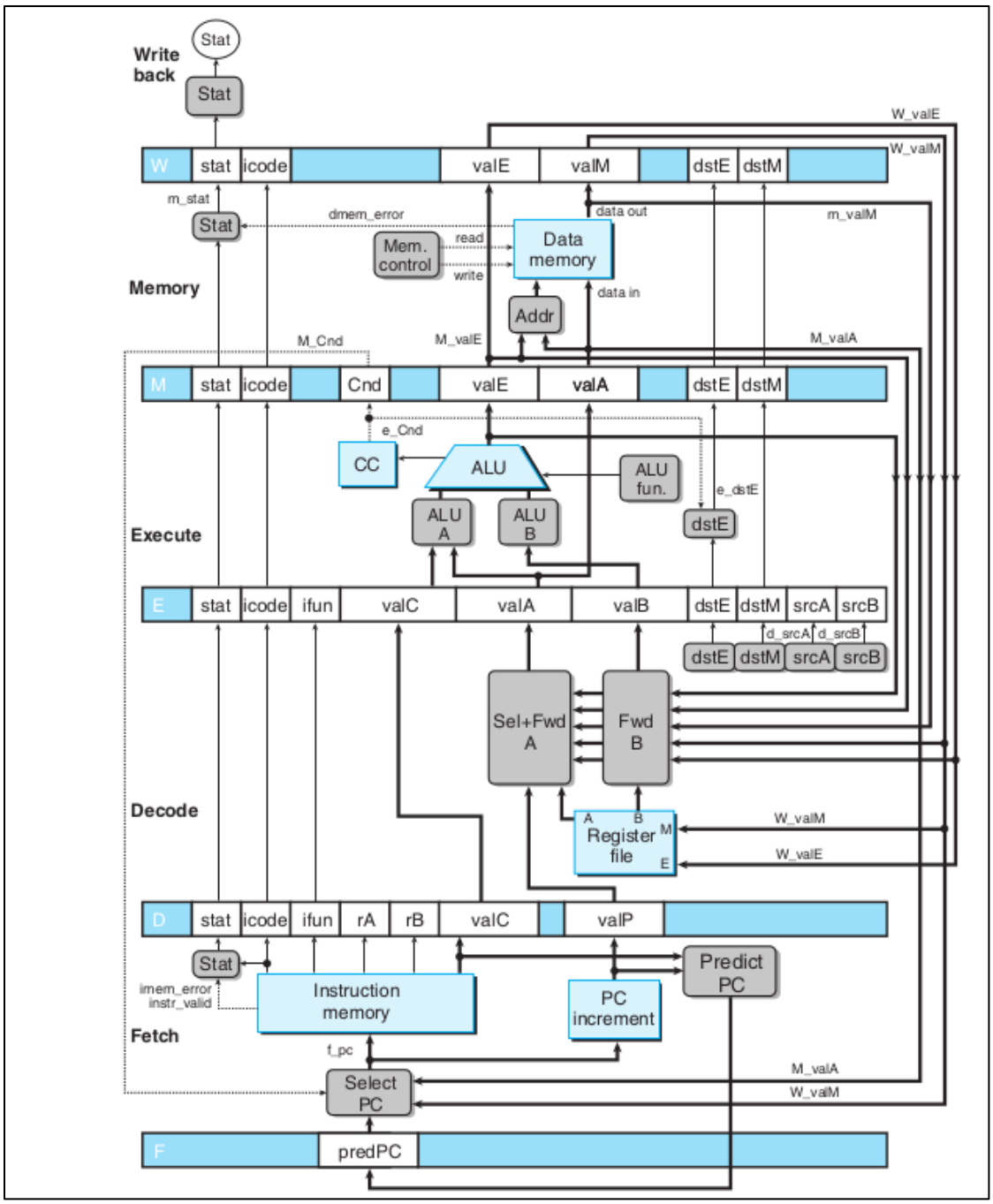
  - It can also use the ALU output (signal e_valE) for operand valA or valB.
  - It can use the value that has just been read from the data memory (signal m_valM) for operand valA or valB.
  - It can use the value in the memory stage (signal M_valE) for operand valA or valB.
  - It can use the value in the write-back stage (signal W_valE or signal W_valM) for operand valA or valB.

Avoiding Load/Use Data Hazards:

One class of data hazards cannot be handled purely by forwarding, because memory reads occur late in the pipeline. These are called load/use hazards and they occur when one instruction reads a value from memory for register while the next instruction needs this value as a source operand. We can avoid a load/use data hazard with a combination of stalling and forwarding. This requires modifications of the control logic.

Avoiding Control Hazards:

Control hazards arise when the processor cannot reliably determine the address of the next instruction based on the current instruction in the fetch stage. control hazards can only occur in our pipelined processor for ret and jump instructions. In case of ret, the processor is stalled for 3 clock cycles after the ret instruction. While in case of jump misprediction, the pipeline can simply cancel the two misfetched instructions by injecting bubbles into the decode and execute stages on the following cycle while also fetching the instruction following the jump instruction.

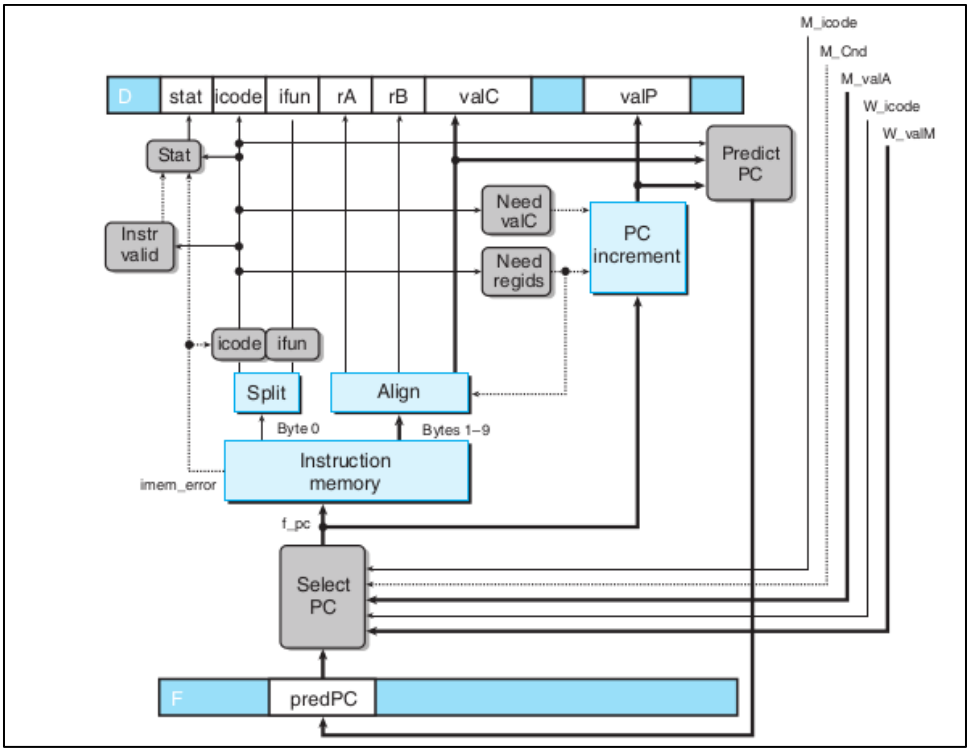


Hardware structure of PIPE

The different stages of the pipelined processor:

PC SELECTION AND FETCH

This stage selects a current value for the program counter and predicts the next PC value. The hardware units for reading the instruction from memory and for extracting the different instruction fields are the same as those we considered for SEQ.



Implementation and Working:

- The logic to obtain the values of icode, ifun, rA, rB, and valC are the same as in the fetch stage of sequential processor.
- The PC selection logic chooses between three program counter sources.
  - f_PC = M_valA if M_icode = IJXX and M_cnd = 0
  - f_PC = W_valM if W_icode == IRET
  - f_PC = F_predPC in all other cases
  - The PC prediction logic is as follows:
  - F_predPC = f_valC if icode is in {IJXX, ICALL}
  - F_predPC = f_valP in all other cases
- At positive edge all the values are stored into the pipelined register variables.

Source code:

```
// FETCH BLOCK

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

    input clk;
    input [63:0] F_predPC;

    input [3:0] M_icode;
    input M_cnd;
    input [63:0] M_valA;

    input [3:0] W_icode;
    input [63:0] W_valM;

    input F_stall, D_stall, D_bubble;

    output reg [63:0] f_predPC;

    output reg [3:0] D_icode;
    output reg [3:0] D_ifun;
    output reg [3:0] D_rA;
    output reg [3:0] D_rB;
    output reg [63:0] D_valC;
    output reg [63:0] D_valP;

    output reg [1:0] D_stat;  //AOK|HLT|ADR|INS

    reg [63:0] PC;

    reg [3:0] icode;
    reg [3:0] ifun;

    reg [3:0] rA;
    reg [3:0] rB;

    reg [63:0] valC;
    reg [63:0] valP;

    reg [1:0] stat;  //AOK|HLT|ADR|INS

    reg instr_valid;
    reg imem_error;

    reg [0:79] instruction; //Instruction encodings range between 1 and 10 bytes
    reg [7:0] memory[0:1023];//memory has 256-32 bits words => 1024-8 bits

    reg [0:7] opcode; //operation codes
    reg [0:7] regids; //register IDs
    initial
    begin
        rA = 4'hf;
        rB = 4'hf;
        valC = 64'd0;
        valP = 64'd0;
        instr_valid = 0;
        imem_error = 0;
        stat = 2'd0;
    end

    // instruction codes
    // Constant values used in HCL descriptions.
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
    always@(*)
    begin

        if(M_icode == IJXX && M_cnd == 0)
        begin
            PC = M_valA;
        end
        else if (W_icode == IRET)
        begin
            PC= W_valM;
        end
        else
        begin
            PC = F_predPC;
        end

    end

    always@(*)
    begin

        if(PC > 64'd1023)
        begin

            imem_error = 1; //invalid address
            stat = 2'd2;
        end
        else
        begin

            imem_error = 0;

            //Instruction encodings range between 1 and 10 bytes
            instruction = {memory[PC], memory[PC+64'd1], memory[PC+64'd2], memory[PC+64'd3], memory[PC+64'd4],
                           memory[PC+64'd5], memory[PC+64'd6], memory[PC+64'd7], memory[PC+64'd8], memory[PC+64'd9]};

            //An instruction consists of a 1-byte instruction specifier, 1-byte register specifier, and an 8-byte constant word.

            // icode:ifun = M1[PC]
            opcode = instruction[0:7];
            icode = opcode[0:3];
            ifun = opcode[4:7];
```

```
if(icode < 4'd0 || icode > 4'd11)
begin

    instr_valid = 1; //invalid instruction
    stat = 2'd3;
end
else
begin

    instr_valid = 0;

    //halt
    if(icode == IHALT)
    begin

        stat = 2'd1;

        //valP = PC+1
        valP = PC + 64'd1;
        f_predPC = valP;

    end
    //nop
    else if (icode == INOP)
    begin

        //valP = PC+1
        valP = PC + 64'd1;
        f_predPC = valP;

    end
    //rrmovq and  cmovXX
    else if (icode == IRRMOVQ)
    begin

        // rA:rB = M1[PC+1]
        regids = instruction[8:15];
        rA = regids[0:3];
        rB = regids[4:7];

        //valP <- PC+2
        valP = PC + 64'd2;
        f_predPC = valP;

    end
    //irmovq
    else if (icode == IIRMOVQ)
    begin

        // rA:rB = M1[PC+1]
        regids = instruction[8:15];
        rA = regids[0:3];
        rB = regids[4:7];

        //valC <- M8[PC+2]
        valC = instruction[16:79];

        //valP <- PC+10
        valP = PC + 64'd10;
        f_predPC = valP;
    end
    // rmmovq
    else if (icode == IRMMOVQ)
    begin

        // rA:rB = M1[PC+1]
        regids = instruction[8:15];
        rA = regids[0:3];
        rB = regids[4:7];

        //valC <- M8[PC+2]
        valC = instruction[16:79];

        //valP <- PC+10
        valP = PC + 64'd10;
        f_predPC = valP;
    end
    // mrmovq
    else if (icode == IMRMOVQ)
    begin

        // rA:rB = M1[PC+1]
        regids = instruction[8:15];
        rA = regids[0:3];
        rB = regids[4:7];

        //valC <- M8[PC+2]
        valC = instruction[16:79];

        //valP <- PC+10
        valP = PC + 64'd10;
        f_predPC = valP;
    end
```

```
                //Opq
                else if (icode == IOPQ)
                begin

                    // rA:rB = M1[PC+1]
                    regids = instruction[8:15];
                    rA = regids[0:3];
                    rB = regids[4:7];

                    //valP <- PC+2
                    valP = PC + 64'd2;
                    f_predPC = valP;

                end
                //jXX
                else if (icode == IJXX)
                begin

                    //valC <- M8[PC+1]
                    valC = instruction[8:71];

                    //valP <- PC+9
                    valP = PC + 64'd9;
                    f_predPC = valC;

                end
                //call
                else if (icode == ICALL)
                begin

                    //valC <- M8[PC+1]
                    valC = instruction[8:71];

                    //valP <- PC+9
                    valP = PC + 64'd9;
                    f_predPC = valC;


                end
                //ret
                else if (icode == IRET)
                begin

                    //valP <- PC+1
                    valP = PC + 64'd1;
                    f_predPC = valP;

                end
                //pushq
                else if (icode == IPUSHQ)
                begin

                    // rA:rB = M1[PC+1]
                    regids = instruction[8:15];
                    rA = regids[0:3];
                    rB = regids[4:7];

                    //valP <- PC+2
                    valP = PC + 64'd2;
                    f_predPC = valP;

                end
                //popq
                else if (icode == IPOPQ)
                begin

                    // rA:rB = M1[PC+1]
                    regids = instruction[8:15];
                    rA = regids[0:3];
                    rB = regids[4:7];

                    //valP <- PC+2
                    valP = PC + 64'd2;
                    f_predPC = valP;

                end

            end

        end

    end

    always@(posedge clk)
    begin

        if (D_stall == 1)
        begin
        end

        else if(D_bubble == 1)
        begin
            D_icode <= 4'b0001; //nop
            D_ifun <= 4'b0000;
            D_rA <= 4'b0000;
            D_rB <= 4'b0000;
            D_valC <= 64'b0;
            D_valP <= 64'b0;
            D_stat <= 2'd0;
        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
```

Test bench:

```
`timescale 1ns / 10ps

`include "fetch.v"

module fetch_tb();

    reg clk;
    reg [63:0] F_predPC;

    reg [3:0] M_icode;
    reg M_cnd;
    reg [63:0] M_valA;

    reg [3:0] W_icode;
    reg [63:0] W_valM;

    reg F_stall, D_stall, D_bubble;

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;

    wire [3:0] D_rA;
    wire [3:0] D_rB;

    wire [63:0] D_valC;
    wire [63:0] D_valP;

    wire [1:0] D_stat;  //AOK|HLT|ADR|INS


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


  always @(D_icode)
  begin
    if(D_icode==0)
      $finish;
  end

  always @(posedge clk) F_predPC <= f_predPC;

  always #10 clk = ~clk;

  initial
  begin

    $dumpfile("fetch_tb.vcd");
    $dumpvars(0, fetch_tb);

    F_predPC = 64'd32;
    clk = 0;
    F_stall = 0;
    D_stall = 0;
    D_bubble = 0;

  end

  initial
  begin
    $monitor($time, "\tclk=%d\n\t\t\tF Reg:\t\tF_predPC = %g\n\t\t\tfetch:\t\tf_predPC = %g\n\t\t\tD Reg:\t\t
    D_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n",clk,F_predPC,f_predPC,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,);
  end

endmodule
```

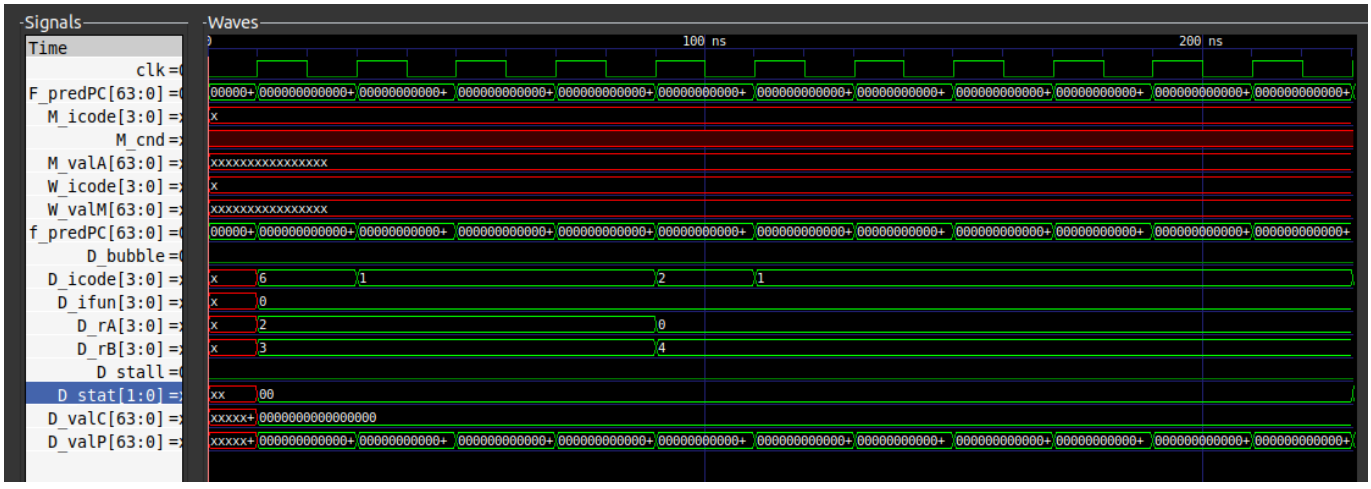
Instructions given to fetch module:

```
addq %rdx, %rbx

nop

rrmovq %rax, %rsp

nop

nop

Halt
```

