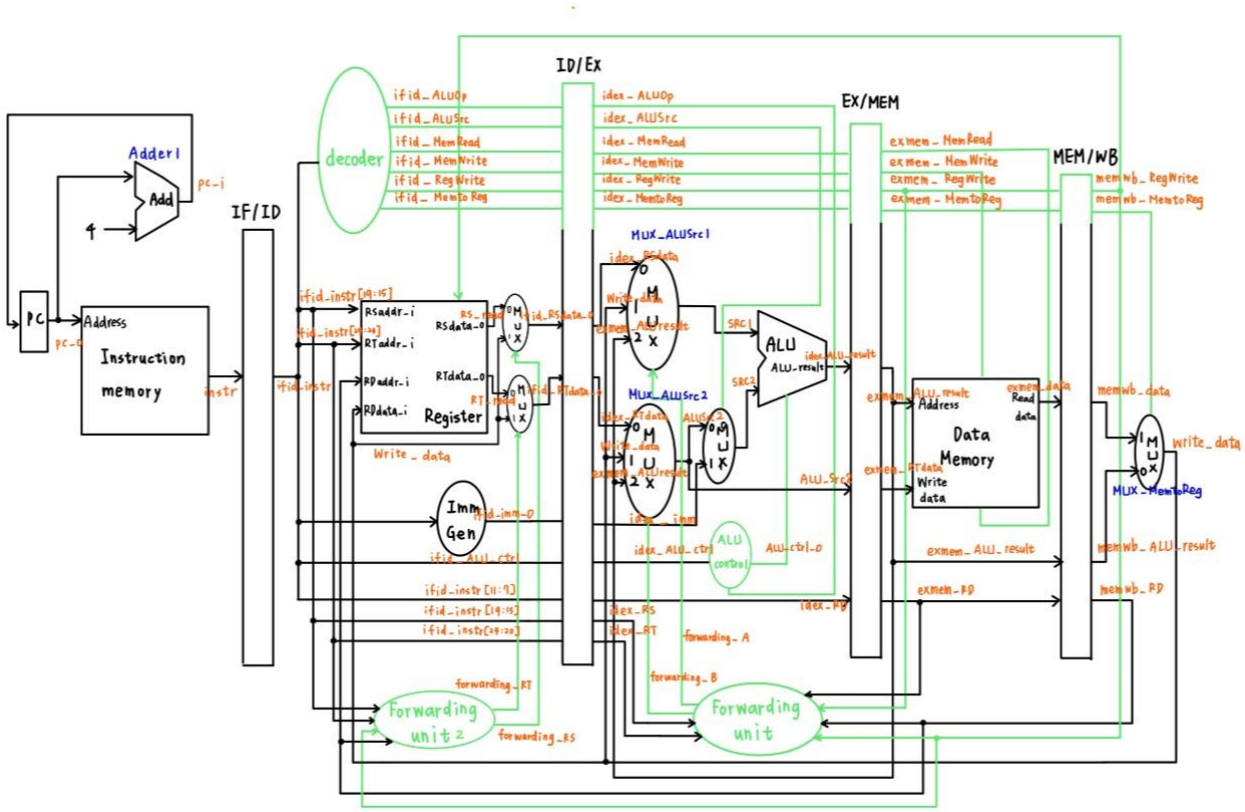
Computer Organization

Architecture diagram:



Detailed description of the implementation:

- Adder

```
assign sum_o = src1_i + src2_i;
```

- IF_ID

```
always @(posedge clk_i) begin
    if(~rst_i)
        instr_o <= 0;
    else
        begin
        instr_o <= instr_i;
        end
end
```

- alu

```
always @(*) begin
    if(rst_n) begin
        case(ALU_control)
            4'b0000:
                result <= src1 - src2; //sub
            4'b0010:
                result <= src1 + src2; //add, addi
            4'b0011:
                result <= src1 & src2; //and, andi
            4'b0100:
                result <= src1 | src2; //or, ori
            4'b0101:
                result <= src1 ^ src2; //xor, xori
            4'b0110:
                result <= (src1 < src2); //slt, slti
            4'b0111:
                result <= src1 << src2; //sll, slli
            4'b1000:
                result <= src1 >>> src2; //sra
            4'b1001:
                result <= src1 >> src2; //srli
        endcase
    end
end
```

- ALU_Ctrl

```
always @(*) begin
    case(ALUOp)
        2'b10: begin //R type
            case(instr)
                4'b0000: ALU_Ctrl_o <= 4'b0010; //add
                4'b1000: ALU_Ctrl_o <= 4'b0000; //sub
                4'b0111: ALU_Ctrl_o <= 4'b0011; //and
                4'b0110: ALU_Ctrl_o <= 4'b0100; //or
                4'b0100: ALU_Ctrl_o <= 4'b0101; //xor
                4'b0010: ALU_Ctrl_o <= 4'b0110; //slt
                4'b0001: ALU_Ctrl_o <= 4'b0111; //sll
                4'b1101: ALU_Ctrl_o <= 4'b1000; //sra
            endcase
        end
        2'b11: begin //I type
            case(instr[3-1:0])
                3'b000: ALU_Ctrl_o <= 4'b0010; //addi
                3'b010: ALU_Ctrl_o <= 4'b0110; //slti
                3'b001: ALU_Ctrl_o <= 4'b0111; //slli
                3'b101: ALU_Ctrl_o <= 4'b1001; //srli
                3'b111: ALU_Ctrl_o <= 4'b0011; //andi
                3'b110: ALU_Ctrl_o <= 4'b0100; //ori
                3'b100: ALU_Ctrl_o <= 4'b0101; //xori
            endcase
        end
    endcase
end
```

- Decoder

```
wire [6:0] opcode;
assign opcode = instr_i[6:0];

assign ALUOp = (opcode == 7'b0110011)?2'b10: //R-type
               (opcode == 7'b0010011)?2'b11: //I-type
               2'bxx;

assign ALUSrc = (opcode == 7'b0110011)?0: //R-type
                (opcode == 7'b0010011)?1: //I-type
                1'bx;

assign MemtoReg = (opcode == 7'b0110011)?0: //R-type
                  (opcode == 7'b0010011)?0: //I-type
                  1'bx;

assign RegWrite = (opcode == 7'b0110011)?1: //R-type
                  (opcode == 7'b0010011)?1: //I-type
                  0; //default

assign MemRead  = 0;
assign MemWrite = 0;
```

- EX_MEM

```
always @(posedge clk_i) begin
    if(~rst_i)
        begin
        MemtoReg_o <= 0;
        RegWrite_o <= 0;
        MemRead_o <= 0;
        MemWrite_o <= 0;
        ALU_result_o <= 0;
        RT_data_o <= 0;
        RD_o <= 0;
        end
    else
        begin
        MemtoReg_o <= MemtoReg_i;
        RegWrite_o <= RegWrite_i;
        MemRead_o <= MemRead_i;
        MemWrite_o <= MemWrite_i;
        ALU_result_o <= ALU_result_i;
        RT_data_o <= RT_data_i;
        RD_o <= RD_i;
        end
end
```

- ForwardingUnit

```
always @(*) begin
    forwarding_A <= 2'b00;
    forwarding_B <= 2'b00;
    if(exmem_WB&&exmem_RD!=0&&exmem_RD==Rs1)
        forwarding_A <= 2'b10;
    if(exmem_WB&&exmem_RD!=0&&exmem_RD==Rs2)
        forwarding_B <= 2'b10;
    if(!(exmem_WB&&exmem_RD!=0&&exmem_RD==Rs1)&&memwb_WB&&memwb_RD!=0&&memwb_RD==Rs1)
        forwarding_A <= 2'b01;
    if(!(exmem_WB&&exmem_RD!=0&&exmem_RD==Rs2)&&memwb_WB&&memwb_RD!=0&&memwb_RD==Rs2)
        forwarding_B <= 2'b01;
end
```

- ForwardingUnit2

```
always @(*) begin
    forwarding_RS <= 0;
    forwarding_RT <= 0;
    if(memwb_WB&&memwb_WB!=0&&memwb_RD==Rs1)
        forwarding_RS <= 1;
    if(memwb_WB&&memwb_WB!=0&&memwb_RD==Rs2)
        forwarding_RT <= 1;
end
```

- MEM_WB

```
always @(posedge clk_i) begin
    if(~rst_i)
        begin
        MemtoReg_o <= 0;
        RegWrite_o <= 0;
        mem_o <= 0;
        ALU_result_o <= 0;
        RD_o <= 0;
        end
    else
        begin
        MemtoReg_o <= MemtoReg_i;
        RegWrite_o <= RegWrite_i;
        mem_o <= mem_i;
        ALU_result_o <= ALU_result_i;
        RD_o <= RD_i;
        end
end
```

- ID_EX

```
always @(posedge clk_i) begin
    if(~rst_i)
        begin
        ALUSrc_o <= 0;
        MemtoReg_o <= 0;
        RegWrite_o <= 0;
        MemRead_o <= 0;
        MemWrite_o <= 0;
        ALUOp_o <= 0;
        RS_data_o <= 0;
        RT_data_o <= 0;
        imm_o <= 0;
        ALU_ctrl_o <= 0;
        RS_o <= 0;
        RT_o <= 0;
        RD_o <= 0;
        end
    else
        begin
        ALUSrc_o <= ALUSrc_i;
        MemtoReg_o <= MemtoReg_i;
        RegWrite_o <= RegWrite_i;
        MemRead_o <= MemRead_i;
        MemWrite_o <= MemWrite_i;
        ALUOp_o <= ALUOp_i;
        RS_data_o <= RS_data_i;
        RT_data_o <= RT_data_i;
        imm_o <= imm_i;
        ALU_ctrl_o <= ALU_ctrl_i;
        RS_o <= RS_i;
        RT_o <= RT_i;
        RD_o <= RD_i;
        end
end
```

- Imm_Gen

```
wire [7-1:0] opcode;
assign opcode = instr_i[6:0];

always @(*) begin
    case(opcode)
        7'b0010011: //I type
            Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[31:20]};
    endcase
end
```

- MUX_2to1

```
always @(*) begin
    if(select_i)
        data_o <= data1_i;
    else
        data_o <= data0_i;
end
```

- MUX_3to1

```
always @(select_i,data0_i,data1_i) begin
    if(select_i == 2'b00)
        data_o <= data0_i;
    else if(select_i == 2'b01)
        data_o <= data1_i;
    else
        data_o <= data2_i;
end
```

Implementation results:

Test Data 1

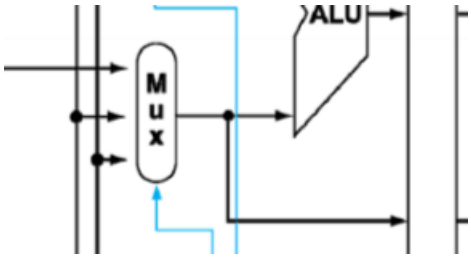
```
# PC =        136
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Registers
# R0 =          0, R1 =         50, R2 =         18, R3 =         32, R4 =         82, R5 =
# R8 =          0, R9 =          0, R10 =          0, R11 =          0, R12 =          0, R13 =
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =
```

Test Data 2

```
# PC =        136
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Data Memory =          0,          0,          0,          0,          0,          0,          0,
# Registers
# R0 =          0, R1 =         23, R2 =         13, R3 =         16, R4 =         29, R5 =
# R8 =          8, R9 =         41, R10 =          0, R11 =          0, R12 =          0, R13 =
# R16 =          0, R17 =          0, R18 =          0, R19 =          0, R20 =          0, R21 =
# R24 =          0, R25 =          0, R26 =          0, R27 =          0, R28 =          0, R29 =
```

Problems encountered and solutions:

1. 加入 ForwardingUnit 之後，ALU_Src2 一開始我們是先用 MUX_2to1(原本、imm)再用 MUX_3to1(2to1、wb、mem)，然後照著講義的圖把結果存到 EX/MEM pipelime register。後來發現這樣子 Write data 有可能會變成要跟 base address 相加的 offset。後來發現對調 2 個 MUX 的順序就能解決這個問題。



2. test_data2 從第 5 個指令開始出錯，仔細看了波形圖之後發現，ID stage 在讀取 register 的時候，WB stage 的新資料還沒有寫回 register，所以會讀取到舊的資料。後來發現這 2 個動作都是在 posedge 時同時發生，所以我們把 register_file 中 always block 的 sensitive list 從 posedge 改成 negedge，這樣就能讓 WB 先完成，並且在下半 cycle 才讀取 register 了。

3. 結果助教後來說不能改 register_file，要另外加一個 ForwardingUnit，還好其實就原本的複製過去改一下就好了，沒有很困難。

Comment:

這次作業我們是複製空白的檔案重新寫的，因為要改的東西太多了，怕一個不注意有地方沒改到或是改錯。還好經過這麼多次作業，我們已經對內容很熟悉了所以也沒有花很多時間。

最大的挑戰就是在 top module 取變數名了吧！因為每個東西都要先存到 pipeline register 再讀出來，原本 1 條線可以搞定的東西要拆成 2 條線，所以取名也要小心不然會出現跳級的現象。我們最後想到要把 pipeline register 冠名到變數名，這樣就可以知道那條線是從哪裡接出來了。

發現這次作業很多指令都用不到之後，我們刪掉了很多原本 CPU 有的東西，像是 zero、branch、PCSrc 等等。雖然之後可能會需要把他們接回去，但我們還是選擇了精簡化這

次的 CPU，以避免 debug 困難。結果我們除了語法錯誤一堆以外都還滿順利的，根本沒花很多時間 debug，自己解決了 register I/O 的問題還開心了很久。

p.s.結果發現這是最後一次 verilog 作業了，果然刪掉一堆用不到的東西是正確的決定，終於不用再畫 architecture 了！