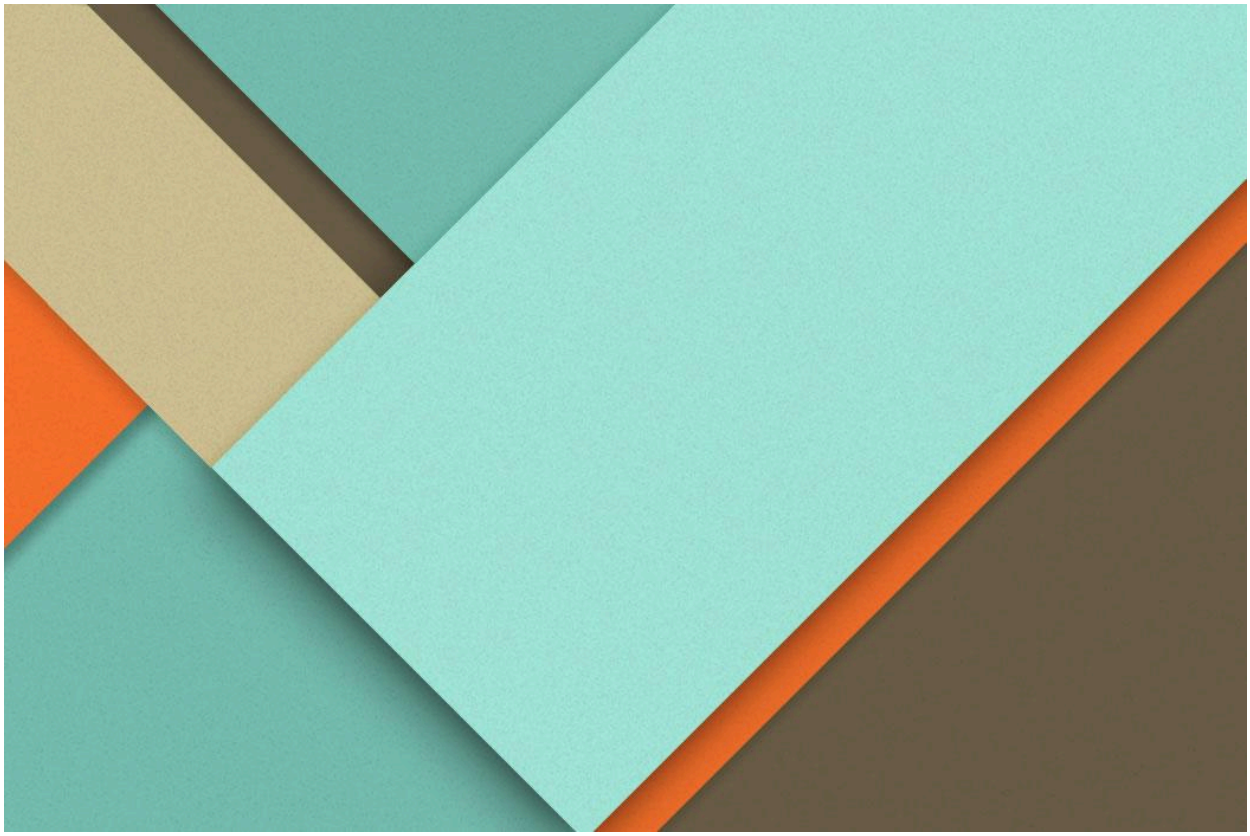


Y86-64

Introduction to Processor Architecture

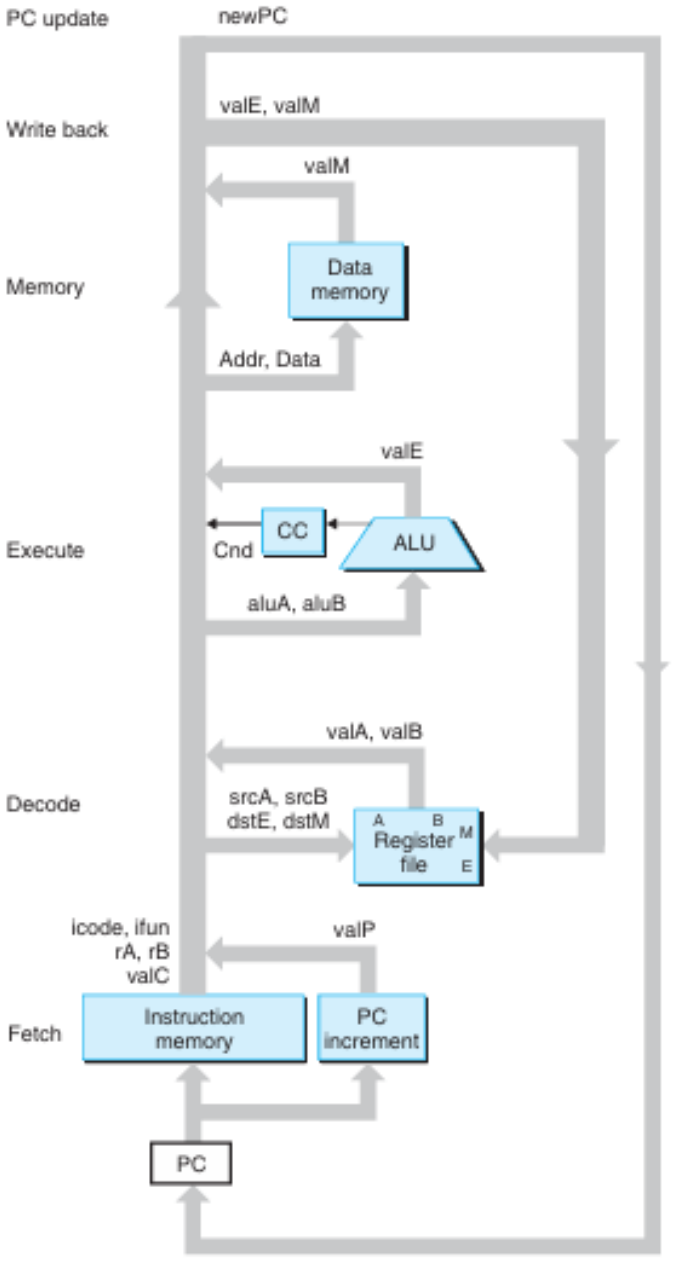
Saumya Balina
2022102069

Meemansa Pandey
2022102036

SEQ Implementation

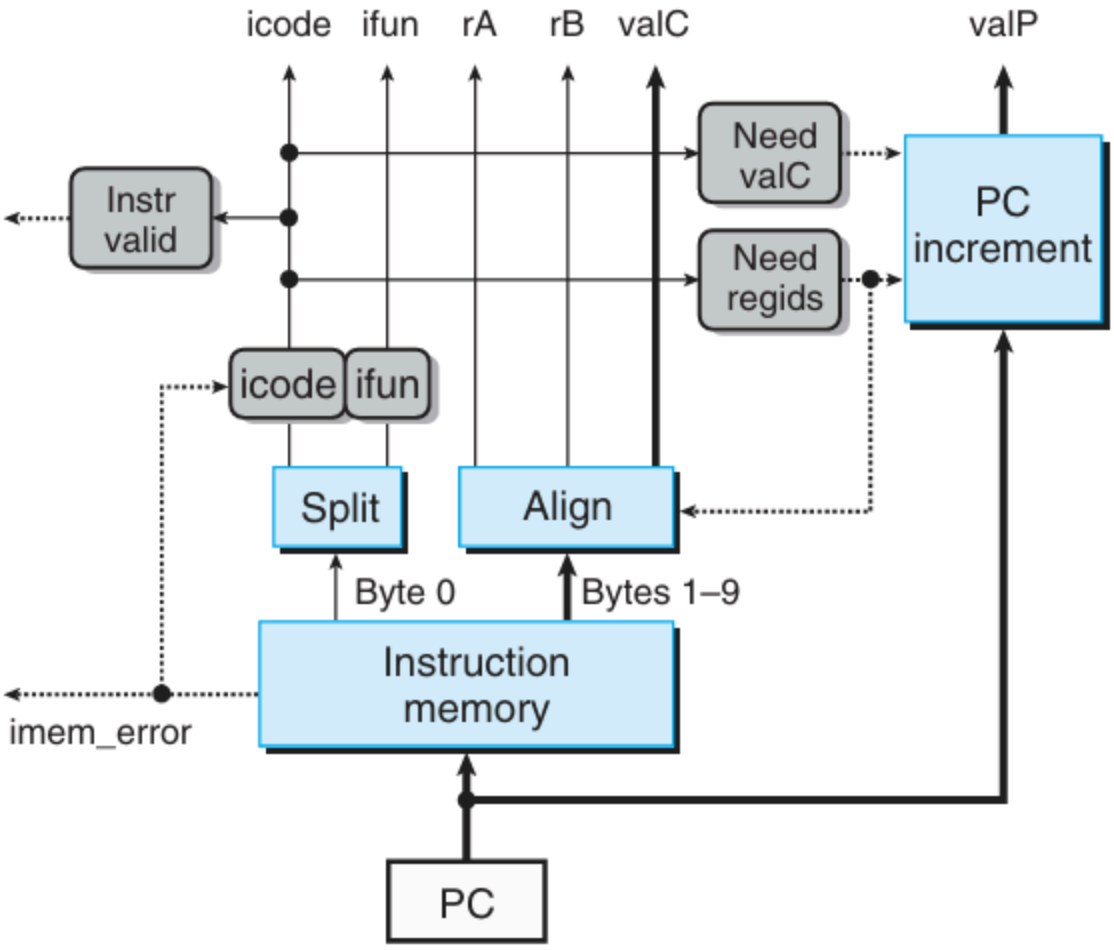
A sequential processor is a type of computer architecture that executes instructions sequentially, one after another.

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, hence it has a longer clock period.



FETCH

The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address. From the instruction it extracts the two 4-bit portions of the instruction specifier byte - icode (the instruction code) and ifun (the instruction function). It possibly fetches a register specifier byte, giving one or both of the register operand specifiers rA and rB. It also possibly fetches an 8-byte constant word valC. It computes valP to be the address of the instruction following the current one in sequential order. Thus, valP equals the value of the PC plus the length of the fetched instruction.



Code

```
module fetch(
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]valP,
output reg[63:0]valC,
output reg mem_error,
output reg i_error,
output reg halt
);

reg [7:0]mem[0:1023];
reg [0:79]inst;
// 4 bits -> icode inst[0:3]
// 4 bits -> ifun inst[4:7]
// 4 bits -> rA inst[8:11]
// 4 bits -> rB inst[12:15]
// 64 bits -> Constant Value[16:79]

initial
begin
// FILE INPUT
$readmemb("1.txt",mem);
end


always @(posedge clk)
begin

    if(PC>=0 && PC<1024)
    begin

        mem_error = 0;
        halt = 0;
        icode = 4'hf;
        ifun = 4'hf;

        inst = {
            mem[PC],
            mem[PC+1],
            mem[PC+9],
            mem[PC+8],
            mem[PC+7],
            mem[PC+6],
            mem[PC+5],
            mem[PC+4],
            mem[PC+3],
            mem[PC+2]
            };

        icode = inst[0:3];
        ifun = inst[4:7];
        i_error = 0;
        valP = 64'hf;
        valC = 64'hf;
```

```
59          rA = 4'hf;
60          rB = 4'hf;
61
62
63          if((icode == 4'd7) || (icode == 4'd8))
64          begin
65              inst = {
66                  mem[PC],
67                  mem[PC+8],
68                  mem[PC+7],
69                  mem[PC+6],
70                  mem[PC+5],
71                  mem[PC+4],
72                  mem[PC+3],
73                  mem[PC+2],
74                  mem[PC+1],
75                  mem[PC]
76              };
77          end
78
79
80          if(icode == 4'd0 && ifun == 4'd0) // halt
81          begin
82              valP = PC+64'd1;
83              halt = 1;
84          end
85          else if(icode == 4'd1 && ifun == 4'd0) // nop
86          begin
87              valP = PC+64'd1;
88          end
89          else if(icode == 4'd2 && ifun < 4'd7) // cmovXX
90          begin
91              rA = inst[8:11];
92              rB = inst[12:15];
93              valP = PC+64'd2;
94          end
95          else if(icode == 4'd3 && ifun == 4'd0) // irmovq
96          begin
97              rA = inst[8:11];
98              rB = inst[12:15];
99              valC = inst[16:79];
100             valP = PC+64'd10;
101         end
102         else if(icode == 4'd4 && ifun == 4'd0) // rmmovq
103         begin
104             rA = inst[8:11];
105             rB = inst[12:15];
106             valC = inst[16:79];
107             valP = PC+64'd10;
108         end
109         else if(icode == 4'd5 && ifun == 4'd0) // mrmovq
110         begin
111             rA = inst[8:11];
112             rB = inst[12:15];
113             valC = inst[16:79];
```

```
113             valC = inst[16:79];
114             valP = PC+64'd10;
115         end
116         else if(icode == 4'd6 && ifun < 4'd4) // OPq
117         begin
118             rA = inst[8:11];
119             rB = inst[12:15];
120             valP = PC+64'd2;
121         end
122         else if(icode == 4'd7 && ifun < 4'd7) // jXX
123         begin
124             valC = inst[8:71];
125             valP = PC+64'd9;
126         end
127         else if(icode == 4'd8 && ifun == 4'd0) // call
128         begin
129             valC = inst[8:71];
130             valP = PC+64'd9;
131         end
132         else if(icode == 4'd9 && ifun == 4'd0) // ret
133         begin
134             valP = PC+64'd1;
135         end
136         else if(icode == 4'd10 && ifun == 4'd0) // pushq
137         begin
138             rA = inst[8:11];
139             rB = inst[12:15];
140             valP = PC+64'd2;
141         end
142         else if(icode == 4'd11 && ifun == 4'd0) // popq
143         begin
144             rA = inst[8:11];
145             rB = inst[12:15];
146             valP = PC+64'd2;
147         end
148         else
149         begin
150             i_error = 1;
151             halt = 1;
152         end
153     end
154     else
155     begin
156         mem_error = 1;
157         halt = 1;
158     end
159 
160 end
161 endmodule
```

Testbench

```
`include "fetch.v"

module testbench;

reg clk;
reg [63:0]PC;
wire [3:0]icode,ifun,rA,rB;
wire [63:0]valP,valC;
wire mem_error,i_error,halt;

fetch call(.clk(clk), .valP(valP), .icode(icode), .ifun(ifun), .rA(rA), .rB(rB), .PC(PC), .valC(valC), .mem_error(mem_error),
        .i_error(i_error), .halt(halt));

initial
begin
    $dumpfile("fetch.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d: icode=%d, ifun=%d, rA=%d, rB=%d, valP=%d, valC=%d, PC=%d, mem_error=%d, i_error=%d, halt=%d",
            clk,icode,ifun,rA,rB,valP,valC,PC,mem_error,i_error,halt);
end

initial
begin
    clk = 1;

    #5 PC = 64'd0;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = valP;

    #10 PC = 64'd76;
    #5 $finish;
end
endmodule
```

Working

The fetch stage includes the instruction memory hardware unit.This unit reads 10 bytes from memory at a time, using the PC as the address of the first byte (byte 0). This byte is interpreted as the instruction byte and is split into two 4-bit quantities. The control logic blocks labeled “icode” and “ifun” then compute the instruction and function codes as equaling either the values read from memory or, in the event that the instruction address is not valid (as indicated by the signal imem_error), the values corresponding to a nop instruction. Based on the value of icode, we can compute three 1-bit signals (shown as dashed lines):

- instr_valid: Returns whether the byte is a legal instruction. This signal is used to detect an illegal instruction.
- Need_regids: Returns whether the instruction includes a register specifier byte
- need_valC. Returns whether the instruction include a constant word

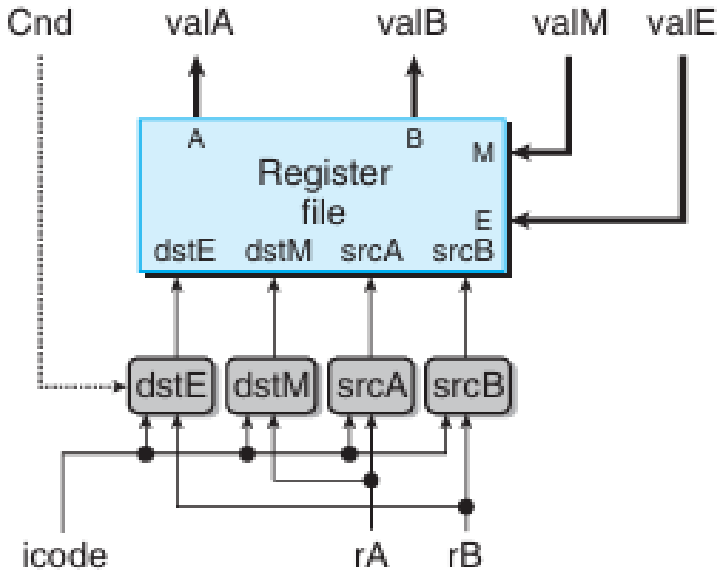
The signals instr_valid and imem_error (generated when the instruction address is out of bounds) are used to generate the status code in the memory stage.

The remaining 9 bytes read from the instruction memory encode some combination of the register specifier byte and the constant word. These bytes are processed by the hardware unit labeled “Align” into the register fields and the constant word. Byte 1 is split into register specifiers rA and rB when the computed signal need_regids is 1. If need_regids is 0, both register specifiers are set to 0xF (RNONE), indicating there are no registers specified by this instruction. Thus, we can assume that the signals rA and rB either encode registers we want to access or indicate that register access is not required.The unit labeled “Align” also generates the constant word valC. This will either be bytes 1–8 or bytes 2–9, depending on the value of signal need_regids. The PC incrementer hardware unit generates the signal valP, based on the current value of the PC, and the two signals need_regids and need_valC. For PC value p, need_regids value r, and need_valC value i, the incrementer generates the value p +1+r +8i.

DECODE AND WRITEBACK

The decode stage reads up to two operands from the register file, giving values valA and valB. It reads the registers designated by instruction fields rA and rB. It also reads the %rsp register for instructions call, ret, pushq and popq.

The write-back stage writes up to two results to the register file.