Output of Test Bench:

```
 10    clk=1
       F Reg:          F_predPC = 34
       fetch:          f_predPC = 35
       D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 34 D_stat = 0

 20    clk=0
       F Reg:          F_predPC = 34
       fetch:          f_predPC = 35
       D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 34 D_stat = 0

 30    clk=1
       F Reg:          F_predPC = 35
       fetch:          f_predPC = 37
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 35 D_stat = 0

 40    clk=0
       F Reg:          F_predPC = 35
       fetch:          f_predPC = 37
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 35 D_stat = 0

 50    clk=1
       F Reg:          F_predPC = 37
       fetch:          f_predPC = 38
       D Reg:          D_icode = 0010 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 37 D_stat = 0

 60    clk=0
       F Reg:          F_predPC = 37
       fetch:          f_predPC = 38
       D Reg:          D_icode = 0010 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 37 D_stat = 0

 70    clk=1
       F Reg:          F_predPC = 38
       fetch:          f_predPC = 39
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 38 D_stat = 0

 80    clk=0
       F Reg:          F_predPC = 38
       fetch:          f_predPC = 39
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 38 D_stat = 0

 90    clk=1
       F Reg:          F_predPC = 39
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 39 D_stat = 0

100    clk=0
       F Reg:          F_predPC = 39
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 39 D_stat = 0

110    clk=1
       F Reg:          F_predPC = 40
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 40 D_stat = 1
```

Gtkwave output:



Command to run the code:

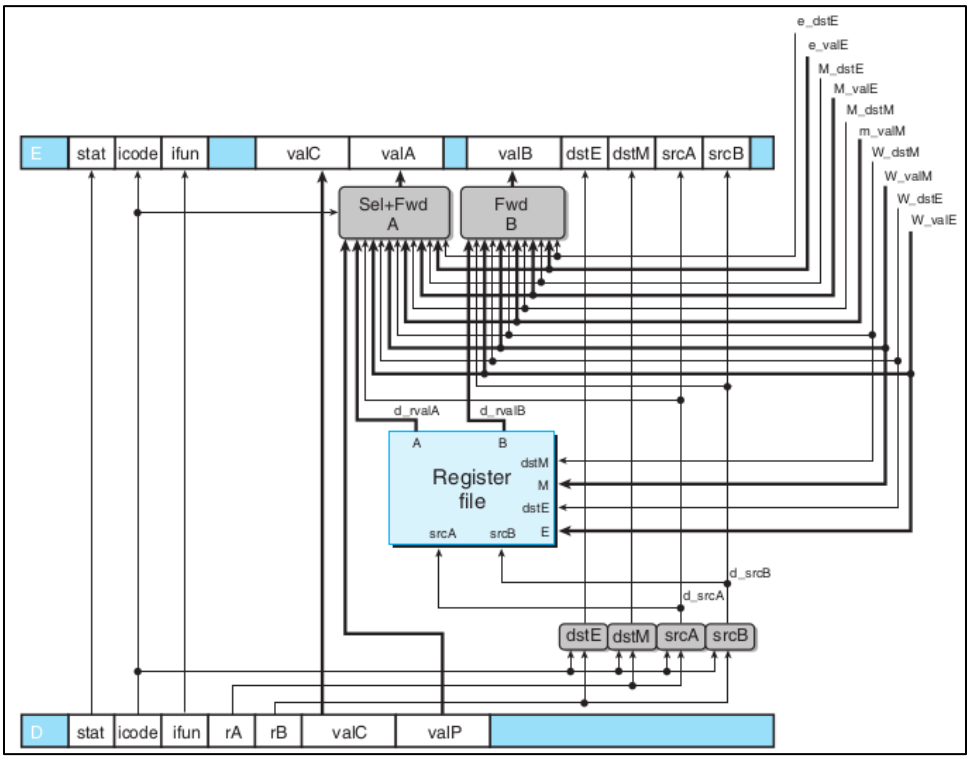
iverilog –o fetch fetch_tb.v

vvp fetch

DECODE

The implementation of this stage is like that of the decode stage in the sequential processor. Most of the complexity of this stage is associated with the forwarding logic.

Like in SEQ decode and write back can be combined as they both access the register file.



Implementation and Working:

- The forwarding logic is as follows:
- d_valA = e_valE if d_srcA = e_dstE (Forward valE from execute)
- d_valA = m_valM if d_srcA = M_dstM (Forward valM from memory)
- d_valA = M_valE if d_srcA = M_dstE (Forward valE from memory)
- d_valA = W_valE if d_srcA = W_dstE (Forward valM from write back)
- d_valA = W_valE if d_srcA = W_dstE (Forward valE from write back)
- At positive edge all the values are stored into the pipelined register variables.

Source code:

```
// DECODE BLOCK

module decode(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE,
              M_dstM, M_dstE, W_dstM, W_dstE, e_valE, m_valM, M_valE, W_valM, W_valE,
              E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat, srcA, srcB, valA, valB);

    input clk;

    input [3:0] D_icode;
    input [3:0] D_ifun;

    input [3:0] D_rA;
    input [3:0] D_rB;

    input [63:0] D_valC;
    input [63:0] D_valP;

    input [1:0] D_stat;  //AOK|HLT|ADR|INS

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

    input E_bubble;

    output reg [3:0] E_icode;
    output reg [3:0] E_ifun;

    output reg [63:0] E_valA;
    output reg [63:0] E_valB;
    output reg [63:0] E_valC;

    output reg [3:0] E_dstE;
    output reg [3:0] E_dstM;
    output reg [3:0] E_srcA;
    output reg [3:0] E_srcB;

    output reg [1:0] E_stat; //AOK|HLT|ADR|INS

    output reg [63:0] valA;
    output reg [63:0] valB;
    output reg [3:0] srcA, srcB;
    reg [3:0] dstE, dstM;

    reg [63:0] Reg_File[0:14];
```

```
// instruction codes
// Constant values used in HCL descriptions.
parameter IHALT   = 4'd0;
parameter INOP    = 4'd1;
parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
parameter IIRMOVQ = 4'd3;
parameter IRMMOVQ = 4'd4;
parameter IMRMOVQ = 4'd5;
parameter IOPQ    = 4'd6;
parameter IJXX    = 4'd7;
parameter ICALL   = 4'd8;
parameter IRET    = 4'd9;
parameter IPUSHQ  = 4'd10;
parameter IPOPQ   = 4'd11;

// registers
parameter Register_rax = 4'd0;
parameter Register_rcx = 4'd1;
parameter Register_rdx = 4'd2;
parameter Register_rbx = 4'd3;
parameter RRSP         = 4'd4;
parameter Register_rbp = 4'd5;
parameter Register_rsi = 4'd6;
parameter Register_rdi = 4'd7;
parameter Register_r8  = 4'd8;
parameter Register_r9  = 4'd9;
parameter Register_r10 = 4'd10;
parameter Register_r11 = 4'd11;
parameter Register_r12 = 4'd12;
parameter Register_r13 = 4'd13;
parameter Register_r14 = 4'd14;
parameter RNONE        = 4'd15;

always@(posedge clk)
begin

    $readmemh("reg_file.txt", Reg_File);

end
```

```
always@(*)
begin
        if (D_icode == IRRMOVQ)
        begin
            srcA = D_rA;
            srcB = RNONE;
            dstE = D_rB;
            dstM = RNONE;
            // valA ← R[rA]
            valA = Reg_File[D_rA];
        end
        else if(D_icode == IIRMOVQ)
        begin
            dstE = D_rB;
            dstM = RNONE;
            srcA = RNONE;
            srcB = RNONE;
        end
        // rmmovq
        else if (D_icode == IRMMOVQ)
        begin
            srcA = D_rA;
            srcB = RNONE;
            dstE = D_rB;
            dstM = RNONE;

            // valA ← R[rA]
            valA = Reg_File[D_rA];

            // valB ← R[rB]
            valB = Reg_File[D_rB];
        end
        // mrmovq
        else if (D_icode == IMRMOVQ)
        begin
            srcA = RNONE;
            srcB = D_rB;
            dstM = D_rA;
            dstE = RNONE;

            // valB ← R[rB]
            valB = Reg_File[D_rB];
        end
          //Opq
          else if (D_icode == IOPQ)
          begin
              srcA = D_rA;
              srcB = D_rB;
              dstE = D_rB;
              dstM = RNONE;

              // valA ← R[rA]
              valA = Reg_File[D_rA];

              // valB ← R[rB]
              valB = Reg_File[D_rB];
          end
          else if (D_icode == ICALL)
          begin
              srcA = RNONE;
              srcB = RRSP;
              dstE = RRSP;
              dstM = RNONE;
              //valB ← R[ %rsp ]
              valB = Reg_File[RRSP];
          end
          //ret
          else if (D_icode == IRET)
          begin
              srcA = RRSP;
              srcB = RRSP;
              dstE = RRSP;
              dstM = RNONE;
              // valA ← R[ %rsp ]
              valA = Reg_File[RRSP];
              // valB ← R[ %rsp ]
              valB = Reg_File[RRSP];
          end
          //pushq
          else if (D_icode == IPUSHQ)
          begin
              srcA = D_rA;
              srcB = RRSP;
              dstE = RRSP;
              dstM = RNONE;

              // valA ← R[rA]
              valA = Reg_File[D_rA];
              // valB ← R[ %rsp ]
              valB = Reg_File[RRSP];
          end
```

```
            //popq
            else if (D_icode == IPOPQ)
            begin
                srcA = RRSP;
                srcB = RRSP;
                dstE = RRSP;
                dstM = D_rA;
                // valA ← R[ %rsp ]
                valA = Reg_File[RRSP];
                // valB ← R[ %rsp ]
                valB = Reg_File[RRSP];
            end
            else
            begin
                srcA = RNONE;
                srcB = RNONE;
                dstE = RNONE;
                dstM = RNONE;
            end
    end
    always@(*)
    begin
        //Forwarding logic for valA
            if(D_icode == ICALL || D_icode == IJXX)
            begin
                valA = D_valP;
            end
            else if (srcA == e_dstE && srcA != RNONE)
            begin
                valA = e_valE;
            end
            else if (srcA == M_dstM && srcA != RNONE)
            begin
                valA = m_valM;
            end
            else if (srcA == M_dstE && srcA != RNONE)
            begin
                valA = M_valE;
            end
            else if (srcA == W_dstM && srcA != RNONE)
            begin
                valA = W_valM;
            end
            else if (srcA == W_dstE && srcA != RNONE)
            begin
                // $display("hi");
                valA = W_valE;
            end
    always@(*)begin//Forwarding logic for valB
            if(srcB == e_dstE && srcB != RNONE)
            begin
                valB = e_valE;
            end
            else if(srcB == M_dstM && srcB != RNONE)
            begin
                valB = m_valM;
            end
            else if(srcB == M_dstE && srcB != RNONE)
            begin
                valB = M_valE;
            end
            else if(srcB == W_dstM && srcB != RNONE)begin
                valB = W_valM;
            end
            else if(srcB == W_dstE && srcB != RNONE)begin
                valB = W_valE;
            end
    end
    always@(posedge clk) begin
        if(E_bubble == 1)begin
            E_icode <= 4'b0001; //nop
            E_ifun <= 4'b0000;
            E_valA <= 4'b0000;
            E_valB <= 4'b0000;
            E_valC <= 4'b0000;
            E_srcA <= RNONE;
            E_srcB <= RNONE;
            E_dstE <= RNONE;
            E_dstM <= RNONE;
            E_stat <= 2'd0;
        end
        else begin
            E_icode <= D_icode;
            E_ifun <= D_ifun;
            E_valA <= valA;
            E_valB <= valB;
            E_valC <= D_valC;
            E_srcA <= srcA;
            E_srcB <= srcB;
            E_dstE <= dstE;
            E_dstM <= dstM;
            E_stat <= D_stat;
        end
    end

endmodule
```

Test bench:

```
`timescale 1ns / 10ps

`include "fetch.v"
`include "decode.v"

module decode_tb();

    reg clk;
    reg [63:0] F_predPC;

    reg [3:0] M_icode;
    reg M_cnd;
    reg [63:0] M_valA;

    reg [3:0] W_icode;

    reg F_stall, D_stall, D_bubble;

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

    reg E_bubble;

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;
    wire [3:0] D_rA;
    wire [3:0] D_rB;
    wire [63:0] D_valC;
    wire [63:0] D_valP;
    wire [1:0] D_stat;  //AOK|HLT|ADR|INS

    wire [63:0] d_valA, d_valB;
    wire [3:0] d_srcA, d_srcB;

    wire [3:0] E_icode;
    wire [3:0] E_ifun;
    wire [63:0] E_valA;
    wire [63:0] E_valB;
    wire [63:0] E_valC;
    wire [3:0] E_dstE;
    wire [3:0] E_dstM;
    wire [3:0] E_srcA;
    wire [3:0] E_srcB;
    wire [1:0] E_stat;  //AOK|HLT|ADR|INS

    fetch fetch_(clk, F_predPC, M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble, f_predPC, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat);
    decode decode_(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE,
                   e_valE, m_valM, M_valE, W_valM, W_valE, E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat,
                   d_srcA, d_srcB, d_valA, d_valB);

    always @(E_icode)
    begin
        if(E_icode==0)
        $finish;
    end

  always @(posedge clk) F_predPC <= f_predPC;

  always #10 clk = ~clk;

  initial
  begin

    $dumpfile("decode_tb.vcd");
    $dumpvars(0, decode_tb);

    F_predPC = 64'd32;
    clk = 0;
    F_stall = 0;
    D_stall = 0;
    D_bubble = 0;
    E_bubble = 0;

  end

  initial
  begin
    $monitor($time, "\tclk=%d\n\t\t\tF Reg:\t\tF_predPC = %g\n\t\t\tfetch:\t\tf_predPC = %g\n\t\t\t
            D Reg:\t\tD_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n\t\t\tdecode:\t\td_valA = %g d_valB = %g\n\t\t\t
            E Reg:\t\tE_icode = %b E_ifun = %b E_valA = %g E_valB = %g E_valC = %g E_dstE = %b E_dstM = %b E_srcA = %g E_srcB = %g E_stat = %g\n",
            clk,F_predPC,f_predPC,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,d_valA,d_valB,E_icode,E_ifun,E_valA,E_valB,E_valC,E_dstE,E_dstM,E_srcA,E_srcB,E_stat);
  end


endmodule
```

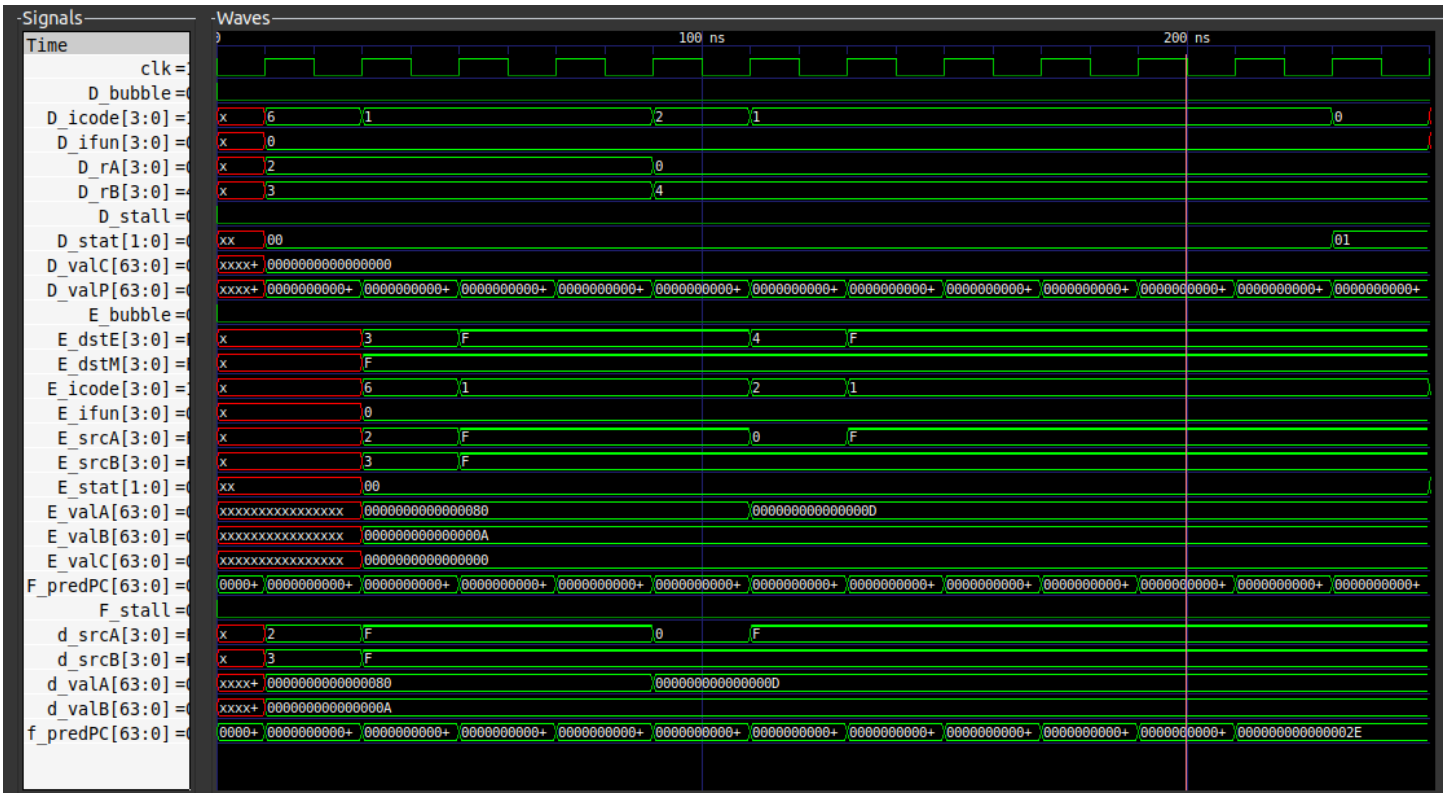
Instructions given to fetch module:

```
addq %rdx, %rbx

nop

rrmovq %rax, %rsp

nop

nop

Halt
```

