CS F342 COMPUTER ARCHITECTURE

ASSIGNMENT 1

Implement 5-stage pipelined processor in Verilog. This processor supports load (lw), store (sw), or immediate (ori), load upper immediate (lui), multiply (mul) and jump to register (jr) instructions only. The processor should implement forwarding to resolve data hazards. The processor has Reset, CLK as inputs and no outputs. The processor has instruction fetch unit, decode, reg read (with 32 32-bit registers), execution, memory and writeback units. The processor also contains four pipelined registers IF/ID, ID/EX, EX/MEM and MEM/WB. When reset is activated the PC, IF/ID, ID/EX, EX/MEM and MEM/WB registers are initialized to 0, the instruction memory and register file get loaded by predefined values.

When the instruction unit starts fetching the first instruction the pipelined registers contain unknown values. When the second instruction is being fetched in the IF unit, the IF/ID register will hold the instruction code for the first instruction. When the third instruction is being fetched by the IF unit, the IF/ID register contains the instruction code of the second instruction, the ID/EX register contains information related to the first instruction and so on. (Assume a 32-bit PC. Also Assume Address and Data size as 32-bit).

The instruction and its 32-bit instruction format are shown below:
lw destinationReg, offset [sourceReg] (Sign extends data specified in instruction field (15:0) to 32-bits, add it with register specified by register number in rs field and store the data corresponding to the memory location defined by the calculated address in the rt register. Opcode for lw is 100011).

op	rs	rt	offset
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	16-bits (15-0)

sw sourceReg, offset [destinationReg] (Signextends data specified in instruction field (15:0) to 32-bits, add it with register specified by register number in rs field. Opcode for sw is 101011).

op	rs	rt	offset
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	16-bits (15-0)

lui destinationReg, sourceReg, immediate (loads the highest 16 bits of the register rt with a constant (immediate value), and clears the lowest 16 bits to zeros. Opcode for lui is 001111.)

op	rs	rt	immediate
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	16-bits (15-0)

ori destinationReg, sourceReg, immediate (Sign extends data specified in instruction field (15:0) to 32-bits, or it with register specified by register number in rs field. And store the result in rt. Opcode for ori is 001110).

op	rs	rt	immediate
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	16-bits (15-0)

mul destinationReg, sourceReg, targetReg (Performs signed multiplication between targetReg and sourceReg. After this operation, result should be stored in destinationReg. Opcode for mul is 011010).

op	rs	rt	rd	shamt	funct
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	5-bits (15-11)	5-bits (10-6)	6-bits (5-0)

jr sourceReg (Jumps to an address stored in register rs. Opcode for j is 000010, take funct to bits as 001000).

op	rs	rt	rd	shamt	funct
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	5-bits (15-11)	5-bits (10-6)	6-bits (5-0)

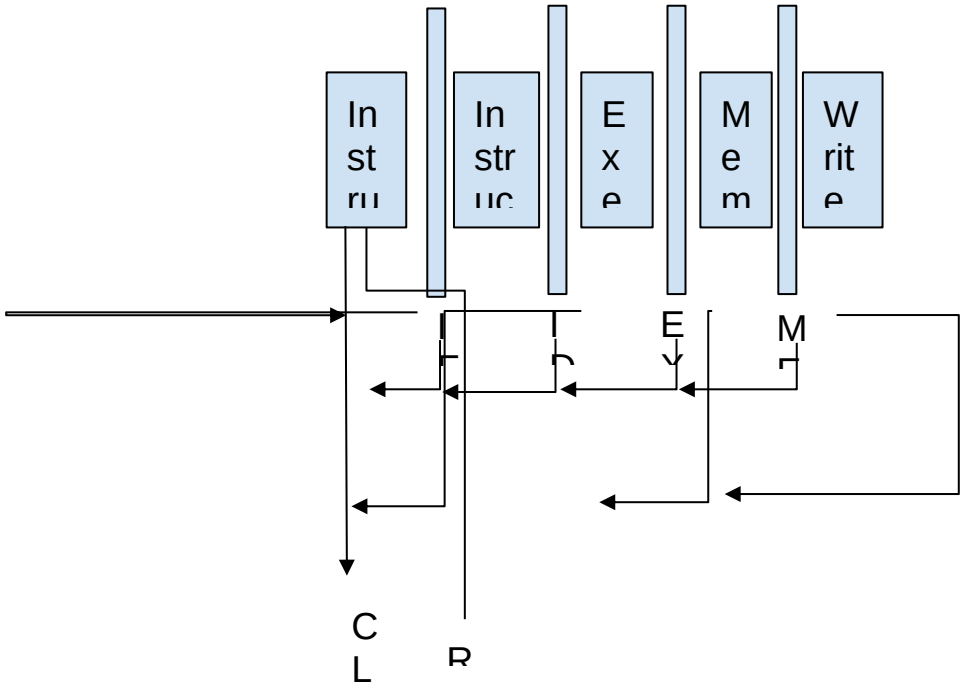
Assume the register file contains 32 registers (R0-R31) each register can hold 32-bit data. On reset, PC and all register file registers should get initialized to 0. Ensure r0 is always zero. Each location in DMEM has 8-bit data. So, to store a 32-bit value, you need 4 locations in the DMEM, stored in big-endian format. Also ensure that on reset, the instruction memory gets initialized with the following instructions, starting at address 0:

lw R1, R11, #12
lui R2, R8, #8

```
        mul R2, R1, R8
        jr R12
        mul R6, R6, R6
L1:     sw R4, 4[R5]
```

The above code should run correctly on the processor implementation. Ensure that you handle the data hazards present, if any.

A partial block level representation of the 5-stage pipelined processor is shown below. **Please note that for registerfile implementation, write should be on the positive edge and read should be on the negative edge of the clock.** Write operation depends on the control signal.



As part of the assignment three files should be submitted in a zipped folder.

1. PDF version of this Document with all the Questions below answered with file name as **IDNO_NAME.pdf.**
2. Design Verilog Files for all the Sub-modules (instruction fetch, Register file, forwarding unit).
3. Design Verilog file for the main processor.

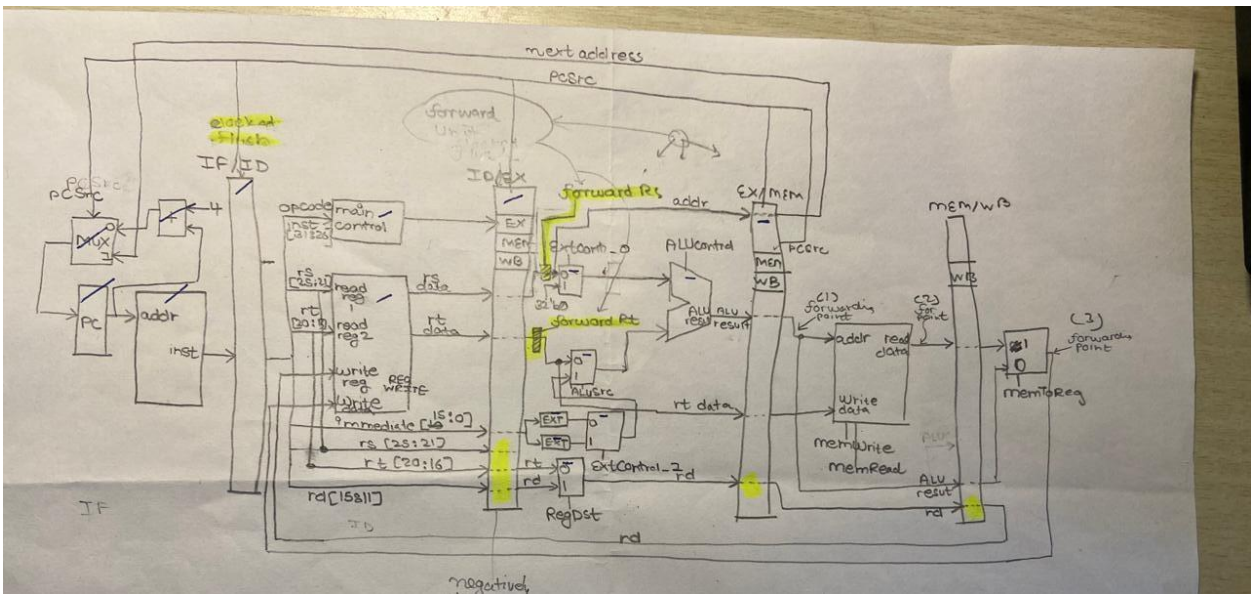
The name of the zipped folder should be in the format IDNO_NAME.zip

NAME: Shivansh Saroj Verma **ID No: 2022AAPS0140G**

Questions Related to Assignment

1. **Draw the complete Datapath and show control signals of the 5-stage pipelined processor. A sample Datapath for 5-stage pipelined MIPS processor has been discussed in class. A ppt named Assignmenthelp.ppt contains this 5-stage processor and is uploaded in CMS. You can modify this according to your specification.**

Answer:



2. **List the control signals used and also the values of control signals for different instructions in a tabular format as follows:**

Instructions	Control Signals							
	ALUSrc	ALUControl	ExtControl	RegDst	MemWrite	PCSrc	MemtoReg	RegWrite
lw	1	00	0	0	0	0	1	1
sw	1	00	0	0	1	0	0	0
lui	1	00	1	0	0	0	0	1

ori	1	01	0	0	0	0	0	1
jr	0	00	0	0	0	1	0	0
mul	0	10	0	1	0	0	0	1

Answer:

3. **In a program, there are 25% load instructions, 1/x of which are immediately followed by an instruction that uses a result, requiring a stall. 10% are stores. 50% are R-type. 10% are branch, 1/y of which are taken. 5% are jumps. What is the average CPI of this program? If the number of instructions are 10^9, and the clock cycle is 100 ps, how much time does a MIPS single cycle pipelined processor take to execute all instructions? Assume the processor always predicts branch not-taken.**
Where x, y, z are related to last 3 digits of your ID No.
If ID number: 20XXXXXXABCG, then x = (A % 8) + 1, y = ((B + 2) % 8) + 1, and z= ((C + 3) % 8) + 1.

Answer: 113.9 milliseconds

4. **Implement the Instruction Fetch block. Copy the <u>image</u> of Verilog code of the Instruction fetch block here**

Answer:

```
PC PC(
    .reset(reset),
    .clk(clk),
    .d(w0),
    .q(w1)
);

inst_mem inst_mem(
    .PC(w1),
    .reset(reset),
    .inst_code(w2)
);

PC_add_4 PC_add_4(
    .PC(w1),
    .out(w3)
);

PCSrc_mux PCSrc_mux(
    .in0(w3),
    .in1(w4),
    .PCSrc(w6),
    .out(w0)
);
```

```
module PC(
input reset,
input clk,
input [31:0] d,
output reg [31:0] q
);
    always@(posedge clk or negedge reset) begin
        if(!reset) q <= 0;
        else q <= d;
    end
endmodule
```

```
module PC_add_4(
input [31:0] PC,
output [31:0] out
);
    assign out = PC + 32'd4;
endmodule
```

```
module inst_mem(
input  [31:0] PC,
input  reset,
output [31:0] inst_code
);
    reg [7:0] Mem [35:0]; // byte addressable memory of 36 locations

    assign inst_code = {Mem[PC],Mem[PC+1],Mem[PC+2],Mem[PC+3]}; // Big Endian specification

    always@(negedge reset) begin
            // initializing the memory
            {Mem[0],Mem[1],Mem[2],Mem[3]}     = 32'h8d61000c; // lw r1,r11,#12
            {Mem[4],Mem[5],Mem[6],Mem[7]}     = 32'h3d020008; // lui r2,r8,#8
            {Mem[8],Mem[9],Mem[10],Mem[11]}   = 32'h68281000; // mul r2, r1, r8
            {Mem[12],Mem[13],Mem[14],Mem[15]} = 32'h09800008; // jr r12
            {Mem[16],Mem[17],Mem[18],Mem[19]} = 32'h68c63000; // mul r6,r6,r6
            {Mem[20],Mem[21],Mem[22],Mem[23]} = 32'haca40004; // sw r4,4[r5]
            {Mem[24],Mem[25],Mem[26],Mem[27]} = 32'h3bdfff00; // or r31, r30, ff00
            {Mem[28],Mem[29],Mem[30],Mem[31]} = 32'h00000000;
            {Mem[32],Mem[33],Mem[34],Mem[35]} = 32'h00000000;
    end
endmodule
```

```
module PCSrc_mux(
input [31:0] in0,
input [31:0] in1,
input PCSrc,
output [31:0] out
);
    assign out = (PCSrc)? in1: in0;
endmodule
```

5. **Implement the Instruction Decode block. Copy the <u>image</u> of Verilog code of the Instruction decode block here**

Answer: w5 is the 32 bit wire carrying the current instruction of the IF/ID block

```
reg_file reg_file(
    .Read_Reg_Num_1(w5[25:21]),
    .Read_Reg_Num_2(w5[20:16]),
    .Write_Reg_Num(w7),
    .Write_Data(w8),
    .Read_Data_1(w9),
    .Read_Data_2(w10),
    .RegWrite(w55),
    .reset(reset),
    .clk(clk)
);

main_control main_control(
    .opcode(w5[31:26]),
    .ALUControl(w12),
    .ALUSrc(w13),
    .RegDst(w14),
    .ExtControl(w15),
    .PCSrc(w16),
    .MemWrite(w17),
    .MemRead(w18),
    .MemtoReg(w19),
    .RegWrite(w20)
);
```

```
module reg_file(
input [4:0] Read_Reg_Num_1,
input [4:0] Read_Reg_Num_2,
input [4:0] Write_Reg_Num,
input [31:0] Write_Data,
output [31:0] Read_Data_1,
output [31:0] Read_Data_2,
input RegWrite,
input reset, // active low reset
input clk
);
    reg [31:0] RegMemory [31:0];
    integer i;

    // register read
    assign Read_Data_1 = RegMemory[Read_Reg_Num_1];
    assign Read_Data_2 = RegMemory[Read_Reg_Num_2];

    // register write
    always@(posedge clk) begin
        if(RegWrite && Write_Reg_Num != 5'b0)
            RegMemory[Write_Reg_Num] = Write_Data;
    end

    // reset
    always@(negedge reset) begin
            for(i=0; i<32; i=i+1)
                RegMemory[i] = 32'b0; // each regsiter
    end

endmodule
```

```
module main_control(
input [5:0] opcode,
output reg [1:0] ALUControl, //
output reg ALUSrc, //
output reg RegDst, //
output reg ExtControl, //
output reg PCSrc, //
output reg MemWrite, //
output reg MemRead, //
output reg MemtoReg, //
output reg RegWrite //
);
    always@(*) begin
        case(opcode)
        6'b100011: begin // lw
                        ALUControl <= 2'b00;
                        ALUSrc <= 1'b1;
                        RegDst <= 1'b0;
                        ExtControl <= 1'b0;
                        PCSrc <= 1'b0;
                        MemWrite <= 1'b0;
                        MemRead <= 1'b1;
                        MemtoReg <= 1'b1;
                        RegWrite <= 1'b1;
                    end
        6'b101011: begin // sw
                        ALUControl <= 2'b00;
                        ALUSrc <= 1'b1;
                        RegDst <= 1'b0;
```

6. **Determine the condition that can be used to detect data hazard?**

Answer:

1. (EX_MEM_opcode == 6'b001111 or EX_MEM_opcode == 6'b001110 or EX_MEM_opcode == 6'b011010 or EX_MEM_opcode == 6'b100011) and (EX_MEM_rd != 1'b0) and (EX_MEM_rd == ID_EX_rs)

2. If (1) is false and (MEM_WB_opcode == 6'b001111 or MEM_WB_opcode == 6'b001110 or MEM_WB_opcode == 6'b011010 or MEM_WB_opcode == 6'b100011) and (MEM_WB_rd != 1'b0) and (MEM_WB_rd == ID_EX_rs)

Same conditions for detecting hazard for rt

7. **Implement the Register File and copy the image of Verilog code of Register file unit here.**

Answer:

```
module reg_file(
input [4:0] Read_Reg_Num_1,
input [4:0] Read_Reg_Num_2,
input [4:0] Write_Reg_Num,
input [31:0] Write_Data,
output [31:0] Read_Data_1,
output [31:0] Read_Data_2,
input RegWrite,
input reset, // active low reset
input clk
);
    reg [31:0] RegMemory [31:0];
    integer i;

    // register read
    assign Read_Data_1 = RegMemory[Read_Reg_Num_1];
    assign Read_Data_2 = RegMemory[Read_Reg_Num_2];

    // register write
    always@(posedge clk) begin
        if(RegWrite && Write_Reg_Num != 5'b0)
            RegMemory[Write_Reg_Num] = Write_Data;
    end

    // reset
    always@(negedge reset) begin
            for(i=0; i<32; i=i+1)
                RegMemory[i] = 32'b0; // each regsiter h
    end

endmodule
```

8. **Implement the forwarding unit and copy the <u>image</u> of Verilog code of forwarding unit here.**

Answer:

```
module forwarding_unit(
input [5:0] EX_MEM_opcode,
input [5:0] MEM_WB_opcode,
input [4:0] EX_MEM_rd,
input [4:0] MEM_WB_rd,
input [4:0] ID_EX_rs,
input [4:0] ID_EX_rt,
output reg [1:0] forward_rs,
output reg [1:0] forward_rt
);
    always@(*) begin
        // EX stage hazard
        if( (EX_MEM_opcode == 6'b001111 || EX_MEM_opcode == 6'b001110 || EX_MEM_opcode == 6'b011010 || EX_MEM_opcode == 6'b100011) && (EX_MEM_rd != 1'b0) && (EX_MEM_rd == ID_EX_rs)) begin
            if(EX_MEM_opcode == 6'b100011)
                forward_rs = 2'b10;
            else
                forward_rs = 2'b01;
        end
        // MEM stage hazard (executes are EX fails)
        else if((MEM_WB_opcode == 6'b001111 || MEM_WB_opcode == 6'b001110 || MEM_WB_opcode == 6'b011010 || MEM_WB_opcode == 6'b100011) && (MEM_WB_rd != 1'b0) && (MEM_WB_rd == ID_EX_rs) )
            forward_rs = 2'b11;
        // default forwarding
        else
            forward_rs = 2'b00;
    end

    // rt forward
    always@(*) begin
        // EX stage hazard
        if( (EX_MEM_opcode == 6'b001111 || EX_MEM_opcode == 6'b001110 || EX_MEM_opcode == 6'b011010 || EX_MEM_opcode == 6'b100011) && (EX_MEM_rd != 1'b0) && (EX_MEM_rd == ID_EX_rt) ) begin
            if(EX_MEM_opcode == 6'b100011)
                forward_rt = 2'b10;
            else
                forward_rt = 2'b01;
        end
```

9. Implement a complete processor in Verilog (using all the Datapath blocks). Copy the <u>image</u> of Verilog code of the processor here. (Use comments to describe your Verilog implementation)

Answer:

```
timescale 1ns / 1ps

module pipeline_proc(
input clk,
input reset
);

    wire [31:0] w0,w1,w2,w3,w4,w5,w8,w9,w10,w31,w32;
    wire [31:0] w37,w38,w39,w40,w41,w42,w44,w45,w46;
    wire [31:0] w53,w56,w57,w60,w61,w62;
    wire [15:0] w33;
    wire [5:0] w21,w47,w59;
    wire [4:0] w7,w34,w35,w36,w43,w48,w58;
    wire [1:0] w12,w23;
    wire [1:0] sel_rs,sel_rt;
    wire w6,w11,w13,w14,w15,w16,w17,w18,w19,w20;
    wire w22,w24,w25,w26,w27,w28,w29,w30,w49,w50;
    wire w51,w52,w54,w55;
    wire zero;

    IF_ID IF_ID(
        .clk(clk),
        .reset(reset),
        .flush(w6),
        .inst_code_d(w2),
        .inst_code_q(w5)
    );
```

```
PC PC(
    .reset(reset),
    .clk(clk),
    .d(w0),
    .q(w1)
);

inst_mem inst_mem(
    .PC(w1),
    .reset(reset),
    .inst_code(w2)
);

PC_add_4 PC_add_4(
    .PC(w1),
    .out(w3)
);

PCSrc_mux PCSrc_mux(
    .in0(w3),
    .in1(w4),
    .PCSrc(w6),
    .out(w0)
);
```

```
reg_file reg_file(
    .Read_Reg_Num_1(w5[25:21]),
    .Read_Reg_Num_2(w5[20:16]),
    .Write_Reg_Num(w7),
    .Write_Data(w8),
    .Read_Data_1(w9),
    .Read_Data_2(w10),
    .RegWrite(w55),
    .reset(reset),
    .clk(clk)
);

main_control main_control(
    .opcode(w5[31:26]),
    .ALUControl(w12),
    .ALUSrc(w13),
    .RegDst(w14),
    .ExtControl(w15),
    .PCSrc(w16),
    .MemWrite(w17),
    .MemRead(w18),
    .MemtoReg(w19),
    .RegWrite(w20)
);
```

```
ID_EX ID_EX(
    .clk(clk), .reset(reset), .flush(w6),
    .ALUSrc_d(w13), .ALUControl_d(w12), .ExtControl_d(w15), .RegDst_d(w14), // EX
    .MemWrite_d(w17), .MemRead_d(w18), .PCSrc_d(w16),                       // MEM
    .MemtoReg_d(w19), .RegWrite_d(w20),                                     // WB
    .opcode_d(w5[31:26]),
    .rs_data_d(w9), .rt_data_d(w10),
    .imm_d(w5[15:0]),
    .rs_d(w5[25:21]), .rt_d(w5[20:16]), .rd_d(w5[15:11]),

    .opcode_q(w21),
    .ALUSrc_q(w22), .ALUControl_q(w23), .ExtControl_q(w24), .RegDst_q(w25),
    .MemWrite_q(w26), .MemRead_q(w27), .PCSrc_q(w28),
    .MemtoReg_q(w29), .RegWrite_q(w30),
    .rs_data_q(w31), .rt_data_q(w32),
    .imm_q(w33),
    .rs_q(w34), .rt_q(w35), .rd_q(w36)
);

sign_extend_32 sign_extend_32(
    .in(w33),
    .out(w41)
);

lui_extend_32 lui_extend_32(
    .in(w33),
    .out(w42)
);
```

```
ExtControl_mux ExtControl_0_mux(
    .in0(w61), // was w31
    .in1(32'b0),
    .ExtControl(w24),
    .out(w37)
);

ExtControl_mux ExtControl_1_mux(
    .in0(w41),
    .in1(w42),
    .ExtControl(w24),
    .out(w38)
);

ALUSrc_mux ALUSrc_mux(
    .in0(w62),
    .in1(w38),
    .ALUSrc(w22),
    .out(w39)
);

RegDst_mux RegDst_mux(
    .in0(w35),
    .in1(w36),
    .RegDst(w25),
    .out(w43)
);
```

```
ALU ALU(
    .in0(w37),
    .in1(w39),
    .control(w23),
    .out(w40),
    .zero(zero)
);

EX_MEM EX_MEM(
    .clk(clk), .reset(reset), .flush(w6),
    .addr_d(w61), .result_d(w40), .rt_data_d(w62),
    .opcode_d(w21),
    .rd_d(w43),
    .MemWrite_d(w26), .MemRead_d(w27), .PCSrc_d(w28),
    .MemtoReg_d(w29), .RegWrite_d(w30),

    .addr_q(w44), .result_q(w45), .rt_data_q(w46),
    .opcode_q(w47),
    .rd_q(w48),
    .MemWrite_q(w49), .MemRead_q(w50), .PCSrc_q(w6),
    .MemtoReg_q(w51), .RegWrite_q(w52)
);
```

```
    data_mem data_mem(
        .clk(clk),
        .reset(reset),
        .address(w45),
        .write_data(w46),
        .MemWrite(w49),
        .MemRead(w50),
        .read_data(w53)
    );

    MEM_WB MEM_WB(
        .clk(clk), .reset(reset),
        .MemtoReg_d(w51), .RegWrite_d(w52),
        .read_data_d(w53), .ALU_result_d(w45),
        .rd_d(w48),
        .opcode_d(w47),

        .MemtoReg_q(w54), .RegWrite_q(w55),
        .read_data_q(w56), .ALU_result_q(w57),
        .rd_q(w7),
        .opcode_q(w59)
    );

    MemtoReg_mux MemtoReg_mux(
        .in0(w57),
        .in1(w56),
        .MemtoReg(w54),
        .out(w8)
    );
```

```
    forwarding_unit forwarding_unit(
        .EX_MEM_opcode(w47),
        .MEM_WB_opcode(w59),
        .EX_MEM_rd(w48),
        .MEM_WB_rd(w7),
        .ID_EX_rs(w34),
        .ID_EX_rt(w35),
        .forward_rs(sel_rs),
        .forward_rt(sel_rt)
    );

    forward_mux rs_mux(
        .i0(w31),
        .i1(w45),
        .i2(w53),
        .i3(w8),
        .sel(sel_rs),
        .out(w61)
    );

    forward_mux rt_mux(
        .i0(w32),
        .i1(w45),
        .i2(w53),
        .i3(w8),
        .sel(sel_rt),
        .out(w62)
    );

    assign w4 = w44;
```

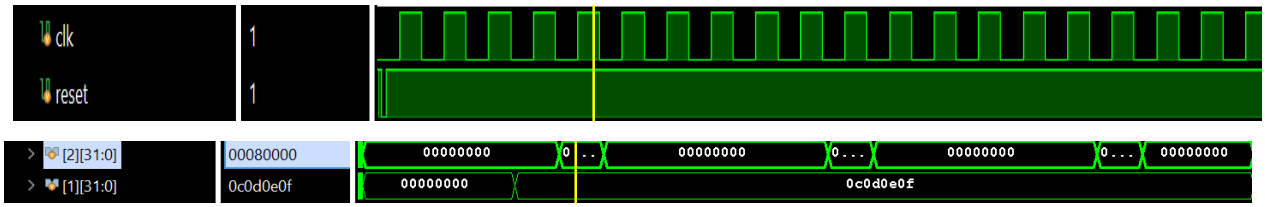
10. **Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

```
reg clk, reset;
pipeline_proc pipeline_proc(clk, reset);

initial begin
    clk = 0;
    reset = 1;
    #1 reset = 0;
    #1 reset = 1;
end

always #5 clk = ~clk;

initial #200 $finish;
endmodule
```

11. **Verify if the register file is getting updated according to the set of instructions (mentioned earlier).**

Copy verified **<u>Register file</u>** waveform here (show only the Registers that get updated, CLK, and RESET):



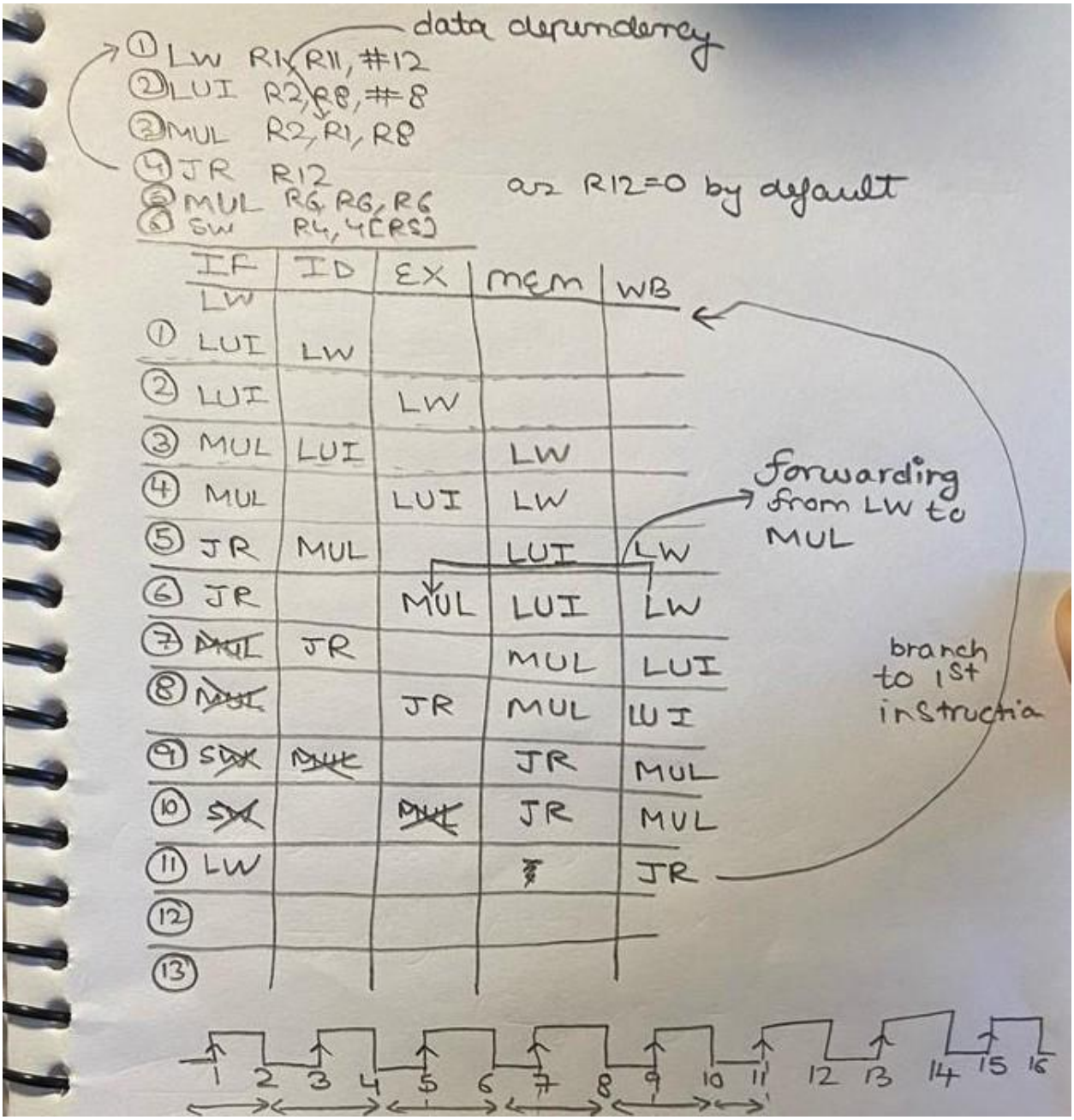
12. **What is the total number of cycles needed to issue the program given above on the pipelined MIPS Processor? What is the CPI of the program?**

Answer: The given program is an infinite loop so no amount of clock cycles given will terminate the program.

The CPI of the program is greater than 1 due to the branch statement that requires flushing.

13. Make a diagram showing the clock by clock execution of each instruction, indicating stalling, forwarding etc wherever necessary.

Answer:



14. Your design synthesisable? Which target FPGA was used for synthesis?

Answer: Yes. Kintex-7.

15. Provide the synthesis report in tabular form (resources consumed)?

Answer:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1534	0	0	41000	3.74
LUT as Logic	1534	0	0	41000	3.74
LUT as Memory	0	0	0	13400	0.00
Slice Registers	2869	0	0	82000	3.50
Register as Flip Flop	2869	0	0	82000	3.50
Register as Latch	0	0	0	82000	0.00
F7 Muxes	384	0	0	20500	1.87
F8 Muxes	64	0	0	10250	0.62

Unrelated Questions

What were the problems you faced during the implementation of the processor?

Answer: Managing a large amount of submodules was a bit of a challenge.

Did you implement the processor on your own? If you took help from someone whose help did you take? Which part of the design did you take help for?

Answer: Did it on my own. Referred to class slides for help.

Honor Code Declaration by student:

- My answers to the above questions are my own work.
- I have not shared the codes/answers written by me with any other students. (I might have helped clear doubts of other students).
- I have not copied other's code/answers to improve my results. (I might have got some doubts cleared from other students).

Name: Shivansh Saroj Verma **Date:** April 14, 2025
ID No.: 2022AAPS0140G