Code

```
module decode_wb(
input clk,Cnd,
input [3:0]icode,
input [3:0]rA,
input [3:0]rB,
input [63:0]valE,
input [63:0]valM,
output reg[63:0]valA,
output reg[63:0]valB,
output reg[63:0]rax,rcx,rdx,rbx,rsp,rbp,rsi,rdi,r8,r9,r10,r11,r12,r13,r14
);

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

initial
begin
    reg_mem[0] = 64'd0;
    reg_mem[1] = 64'd1;
    reg_mem[2] = 64'd2;
    reg_mem[3] = 64'd3;
    reg_mem[4] = 64'd32;
    reg_mem[5] = 64'd5;
    reg_mem[6] = 64'd6;
    reg_mem[7] = 64'd7;
    reg_mem[8] = 64'd8;
    reg_mem[9] = 64'd9;
    reg_mem[10] = 64'd10;
    reg_mem[11] = 64'd11;
    reg_mem[12] = 64'd12;
    reg_mem[13] = 64'd13;
    reg_mem[14] = 64'd14;
end
```

```
always @(*)
begin

    // valA = 64'hf;
    // valB = 64'hf;

    if(clk == 1)
    begin
        if(icode == 4'd2) // cmovXX
        begin
            valA = reg_mem[rA];
            valB = 0;
        end
        else if(icode == 4'd4) // rmmovq
        begin
            valA = reg_mem[rA];
            valB = reg_mem[rB];
        end
        else if(icode == 4'd5) // mrmovq
        begin
            valB = reg_mem[rB];
        end
        else if(icode == 4'd6) // OPq
        begin
            valA = reg_mem[rA];
            valB = reg_mem[rB];
        end
        else if(icode == 4'd8) // call
        begin
            valB = reg_mem[4];
        end
        else if(icode == 4'd9) // ret
        begin
            valA = reg_mem[4];
            valB = reg_mem[4];
        end
        else if(icode == 4'd10) // pushq
        begin
            valA = reg_mem[rA];
            valB = reg_mem[4];
        end
        else if(icode == 4'd11) // popq
        begin
            valA = reg_mem[4];
            valB = reg_mem[4];
        end
    end
end
```

```
always @(negedge clk)
begin

    if(icode == 4'd2) // cmovXX
    begin
        if(Cnd == 1)
        begin
            reg_mem[rB] = valE;
        end
    end
    else if(icode == 4'd3) // irmovq
    begin
        reg_mem[rB] = valE;
    end
    else if(icode == 4'd5) // mrmovq
    begin
        reg_mem[rA] = valM;
    end
    else if(icode == 4'd6) // OPq
    begin
        reg_mem[rB] = valE;
    end
    else if(icode == 4'd8) // call
    begin
        reg_mem[4] = valE;
    end
    else if(icode == 4'd9) // ret
    begin
        reg_mem[4] = valE;
    end
    else if(icode == 4'd10) // pushq
    begin
        reg_mem[4] = valE;
    end
    else if(icode == 4'd11) // popq
    begin
        reg_mem[4] = valE;
        reg_mem[rA] = valM;
    end

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

end

```

Testbench

```
`include "decode.v"

module testbench;

reg clk;
reg [3:0]icode,rA,rB;
wire [63:0]valA,valB;

decode call (.clk(clk), .icode(icode), .rA(rA), .rB(rB), .valA(valA), .valB(valB));

initial
begin
    $dumpfile("decode.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d: icode=%d, rA=%d, rB=%d, valA=%d, valB=%d",clk,icode,rA,rB,valA,valB);
end

initial
begin

    clk = 1;
    #5 icode = 4'd2; rA = 4'd0; rB = 4'd1;

    #10 icode = 4'd4; rA = 4'd1; rB = 4'd2;

    #10 icode = 4'd5; rA = 4'd0; rB = 4'd3;

    #10 icode = 4'd6; rA = 4'd1; rB = 4'd3;

    #10 icode = 4'd8; rA = 4'd8; rB = 4'd0;

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

    #10 icode = 4'd10; rA = 4'd2; rB = 4'd0;

    #10 icode = 4'd11; rA = 4'd9; rB = 4'd1;

    #10 icode = 4'd3; rA = 4'd2; rB = 4'd8;
    #5 $finish;

end
endmodule
```

```
include "writeback.v"

module testbench;

    reg clk;
    reg [3:0]icode;
    reg [3:0]rA;
    reg [3:0]rB;
    reg [63:0]valE;
    reg [63:0]valM;
    reg[63:0]valA,valB,valC;
    reg[3:0]ifun;
    reg Cnd;
    wire [63:0]rax,rcx,rdx,rbx,rsp,rbp,rsi,rdi,r8,r9,r10,r11,r12,r13,r14;

    writeback call(.clk(clk), .icode(icode), .rA(rA), .rB(rB), .valE(valE), .valM(valM), .Cnd(Cnd), .rax(rax), .rcx(rcx), .rdx(rdx), .rbx(rbx), .rsp(rsp), .rbp(rbp), .rsi(rsi), .rdi(rdi),
                .r8(r8), .r9(r9), .r10(r10), .r11(r11), .r12(r12), .r13(r13), .r14(r14));

    initial
    begin
        $dumpfile("writeback.vcd");
        $dumpvars(0,testbench);
    end

    always #5 clk = ~clk;
    always@(negedge clk)
    begin
        #1
        $monitor($time," clk %d: icode=%d, rA=%d, rB=%d, valE=%d, valM=%d, Cnd=%d, rax=%d, rcx=%d, rdx=%d, rbx=%d, rsp=%d, rbp=%d, rsi=%d, rdi=%d, r8=%d, r9=%d, r10=%d, r11=%d, r12=%d, r13=%d, r14=%d",
                clk,icode,rA,rB,valE,valM,Cnd,rax,rcx,rdx,rbx,rsp,rbp,rsi,rdi,r8,r9,r10,r11,r12,r13,r14);
    end

    initial
    begin

        clk = 1;
        #5 icode = 4'd2; rA = 4'd0; rB = 4'd1; valE = 64'd100; valM = 64'd10; Cnd = 0;

        #10 icode = 4'd2; rA = 4'd0; rB = 4'd1; valE = 64'd100; valM = 64'd10; Cnd = 1;

        #10 icode = 4'd3; rA = 4'd2; rB = 4'd3; valE = 64'd200; valM = 64'd20; Cnd = 0;

        #10 icode = 4'd5; rA = 4'd0; rB = 4'd2; valE = 64'd300; valM = 64'd30; Cnd = 0;

        #10 icode = 4'd6; rA = 4'd1; rB = 4'd2; valE = 64'd400; valM = 64'd40; Cnd = 0;

        #10 icode = 4'd8; rA = 4'd8; rB = 4'd9; valE = 64'd500; valM = 64'd50; Cnd = 0;

        #10 icode = 4'd9; rA = 4'd2; rB = 4'd1; valE = 64'd600; valM = 64'd60; Cnd = 1;

        #10 icode = 4'd10; rA = 4'd3; rB = 4'd1; valE = 64'd700; valM = 64'd70; Cnd = 0;

        #10 icode = 4'd11; rA = 4'd0; rB = 4'd2; valE = 64'd800; valM = 64'd80; Cnd = 0;

        #10 icode = 4'd7; rA = 4'd0; rB = 4'd1; valE = 64'd100; valM = 64'd10; Cnd = 0;
        #5 $finish;
    end
endmodule
```

Working

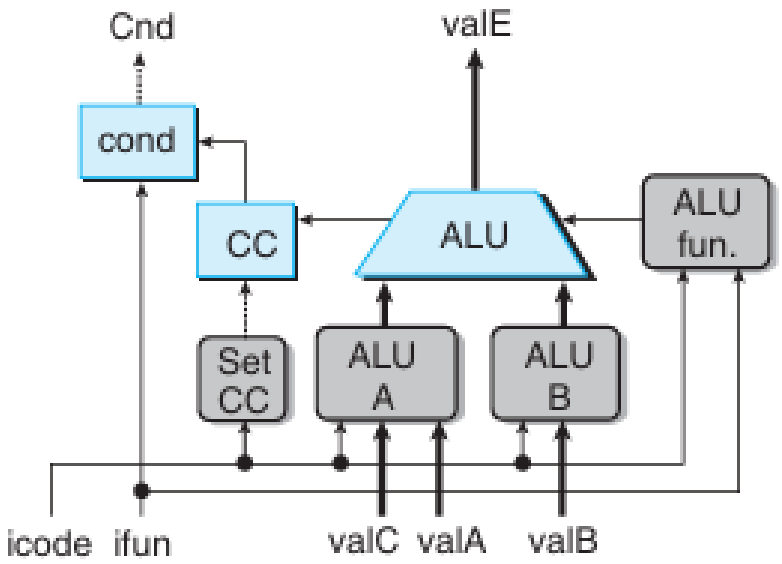
The register file has four ports. It supports up to two simultaneous reads (on ports A and B) and two simultaneous writes (on ports E and M). Each port has both an address connection and a data connection, where the address connection is a register ID, and the data connection is a set of 64 wires serving as either an output word (for a read port) or an input word (for a write port) of the register file. The two read ports have address inputs srcA and srcB, while the two write ports have address inputs dstE and dstM. The special identifier 0xF (RNONE) on an address port indicates that no register should be accessed.

The four blocks at the bottom generate the four different register IDs for the register file, based on the instruction code icode, the register specifiers rA and rB, and possibly the condition signal Cndcomputed in the execute stage. Register ID srcA indicates which register should be read to generate valA. Register ID dstE indicates the destination register for write port E, where the computed value valE is stored.

EXECUTE

In the execute stage, the arithmetic/logic unit (ALU) either performs the operation specified by the instruction, computes the effective address of a memory reference, or increments or

decrements the stack pointer. The value computed is valE. The condition codes are possibly set. For a conditional move instruction, the stage will evaluate the condition codes and move condition and enable the updating of the destination register only if the condition holds. Similarly, for a jump instruction, it determines whether or not the branch should be taken.



Code

```
`include "./ALU/alu.v"

module execute(
    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
);

reg [1:0]control;
reg signed [63:0]A;
reg signed [63:0]B;
wire signed [63:0]result;
wire overflow;

alu call(.control(control), .A(A), .B(B), .result(result), .overflow(overflow));

reg [2:0]CC; // Condition Code [zf, sf, of]
reg zf; // Zero flag
reg sf; // Sign flag
reg of; // Overflow flag

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

always @(*) begin
    if(icode == 4'd6 && clk == 1) // OPq
    begin
        zf = (result == 64'd0);
        sf = (result < 64'd0);
        // sf = (result[63] == 1);
        of = ((A < 64'd0 == B < 64'd0) && (result < 64'd0 != A < 64'd0));
        // of = (((A[63] == 1) == (B[63] == 1)) && ((result[63] == 1) ~= (A[63] == 1)));
        // of = overflow;
    end
end
```

```
always @(*) begin
    if(clk == 1)
    begin
        Cnd = 0;
        if(icode == 4'd2) // cmovXX
        begin
            if(ifun == 4'd0) // rrmovq
            begin
                Cnd = 1;
            end
            else if(ifun == 4'd1) // cmovle
            begin
                // (sf^of)||zf
                if((sf^of)|zf) begin
                    Cnd = 1;
                end
            end
            else if(ifun == 4'd2) // cmovl
            begin
                // (sf^of)
                if((sf^of)) begin
                    Cnd = 1;
                end
            end
            else if(ifun == 4'd3) // cmove
            begin
                // zf
                if(zf) begin
                    Cnd = 1;
                end
            end
            else if(ifun == 4'd4) // cmovne
            begin
                // !zf
                if(~zf) begin
                    Cnd = 1;
                end
            end
            else if(ifun == 4'd5) // cmovge
            begin
                // !(sf^of)
                if(~(sf^of)) begin
                    Cnd = 1;
                end
            end
            else if(ifun == 4'd6) // cmovg
```

```
        else if(ifun == 4'd6) // cmovg
        begin
            // !(sf^of) && !zf
            if((~(sf^of)) & ~zf) begin
                Cnd = 1;
            end
        end
        control = 2'd0;
        A = valB;
        B = valA;
        valE = result;
    end
    else if(icode == 4'd3) // irmovq
    begin
        control = 2'd0;
        A = 64'd0;
        B = valC;
        valE = result;
        // valE = 64'd0 + valC;
    end
    else if(icode == 4'd4) // rmmovq
    begin
        control = 2'd0;
        A = valB;
        B = valC;
        valE = result;
        // valE = valB + valC;
    end
    else if(icode == 4'd5) // mrmovq
    begin
        control = 2'd0;
        A = valB;
        B = valC;
        valE = result;
        // valE = valB + valC;
    end
```

```
    else if(icode == 4'd6) // OPq
    begin
        A = valA;
        B = valB;
        if(ifun == 4'd0) // add
        begin
            control = 2'd0;
        end
        else if(ifun == 4'd1) // sub
        begin
            A = valB;
            B = valA;
            control = 2'd1;
        end
        else if(ifun == 4'd2) // and
        begin
            control = 2'd2;
        end
        else if(ifun == 4'd3) // xor
        begin
            control = 2'd3;
        end
        valE = result;
    end
    else if(icode == 4'd7) // jxx
    begin
        if(ifun == 4'd0) // jmp
        begin
            Cnd = 1;
        end
        else if(ifun == 4'd1) // jle
        begin
            // (sf^of)||zf
            if((sf^of)|zf) begin
                Cnd = 1;
            end
        end
        else if(ifun == 4'd2) // jl
        begin
            // (sf^of)
            if((sf^of)) begin
                Cnd = 1;
            end
        end
        else if(ifun == 4'd3) //je
```

```
        else if(ifun == 4'd3) //je
        begin
            // zf
            if(zf) begin
                Cnd = 1;
            end
        end
        else if(ifun == 4'd4) //jne
        begin
            // !zf
            if(!zf) begin
                Cnd = 1;
            end
        end
        else if(ifun == 4'd5) //jge
        begin
            // !(sf^of)
            if(!(sf^of)) begin
                Cnd = 1;
            end
        end
        else if(ifun == 4'd6) //jg
        begin
            // !(sf^of) && !zf
            if((!(sf^of)) && !zf) begin
                Cnd = 1;
            end
        end
    end
    else if(icode == 4'd8) // call
    begin
        control = 2'd1;
        A = valB;
        B = 64'd8;
        valE = result;
    end
    else if(icode == 4'd9) // ret
    begin
        control = 2'd0;
        A = valB;
        B = 64'd8;
        valE = result;
    end
    else if(icode == 4'd10) // pushq
```

```
        else if(icode == 4'd8) // call
        begin
            control = 2'd1;
            A = valB;
            B = 64'd8;
            valE = result;
        end
        else if(icode == 4'd9) // ret
        begin
            control = 2'd0;
            A = valB;
            B = 64'd8;
            valE = result;
        end
        else if(icode == 4'd10) // pushq
        begin
            control = 2'd1;
            A = valB;
            B = 64'd8;
            valE = result;
        end
        else if(icode == 4'd11) // popq
        begin
            control = 2'd0;
            A = valB;
            B = 64'd8;
            valE = result;
        end
    end
end

endmodule
```

Testbench

```
`include "execute.v"

module testbench;

    reg clk;
    reg [3:0]icode;
    reg [3:0]ifun;
    reg [63:0]valA;
    reg [63:0]valB;
    reg [63:0]valC;
    wire signed[63:0]valE;
    wire Cnd;

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

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

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

    initial
    begin
        $monitor($time," clk %d: icode=%d, ifun=%d, valA=%d, valB=%d, valC=%d, valE=%d, Cnd=%d",clk,icode,ifun,valA,valB,valC,valE,Cnd);
    end

    initial
    begin
        #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

endmodule
```

Working

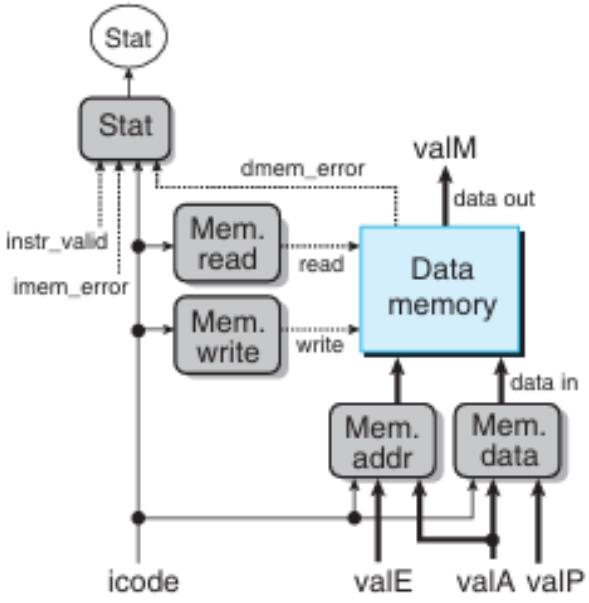
The execute stage includes the arithmetic/logic unit (ALU). This unit performs the operation add, subtract, and, or xor on inputs aluA and aluB based on the setting of the ALUfun signal. These data and control signals are generated by three control blocks. The ALU output becomes the signal valE. The operands are listed with aluB first, followed by aluA to make sure that the subq instruction subtracts valA from valB. We can see that the value of aluA can be valA, valC, or either −8 or +8, depending on the instruction type. It is mostly used as an adder. For the OPq instructions, however, we want it to use the operation encoded in the ifun field of the instruction.

The execute stage also includes the condition code register. Our ALU generates the three signals on which the condition codes are based—zero, sign, and overflow—every time it

operates. However, we only want to set the condition codes when an OPq instruction is executed. We therefore generate a signal set_cc that controls whether or not the condition code register should be updated. The hardware unit labeled “cond” uses a combination of the condition codes and the function code to determine whether a conditional branch or data transfer should take place.

MEMORY

The memory stage is responsible for reading from and writing data into the main memory. Reading and writing depends on 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.



Code

```
module memory(
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
);

reg [63:0]data_mem[0:1023];
reg read,write;
reg [63:0]address;
reg [63:0]data_w;

initial
begin
    data_mem[0] = 64'd0;
    data_mem[1] = 64'd1;
    data_mem[2] = 64'd2;
    data_mem[3] = 64'd3;
    data_mem[4] = 64'd4;
    data_mem[5] = 64'd5;
    data_mem[6] = 64'd6;
    data_mem[7] = 64'd7;
    data_mem[8] = 64'd8;
    data_mem[9] = 64'd9;
    data_mem[10] = 64'd10;
end

always @(negedge clk)
begin

    address = 64'hf;
    data_w = 64'hf;
    read = 0; write = 0;
    valM = 64'hf;
    dmem_error = 0;

    if(icode == 4'd4) // rmmovq
    begin
        data_w = valA;
        address = valE;
        write = 1;
    end
```

```
else if(icode == 4'd5) // mrmovq
begin
    address = valE;
    read = 1;
end
else if(icode == 4'd8) // call
begin
    data_w = valP;
    address = valE;
    write = 1;
end
else if(icode == 4'd9) // ret
begin
    address = valA;
    read = 1;
end
else if(icode == 4'd10) // pushq
begin
    data_w = valA;
    address = valE;
    write = 1;
end
else if(icode == 4'd11) // popq
begin
    address = valA;
    read = 1;
end
```

```
    if(address>=0 && address<1024)
    begin
        if(read == 1 && write == 0)
        begin
            valM = data_mem[address];
        end
        else if(read == 0 && write == 1)
        begin
            data_mem[address] = data_w;
        end
        else if(read == 1 && write == 1)
        begin
            dmem_error = 1;
        end
    end
    else
    begin
        dmem_error = 1;
    end

        // $display($time, "dmem_error=%d", dmem_error);

end
endmodule
```

Testbench

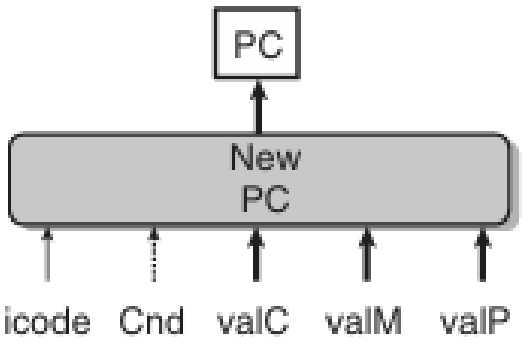
```
`include "memory.v"

module testbench;

reg clk;
reg [3:0]icode;
reg [63:0]valA,valE,valP;
wire [63:0]valM;
wire dmem_error;

memory call(.clk(clk), .icode(icode), .valA(valA), .valE(valE), .valP(valP), .valM(valM), .dmem_error(dmem_error));

initial
begin
    $dumpfile("memory.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d: icode=%d, valA=%d, valE=%d, valP=%d, valM=%d, dmem_error=%d",clk,icode,valA,valE,valP,valM,dmem_error);
end

initial
begin
    clk = 1;

    #5 icode = 4'd4; valA = 64'd100; valE = 64'd24; valP = 64'd0;

    #10 icode = 4'd5; valA = 64'd0; valE = 64'd8; valP = 64'd3;

    #10 icode = 4'd8; valA = 64'd0; valE = 64'd32; valP = 64'd64;

    #10 icode = 4'd9; valA = 64'd32; valE = 64'd0; valP = 64'd1;

    #10 icode = 4'd10; valA = 64'd10; valE = 64'd20; valP = 64'd2;

    #10 icode = 4'd11; valA = 64'd20; valE = 64'd0; valP = 64'd7;

    #10 icode = 4'd4; valA = 64'd0; valE = 64'd1200; valP = 64'd0;

    #10 icode = 4'd3; valA = 64'd1; valE = 64'd1; valP = 64'd1;
    #5 $finish;
end
endmodule
```

Working

The memory stage has the task of either reading or writing program data. Two control blocks generate the values for the memory address and the memory input data (for write operations). Two other blocks generate the control signals indicating whether to perform a read or a write operation. When a read operation is performed, the data memory generates the value valM. The address for memory reads and writes is always valE or valA. The control signal mem_read is set for instructions that read data from memory. A final function for the memory stage is to compute the status code Stat resulting from the instruction execution according to the values of icode, imem_ error, and instr_valid generated in the fetch stage and the signal dmem_error generated by the data memory.

PC UPDATE

The PC update stage is responsible for calculating the value of the updated PC, which is the value of the PC that contains the location of the next instruction. The value of an updated PC depends on the values of icode, valC, valM, valP and Cnd.



Code

```
module pc_update(
input clk,Cnd,
input [3:0]icode,
input [63:0]valP,
input [63:0]valC,
input [63:0]valM,
output reg[63:0]PC_u
);

always @(*)
begin

    PC_u = 64'd0;

    PC_u = valP;
    if(icode == 4'd7) // jXX
    begin
        if(Cnd == 1)
        begin
            PC_u = valC;
        end
    end
    else if(icode == 4'd8) // call
    begin
        PC_u = valC;
    end
    else if(icode == 4'd9) // ret
    begin
        PC_u = valM;
    end
end
endmodule
```

Testbench

```
`include "pc_update.v"

module testbench;

reg clk,Cnd;
reg [3:0]icode;
reg [63:0]valP,valC,valM;
wire [63:0]PC_u;

pc_update call(.clk(clk), .Cnd(Cnd), .icode(icode), .valP(valP), .valC(valC), .valM(valM), .PC_u(PC_u));

initial
begin
    $dumpfile("pc_update.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d; icode=%d; Cnd=%d; valP=%d; valC=%d; valM=%d; PC=%d",clk,icode,Cnd,valP,valC,valM,PC_u);
end

initial
begin
    clk = 1;
    #5 icode = 4'd7; Cnd = 0; valP = 64'd4; valC = 64'd8; valM = 64'd0;

    #10 icode = 4'd7; Cnd = 1; valP = 64'd4; valC = 64'd8; valM = 64'd0;

    #10 icode = 4'd8; Cnd = 0; valP = 64'd1; valC = 64'd2; valM = 64'd3;

    #10 icode = 4'd9; Cnd = 1; valP = 64'd3; valC = 64'd6; valM = 64'd9;

    #10 icode = 4'd3; Cnd = 0; valP = 64'd10; valC = 64'd20; valM = 64'd30;

    #10 icode = 4'd5; Cnd = 0; valP = 64'd5; valC = 64'd10; valM = 64'd15;
    #5 $finish;
end
endmodule
```

Working

The final stage in SEQ generates the new value of the program counter. The new PC will be valC, valM, or valP, depending on the instruction type and whether or not a branch should be taken.

Combined

Code

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

module testbench;

reg clk;
reg [63:0]PC;
wire [3:0]icode,ifun,rA,rB;
wire [63:0]valP,valC,valA,valB,valE,valM,PC_u;
wire mem_error,i_error,halt,dmem_error,Cnd;
wire [63:0]rax,rcx,rdx,rbx,rsp,rbp,rsi,rdi,r8,r9,r10,r11,r12,r13,r14;

reg [1:0]s_c; // status conditions

fetch fet(.clk(clk), .valP(valP), .icode(icode), .ifun(ifun), .rA(rA), .rB(rB), .PC(PC), .valC(valC), .mem_error(mem_error), .i_error(i_error), .halt(halt));
decode_wb dw(.clk(clk), .Cnd(Cnd), .icode(icode), .rA(rA), .rB(rB), .valE(valE), .valM(valM), .valA(valA), .valB(valB), .rax(rax), .rcx(rcx), .rdx(rdx), .rbx(rbx), .rsp(rsp),
             .rbp(rbp), .rsi(rsi), .rdi(rdi), .r8(r8), .r9(r9), .r10(r10), .r11(r11), .r12(r12), .r13(r13), .r14(r14));
execute exe(.clk(clk), .icode(icode), .ifun(ifun), .valA(valA), .valB(valB), .valC(valC), .valE(valE), .Cnd(Cnd));
memory mem(.clk(clk), .icode(icode), .valA(valA), .valE(valE), .valP(valP), .valM(valM), .dmem_error(dmem_error));
pc_update pcu(.clk(clk), .Cnd(Cnd), .icode(icode), .valP(valP), .valC(valC), .valM(valM), .PC_u(PC_u));


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

    clk = 1;
    PC = 64'd0;
    s_c = 2'd0; // 0->AOK, 1->HLT, 2->ADR, 3->INS
end
```

```
always #5 clk = ~clk;

always @(*)
begin
    PC = PC_u;
end

always @(*)
begin
    if(halt == 1)
    begin
        s_c = 2'd1;
    end
    else if((dmem_error == 1) || (mem_error == 1))
    begin
        s_c = 2'd2;
        // $display($time, " mem_error=%d, dmem_error=%d",mem_error, dmem_error);
    end
    else if(i_error == 1)
    begin
        s_c = 2'd3;
    end

    if(s_c != 0)
    begin
        $finish;
    end
end


initial
begin
    $monitor($time," clk %d: icode=%d, ifun=%d, rA=%d, rB=%d, valP=%d, valC=%d, PC=%d, mem_error=%d, i_error=%d, halt=%d",
            clk,icode,ifun,rA,rB,valP,valC,PC,mem_error,i_error,halt);
end

endmodule
```

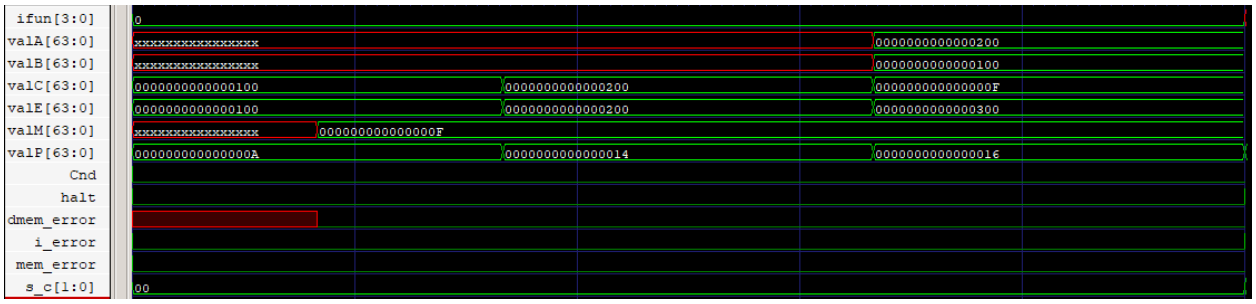
Input

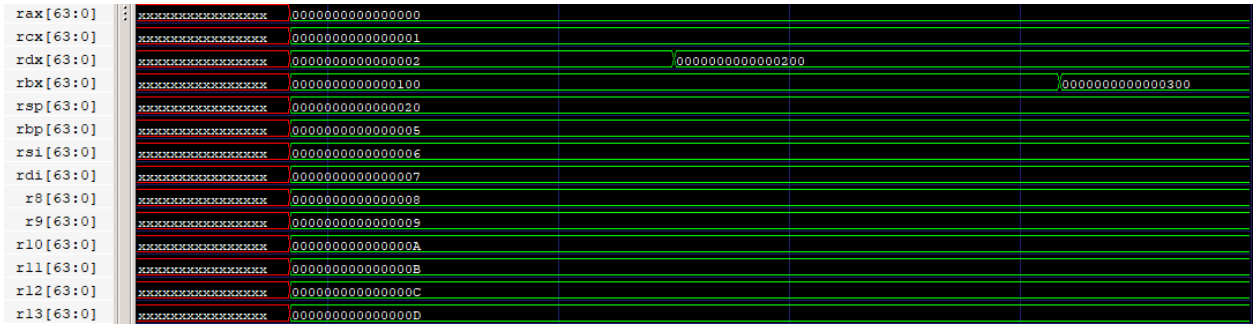
```
00110000
11110011
00000000
00000001
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110010
00000000
00000010
00000000
00000000
00000000
00000000
00000000
00000000
01100000
00100011
```

Output

```
                0 clk 1: icode= 3, ifun= 0, rA=15, rB= 3, valP=                  10, valC=                 256, PC=                  10, mem_error=0, i_error=0, halt=0
                5 clk 0: icode= 3, ifun= 0, rA=15, rB= 3, valP=                  10, valC=                 256, PC=                  10, mem_error=0, i_error=0, halt=0
               10 clk 1: icode= 3, ifun= 0, rA=15, rB= 2, valP=                  20, valC=                 512, PC=                  20, mem_error=0, i_error=0, halt=0
               15 clk 0: icode= 3, ifun= 0, rA=15, rB= 2, valP=                  20, valC=                 512, PC=                  20, mem_error=0, i_error=0, halt=0
               20 clk 1: icode= 6, ifun= 0, rA= 2, rB= 3, valP=                  22, valC=                  15, PC=                  22, mem_error=0, i_error=0, halt=0
               25 clk 0: icode= 6, ifun= 0, rA= 2, rB= 3, valP=                  22, valC=                  15, PC=                  22, mem_error=0, i_error=0, halt=0
.\seq.v:60: $finish called at 30 (1s)
               30 clk 1: icode= x, ifun= x, rA=15, rB=15, valP=                  15, valC=                  15, PC=                  15, mem_error=0, i_error=1, halt=1
```

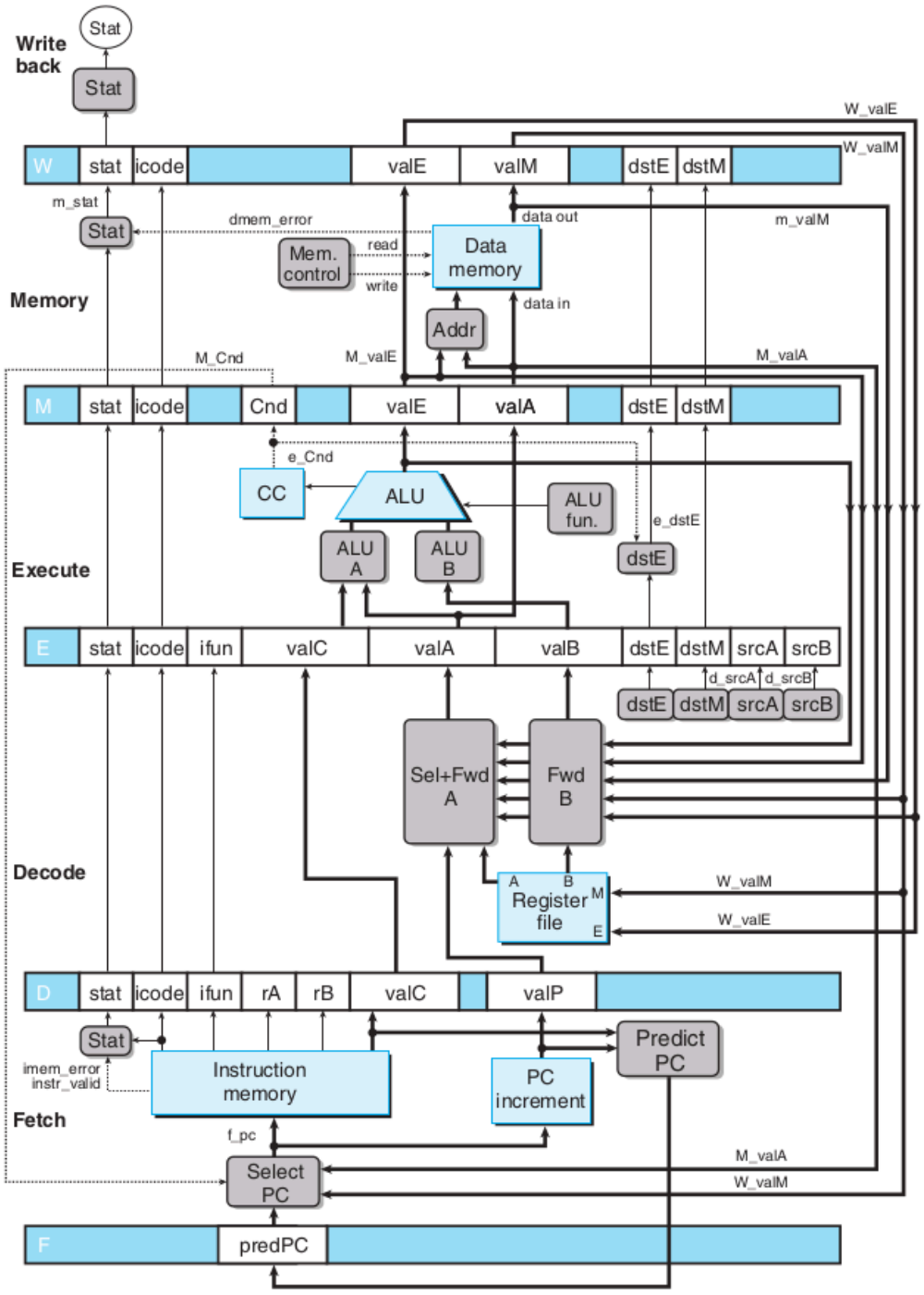
GTK Plots





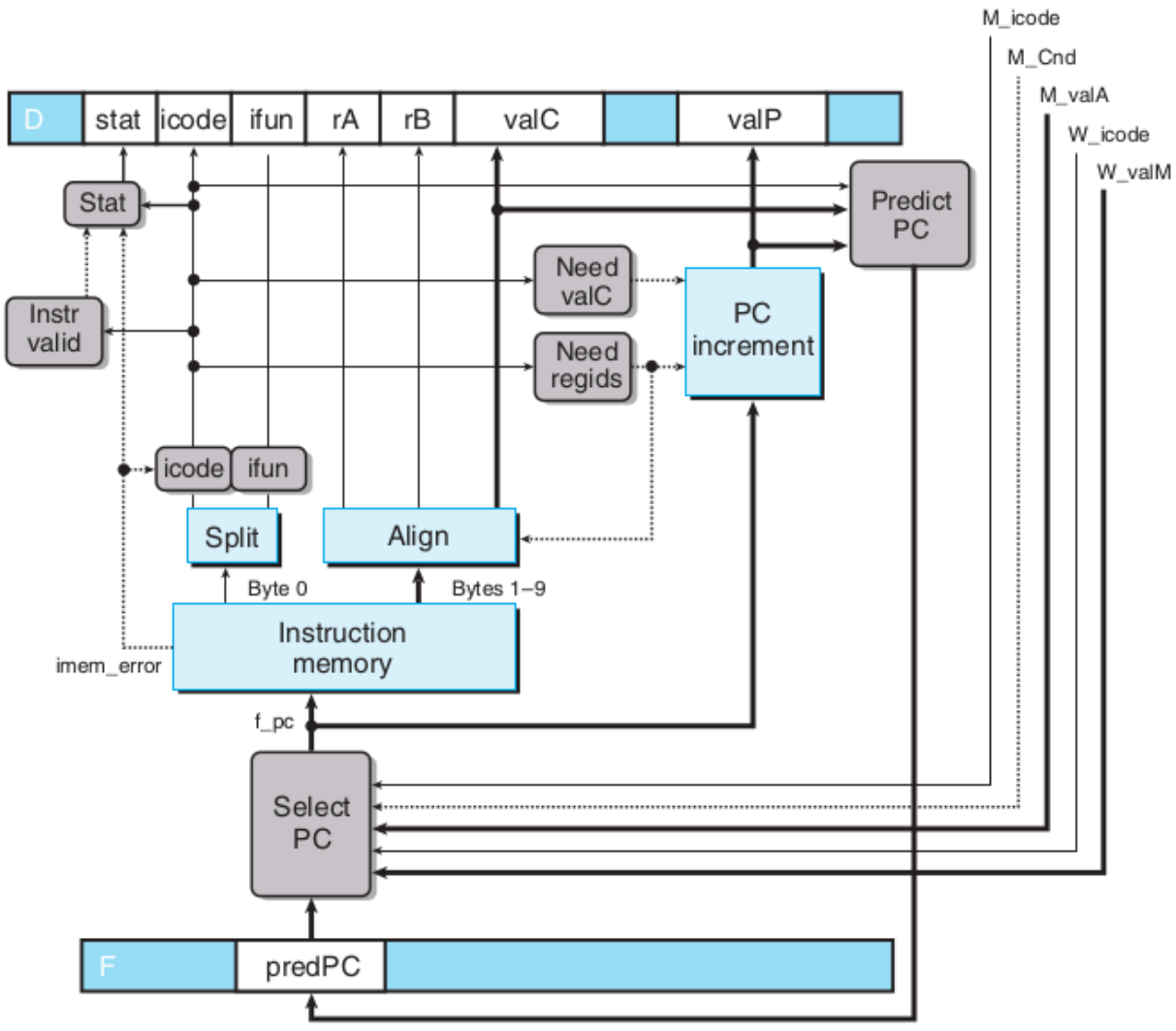
Pipeline Implementation

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



PC SELECTION AND FETCH

This stage selects a current value for the program counter and predicts the next PC value. The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address and extracts the instruction specifier byte - icode and ifun, alongwith values of rA, rB, valC and valP.



Code

```
module fetch(
input clk,
input [63:0]F_predPC,
input signed[63:0]M_valA,
input signed[63:0]W_valM,
input [3:0]M_icode,
input M_Cnd,
input [3:0]W_icode,
input D_stall,
input D_bubble,
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_valP,
output reg[63:0]D_valC,
output reg[63:0]f_predPC,
output reg [1:0]D_stat
);

reg [7:0]mem[0:1023];
reg [0:79]inst;
reg [63:0]f_pc;
reg imem_error;
reg instr_valid;

reg[3:0]f_icode;
reg[3:0]f_ifun;
reg[3:0]f_rA;
reg[3:0]f_rB;
reg[63:0]f_valP;
reg[63:0]f_valC;
reg [1:0]f_stat;

initial
begin
    // $readmemb("call_ret.txt",mem);
    // $readmemb("mrmovq.txt",mem);
    // $readmemb("jXX.txt",mem);
    // $readmemb("push_pop.txt",mem);
    // $readmemb("exp_hdll.txt",mem);

    imem_error = 0;
    f_icode = 4'hf;
    f_ifun = 4'hf;
    instr_valid = 0;
    f_valP = 64'd0;
    f_valC = 64'd0;
    f_predPC = 64'd0;
    f_rA = 4'hf;
    f_rB = 4'hf;
end
```

```
54  always @(*)
55  begin
56
57      imem_error = 0; instr_valid = 0; f_stat = 0;
58
59      if(M_icode == 4'd7 && M_Cnd == 0) // Mispredicted Branch
60      begin
61          f_pc = M_valA;
62      end
63      else if(W_icode == 4'd9) // Return
64      begin
65          f_pc = W_valM;
66      end
67      else
68      begin
69          f_pc = F_predPC;
70      end
71
72
73      if(f_pc>=0 && f_pc<1024)
74      begin
75
76          inst = {
77              mem[f_pc],
78              mem[f_pc+1],
79              mem[f_pc+9],
80              mem[f_pc+8],
81              mem[f_pc+7],
82              mem[f_pc+6],
83              mem[f_pc+5],
84              mem[f_pc+4],
85              mem[f_pc+3],
86              mem[f_pc+2]
87          };
88
89          f_icode = inst[0:3];
90          f_ifun = inst[4:7];
91
92          if((f_icode == 4'd7) || (f_icode == 4'd8))
93          begin
94              inst = {
95                  mem[f_pc],
96                  mem[f_pc+8],
97                  mem[f_pc+7],
98                  mem[f_pc+6],
99                  mem[f_pc+5],
100                 mem[f_pc+4],
101                 mem[f_pc+3],
102                 mem[f_pc+2],
103                 mem[f_pc+1],
104                 mem[f_pc]
105             };
106         end
```

```
108         if(f_icode == 4'd0 && f_ifun == 4'd0) // halt
109         begin
110             f_valP = f_pc+64'd1;
111             f_predPC = f_valP;
112             f_stat = 2'd1;
113         end
114         else if(f_icode == 4'd1 && f_ifun == 4'd0) // nop
115         begin
116             f_valP = f_pc+64'd1;
117             f_predPC = f_valP;
118         end
119         else if(f_icode == 4'd2 && f_ifun < 4'd7) // cmovXX
120         begin
121             f_rA = inst[8:11];
122             f_rB = inst[12:15];
123             f_valP = f_pc+64'd2;
124             f_predPC = f_valP;
125         end
126         else if(f_icode == 4'd3 && f_ifun == 4'd0) // irmovq
127         begin
128             f_rA = inst[8:11];
129             f_rB = inst[12:15];
130             f_valC = inst[16:79];
131             f_valP = f_pc+64'd10;
132             f_predPC = f_valP;
133         end
134         else if(f_icode == 4'd4 && f_ifun == 4'd0) // rmmovq
135         begin
136             f_rA = inst[8:11];
137             f_rB = inst[12:15];
138             f_valC = inst[16:79];
139             f_valP = f_pc+64'd10;
140             f_predPC = f_valP;
141         end
142         else if(f_icode == 4'd5 && f_ifun == 4'd0) // mrmovq
143         begin
144             f_rA = inst[8:11];
145             f_rB = inst[12:15];
146             f_valC = inst[16:79];
147             f_valP = f_pc+64'd10;
148             f_predPC = f_valP;
149         end
150         else if(f_icode == 4'd6 && f_ifun < 4'd4) // OPq
151         begin
152             f_rA = inst[8:11];
153             f_rB = inst[12:15];
154             f_valP = f_pc+64'd2;
155             f_predPC = f_valP;
156         end
157         else if(f_icode == 4'd7 && f_ifun < 4'd7) // jXX
158         begin
159             f_valC = inst[8:71];
160             f_valP = f_pc+64'd9;
161             f_predPC = f_valC;
```

```
163         else if(f_icode == 4'd8 && f_ifun == 4'd0) // call
164         begin
165             f_valC = inst[8:71];
166             f_valP = f_pc+64'd9;
167             f_predPC = f_valC;
168         end
169         else if(f_icode == 4'd9 && f_ifun == 4'd0) // ret
170         begin
171             f_valP = f_pc+64'd1;
172             f_predPC = f_valP;
173         end
174         else if(f_icode == 4'd10 && f_ifun == 4'd0) // pushq
175         begin
176             f_rA = inst[8:11];
177             f_rB = inst[12:15];
178             f_valP = f_pc+64'd2;
179             f_predPC = f_valP;
180         end
181         else if(f_icode == 4'd11 && f_ifun == 4'd0) // popq
182         begin
183             f_rA = inst[8:11];
184             f_rB = inst[12:15];
185             f_valP = f_pc+64'd2;
186             f_predPC = f_valP;
187         end
188         else
189         begin
190             instr_valid = 1;
191         end
192     end
193     else
194     begin
195         imem_error = 1;
196     end
197
198     if(imem_error == 1)
199     begin
200         f_stat = 2'd2;
201     end
202     else if(instr_valid == 1)
203     begin
204         f_stat = 2'd3;
205     end
206 end
```

```
208 always @(posedge clk)
209 begin
210
211     if(D_stall == 1)
212     begin
213         // f_predPC <= F_predPC;
214     end
215     else if(D_bubble == 1)
216     begin
217
218         D_icode <= 4'd1; // nop
219         D_ifun <= 4'd0;
220         D_rA <= 4'hf;
221         D_rB <= 4'hf;
222         D_valC <= 64'd0;
223         D_valP <= 64'd0;
224         D_stat <= 2'd0;
225     end
226     else
227     begin
228         D_icode <= f_icode;
229         D_ifun <= f_ifun;
230         D_rA <= f_rA;
231         D_rB <= f_rB;
232         D_valC <= f_valC;
233         D_valP <= f_valP;
234         D_stat <= f_stat;
235     end
236
237 end
238 endmodule
```

Testbench

```
`include "fetch.v"

module testbench;

reg clk,M_Cnd,D_stall,D_bubble;
reg [63:0]F_predPC,M_valA,W_valM;
reg [3:0]M_icode,W_icode;
wire [3:0]D_icode,D_ifun,D_rA,D_rB;
wire [63:0]D_valP,D_valC,f_predPC;
wire [1:0]D_stat;

fetch call(.clk(clk),.M_Cnd(M_Cnd),.D_stall(D_stall),.D_bubble(D_bubble),.F_predPC(F_predPC),.M_valA(M_valA), .W_valM(W_valM), .M_icode(M_icode), .W_icode(W_icode), .D_icode(D_icode), .D_ifun(D_

initial
begin
    $dumpfile("fetch.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time,"clk =%d: icode=%d, ifun=%d, rA=%d, rB=%d, valP=%d, valC=%d, predPC=%d,stat=%d", clk,D_icode,D_ifun,D_rA,D_rB,D_valP,D_valC,f_predPC,D_stat);
end

initial
begin
    clk = 0;

    #5 F_predPC = 64'd0;

    #10 F_predPC = f_predPC;

    #10 F_predPC = f_predPC;

    #10 F_predPC = f_predPC;

    #10 F_predPC = f_predPC;

    #10 F_predPC = f_predPC;

    #10 F_predPC = f_predPC; D_stall = 1;

    #10 F_predPC = f_predPC; D_stall= 1;


    #5 $finish;
end
endmodule
```

Working

The working of the fetch stage in PIPE hardware is similar to that of the fetch stage in SEQ. The fetch unit reads 10 bytes from memory at a time, using the PC as the address of the first byte. From the instruction it extracts the two 4-bit portions of the instruction specifier byte - f_icode (the instruction code) and f_ifun (the instruction function). It possibly fetches a register specifier byte, giving one or both of the register operand specifiers f_rA and f_rB. It also possibly fetches an 8-byte constant word f_valC. It computes f_valP to be the address of the instruction following the current one in sequential order.

If an invalid memory location is accessed or an invalid instruction is fetched, the imem_error and instr_valid flags are generated respectively. A flag is also generated if a halt instruction is fetched. The f_stat signal is updated accordingly.

- f_stat = 0 → Normal Operation
- f_stat = 1 → Halt Instruction encountered
- f_stat = 2 → Bad Instruction Address encountered
- f_stat = 3 → Invalid Instruction encountered

The PC selection logic chooses between three program counter sources. As a mispredicted branch enters the memory stage, the value of valP for this instruction (indicating the address of the following instruction) is read from pipeline register M (M_valA). When a ret instruction enters the write-back stage, the return address is read from the pipeline register W (W_valM). All other cases use the predicted value of the PC, stored in register F (F_predPC). The PC prediction logic chooses valC for the fetched instruction when it is either a call or a jump instruction, otherwise the PC is set to valP.

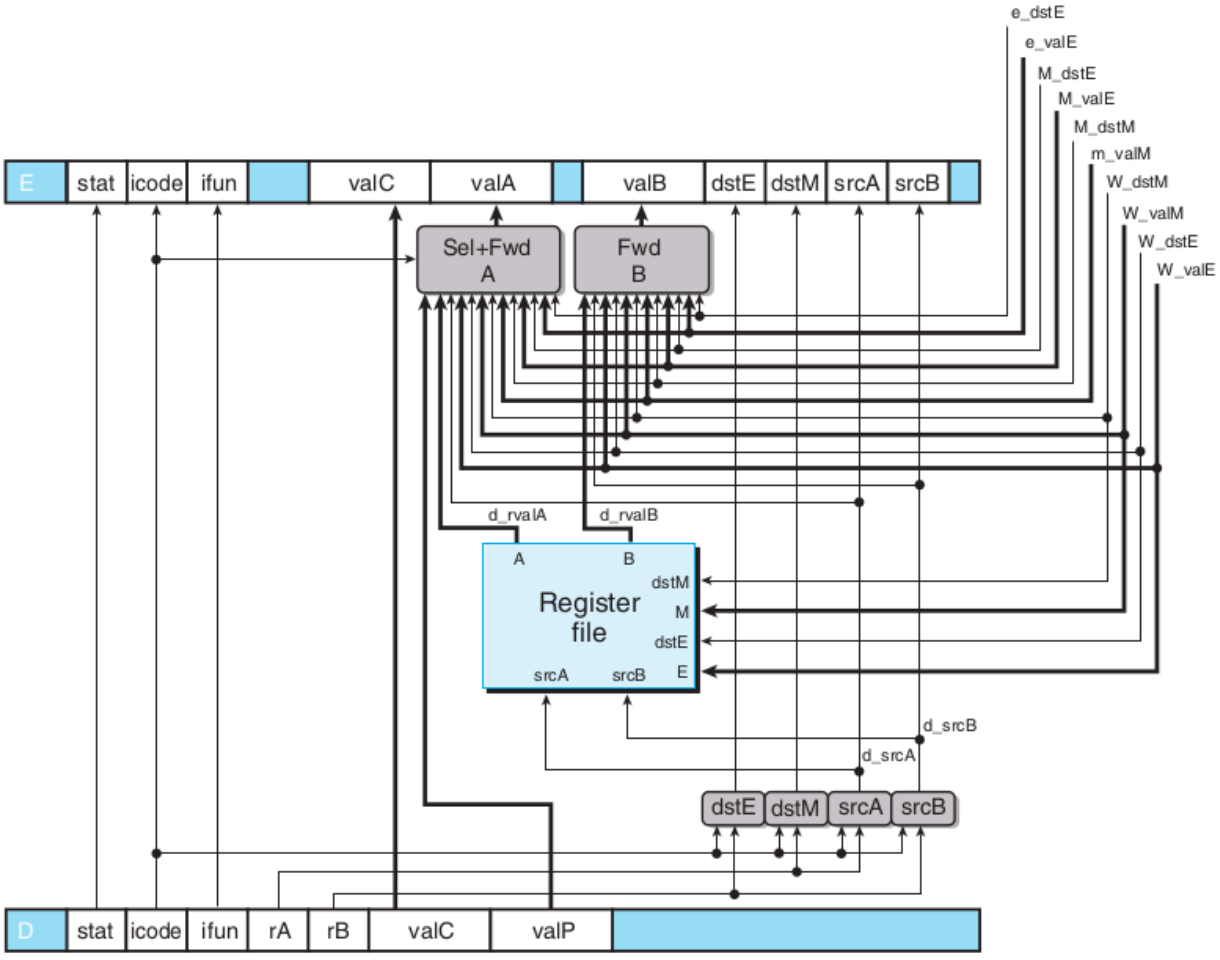
The pipeline register bank F holds the predicted value of the program counter (F_predPC).

The pipeline register bank D sits between the fetch and decode stages. It holds information about the most recently fetched instruction for processing by the decode stage.

DECODE AND WRITEBACK

The decode stage reads the registers designated by fields D_rA and D_rB and stores them in d_valA and d_valB respectively. For the instructions call, ret, pushq and popq, it also reads the %rsp register to access the stack pointer.

The write-back stage writes values W_valE and W_valM in the registers designated by the fields W_dstE and W_dstM respectively.



Code

DECODE STAGE

```
module decode_wb(
    input clk,
    input [1:0]D_stat,
    input [3:0]D_icode,
    input [3:0]D_ifun,
    input [3:0]D_rA,
    input [3:0]D_rB,
    input signed[63:0]D_valC,
    input [63:0]D_valP,
    input E_bubble,
    input [3:0]e_dstE,
    input signed[63:0]e_valE,
    input [3:0]M_dstM,
    input signed [63:0]m_valM,
    input [3:0]M_dstE,
    input signed[63:0]M_valE,
    input [3:0]W_dstM,
    input signed[63:0]W_valM,
    input [3:0]W_dstE,
    input signed[63:0]W_valE,
    input [1:0]W_stat,
    input [3:0]W_icode,
    input W_stall,
    output reg[1:0]E_stat,
    output reg[3:0]E_icode,
    output reg[3:0]E_ifun,
    output reg[63:0]E_valC,
    output reg[63:0]E_valA,
    output reg[63:0]E_valB,
    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[63:0]rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14,
    output reg[3:0]d_srcA,
    output reg[3:0]d_srcB
);

reg [3:0]icode;
reg [3:0]ifun;
reg [3:0]srcA;
reg [3:0]srcB;
reg [3:0]dstE;
reg [3:0]dstM;
reg signed[63:0]valA;
reg signed[63:0]valB;
reg signed[63:0]valC;
reg signed[63:0]valP;

always @(*) begin
    icode = D_icode;
    ifun = D_ifun;
    valC = D_valC;
    valP = D_valP;
end
```

```
57  reg [63:0]reg_mem[0:14];
58
59  initial
60  begin
61      rsp = 64'd32;
62      reg_mem[0] = rax;
63      reg_mem[1] = rcx;
64      reg_mem[2] = rdx;
65      reg_mem[3] = rbx;
66      reg_mem[4] = rsp;
67      reg_mem[5] = rbp;
68      reg_mem[6] = rsi;
69      reg_mem[7] = rdi;
70      reg_mem[8] = r8;
71      reg_mem[9] = r9;
72      reg_mem[10] = r10;
73      reg_mem[11] = r11;
74      reg_mem[12] = r12;
75      reg_mem[13] = r13;
76      reg_mem[14] = r14;
77
78  end
79
80  always @(*)
81  begin
82      if(icode == 4'd2) // cmovXX
83      begin
84          srcA = D_rA;
85          srcB = D_rB;
86          dstE = D_rB;
87          dstM = 4'd15;
88          valA = reg_mem[D_rA];
89          valB = 4'd0;
90      end
91      else if(icode == 4'd3) // irmovq
92      begin
93          srcA = 4'd15;
94          srcB = 4'd15;
95          dstE = D_rB;
96          dstM = 4'd15;
97      end
98      else if(icode == 4'd4) // rmmovq
99      begin
100         srcA = D_rA;
101         srcB = D_rB;
102         dstE = D_rB;
103         dstM = 4'd15;
104         valA = reg_mem[D_rA];
105         valB = reg_mem[D_rB];
106     end
107     else if(icode == 4'd5) // mrmovq
108     begin
109         srcA = 4'd15;
110         srcB = D_rB;
111         dstE = 4'd15;
112         dstM = D_rA;
113         valB = reg_mem[D_rB];
114     end
```

```
115     else if(icode == 4'd6) // OPq
116     begin
117         srcA = D_rA;
118         srcB = D_rB;
119         dstE = D_rB;
120         dstM = 4'd15;
121         valA = reg_mem[D_rA];
122         valB = reg_mem[D_rB];
123     end
124     else if(icode == 4'd8) // call
125     begin
126         srcA = 4'd15;
127         srcB = 4'd4;
128         dstE = 4'd4;
129         dstM = 4'd15;
130         valB = reg_mem[4];
131     end
132     else if(icode == 4'd9) // ret
133     begin
134         srcA = 4'd4;
135         srcB = 4'd4;
136         dstE = 4'd4;
137         dstM = 4'd15;
138         valA = reg_mem[4];
139         valB = reg_mem[4];
140     end
141     else if(icode == 4'd10) // pushq
142     begin
143         srcA = D_rA;
144         srcB = 4'd4;
145         dstE = 4'd4;
146         dstM = 4'd15;
147         valA = reg_mem[D_rA];
148         valB = reg_mem[4];
149     end
150     else if(icode == 4'd11) // popq
151     begin
152         srcA = 4'd4;
153         srcB = 4'd4;
154         dstE = 4'd4;
155         dstM = 4'd15;
156         valA = reg_mem[4];
157         valB = reg_mem[4];
158     end
159     else
160     begin
161         srcA = 4'd15;
162         srcB = 4'd15;
163         dstE = 4'd15;
164         dstM = 4'd15;
165     end
166     d_srcA = srcA;
167     d_srcB = srcB;
168 end
```

FORWARDING LOGIC

```
170  // Forwarding logic for valA
171  always @(*) begin
172      if(icode == 4'd7 || icode == 4'd8)
173      begin
174          valA = D_valP;
175      end
176      else if(srcA == e_dstE && srcA != 4'd15)
177      begin
178          valA = e_valE;
179      end
180      else if(srcA == M_dstM && srcA != 4'd15)
181      begin
182          valA = m_valM;
183      end
184      else if(srcA == M_dstE && srcA != 4'd15)
185      begin
186          valA = M_valE;
187      end
188      else if(srcA == W_dstM && srcA != 4'd15)
189      begin
190          valA = W_valM;
191      end
192      else if(srcA == W_dstE && srcA != 4'd15)
193      begin
194          valA = W_valE;
195      end
196      // $display("icode=%d, srcA=%d, e_dste=%d, M_dstM=%d, M_dstE=%d, W_dstM=%d, W_dstE=%d",icode,srcA,e_dstE,M_dstM
197  end
198
199  // Forwarding logic for valB
200  always @(*) begin
201      if(srcB == e_dstE && srcB != 4'd15)
202      begin
203          valB = e_valE;
204      end
205      else if(srcB == M_dstM && srcB != 4'd15)
206      begin
207          valB = m_valM;
208      end
209      else if(srcB == M_dstE && srcB != 4'd15)
210      begin
211          valB = M_valE;
212      end
213      else if(srcB == W_dstM && srcB != 4'd15)
214      begin
215          valB = W_valM;
216      end
217      else if(srcB == W_dstE && srcB != 4'd15)
218      begin
219          valB = W_valE;
220      end
221      // $display("icode=%d, srcB=%d, M_dstM=%d, valB=%d, m_valM=%d",icode,srcB,M_dstM,W_dstM,W_dstE,e_dstE,M_dstE,va
222  end
```

WRITE BACK STAGE

```
224  always @(posedge clk)
225  begin
226      if(E_bubble == 1) begin
227          E_stat <= 2'd0;
228          E_icode <= 4'd1; // nop
229          E_ifun <= 4'd0;
230          E_valC <= 64'd0;
231          E_valA <= 64'd0;
232          E_valB <= 64'd0;
233          E_dstE <= 4'd15;
234          E_dstM <= 4'd15;
235          E_srcA <= 4'd15;
236          E_srcB <= 4'd15;
237      end
238      else begin
239          E_stat <= D_stat;
240          E_icode <= D_icode;
241          E_ifun <= D_ifun;
242          E_valC <= D_valC;
243          E_valA <= valA;
244          E_valB <= valB;
245          E_dstE <= dstE;
246          E_dstM <= dstM;
247          E_srcA <= srcA;
248          E_srcB <= srcB;
249      end
250  end
251
252  // Write Back
253  always @(posedge clk)
254  begin
255      if(W_stall == 1)
256      begin
257      end
258      else
259      begin
260          if(W_icode == 4'd2) // cmovXX
261          begin
262              reg_mem[W_dstE] = W_valE;
263          end
264          else if(W_icode == 4'd3) // irmovq
265          begin
266              reg_mem[W_dstE] = W_valE;
267          end
268          else if(W_icode == 4'd5) // mrmovq
269          begin
270              reg_mem[W_dstM] = W_valM;
271          end
272          else if(W_icode == 4'd6) // OPq
273          begin
274              reg_mem[W_dstE] = W_valE;
275          end
276          else if(W_icode == 4'd8) // call
277          begin
278              reg_mem[W_dstE] = W_valE;
279          end
```

```
280             else if(W_icode == 4'd9) // ret
281             begin
282                 reg_mem[W_dstE] = W_valE;
283             end
284             else if(W_icode == 4'd10) // pushq
285             begin
286                 reg_mem[W_dstE] = W_valE;
287             end
288             else if(W_icode == 4'd11) // popq
289             begin
290                 reg_mem[W_dstE] = W_valE;
291                 reg_mem[W_dstM] = W_valM;
292             end
293         end
294 
295         rax <= reg_mem[0];
296         rcx <= reg_mem[1];
297         rdx <= reg_mem[2];
298         rbx <= reg_mem[3];
299         rsp <= reg_mem[4];
300         rbp <= reg_mem[5];
301         rsi <= reg_mem[6];
302         rdi <= reg_mem[7];
303         r8 <= reg_mem[8];
304         r9 <= reg_mem[9];
305         r10 <= reg_mem[10];
306         r11 <= reg_mem[11];
307         r12 <= reg_mem[12];
308         r13 <= reg_mem[13];
309         r14 <= reg_mem[14];
310 
311 
312     end
313 
314     endmodule
```

Testbench

```
`include "decode_wb.v"

module testbench;

reg clk;
reg [1:0]D_stat;
reg [3:0]D_icode;
reg [3:0]D_ifun;
reg[3:0] D_rA;
reg[3:0] D_rB;
reg signed [63:0]D_valC;
reg [63:0]D_valP;
reg E_bubble;
reg [3:0]e_dstE;
reg signed [63:0]e_valE;
reg [3:0]M_dstM;
reg signed [63:0]m_valM;
reg [3:0]M_dstE;
reg signed [63:0]M_valE;
reg [3:0]W_dstM;
reg signed [63:0]W_valM;
reg [3:0]W_dstE;
reg signed [63:0]W_valE;
reg [1:0]W_stat;
reg [3:0]W_icode;
reg W_stall;

wire [1:0]E_stat;
wire [3:0]E_icode;
wire [3:0]E_ifun;
wire [63:0]E_valC;
wire [63:0]E_valA;
wire [63:0]E_valB;
wire [3:0]E_dstE;
wire [3:0]E_dstM;
wire [3:0]E_srcA;
wire [3:0]E_srcB;
wire [63:0]rax, rcx, rdx, rbx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14;

decode_wb call(.clk(clk), .D_stat(D_stat), .D_icode(D_icode), .D_ifun(D_ifun), .D_rA(D_rA), .D_rB(D_rB), .D_valC(D_valC), .D_valP(D_valP), .E_bubble(E_bubble), .e_dstE(e_dstE), .e_valE(e_valE),
            .rax(rax), .rcx(rcx), .rdx(rdx), .rbx(rbx), .rsp(rsp), .rbp(rbp), .rsi(rsi), .rdi(rdi), .r8(r8), .r9(r9), .r10(r10), .r11(r11), .r12(r12), .r13(r13), .r14(r14));

initial
begin
    $dumpfile("decode_wb.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d: D_stat=%d, D_icode=%d, D_rA=%d, D_rB=%d, D_valC=%d, D_valP=%d, E_bubble=%d, E_stat=%d, E_icode=%d, E_valC=%d, E_valA=%d, E_valB=%d",clk,D_stat,D_icode,D_rA,D_rB,D_va
end

initial
begin
    // clk = 1;
    // D_stat = 2'd0; D_ifun = 4'd0; E_bubble = 0;
    // e_dstE = 4'd15; e_valE = 0;
    // M_dstM = 4'd15; m_valM = 0;
    // M_dstE = 4'd15; M_valE = 0;
    // W_dstM = 4'd15; W_valM = 0;
    // W_dstE = 4'd15; W_valE = 0;
    // W_stat = 2'd0; W_icode = 4'd1; W_stall = 0;
    // D_valC = 64'd1; D_valP = 64'd0;

    // #5 D_icode = 4'd2; D_rA = 4'd0; D_rB = 4'd1;

    // #10 D_icode = 4'd4; D_rA = 4'd1; D_rB = 4'd2;

    // #10 icode = 4'd5; rA = 4'd0; rB = 4'd3;

    // #10 icode = 4'd6; rA = 4'd1; rB = 4'd3;

    // #10 icode = 4'd8; rA = 4'd8; rB = 4'd0;

    // #10 icode = 4'd9; rA = 4'd9; rB = 4'd10;

    // #10 icode = 4'd10; rA = 4'd2; rB = 4'd0;

    // #10 icode = 4'd11; rA = 4'd9; rB = 4'd1;

    // #10 icode = 4'd3; rA = 4'd2; rB = 4'd8;
    #5 $finish;
end

endmodule
```

Working

The register file supports two simultaneous reads (ports A and B) and two simultaneous writes (ports E and M). Each port has both an address connection and a data connection, where address connection is a register ID (d_srcA, d_srcB, W_dstE, W_dstM) and the data connection is a set of 64 wires serving as either an output word (d_valA and d_valB) or an input word (W_valE and W_valM) of the register file. The special identifier 0xF (RNONE) on an address port indicates that no register should be accessed.

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

FORWARDING PRIORITY

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

If none of the forwarding conditions hold, the block should select the value read from register port A and B, as its output.

The overall processor state Stat is computed by a block based on the status value in pipeline register W. Since pipeline W holds the state of the most recently completed instruction, the value W_stat is used as an indication of the overall processor status. The only special case to consider is when there is a bubble in the write-back stage. We want the status code to represent a normal operation.

W_stat = 0 → AOK (Normal Operation)

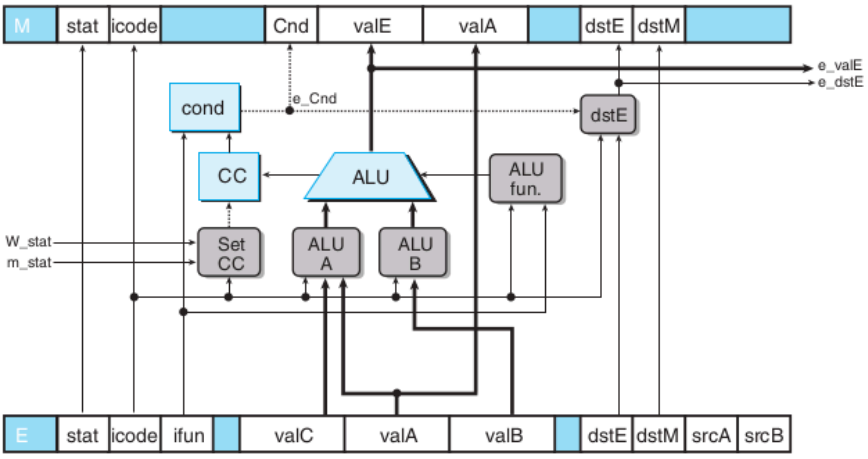
W_stat = 1 → HLT (Halt Instruction encountered)

W_stat = 2 → ADR (Bad Instruction/Data Memory accessed)

W_stat = 3 → INS (Invalid Instruction fetched)

EXECUTE

In the execute stage, the arithmetic logic unit (ALU) either performs the operation specified by the instruction, computes the effective address of a memory reference or increments or decrements the stack pointer by computing and producing the output valE. The condition codes are also possibly set in this stage.



Code

```
`include "./ALU/alu.v"

module execute(
    input clk,
    input [1:0]E_stat,
    input [3:0]E_icode,
    input [3:0]E_ifun,
    input signed[63:0]E_valC,
    input signed[63:0]E_valA,
    input signed[63:0]E_valB,
    input [3:0]E_dstE,
    input [3:0]E_dstM,
    input set_cc,
    input M_bubble,
    output reg[1:0]M_stat,
    output reg[3:0]M_icode,
    output reg M_Cnd,
    output reg[63:0]M_valE,
    output reg[63:0]M_valA,
    output reg[63:0]e_valE,
    output reg[3:0]M_dstE,
    output reg[3:0]M_dstM,
    output reg[3:0]e_dstE,
    output reg e_Cnd,
    output reg c_z,
    output reg c_s,
    output reg c_o
);

reg [1:0]control;
reg signed [63:0]A;
reg signed [63:0]B;
wire signed [63:0]result;
wire overflow;

alu call(.control(control), .A(A), .B(B), .result(result), .overflow(overflow));
```

```
38  reg zf; // Zero flag
39  reg sf; // Sign flag
40  reg of; // Overflow flag
41
42  reg [3:0]icode;
43  reg [3:0]ifun;
44  reg signed [63:0]valA;
45  reg signed [63:0]valB;
46  reg [63:0]valC;
47  reg signed [63:0]valE;
48  reg Cnd;
49
50  initial begin
51      zf = 0;
52      sf = 0;
53      of = 0;
54      Cnd = 1'd0;
55      valE = 64'd0;
56  end
57
58  always @(*) begin
59      if(icode == 4'd6 && clk == 1 && set_cc == 1) // OPq
60      begin
61          zf = (result == 64'd0);
62          sf = (result < 64'd0);
63          // sf = (result[63] == 1);
64          of = ((A < 64'd0 == B < 64'd0) && (result < 64'd0 != A < 64'd0));
65          // of = (((A[63] == 1) == (B[63] == 1)) && ((result[63] == 1) ~= (A[63] == 1)));
66          // of = overflow;
67      end
68      // $display("icode=%d,ifun=%d,zf=%d, sf=%d, of=%d",icode,ifun,zf,sf,of);
69      c_z = zf; c_s = sf; c_o = of;
70
71  end
72
73  always @(*) begin
74      e_dstE = E_dstE;
75      icode = E_icode;
76      ifun = E_ifun;
77      valA = E_valA;
78      valB = E_valB;
79      valC = E_valC;
80      e_valE = valE;
81      e_Cnd = Cnd;
82  end
83
84  always @(*) begin
85      Cnd = 0;
86      if(icode == 4'd2) // cmovXX
87      begin
88          if(ifun == 4'd0) // rrmovq
89          begin
90              Cnd = 1;
91          end
```

```
92          else if(ifun == 4'd1) // cmovle
93          begin
94              // (sf^of)||zf
95              if((sf^of)|zf) begin
96                  Cnd = 1;
97              end
98          end
99          else if(ifun == 4'd2) // cmovl
100         begin
101             // (sf^of)
102             if((sf^of)) begin
103                 Cnd = 1;
104             end
105         end
106         else if(ifun == 4'd3) // cmove
107         begin
108             // zf
109             if(zf) begin
110                 Cnd = 1;
111             end
112         end
113         else if(ifun == 4'd4) // cmovne
114         begin
115             // !zf
116             if(~zf) begin
117                 Cnd = 1;
118             end
119         end
120         else if(ifun == 4'd5) // cmovge
121         begin
122             // !(sf^of)
123             if(~(sf^of)) begin
124                 Cnd = 1;
125             end
126         end
127         else if(ifun == 4'd6) // cmovg
128         begin
129             // !(sf^of) && !zf
130             if((~(sf^of)) & ~zf) begin
131                 Cnd = 1;
132             end
133         end
134         // control = 2'd0;
135         // A = valB;
136         // B = valA;
137         // valE = result;
138         if(e_Cnd == 1)
139         begin
140             e_dstE = E_dstE;
141         end
142         else
143         begin
144             e_dstE = 4'd15;
145         end
146     end
```

```
147     else if(icode == 4'd3) // irmovq
148     begin
149         control = 2'd0;
150         A = 64'd0;
151         B = valC;
152         valE = result;
153         // valE = 64'd0 + valC;
154     end
155     else if(icode == 4'd4) // rmmovq
156     begin
157         control = 2'd0;
158         A = valB;
159         B = valC;
160         valE = result;
161         // valE = valB + valC;
162     end
163     else if(icode == 4'd5) // mrmovq
164     begin
165         control = 2'd0;
166         A = valB;
167         B = valC;
168         valE = result;
169         // valE = valB + valC;
170     end
171     else if(icode == 4'd6) // OPq
172     begin
173         A = valA;
174         B = valB;
175         if(ifun == 4'd0) // add
176         begin
177             control = 2'd0;
178         end
179         else if(ifun == 4'd1) // sub
180         begin
181             A = valB;
182             B = valA;
183             control = 2'd1;
184         end
185         else if(ifun == 4'd2) // and
186         begin
187             control = 2'd2;
188         end
189         else if(ifun == 4'd3) // xor
190         begin
191             control = 2'd3;
192         end
193         valE = result;
194     end
195     else if(icode == 4'd7) // jxx
196     begin
197         if(ifun == 4'd0) // jmp
198         begin
199             Cnd = 1;
200         end
```

```
201         else if(ifun == 4'd1) // jle
202         begin
203             // (sf^of)||zf
204             if((sf^of)|zf) begin
205                 Cnd = 1;
206             end
207         end
208         else if(ifun == 4'd2) // jl
209         begin
210             // (sf^of)
211             if((sf^of)) begin
212                 Cnd = 1;
213             end
214         end
215         else if(ifun == 4'd3) //je
216         begin
217             // zf
218             if(zf) begin
219                 Cnd = 1;
220             end
221         end
222         else if(ifun == 4'd4) //jne
223         begin
224             // !zf
225             if(!zf) begin
226                 Cnd = 1;
227             end
228         end
229         else if(ifun == 4'd5) //jge
230         begin
231             // !(sf^of)
232             if(!(sf^of)) begin
233                 Cnd = 1;
234             end
235         end
236         else if(ifun == 4'd6) //jg
237         begin
238             // !(sf^of) && !zf
239             if((!(sf^of)) && !zf) begin
240                 Cnd = 1;
241             end
242         end
243         if(e_Cnd == 1)
244         begin
245             e_dstE = E_dstE;
246         end
247         else
248         begin
249             e_dstE = 4'd15;
250         end
251     end
252     else if(icode == 4'd8) // call
253     begin
254         control = 2'd1;
255         A = valB;
256         B = 64'd8;
```

```
257             valE = result;
258         end
259         else if(icode == 4'd9) // ret
260         begin
261             control = 2'd0;
262             A = valB;
263             B = 64'd8;
264             valE = result;
265         end
266         else if(icode == 4'd10) // pushq
267         begin
268             control = 2'd1;
269             A = valB;
270             B = 64'd8;
271             valE = result;
272         end
273         else if(icode == 4'd11) // popq
274         begin
275             control = 2'd0;
276             A = valB;
277             B = 64'd8;
278             valE = result;
279         end
280     end
281
282     always @(posedge clk)
283     begin
284         if(M_bubble == 1) begin
285             M_stat <= 2'd0;
286             M_icode <= 4'd1; // nop
287             M_Cnd <= 1'd0;
288             M_valE <= 64'd0;
289             M_valA <= 64'd0;
290             M_dstE <= 4'd15;
291             M_dstM <= 4'd15;
292         end
293         else begin
294             M_stat <= E_stat;
295             M_icode <= E_icode;
296             M_Cnd <= e_Cnd;
297             M_valE <= e_valE;
298             M_valA <= E_valA;
299             M_dstE <= e_dstE;
300             M_dstM <= E_dstM;
301         end
302     end
303
304     endmodule
```

Testbench

```
`include "decode_wb.v"

module testbench;

reg clk;
reg [1:0]E_stat;
reg [3:0]E_icode;
reg [3:0]E_ifun;
reg signed [63:0]E_valC;
reg signed [63:0]E_valA;
reg signed [63:0]E_valB;
reg [3:0]E_dstE;
reg [3:0]E_dstM;
reg set_cc;
reg M_bubble;

wire [1:0]M_stat;
wire [3:0]M_icode;
wire [3:0]M_Cnd;
wire [63:0]M_valE;
wire [63:0]M_valA;
wire [63:0]e_valE;
wire [3:0]M_dstE;
wire [3:0]M_dstM;
wire [3:0]e_dstE;
wire e_Cnd;

decode_wb call(.clk(clk), .E_stat(E_stat), .E_icode(E_icode), .E_ifun(E_ifun),
.E_valC(E_valC), .E_valA(E_valA), .E_valB(E_valB), .E_dstE(E_dstE),
.E_dstM(E_dstM), .set_cc(set_cc), .M_bubble(M_bubble), .M_stat(M_stat), .M_icode(M_icode),
.M_valE(M_valE), .M_valA(M_valA), .e_valE(e_valE), .e_dstE(e_dstE), .M_dstM(M_dstM), .e_dstE(e_dstE),
.e_Cnd(e_Cnd));

initial
begin
    $dumpfile("execute.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk %d: E_stat=%d, E_icode=%d, E_valC=%d, E_valA=%d, E_valB=%d, set_cc=%d, M_bubble=%d, M_stat=%d, M_icode=%d, M_Cnd=%d, M_valE=%d, M_valA=%d, e_valE=%d",clk,D_stat,D_icode,
end

initial
begin
    clk = 0; E_stat = 2'd0;
    #5 $finish;
end

endmodule
```

Working

The main operation in the execution stage is performed by the ALU unit (Arithmetic Logic Unit). This unit performs the operation of add, subtract, and, or xor on inputs aluA and aluB based on the setting of the ALUfun. The ALU output becomes the signal e_valE.

The memory locations are computed by adding E_valB and E_valC (displacement) in instructions which require access to data memory such as rmmovq and mrmovq. The stack pointer is updated by incrementing/decrementing the value stored in %rsp register by 8 in instructions call, ret, pushq and popq.

The execution stage also includes the condition code register. The ALU generates three signals on which the condition codes are based - zero,sign and overflow - everytime it operates. The signal set_cc determines when the condition codes are to be updated. The condition codes are updated when an OPq instruction is executed, provided that the status codes in the memory block and write back stage are not set to that of a halt instruction, memory error or an invalid instruction. In such a case, the set_cc signal is set 0 and the condition codes are not updated. The condition codes are evaluated by the execution stage along with the move conditions and it is then determined if the destination register will be updated in a conditional move instruction. Similarly, for a jump instruction, the condition codes determine whether or not the branch should be taken.

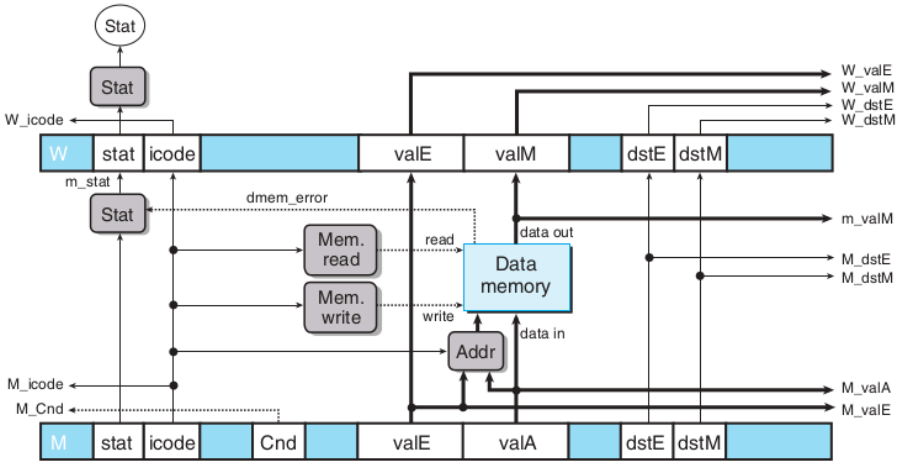
The signals e_valE and e_dstE are directed towards the decode stage as one of the forwarding sources.

The pipeline register bank E sits between the decode and the execute stages. It holds the information about the most recently decoded instruction and the value read from the register file for processing by the execute stage.

The pipeline register bank M sits between the execute and the 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.

MEMORY

The memory stage may write data to memory, or it may read data from memory and output the read value as m_valM.



Code

```
module memory(
input clk,
input [3:0]M_icode,
input signed[63:0]M_valA,
input signed[63:0]M_valE,
input [3:0]M_dstE,
input [3:0]M_dstM,
input [1:0]M_stat,
input W_stall,
output reg [3:0]W_icode,
output reg [63:0]W_valE,
output reg [63:0]W_valM,
output reg [3:0]W_dstE,
output reg [3:0]W_dstM,
output reg [1:0]W_stat,
output reg [63:0]m_valM,
output reg [1:0]m_stat
);

reg [63:0]data_mem[0:1023];
reg read,write;
reg [63:0]data_in;
reg [63:0]address;
reg dmem_error;

initial
begin
    address = 64'd0;
    m_valM = 64'd0;
    data_in = 64'd0;
    dmem_error = 0;
end
```

```
34  always @(*)
35  begin
36
37      read = 0; write = 0;
38
39      if(M_icode == 4'd4) // rmmovq
40      begin
41          data_in = M_valA;
42          address = M_valE;
43          write = 1;
44      end
45      else if(M_icode == 4'd5) // mrmovq
46      begin
47          address = M_valE;
48          read = 1;
49      end
50      else if(M_icode == 4'd8) // call
51      begin
52          data_in = M_valA;
53          address = M_valE;
54          write = 1;
55      end
56      else if(M_icode == 4'd9) // ret
57      begin
58          address = M_valA;
59          read = 1;
60      end
61      else if(M_icode == 4'd10) // pushq
62      begin
63          data_in = M_valA;
64          address = M_valE;
65          write = 1;
66      end
67      else if(M_icode == 4'd11) // popq
68      begin
69          address = M_valA;
70          read = 1;
71      end
72
73      // $display("address = %d",address);
74      // $display("read %d, write = %d",read,write);
75      if(address>=0 && address<1024)
76      begin
77          if(read == 1 && write == 0)
78          begin
79              m_valM = data_mem[address];
80              // $display("m_valM = %d",m_valM);
81          end
82          else if(write == 1 && read == 0)
83          begin
84              data_mem[address] = data_in;
85          end
86          else if(read == 1 && write == 1)
87          begin
88              dmem_error = 1;
89          end
90      end
```

```
91      else
92      begin
93          dmem_error = 1;
94      end
95
96      if(dmem_error == 1 && M_stat == 0)
97      begin
98          m_stat = 2'd2;
99      end
100     else
101     begin
102         m_stat = M_stat;
103     end
104 end
105
106 always @(posedge clk)
107 begin
108
109     if(W_stall == 0)
110     begin
111
112         W_icode <= M_icode;
113         W_valE <= M_valE;
114         W_valM <= m_valM;
115         W_dstE <= M_dstE;
116         W_dstM <= M_dstM;
117         W_stat <= m_stat;
118     end
119
120 end
121 endmodule
```

Testbench

```
`include "memory.v"

module testbench;

reg clk,W_stall;
reg [3:0]M_icode,M_dstE,M_dstM;
reg [63:0]M_valA,M_valE;
reg [1:0]M_stat;
wire [3:0]W_icode,W_dstE,W_dstM;
wire [63:0]W_valE,W_valM;
wire [1:0]W_stat;

memory call(.clk(clk), .M_icode(M_icode), .M_valA(M_valA), .M_valE(M_valE), .M_dstE(M_dstE), .M_dstM(M_dstM), .M_stat(M_stat), .W_stall(W_stall), .W_icode(W_icode), .W_valE(W_valE), .W_valM(W_va

initial
begin
    $dumpfile("memory.vcd");
    $dumpvars(0,testbench);
end

always #5 clk = ~clk;
initial
begin
    $monitor($time," clk=%d: icode=%d, valA=%d, valE=%d, valM=%d, stat=%d, W_icode=%d",clk,M_icode,M_valA,M_valE,W_valM,W_stat,W_icode);
end

initial
begin
    clk = 1;

    #5 M_icode = 4'd4; M_valA = 64'd100; M_valE = 64'd24; W_stall = 0;

    #10 M_icode = 4'd5; M_valA = 64'd15; M_valE = 64'd8; W_stall = 0;

    #10 M_icode = 4'd8; M_valA = 64'd6; M_valE = 64'd32; W_stall = 0;

    #10 M_icode = 4'd9; M_valA = 64'd32; M_valE = 64'd0; W_stall = 0;

    #10 M_icode = 4'd10; M_valA = 64'd10; M_valE = 64'd20; W_stall = 0;

    #10 M_icode = 4'd11; M_valA = 64'd20; M_valE = 64'd0; W_stall = 0;

    #10 M_icode = 4'd4; M_valA = 64'd0; M_valE = 64'd1200; W_stall = 1;

    #10 M_icode = 4'd3; M_valA = 64'd1; M_valE = 64'd1; W_stall = 0;
    #5 $finish;
end
endmodule
```

Working

The memory stage has the task of either reading or writing program data. Two control blocks generate the values for the memory address and the memory input data (for write operations). Two other blocks generate the control signals indicating whether to perform a read or a write operation. When a read operation is performed, the data memory generates the value m_valM. The address is either the value of M_valA or M_valE. The value to be written is M_valA, is either the value of d_rvalA or d_valP or values obtained from forwarded signals from execute,memory and write back stage, determined by the "Sel+Fwd" block in the decode stage and passed as d_valA to further stages.

If an out of bounds is accessed, a dmem_error flag is raised. If Stat is 0, it is updated to represent a memory error.

The signals m_valM, M_dstE, M_dstM,M_valA and M_valE are directed towards the decode stage as one of the forwarded values. M_icode and M_Cnd are towards the pipeline control logic.M_valA is also directed to the select PC block and used in case of a mispredicted branch in jump instruction.

The pipeline register bank 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.

Combined Pipelined Processor

```
`include "F_reg.v"
`include "fetch.v"
`include "decode_wb.v"
`include "execute.v"
`include "memory.v"
`include "pipeline_control_logic.v"

module testbench;

reg clk;
reg [63:0]F_predPC;
wire [3:0]D_icode,D_ifun,D_rA,D_rB;
wire signed[63:0]D_valP,D_valC,f_predPC;
wire [1:0]D_stat;
wire [3:0]E_icode,E_ifun,E_dstE,E_dstM,E_srcA,E_srcB;
wire signed[63:0]E_valC,E_valA,E_valB,rax,rcx,rdx,rbx,rsp,rbp,rsi,rdi,r8,r9,r10,r11,r12,r13,r14;
wire [1:0]E_stat;
wire [3:0]M_icode,M_dstE,M_dstM,e_dstE;
wire signed[63:0]M_valE,M_valA,e_valE;
wire [1:0]M_stat;
wire e_Cnd, M_Cnd;
wire [3:0]W_icode,W_dstE,W_dstM;
wire signed[63:0]W_valE,W_valM;
wire [1:0]W_stat;
wire F_stall,D_stall,D_bubble,E_bubble,M_bubble,W_stall,set_cc;
wire [3:0]d_srcA,d_srcB;
wire signed[63:0]m_valM;
wire [1:0]m_stat;
wire c_z,c_o,c_s;

reg [1:0]stat; // status conditions

// F_reg fr(.clk(clk),.f_predPC(f_predPC),.F_stall(F_stall),.F_predPC(F_predPC));
fetch fet(.clk(clk),.F_predPC(F_predPC),.M_valA(M_valA),.W_valM(W_valM),.M_icode(M_icode),.M_Cnd(M_Cnd),.W_icode(W_icode),.D_stall(D_stall),.D_bubble(D_bubble),.D_icode(D_icode),.D_ifun(D_ifun)
decode_wb dw(.clk(clk),.D_stat(D_stat),.D_icode(D_icode),.D_ifun(D_ifun),.D_rA(D_rA),.D_rB(D_rB),.D_valC(D_valC),.D_valP(D_valP),.E_bubble(E_bubble),.e_dstE(e_dstE),.e_valE(e_valE),.M_dstM(M_ds
             .W_valE(W_valE),.W_stat(W_stat),.W_icode(W_icode),.W_stall,.E_stat(E_stat),.E_icode(E_icode),.E_ifun(E_ifun),.E_valC(E_valC),.E_valA(E_valA),.E_valB(E_valB),.E_dstE(E_dstE),.E_dstM
             .rsi(rsi),.rdi(rdi),.r8(r8),.r9(r9),.r10(r10),.r11(r11),.r12(r12),.r13(r13),.r14(r14),.d_srcA(d_srcA),.d_srcB(d_srcB));
execute exe(.clk(clk),.E_stat(E_stat),.E_icode(E_icode),.E_ifun(E_ifun),.E_valC(E_valC),.E_valA(E_valA),.E_valB(E_valB),.E_dstE(E_dstE),.E_dstM(E_dstM),.set_cc(set_cc),.M_bubble(M_bubble),.M_st
memory mry(.clk(clk),.M_icode(M_icode),.M_valA(M_valA),.M_valE(M_valE),.M_dstE(M_dstE),.M_dstM(M_dstM),.M_stat(M_stat),.W_stall(W_stall),.W_icode(W_icode),.W_valE(W_valE),.W_valM(W_valM),.W_dst
pipe_control_logic pcl(.D_icode(D_icode),.E_icode(E_icode),.M_icode(M_icode),.m_stat(m_stat),.W_stat(W_stat),.d_srcA(d_srcA),.d_srcB(d_srcB),.E_dstM(E_dstM),.e_Cnd(e_Cnd),.F_stall(F_stall),.D_s


always @(W_stat)
begin
    stat = W_stat;
end

always @(stat)
begin

    if(stat != 0)
    begin
        if(stat == 1)
        begin
            $display($time ," Halt");
            $finish;
        end
        else if(stat == 2)
        begin
            $display($time ," Memory Error");
            $finish;
        end
        else if(stat == 3)
        begin
            $display($time ," Instruction Invalid");
            $finish;
        end
    end
end


always @(posedge clk)
begin

    if(F_stall == 0)
    begin
        F_predPC <= f_predPC;
    end
end

always #5 clk = ~clk;

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

    clk = 0;
    F_predPC = 64'd0;
end


always @(negedge clk)
begin
    $display($time," clk : %d",clk);
    $display("FETCH STAGE-----------------------------------------------------------------------------------------------");
    $display("icode=%d, ifun=%d, rA=%d, rB=%d, valP=%d, valC=%d, predPC=%d, stat=%d", D_icode,D_ifun,D_rA,D_rB,D_valP,D_valC,f_predPC,D_stat);
    $display("DECODE STAGE----------------------------------------------------------------------------------------------");
    $display("icode=%d, E_valA=%d, E_valB=%d, stat=%d", E_icode, E_valA, E_valB, E_stat);
    $display("EXECUTE STAGE---------------------------------------------------------------------------------------------");
    $display("icode=%d, valE=%d, valA=%d, stat=%d, set_cc=%d", M_icode, M_valE, M_valA, M_stat,set_cc);
    $display("MEMORY STAGE----------------------------------------------------------------------------------------------");
    $display("icode=%d, stat=%d, valM=%d", W_icode, W_stat, W_valM);
    $display("F_stall=%d, D_stall=%d, D_bubble=%d, E_bubble=%d, M_bubble=%d, W_stall=%d",F_stall,D_stall,D_bubble,E_bubble,M_bubble,W_stall);
    $display("zf = %d, sf = %d, of = %d",c_z,c_s,c_o);
end
```

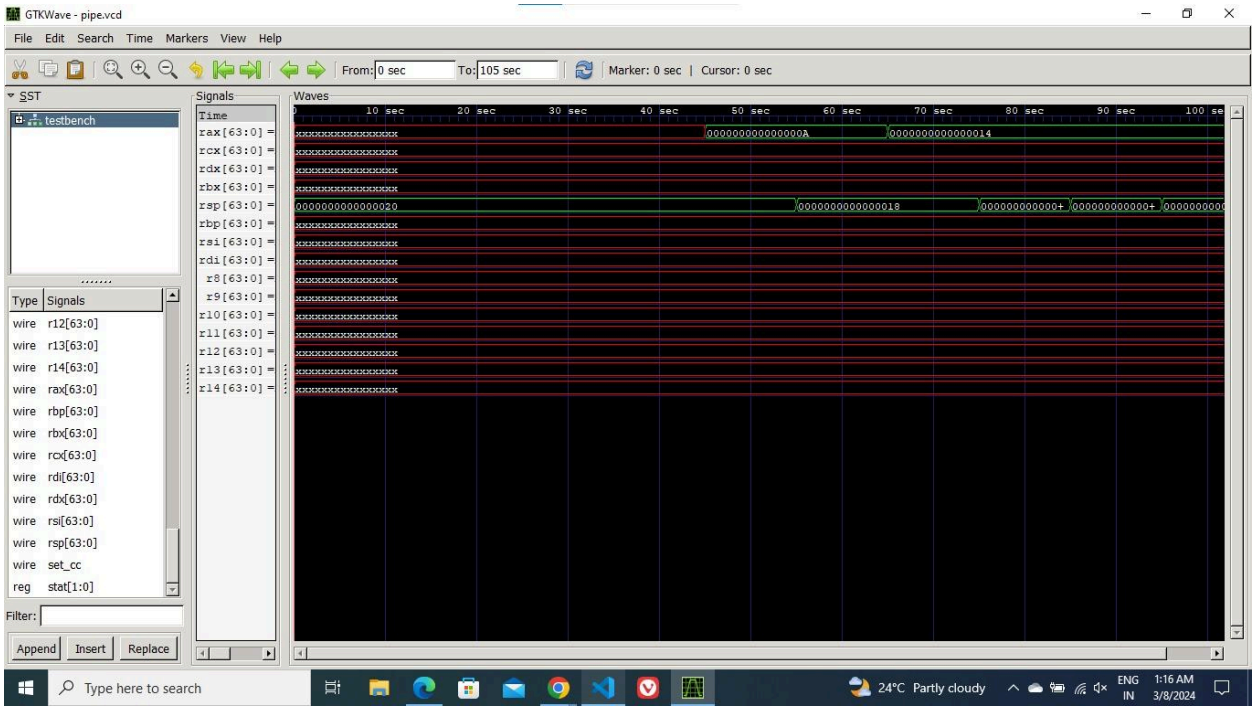
Testcase

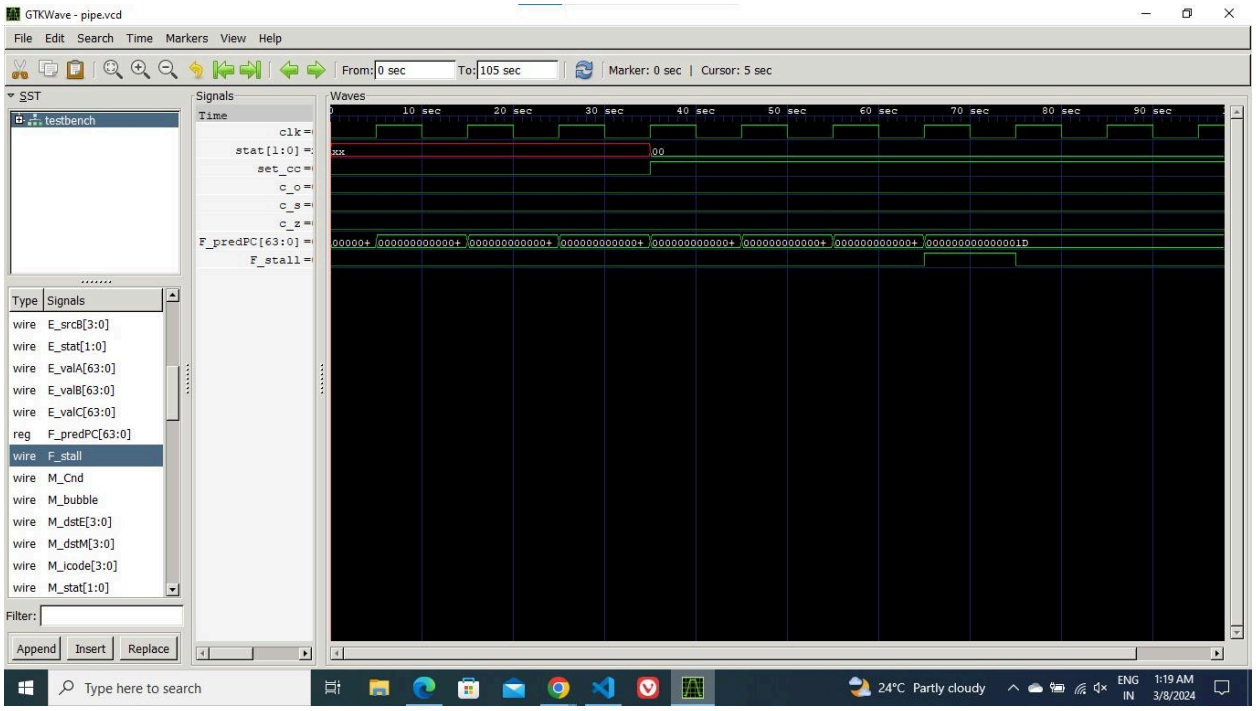
```
_start:
    # Pushing values onto the stack
    movq $10, %rax   # Move the value 10 into the register %rax
    pushq %rax       # Push the value of %rax onto the stack
    movq $20, %rax   # Move the value 20 into the register %rax
    pushq %rax       # Push the value of %rax onto the stack

    # Popping values from the stack
    popq %rax        # Pop the top value from the stack into %rax
    popq %rbx        # Pop the next value from the stack into %rbx
```

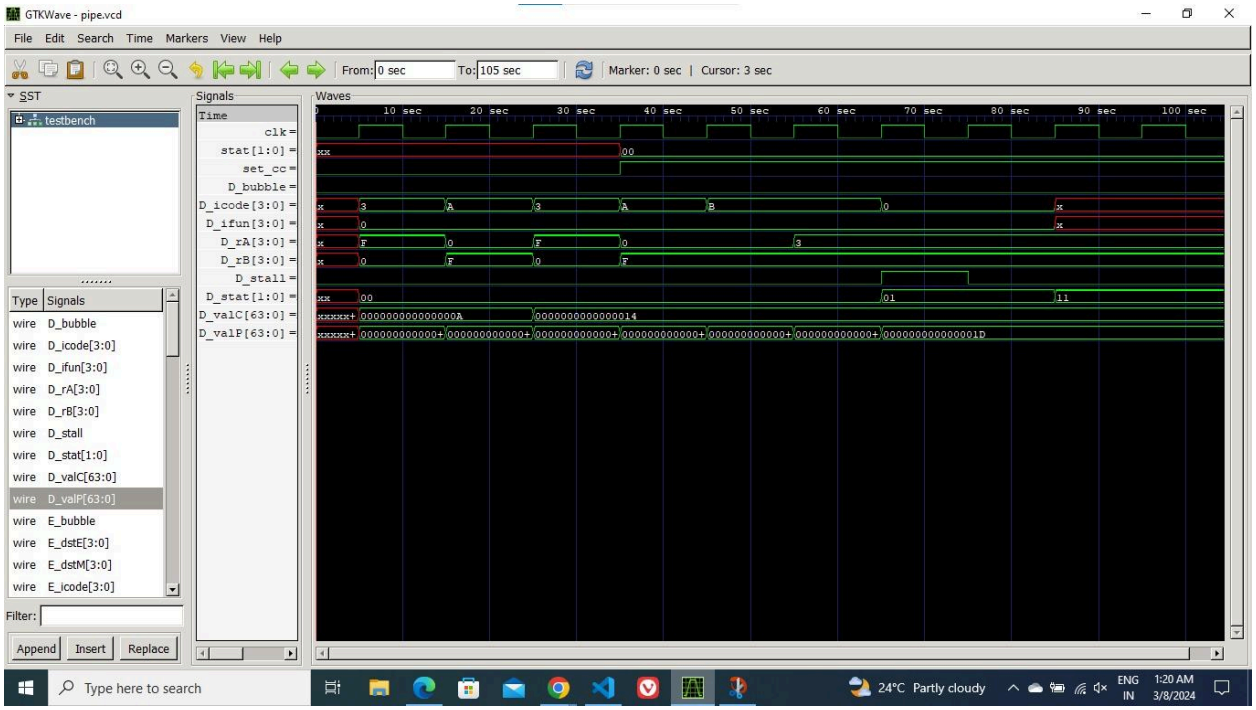
```
00110000
11110000
00001010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10100000
00001111
00110000
11110000
00010100
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10100000
00001111
10110000
00001111
10110000
00111111
00000000
```

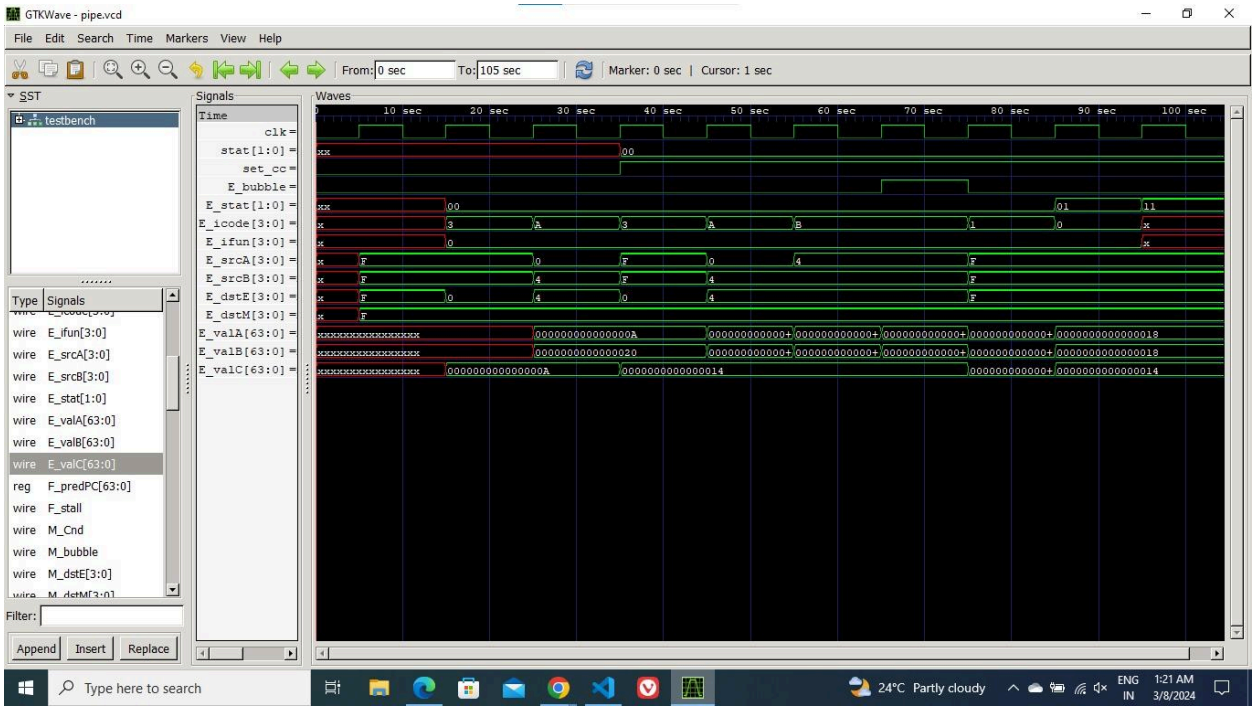
```
VCD info: dumpfile pipe.vcd opened for output.
                10 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 0, valP=                 10, valC=                 10, predPC=                 12, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= x, E_valA=                  x, E_valB=                  x, stat=x
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                20 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode=10, ifun= 0, rA= 0, rB=15, valP=                 12, valC=                 10, predPC=                 22, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                30 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 0, valP=                 22, valC=                 20, predPC=                 24, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode=10, E_valA=                 10, E_valB=                 32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 3, valE=                 10, valA=                  x, stat=0, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                40 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode=10, ifun= 0, rA= 0, rB=15, valP=                 24, valC=                 20, predPC=                 26, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 3, E_valA=                 10, E_valB=                 32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode=10, valE=                 24, valA=                 10, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                50 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode=11, ifun= 0, rA= 0, rB=15, valP=                 26, valC=                 20, predPC=                 28, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode=10, E_valA=                 20, E_valB=                 24, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 3, valE=                 20, valA=                 10, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode=10, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
```

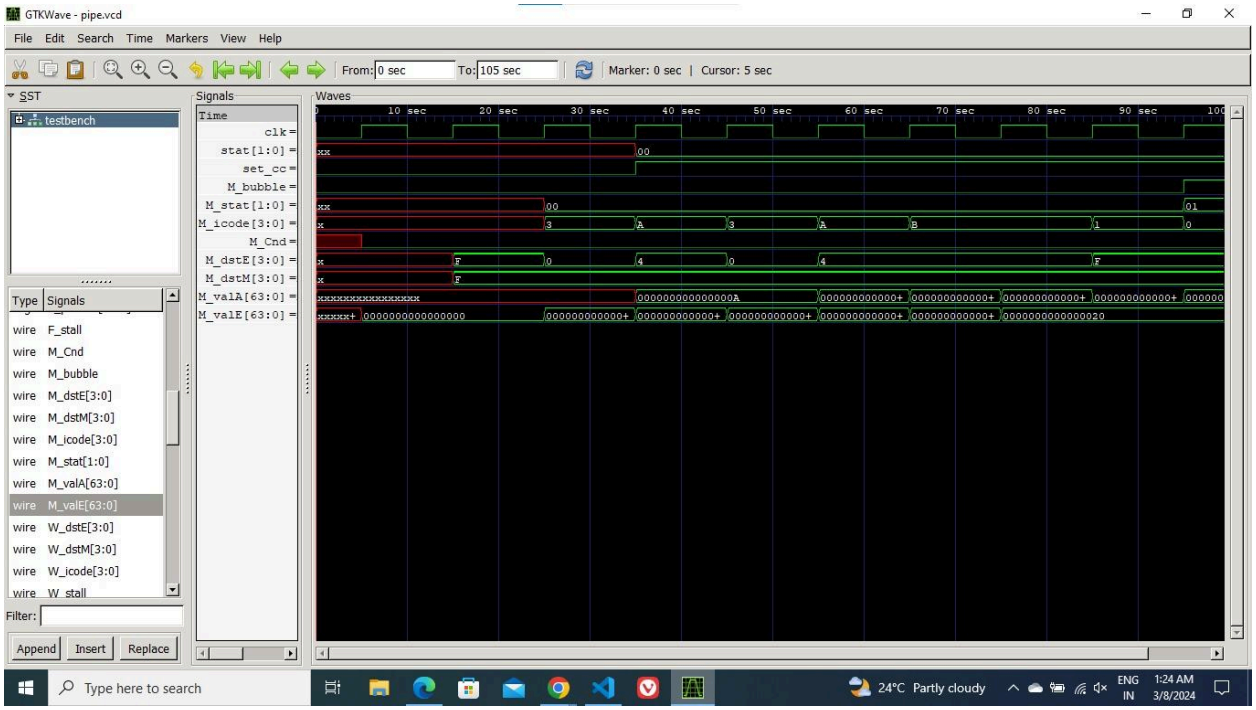
```
                60 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode=11, ifun= 0, rA= 3, rB=15, valP=                28, valC=                20, predPC=                29, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode=11, E_valA=                16, E_valB=                16, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode=10, valE=                16, valA=                20, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                 0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                70 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 0, ifun= 0, rA= 3, rB=15, valP=                29, valC=                20, predPC=                29, stat=1
DECODE STAGE----------------------------------------------------------------------------------------------------
icode=11, E_valA=                24, E_valB=                24, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode=11, valE=                24, valA=                16, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode=10, stat=0, valM=                 0
F_stall=1, D_stall=1, D_bubble=0, E_bubble=1, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                80 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 0, ifun= 0, rA= 3, rB=15, valP=                29, valC=                20, predPC=                29, stat=1
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 1, E_valA=                 0, E_valB=                 0, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode=11, valE=                32, valA=                24, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode=11, stat=0, valM=                20
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                90 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= x, ifun= x, rA= 3, rB=15, valP=                29, valC=                20, predPC=                29, stat=3
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 0, E_valA=                24, E_valB=                24, stat=1
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 1, valE=                32, valA=                 0, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode=11, stat=0, valM=                10
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
               100 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= x, ifun= x, rA= 3, rB=15, valP=                29, valC=                20, predPC=                29, stat=3
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= x, E_valA=                24, E_valB=                24, stat=3
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 0, valE=                32, valA=                24, stat=1, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= 1, stat=0, valM=                10
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=1, W_stall=0
zf = 0, sf = 0, of = 0
                105 Halt
```

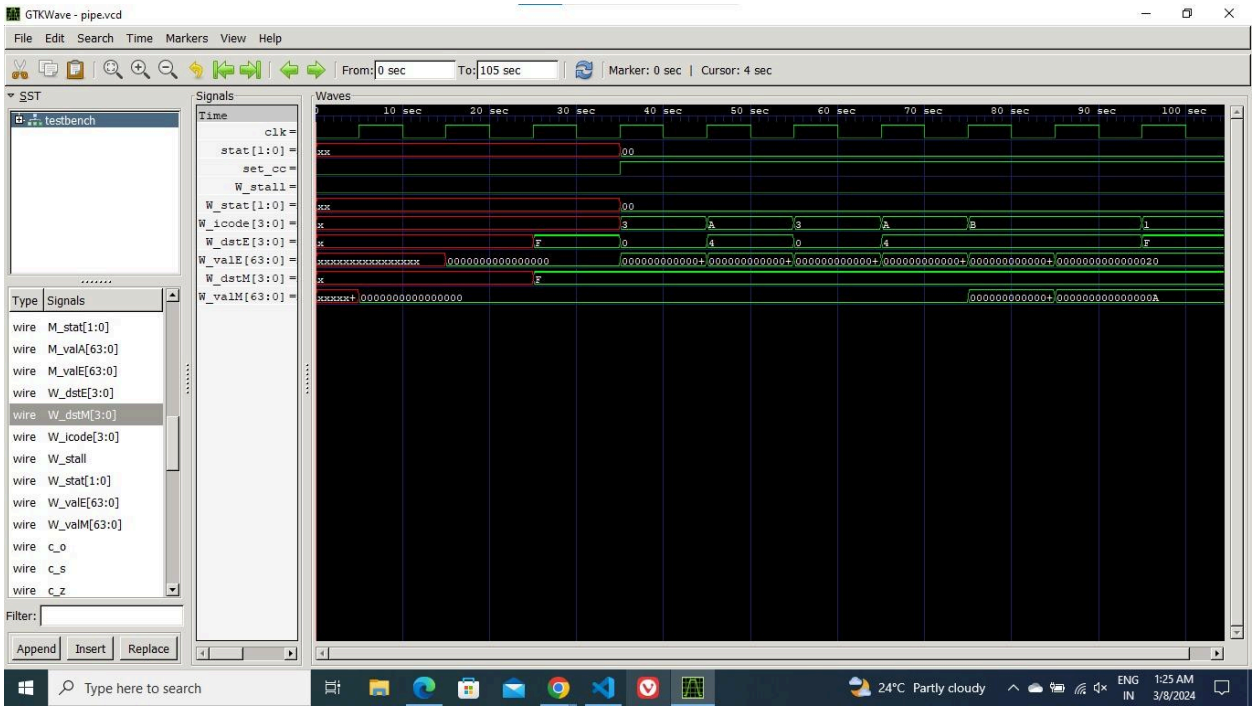

GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 0 sec
SST
testbench
Signals
Waves
Time
rax[63:0]
rcx[63:0]
rdx[63:0]
rbx[63:0]
rsp[63:0]
rbp[63:0]
rsi[63:0]
rdi[63:0]
r8[63:0]
r9[63:0]
r10[63:0]
r11[63:0]
r12[63:0]
r13[63:0]
r14[63:0]
Type Signals
wire r12[63:0]
wire r13[63:0]
wire r14[63:0]
wire rax[63:0]
wire rbp[63:0]
wire rbx[63:0]
wire rcx[63:0]
wire rdi[63:0]
wire rdx[63:0]
wire rsi[63:0]
wire rsp[63:0]
wire set_cc
reg stat[1:0]
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:16 AM
3/8/2024


GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 5 sec
SST
testbench
Signals
Waves
Time
clk
stat[1:0]
set_cc
c_o
c_s
c_z
F_predPC[63:0]
F_stall
Type Signals
wire E_srcB[3:0]
wire E_stat[1:0]
wire E_valA[63:0]
wire E_valB[63:0]
wire E_valC[63:0]
reg F_predPC[63:0]
wire F_stall
wire M_Cnd
wire M_bubble
wire M_dstE[3:0]
wire M_dstM[3:0]
wire M_icode[3:0]
wire M_stat[1:0]
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:19 AM
3/8/2024


GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 3 sec
SST
testbench
Signals
Waves
Time
clk
stat[1:0]
set_cc
D_bubble
D_icode[3:0]
D_ifun[3:0]
D_rA[3:0]
D_rB[3:0]
D_stall
D_stat[1:0]
D_valC[63:0]
D_valP[63:0]
Type Signals
wire D_bubble
wire D_icode[3:0]
wire D_ifun[3:0]
wire D_rA[3:0]
wire D_rB[3:0]
wire D_stall
wire D_stat[1:0]
wire D_valC[63:0]
wire D_valP[63:0]
wire E_bubble
wire E_dstE[3:0]
wire E_dstM[3:0]
wire E_icode[3:0]
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:20 AM 3/8/2024


GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 1 sec
SST
testbench
Signals
Waves
Time
clk
stat[1:0]
set_cc
E_bubble
E_stat[1:0]
E_icode[3:0]
E_ifun[3:0]
E_srcA[3:0]
E_srcB[3:0]
E_dstE[3:0]
E_dstM[3:0]
E_valA[63:0]
E_valB[63:0]
E_valC[63:0]
Type Signals
wire E_ifun[3:0]
wire E_srcA[3:0]
wire E_srcB[3:0]
wire E_stat[1:0]
wire E_valA[63:0]
wire E_valB[63:0]
wire E_valC[63:0]
reg F_predPC[63:0]
wire F_stall
wire M_Cnd
wire M_bubble
wire M_dstE[3:0]
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:21 AM 3/8/2024


GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 5 sec
SST
testbench
Signals
Time
clk
stat[1:0]
set_cc
M_bubble
M_stat[1:0]
M_icode[3:0]
M_Cnd
M_dstE[3:0]
M_dstM[3:0]
M_valA[63:0]
M_valE[63:0]
Waves
Type Signals
wire F_stall
wire M_Cnd
wire M_bubble
wire M_dstE[3:0]
wire M_dstM[3:0]
wire M_icode[3:0]
wire M_stat[1:0]
wire M_valA[63:0]
wire M_valE[63:0]
wire W_dstE[3:0]
wire W_dstM[3:0]
wire W_icode[3:0]
wire W_stall
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:24 AM
3/8/2024


GTKWave - pipe.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 105 sec
Marker: 0 sec | Cursor: 4 sec
SST
testbench
Signals
Time
clk
stat[1:0]
set_cc
W_stall
W_stat[1:0]
W_icode[3:0]
W_dstE[3:0]
W_valE[63:0]
W_dstM[3:0]
W_valM[63:0]
Waves
Type Signals
wire M_stat[1:0]
wire M_valA[63:0]
wire M_valE[63:0]
wire W_dstE[3:0]
wire W_dstM[3:0]
wire W_icode[3:0]
wire W_stall
wire W_stat[1:0]
wire W_valE[63:0]
wire W_valM[63:0]
wire c_o
wire c_s
wire c_z
Filter:
Append
Insert
Replace
Type here to search
24°C Partly cloudy
ENG IN
1:25 AM
3/8/2024

Pipeline Control Logic

```
module pipe_control_logic(
    input [3:0]D_icode,
    input [3:0]E_icode,
    input [3:0]M_icode,
    input [1:0]m_stat,
    input [1:0]W_stat,
    input [3:0]d_srcA,
    input [3:0]d_srcB,
    input [3:0]E_dstM,
    input e_Cnd,
    output reg F_stall,
    output reg D_stall,
    output reg D_bubble,
    output reg E_bubble,
    output reg M_bubble,
    output reg W_stall,
    output reg set_cc
);

always@(*) begin

    F_stall = 0;
    D_stall = 0;
    D_bubble = 0;
    E_bubble = 0;
    M_bubble = 0;
    W_stall = 0;
    set_cc = 0;

    if(((E_icode == 4'd5 || E_icode == 4'd11) && (E_dstM == d_srcA || E_dstM == d_srcB)) || (D_icode == 4'd9 || E_icode == 4'd9 || M_icode == 4'd9))
    begin
        F_stall = 1;
    end
    if((E_icode == 4'd5 || E_icode == 4'd11) && (E_dstM == d_srcA || E_dstM == d_srcB))
    begin
        D_stall = 1;
    end
    if((E_icode == 4'd7 && e_Cnd == 0) || (!((E_icode == 4'd5 || E_icode == 4'd11) && (E_dstM == d_srcA || E_dstM == d_srcB)) && (D_icode == 4'd9 || E_icode == 4'd9 || M_icode == 4'd9)))
    begin
        D_bubble = 1;
    end
    if((E_icode == 4'd7 && e_Cnd == 0) || ((E_icode == 4'd5 || E_icode == 4'd11) && (E_dstM == d_srcA || E_dstM == d_srcB)))
    begin
        E_bubble = 1;
    end
    if(m_stat == 2'd1 || m_stat == 2'd2 || m_stat == 2'd3 || W_stat == 2'd1 || W_stat == 2'd2 || W_stat == 2'd3)
    begin
        M_bubble = 1;
    end
    if(W_stat == 2'd1 || W_stat == 2'd2 || W_stat == 2'd3)
    begin
        W_stall = 1;
    end
    if(E_icode == 4'd6 && !(m_stat == 2'd1 || m_stat == 2'd2 || m_stat == 2'd3) || !(W_stat == 2'd1 || W_stat == 2'd2 || W_stat == 2'd3))
    begin
        set_cc = 1;
    end
end
```

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

- **Data Dependencies -** results computed by one instruction are used as the data for a following instruction
- **Control Dependencies -** one instruction determines the location of the following instruction, such as when executing a jump,call or return

When such dependencies have the potential to cause an erroneous computation by the pipeline, they are called hazards.

DATA HAZARD

A data hazard can arise for an instruction when one of its operands is updated by any of the three preceding instructions. These hazards can occur because our pipelined processor reads the operands for an instruction from the register file in the decode stage but does not write the results for the instruction to the register file until three cycles later, after the instruction passes through the write - back stage.

Such hazards can be avoided by **STALLING**, where the processor holds back one or more instructions in the pipeline until the hazard condition no longer holds, and **FORWARDING**, where a result value is directly passed from one pipeline stage to an earlier stage via feedback paths. Stalling increases the latency, thus we prefer forwarding to deal with data hazards.

Load/Use Data Hazards - These cannot be handled purely by forwarding because memory reads occur late in the pipeline. One instruction reads a value from memory 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.

CONTROL HAZARD

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 occur only in our pipelined processor for ret and jump instructions. These are avoided by introducing a **BUBBLE** and **STALLING** some stages.

Dealing with Control Hazard in ret

Stall the fetch stage while ret passes through decode, execute and memory injecting three bubbles in the decode stage. PC selection logic will choose the return address once ret reaches the write back stage

```
00110000
11110011
00001010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110010
00000101
00000000
00000000
00000000
00000000
00000000
00000000
00000000
10000000
00101000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110001
00011001
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100000
00110010
10010000
```

```
WARNING: ./fetch.v:37: $readmemb(call_ret.txt): Not enough words in the file for the requested range [0:1023].
VCD info: dumpfile pipe.vcd opened for output.
                10 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 3, valP=                 10, valC=                 10, predPC=                 20, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= x, E_valA=                  x, E_valB=                  x, stat=x
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                20 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 2, valP=                 20, valC=                  5, predPC=                 40, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                30 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 8, ifun= 0, rA=15, rB= 2, valP=                 29, valC=                 40, predPC=                 42, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 3, valE=                 10, valA=                  x, stat=0, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                40 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 6, ifun= 0, rA= 3, rB= 2, valP=                 42, valC=                 40, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 8, E_valA=                 29, E_valB=                 32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 3, valE=                  5, valA=                  x, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                50 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------
icode= 9, ifun= 0, rA= 3, rB= 2, valP=                 43, valC=                 40, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------
icode= 6, E_valA=                 10, E_valB=                  5, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------
icode= 8, valE=                 24, valA=                 29, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=1, D_stall=0, D_bubble=1, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
```

```
                60 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 1, ifun= 0, rA=15, rB=15, valP=                  0, valC=                  0, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 9, E_valA=                24, E_valB=                24, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 6, valE=                 15, valA=                10, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 8, stat=0, valM=                  0
F_stall=1, D_stall=0, D_bubble=1, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                70 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 1, ifun= 0, rA=15, rB=15, valP=                  0, valC=                  0, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 1, E_valA=                32, E_valB=                32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 9, valE=                 32, valA=                24, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 6, stat=0, valM=                  0
F_stall=1, D_stall=0, D_bubble=1, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                80 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 1, ifun= 0, rA=15, rB=15, valP=                  0, valC=                  0, predPC=                 39, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 1, E_valA=                32, E_valB=                32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 1, valE=                 32, valA=                32, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 9, stat=0, valM=                 29
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                90 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 1, valP=                 39, valC=                 25, predPC=                 40, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 1, E_valA=                32, E_valB=                32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 1, valE=                 32, valA=                32, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 1, stat=0, valM=                 29
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
               100 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 0, ifun= 0, rA=15, rB= 1, valP=                 40, valC=                 25, predPC=                 42, stat=1
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 3, E_valA=                32, E_valB=                32, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 1, valE=                 32, valA=                32, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 1, stat=0, valM=                 29
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
               110 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 6, ifun= 0, rA= 3, rB= 2, valP=                 42, valC=                 25, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 0, E_valA=                32, E_valB=                32, stat=1
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 3, valE=                 25, valA=                32, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 1, stat=0, valM=                 29
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
```

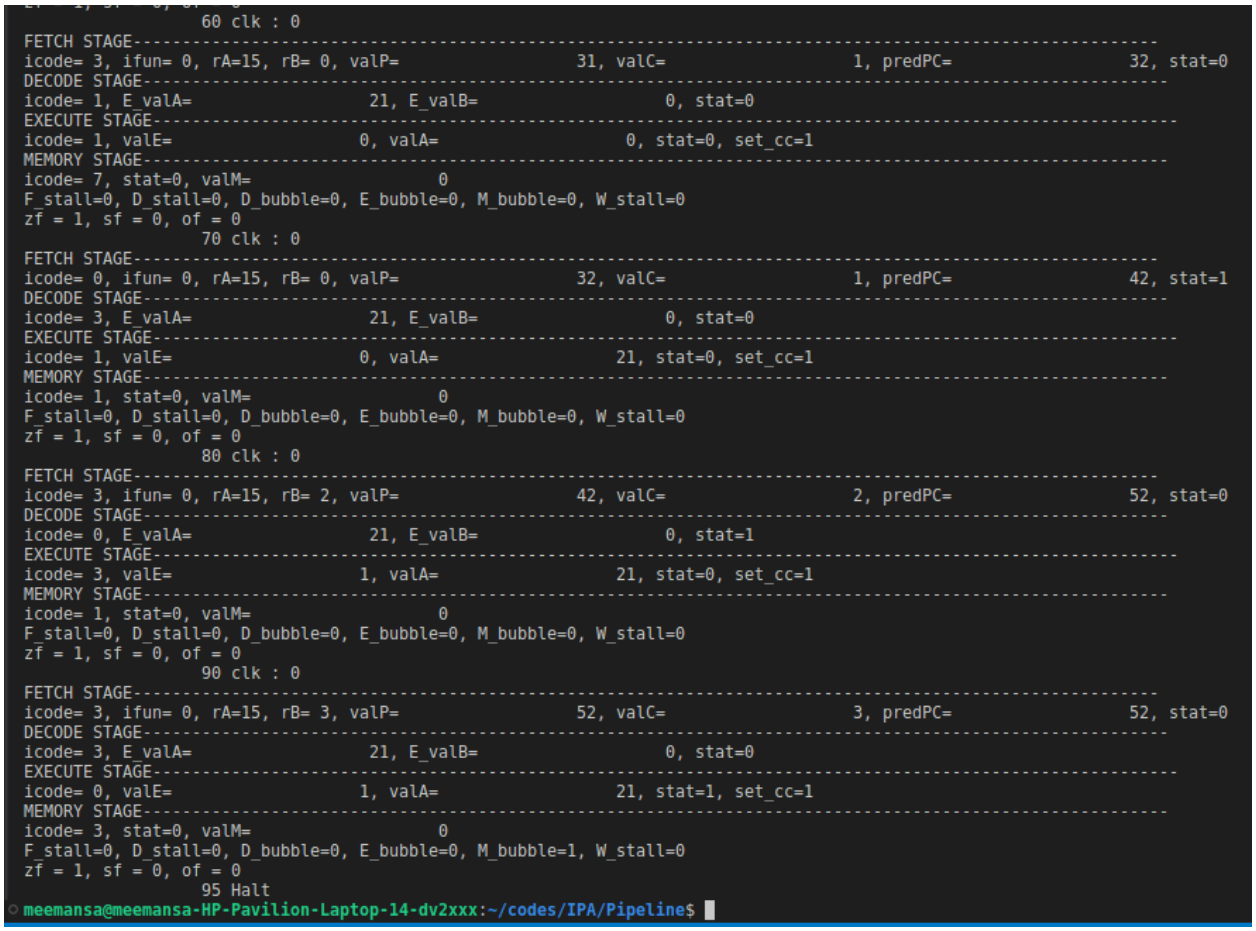
```
               120 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 9, ifun= 0, rA= 3, rB= 2, valP=                 43, valC=                 25, predPC=                 43, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 6, E_valA=                10, E_valB=                15, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 0, valE=                 25, valA=                32, stat=1, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                 29
F_stall=1, D_stall=0, D_bubble=1, E_bubble=0, M_bubble=1, W_stall=0
zf = 0, sf = 0, of = 0
               125 Halt
meemansa@meemansa-HP-Pavilion-Laptop-14-dv2xxx:~/codes/IPA/Pipeline$
```

Dealing with Control Hazard in jump

In case of a mispredicted jump, cancel the two misfetched instructions by injecting bubbles into the decode and execute stages while also fetching the instructions following the jump instruction.

```
00110000
11110000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100011
00000000
01110100
00010110
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110000
00000001
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110010
00000010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110011
00000011
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
VCD info: dumpfile pipe.vcd opened for output.
                10 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 0, valP=                10, valC=                 0, predPC=                12, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= x, E_valA=                  x, E_valB=                  x, stat=x
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                20 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 6, ifun= 3, rA= 0, rB= 0, valP=                12, valC=                 0, predPC=                22, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                30 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 7, ifun= 4, rA= 0, rB= 0, valP=                21, valC=                22, predPC=                22, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 6, E_valA=                  0, E_valB=                  0, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 3, valE=                  0, valA=                  x, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
                40 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode=15, ifun= 0, rA= 0, rB= 0, valP=                21, valC=                22, predPC=                22, stat=3
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 7, E_valA=                 21, E_valB=                  0, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 6, valE=                  0, valA=                  0, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=1, E_bubble=1, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
                50 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 1, ifun= 0, rA=15, rB=15, valP=                 0, valC=                 0, predPC=                31, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 1, E_valA=                  0, E_valB=                  0, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 7, valE=                  0, valA=                 21, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 6, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
```


60 clk : 0
FETCH STAGE
icode= 3, ifun= 0, rA=15, rB= 0, valP= 31, valC= 1, predPC= 32, stat=0
DECODE STAGE
icode= 1, E_valA= 21, E_valB= 0, stat=0
EXECUTE STAGE
icode= 1, valE= 0, valA= 0, stat=0, set_cc=1
MEMORY STAGE
icode= 7, stat=0, valM= 0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
70 clk : 0
FETCH STAGE
icode= 0, ifun= 0, rA=15, rB= 0, valP= 32, valC= 1, predPC= 42, stat=1
DECODE STAGE
icode= 3, E_valA= 21, E_valB= 0, stat=0
EXECUTE STAGE
icode= 1, valE= 0, valA= 21, stat=0, set_cc=1
MEMORY STAGE
icode= 1, stat=0, valM= 0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
80 clk : 0
FETCH STAGE
icode= 3, ifun= 0, rA=15, rB= 2, valP= 42, valC= 2, predPC= 52, stat=0
DECODE STAGE
icode= 0, E_valA= 21, E_valB= 0, stat=1
EXECUTE STAGE
icode= 3, valE= 1, valA= 21, stat=0, set_cc=1
MEMORY STAGE
icode= 1, stat=0, valM= 0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 1, sf = 0, of = 0
90 clk : 0
FETCH STAGE
icode= 3, ifun= 0, rA=15, rB= 3, valP= 52, valC= 3, predPC= 52, stat=0
DECODE STAGE
icode= 3, E_valA= 21, E_valB= 0, stat=0
EXECUTE STAGE
icode= 0, valE= 1, valA= 21, stat=1, set_cc=1
MEMORY STAGE
icode= 3, stat=0, valM= 0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=1, W_stall=0
zf = 1, sf = 0, of = 0
95 Halt
meemansa@meemansa-HP-Pavilion-Laptop-14-dv2xxx:~/codes/IPA/Pipeline$

Dealing with Load/Use Data Hazard

Stall the instruction which requires operand from memory in the decode stage while injecting a bubble in the execute stage. The value obtained from the write back stage is then forwarded to the memory stage.

```
00110000
11110010
10000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110001
00000011
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01000000
00010010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00110000
11110011
00001010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01010000
00000010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
01100000
00110000
00000000
```

```
WARNING: ./fetch.v:38: $readmemb(mrmovq.txt): Not enough words in the file for the requested range [0:1023].
VCD info: dumpfile pipe.vcd opened for output.
                10 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 2, valP=                10, valC=               128, predPC=                20, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= x, E_valA=                  x, E_valB=                  x, stat=x
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                20 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 1, valP=                20, valC=                3, predPC=                30, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= x, valE=                  0, valA=                  x, stat=x, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                30 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 4, ifun= 0, rA= 1, rB= 2, valP=                30, valC=                0, predPC=                40, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  x, E_valB=                  x, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 3, valE=                128, valA=                  x, stat=0, set_cc=0
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= x, stat=x, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                40 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 3, ifun= 0, rA=15, rB= 3, valP=                40, valC=               10, predPC=                50, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 4, E_valA=                  3, E_valB=                128, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 3, valE=                  3, valA=                  x, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                50 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 5, ifun= 0, rA= 0, rB= 2, valP=                50, valC=                0, predPC=                52, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 3, E_valA=                  3, E_valB=                128, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 4, valE=                128, valA=                  3, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                  0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
```

```
                60 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 6, ifun= 0, rA= 3, rB= 0, valP=                52, valC=                 0, predPC=                53, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 5, E_valA=                  3, E_valB=                128, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 3, valE=                10, valA=                 3, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 4, stat=0, valM=                   0
F_stall=1, D_stall=1, D_bubble=0, E_bubble=1, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                70 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 6, ifun= 0, rA= 3, rB= 0, valP=                52, valC=                 0, predPC=                53, stat=0
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 1, E_valA=                  0, E_valB=                  0, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 5, valE=               128, valA=                 3, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 3, stat=0, valM=                   0
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                80 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= 0, ifun= 0, rA= 3, rB= 0, valP=                53, valC=                 0, predPC=                53, stat=1
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 6, E_valA=                 10, E_valB=                  3, stat=0
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 1, valE=                128, valA=                 0, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 5, stat=0, valM=                   3
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
                90 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= x, ifun= x, rA= 3, rB= 0, valP=                53, valC=                 0, predPC=                53, stat=3
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= 0, E_valA=                 10, E_valB=                128, stat=1
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 6, valE=                13, valA=                10, stat=0, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 1, stat=0, valM=                   3
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=0, W_stall=0
zf = 0, sf = 0, of = 0
               100 clk : 0
FETCH STAGE-----------------------------------------------------------------------------------------------------------
icode= x, ifun= x, rA= 3, rB= 0, valP=                53, valC=                 0, predPC=                53, stat=3
DECODE STAGE----------------------------------------------------------------------------------------------------------
icode= x, E_valA=                 10, E_valB=                128, stat=3
EXECUTE STAGE---------------------------------------------------------------------------------------------------------
icode= 0, valE=                13, valA=                10, stat=1, set_cc=1
MEMORY STAGE----------------------------------------------------------------------------------------------------------
icode= 6, stat=0, valM=                   3
F_stall=0, D_stall=0, D_bubble=0, E_bubble=0, M_bubble=1, W_stall=0
zf = 0, sf = 0, of = 0
               105 Halt
meemansa@meemansa-HP-Pavilion-Laptop-14-dv2xxx:~/codes/IPA/Pipeline$
```

Challenges faced

- It was hard to get started initially for converting the design theory learnt in class to the hardware circuit implementation in the code
- In the seq state, we had to ensure that each stage is timed correctly with the clock.
- It was challenging to deal with all the control hazards and special cases in the pipelined processor
- Debugging errors in the code for each functionality was very tedious
- In pipelining, it was difficult to keep track of all inputs and outputs since each stage had multiple inputs and outputs coming at the same time