Output of Test Bench:

```
 0    clk=0
      F Reg:          F_predPC = 32
      fetch:          f_predPC = 34
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = xxxx D_rB = xxxx D_valC = 0 D_valP = 0 D_stat = 0
      decode:         d_valA = 0 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0

 10   clk=1
      F Reg:          F_predPC = 34
      fetch:          f_predPC = 35
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 34 D_stat = 0
      decode:         d_valA = 128 d_valB = 10
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0

 20   clk=0
      F Reg:          F_predPC = 34
      fetch:          f_predPC = 35
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 34 D_stat = 0
      decode:         d_valA = 128 d_valB = 10
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0

 30   clk=1
      F Reg:          F_predPC = 35
      fetch:          f_predPC = 37
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 35 D_stat = 0
      decode:         d_valA = 128 d_valB = 10
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 128 E_valB = 10 E_valC = 0 E_dstE = 0011 E_dstM = 1111 E_srcA = 2 E_srcB = 3 E_stat = 0

 40   clk=0
      F Reg:          F_predPC = 35
      fetch:          f_predPC = 37
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0010 D_rB = 0011 D_valC = 0 D_valP = 35 D_stat = 0
      decode:         d_valA = 128 d_valB = 10
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 128 E_valB = 10 E_valC = 0 E_dstE = 0011 E_dstM = 1111 E_srcA = 2 E_srcB = 3 E_stat = 0

 50   clk=1
      F Reg:          F_predPC = 37
      fetch:          f_predPC = 38
      D Reg:          D_icode = 0010 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 37 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 128 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

 60   clk=0
      F Reg:          F_predPC = 37
      fetch:          f_predPC = 38
      D Reg:          D_icode = 0010 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 37 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 128 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

 70   clk=1
      F Reg:          F_predPC = 38
      fetch:          f_predPC = 39
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 38 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0010 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 0100 E_dstM = 1111 E_srcA = 0 E_srcB = 15 E_stat = 0

 80   clk=0
      F Reg:          F_predPC = 38
      fetch:          f_predPC = 39
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 38 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0010 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 0100 E_dstM = 1111 E_srcA = 0 E_srcB = 15 E_stat = 0

 90   clk=1
      F Reg:          F_predPC = 39
      fetch:          f_predPC = 40
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 39 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

100   clk=0
      F Reg:          F_predPC = 39
      fetch:          f_predPC = 40
      D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 39 D_stat = 0
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

110   clk=1
      F Reg:          F_predPC = 40
      fetch:          f_predPC = 40
      D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 40 D_stat = 1
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

120   clk=0
      F Reg:          F_predPC = 40
      fetch:          f_predPC = 40
      D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 40 D_stat = 1
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0

130   clk=1
      F Reg:          F_predPC = 40
      fetch:          f_predPC = 40
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0100 D_valC = 0 D_valP = 40 D_stat = 1
      decode:         d_valA = 13 d_valB = 10
      E Reg:          E_icode = 0000 E_ifun = 0000 E_valA = 13 E_valB = 10 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
```

Gtkwave output:



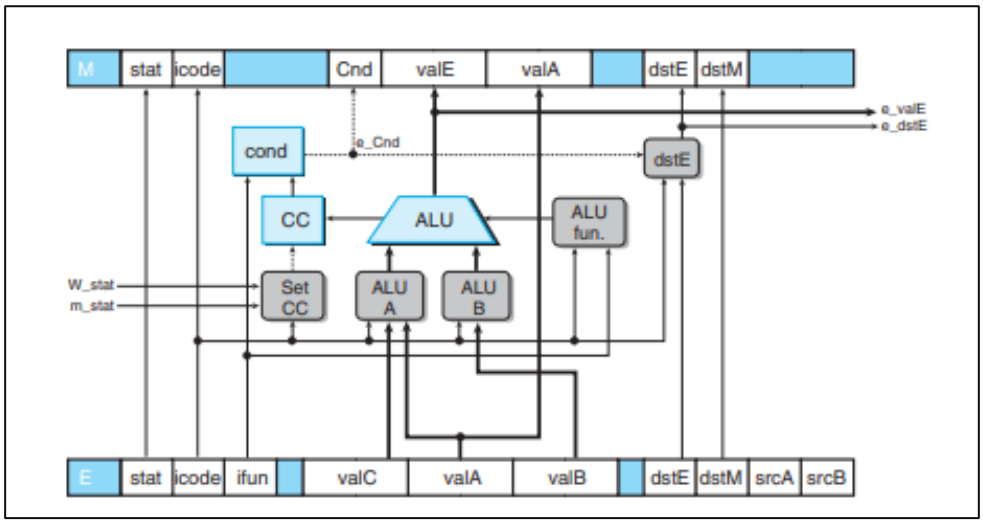
Command to run the code:

iverilog –o decode decode_tb.v

vvp decode

EXECUTE

The implementation of this stage is almost the same as that of the execute stage in SEQ.



Implementation and Working:

1. The values of e_valE and e_cnd is set in the similar way as done in the sequential implementation of processor.
2. The values from the execute register and the values calculated in this stage are stored in the memory register at the positive edge of the clock.

```
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;
```

Source code:

```
`include "alu.v"

module execute(clk, E_stat, E_icode, E_ifun, E_valC, E_valA, E_valB, E_dstE, E_dstM, set_cc, M_stat, M_icode, M_cnd, M_valE,
               M_valA, M_dstE, M_dstM, e_valE, e_dstE, e_cnd);

    input clk;
    input[1:0] E_stat;
    input [3:0] E_icode, E_ifun;
    input [63:0] E_valC, E_valA, E_valB;
    input [3:0] E_dstE, E_dstM;
    input set_cc;

    output reg [1:0] M_stat;
    output reg [3:0] M_icode;
    output reg M_cnd;
    output reg [63:0] M_valE, M_valA, e_valE;
    output reg [3:0] M_dstE, M_dstM, e_dstE;
    output reg e_cnd;

    reg [3:0] icode;
    reg [3:0] ifun;
    reg [63:0] valA;
    reg [63:0] valB;
    reg [63:0] valC;
    reg signed [63:0] valE;
    reg signed cnd;

    // Condition Code [ZF, SF, OF]
    reg [2:0] CC;
    reg zf;
    reg sf;
    reg of;

    // reg [63:0] aluA, aluB;
    wire signed [63:0]ans;
    wire [2:0] conCode;
    reg signed [63:0] aluA, aluB;
    reg [1:0] op;
```

```
// Instruction codes
parameter IHALT   = 4'd0;
parameter INOP    = 4'd1;
parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
parameter IIRMOVQ = 4'd3;
parameter IRMMOVQ = 4'd4;
parameter IMRMOVQ = 4'd5;
parameter IOPQ    = 4'd6;
parameter IJXX    = 4'd7;
parameter ICALL   = 4'd8;
parameter IRET    = 4'd9;
parameter IPUSHQ  = 4'd10;
parameter IPOPQ   = 4'd11;

//Intialisation
initial
begin
    valE = 64'd0;
    cnd = 1'b0;
    CC = 3'd0;
    zf = 0;
    sf = 0;
    of = 0;
end

always@(*)
begin
    // e_valE = 63'd0;
    // e_cnd = 1'b1;
    e_dstE = E_dstE;
end
```

```
always@(*)
begin
    valA = E_valA;
    valB = E_valB;
    valC = E_valC;
    icode = E_icode;
    ifun = E_ifun;
end

always@(*)
begin
    e_valE = valE;
    e_cnd = cnd;
end

always@(*)
begin
    zf = CC[2];
    sf = CC[1];
    of = CC[0];
end

alu_ uut(aluA, aluB, op, ans, conCode);
```

```
always @(*)
begin
    if(icode == IRRMOVQ)
    begin
        valE = valA;
        if(ifun == 4'd0)
        begin
            cnd = 1;
        end
        else if(ifun == 4'd1)
        begin
            cnd = ((sf^of)| zf);
        end
        else if(ifun == 4'd2)
        begin
            cnd = (sf^of);
        end
        else if(ifun == 4'd3)
        begin
            cnd = zf;
        end
        else if(ifun == 4'd4)
        begin
            cnd = ~zf;
        end
        else if(ifun == 4'd5)
        begin
            cnd = ~(sf^of);
        end
        else if(ifun == 4'd6)
        begin
            cnd = ((~(sf^of))&~zf);
        end
        e_dstE = e_cnd ? E_dstE : 4'd15;
    end
```

```
    else if(icode == IIRMOVQ)
    begin
        valE = valC;
    end
    else if(icode == IRMMOVQ)
    begin
        valE = valC+valB;
    end
    else if(icode == IMRMOVQ)
    begin
        valE = valC+valB;
    end
    else if(icode == IOPQ)
    begin
        aluA = valB;
        aluB = valA;
        if(ifun==4'b0000)
            op = 2'b00;
        else if(ifun==4'b0001)
            op = 2'b01;
        else if(ifun==4'b0010)
            op = 2'b10;
        else
            op = 2'b11;
        // assign ans_ = ans;
        valE = ans;
        // assign CC_ = conCode;
        if(set_cc==1)
            CC = conCode;
    end
```

```
    else if(icode== IJXX)
    begin
        if(ifun == 4'd0)
        begin
            cnd = 1;
        end
        else if(ifun == 4'd1)
        begin
            cnd = ((sf^of)| zf);
        end
        else if(ifun == 4'd2)
        begin
            cnd = (sf^of);
        end
        else if(ifun == 4'd3)
        begin
            cnd = zf;
        end
        else if(ifun == 4'd4)
        begin
            cnd = ~zf;
        end
        else if(ifun == 4'd5)
        begin
            cnd = ~(sf^of);
        end
        else if(ifun == 4'd6)
        begin
            cnd = ((~(sf^of))&~zf);
        end
        e_dstE = e_cnd ? E_dstE : 4'd15;
    end
        else if(icode == ICALL)
        begin
            valE = -64'd1+valB;
        end
        else if(icode == IRET)
        begin
            valE = 64'd1+valB;
        end
        else if(icode == IPUSHQ)
        begin
            valE = -64'd1+valB;
        end
        else if(icode == IPOPQ)
        begin
            valE = 64'd1+valB;
        end
    end

    always@(posedge clk)
    // always@(*)
    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
```

Test bench:

```
`timescale 1ns/10ps

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

module execute_tb();

    reg clk;
    reg [63:0] F_predPC;

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

    wire [3:0] M_icode;
    wire M_cnd;
    wire [63:0] M_valA;


    wire [3:0] W_icode;

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

    // reg E_bubble;

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;

    wire [3:0] D_rA;
    wire [3:0] D_rB;

    wire [63:0] D_valC;
    wire [63:0] D_valP;
```

```
  wire [1:0] D_stat;  //AOK|HLT|ADR|INS

  wire [3:0] E_icode;
  wire [3:0] E_ifun;

  wire [63:0] E_valA;
  wire [63:0] E_valB;
  wire [63:0] E_valC;

  wire [3:0] E_dstE;
  wire [3:0] E_dstM;
  wire [3:0] E_srcA;
  wire [3:0] E_srcB;

  wire [1:0] E_stat;  //AOK|HLT|ADR|INS


  wire[0:3] M_stat;
  wire e_cnd;

  fetch fetch_(clk, F_predPC, M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble, f_predPC, D_icode, D_ifun,
                D_rA, D_rB, D_valC, D_valP, D_stat);
  decode decode_(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE, e_valE, m_valM, M_valE, W_valM,
                 W_valE, E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat);
  execute execute_(clk, E_stat, E_icode, E_ifun, E_valC, E_valA, E_valB, E_dstE, E_dstM, set_cc, M_stat, M_icode, M_cnd, M_valE, M_valA, M_dstE,
                M_dstM, e_valE, e_dstE, e_cnd);

  always @(M_icode)
  begin
      if(M_icode==0)
      $finish;
  end

always @(posedge clk) F_predPC <= f_predPC;

always #10 clk = ~clk;

initial
begin

  $dumpfile("execute_tb.vcd");
  $dumpvars(0, execute_tb);

  F_predPC = 64'd32;
  clk = 0;
  F_stall = 0;
  D_stall = 0;
  D_bubble = 0;
  E_bubble = 0;

end

initial
begin
  $monitor($time, "\tclk=%d\tF_predPC = %g\tf_predPC = %g\n\t\t\tD_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n
  \t\t\tE_icode = %b E_ifun = %b E_valA = %g E_valB = %g E_valC = %g E_stat = %g\n\t\t\tM_icode = %b M_cnd = %b M_valA = %g M_valE = %g M_stat = %g\n"
  ,clk,F_predPC,f_predPC, D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,E_icode,E_ifun,E_valA,E_valB,E_valC, E_stat,M_icode,M_cnd,M_valA,M_valE,M_stat);
end

endmodule
```

Instructions given to execute:

```
irmovq $0x0, %rax
irmovq $0x10, %rdx
irmovq $0xc, %rbx
jmp check
check:
        addq %rax, %rbx
        je rbxres
        addq %rax, %rdx
        je rdxres
        jmp loop2
rbxres:
rdxres:
Loop2:
        halt
```

Output of Test Bench:

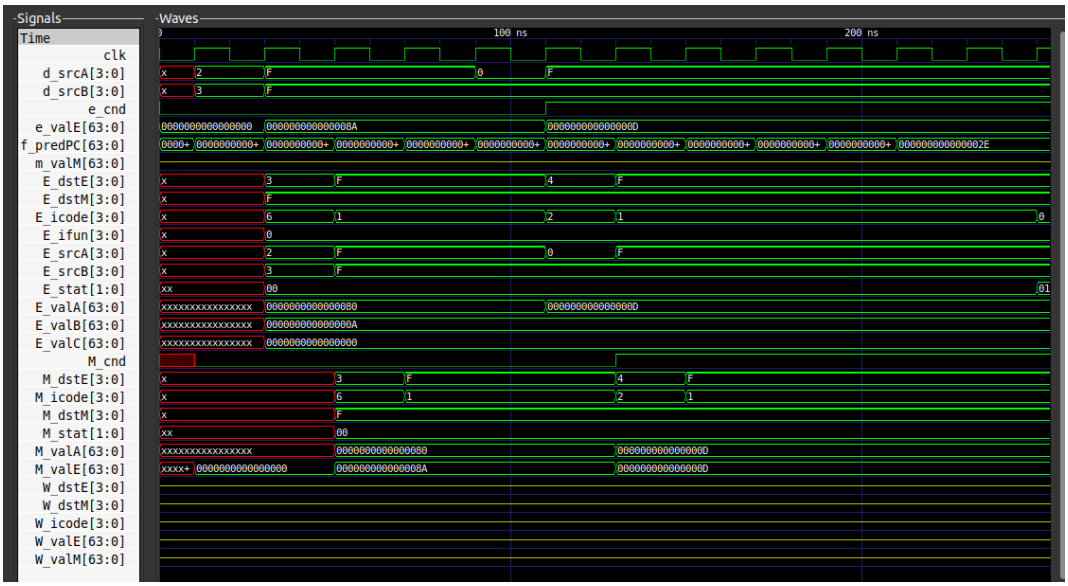
```
  0    clk=0   F_predPC = 0    f_predPC = 10
       D_icode = xxxx D_ifun = xxxx D_rA = xxxx D_rB = xxxx D_valC = 0 D_valP = 0 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_stat = 0
       M_icode = xxxx M_cnd = x M_valA = 0 M_valE = 0 M_stat = 0

 10    clk=1   F_predPC = 10   f_predPC = 20
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 1 D_valP = 10 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_stat = 0

 20    clk=0   F_predPC = 10   f_predPC = 20
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 1 D_valP = 10 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_stat = 0

 30    clk=1   F_predPC = 20   f_predPC = 30
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 16 D_valP = 20 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_stat = 0

 40    clk=0   F_predPC = 20   f_predPC = 30
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 16 D_valP = 20 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_stat = 0

 50    clk=1   F_predPC = 30   f_predPC = 39
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 12 D_valP = 30 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 16 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1 M_stat = 0

 60    clk=0   F_predPC = 30   f_predPC = 39
       D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 12 D_valP = 30 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 16 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1 M_stat = 0

 70    clk=1   F_predPC = 39   f_predPC = 41
       D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 39 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 12 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 16 M_stat = 0

 80    clk=0   F_predPC = 39   f_predPC = 41
       D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 39 D_stat = 0
       E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 12 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 16 M_stat = 0
```

```
 90    clk=1   F_predPC = 41   f_predPC = 122
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 41 D_stat = 0
       E_icode = 0111 E_ifun = 0000 E_valA = 39 E_valB = 0 E_valC = 39 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 12 M_stat = 0

100    clk=0   F_predPC = 41   f_predPC = 122
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 41 D_stat = 0
       E_icode = 0111 E_ifun = 0000 E_valA = 39 E_valB = 0 E_valC = 39 E_stat = 0
       M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 12 M_stat = 0

110    clk=1   F_predPC = 122  f_predPC = 122
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 12 E_valC = 39 E_stat = 0
       M_icode = 0111 M_cnd = 1 M_valA = 39 M_valE = 12 M_stat = 0

120    clk=0   F_predPC = 122  f_predPC = 122
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 12 E_valC = 39 E_stat = 0
       M_icode = 0111 M_cnd = 1 M_valA = 39 M_valE = 12 M_stat = 0

130    clk=1   F_predPC = 122  f_predPC = 122
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = 0110 M_cnd = 1 M_valA = 1 M_valE = 13 M_stat = 0

140    clk=0   F_predPC = 122  f_predPC = 122
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = 0110 M_cnd = 1 M_valA = 1 M_valE = 13 M_stat = 0

150    clk=1   F_predPC = 122  f_predPC = 52
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0

160    clk=0   F_predPC = 122  f_predPC = 52
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0

170    clk=1   F_predPC = 52   f_predPC = 125
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0
```

```
180    clk=0   F_predPC = 52   f_predPC = 125
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0

190    clk=1   F_predPC = 125  f_predPC = 125
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 17 E_valC = 122 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0

200    clk=0   F_predPC = 125  f_predPC = 125
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 17 E_valC = 122 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_stat = 0

210    clk=1   F_predPC = 125  f_predPC = 125
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 18 M_stat = 0

220    clk=0   F_predPC = 125  f_predPC = 125
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 18 M_stat = 0

230    clk=1   F_predPC = 125  f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0

240    clk=0   F_predPC = 125  f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0

250    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0

260    clk=0   F_predPC = 62   f_predPC = 62
       D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0
```

```
270    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = 0000 E_ifun = 0000 E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 1
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0

280    clk=0   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = 0000 E_ifun = 0000 E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 1
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 0

290    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_stat = 1
       M_icode = 0000 M_cnd = 0 M_valA = 61 M_valE = 18 M_stat = 1
```

Gtkwave output:



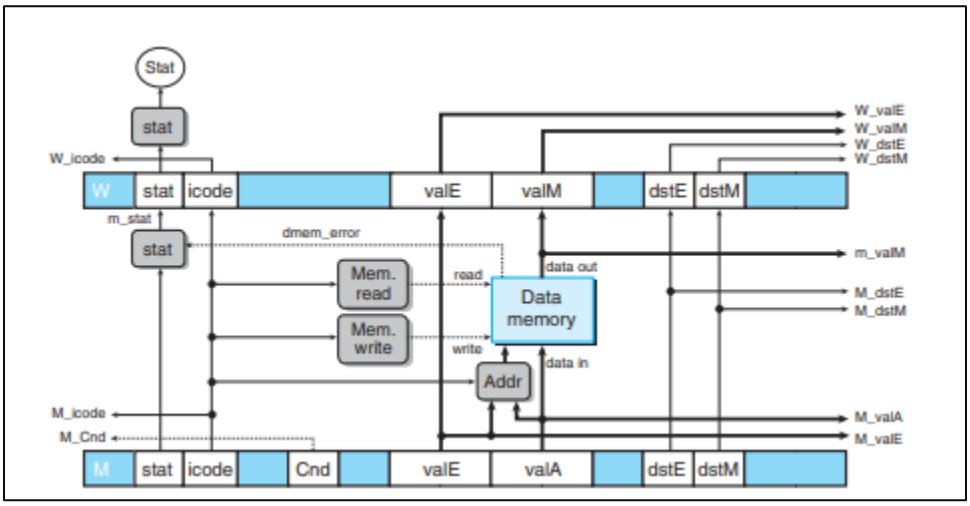
Command to run the code:

iverilog –o execute execute_tb.v

vvp execute

MEMORY

The implementation of the memory stage remains almost the same as that in sequential processor.



Implementation and Working:

- We read from and write to the data memory in same way as we did for the sequential processor.
- We store the values from the memory register and the output values from memory stage in the write back register at the positive edge of the clock.

```
W_stat <= m_stat;
W_icode <= M_icode;
W_valE <= M_valE;
W_valM <= m_valM;
W_dstE <= M_dstE;
W_dstM <= M_dstM;
```

Source code:

```
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 [1:0] 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 [1:0] 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 [1:0] m_stat;

    reg [63:0] data_memory [1023:0];
    reg [63:0] memory_address;
    reg dmem_error = 0;

    // Initialization
    initial
    begin
        m_valM = 64'd0;
        dmem_error = 1'b0;
        memory_address = 1'b0;
    end
```

```
    // instruction codes
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
    // Hardcoding Data Memory
    initial
    begin
        data_memory[0] = 64'd0;
        data_memory[1] = 64'd1;
        data_memory[2] = 64'd2;
        data_memory[3] = 64'd10;
        data_memory[4] = 64'd4;
        data_memory[5] = 64'd5;
        data_memory[6] = 64'd6;
        data_memory[7] = 64'd7;
        data_memory[8] = 64'd8;
        data_memory[9] = 64'd9;
        data_memory[10] = 64'd10;
        data_memory[11] = 64'd11;
        data_memory[12] = 64'd12;
        data_memory[13] = 64'd13;
        data_memory[14] = 64'd14;
        data_memory[15] = 64'd15;
        data_memory[16] = 64'd16;
        data_memory[17] = 64'd17;
        data_memory[18] = 64'd18;
        data_memory[20] = 64'd20;
        data_memory[128] = 64'd4;
    end
```

```
    always@(*)
    begin
        if(M_icode == IRMMOVQ)
        begin
            memory_address = M_valE;
            data_memory[M_valE] = M_valA;
        end
        else if (M_icode == IMRMOVQ)
        begin
            memory_address = M_valE;
            m_valM = data_memory[M_valE];
        end
        else if (M_icode == ICALL)
        begin
            memory_address = M_valE;
            data_memory[M_valE] = M_valA;
        end
        else if (M_icode == IRET)
        begin
            memory_address = M_valA;
            m_valM = data_memory[M_valA];
        end
        else if (M_icode == IPUSHQ)
        begin
            memory_address = M_valE;
            data_memory[M_valE] = M_valA;
        end
        else if (M_icode == IPOPQ)
        begin
            memory_address = M_valA;
            m_valM = data_memory[M_valA];
        end
    end
```

```
    always@(*)
    begin
        if(memory_address>64'd1023)
        begin
            dmem_error = 1'd1;
        end
    end

    always@(*)
    begin
        if(dmem_error==1)
            m_stat = 2'd2;
        else
            m_stat = M_stat;
    end

    always@(posedge clk)
    // always@(*)
    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
```

Test bench:

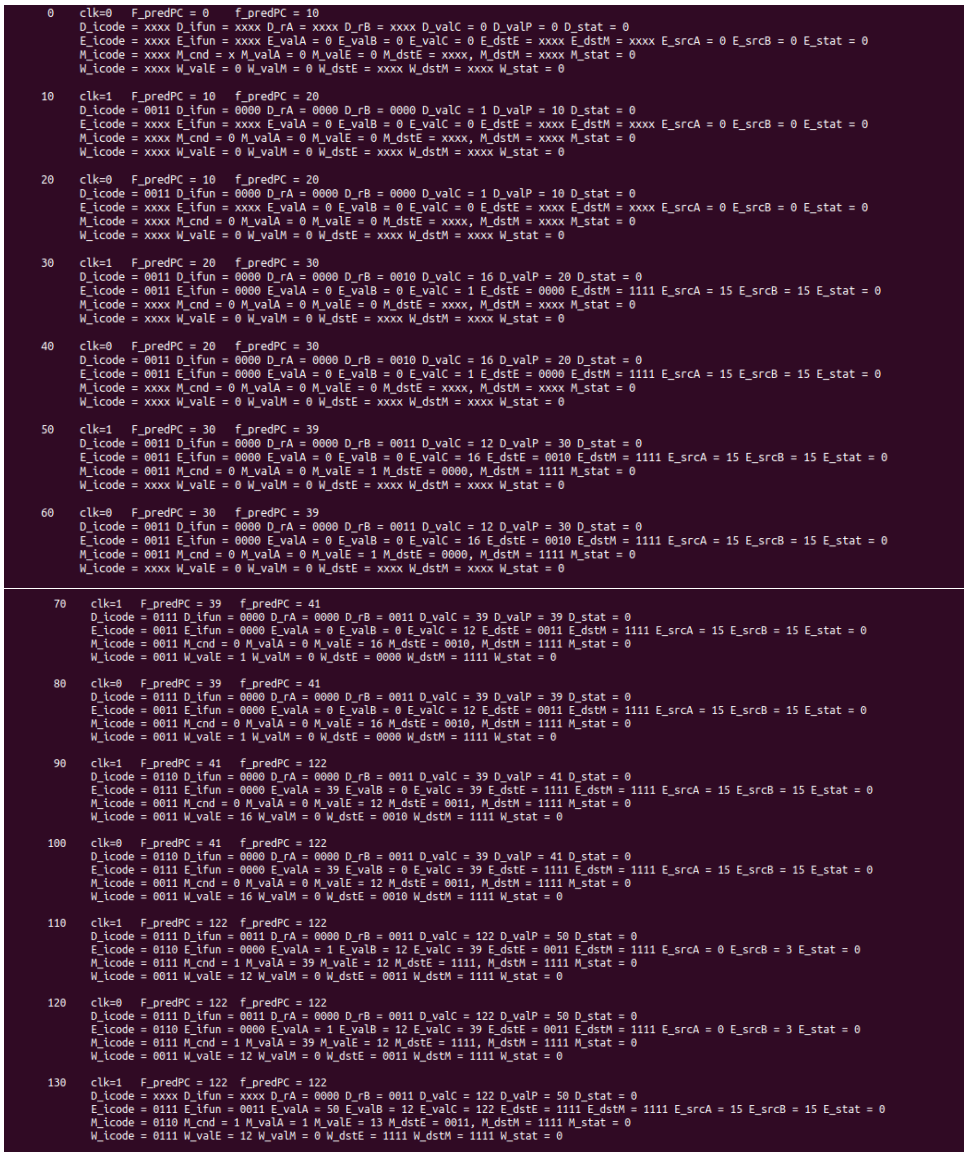
```
`timescale 1ns/10ps

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

module memory_tb();

    reg clk;
    reg [63:0] F_predPC;
    reg F_stall, D_stall, D_bubble, E_bubble, W_stall;
    reg set_cc;

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;
    wire [3:0] D_rA;
    wire [3:0] D_rB;
    wire [63:0] D_valC;
    wire [63:0] D_valP;
    wire [1:0] D_stat;  //AOK|HLT|ADR|INS
    wire [3:0] d_srcA;
    wire [3:0] d_srcB;
    wire [63:0] d_valA;
    wire [63:0] d_valB;

    wire [3:0] E_icode;
    wire [3:0] E_ifun;
    wire [3:0] E_srcA;
    wire [3:0] E_srcB;
    wire [63:0] E_valA;
    wire [63:0] E_valB;
    wire [63:0] E_valC;
    wire [3:0] E_dstE;
    wire [3:0] E_dstM;
    wire [1:0] E_stat;  //AOK|HLT|ADR|INS

   wire [3:0] M_icode;
   wire [63:0] M_valA;
   wire [63:0] M_valM;
   wire [63:0] M_valE;
   wire [3:0] M_dstM;
   wire [3:0] M_dstE;
   wire M_cnd;
   wire[1:0] M_stat;

   wire [3:0] e_dstE;
   wire [63:0] e_valE;
   wire e_cnd;

   wire [3:0] W_icode;
   wire [3:0] W_dstE;
   wire [3:0] W_dstM;
   wire [63:0] W_valE;
   wire [63:0] W_valM;
   wire [1:0] W_stat;

   wire [63:0] m_valM;
   wire [1:0] m_stat;

   fetch fetch_(clk, F_predPC, M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble, f_predPC, D_icode, D_ifun,
                D_rA, D_rB, D_valC, D_valP, D_stat);
   decode decode_(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE, e_valE, m_valM, M_valE, W_valM, W_valE,
                  E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat, d_srcA, d_srcB, d_valA, d_valB);
   execute execute_(clk, E_stat, E_icode, E_ifun, E_valC, E_valA, E_valB, E_dstE, E_dstM, set_cc, M_stat, M_icode, M_cnd, M_valE, M_valA, M_dstE,
              M_dstM, e_valE, e_dstE, e_cnd);
   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);

   always @(W_icode)
   begin
       if(W_icode==0)
       $finish;
   end

always @(posedge clk) F_predPC <= f_predPC;

always #10 clk = ~clk;

initial
begin

   $dumpfile("memory_tb.vcd");
   $dumpvars(0, memory_tb);

   F_predPC = 64'd32;
   clk = 0;
   F_stall = 0;
   D_stall = 0;
   D_bubble = 0;
   E_bubble = 0;

end

initial
begin
   $monitor($time, "\tclk=%d\tF_predPC = %g\tf_predPC = %g\n\t\t\tD_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n
   \t\t\tE_icode = %b E_ifun = %b E_valA = %g E_valB = %g E_valC = %g E_dstE = %b E_dstM = %b E_srcA = %g E_srcB = %g E_stat = %g\n
   \t\t\tM_icode = %b M_cnd = %b M_valA = %g M_valE = %g M_dstE = %b, M_dstM = %b M_stat = %g\n
   \t\t\tW_icode = %b W_valE = %g W_valM = %g W_dstE = %b W_dstM = %b W_stat = %g\n"
   ,clk,F_predPC,f_predPC, D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,E_icode,E_ifun,E_valA,E_valB,E_valC,E_dstE,E_dstM,E_srcA,E_srcB,E_stat,
   M_icode,M_cnd,M_valA,M_valE,M_dstE,M_dstM,M_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,W_stat);
end

endmodule
```

Instructions given to memory:

```
irmovq $0x0, %rax

irmovq $0x10, %rdx

irmovq $0xc, %rbx

jmp check
```

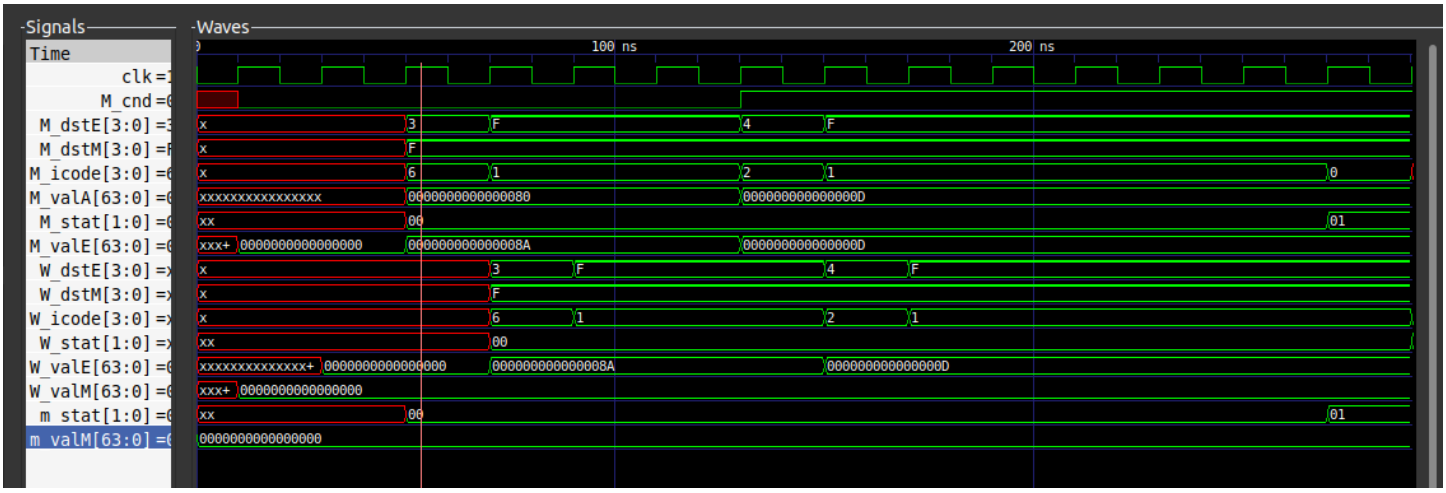
```
check:
        addq %rax, %rbx
        je rbxres
        addq %rax, %rdx
        je rdxres
        jmp loop2
rbxres:
rdxres:
Loop2:
        Halt
```

Output of Test Bench:



```
140    clk=0   F_predPC = 122  f_predPC = 122
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 50 E_valB = 12 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0110 M_cnd = 1 M_valA = 1 M_valE = 13 M_dstE = 0011, M_dstM = 1111 M_stat = 0
       W_icode = 0111 W_valE = 12 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

150    clk=1   F_predPC = 122  f_predPC = 52
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0110 W_valE = 13 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0

160    clk=0   F_predPC = 122  f_predPC = 52
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0110 W_valE = 13 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0

170    clk=1   F_predPC = 52   f_predPC = 125
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0111 W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

180    clk=0   F_predPC = 52   f_predPC = 125
       D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 12 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0111 W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

190    clk=1   F_predPC = 125  f_predPC = 125
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 17 E_valC = 122 E_dstE = 0010 E_dstM = 1111 E_srcA = 0 E_srcB = 2 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

200    clk=0   F_predPC = 125  f_predPC = 125
       D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 17 E_valC = 122 E_dstE = 0010 E_dstM = 1111 E_srcA = 0 E_srcB = 2 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

210    clk=1   F_predPC = 125  f_predPC = 125
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 18 M_dstE = 0010, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

220    clk=0   F_predPC = 125  f_predPC = 125
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 18 M_dstE = 0010, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

230    clk=1   F_predPC = 125  f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0110 W_valE = 18 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

240    clk=0   F_predPC = 125  f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0110 W_valE = 18 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

250    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0111 W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

260    clk=0   F_predPC = 62   f_predPC = 62
       D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = 0111 W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

270    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = 0000 E_ifun = 0000 E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

280    clk=0   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = 0000 E_ifun = 0000 E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       W_icode = xxxx W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

290    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       M_icode = 0000 M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       W_icode = xxxx W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

300    clk=0   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       M_icode = 0000 M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       W_icode = xxxx W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

310    clk=1   F_predPC = 62   f_predPC = 62
       D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 62 D_stat = 1
       E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 17 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 18 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       W_icode = 0000 W_valE = 18 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 1
```

Gtkwave output:



Command to run the code:

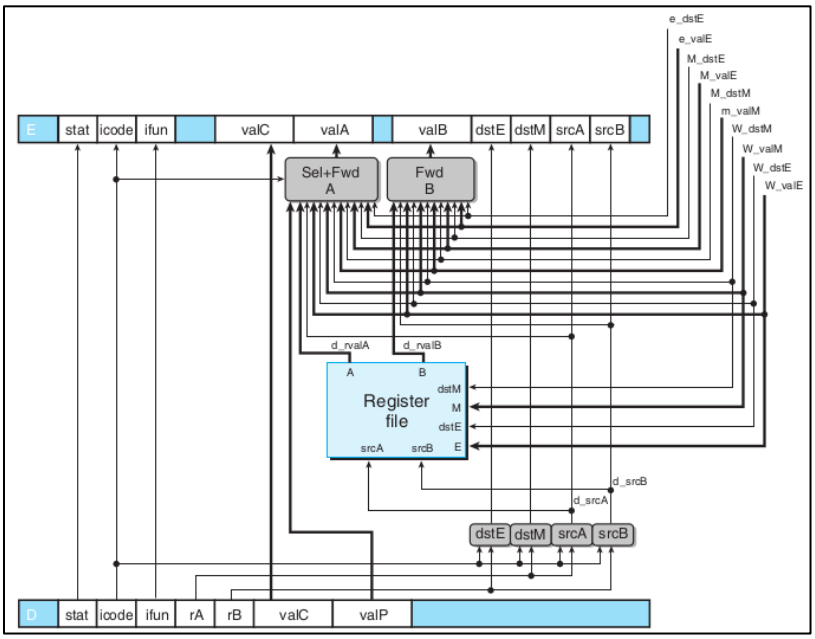
iverilog –o memory memory_tb.v

vvp memory

WRITE BACK

The implementation of this stage is same as that of the write back stage in SEQ.

Like in SEQ decode and write back can be combined as they both access the register file.



Implementation and Working:

- The only change made is while reading the values stored in the registers from the text file into a register array and writing the updated register values into the text file. Instead of always reading when clock is 1 and writing when clock is at 0, we read always and write into the text file only when there is any update in the register values.

Source code:

```
// WRITE BACK BLOCK

module write_back(clk, W_icode, W_dstE, W_dstM, W_valE, W_valM);

    input clk;
    input [3:0] W_icode;

    input [3:0] W_dstE;
    input [3:0] W_dstM;

    input [63:0] W_valE;
    input [63:0] W_valM;


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

    // instruction codes
    // Constant values used in HCL descriptions.
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;
```

```
// registers
parameter Register_rax = 4'd0;
parameter Register_rcx = 4'd1;
parameter Register_rdx = 4'd2;
parameter Register_rbx = 4'd3;
parameter RRSP         = 4'd4;
parameter Register_rbp = 4'd5;
parameter Register_rsi = 4'd6;
parameter Register_rdi = 4'd7;
parameter Register_r8  = 4'd8;
parameter Register_r9  = 4'd9;
parameter Register_r10 = 4'd10;
parameter Register_r11 = 4'd11;
parameter Register_r12 = 4'd12;
parameter Register_r13 = 4'd13;
parameter Register_r14 = 4'd14;
parameter RNONE        = 4'd15;

always@(posedge clk)
begin
    $readmemh("reg_file.txt", Reg_File);
end
```

```
always@(*)
begin
    //rrmovq and  cmovXX
    if (W_icode == IRRMOVQ)
    begin

        // R[rB] ← valE
        Reg_File[W_dstE] = W_valE;

        $writememh("reg_file.txt", Reg_File);

    end
    //irmovq
    else if (W_icode == IIRMOVQ)
    begin

        // R[rB] ← valE
        Reg_File[W_dstE] = W_valE;

        $writememh("reg_file.txt", Reg_File);

    end
    // mrmovq
    else if (W_icode == IMRMOVQ)
    begin

        // R[rA] ← valM
        Reg_File[W_dstM] = W_valM;

        $writememh("reg_file.txt", Reg_File);

    end
```

```
        //Opq
        else if (W_icode == IOPQ)
        begin

            // R[rB] ← valE
            Reg_File[W_dstE] = W_valE;

            $writememh("reg_file.txt", Reg_File);

        end
        //call
        else if (W_icode == ICALL)
        begin

            // R[ %rsp ] ← valE
            Reg_File[W_dstE] = W_valE;

            $writememh("reg_file.txt", Reg_File);

        end
        //ret
        else if (W_icode == IRET)
        begin

            // R[ %rsp ] ← valE
            Reg_File[W_dstE] = W_valE;

            $writememh("reg_file.txt", Reg_File);

        end

       //pushq
       else if (W_icode == IPUSHQ)
       begin

           // R[ %rsp ] ← valE
           Reg_File[W_dstE] = W_valE;

           $writememh("reg_file.txt", Reg_File);

       end
       //popq
       else if (W_icode == IPOPQ)
       begin

           // R[ %rsp ] ← valE
           Reg_File[W_dstE] = W_valE;

           // R[rA] ← valM
           Reg_File[W_dstM] = W_valM;

           $writememh("reg_file.txt", Reg_File);

       end
   end
endmodule
```

Test bench:

write_back_tb.v

```
`timescale 1ns/10ps

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

module write_back_tb();

    reg clk;
    reg [63:0] F_predPC;

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

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;
    wire [3:0] D_rA;
    wire [3:0] D_rB;
    wire [63:0] D_valC;
    wire [63:0] D_valP;
    wire [1:0] D_stat;  //AOK|HLT|ADR|INS

    wire [63:0] d_valA, d_valB;
    wire [3:0] d_srcA, d_srcB;

  wire [3:0] E_icode;
  wire [3:0] E_ifun;
  wire [3:0] E_srcA;
  wire [3:0] E_srcB;
  wire [63:0] E_valA;
  wire [63:0] E_valB;
  wire [63:0] E_valC;
  wire [3:0] E_dstE;
  wire [3:0] E_dstM;
  wire [1:0] E_stat;  //AOK|HLT|ADR|INS

  wire [3:0] M_icode;
  wire [63:0] M_valA;
  wire [63:0] M_valM;
  wire [63:0] M_valE;
  wire [3:0] M_dstM;
  wire [3:0] M_dstE;
  wire M_cnd;
  wire[1:0] M_stat;

  wire [3:0] e_dstE;
  wire [63:0] e_valE;
  wire e_cnd;

  wire [3:0] W_icode;
  wire [3:0] W_dstE;
  wire [3:0] W_dstM;
  wire [63:0] W_valE;
  wire [63:0] W_valM;
  wire [1:0] W_stat;

  wire [63:0] m_valM;
  wire [1:0] m_stat;

   fetch fetch_(clk, F_predPC, M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble, f_predPC, D_icode, D_ifun, D_rA, D_rB,
                D_valC, D_valP, D_stat);
   decode decode_(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE, e_valE, m_valM, M_valE,
                  W_valM, W_valE, E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat, d_srcA, d_srcB, d_valA, d_valB);
   execute execute_(clk, E_stat, E_icode, E_ifun, E_valC, E_valA, E_valB, E_dstE, E_dstM, set_cc, M_stat, M_icode, M_cnd, M_valE, M_valA,
                    M_dstE, M_dstM, e_valE, e_dstE, e_cnd);
   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);
   write_back write_back_(clk, W_icode, W_dstE, W_dstM, W_valE, W_valM);

   always @(W_icode)
   begin
       if(W_icode==0)
       $finish;
   end

 always @(posedge clk) F_predPC <= f_predPC;

 always #10 clk = ~clk;

 initial
 begin
   $dumpfile("write_back_tb.vcd");
   $dumpvars(0, write_back_tb);

   F_predPC = 64'd32;
   clk = 0;
   F_stall = 0;
   D_stall = 0;
   D_bubble = 0;
   E_bubble = 0;
 end

 initial
 begin
   $monitor($time, "\tclk=%d\n\t\t\tF Reg:\t\tF_predPC = %g\n\t\t\tfetch:\t\tf_predPC = %g\n\t\t\t
   D Reg:\t\tD_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n\t\t\t
   decode:\t\td_valA = %g d_valB = %g\n\t\t\tE Reg:\t\tE_icode = %b E_ifun = %b E_valA = %g E_valB = %g E_valC = %g
   E_dstE = %b E_dstM = %b E_srcA = %g E_srcB = %g E_stat = %g\n\t\t\texecute:\te_cnd = %b e_valE = %g\n\t\t\t
   M Reg:\t\tM_icode = %b M_cnd = %b M_valA = %g M_valE = %g M_dstE = %b, M_dstM = %b M_stat = %g\n\t\t\tmemory:
   \t\tm_valM = %g\n\t\t\tW Reg:\t\tW_icode = %b W_valE = %g W_valM = %g W_dstE = %b W_dstM = %b W_stat = %g\n",
   clk,F_predPC,f_predPC,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,d_valA,d_valB,E_icode,E_ifun,E_valA,E_valB,E_valC,E_dstE,
   E_dstM,E_srcA,E_srcB,E_stat,e_cnd,e_valE,M_icode,M_cnd,M_valA,M_valE,M_dstE,M_dstM,M_stat,m_valM,W_icode,W_valE,W_valM,W_dstE,W_dstM,W_stat);
 end

endmodule
```

Instructions given to write back:

```
irmovq $0x0, %rax
```

```
irmovq $0x10, %rdx
irmovq $0xc, %rbx
jmp check
check:
    addq %rax, %rbx
    je rbxres
    addq %rax, %rdx
    je rdxres
    jmp loop2
rbxres:
rdxres:
Loop2:
    Halt
```

Output of Test Bench:

```
  0    clk=0
       F Reg:          F_predPC = 0
       fetch:          f_predPC = 10
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = xxxx D_rB = xxxx D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = x M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

  10   clk=1
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 1 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

  20   clk=0
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 1 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

  30   clk=1
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 16 D_valP = 20 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1 E_dstE = 0000 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
```

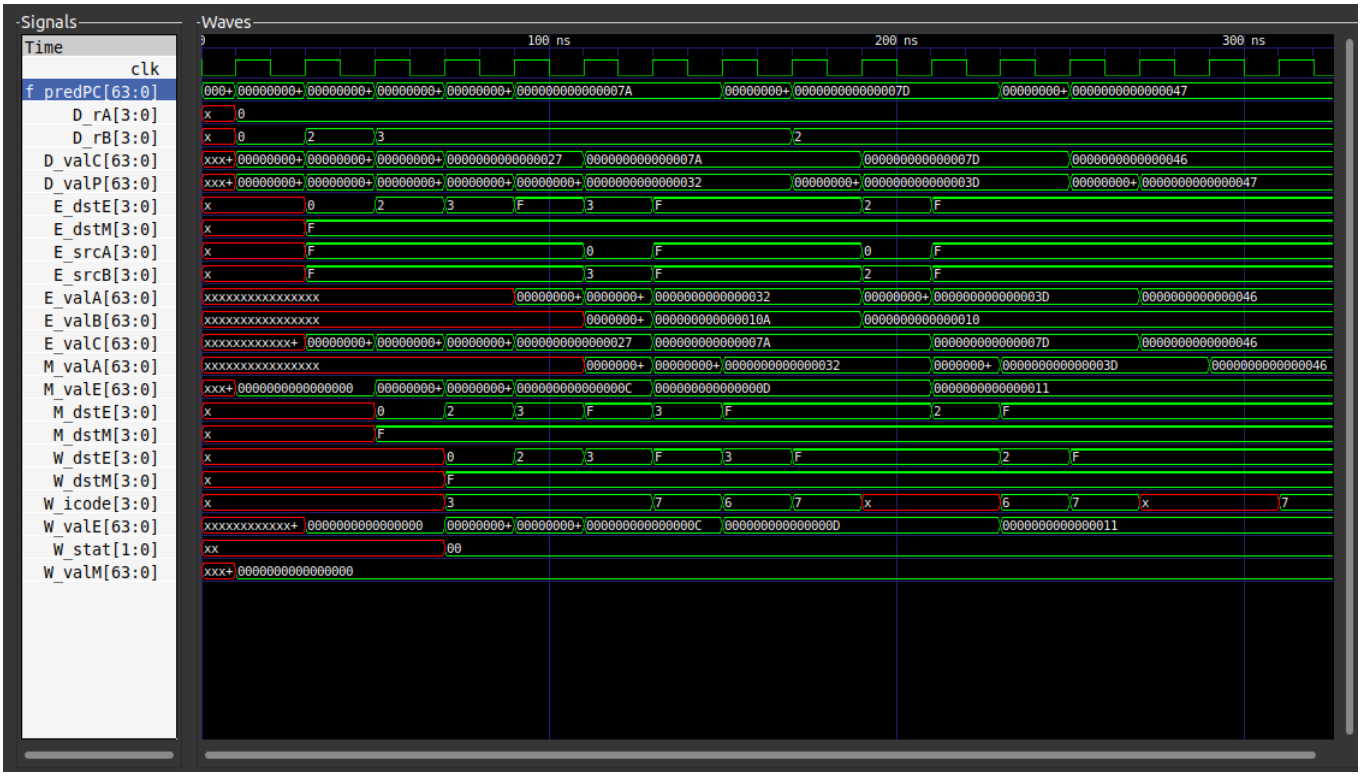
```
40    clk=0
      F Reg:          F_predPC = 20
      fetch:          f_predPC = 30
      D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 16 D_valP = 20 D_stat = 0
      decode:         d_valA = 0 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1 E_dstE = 0000 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 1
      M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

50    clk=1
      F Reg:          F_predPC = 30
      fetch:          f_predPC = 39
      D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 12 D_valP = 30 D_stat = 0
      decode:         d_valA = 0 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 16 E_dstE = 0010 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 16
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1 M_dstE = 0000, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

60    clk=0
      F Reg:          F_predPC = 30
      fetch:          f_predPC = 39
      D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 12 D_valP = 30 D_stat = 0
      decode:         d_valA = 0 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 16 E_dstE = 0010 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 16
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1 M_dstE = 0000, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0

70    clk=1
      F Reg:          F_predPC = 39
      fetch:          f_predPC = 41
      D Reg:          D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 39 D_stat = 0
      decode:         d_valA = 39 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 12 E_dstE = 0011 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 12
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 16 M_dstE = 0010, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 1 W_valM = 0 W_dstE = 0000 W_dstM = 1111 W_stat = 0

80    clk=0
      F Reg:          F_predPC = 39
      fetch:          f_predPC = 41
      D Reg:          D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 39 D_stat = 0
      decode:         d_valA = 39 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 12 E_dstE = 0011 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 12
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 16 M_dstE = 0010, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 1 W_valM = 0 W_dstE = 0000 W_dstM = 1111 W_stat = 0

90    clk=1
      F Reg:          F_predPC = 41
      fetch:          f_predPC = 122
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 41 D_stat = 0
      decode:         d_valA = 1 d_valB = 12
      E Reg:          E_icode = 0111 E_ifun = 0000 E_valA = 39 E_valB = 0 E_valC = 39 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 1 e_valE = 12
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 12 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 16 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

100   clk=0
      F Reg:          F_predPC = 41
      fetch:          f_predPC = 122
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0011 D_valC = 39 D_valP = 41 D_stat = 0
      decode:         d_valA = 1 d_valB = 12
      E Reg:          E_icode = 0111 E_ifun = 0000 E_valA = 39 E_valB = 0 E_valC = 39 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 1 e_valE = 12
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 12 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 16 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

110   clk=1
      F Reg:          F_predPC = 122
      fetch:          f_predPC = 122
      D Reg:          D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
      decode:         d_valA = 50 d_valB = 266
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 12 E_valC = 39 E_dstE = 0011 E_dstM = 1111 E_srcA = 0 E_srcB = 3 E_stat = 0
      execute:        e_cnd = 1 e_valE = 13
      M Reg:          M_icode = 0111 M_cnd = 1 M_valA = 39 M_valE = 12 M_dstE = 1111, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 12 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0

120   clk=0
      F Reg:          F_predPC = 122
      fetch:          f_predPC = 122
      D Reg:          D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
      decode:         d_valA = 50 d_valB = 266
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 12 E_valC = 39 E_dstE = 0011 E_dstM = 1111 E_srcA = 0 E_srcB = 3 E_stat = 0
      execute:        e_cnd = 1 e_valE = 13
      M Reg:          M_icode = 0111 M_cnd = 1 M_valA = 39 M_valE = 12 M_dstE = 1111, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 12 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0

130   clk=1
      F Reg:          F_predPC = 122
      fetch:          f_predPC = 122
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
      decode:         d_valA = 50 d_valB = 266
      E Reg:          E_icode = 0111 E_ifun = 0011 E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0110 M_cnd = 1 M_valA = 1 M_valE = 13 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0111 W_valE = 12 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

140   clk=0
      F Reg:          F_predPC = 122
      fetch:          f_predPC = 122
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
      decode:         d_valA = 50 d_valB = 266
      E Reg:          E_icode = 0111 E_ifun = 0011 E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0110 M_cnd = 1 M_valA = 1 M_valE = 13 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0111 W_valE = 12 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

150   clk=1
      F Reg:          F_predPC = 122
      fetch:          f_predPC = 52
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
      decode:         d_valA = 50 d_valB = 266
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0110 W_valE = 13 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0
```

```
160    clk=0
       F Reg:          F_predPC = 122
       fetch:          f_predPC = 52
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0011 D_valC = 122 D_valP = 50 D_stat = 0
       decode:         d_valA = 50 d_valB = 266
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 13
       M Reg:          M_icode = 0111 M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0110 W_valE = 13 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0

170    clk=1
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 125
       D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       decode:         d_valA = 1 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 13
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

180    clk=0
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 125
       D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 122 D_valP = 52 D_stat = 0
       decode:         d_valA = 1 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 50 E_valB = 266 E_valC = 122 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 13
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

190    clk=1
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 125
       D Reg:          D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 16 E_valC = 122 E_dstE = 0010 E_dstM = 1111 E_srcA = 0 E_srcB = 2 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

200    clk=0
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 125
       D Reg:          D_icode = 0111 D_ifun = 0011 D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 1 E_valB = 16 E_valC = 122 E_dstE = 0010 E_dstM = 1111 E_srcA = 0 E_srcB = 2 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 50 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

210    clk=1
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 125
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 17 M_dstE = 0010, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

220    clk=0
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 125
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = 0111 E_ifun = 0011 E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = 0110 M_cnd = 0 M_valA = 1 M_valE = 17 M_dstE = 0010, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 13 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

230    clk=1
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 70
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0110 W_valE = 17 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

240    clk=0
       F Reg:          F_predPC = 125
       fetch:          f_predPC = 70
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 125 D_valP = 61 D_stat = 0
       decode:         d_valA = 61 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = 0111 M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0110 W_valE = 17 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0

250    clk=1
       F Reg:          F_predPC = 70
       fetch:          f_predPC = 71
       D Reg:          D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 70 D_stat = 0
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

260    clk=0
       F Reg:          F_predPC = 70
       fetch:          f_predPC = 71
       D Reg:          D_icode = 0111 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 70 D_stat = 0
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 61 E_valB = 16 E_valC = 125 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

270    clk=1
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = 0111 E_ifun = 0000 E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0
```

```
280    clk=0
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = 0111 E_ifun = 0000 E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 61 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

290    clk=1
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = 0000 E_ifun = 0000 E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = 0111 M_cnd = 1 M_valA = 70 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

300    clk=0
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = 0000 E_ifun = 0000 E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = 0111 M_cnd = 1 M_valA = 70 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

310    clk=1
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = 0000 M_cnd = 1 M_valA = 70 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0
```

```
320    clk=0
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = 0000 M_cnd = 1 M_valA = 70 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 0
       W Reg:          W_icode = 0111 W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 0

330    clk=1
       F Reg:          F_predPC = 71
       fetch:          f_predPC = 71
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0000 D_rB = 0010 D_valC = 70 D_valP = 71 D_stat = 1
       decode:         d_valA = 70 d_valB = 16
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 70 E_valB = 16 E_valC = 70 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 1 e_valE = 17
       M Reg:          M_icode = xxxx M_cnd = 1 M_valA = 70 M_valE = 17 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 0
       W Reg:          W_icode = 0000 W_valE = 17 W_valM = 0 W_dstE = 1111 W_dstM = 1111 W_stat = 1
```

Gtkwave output:



Command to run the code:

iverilog –o write_back write_back_tb.v

vvp write_back

PIPE CONTROL

This block is used for handling data and control hazards. In cases of return, mispredicted branches and load/use hazards, we can observe misbehaviour and incorrect results. To handle such cases, we might need to stall some stage and insert a bubble in some.

- The return instruction can be detected using the condition given below,
  - IRET in { D_icode, E_icode, M_icode}
- Branch misprediction can be detected using the condition given below,
  - E_icode = IJXX & !e_cnd
- Load/use hazard can be detected using the condition given below,
  - E_icode in { MRMOVQ, IPOPQ } && E_dstM in {d_srcA, d_srcB}

These hazards can be handled using the scheme given below for each stage:

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

Source Code:

```
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 [1:0] 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;
    output reg set_cc;

    // Instruction codes
    parameter IHALT   = 4'd0;
    parameter INOP    = 4'd1;
    parameter IRRMOVQ = 4'd2; //rrmovq and  cmovXX
    parameter IIRMOVQ = 4'd3;
    parameter IRMMOVQ = 4'd4;
    parameter IMRMOVQ = 4'd5;
    parameter IOPQ    = 4'd6;
    parameter IJXX    = 4'd7;
    parameter ICALL   = 4'd8;
    parameter IRET    = 4'd9;
    parameter IPUSHQ  = 4'd10;
    parameter IPOPQ   = 4'd11;


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

        if((E_icode == IMRMOVQ | E_icode == IPOPQ) & (E_dstM == d_srcA | E_dstM == d_srcB))
        begin
            F_stall = 1;
            D_stall = 1;
            E_bubble = 1;
        end
        else if((E_icode == IJXX & e_cnd==0))
        begin
            D_bubble = 1;
            E_bubble = 1;
        end
        else if(D_icode == IRET | E_icode == IRET | M_icode == IRET)
        begin
            F_stall = 1;
            D_bubble = 1;
        end
        else if(E_icode == 4'h0 | m_stat!=2'b0 | W_stat!=2'b0)
        begin
            set_cc = 0;
        end
    end

endmodule
```

PIPELINE PROCESSOR

Source code:

```
`timescale 1ns/10ps

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

module pipe();

    reg clk;
    reg [63:0] F_predPC;

    reg [1:0] stat;

    wire [63:0] f_predPC;

    wire [3:0] D_icode;
    wire [3:0] D_ifun;
    wire [3:0] D_rA;
    wire [3:0] D_rB;
    wire [63:0] D_valC;
    wire [63:0] D_valP;
    wire [1:0] D_stat;  //AOK|HLT|ADR|INS

    wire [63:0] d_valA, d_valB;
    wire [3:0] d_srcA, d_srcB;

    wire [3:0] E_icode;
    wire [3:0] E_ifun;
    wire [3:0] E_srcA;
    wire [3:0] E_srcB;
    wire [63:0] E_valA;
    wire [63:0] E_valB;
    wire [63:0] E_valC;
    wire [3:0] E_dstE;
    wire [3:0] E_dstM;
    wire [1:0] E_stat;  //AOK|HLT|ADR|INS

    wire [3:0] M_icode;
    wire [63:0] M_valA;
    wire [63:0] M_valE;
    wire [3:0] M_dstM;
    wire [3:0] M_dstE;
    wire M_cnd;
    wire[1:0] M_stat;

    wire [3:0] e_dstE;
    wire [63:0] e_valE;
    wire e_cnd;

    wire [3:0] W_icode;
    wire [3:0] W_dstE;
    wire [3:0] W_dstM;
    wire [63:0] W_valE;
    wire [63:0] W_valM;
    wire [1:0] W_stat;

    wire [63:0] m_valM;
    wire [1:0] m_stat;

    wire F_stall, D_stall, D_bubble, E_bubble;
    wire set_cc;

    fetch fetch_(clk, F_predPC, M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble, f_predPC, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat);
    decode decode_(clk, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP, D_stat, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE, e_valE, m_valM,
                   M_valE, W_valM, W_valE, E_bubble, E_icode, E_ifun, E_valA, E_valB, E_valC, E_dstE, E_dstM, E_srcA, E_srcB, E_stat, d_srcA, d_srcB, d_valA, d_valB);
    execute execute_(clk, E_stat, E_icode, E_ifun, E_valC, E_valA, E_valB, E_dstE, E_dstM, set_cc,
                     M_stat, M_icode, M_cnd, M_valE, M_valA, M_dstE, M_dstM, e_valE, e_dstE, e_cnd);
    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);
    write_back write_back_(clk, W_icode, W_dstE, W_dstM, W_valE, W_valM);
    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@(W_stat)
    begin
        stat = W_stat;
    end

    always@(stat)
    begin
        if(stat != 2'd0)
        begin
            $finish;
        end
    end

 always @(posedge clk)
 begin
   if (F_stall == 0)
   begin
     F_predPC <= f_predPC;
   end
 end

always #10 clk = ~clk;

initial
begin

  $dumpfile("pipe.vcd");
  $dumpvars(0, pipe);

  F_predPC = 64'd0;
  clk = 0;
end

initial
begin
  $monitor($time, "\tclk=%d\n\t\t\tF Reg:\t\t
  F_predPC = %g\n\t\t\tfetch:\t\tf_predPC = %g\n\t\t\t
            D Reg:\t\tD_icode = %b D_ifun = %b D_rA = %b D_rB = %b D_valC = %g D_valP = %g D_stat = %g\n\t\t\t
            decode:\t\td_valA = %g d_valB = %g\n\t\t\t
            E Reg:\t\tE_icode = %b E_ifun = %b E_valA = %g E_valB = %g E_valC = %g E_dstE = %b E_dstM = %b E_srcA = %g E_srcB = %g E_stat = %g\n\t\t\t
            execute:\te_cnd = %b e_valE = %g\n\t\t\t
            M Reg:\t\tM_icode = %b M_cnd = %b M_valA = %g M_valE = %g M_dstE = %b, M_dstM = %b M_stat = %g\n\t\t\t
            memory:\t\tm_valM = %g\n\t\t\tW Reg:\t\tW_icode = %b W_valE = %g W_valM = %g W_dstE = %b W_dstM = %b W_stat = %g\n\t\t\t
            F_stall = %g D_stall = %g D_bubble = %g E_bubble = %g set_cc = %g\n",clk,F_predPC,f_predPC,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,d_valA,d_valB,
            E_icode,E_ifun,E_valA,E_valB,E_valC,E_dstE,E_dstM,E_srcA,E_srcB,E_stat,e_cnd,e_valE,M_icode,M_cnd,M_valA,M_valE,M_dstE,M_dstM,M_stat,m_valM,
            W_icode,W_valE,W_valM,W_dstE,W_dstM,W_stat,F_stall,D_stall,D_bubble,E_bubble,set_cc);
end

endmodule
```

1st set of instructions given to the Processor:

```
irmovq $128 %rdx

irmovq $3 %rcx

rmmovq %rcx 0(%rdx)

irmovq $10 %rbx

mrmovq 0(%rdx) %rax

addq %rbx %rax

halt
```

Output of 1st set of instruction:

```
  0    clk=0
       F Reg:          F_predPC = 0
       fetch:          f_predPC = 10
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = xxxx D_rB = xxxx D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = x M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 10    clk=1
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0010 D_valC = 128 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 20    clk=0
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0010 D_valC = 128 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 30    clk=1
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0001 D_valC = 3 D_valP = 20 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 128 E_dstE = 0010 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 128
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 40    clk=0
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0001 D_valC = 3 D_valP = 20 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 128 E_dstE = 0010 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 128
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 50    clk=1
       F Reg:          F_predPC = 30
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0100 D_ifun = 0000 D_rA = 0001 D_rB = 0010 D_valC = 0 D_valP = 30 D_stat = 0
       decode:         d_valA = 3 d_valB = 0
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 3 E_dstE = 0001 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 3
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 128 M_dstE = 0010, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
60    clk=0
      F Reg:          F_predPC = 30
      fetch:          f_predPC = 40
      D Reg:          D_icode = 0100 D_ifun = 0000 D_rA = 0001 D_rB = 0010 D_valC = 0 D_valP = 30 D_stat = 0
      decode:         d_valA = 3 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 3 E_dstE = 0001 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 3
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 128 M_dstE = 0010, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

70    clk=1
      F Reg:          F_predPC = 40
      fetch:          f_predPC = 50
      D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0011 D_valC = 10 D_valP = 40 D_stat = 0
      decode:         d_valA = 3 d_valB = 0
      E Reg:          E_icode = 0100 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 0 E_dstE = 0010 E_dstM = 1111 E_srcA = 1 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 3 M_dstE = 0001, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 128 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

80    clk=0
      F Reg:          F_predPC = 40
      fetch:          f_predPC = 50
      D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0011 D_valC = 10 D_valP = 40 D_stat = 0
      decode:         d_valA = 3 d_valB = 0
      E Reg:          E_icode = 0100 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 0 E_dstE = 0010 E_dstM = 1111 E_srcA = 1 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 3 M_dstE = 0001, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 128 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
90    clk=1
      F Reg:          F_predPC = 50
      fetch:          f_predPC = 52
      D Reg:          D_icode = 0101 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 0 D_valP = 50 D_stat = 0
      decode:         d_valA = 3 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 10 E_dstE = 0011 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 10
      M Reg:          M_icode = 0100 M_cnd = 0 M_valA = 3 M_valE = 0 M_dstE = 0010, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 3 W_valM = 0 W_dstE = 0001 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

100   clk=0
      F Reg:          F_predPC = 50
      fetch:          f_predPC = 52
      D Reg:          D_icode = 0101 D_ifun = 0000 D_rA = 0000 D_rB = 0010 D_valC = 0 D_valP = 50 D_stat = 0
      decode:         d_valA = 3 d_valB = 0
      E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 10 E_dstE = 0011 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 10
      M Reg:          M_icode = 0100 M_cnd = 0 M_valA = 3 M_valE = 0 M_dstE = 0010, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0011 W_valE = 3 W_valM = 0 W_dstE = 0001 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

110   clk=1
      F Reg:          F_predPC = 52
      fetch:          f_predPC = 53
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 52 D_stat = 0
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = 0101 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 0000 E_srcA = 15 E_srcB = 2 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 3 M_valE = 10 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0100 W_valE = 0 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0
      F_stall = 1 D_stall = 1 D_bubble = 0 E_bubble = 1 set_cc = 1
```

```
120   clk=0
      F Reg:          F_predPC = 52
      fetch:          f_predPC = 53
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 52 D_stat = 0
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = 0101 E_ifun = 0000 E_valA = 3 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 0000 E_srcA = 15 E_srcB = 2 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 3 M_valE = 10 M_dstE = 0011, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 0
      W Reg:          W_icode = 0100 W_valE = 0 W_valM = 0 W_dstE = 0010 W_dstM = 1111 W_stat = 0
      F_stall = 1 D_stall = 1 D_bubble = 0 E_bubble = 1 set_cc = 1

130   clk=1
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 52 D_stat = 0
      decode:         d_valA = 10 d_valB = 3
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0101 M_cnd = 0 M_valA = 3 M_valE = 0 M_dstE = 1111, M_dstM = 0000 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0011 W_valE = 10 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

140   clk=0
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = 0110 D_ifun = 0000 D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 52 D_stat = 0
      decode:         d_valA = 10 d_valB = 3
      E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
      execute:        e_cnd = 0 e_valE = 0
      M Reg:          M_icode = 0101 M_cnd = 0 M_valA = 3 M_valE = 0 M_dstE = 1111, M_dstM = 0000 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0011 W_valE = 10 W_valM = 0 W_dstE = 0011 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
150   clk=1
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 10 E_valB = 3 E_valC = 0 E_dstE = 0000 E_dstM = 1111 E_srcA = 3 E_srcB = 0 E_stat = 0
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = 1111, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0101 W_valE = 0 W_valM = 3 W_dstE = 1111 W_dstM = 0000 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

160   clk=0
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = 0110 E_ifun = 0000 E_valA = 10 E_valB = 3 E_valC = 0 E_dstE = 0000 E_dstM = 1111 E_srcA = 3 E_srcB = 0 E_stat = 0
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = 1111, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0101 W_valE = 0 W_valM = 3 W_dstE = 1111 W_dstM = 0000 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

170   clk=1
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 10 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0110 M_cnd = 0 M_valA = 10 M_valE = 13 M_dstE = 0000, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0001 W_valE = 0 W_valM = 3 W_dstE = 1111 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
180   clk=0
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 10 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = 0110 M_cnd = 0 M_valA = 10 M_valE = 13 M_dstE = 0000, M_dstM = 1111 M_stat = 0
      memory:         m_valM = 3
      W Reg:          W_icode = 0001 W_valE = 0 W_valM = 3 W_dstE = 1111 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

190   clk=1
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 10 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 10 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 1
      memory:         m_valM = 3
      W Reg:          W_icode = 0110 W_valE = 13 W_valM = 3 W_dstE = 0000 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0

200   clk=0
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 10 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 10 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 1
      memory:         m_valM = 3
      W Reg:          W_icode = 0110 W_valE = 13 W_valM = 3 W_dstE = 0000 W_dstM = 1111 W_stat = 0
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0
```

```
210   clk=1
      F Reg:          F_predPC = 53
      fetch:          f_predPC = 53
      D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = 0011 D_rB = 0000 D_valC = 0 D_valP = 53 D_stat = 1
      decode:         d_valA = 10 d_valB = 0
      E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 10 E_valB = 0 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
      execute:        e_cnd = 0 e_valE = 13
      M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 10 M_valE = 13 M_dstE = 1111, M_dstM = 1111 M_stat = 1
      memory:         m_valM = 3
      W Reg:          W_icode = xxxx W_valE = 13 W_valM = 3 W_dstE = 1111 W_dstM = 1111 W_stat = 1
      F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0
```

Register file before running the pipe.v:

reg_file.txt

```
1   // 0x00000000
2   0000000000000000
3   0000000000000000
4   0000000000000000
5   0000000000000000
6   0000000000000000
7   0000000000000000
8   0000000000000000
9   0000000000000000
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

Register file after running the pipe.v

reg_file.txt

```
1   // 0x00000000
2   000000000000000d
3   0000000000000003
4   0000000000000080
5   000000000000000a
6   0000000000000000
7   0000000000000000
8   0000000000000000
9   0000000000000000
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

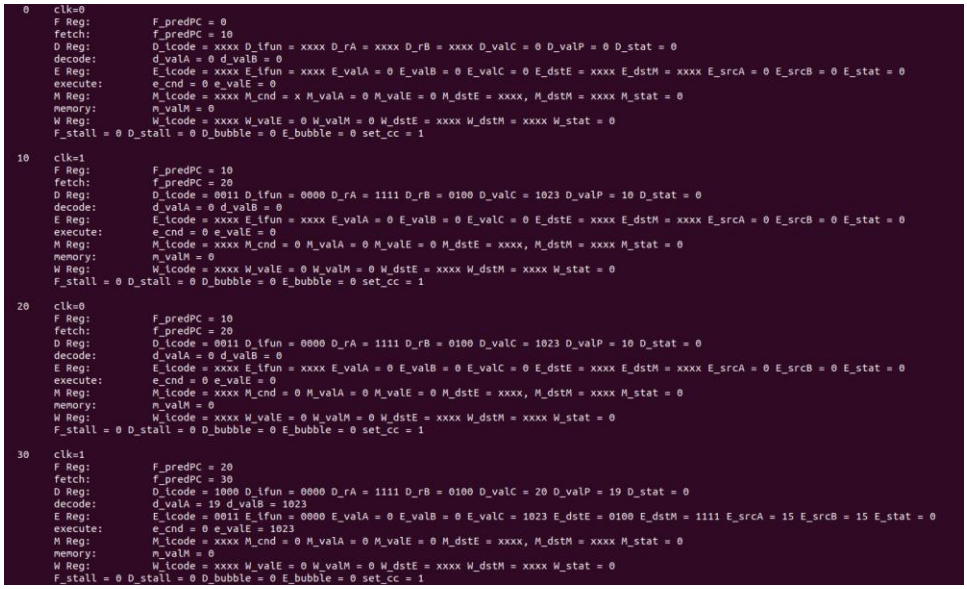
Gtkwave output:

2st set of instructions given to the Processor:

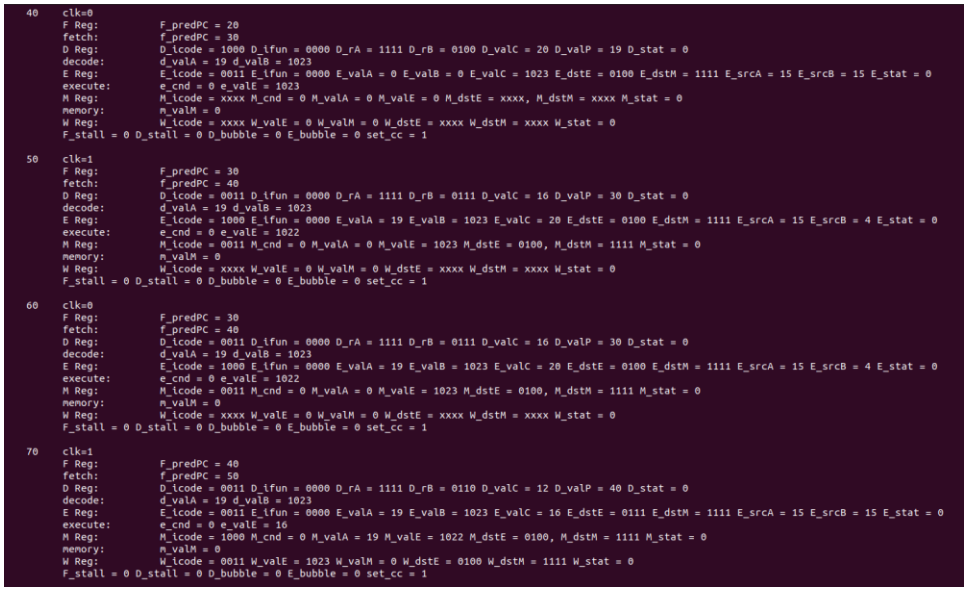
```
irmovq stack %rsp
call main
halt
main:
irmovq $0x10 %rdi
irmovq $0xc %rsi
call gcd
```

```
ret

gcd(%rdx,%rbx)

swap:

        pushq %rdi

        Pushq %rsi

        popq

        Popq

        ret
```

Output of 2st set of instruction:



```
  0    clk=0
       F Reg:          F_predPC = 0
       fetch:          f_predPC = 10
       D Reg:          D_icode = xxxx D_ifun = xxxx D_rA = xxxx D_rB = xxxx D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = x M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 10    clk=1
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0100 D_valC = 1023 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 20    clk=0
       F Reg:          F_predPC = 10
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0100 D_valC = 1023 D_valP = 10 D_stat = 0
       decode:         d_valA = 0 d_valB = 0
       E Reg:          E_icode = xxxx E_ifun = xxxx E_valA = 0 E_valB = 0 E_valC = 0 E_dstE = xxxx E_dstM = xxxx E_srcA = 0 E_srcB = 0 E_stat = 0
       execute:        e_cnd = 0 e_valE = 0
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 30    clk=1
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 1000 D_ifun = 0000 D_rA = 1111 D_rB = 0100 D_valC = 20 D_valP = 19 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1023 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```



```
 40    clk=0
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 1000 D_ifun = 0000 D_rA = 1111 D_rB = 0100 D_valC = 20 D_valP = 19 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 0 E_valB = 0 E_valC = 1023 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = xxxx M_cnd = 0 M_valA = 0 M_valE = 0 M_dstE = xxxx, M_dstM = xxxx M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 50    clk=1
       F Reg:          F_predPC = 30
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0111 D_valC = 16 D_valP = 30 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 1000 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 20 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1023 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 60    clk=0
       F Reg:          F_predPC = 30
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0111 D_valC = 16 D_valP = 30 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 1000 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 20 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 0 M_valE = 1023 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = xxxx W_valE = 0 W_valM = 0 W_dstE = xxxx W_dstM = xxxx W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 70    clk=1
       F Reg:          F_predPC = 40
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 12 D_valP = 40 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 16 E_dstE = 0111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 10
       M Reg:          M_icode = 1000 M_cnd = 0 M_valA = 19 M_valE = 1022 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 1023 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
 80    clk=0
       F Reg:          F_predPC = 40
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 12 D_valP = 40 D_stat = 0
       decode:         d_valA = 19 d_valB = 1023
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 16 E_dstE = 0111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 16
       M Reg:          M_icode = 1000 M_cnd = 0 M_valA = 19 M_valE = 1022 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 1023 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

 90    clk=1
       F Reg:          F_predPC = 50
       fetch:          f_predPC = 52
       D Reg:          D_icode = 1000 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 50 D_valP = 49 D_stat = 0
       decode:         d_valA = 49 d_valB = 1022
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 12 E_dstE = 0110 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 12
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 19 M_valE = 16 M_dstE = 0111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1000 W_valE = 1022 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

100    clk=0
       F Reg:          F_predPC = 50
       fetch:          f_predPC = 52
       D Reg:          D_icode = 1000 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 50 D_valP = 49 D_stat = 0
       decode:         d_valA = 49 d_valB = 1022
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 19 E_valB = 1023 E_valC = 12 E_dstE = 0110 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 12
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 19 M_valE = 16 M_dstE = 0111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1000 W_valE = 1022 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

110    clk=1
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 54
       D Reg:          D_icode = 1010 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 52 D_stat = 0
       decode:         d_valA = 16 d_valB = 1021
       E Reg:          E_icode = 1000 E_ifun = 0000 E_valA = 49 E_valB = 1022 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1021
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 19 M_valE = 12 M_dstE = 0110, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 16 W_valM = 0 W_dstE = 0111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
120    clk=0
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 54
       D Reg:          D_icode = 1010 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 52 D_stat = 0
       decode:         d_valA = 16 d_valB = 1021
       E Reg:          E_icode = 1000 E_ifun = 0000 E_valA = 49 E_valB = 1022 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 15 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1021
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 19 M_valE = 12 M_dstE = 0110, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 16 W_valM = 0 W_dstE = 0111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

130    clk=1
       F Reg:          F_predPC = 54
       fetch:          f_predPC = 56
       D Reg:          D_icode = 1010 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 54 D_stat = 0
       decode:         d_valA = 12 d_valB = 1020
       E Reg:          E_icode = 1010 E_ifun = 0000 E_valA = 16 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 7 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1020
       M Reg:          M_icode = 1000 M_cnd = 0 M_valA = 49 M_valE = 1021 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 12 W_valM = 0 W_dstE = 0110 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

140    clk=0
       F Reg:          F_predPC = 54
       fetch:          f_predPC = 56
       D Reg:          D_icode = 1010 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 54 D_stat = 0
       decode:         d_valA = 12 d_valB = 1020
       E Reg:          E_icode = 1010 E_ifun = 0000 E_valA = 16 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 7 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1020
       M Reg:          M_icode = 1000 M_cnd = 0 M_valA = 49 M_valE = 1021 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 0011 W_valE = 12 W_valM = 0 W_dstE = 0110 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

150    clk=1
       F Reg:          F_predPC = 56
       fetch:          f_predPC = 58
       D Reg:          D_icode = 1011 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 56 D_stat = 0
       decode:         d_valA = 1019 d_valB = 1019
       E Reg:          E_icode = 1010 E_ifun = 0000 E_valA = 12 E_valB = 1020 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 6 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1019
       M Reg:          M_icode = 1010 M_cnd = 0 M_valA = 16 M_valE = 1020 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1000 W_valE = 1021 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
160    clk=0
       F Reg:          F_predPC = 56
       fetch:          f_predPC = 58
       D Reg:          D_icode = 1011 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 56 D_stat = 0
       decode:         d_valA = 1019 d_valB = 1019
       E Reg:          E_icode = 1010 E_ifun = 0000 E_valA = 12 E_valB = 1020 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 6 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1019
       M Reg:          M_icode = 1010 M_cnd = 0 M_valA = 16 M_valE = 1020 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1000 W_valE = 1021 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

170    clk=1
       F Reg:          F_predPC = 58
       fetch:          f_predPC = 59
       D Reg:          D_icode = 1011 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 58 D_stat = 0
       decode:         d_valA = 1020 d_valB = 1020
       E Reg:          E_icode = 1011 E_ifun = 0000 E_valA = 1019 E_valB = 1019 E_valC = 50 E_dstE = 0100 E_dstM = 0111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1020
       M Reg:          M_icode = 1010 M_cnd = 0 M_valA = 12 M_valE = 1019 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1010 W_valE = 1020 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

180    clk=0
       F Reg:          F_predPC = 58
       fetch:          f_predPC = 59
       D Reg:          D_icode = 1011 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 58 D_stat = 0
       decode:         d_valA = 1020 d_valB = 1020
       E Reg:          E_icode = 1011 E_ifun = 0000 E_valA = 1019 E_valB = 1019 E_valC = 50 E_dstE = 0100 E_dstM = 0111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1020
       M Reg:          M_icode = 1010 M_cnd = 0 M_valA = 12 M_valE = 1019 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 0
       W Reg:          W_icode = 1010 W_valE = 1020 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

190    clk=1
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 1001 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 59 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1011 E_ifun = 0000 E_valA = 1020 E_valB = 1020 E_valC = 50 E_dstE = 0100 E_dstM = 0110 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1021
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1019 M_valE = 1020 M_dstE = 0100, M_dstM = 0111 M_stat = 0
       memory:         m_valM = 12
       W Reg:          W_icode = 1010 W_valE = 1019 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1
```

```
200    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 1001 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 59 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1011 E_ifun = 0000 E_valA = 1020 E_valB = 1020 E_valC = 50 E_dstE = 0100 E_dstM = 0110 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1021
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1019 M_valE = 1020 M_dstE = 0100, M_dstM = 0111 M_stat = 0
       memory:         m_valM = 12
       W Reg:          W_icode = 1010 W_valE = 1019 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

210    clk=1
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1020 M_valE = 1021 M_dstE = 0100, M_dstM = 0110 M_stat = 0
       memory:         m_valM = 16
       W Reg:          W_icode = 1011 W_valE = 1020 W_valM = 12 W_dstE = 0100 W_dstM = 0111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

220    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1020 M_valE = 1021 M_dstE = 0100, M_dstM = 0110 M_stat = 0
       memory:         m_valM = 16
       W Reg:          W_icode = 1011 W_valE = 1020 W_valM = 12 W_dstE = 0100 W_dstM = 0111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1
```

```
200    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 1001 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 59 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1011 E_ifun = 0000 E_valA = 1020 E_valB = 1020 E_valC = 50 E_dstE = 0100 E_dstM = 0110 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1021
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1019 M_valE = 1020 M_dstE = 0100, M_dstM = 0111 M_stat = 0
       memory:         m_valM = 12
       W Reg:          W_icode = 1010 W_valE = 1019 W_valM = 0 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

210    clk=1
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1020 M_valE = 1021 M_dstE = 0100, M_dstM = 0110 M_stat = 0
       memory:         m_valM = 16
       W Reg:          W_icode = 1011 W_valE = 1020 W_valM = 12 W_dstE = 0100 W_dstM = 0111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

220    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1011 M_cnd = 0 M_valA = 1020 M_valE = 1021 M_dstE = 0100, M_dstM = 0110 M_stat = 0
       memory:         m_valM = 16
       W Reg:          W_icode = 1011 W_valE = 1020 W_valM = 12 W_dstE = 0100 W_dstM = 0111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1
```

```
230    clk=1
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 1011 W_valE = 1021 W_valM = 16 W_dstE = 0100 W_dstM = 0110 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

240    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 59
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 1001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 1011 W_valE = 1021 W_valM = 16 W_dstE = 0100 W_dstM = 0110 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

250    clk=1
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 1001 W_valE = 1022 W_valM = 49 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
260    clk=0
       F Reg:          F_predPC = 59
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1021 d_valB = 1021
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 1001 W_valE = 1022 W_valM = 49 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

270    clk=1
       F Reg:          F_predPC = 50
       fetch:          f_predPC = 52
       D Reg:          D_icode = 1001 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 50 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

280    clk=0
       F Reg:          F_predPC = 50
       fetch:          f_predPC = 52
       D Reg:          D_icode = 1001 D_ifun = 0000 D_rA = 0110 D_rB = 1111 D_valC = 50 D_valP = 50 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1021 E_valB = 1021 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1022
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1
```

```
290    clk=1
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 54
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

300    clk=0
       F Reg:          F_predPC = 52
       fetch:          f_predPC = 54
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 1001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 50 E_dstE = 0100 E_dstM = 1111 E_srcA = 4 E_srcB = 4 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1021 M_valE = 1022 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 49
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

310    clk=1
       F Reg:          F_predPC = 54
       fetch:          f_predPC = 56
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 1001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1
```

```
320    clk=0
       F Reg:          F_predPC = 54
       fetch:          f_predPC = 56
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 1001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 0100, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1022 W_valM = 49 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 1 D_stall = 0 D_bubble = 1 E_bubble = 0 set_cc = 1

330    clk=1
       F Reg:          F_predPC = 56
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 1001 W_valE = 1023 W_valM = 19 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

340    clk=0
       F Reg:          F_predPC = 56
       fetch:          f_predPC = 20
       D Reg:          D_icode = 0001 D_ifun = 0000 D_rA = 0000 D_rB = 0000 D_valC = 0 D_valP = 0 D_stat = 0
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 1001 W_valE = 1023 W_valM = 19 W_dstE = 0100 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1
```

```
350    clk=1
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 20 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

360    clk=0
       F Reg:          F_predPC = 20
       fetch:          f_predPC = 30
       D Reg:          D_icode = 0000 D_ifun = 0000 D_rA = 0111 D_rB = 1111 D_valC = 50 D_valP = 20 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0001 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 0 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 0
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 1

370    clk=1
       F Reg:          F_predPC = 30
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0111 D_valC = 16 D_valP = 30 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0000 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 50 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0
```

```
380    clk=0
       F Reg:          F_predPC = 30
       fetch:          f_predPC = 40
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0111 D_valC = 16 D_valP = 30 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0000 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 50 E_dstE = 1111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 0 e_valE = 1023
       M Reg:          M_icode = 0001 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 0
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0

390    clk=1
       F Reg:          F_predPC = 40
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 12 D_valP = 40 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 16 E_dstE = 0111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 0 e_valE = 16
       M Reg:          M_icode = 0000 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0

400    clk=0
       F Reg:          F_predPC = 40
       fetch:          f_predPC = 50
       D Reg:          D_icode = 0011 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 12 D_valP = 40 D_stat = 1
       decode:         d_valA = 1022 d_valB = 1022
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 16 E_dstE = 0111 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 0 e_valE = 16
       M Reg:          M_icode = 0000 M_cnd = 0 M_valA = 1022 M_valE = 1023 M_dstE = 1111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 19
       W Reg:          W_icode = 0001 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 0
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0
```

```
410    clk=1
       F Reg:          F_predPC = 50
       fetch:          f_predPC = 52
       D Reg:          D_icode = 1000 D_ifun = 0000 D_rA = 1111 D_rB = 0110 D_valC = 50 D_valP = 49 D_stat = 1
       decode:         d_valA = 49 d_valB = 1023
       E Reg:          E_icode = 0011 E_ifun = 0000 E_valA = 1022 E_valB = 1022 E_valC = 12 E_dstE = 0110 E_dstM = 1111 E_srcA = 15 E_srcB = 15 E_stat = 1
       execute:        e_cnd = 0 e_valE = 12
       M Reg:          M_icode = 0011 M_cnd = 0 M_valA = 1022 M_valE = 16 M_dstE = 0111, M_dstM = 1111 M_stat = 1
       memory:         m_valM = 19
       W Reg:          W_icode = 0000 W_valE = 1023 W_valM = 19 W_dstE = 1111 W_dstM = 1111 W_stat = 1
       F_stall = 0 D_stall = 0 D_bubble = 0 E_bubble = 0 set_cc = 0
```

Register file before running the pipe.v:

reg_file.txt

```
1   // 0x00000000
2   0000000000000000
3   0000000000000000
4   0000000000000000
5   0000000000000000
6   0000000000000000
7   0000000000000000
8   0000000000000000
9   0000000000000000
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

Register file after running the pipe.v

Project > PIPE > reg_file.txt

```
1   // 0x00000000
2   0000000000000000
3   0000000000000000
4   0000000000000000
5   0000000000000000
6   00000000000003ff
7   0000000000000000
8   0000000000000010
9   000000000000000c
10  0000000000000000
11  0000000000000000
12  0000000000000000
13  0000000000000000
14  0000000000000000
15  0000000000000000
16  0000000000000000
```

Command to run the code for pipelined processor:

iverilog –o pipe pipe.v

vvp pipe

Processor Features

Processor Frequency:

The frequency of the processor is measure as follows:

$$f = \frac{1}{T_{clk}} Hz$$

Where T_{clk} is the time period of the clock cycle. We have taken T_{clk} = 1ns

$$f = \frac{1}{1 \times 10^{-9}} Hz = 10^{9} Hz = 1GHz$$

Memory Initialization:

The data memory and instruction memory are separated. The instruction memory has 256-32 bits words, that is , 1024 bytes. This is equivalent to 1kB of instruction memory. While the data memory had 1024-64 bits which is 8kB od data memory.

ALU DESIGN

An arithmetic-logic unit is the part of CPU that carries out arithmetic and logical operations on the operands in computer instruction words.

We designed an ALU that performs four different arithmetic and logical operations on 64-bit operands: ADD, SUB, AND, XOR. This ALU takes two 64-bit operands and the operations to be performed as its inputs and returns a 64-bit output and an overflow bit (in case of ADD and SUB).

We have created three different modules for the specified operations (ADD and SUB are performed using the same module). A separate test bench has been written to test the functioning of each module separately.

ALU:

A wrapper ALU unit was created from where the ADD, SUBTRACT, AND, and XOR modules are called based on the control input. The ALU unit takes as input the control signal, and two 64-bit inputs, and returns the 64-bit output corresponding to the control signal chosen.

Control 0 - ADD x and y

Control 1 – Subtract y from x

Control 2 – AND x and y

Control 3 – XOR x and y

Every time an input is given, all the four operations add, subtract, AND, XOR are computed and based on the control signal the required value is given as output.

