**Project** **1**

*Report 2*

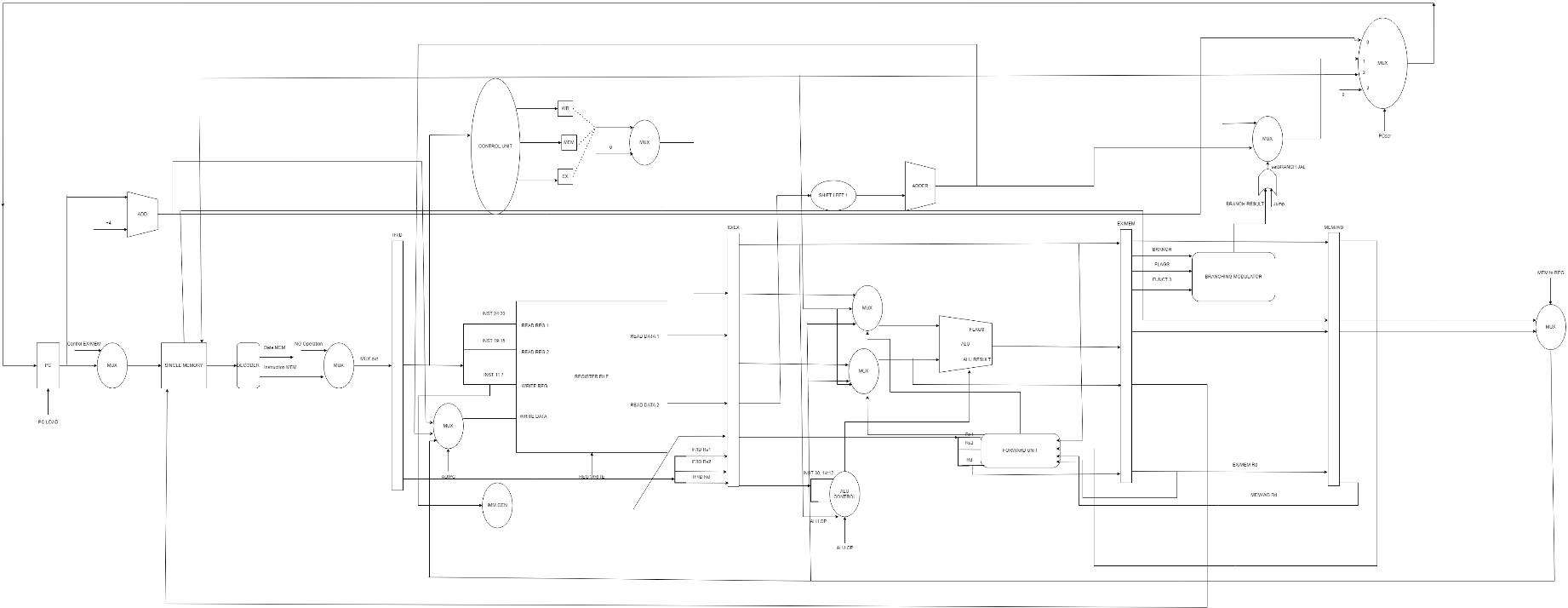
*By*

***Mohamed waleed elshemy*** *&* ***Noureldeen Amr Allama***

Project **Milestone Update**:

In this Milestone, we proceeded with implementing the necessary changes to convert and adapt our Single Cycle (SC) we have done in the previous milestone into the Pipelined Cycle (PC) this milestone requires, for faster performance and exception/hazard handling. We also finished support for the instructions that remained unsupported from the previous milestone. The Pipelined Datapath now supports the full required RV32I base integer instruction (40 instructions)—including the previously unsupported instructions (LUI, AUPIC, JAL, JALR, BNE, BLT, BGE, BLTU, BGEU, LB, LH, LBU, LHU, SB, SH). Test cases for each instruction set was also provided. The main Pipelined implementation required certain changes and additions to the previous Single Cycle:

1. The fusion of both the Instruction Memory and the Data Memory into one cohesive Single-Ported Memory.
2. The addition of 4 new register for the 5 procedures of processing an instruction (IF/ID)-(ID/EX)-(EX/MEM)-(MEM/WB) that takes the input and produces the outputs previously given to each component in each separate stage. (Fetching-Decoding-Execution-Memory-Write Back)
3. The addition/alteration of the Control Unit to propagate its wires and selection lines through the aforementioned stages, and through each of the 4 registers. This also required grouping such wires/lines based on their stage use.
4. The addition of various MUXs, Adders, Units, and path adjustments for the pipelined elements support (Forwarding, Branching, etc.…)
5. We also adjusted the memory to take 4 cycles for the sake of the cache taking 1 cycle.

**NEW Datapath:**

The **New** **Single Memory**:

As previously mentioned, we made our pipelined implantation functional on only one single memory instead of two (Instruction Memory and Data Memory). This is the code for such memory:

module memory(  
input clk,  
input MemRead,  
input MemWrite,  
input [2:0] funct3,  
input [31:0] addr,  
input [31:0] data\_in,  
output reg [31:0] data\_out  
);  
  
    reg [7:0] mem [0:255];  
     
    always @ (\*) begin  
        if(MemRead) begin  
        case (funct3)  
            3'b000:  //load byte  
            data\_out = $signed({mem[addr]});      
     
            3'b001: // load half word  
            data\_out = $signed({mem[addr], mem[addr+1]});    
     
            3'b010: // load word  
            data\_out = $signed({mem[addr],mem[addr+1],mem[addr+2],mem[addr+3]});  
     
            3'b100: // load byte unsigned  
            data\_out = mem[addr];                
     
            3'b101: //load half word unsigned  
            data\_out = {mem[addr], mem[addr+1]};  
     
            default: data\_out = 32'hZZZZZZZZ;  
        endcase  
        end else begin  
        data\_out = 32'hZZZZZZZZ;  
        end  
    end  
     
    always @ (posedge clk) begin  
        if(MemWrite) begin  
            case (funct3)  
                3'b000: //store byte  
                mem[addr] = data\_in[7:0];  
     
                3'b001: //store half word  
                {mem[addr], mem[addr+1]} = data\_in[15:0];      
     
                3'b010: //store word  
                {mem[addr],mem[addr+1],mem[addr+2],mem[addr+3]}= data\_in;      
     
                default: mem[addr] = mem[addr];  
            endcase  
        end else begin  
            mem[addr] = mem[addr];  
        end  
    end  
    initial begin  
   $readmemh("C://Users//mohamedwaleedelshemy//Downloads/test.mem", mem);  
    end  
  
endmodule

**Branch-Type Instructions:**

For the Branch-Type Instructions, we added the necessary Datapath changes (MUXs, Wires,…) and also added the following modification in the Control Module:

else if (inst == 5'b11000)begin  //branch  
            Branch = 1;  
            MemRead = 0;  
            ALUOp = 3'b001;  
            MemWrite = 0;  
            ALUsrc = 0;  
            RegWrite = 0;  
            auipc = 0;  
            jump = 0;  
            srcPC =2'b01;  
            pcload = 0;  
            flush = 0;

**Immediate Instructions:**

We remodeled our previous Single Cycle to further support immediate instructions—LUI and auipc here—and once again modified both the Datapath (added a MUX that takes the selection line of auipc for example) and the Control Module:

else if (inst == 5'b01101)begin  //LUI  
            Branch = 0;  
            MemRead = 0;  
            MemtoReg =0;  
            ALUOp = 3'b100;  
            MemWrite = 0;  
            ALUsrc = 1;  
            RegWrite = 1;  
            auipc = 0;  
            jump = 0;  
            srcPC =2'b00;  
            pcload = 0;  
            flush = 0;

else if (inst == 5'b00101)begin  //auipc  
            Branch = 0;  
            MemRead = 0;  
            MemtoReg =0;  
            ALUOp = 3'b000;  
            MemWrite = 0;  
            ALUsrc = 1;  
            RegWrite = 1;  
            auipc = 1;  
            jump = 0;  
            srcPC =2'b00;  
            pcload = 0;  
            flush = 0;  
        end

**Jump And Link Instructions:**

As specified in the project RV321 description, JAL and JALR were modeled in the Datapath and Control Module as follows:

else if (inst == 5'b11001)begin  // jalr  
            Branch = 0;  
            MemRead = 0;  
            MemtoReg =0;  
            ALUOp = 3'b000;  
            MemWrite = 0;  
            ALUsrc = 1;  
            RegWrite = 1;  
            auipc = 0;  
            jump = 1;  
            srcPC =2'b10;  
            pcload = 0;  
            flush = 1;  
        end    
        else if (inst == 5'b11011)begin  //jal  
            Branch = 0;  
            MemRead = 0;  
            MemtoReg =0;  
            ALUOp = 3'b000;  
            MemWrite = 0;  
            ALUsrc = 1;  
            RegWrite = 1;  
            auipc = 0;  
            jump = 1;  
            srcPC =2'b01;  
            pcload = 0;  
            flush = 1;

**BONUS Multiplication:**

// MULTIPLICATION

{3'b010, 3'b000, 1'b0, 1'b1}: ALUSelection = `ALU\_MUL;

{3'b010, 3'b001, 1'b0, 1'b1}: ALUSelection = `ALU\_MULH;

{3'b010, 3'b010, 1'b0, 1'b1}: ALUSelection = `ALU\_MULHSU;

{3'b010, 3'b011, 1'b0, 1'b1}: ALUSelection = `ALU\_MULHU;

{3'b010, 3'b100, 1'b0, 1'b1}: ALUSelection = `ALU\_DIV;

{3'b010, 3'b101, 1'b0, 1'b1}: ALUSelection = `ALU\_DIVU;

{3'b010, 3'b110, 1'b0, 1'b1}: ALUSelection = `ALU\_REM;

{3'b010, 3'b111, 1'b0, 1'b1}: ALUSelection = `ALU\_REMU;

**New Test Cases**:

