Project Report

By Rohan Gupta - 2020112022, Anjali Singh - 2020102004

Overview

The goal of the project is to build a processor architecture design based on Y86-64 ISA, by making various design modules using verilog.

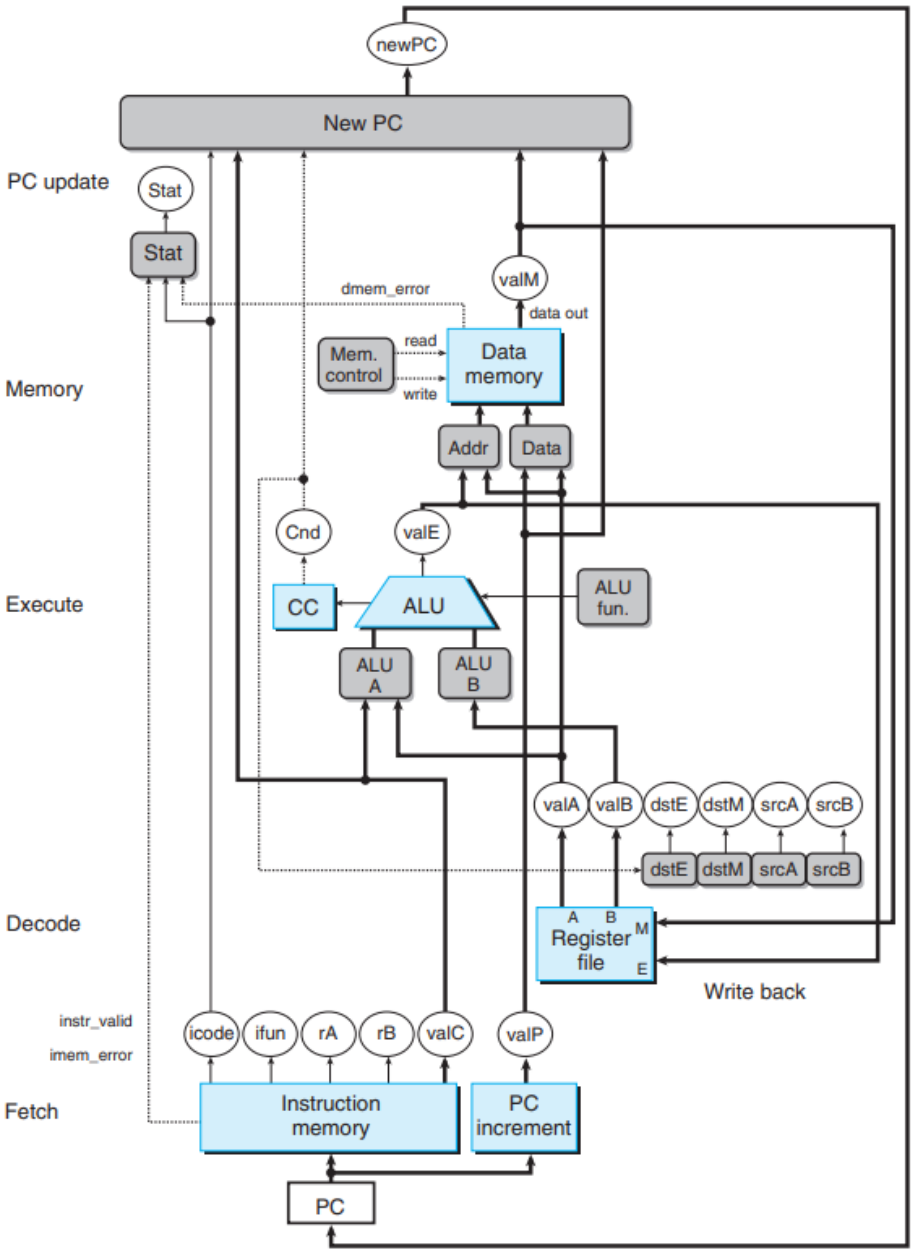
Modules

The two modules we are going to build here are sequential and pipelining.

Sequential

The SEQ hardware structure includes the six basic stages: fetch, decode, execute, memory, write back and PC update. The notations icode, ifun indicate components of the instruction bytes; rA, rB indicate components of register specifier byte. The following are the various stages described with their verilog modules.

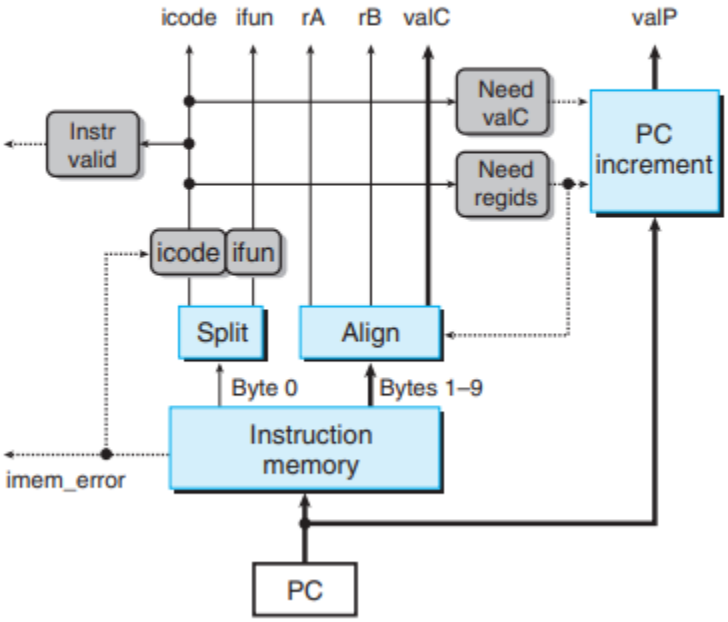
Hardware Structure



Fetch

- Here, the program counter register is used as an address.
- The instruction memory reads the bytes of an instruction.
- The PC incrementer is used to compute valP, which is again fetched and used in the fetch module, but we haven't used this module.
- We have used several modules such as: spilt, align, PC increment, instruction valids, for their different functionalities.
- An instruction memory is also created to store data, from which they can be read and written from one part of the memory.

Structure



Verilog Modules

- **Fetch**

```
module split(Byte0, icode, ifun);

input [7:0] Byte0;
output [3:0] icode;
output [3:0] ifun;

    assign icode = Byte0[7:4];
    assign ifun = Byte0[3:0];

endmodule

module Need_VALC(icode, need_valC);
input [3:0] icode;
output reg need_valC;

always @(icode)
    begin
```

```
    case (icode)
        4'h3, 4'h4, 4'h5, 4'h7, 4'h8:
            begin
                assign need_valC = 1'b1;
            end
        4'h1, 4'h2, 4'h6, 4'h6, 4'h9, 4'hA, 4'hB:
            begin
                assign need_valC = 1'b0;
            end

        default: need_valC = 1'b0;
    endcase
    end
endmodule

module Need_REGIDS(icode, need_regids);

input [3:0] icode;
output reg need_regids;

    always @(icode)
        begin
        case (icode)
            4'h2, 4'h3, 4'h4, 4'h5, 4'h6, 4'hA, 4'hB:
                begin
                    assign need_regids = 1'b1;
                end
            4'h1, 4'h7, 4'h8, 4'h9:
                begin
                    assign need_regids = 1'b0;
                end

            default: need_regids = 1'b0;
        endcase
        end

endmodule


module align(Byte19,need_regids,rA,rB,valC);
```

```
input [71:0] Byte19;
input need_regids;
output [3:0] rA;
output [3:0] rB;
output [63:0] valC;

    assign rA = Byte19[7:4] ;
    assign rB = Byte19[3:0];
    assign valC = need_regids ? Byte19[71:8]:Byte19[63:0];


endmodule

module PC_INCREMENT(pc, icode, need_regids, need_valC, valP);

input[63:0] pc;
input[3:0] icode;
input need_regids;
input need_valC;

reg halt;
output [63:0] valP;

always @(icode) begin

      if(icode == 4'b000) begin
        halt <= 1'b1;
    end
    else
        halt <= 1'b0;

end


    assign valP = halt ? pc:(need_valC ? (need_regids ?
pc+10:pc+9):(need_regids ? pc+2:pc+1));


endmodule
```

```
module INSTR_VALID(icode, instr_valid);
input [3:0] icode;
output reg instr_valid;

always @(icode)
    begin
    case (icode)
        4'h0, 4'h1, 4'h2, 4'h3, 4'h4, 4'h5, 4'h6, 4'h7, 4'h8, 4'h9,
4'hA, 4'hB:
            begin
                assign instr_valid = 1'b1;
            end

        default: instr_valid = 1'b0;
    endcase
    end


endmodule
```

- **Instruction Memory**

```
module instruction_memory(clk, pc, imem_error, Byte0, Byte19);

input clk;
input[63:0] pc;
output reg imem_error;
output reg[7:0] Byte0;
output reg[71:0] Byte19;

reg [0:7] instr_mem[0:2047];

initial begin
//nopo
instr_mem[0] <= 8'b00010000; //icode ifun

//irmovq %512 %r12
instr_mem[1] <= 8'b00110000; //icode ifun
instr_mem[2] <= 8'b11110100; //reg 15 8
instr_mem[3] <= 8'b00000000; // 1 0
```

```
instr_mem[4] <= 8'b00000010;
instr_mem[5] <= 8'b00000000;
instr_mem[6] <= 8'b00000000;
instr_mem[7] <= 8'b00000000;
instr_mem[8] <= 8'b00000000;
instr_mem[9] <= 8'b00000000;
instr_mem[10] <= 8'b00000000; // 0 0

//irmovq %16 %rdi
instr_mem[11] <= 8'b00110000; //icode ifun 3 0
instr_mem[12] <= 8'b11110111; //reg F 7
instr_mem[13] <= 8'b00010000; // 10 : 0 A
instr_mem[14] <= 8'b00000000;
instr_mem[15] <= 8'b00000000;
instr_mem[16] <= 8'b00000000;
instr_mem[17] <= 8'b00000000;
instr_mem[18] <= 8'b00000000;
instr_mem[19] <= 8'b00000000;
instr_mem[20] <= 8'b00000000;

//irmovq %10 %r12
instr_mem[21] <= 8'b00110000; //icode ifun
instr_mem[22] <= 8'b11111100; //reg 15 8
instr_mem[23] <= 8'b00001010; // 1 0
instr_mem[24] <= 8'b00000000;
instr_mem[25] <= 8'b00000000;
instr_mem[26] <= 8'b00000000;
instr_mem[27] <= 8'b00000000;
instr_mem[28] <= 8'b00000000;
instr_mem[29] <= 8'b00000000;
instr_mem[30] <= 8'b00000000; // 0 0

//rmmovq %r12 %(rdi)
instr_mem[31] <= 8'b01000000; //icode ifun 3 0
instr_mem[32] <= 8'b11000111; //reg
instr_mem[33] <= 8'b00000000; // 10 : 0 A
instr_mem[34] <= 8'b00000000;
instr_mem[35] <= 8'b00000000;
instr_mem[36] <= 8'b00000000;
instr_mem[37] <= 8'b00000000;
```

```
instr_mem[38] <= 8'b00000000;
instr_mem[39] <= 8'b00000000;
instr_mem[40] <= 8'b00000000;


//mrmovq %r13 %(rdi)
instr_mem[41] <= 8'b01010000; //icode ifun 3 0
instr_mem[42] <= 8'b11010111; //reg
instr_mem[43] <= 8'b00000000; // 10 : 0 A
instr_mem[44] <= 8'b00000000;
instr_mem[45] <= 8'b00000000;
instr_mem[46] <= 8'b00000000;
instr_mem[47] <= 8'b00000000;
instr_mem[48] <= 8'b00000000;
instr_mem[49] <= 8'b00000000;
instr_mem[50] <= 8'b00000000;


//call
instr_mem[51] <= 8'b10000000; //icode ifun: 8 0
instr_mem[52] <= 8'b01110000;
instr_mem[53] <= 8'b00000000;
instr_mem[54] <= 8'b00000000;
instr_mem[55] <= 8'b00000000;
instr_mem[56] <= 8'b00000000;
instr_mem[57] <= 8'b00000000;
instr_mem[58] <= 8'b00000000;
instr_mem[59] <= 8'b00000000;

//halt
instr_mem[60] <= 8'b00000000; // 00

// irmovq $8 %r8
instr_mem[112] <= 8'b00110000; //icode ifun
instr_mem[113] <= 8'b11111000; //reg
instr_mem[114] <= 8'b00001000;
instr_mem[115] <= 8'b00000000;
instr_mem[116] <= 8'b00000000;
instr_mem[117] <= 8'b00000000;
instr_mem[118] <= 8'b00000000;
```

```
instr_mem[119] <= 8'b00000000;
instr_mem[120] <= 8'b00000000;
instr_mem[121] <= 8'b00000000;

// irmovq %1 %r9
instr_mem[122] <= 8'b00110000; //icode ifun
instr_mem[123] <= 8'b11111001; //reg
instr_mem[124] <= 8'b00000001;
instr_mem[125] <= 8'b00000000;
instr_mem[126] <= 8'b00000000;
instr_mem[127] <= 8'b00000000;
instr_mem[128] <= 8'b00000000;
instr_mem[129] <= 8'b00000000;
instr_mem[130] <= 8'b00000000;
instr_mem[131] <= 8'b00000000;

// xorq %rax % rax  %rax =0
instr_mem[132] <= 8'b01100011; //icode ifun
instr_mem[133] <= 8'b00000000; //reg

// andq %rsi %rsi -- set CC %rsi =6
instr_mem[134] <= 8'b01100010; //icode ifun
instr_mem[135] <= 8'b01100110; //reg

// jmp test- some memory address
instr_mem[136] <= 8'b01110000; //icode ifun
instr_mem[137] <= 8'b10010001; //reg
instr_mem[138] <= 8'b00000000;
instr_mem[139] <= 8'b00000000;
instr_mem[140] <= 8'b00000000;
instr_mem[141] <= 8'b00000000;
instr_mem[142] <= 8'b00000000;
instr_mem[143] <= 8'b00000000;
instr_mem[144] <= 8'b00000000;


// loop:
instr_mem[145] <= 8'b01100001; //icode ifun
instr_mem[146] <= 8'b10011000; //reg
```

```
/* // loop:
instr_mem[145] <= 8'b01100001; //icode ifun
instr_mem[146] <= 8'b10011000; //reg */

//test:
//jne loop;
instr_mem[147] <= 8'b01110100; //icode ifun
instr_mem[148] <= 8'b10010001; //reg
instr_mem[149] <= 8'b00000000; //
instr_mem[150] <= 8'b00000000; //
instr_mem[151] <= 8'b00000000; //
instr_mem[152] <= 8'b00000000; //
instr_mem[153] <= 8'b00000000; //
instr_mem[154] <= 8'b00000000; //
instr_mem[155] <= 8'b00000000; //

//ret
instr_mem[156] <= 8'b10010000; // 9 0

end


always @(*) begin

    if(pc > 64'd2047) begin
        imem_error <= 1'b1;
        Byte0 <= 8'b00000000;

    end
    else begin
        imem_error = 1'b0;
        Byte0 = instr_mem[pc];
        Byte19[7:0] <= instr_mem[pc+1];
        Byte19[15:8] <= instr_mem[pc+2];
        Byte19[23:16] <= instr_mem[pc+3];
        Byte19[31:24] <= instr_mem[pc+4];
        Byte19[39:32] <= instr_mem[pc+5];
        Byte19[47:40] <= instr_mem[pc+6];
        Byte19[55:48] <= instr_mem[pc+7];
        Byte19[63:56] <= instr_mem[pc+8];
```

```
        Byte19[71:64] <= instr_mem[pc+9];
    end
end

endmodule
```

Testbench

```
`timescale 1ns / 1ps
module fetch_tb;

reg clk;
reg [63:0] pc;
wire [7:0] Byte0;
wire [71:0] Byte19;
wire [3:0] icode;
wire [3:0] ifun;

wire need_regids;
wire need_valC;
wire instr_valid;
wire imem_error;

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

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

reg [7:0] instr_mem [2047:0];

split sp(.Byte0(Byte0), .icode(icode), .ifun(ifun));
align al(.Byte19(Byte19), .need_regids(need_regids), .rA(rA), .rB(rB),
.valC(valC));
PC_INCREMENT PC_i(.pc(pc), .need_regids(need_regids),
.need_valC(need_valC), .valP(valP));
INSTR_VALID i_valid(.icode(icode), .instr_valid(instr_valid));
Need_REGIDS nreg(.icode(icode), .need_regids(need_regids));
Need_VALC n_valC(.icode(icode), .need_valC(need_valC));
```

```
instruction_memory
InstMem(.clk(clk),.pc(pc),.imem_error(imem_error),.Byte0(Byte0),.Byte19(By
te19));

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

    clk = 0;
    pc = 64'd1;

    #10 clk = ~clk; pc=64'd0;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=64'd112;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
    #10 clk = ~clk; pc=valP;
    #10 clk = ~clk;
  end

initial begin
```

```
    $monitor("clk=%d, pc=%d, icode=%d, ifun=%d, rA=%b, rB=%b, valC=%d,
valP=%d, imeme_err = %b\n", clk, pc, icode, ifun, rA, rB, valC,
valP,imem_error);

end

endmodule
```

Output

```
clk=0, pc=                1, icode= 3, ifun= 0, rA=1111, rB=0100, valC=              512,
valP=               11, imeme_err = 0

clk=1, pc=                0, icode= 1, ifun= 0, rA=0011, rB=0000, valC=         33616944,
valP=                1, imeme_err = 0

clk=0, pc=                0, icode= 1, ifun= 0, rA=0011, rB=0000, valC=         33616944,
valP=                1, imeme_err = 0

clk=1, pc=                1, icode= 3, ifun= 0, rA=1111, rB=0100, valC=              512,
valP=               11, imeme_err = 0

clk=0, pc=                1, icode= 3, ifun= 0, rA=1111, rB=0100, valC=              512,
valP=               11, imeme_err = 0

clk=1, pc=               11, icode= 3, ifun= 0, rA=1111, rB=0111, valC=               16,
valP=               21, imeme_err = 0

clk=0, pc=               11, icode= 3, ifun= 0, rA=1111, rB=0111, valC=               16,
valP=               21, imeme_err = 0

clk=1, pc=               21, icode= 3, ifun= 0, rA=1111, rB=1100, valC=               10,
valP=               31, imeme_err = 0

clk=0, pc=               21, icode= 3, ifun= 0, rA=1111, rB=1100, valC=               10,
valP=               31, imeme_err = 0
clk=1, pc=              112, icode= 3, ifun= 0, rA=1111, rB=1000, valC=                8,
valP=              122, imeme_err = 0

clk=0, pc=              112, icode= 3, ifun= 0, rA=1111, rB=1000, valC=                8,
valP=              122, imeme_err = 0
```

clk=1, pc= 122, icode= 3, ifun= 0, rA=1111, rB=1001, valC= 1, valP= 132, imeme_err = 0

clk=0, pc= 122, icode= 3, ifun= 0, rA=1111, rB=1001, valC= 1, valP= 132, imeme_err = 0

clk=1, pc= 132, icode= 6, ifun= 3, rA=0000, rB=0000, valC= 2440062562, valP= 134, imeme_err = 0

clk=0, pc= 132, icode= 6, ifun= 3, rA=0000, rB=0000, valC= 2440062562, valP= 134, imeme_err = 0

clk=1, pc= 134, icode= 6, ifun= 2, rA=0110, rB=0110, valC= 37232, valP= 136, imeme_err = 0

clk=0, pc= 134, icode= 6, ifun= 2, rA=0110, rB=0110, valC= 37232, valP= 136, imeme_err = 0

clk=1, pc= 136, icode= 7, ifun= 0, rA=1001, rB=0001, valC= 145, valP= 145, imeme_err = 0

clk=0, pc= 136, icode= 7, ifun= 0, rA=1001, rB=0001, valC= 145, valP= 145, imeme_err = 0

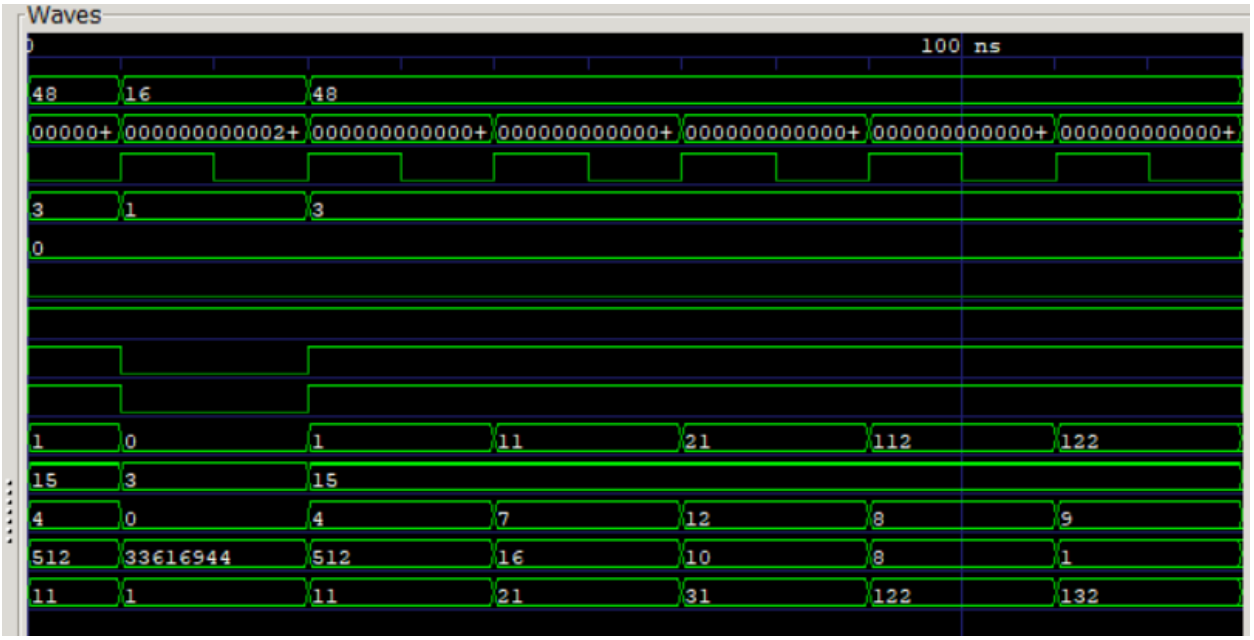
clk=1, pc= 145, icode= 6, ifun= 1, rA=1001, rB=1000, valC= 37236, valP= 147, imeme_err = 0

clk=0, pc= 145, icode= 6, ifun= 1, rA=1001, rB=1000, valC= 37236, valP= 147, imeme_err = 0

clk=1, pc= 147, icode= 7, ifun= 4, rA=1001, rB=0001, valC= 145, valP= 156, imeme_err = 0

clk=0, pc= 147, icode= 7, ifun= 4, rA=1001, rB=0001, valC= 145, valP= 156, imeme_err = 0

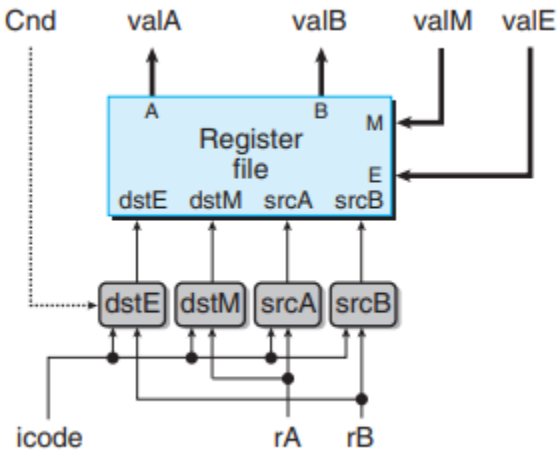
GTKwave Output



Decode/Write Back

- It has two read ports, A and B, using which valA and valB are read simultaneously.
- It also has two write ports, E and M.

Structure



Verilog Modules

```
`timescale 1ns/1ps
module registerfile(clk, dstE, dstM, srcA, srcB, valE, valM, valA, valB);
```

```
input [3:0] dstE;
input [3:0] dstM;
input [3:0] srcA;
input [3:0] srcB;
input clk;
input [63:0] valE;
input [63:0] valM;

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


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

parameter rax = 4'h0 ;
parameter rcx = 4'h1 ;
parameter rdx = 4'h2 ;
parameter rbx = 4'h3 ;
parameter rsp = 4'h4 ;
parameter rbp = 4'h5 ;
parameter rsi = 4'h6 ;
parameter rdi = 4'h7 ;
parameter r8 = 4'h8 ;
parameter r9 = 4'h9 ;
parameter r10 = 4'hA ;
parameter r11 = 4'hB;
parameter r12 = 4'hC ;
parameter r13 = 4'hD;
parameter r14 = 4'hE ;
parameter rnone = 4'hF ;

initial
begin
register_file[4'h0] <= 64'h0;
register_file[4'h1] <= 64'h0;
register_file[4'h2] <= 64'h0;
register_file[4'h3] <= 64'h0;
register_file[4'h4] <= 64'h0;
register_file[4'h5] <= 64'h0;
```

```
register_file[4'h6] <= 64'h0;
register_file[4'h7] <= 64'h0;
register_file[4'h8] <= 64'h0;
register_file[4'h9] <= 64'h0;
register_file[4'hA] <= 64'h0;
register_file[4'hB] <= 64'h0;
register_file[4'hC] <= 64'h0;
register_file[4'hD] <= 64'h0;
register_file[4'hE] <= 64'h0;
end
always @(*) begin

    if (srcA != rnone) begin
        valA <= register_file[srcA];

    end

    if (srcB != rnone) begin
        valB <= register_file[srcB];
    end
end

always @(posedge(clk)) begin

    if(dstE != rnone) begin
        register_file[dstE] <= valE;
    end

    if(dstM != rnone) begin
        register_file[dstM] <= valM;
    end

end
endmodule

module srcA_logic(icode,rA,srcA);

input[3:0]icode;
input[3:0]rA;
```

```
output reg[3:0]srcA;

parameter rsp = 4'h4 ;
parameter rnone = 4'hF ;


always @(icode,rA) begin

    case (icode)
    4'h2, 4'h3,4'h4,4'h5,4'h8,4'h6, 4'hA:
    begin
        srcA<= rA;
    end
    4'h9,4'hB:
    begin
        srcA <= rsp;
    end
        default: srcA <= rnone;
    endcase

end

endmodule


module srcB_logic(icode,rB,srcB);


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

output reg[3:0]srcB;

parameter rsp = 4'h4 ;
parameter rnone = 4'hF ;


always @(icode,rB) begin

    case (icode)
```

```
    4'h6, 4'h4, 4'h5:
    begin
        srcB<= rB;
    end
    4'hA, 4'hB, 4'h8, 4'h9:
    begin
        srcB <= rsp;
    end
        default: srcB <= rnone;
    endcase

end

endmodule


module dstE_logic(icode,ifun,rB,cnd,dstE);

input[3:0]icode;
input [3:0] ifun;
input[3:0]rB;
input cnd;
output reg[3:0]dstE;

parameter rsp = 4'h4 ;
parameter rnone = 4'hF ;


always @(icode,ifun,rB,cnd) begin

    case (icode)

    4'h2: begin
        if(cnd==1'b1)
            dstE <= rB;
        else
            dstE <= rnone;
    end

    4'h3, 4'h6:
```

```
        dstE <= rB;
    4'hA, 4'hB, 4'h8, 4'h9:
        dstE <= rsp;
        default: dstE <= rnone;
    endcase

end

endmodule


module dstM_logic(icode,rA,dstM);


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

output reg[3:0]dstM;


parameter rsp = 4'h4 ;
parameter rnone = 4'hF ;

always @(icode,rA) begin
     case (icode)
     4'h5, 4'hB:
        dstM <= rA;
         default: dstM <= rnone;
     endcase

end
endmodule
```

Testbench

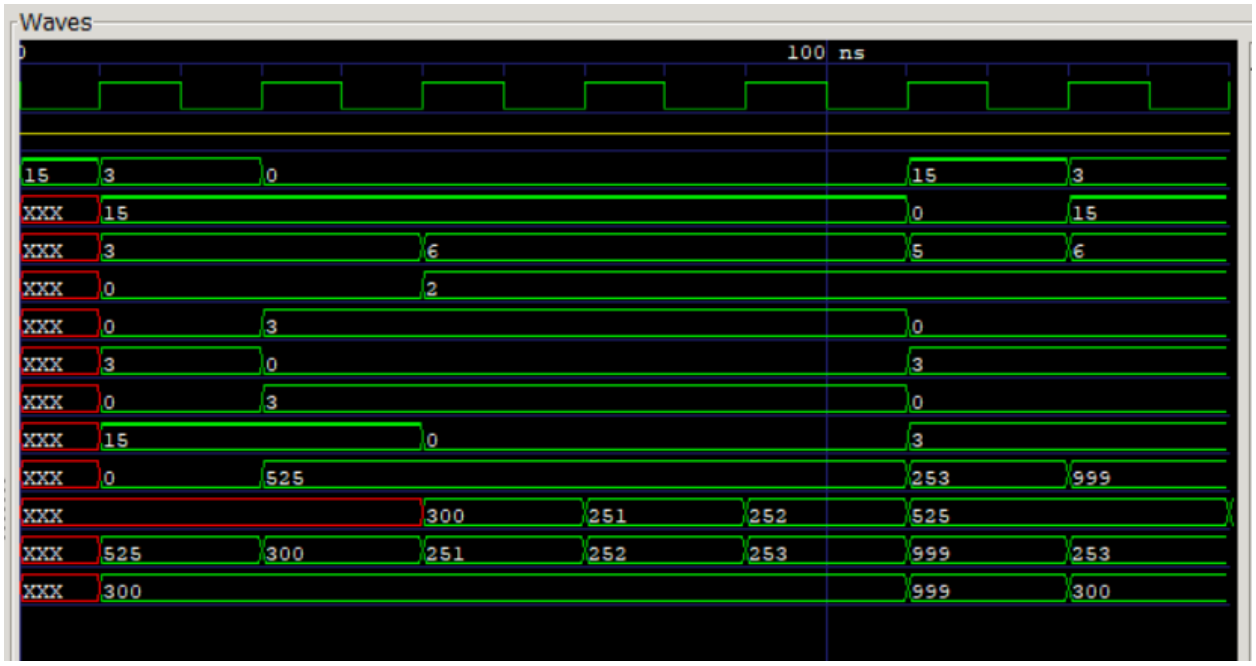
```
`timescale 1ns / 1ps
module decode_test;
    integer k;

    reg [63:0] valE;
    reg [3:0] rA;
```

```
    reg [3:0] rB;
    reg [3:0] icode;
    wire [3:0] dstE;
    wire [3:0] dstM;
    wire [3:0] srcA;
    wire [3:0] srcB;
    wire [63:0] valA;
    wire [63:0] valB;
    reg [63:0] valM;
    wire cnd;
    reg [3:0] ifun;
    reg clk;

    registerfile reg_f(.clk(clk),.dstE(dstE), .dstM(dstM), .srcA(srcA) ,
.srcB(srcB) , .valA(valA) , .valB(valB) , .valM(valM), .valE(valE));
    srcA_logic sA_1(.icode(icode),.rA(rA),.srcA(srcA));
    srcB_logic sB_1(.icode(icode),.rB(rB),.srcB(srcB));
    dstE_logic
dE_1(.icode(icode),.rB(rB),.cnd(cnd),.ifun(ifun),.dstE(dstE));
    dstM_logic dM_1(.icode(icode),.rA(rA),.dstM(dstM));

    initial begin
    clk <= 1'b0;
        $dumpfile("decode_test.vcd");
        $dumpvars(0,decode_test);
        #10;
        // irmovq
        icode <= 4'h3;
        ifun  <= 4'h0;
        rA <= 4'h0;
        rB <= 4'h3;
        valE <= 64'd525;
        valM <= 64'd300;
        #20;

        // moved 64'd525 to register rbx;

        icode <= 4'h3;
        ifun  <= 4'h0;
        rA <= 4'h3;
```

```
        rB <= 4'h0;
        valE <= 64'd300;
        valM <= 64'd300;
        #20;

        // moved 64'd300 to register rax;
        icode <= 4'h6;
        ifun  <= 4'h2;
        rA <= 4'h3;
        rB <= 4'h0;
        valE <= 64'd251;
        valM <= 64'd300;
        #20;

        icode <= 4'h6;
        ifun  <= 4'h2;
        rA <= 4'h3;
        rB <= 4'h0;
        valE <= 64'd252;
        valM <= 64'd300;
        #20;
        icode <= 4'h6;
        ifun  <= 4'h2;
        rA <= 4'h3;
        rB <= 4'h0;
        valE <= 64'd253;
        valM <= 64'd300;
        #20;

        //check memory to reg
        icode <= 4'h5;
        ifun  <= 4'h2;
        rA <= 4'h0;
        rB <= 4'h3;
        valE <= 64'd999;
        valM <= 64'd999;

        #20;
        icode <= 4'h6;
        ifun  <= 4'h2;
```

```
        rA <= 4'h0;
        rB <= 4'h3;
        valE <= 64'd253;
        valM <= 64'd300;

        #20 $finish;

    end
    always #10 clk = ~clk;

endmodule
```

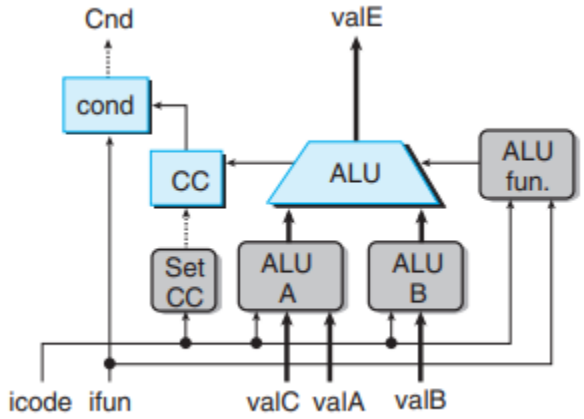
GTKWave Output



Execute

- It includes the ALU operations, which we made previously.
- The arithmetic and logic operations used here are AND, SUB, ADD and XOR.
- We have created all the ALU functions in separate files, and then compiled them into one as ‘Execute’.
- Several modules such as ALU block, which contains the ALU functions with both the ports A and B as inputs, ALU fun, with ‘ifun’ as one of the inputs are made and then ALU for A and B are defined.

Structure



Verilog Modules

```
`timescale 1ns / 1ps
module alu_block(aluA, aluB, alufun, valE, cf);

input[63:0] aluA;
input[63:0] aluB;
input[1:0] alufun;

output reg[63:0] valE;
output reg[2:0] cf;

wire signed [63:0] out1;
wire [2:0] cf_add;
wire signed [63:0] out2;
wire [2:0] cf_sub;
wire signed [63:0] out3;
wire [2:0] cf_and;
wire signed [63:0] out4;
wire [2:0] cf_xor;

    and_64bit g1(aluA, aluB, out3, cf_and);
    xor_64bit g2(aluA, aluB, out4, cf_xor);
    add_64bit g3(aluA, aluB, out1, cf_add);
    sub_64bit g4(aluB,aluA, out2, cf_sub);
```

```
    always@(*)
    begin
        case(alufun)
            2'b00:begin
                valE <= out1;
                cf <= cf_add;
            end
            2'b01:begin
                valE <= out2;
                cf <= cf_sub;

            end
            2'b10:begin
                valE <= out3;
                cf <= cf_and;
            end
            2'b11:begin
                valE <= out4;
                cf <= cf_xor;
            end

        endcase
    end

endmodule

module ALU_A(icode, valC, valA, aluA);

input[3:0] icode;
input[63:0] valC;
input[63:0] valA;

output reg[63:0] aluA;

always @(icode, valC, valA) begin

    case (icode)
    4'h2,4'h6:
        aluA <= valA;
```

```
    4'h5,4'h4,4'h3:
        aluA <= valC;
    4'h8,4'hA:
        aluA <= -64'd8;
    4'h9,4'hB:
        aluA <= 64'd8;
    endcase
end

endmodule


module ALU_B(icode, valB, aluB);

input[3:0] icode;
input[63:0] valB;

output reg[63:0] aluB;

always @(icode, valB) begin

    case (icode)
    4'h4,4'h5,4'h6, 4'h8,4'h9, 4'hA, 4'hB:
        aluB <= valB;
    4'h3,4'h4:
        aluB <= 64'b0;
    endcase
end
endmodule
module ALU_fun(icode, ifun, alufun);

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

output reg[1:0] alufun;

always @(icode) begin

    case (icode)
    4'h6: begin
```

```
        alufun[1] = ifun[1];
        alufun[0] = ifun[0];
    end
        default:alufun <= 2'b00;
    endcase

end
endmodule


module set_CC(icode, cf, outf);

input [3:0] icode;
input [2:0]cf;
output reg[2:0] outf;

always @ (icode,cf)
begin
if( icode == 4'h6)
    outf <= cf;
end

endmodule


module CND(ifun, outf, cnd);

    input[3:0] ifun;
    input[2:0] outf;
    output reg cnd;

    always @(ifun,outf)
    begin
        case(ifun)
        4'h0: cnd <= 1'b1;

        4'h1:
        begin
            if((outf[1]^outf[2]) || outf[0])
            cnd <= 1'b1;
```

```
            else
            cnd <= 1'b0;
        end

        4'h2:
        begin
            if(outf[1]^outf[2])
            cnd <= 1'b1;
            else
            cnd <= 1'b0;
        end

        4'h3:
        begin
            if(outf[0])
            cnd <= 1'b1;
            else
            cnd <= 1'b0;
        end

        4'h4:
        begin
            if(~outf[0])
            cnd <= 1'b1;
            else
            cnd <= 1'b0;
        end

        4'h5:
        begin
            if(~(outf[1]^outf[2]))
            cnd <= 1'b1;
            else
            cnd <= 1'b0;
        end

        4'h6:
        begin
            if(~(outf[1]^outf[2]) & ~outf[0])
            cnd <= 1'b1;
```

```
            else
            cnd <= 1'b0;
        end
        endcase

    end

endmodule
```

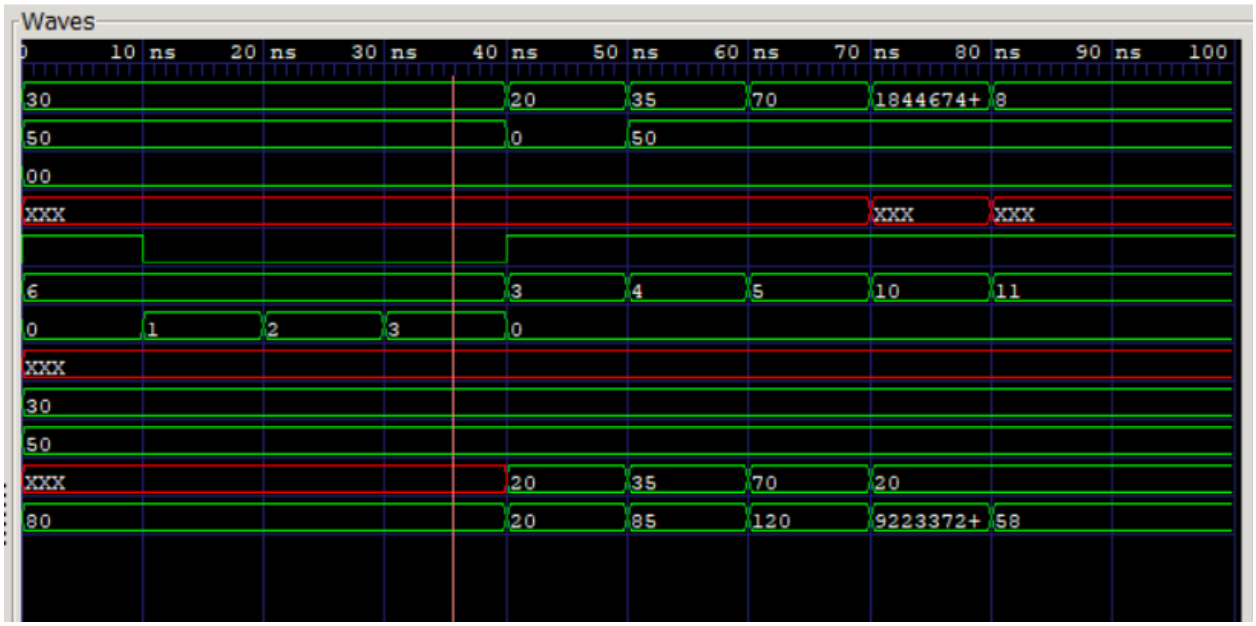
Testbench

```
`timescale 1ns / 1ps
`include "Add/add_1bit.v"
`include "Add/add_64bit.v"
`include "And/and_1bit.v"
`include "And/and_64bit.v"
`include "Or/or_1bit.v"
`include "Sub/not_1bit.v"
`include "Sub/not_64bit.v"
`include "Sub/sub_64bit.v"
`include "Xor/xor_1bit.v"
`include "Xor/xor_64bit.v"

module execute_test;

    integer k;

    wire [63:0] aluA;
    wire [63:0] aluB;
    wire [1:0] alufun;
    wire [2:0] cf;
    wire [63:0] valE;
    reg [63:0] valA;
    reg [63:0] valB;
    reg [63:0] valC;
    reg [3:0] icode;
    reg [3:0] ifun;
    wire [2:0] outf;
    wire cnd;
```

```
    alu_block
alulogic(.aluA(aluA),.aluB(aluB),.alufun(alufun),.cf(cf),.valE(valE));
    ALU_A alualogic(.icode(icode),.valA(valA),.valC(valC),.aluA(aluA));
    ALU_B alublogic(.icode(icode),.valB(valB),.aluB(aluB));
    ALU_fun alufunlogic(.icode(icode),.ifun(ifun),.alufun(alufun));
    set_CC cclogiv(.icode(icode),.cf(cf),.outf(outf));
    CND cndlogic(.ifun(ifun),.outf(outf),.cnd(cnd));



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

        // OpXX rA rB
        icode <= 4'h6;
        ifun  <= 4'h0;
        valA  <= 64'd30;
        valB  <= 64'd50;
        #10;
        icode <= 4'h6;
        ifun  <= 4'h1;
        valA  <= 64'd30;
        valB  <= 64'd50;
        #10;
        icode <= 4'h6;
        ifun  <= 4'h2;
        valA  <= 64'd30;
        valB  <= 64'd50;
        #10;
        icode <= 4'h6;
        ifun  <= 4'h3;
        valA  <= 64'd30;
        valB  <= 64'd50;
        #10;

        // move instructions
        icode <= 4'h3;
        ifun  <= 4'h0;
        valA  <= 64'd30;
```

```
        valB  <= 64'd50;
        valC  <= 64'd20;
        #10;
        icode <= 4'h4;
        ifun  <= 4'h0;
        valA  <= 64'd30;
        valB  <= 64'd50;
        valC  <= 64'd35;
        #10;
        icode <= 4'h5;
        ifun  <= 4'h0;
        valA  <= 64'd30;
        valB  <= 64'd50;
        valC  <= 64'd70;
        #10;

        // push pop instructions
        icode <= 4'hA;
        ifun  <= 4'h0;
        valA  <= 64'd30;
        valB  <= 64'd50;
        valC  <= 64'd20;
        #10;
        icode <= 4'hB;
        ifun  <= 4'h0;
        valA  <= 64'd30;
        valB  <= 64'd50;
        valC  <= 64'd20;
        #10;


        #10 $finish;
    end

endmodule
```

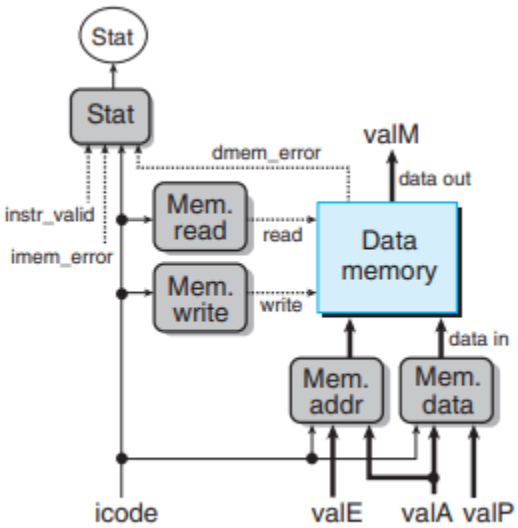
GTKwave Output



Memory

- It is used to either read or write program data.
- There are four blocks: two of which are control blocks used to generate values for memory address and memory input data and the other two blocks generate the control signals on whether to read or write the program data.

Structure



Verilog Modules

```
`timescale 1ns / 1ps

module RAM(memaddr, memdata, read, write, valM, dmemerror);

input[63:0] memaddr;
input[63:0] memdata;
input read;
input write;

reg [63:0] memory[8191:0];

output reg[63:0] valM;
output reg dmemerror;


always @(write, read, memdata, memaddr) begin

    if(read && !write) begin
        valM <= memory[memaddr];
    end

    if(write && !read) begin
        memory[memaddr] <= memdata;
    end

    if (memaddr >= 64'd258) begin
        dmemerror <= 1'b1;
    end
    else begin
        dmemerror <= 1'b0;

    end
end

endmodule

module MEM_addr(icode, valE, valA, memaddr);

input[3:0] icode;
```

```
input [63:0] valE;
input[63:0] valA;

output reg[63:0] memaddr;

always @(icode, valE, valA) begin


    case (icode)
        4'h4, 4'h5, 4'hA, 4'h8:
            memaddr <= valE;
        4'h9, 4'hB:
            memaddr <= valA;
    endcase

end

endmodule

module MEM_data(icode, valA, valP, memdata);

input[3:0] icode;
input [63:0] valA;
input[63:0] valP;

output reg[63:0] memdata;

always @(icode, valA, valP) begin

    case (icode)
        4'h4, 4'hA:
            memdata <= valA;
        4'h8:
            memdata<= valP;

    endcase
end

endmodule
```

```
module MEM_read (icode, read);

input [3:0] icode;
output reg read;

always @(icode) begin


    case (icode)
        4'h5, 4'hB, 4'h9:
            read <= 1'b1;
        default: read <= 1'b0;
    endcase
end
endmodule

module MEM_write (icode, write);

input [3:0] icode;
output reg write;

always @(icode) begin


    case (icode)
        4'h4, 4'hA, 4'h8:
            write <= 1'b1;

        default: write <= 1'b0;
    endcase
end
endmodule

module STAT(icode, instr_valid, imem_error, dmemerror, stat);
    input [3:0] icode;
    input imem_error;
    input instr_valid;
    input dmemerror;
    output reg[2:0] stat;
```

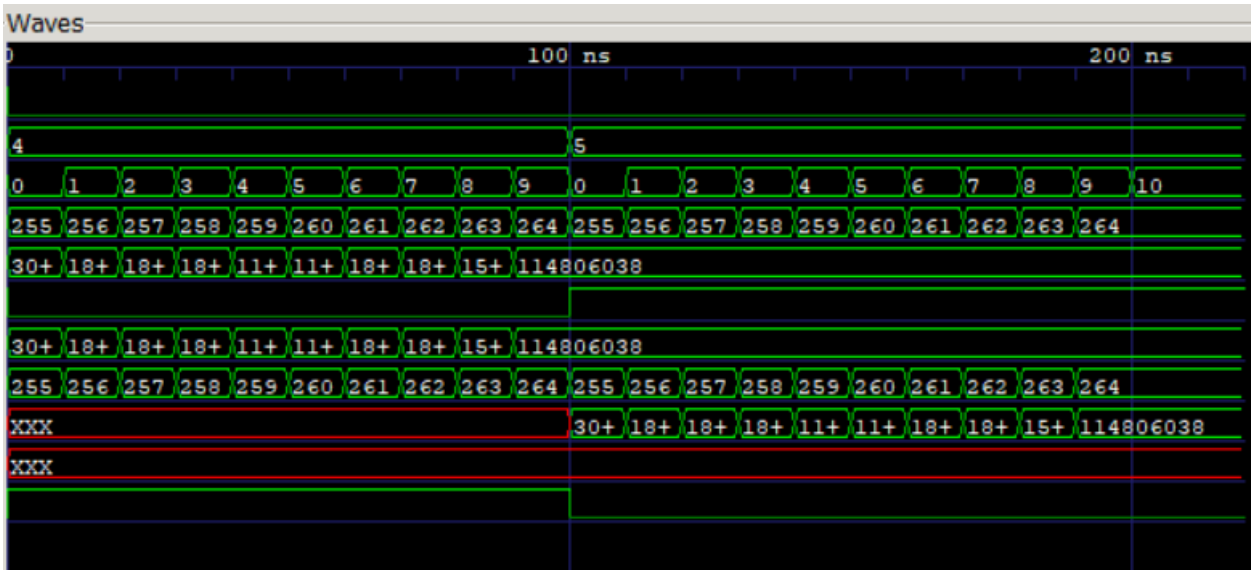
```
    parameter SAOK = 3'h1;
    parameter SHLT = 3'h2;
    parameter SADR = 3'h3;
    parameter SINS = 3'h4;

    always @(*)
    begin
        if(imem_error || dmemerror)
            stat <= SADR;
        else if (!instr_valid)
            stat <= SINS;
        else if (icode == 4'h0)
            stat <= SHLT;
        else stat <= SAOK;
    end

endmodule
```

Testbench

```
`timescale 1ns / 1ps
module ram_test;

    reg [3:0] icode;
    wire [63:0] memaddr;
    wire [63:0] memdata;
    reg [63:0] valE;
    reg [63:0] valA;
    reg [63:0] valP;
    wire [63:0] valM;
    wire read;
    wire write;

    RAM ram1(.memaddr(memaddr), .memdata(memdata), .read(read),
.write(write), .valM(valM), .dmemerror(dmemerror));
    MEM_addr Ma(.icode(icode), .valA(valA), .valE(valE),
.memaddr(memaddr));
    MEM_read Mr(.icode(icode), .read(read));
    MEM_write Mw(.icode(icode), .write(write));
```

```
    MEM_data Md(.icode(icode), .valA(valA), .valP(valP),
.memdata(memdata));


    integer k;
    parameter base_addr = 64'hFF;

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


    for(k=0;k<10;k++)
    begin
        icode <= 4'h4;
        valE  <= k + base_addr;
        valA  <= k+ $random;
        #10;
    end

    for(k=0;k<10;k++)
    begin
        icode <= 4'h5;
        valE  <= k + base_addr;
        #10;
    end

    #20 $finish;

    end

endmodule
```

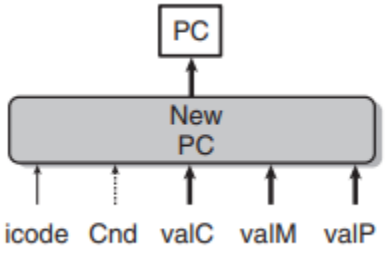
GTKWave Output



PC Update

- The new value of PC computed is either valP, valC or valM.
- The program counter PC is the only SEQ register which is based on the clock.

Structure



Verilog Modules

```
`timescale 1ns / 1ps

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

input clk, cnd;
input [63:0] valC;
input [63:0] valM;
```

```
input [63:0] valP;
input [3:0] icode;

output reg [63:0] newpc;

always@(*) begin


    if (icode == 4'b0111) //icode = 7
    begin
        if(cnd == 1'b1)
        begin
            newpc <= valC;
        end
        else begin
            newpc <= valP;
        end
    end

    else if (icode == 4'b1000) // icode = 8; for call
    begin
        newpc <= valC;
    end

    else if (icode == 4'b1001) // icode = 9; for ret
    begin
        newpc <= valM;
    end
    else
    begin
        newpc<= valP;
    end


end
endmodule
```

Processor - Sequence

- Now, since all the stage modules have been made, now, we are going to include all the files and define them in a separate module, from where we can run our final Y86-64 sequential processor.
- Here, we have defined all the variables used in the stages and every module created in all the five stages.

Verilog Module

```
`timescale 1ns / 1ps

`include "fetch/fetch.v"
`include "fetch/instrmem.v"
`include "decode/decode.v"
`include "Execute/execute.v"
`include "memory/ram.v"
`include "pc_update/pc_update.v"


`include "Execute/And/and_1bit.v"
`include "Execute/And/and_64bit.v"
`include "Execute/Xor/xor_1bit.v"
`include "Execute/Xor/xor_64bit.v"
`include "Execute/Or/or_1bit.v"

`include "Execute/Add/add_1bit.v"
`include "Execute/Add/add_64bit.v"

`include "Execute/Sub/not_1bit.v"
`include "Execute/Sub/not_64bit.v"
`include "Execute/Sub/sub_64bit.v"


module proc;

reg clk;
reg [63:0] pc;

// fetch, intruction memory
wire [3:0] icode;
wire [3:0] ifun;
wire [7:0] Byte0;
```

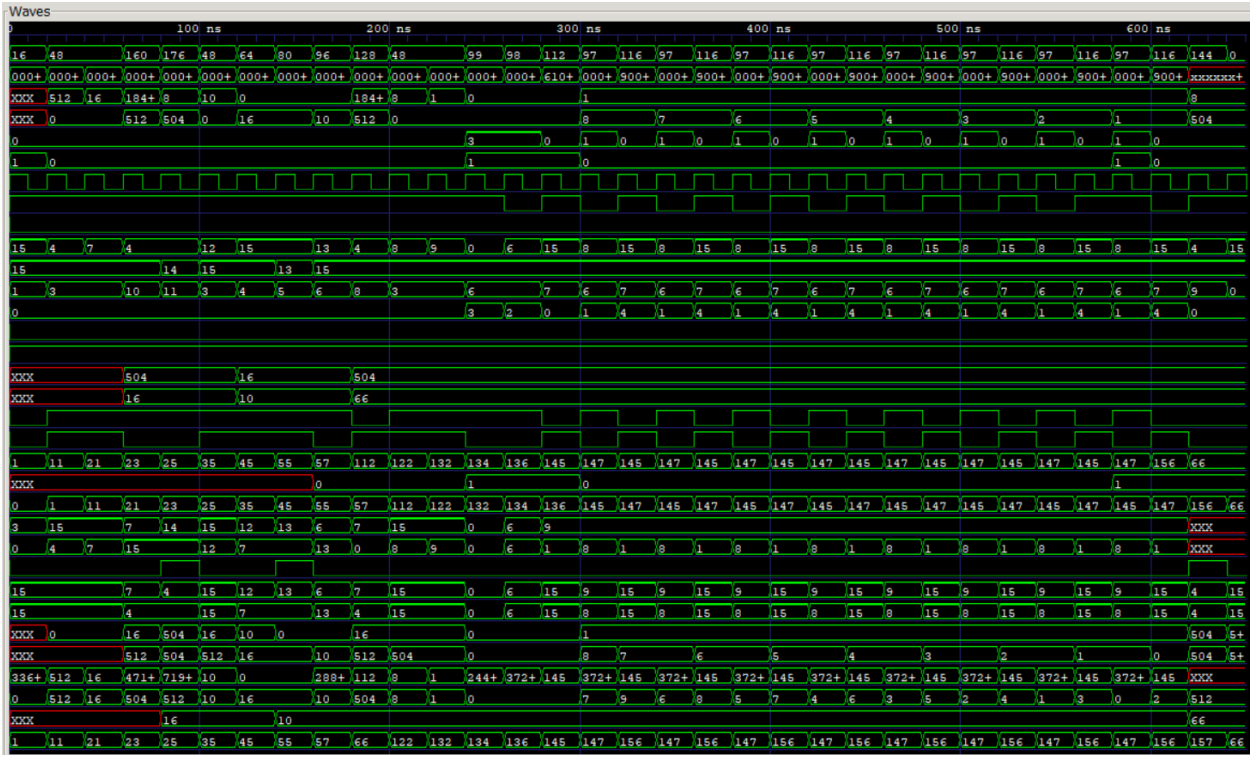
```
wire [71:0] Byte19;
wire need_valC;
wire need_regids;
wire [63:0] valC;
wire [3:0] rA;
wire [3:0] rB;
wire [63:0] valP;
wire imem_error;
wire instr_valid; // reg or wire?

// decode
wire [3:0] dstE;
wire [3:0] dstM;
wire [3:0] srcA;
wire [3:0] srcB;
wire [63:0] valE;
wire [63:0] valM;
wire [63:0] valA;
wire [63:0] valB;
// execute
wire [63:0] aluA;
wire [63:0] aluB;
wire [1:0] alufun;
wire [2:0] cf;
wire [2:0] outf;
wire cnd;

// memory
wire [63:0] memaddr;
wire [63:0] memdata;
wire read;
wire write;
wire dmemerror;
reg [2:0] stat;

// pc update
wire [63:0] newpc;


// fetch
```

```
split sp(.Byte0(Byte0), .icode(icode), .ifun(ifun));
align al(.Byte19(Byte19), .need_regids(need_regids), .rA(rA), .rB(rB),
.valC(valC));
PC_INCREMENT PC_i(.pc(pc), .icode(icode), .need_regids(need_regids),
.need_valC(need_valC), .valP(valP));
INSTR_VALID i_valid(.icode(icode), .instr_valid(instr_valid));
Need_REGIDS nreg(.icode(icode), .need_regids(need_regids));
Need_VALC n_valC(.icode(icode), .need_valC(need_valC));
instruction_memory
InstMem(.clk(clk),.pc(pc),.imem_error(imem_error),.Byte0(Byte0),.Byte19(By
te19));

// decode
registerfile reg_f(.clk(clk), .dstE(dstE), .dstM(dstM), .srcA(srcA) ,
.srcB(srcB) , .valA(valA) , .valB(valB) , .valM(valM), .valE(valE));
srcA_logic sA_l(.icode(icode), .rA(rA), .srcA(srcA));
srcB_logic sB_l(.icode(icode), .rB(rB), .srcB(srcB));
dstE_logic dE_l(.icode(icode), .rB(rB), .cnd(cnd), .ifun(ifun),
.dstE(dstE));
dstM_logic dM_l(.icode(icode), .rA(rA), .dstM(dstM));

// execute
alu_block alulogic(.aluA(aluA), .aluB(aluB), .alufun(alufun), .cf(cf),
.valE(valE));
ALU_A alualogic(.icode(icode), .valA(valA), .valC(valC), .aluA(aluA));
ALU_B alublogic(.icode(icode), .valB(valB), .aluB(aluB));
ALU_fun alufunlogic(.icode(icode), .ifun(ifun), .alufun(alufun));
set_CC cclogiv(.icode(icode), .cf(cf), .outf(outf));
CND cndlogic(.ifun(ifun), .outf(outf), .cnd(cnd));

// memory
RAM ram1(.memaddr(memaddr), .memdata(memdata), .read(read), .write(write),
.valM(valM), .dmemerror(dmemerror));
MEM_addr Ma(.icode(icode), .valA(valA), .valE(valE), .memaddr(memaddr));
MEM_read Mr(.icode(icode), .read(read));
MEM_write Mw(.icode(icode), .write(write));
MEM_data Md(.icode(icode), .valA(valA), .valP(valP), .memdata(memdata));

// PC Update
```

```
pc_update pc_new(.clk(clk), .cnd(cnd), .icode(icode), .valC(valC),
.valM(valM), .valP(valP), .newpc(newpc));

/* // sub 64 bit
wire [63:0] suba;
wire [63:0] subb;
wire [63:0] subout;
wire [2:0] subflag;
sub_64bit check(.a(suba),.b(subb),.out(subout),.cf_sub(subflag));
// add 64 bit
wire [63:0] adda;
wire [63:0] addb;
wire [63:0] addout;
wire [2:0] addflag;
add_64bit chec_k(.a(adda),.b(addb),.out(addout),.cf_add(addflag)); */


initial
begin

    $dumpfile("proc.vcd");
    $dumpvars(0, proc);
    // $readmemh("rom.mem", instr_mem);

    clk = 1'b1;
    pc = 64'd0;

end


always @(posedge clk)
      begin
        pc <= newpc;
      end

    always #10 clk <= ~clk;
    initial
        #650 $finish;

initial begin
```

```
  //$monitor("clk=%d, pc=%d, icode=%d, ifun=%d, rA=%b, rB=%b, valC=%d,
valP=%d, aluA = %d, aluB = %d, valA=%d, valB=%d,valE=%d, valM=%d, alufun
=%d,read = %b,write = %b,memaddr=%d,memdata = %d,newpc = %d\n", clk, pc,
icode, ifun, rA, rB, valC, valP, aluA, aluB, valA, valB, valE,
valM,alufun,read,write,memaddr,memdata,newpc);
  //$monitor("clk=%d, pc=%d, icode=%d, ifun=%d, rA=%b, rB=%b",clk, pc,
icode, ifun, rA, rB,);
    $monitor("clk=%d, pc=%d, icode=%d, ifun=%d, rA=%b, rB=%b, valC=%d,
valP=%d, aluA = %d, aluB = %d, alufun = %b ,valA=%d, valB=%d,valE=%d, outf
= %b,cf =%b,cnd =%b\n",
    clk, pc, icode, ifun, rA, rB, valC, valP, aluA, aluB,alufun,valA,
valB, valE,outf,cf,cnd );
end
endmodule
```

GTKwave Output



Instruction Supported

- halt
- nop

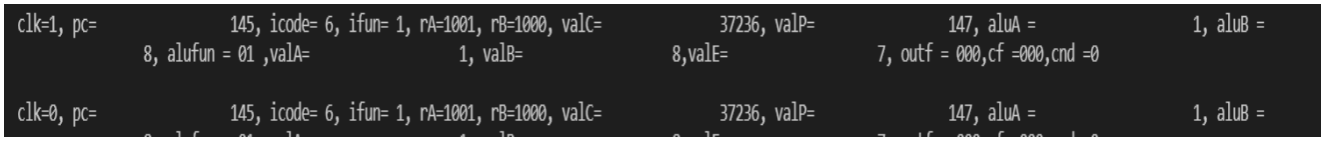
- cmovxx
- irmovq
- rmmovq
- Opq
- jxx
- call
- ret
- pushq
- popq

Instruction Memory:

1. 0x0 Nop
2. 0x1 irmovq %512 %rsi – Initializing the stack pointer
3. 0x11 irmovq %16 %rdi
4. 0x21 pushq %rdi
5. 0x23 popq %r14
6. 0x25 irmovq %10 %r12
7. 0x35 rmmovq %r12 %(rdi)
8. 0x45 mrmovq %r13 %(rdi)
9. 0x55 add %r12 %r13
10. 0x57 call
11. 0x66 halt
12. 0x112 irmovq $8 %r8
13. 0x122 irmovq %1 %r9
14. 0x132 Xorq %rax %rax
15. 0x134 Andq %rsi %rsi
16. 0x136 jmp %112
17. 0x145 subq %r9 %r8
18. 0x147 jne 147
19. 0x156 ret

Execution of Instructions in Sequential

1. **0x145 subq %r9 %r8**
2. **0x147 jne 147**
3. **0x156 ret**



```
clk=1, pc=              145, icode= 6, ifun= 1, rA=1001, rB=1000, valC=          37236, valP=          147, aluA =                1, aluB =
             7, alufun = 01 ,valA=                1, valB=              7,valE=              6, outf = 000,cf =000,cnd =0

clk=0, pc=              145, icode= 6, ifun= 1, rA=1001, rB=1000, valC=          37236, valP=          147, aluA =                1, aluB =
             7, alufun = 01 ,valA=                1, valB=              7,valE=              6, outf = 000,cf =000,cnd =0
```

```
clk=1, pc=              147, icode= 7, ifun= 4, rA=1001, rB=0001, valC=          145, valP=          156, aluA =                1, aluB =
             8, alufun = 00 ,valA=                1, valB=              7,valE=              9, outf = 000,cf =000,cnd =1

clk=0, pc=              147, icode= 7, ifun= 4, rA=1001, rB=0001, valC=          145, valP=          156, aluA =                1, aluB =
             8, alufun = 00 ,valA=                1, valB=              7,valE=              9, outf = 000,cf =000,cnd =1
```

```
clk=1, pc=              156, icode= 9, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=          157, aluA =                8, aluB =
           504, alufun = 00 ,valA=              504, valB=            504,valE=            512, outf = 001,cf =000,cnd =1

clk=0, pc=              156, icode= 9, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=          157, aluA =                8, aluB =
           504, alufun = 00 ,valA=              504, valB=            504,valE=            512, outf = 001,cf =000,cnd =1
```

1. **0x25 irmovq 10 %r12**
2. **0x35 rmmovq %r12 %(rdi)**
3. **0x45 mrmovq %r13 %(rdi)**

```
clk=1, pc=               25, icode= 3, ifun= 0, rA=1111, rB=1100, valC=           10, valP=           35, aluA =               10, aluB =
             0, alufun = 00 ,valA=               16, valB=            512,valE=            10, outf = xxx,cf =000,cnd =1

clk=0, pc=               25, icode= 3, ifun= 0, rA=1111, rB=1100, valC=           10, valP=           35, aluA =               10, aluB =
             0, alufun = 00 ,valA=               16, valB=            512,valE=            10, outf = xxx,cf =000,cnd =1
```

```
clk=1, pc=               35, icode= 4, ifun= 0, rA=1100, rB=0111, valC=            0, valP=           45, aluA =                0, aluB =
            16, alufun = 00 ,valA=               10, valB=             16,valE=            16, outf = xxx,cf =000,cnd =1

clk=0, pc=               35, icode= 4, ifun= 0, rA=1100, rB=0111, valC=            0, valP=           45, aluA =                0, aluB =
            16, alufun = 00 ,valA=               10, valB=             16,valE=            16, outf = xxx,cf =000,cnd =1
```

```
clk=1, pc=               45, icode= 5, ifun= 0, rA=1101, rB=0111, valC=            0, valP=           55, aluA =                0, aluB =
            16, alufun = 00 ,valA=                0, valB=             16,valE=            16, outf = xxx,cf =000,cnd =1

clk=0, pc=               45, icode= 5, ifun= 0, rA=1101, rB=0111, valC=            0, valP=           55, aluA =                0, aluB =
            16, alufun = 00 ,valA=                0, valB=             16,valE=            16, outf = xxx,cf =000,cnd =1
```

1. **0x57 call**
2. **0x66 halt**

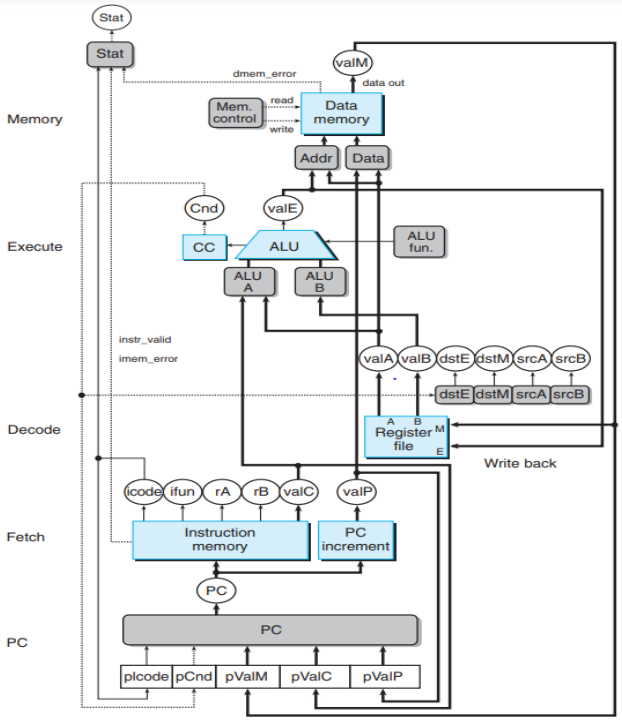
```
clk=1, pc=               57, icode= 8, ifun= 0, rA=0111, rB=0000, valC=          112, valP=                66, aluA = 18446744073709551608, aluB =
           512, alufun = 00 ,valA=               16, valB=            512,valE=            504, outf = 000,cf =000,cnd =1

clk=0, pc=               57, icode= 8, ifun= 0, rA=0111, rB=0000, valC=          112, valP=                66, aluA = 18446744073709551608, aluB =
           512, alufun = 00 ,valA=               16, valB=            512,valE=            504, outf = 000,cf =000,cnd =1
```

```
clk=1, pc=               66, icode= 0, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=           66, aluA =                8, aluB =
           504, alufun = 00 ,valA=              512, valB=            512,valE=            512, outf = 001,cf =000,cnd =1

clk=0, pc=               66, icode= 0, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=           66, aluA =                8, aluB =
           504, alufun = 00 ,valA=              512, valB=            512,valE=            512, outf = 001,cf =000,cnd =1

clk=1, pc=               66, icode= 0, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=           66, aluA =                8, aluB =
           504, alufun = 00 ,valA=              512, valB=            512,valE=            512, outf = 001,cf =000,cnd =1

clk=0, pc=               66, icode= 0, ifun= 0, rA=xxxx, rB=xxxx, valC=            x, valP=           66, aluA =                8, aluB =
           504, alufun = 00 ,valA=              512, valB=            512,valE=            512, outf = 001,cf =000,cnd =1
```

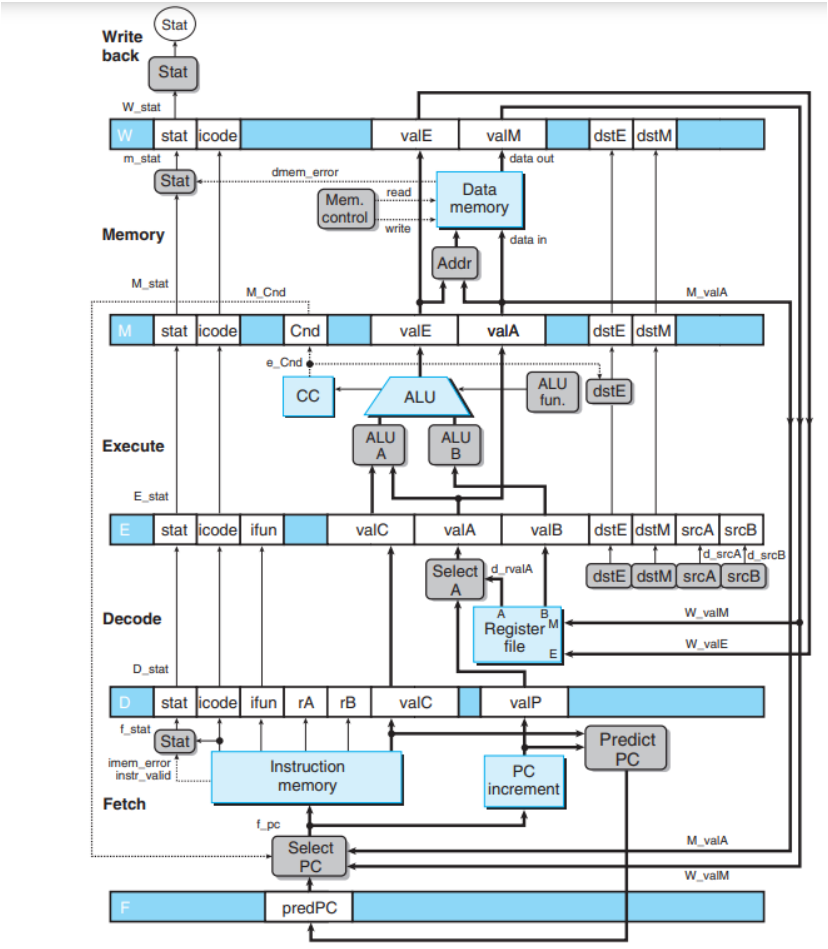
Pipelining

- It includes the rearranging of the computation stages.
- We do the process of rearranging such that the PC update stage comes at the beginning of the clock cycle rather than at the end.
- Pipeline registers are being inserted between the stages of SEQ+, and further rearranging, which helps in generating the PIPE.
- The PIPE registers are F, D, E, M and W.

SEQ+ Hardware Structure



Hardware Structure of PIPE



Pipeline Control Logic

Verilog Modules

```
`timescale 1ns / 1ps

module PIPE_CONTROL_LOGIC(

input [3:0] D_icode,
input [3:0] d_srcA,
input [3:0] d_srcB,
input [3:0] E_icode,
input [3:0] E_dstM,
input e_Cnd,
input [3:0] M_icode,
input [2:0] m_stat,
```

```
input [2:0] W_stat,

output reg F_stall,
output reg D_stall,
output reg D_bubble,
output reg E_bubble,
output reg M_bubble,
output reg W_stall
);
parameter SBUB = 3'h0;
parameter SAOK = 3'h1;
parameter SHLT = 3'h2;
parameter SADR = 3'h3;
parameter SINS = 3'h4;

    initial
    begin
        F_stall <= 1'b0;
        D_stall <= 1'b0;
        D_bubble <= 1'b0;
        E_bubble <= 1'b0;
        M_bubble <= 1'b0;
        W_stall <= 1'b0;
    end


    always @(*) begin
        F_stall <= ( ((E_icode == 4'h5 || E_icode == 4'hB) && (E_dstM ==
d_srcA || E_dstM == d_srcB)) || (D_icode == 4'h9 || E_icode == 4'h9 ||
M_icode == 4'h9));
        D_bubble <= (( E_icode == 4'h7 && !e_Cnd ) || !(E_icode == 4'h5 ||
E_icode == 4'hB)  && (E_dstM == d_srcA || E_dstM== d_srcB)&& (D_icode ==
4'h9 || E_icode == 4'h9 || M_icode == 4'h9));
        D_stall <= (E_icode == 4'h5 || E_icode == 4'hB) && (E_dstM ==
d_srcA || E_dstM == d_srcB);
        E_bubble <= (( E_icode == 4'h7 && !e_Cnd ) || (E_icode == 4'h5 ||
E_icode == 4'hB)  && (E_dstM == d_srcA || E_dstM== d_srcB));
        M_bubble <= (m_stat == SADR ||  m_stat == SHLT || m_stat == SINS)
|| (W_stat == SADR ||  W_stat == SHLT || W_stat == SINS);
        W_stall <= (W_stat == SADR ||  W_stat == SHLT || W_stat == SINS);
```

```
    end

endmodule
```

Outputs of Stage _Register File have been put in text files

Instruction Supported

- halt
- nop
- cmovxx
- irmovq
- rmmovq
- Opq
- jxx
- call
- ret
- pushq
- popq

Instruction Memory:

1. 0x0 Nop
2. 0x1 irmovq %512 %rsi – Initializing the stack pointer
3. 0x11 irmovq %16 %rdi
4. 0x21 pushq %rdi
5. 0x23 popq %r14
6. 0x25 irmovq %10 %r12
7. 0x35 rmmovq %r12 %(rdi)
8. 0x45 mrmovq %r13 %(rdi)
9. 0x55 add %r12 %r13
10. 0x57 call
11. 0x66 halt
12. 0x112 irmovq $8 %r8
13. 0x122 irmovq %1 %r9
14. 0x132 Xorq %rax %rax
15. 0x134 Andq %rsi %rsi
16. 0x136 jmp %112
17. 0x145 subq %r9 %r8
18. 0x147 jne 147
19. 0x156 ret

1. **0x145 subq %r9 %r8**
2. **0x147 jne 147**

3. **0x156 ret**

```
clk=1,F_PC=     145,M_stat=001,M_icode=7,M_Cnd=1,M_valE=     4,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
clk=0,F_PC=     145,M_stat=001,M_icode=7,M_Cnd=1,M_valE=     4,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
clk=1,F_PC=     147,M_stat=001,M_icode=6,M_Cnd=0,M_valE=     1,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0,
clk=0,F_PC=     147,M_stat=001,M_icode=6,M_Cnd=0,M_valE=     1,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0,
clk=1,F_PC=     145,M_stat=001,M_icode=7,M_Cnd=1,M_valE=     3,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
clk=0,F_PC=     145,M_stat=001,M_icode=7,M_Cnd=1,M_valE=     3,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
clk=1,F_PC=     147,M_stat=001,M_icode=6,M_Cnd=1,M_valE=     0,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0,
clk=0,F_PC=     147,M_stat=001,M_icode=6,M_Cnd=1,M_valE=     0,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0,
clk=1,F_PC=     156,M_stat=001,M_icode=7,M_Cnd=0,M_valE=     2,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
clk=0,F_PC=     156,M_stat=001,M_icode=7,M_Cnd=0,M_valE=     2,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0,
```

Execution of Instructions In Pipeline

Load use hazard

1. mrmovq %r13 %(rdi)
2. add %r12 %r13

Stall Values For above Instructions

```
clk=1,F_PC=     45,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=     45,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=     55,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=     55,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=     57,F_stall = 1,D_stall = 1, D_bubble = 0, W_stall = 0,E_bubble = 1, M_bubble = 0
clk=0,F_PC=     57,F_stall = 1,D_stall = 1, D_bubble = 0, W_stall = 0,E_bubble = 1, M_bubble = 0
```

Memory stage reg values for above instructions

```
clk=1,F_PC=     156,M_stat=001,M_icode=7,M_Cnd=0,M_valE=                 2,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0
clk=0,F_PC=     156,M_stat=001,M_icode=7,M_Cnd=0,M_valE=                 2,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0
clk=1,F_PC=     157,M_stat=001,M_icode=6,M_Cnd=1,M_valE=18446744073709551615,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0
clk=0,F_PC=     157,M_stat=001,M_icode=6,M_Cnd=1,M_valE=18446744073709551615,M_valA=       1,M_dstE=8,M_dstM=f,M_bubble=0
clk=1,F_PC=     157,M_stat=001,M_icode=7,M_Cnd=1,M_valE=                 1,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0
clk=0,F_PC=     157,M_stat=001,M_icode=7,M_Cnd=1,M_valE=                 1,M_valA=     156,M_dstE=f,M_dstM=f,M_bubble=0
clk=1,F_PC=     157,M_stat=001,M_icode=9,M_Cnd=1,M_valE=               504,M_valA=     496,M_dstE=4,M_dstM=f,M_bubble=0
clk=0,F_PC=     157,M_stat=001,M_icode=9,M_Cnd=1,M_valE=               504,M_valA=     496,M_dstE=4,M_dstM=f,M_bubble=0
clk=1,F_PC=      66,M_stat=001,M_icode=x,M_Cnd=1,M_valE=               504,M_valA=      10,M_dstE=f,M_dstM=f,M_bubble=0
clk=0,F_PC=      66,M_stat=001,M_icode=x,M_Cnd=1,M_valE=               504,M_valA=      10,M_dstE=f,M_dstM=f,M_bubble=0
```

Mispredicted Branch

0x145

4. subq %r9 %r8
5. jne 147

```
clk=1,F_PC=             145,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             145,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
```

```
clk=0,F_PC=             145,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             145,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             145,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             147,F_stall = 0,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
```

Processing Ret

1. jne 147
2. ret

```
clk=1,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             156,F_stall = 0,D_stall = 0, D_bubble = 0, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = 0, D_bubble = 1, W_stall = 0,E_bubble = 0, M_bubble = 0
clk=1,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=             157,F_stall = 1,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=1,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
clk=0,F_PC=              66,F_stall = x,D_stall = x, D_bubble = x, W_stall = 0,E_bubble = x, M_bubble = 0
```

```
clk=1,F_PC=             156,f_stat=001,f_icode=9,f_ifun=0,f_rA=x,f_rB=x,f_valC=             x,f_valP=          157
clk=0,F_PC=             156,f_stat=001,f_icode=9,f_ifun=0,f_rA=x,f_rB=x,f_valC=             x,f_valP=          157
clk=1,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=0,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=1,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=0,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=1,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=0,F_PC=             157,f_stat=001,f_icode=x,f_ifun=x,f_rA=x,f_rB=x,f_valC=             x,f_valP=          158
clk=1,F_PC=              66,f_stat=010,f_icode=0,f_ifun=0,f_rA=x,f_rB=x,f_valC=             x,f_valP=           66
clk=0,F_PC=              66,f_stat=010,f_icode=0,f_ifun=0,f_rA=x,f_rB=x,f_valC=             x,f_valP=           66
```

Challenges Faced

1. There were errors in variable declaration which took a lot of time to debug.

2. There were errors in my ALU subtraction block due to which I had to suffer 2-3 hours..
3. Figuring out the control logic and implementing it correctly was also rather challenging especially in case of the pipelined processor.

Acknowledgements

We had a great learning experience in knowing about different concepts and in debugging the codes. I would also like to thank our prof. Deepak Gangadharan and our TAs for giving us good teachings.