COMMANDS TO RUN THE FILES:

- iverilog -o alu_tb alu_tb.v alu.v
- vvp alu_tb
- gtkwave alu_tb.vcd

Source Code:

Project > ALU > alu.v

```
1   `include "ADDSUB/addsub.v"
2   `include "AND/and.v"
3   `include "XOR/xor.v"
4
5   module alu (inp1, inp2, op, out, overflow);
6       input signed [63:0] inp1, inp2;
7       input  [1:0] op;
8       output signed [63:0] out;
9       output overflow;
10
11      wire signed [63:0] out_add, out_sub, out_and, out_xor;
12      reg signed [63:0] ans;
13      reg overflow_;
14
15      addsub_ add (inp1, inp2, 1'b0, out_add, overflow1);
16      addsub_ sub (inp1, inp2, 1'b1, out_sub, overflow2);
17      and_ a1 (inp1, inp2, out_and);
18      xor_ x1 (inp1, inp2, out_xor);
19
20      always@(*)
21      begin
22          if(op==2'b00)
23          begin
24              ans = out_add;
25              overflow_ = overflow1;
26          end
27          else if(op == 2'b01)
28          begin
29              ans = out_sub;
30              overflow_ = overflow2;
31          end
32          else if(op == 2'b10)
33              ans = out_and;
34          else if(op == 2'b11)
35              ans = out_xor;
36      end
37
38      assign out = ans;
39      assign overflow = overflow_;
40
41  endmodule
```

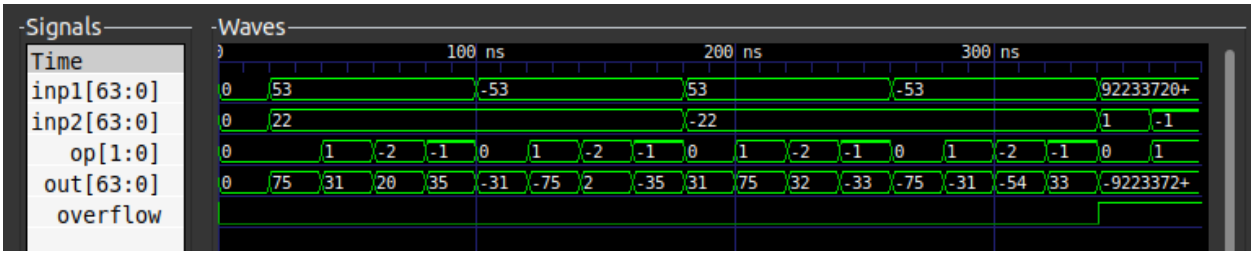
The output produced for some input combinations:

```
VCD info: dumpfile alu_tb.vcd opened for output.
                   0 Operation = 00
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     out  = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     OF = 0

                  20 Operation = 00
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000001001011 =                   75
                     OF = 0

                  40 Operation = 01
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000011111 =                   31
                     OF = 0

                  60 Operation = 10
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000010100 =                   20
                     OF = 0

                  80 Operation = 11
                     inp1 = 00000000000000000000000000000000000000000000000000000000000000110101 =                   53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000100011 =                   35
                     OF = 0

                 100 Operation = 00
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111111100001 =                  -31
                     OF = 0

                 120 Operation = 01
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111110110101 =                  -75
                     OF = 0

                 140 Operation = 10
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 00000000000000000000000000000000000000000000000000000000000000000010 =                    2
                     OF = 0

                 160 Operation = 11
                     inp1 = 11111111111111111111111111111111111111111111111111111111111111001011 =                  -53
                     inp2 = 00000000000000000000000000000000000000000000000000000000000000010110 =                   22
                     out  = 11111111111111111111111111111111111111111111111111111111111111011101 =                  -35
                     OF = 0
```

```
180 Operation = 00
    inp1 = 00000000000000000000000000000000000000000000000000000000000110101 =                  53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 00000000000000000000000000000000000000000000000000000000000011111 =                  31
    OF = 0

200 Operation = 01
    inp1 = 00000000000000000000000000000000000000000000000000000000000110101 =                  53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 00000000000000000000000000000000000000000000000000000000001001011 =                  75
    OF = 0

220 Operation = 10
    inp1 = 00000000000000000000000000000000000000000000000000000000000110101 =                  53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 00000000000000000000000000000000000000000000000000000000000100000 =                  32
    OF = 0

240 Operation = 11
    inp1 = 00000000000000000000000000000000000000000000000000000000000110101 =                  53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 11111111111111111111111111111111111111111111111111111111111011111 =                 -33
    OF = 0

260 Operation = 00
    inp1 = 11111111111111111111111111111111111111111111111111111111111001011 =                 -53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 11111111111111111111111111111111111111111111111111111111110110101 =                 -75
    OF = 0

280 Operation = 01
    inp1 = 11111111111111111111111111111111111111111111111111111111111001011 =                 -53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 11111111111111111111111111111111111111111111111111111111111100001 =                 -31
    OF = 0

300 Operation = 10
    inp1 = 11111111111111111111111111111111111111111111111111111111111001011 =                 -53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 11111111111111111111111111111111111111111111111111111111111001010 =                 -54
    OF = 0

320 Operation = 11
    inp1 = 11111111111111111111111111111111111111111111111111111111111001011 =                 -53
    inp2 = 11111111111111111111111111111111111111111111111111111111111101010 =                 -22
    out  = 00000000000000000000000000000000000000000000000000000000000100001 =                  33
    OF = 0

340 Operation = 00
    inp1 = 01111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
    inp2 = 00000000000000000000000000000000000000000000000000000000000000001 =                    1
    out  = 10000000000000000000000000000000000000000000000000000000000000000 = -9223372036854775808
    OF = 1
```

```
360 Operation = 01
    inp1 = 01111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
    inp2 = 11111111111111111111111111111111111111111111111111111111111111111 =                   -1
    out  = 10000000000000000000000000000000000000000000000000000000000000000 = -9223372036854775808
    OF = 1
```

The gtk waveforms for these input combinations:



ADDITION AND SUBTRACT OPERATION:

The addition and subtraction operations were performed with the help of Full Adders. A single Full Adder performs the addition of two one-bit numbers and the carry input. For performing the addition of binary numbers with more than one bit, more than one full adder is required in parallel, and the number of Full Adders depends on the number of bits, that is, we implement a ripple adder.

Adder:

By connecting 64 full adders in parallel, a 64-bit Parallel Adder can be constructed.

Subtractor:

A 64-bit parallel subtractor can be implemented using 64 full adders. The subtraction operation is performed by considering the principle that the addition of minuend and the complement of the subtrahend is equivalent to the subtraction process. We know that the subtraction of A by B is obtained by taking 2's complement of B and adding it to A. The 2's complement of B is obtained by taking 1's complement and adding 1 to the least significant pair of bits. Hence, we can obtain the 1's complement of B with the inverters and a 1 can be added to the sum through the input carry.

Implementation of addition and subtraction in one module:

The operations of both addition and subtraction can be performed by one common binary adder. Such a binary circuit can be designed by adding an XOR gate with each full adder. The mode input control line M relates to the carry input of the least significant bit of the full adder. This control line decides the type of operation, whether addition or subtraction.

When M = 1, the circuit is a subtractor, and when M=0, the circuit becomes an adder. The XOR gate consists of two inputs to which one is connected to the B and the other to input M.

When M = 0, B XOR 0 produces B. Hence, Addition operation is performed.

When M = 1, B XOR of 1 produces complement of B and the carry is 1. Hence the complemented B inputs are added to A and 1 is added through the input carry (2's complement operation). Therefore, the subtraction operation is performed.

We created a module named "addsub_" to perform the addition and subtraction operations for 64-bit inputs. This module takes two 64-bits numbers as input and returns a 64-bit number as output.

This was implemented by creating a full adder module and using the module within a 'for' loop which runs 64 times for all the 64 bits while specifying the control line input 'M' that decides the type of operation.

- Overflow:

When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow. An overflow may occur if the two numbers added are both positive or

both negative. An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow occurs. If the two carries are applied to an XOR gate, an overflow is detected when the output of the gate is equal to 1.

Commands to run the files:

- iverilog -o addsub_tb addsub_tb.v addsub.v
- vvp addsub_tb
- gtkwave addsub_tb.vcd

Source Code:

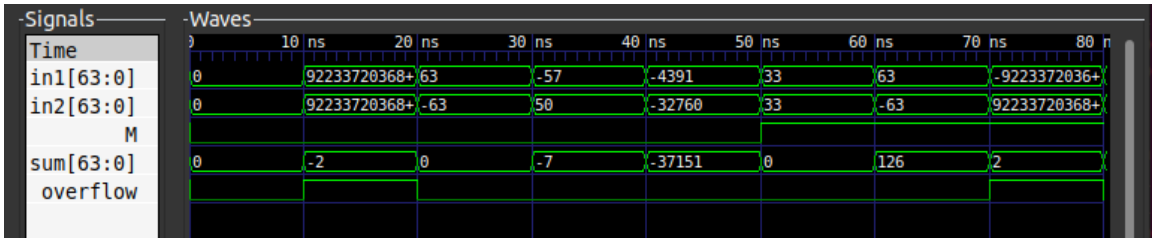
Project > ALU > ADDSUB > addsub.v

```
// Implementation of a 64 bit Adder and Subtractor using 64 1-bit Adders

module FA(a, b, c, M, sum, carry);

input a, b, c;
input M;
output sum, carry;
wire B, x, y, z;

xor x1(B, b, M);

xor x2(x, a, B);
xor x3(sum, x, c);
and a1(y, a, B);
and a2(z, x, c);
or o1(carry, y, z);

endmodule


module addsub (in1, in2, M, sum, overflow);

input [63:0]in1;
input [63:0]in2;
input M;

output [63:0]sum;
output overflow;

wire [64:0]C;

assign C[0] = M;

genvar i;

generate
    for(i = 0; i < 64; i=i+1)
    begin
        FA full_adder(in1[i], in2[i], C[i], M, sum[i], C[i+1]);
    end
endgenerate


xor x1(overflow, C[64], C[63]);

endmodule

```

The output produced for some input combinations:

```
VCD info: dumpfile addsub_tb.vcd opened for output.
                   0 in1 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     in2 = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                     M = 0
                     SUM = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                    overflow = 0

                  10 in1 = 01111111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
                     in2 = 01111111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
                     M = 0
                     SUM = 11111111111111111111111111111111111111111111111111111111111111111110 =                   -2
                    overflow = 1

                  20 in1 = 00000000000000000000000000000000000000000000000000000000000000111111 =                   63
                     in2 = 11111111111111111111111111111111111111111111111111111111111111000001 =                  -63
                     M = 0
                     SUM = 00000000000000000000000000000000000000000000000000000000000000000000 =                    0
                    overflow = 0

                  30 in1 = 11111111111111111111111111111111111111111111111111111111111111000111 =                  -57
                     in2 = 00000000000000000000000000000000000000000000000000000000000000110010 =                   50
                     M = 0
                     SUM = 11111111111111111111111111111111111111111111111111111111111111111001 =                   -7
                    overflow = 0

                  40 in1 = 11111111111111111111111111111111111111111111111111111110110111011001 =                -4391
                     in2 = 11111111111111111111111111111111111111111111111110000000000001000000 =               -32760
                     M = 0
                     SUM = 11111111111111111111111111111111111111111111111101101110111100001 =               -37151
                    overflow = 0

                  50 in1 = 00000000000000000000000000000000000000000000000000000000000100001 =                   33
                     in2 = 00000000000000000000000000000000000000000000000000000000000100001 =                   33
                     M = 1
                     SUM = 00000000000000000000000000000000000000000000000000000000000000000 =                    0
                    overflow = 0

                  60 in1 = 00000000000000000000000000000000000000000000000000000000000111111 =                   63
                     in2 = 11111111111111111111111111111111111111111111111111111111111000001 =                  -63
                     M = 1
                     SUM = 00000000000000000000000000000000000000000000000000000000001111110 =                  126
                    overflow = 0

                  70 in1 = 10000000000000000000000000000000000000000000000000000000000000000001 = -9223372036854775807
                     in2 = 01111111111111111111111111111111111111111111111111111111111111111111 =  9223372036854775807
                     M = 1
                     SUM = 00000000000000000000000000000000000000000000000000000000000000000010 =                    2
                    overflow = 1

                  80 in1 = 11111111111111111111111111111111111111111111111111111110110111011001 =                -4391
                     in2 = 11111111111111111111111111111111111111111111111110000000000001000000 =               -32760
                     M = 1
                     SUM = 00000000000000000000000000000000000000000000000110111011010001 =                   28369
                    overflow = 0
```

The gtk waveforms for these input combinations:



AND OPERATION:

We created a module named “and_” to perform AND operation for 64-bits input. This module takes two 64-bits numbers as input and returns a 64-bit number as output who's each bits the AND of corresponding bits of the two inputs. We wrote a for loop that runs 64 times and AND’s all the bits and stores the result in the corresponding output bit.

Commands to run the files:

- iverilog -o and_tb and_tb.v and.v
- vvp and_tb
- gtkwave and_tb.vcd

Source Code

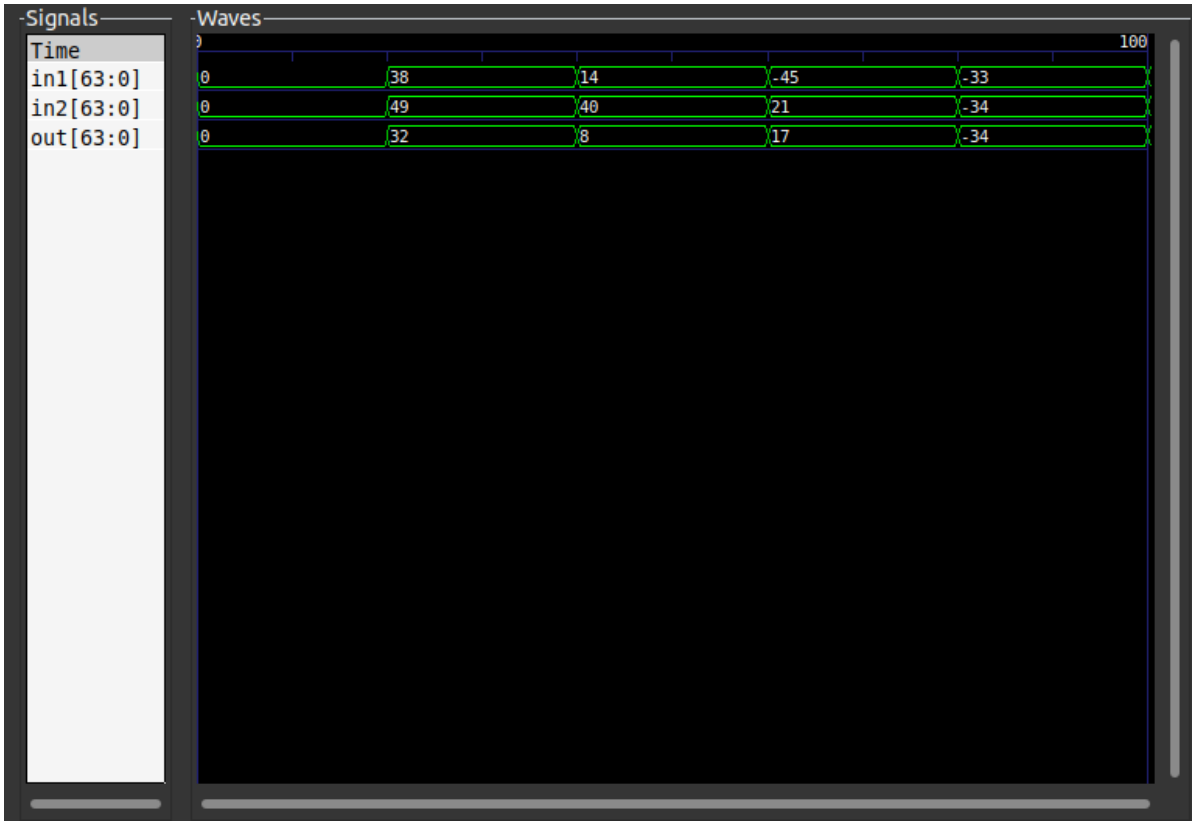
ALU > AND > and.v

```
1   module and_ (in1, in2, out);
2
3       input  [63:0] in1, in2;
4       output [63:0] out;
5
6       genvar i;
7       generate
8           for (i = 0; i < 64; i = i + 1) begin
9               and(out[i], in1[i], in2[i]);
10          end
11      endgenerate
12
13  endmodule
```

The output produced for some input combinations:

```
VCD info: dumpfile and_tb.vcd opened for output.
                   0 in1 = 0000000000000000000000000000000000000000000000000000000000000000
                     in2 = 0000000000000000000000000000000000000000000000000000000000000000
                     out = 0000000000000000000000000000000000000000000000000000000000000000

                  20 in1 = 0000000000000000000000000000000000000000000000000000000000100110
                     in2 = 0000000000000000000000000000000000000000000000000000000000110001
                     out = 0000000000000000000000000000000000000000000000000000000000100000

                  40 in1 = 0000000000000000000000000000000000000000000000000000000000001110
                     in2 = 0000000000000000000000000000000000000000000000000000000000101000
                     out = 0000000000000000000000000000000000000000000000000000000000001000

                  60 in1 = 1111111111111111111111111111111111111111111111111111111111010011
                     in2 = 0000000000000000000000000000000000000000000000000000000000010101
                     out = 0000000000000000000000000000000000000000000000000000000000010001

                  80 in1 = 1111111111111111111111111111111111111111111111111111111111011111
                     in2 = 1111111111111111111111111111111111111111111111111111111111011110
                     out = 1111111111111111111111111111111111111111111111111111111111011110

                 100 in1 = 0000000000000000000000000000000000000000000000000000000000101111
                     in2 = 0000000000000000000000000000000000000000000000000000000000111001
                     out = 0000000000000000000000000000000000000000000000000000000000101001
```

The gtk waveforms for these input combinations:



XOR OPERATION:

We created a module named “xor_” to perform XOR operation for 64-bits input. This module takes two 64-bits numbers as input and returns a 64-bit number as output who's each bits the XOR of corresponding bits of the two inputs. We wrote a for loop that runs 64 times and XOR's all the bits and stores the result in the corresponding output bit.

Commands to run the files:

- iverilog -o xor_tb xor_tb.v xor.v
- vvp xor_tb
- gtkwave xor_tb.vcd

Source Code:

ALU > XOR > xor.v

```
1   module xor_ (in1, in2, out);
2
3       input  [63:0] in1, in2;
4       output [63:0] out;
5
6       genvar i;
7       generate
8           for (i = 0; i < 64; i = i + 1) begin
9               xor(out[i], in1[i], in2[i]);
10          end
11      endgenerate
12
13  endmodule
```

The output produced for some input combinations:

```
VCD info: dumpfile xor_tb.vcd opened for output.
                   0 in1 = 0000000000000000000000000000000000000000000000000000000000000000
                     in2 = 0000000000000000000000000000000000000000000000000000000000000000
                     out = 0000000000000000000000000000000000000000000000000000000000000000

                  20 in1 = 0000000000000000000000000000000000000000000000000000000000100110
                     in2 = 0000000000000000000000000000000000000000000000000000000000110001
                     out = 0000000000000000000000000000000000000000000000000000000000010111

                  40 in1 = 0000000000000000000000000000000000000000000000000000000000001110
                     in2 = 0000000000000000000000000000000000000000000000000000000000101000
                     out = 0000000000000000000000000000000000000000000000000000000000100110

                  60 in1 = 1111111111111111111111111111111111111111111111111111111111111111
                     in2 = 0000000000000000000000000000000000000000000000000000000000010101
                     out = 1111111111111111111111111111111111111111111111111111111111000110

                  80 in1 = 1111111111111111111111111111111111111111111111111111111111111111
                     in2 = 1111111111111111111111111111111111111111111111111111111111111110
                     out = 0000000000000000000000000000000000000000000000000000000000000001

                 100 in1 = 0000000000000000000000000000000000000000000000000000000000101111
                     in2 = 0000000000000000000000000000000000000000000000000000000000111001
                     out = 0000000000000000000000000000000000000000000000000000000000010110
```

The gtk waveforms for these input combinations: