CS F342 COMPUTER ARCHITECTURE
ASSIGNMENT 1

Implement 6-stage pipelined processor in Verilog. This processor supports **load (lw), store (sw), jump (j), or (or), and and immediate (andi)** 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 five pipelined registers IF/ID, ID/RR, RR/EX, EX/MEM and MEM/WB. When reset is activated the PC, IF/ID, ID/RR, RR/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/RR 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 in rt from memory address [rs+offset]. 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 sourceReg1, offset [sourceReg2] (Sign extends data specified in instruction field (15:0) to 32-bits, add it with register specified by register number in rs field. store the data from the reg rt to memory address [rs+offset]. 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)

or destinationReg, sourceReg1, sourceReg2 (Perform or operation on the registers specified by registers specified by register numbers in rs and rt fields and save the result in the register specified by register specified by register number in rd field. Opcode for or is 000000 and function is 100101).

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)

andi destinationReg, sourceReg, immediate (Signextends data specified in instruction field (15:0) to 32-bits, and it with register specified by register number in rs field. And store the result in rt. Opcode for andi is 001100).

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

j target (Shift left by 2 the data specified in offset field (25:0) to 28-bits, and append the first 4 bits of PC+4. Opcode for j is 000010).

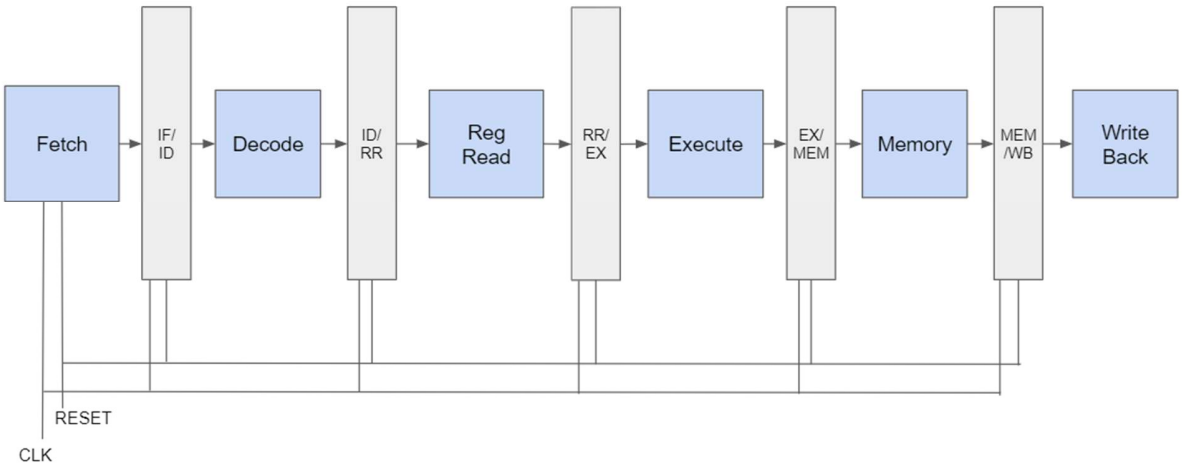
op	offset
6 bits (31-26)	26-bits (25-0)

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. DMEM[10] should have the value 8'd25 on reset. Also ensure that on reset, the instruction memory gets initialized with the following instructions, starting at address 0:

```
lw R1, R2, #10
sw R1, R3, #5
or R2, R5, R3
or R1, R6, R7
andi R1, R3, #10
```

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 6-stage pipelined processor is shown below. **Please note that for registerfile implementation, write should be on positive edge and read should be on negative edge of the clock.** Write operation depends on 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

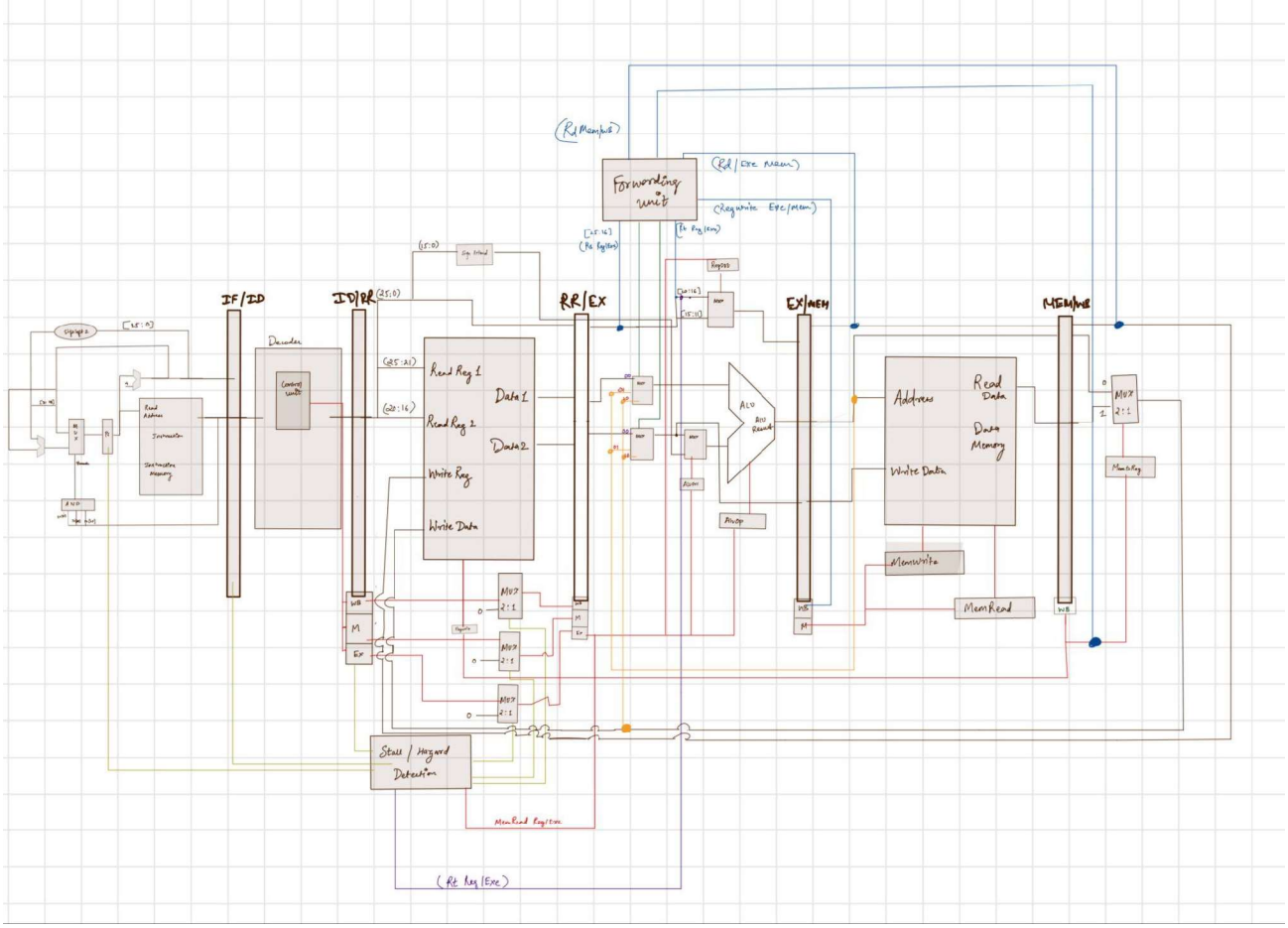
---

NAME: Sanket Lonkar **ID No: 2020B1A82099G**

Questions Related to Assignment

1. **Draw the complete Datapath and show control signals of the 6-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:**

   Answer:

Instructions	Control Signals								
	RegDst	AluSrc	Memto Reg	Regw rite	MemR ead	Mem Write	Branch	ALuop	
lw	0	1	1	1	1	0	0	0	0
sw	0	1	0	0	0	1	0	0	0
or	1	0	0	1	0	0	0	1	0

andi	0	1	0	1	0	0	0	0	1
j	0	0	0	0	0	0	1	1	1

3. **Calculate the average cpi for the following instructions.**

   lw R1, R2, #10
   sw R1, R3, #5
   or R2, R5, R3
   or R1, R6, R7
   andi R1, R3, #10

   Answer: Theoretically, CPI calculation
   Load word will take 6 cycles, store word will take (5stage + 1stall) and ,
   each or will take 6 cycles each also andi will take 6 cycles as this pipeliend processor the total cycles are 6+(no of inst -1) so , it should be
   cpi = (6+(5-1))/5 = 2;
   Observation :
   A total 11 cycles were seen in the waveform and no of instructions are five so it would be 11/5 = 2.2 CPI

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

Answer:

```
module instruction_fetch(
    input clk,
    input reset,
    input stall,
    output [31:0] Instruction_Code
    );
    reg [31:0] PC;
    wire [31:0] PCjump,PCnext1,PCnextmuxout;
    wire branch;
    instruction_memory im(reset,PC,Instruction_Code);
    assign PCjump = {PCnext1[31:28],Instruction_Code[25:0],2'b0} + PCnext1;
    assign PCnext1 = (reset&~stall&(PCnext1+4<37))?(PC+32'b100):32'b0;
//     assign branch = reset?(~Instruction_Code[31]&~Instruction_Code[29]&Instruction_Code[27]):0;
//    mux_21 m3(PCnext1,PCjump,PCnextmuxout,branch);
    always@(posedge clk)
    begin
        if(!stall)
            PC <= PCnextmuxout;
    end
    always@(negedge reset)
    begin
        if(reset==0)
        PC<=0;
    end
    assign branch = reset?(~Instruction_Code[31]&~Instruction_Code[29]&Instruction_Code[27]):0;
    mux_21 m3(PCnext1,PCjump,PCnextmuxout,branch);
endmodule
```

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

Answer:

```
module instruction_decode(
    input [31:0] instruction_code,
     output RegWriteD,
     output  MemtoRegD,
     output  MemWriteD,
     output [1:0] AlUControlD20,
     output  ALUSrcD,
     output  RegDstD,
     output JumpD,
     output MemRead
);


controlunit A(instruction_code[31:26],RegWriteD,MemtoRegD,MemWriteD,AlUControlD20,ALUSrcD,RegDstD,JumpD,MemRead);

endmodule
```

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

Answer: Execution unit hazard detection

If Exe/Mem RegWrite = 1 and Exe/Mem Rd != $zero and

Reg/Exe Rs = Exe/Mem Rd

OR

If Exe/Mem RegWrite = 1 and Exe/Mem Rd != $zero and

Reg/Exe Rt = Exe/Mem Rd

Memory Hazard detection

Mem/Writeback RegWrite =1 and Mem/Writeback Rd != $zero and

Reg/Exe Rs =Mem/Writeback Rd

And not (If Exe/Mem RegWrite = 1 and Exe/Mem Rd != $zero and

Reg/Exe Rs = Exe/Mem Rd)

Mem/Writeback RegWrite =1 and Mem/Writeback Rd != $zero and

Reg/Exe Rt =Mem/Writeback Rd

And not (If Exe/Mem RegWrite = 1 and Exe/Mem Rd != $zero and

Reg/Exe Rt = Exe/Mem Rd)

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

Answer:

```
//////////////////////////////////////////////////////////////////////////////////
module Register_File(
    input [4:0] Read_reg_num1,
    input [4:0] Read_reg_num2,
    input [4:0] Write_reg_num,
    input [31:0] Write_Dat,
    input reset,
    input clock,
    input RegWriteout,
    output [31:0] Read_Dat11,
    output [31:0] Read_Dat21
    );

    reg [31:0] Regmemory [31:0];
    reg [31:0] Read_Dat1;
    reg [31:0] Read_Dat2;
//    wire iszero;
//    assign iszero = ~(Write_reg_num[0]|Write_reg_num[1]|Write_reg_num[2]|Write_reg_num[3]|Write_reg_num[4]);
    assign Read_Dat11 = Read_Dat1;
    assign Read_Dat21 = Read_Dat2;



    always@(negedge reset) begin
    if(reset == 0)
    begin
        Regmemory[0] = 32'h0; Regmemory[1] = 32'h1; Regmemory[2] = 32'h2; Regmemory[3] = 32'h3;
        Regmemory[4] = 32'h4; Regmemory[5] = 32'h5; Regmemory[6] = 32'h6; Regmemory[7] = 32'h7;
        Regmemory[8] = 32'h8; Regmemory[9] = 32'h9; Regmemory[10] = 32'hA; Regmemory[11] = 32'h3;
        Regmemory[12] = 32'hC; Regmemory[13] = 32'hD; Regmemory[14] = 32'hE; Regmemory[15] = 32'hF;
        Regmemory[16] = 32'h0; Regmemory[17] = 32'h2; Regmemory[18] = 32'h3; Regmemory[19] = 32'h4;
        Regmemory[20] = 32'h5; Regmemory[21] = 32'h6; Regmemory[22] = 32'h7; Regmemory[23] = 32'h8;
        Regmemory[24] = 32'h0; Regmemory[25] = 32'h2; Regmemory[26] = 32'h3; Regmemory[27] = 32'h4;
        Regmemory[28] = 32'h5; Regmemory[29] = 32'h6; Regmemory[30] = 32'h4; Regmemory[31] = 32'h7;
     end
     end

     always@(negedge clock)
     begin
         Read_Dat1 = Regmemory[Read_reg_num1];
          Read_Dat2 = Regmemory[Read_reg_num2];
          end

     always @(posedge clock) begin
       if(RegWriteout == 1 & reset ==1)      // iszero != 1 &
           Regmemory[Write_reg_num] = Write_Dat;

     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 [4:0] Rd_exe_mem,
    input [4:0] Rd_mem_wb,
    input [4:0] Rs_reg_exe,
    input [4:0] Rt_reg_exe,
    input Regwrite_exe_mem,
    input Regwrite_mem_wb,
    output [1:0] Rs_cont,
    output [1:0] Rt_cont
);

reg [1:0] rs, rt;

always @(*) begin
    if (Rs_reg_exe == Rd_exe_mem & Regwrite_exe_mem == 1 & Rd_exe_mem != 5'b0)
        rs <= 2'b01;
    else if (Rs_reg_exe == Rd_mem_wb & Regwrite_mem_wb == 1 && Rd_mem_wb != 5'b0)
        rs <= 2'b10;
    else
        rs <= 2'b00;
end

always @(*) begin
    if (Rt_reg_exe == Rd_exe_mem & Regwrite_exe_mem == 1 & Rd_exe_mem != 5'b0)
        rt <= 2'b01;
    else if (Rt_reg_exe == Rd_mem_wb & Regwrite_mem_wb == 1 & Rd_mem_wb != 5'b0)
        rt <= 2'b10;
    else
        rt <= 2'b00;
end

assign Rs_cont = rs;
assign Rt_cont = rt;

endmodule
```

9. **Implement 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:

```
module Processor(
    input clock,
    input reset
    );

wire [31:0] instruction_code;
wire [31:0] instruction_code_out;
wire RegWriteD,MemtoRegD,MemWriteD,ALUSrcD,RegDstD,JumpD,MemRead;
wire [1:0] AlUControlD20;
wire stall;


wire [25:0] instruction_code_out1;
wire [1:0] AlUControlD20out;
wire RegWriteout,MemtoRegDout,MemWriteDout,ALUSrcDout,RegDstDout,JumpDout,MemReadout;

wire [31:0] Read_dat1;
wire [31:0] Read_dat2;
// make wire for RegWriteout4
// make wire for write_reg_num_out1
// make wire for Write_Dat
wire RegWriteout4;
wire [4:0] write_reg_num_out1;
wire [31:0] Write_Dat;


wire [31:0] immediateData_in;
wire[31:0] immediateData_out;
wire[31:0] RS_out;
wire[31:0] Rt_out;

wire [1:0] AlUControlD20out1;
wire RegWriteout1,MemtoRegDout1,MemWriteDout1,ALUSrcDout1,RegDstDout1,JumpDout1,MemReadout1;

wire [25:0] instruction_code_out2;
wire [1:0] AlUControlD20out2;
wire RegWriteout2,MemtoRegDout2,MemWriteDout2,ALUSrcDout2,RegDstDout2,JumpDout2,MemReadout2;
wire [31:0] ALU_result;
wire [31:0] ALU_result_out,Rt_out1;
wire [1:0] Rs_cont,Rt_cont;
wire [31:0] muxout1,muxout2,ALUin2;
wire [4:0] write_reg_num;


wire [4:0] write_reg_num_out;
wire RegWriteout3,MemWriteDout3,MemReadout3,MemtoRegDout3;

wire [31:0] Mem_Read_dat;

wire [31:0] ALU_result_out1,Mem_Read_dat_out;
wire MemtoRegDout4;

wire [31:0] PCnext_out1;
```

```
    wire [31:0] PCnext_out1;

    wire [5:0] opcode;
    wire [5:0] opcodeidrr;




    // fetch,regbank if_id,instruction decode instance
    instruction_fetch A(clock,reset,stall,instruction_code);

    // instr fetch and decode regiter bank instance
    IF_ID B (clock,reset,stall,instruction_code,instruction_code_out,opcode);
    // instruction decode instacne
    instruction_decode C(instruction_code_out,RegWriteD,MemtoRegD,MemWriteD,ALUControlD20,ALUSrcD,RegDstD,JumpD,MemRead);
    ///  decode and register read regiter bank instance
    ID_RR D(opcode,clock,reset,stall,instruction_code_out,RegWriteD,MemtoRegD,MemWriteD,ALUControlD20,ALUSrcD,RegDstD,JumpD,MemRead,instruction_code_out1,RegWriteout,MemtoRegDout,MemWriteDout,ALUControlD20out,ALUSrcDout,RegDstDout,JumpDout,MemReadout,opcodeidrr
    // register file instance
    Register_File E(instruction_code_out1[25:21],instruction_code_out1[20:16],write_reg_num_out1,Write_Dat,reset,clock,RegWriteout4,Read_dat1,Read_dat2);

    // sign extending the immediate value in case of load , store word
    assign immediateData_in[31:16] = {16{instruction_code_out1[15]}};
    assign immediateData_in[15:0] = instruction_code_out1[15:0];

    //  selecting the control signal , 0 if to stall , 1 send the original control signal to rr_ex register bank
    mux_2singleR(RegWriteout,1'b0,RegWriteout1,stall);
    mux_2singleS(ALUSrcDout,1'b0,ALUSrcDout1,stall);
    mux_2singleT(RegDstDout,1'b0,RegDstDout1,stall);
    mux_2singleU(ALUControlD20out[0],1'b0,ALUControlD20out1[0],stall);
    mux_2singleV(ALUControlD20out[1],1'b0,ALUControlD20out1[1],stall);
    mux_2singleW(MemWriteDout,1'b0,MemWriteDout1,stall);
    mux_2singleX(MemReadout,1'b0,MemReadout1,stall);
    mux_2singleY(MemtoRegDout,1'b0,MemtoRegDout1,stall);


    // rr_ex regiter bank instance
    RR_EX F(Read_dat1,Read_dat2,clock,reset,stall,immediateData_in,instruction_code_out1,RegWriteout1,ALUSrcDout1,RegDstDout1,ALUControlD20out1,MemWriteDout1,MemReadout1,MemtoRegDout1,RS_out,Rt_out,immediateData_out,MemReadout2,MemtoRegDout2,instruction_code_ou

    //  4 -1 mux (32 bits) selecting between forwarded values and rs_out / rt_out
    mux_41 G(RS_out,ALU_result_out,Write_Dat,32'b0,muxout1,Rs_cont);
    mux_41 H(Rt_out,ALU_result_out,Write_Dat,32'b0,muxout2,Rt_cont);

    // 2-1 mux for selecting 2 input to alu (between immeadiate value and normal register read value)
    mux_21 I(muxout2,immediateData_out,ALUin2,ALUSrcDout2);

    // 2-1 mux (5 bits) to select back rt ot rd in reister file to  write back in reister file
    mux_215bits J(instruction_code_out2[20:16],instruction_code_out2[15:11],write_reg_num,RegDstDout2);

    // Alu instance
    ALU K(muxout1,ALUin2,ALUControlD20out2,ALU_result);

    // ex-mem register bank instance
```

```
// rr_ex regiter bank instance
RR_EX F(Read_dat1,Read_dat2,clock,reset,stall,immediateData_in,instruction_code_out1,RegWriteout1,ALUSrcDout1,RegDstDout1,ALUControlD20out1,MemWriteDout1,MemReadout1,MemtoRegDout1,RS_out,Rt_out,immediateData_out,MemReadout2,MemtoRegDout2,instruction_code_ou

//  4 -1 mux (32 bits) selecting between forwarded values and rs_out / rt_out
mux_41 G(RS_out,ALU_result_out,Write_Dat,32'b0,muxout1,Rs_cont);
mux_41 H(Rt_out,ALU_result_out,Write_Dat,32'b0,muxout2,Rt_cont);

// 2-1 mux for selecting 2 input to alu (between immeadiate value and normal register read value)
mux_21 I(muxout2,immediateData_out,ALUin2,ALUSrcDout2);

// 2-1 mux (5 bits) to select back rt ot rd in reister file to  write back in reister file
mux_215bits J(instruction_code_out2[20:16],instruction_code_out2[15:11],write_reg_num,RegDstDout2);

// Alu instance
ALU K(muxout1,ALUin2,ALUControlD20out2,ALU_result);

// ex-mem register bank instance
EX_MEM L(clock,reset,write_reg_num,ALU_result,muxout2,MemWriteDout2,RegWriteout2,MemReadout2,MemtoRegDout2,ALU_result_out,Rt_out1,RegWriteout3,MemWriteDout3,MemReadout3,MemtoRegDout3,write_reg_num_out);

// mem instance
MEM M(ALU_result_out,Rt_out1,reset,MemWriteDout3,MemReadout3,Mem_Read_dat);
// mem-wb register instacne
MEM_WB N(clock,reset,ALU_result_out,write_reg_num_out,Mem_Read_dat,MemtoRegDout3,RegWriteout3,write_reg_num_out1,ALU_result_out1,Mem_Read_dat_out,MemtoRegDout4,RegWriteout4);

//  write back mux instacne
mux_21 O(ALU_result_out1,Mem_Read_dat_out,Write_Dat,MemtoRegDout4);

// forwarding unit
forwarding_unit P(write_reg_num_out,write_reg_num_out1,instruction_code_out2[25:21],instruction_code_out2[20:16],RegWriteout3,RegWriteout4,Rs_cont,Rt_cont);
// stalling unit
stalling_unit Q( instruction_code_out1[25:21],instruction_code_out1[20:16],instruction_code_out2[20:16],MemReadout2,stall);

endmodule
```

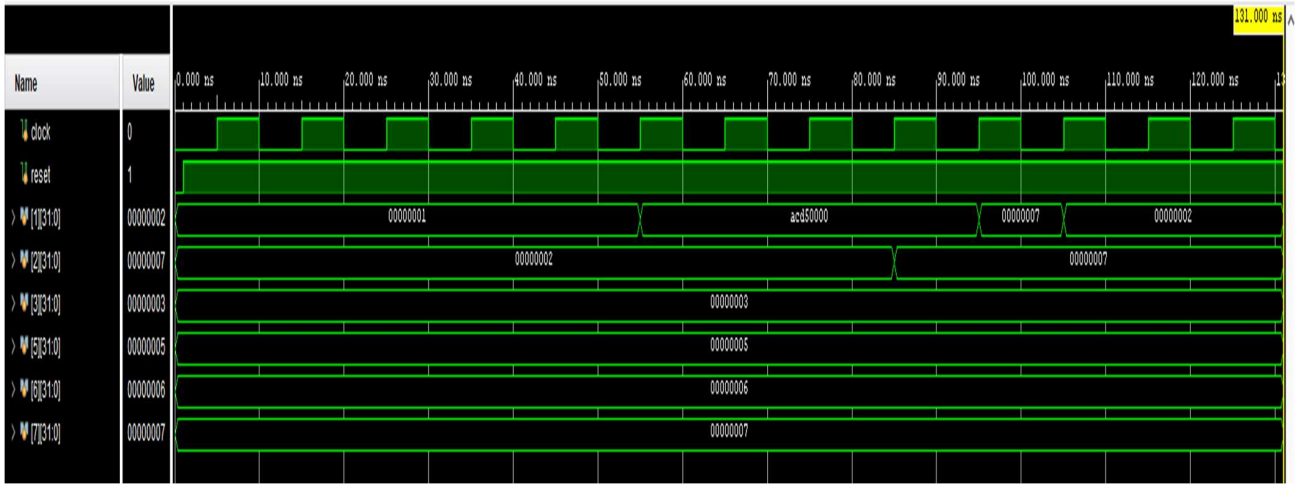
10. Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.

Answer:

```
module Processor_test();

reg clock =0;
reg reset =0;

Processor dut (clock,reset);

//always
//begin

//clock = ~clock; #10;
//end
//initial
//begin
//reset = 1; #5;
//reset =0; #200;
//reset =1;

//end
initial begin
    reset=1'b0; #1;
    reset=1'b1; #130;
    $stop;
end

initial begin
        clock=1'b0;
        forever begin
            #5 clock=~clock;
            end
        end
endmodule
```

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

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

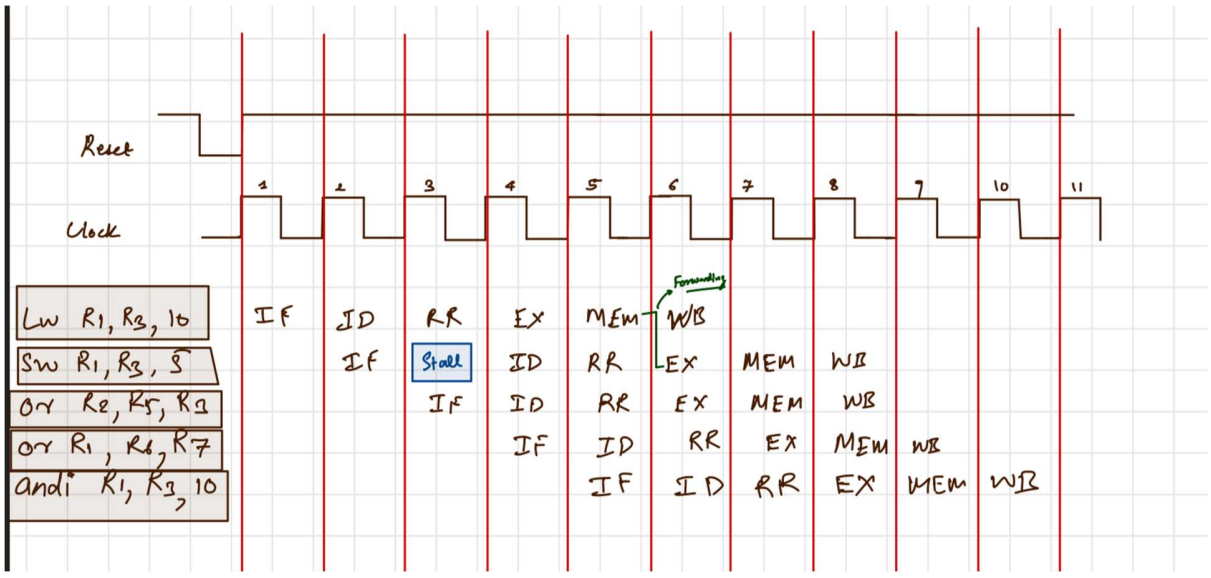


12. What are the total number of cycles needed to issue the program given above on the pipelined MIPS Processor? What is the CPI of the program?

Answer: 11 cycles and 5 instructions so CPI = 11/5 = 2.2

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 , Zynq-7000,xc7z020clg484-1

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

Answer:

```
Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2021.2 (win64) Build 3367213 Tue Oct 19 02:48:09 MDT 2021
| Date         : Sun Apr 14 20:23:49 2024
| Host         : Ace running 64-bit major release  (build 9200)
| Command      : report_utilization -file Processor_utilization_synth.rpt -pb Processor_utilization_synth.pb
| Design       : Processor
| Device       : xc7z020clg484-1
| Speed File   : -1
| Design State : Synthesized
----------------------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking
7. Specific Feature
8. Primitives
9. Black Boxes
10. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs              |    0 |     0 |          0 |     53200 |  0.00 |
|   LUT as Logic          |    0 |     0 |          0 |     53200 |  0.00 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |    0 |     0 |          0 |    106400 |  0.00 |
|   Register as Flip Flop |    0 |     0 |          0 |    106400 |  0.00 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice	0	0	0	13300	0.00
SLICEL	0	0			
SLICEM	0	0			
LUT as Logic	0	0	0	53200	0.00
LUT as Memory	0	0	0	17400	0.00
LUT as Distributed RAM	0	0			
LUT as Shift Register	0	0			
Slice Registers	0	0	0	106400	0.00
Register driven from within the Slice	0				
Register driven from outside the Slice	0				
Unique Control Sets	0		0	13300	0.00

* * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for more information regarding control sets.

3. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	140	0.00
RAMB36/FIFO*	0	0	0	140	0.00
RAMB18	0	0	0	280	0.00

3. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	0	0	0	140	0.00
RAMB36/FIFO*	0	0	0	140	0.00
RAMB18	0	0	0	280	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	220	0.00

5. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	0	0	0	200	0.00
Bonded IPADs	0	0	0	2	0.00
Bonded IOPADs	0	0	0	130	0.00
PHY_CONTROL	0	0	0	4	0.00
PHASER_REF	0	0	0	4	0.00
OUT_FIFO	0	0	0	16	0.00
IN_FIFO	0	0	0	16	0.00
IDELAYCTRL	0	0	0	4	0.00
IBUFDS	0	0	0	192	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	200	0.00
ILOGIC	0	0	0	200	0.00
OLOGIC	0	0	0	200	0.00

```
6. Clocking
-----------

+------------+------+-------+------------+-----------+-------+
|  Site Type | Used | Fixed | Prohibited | Available | Util% |
+------------+------+-------+------------+-----------+-------+
| BUFGCTRL   |    0 |     0 |          0 |        32 |  0.00 |
| BUFIO      |    0 |     0 |          0 |        16 |  0.00 |
| MMCME2_ADV |    0 |     0 |          0 |         4 |  0.00 |
| PLLE2_ADV  |    0 |     0 |          0 |         4 |  0.00 |
| BUFMRCE    |    0 |     0 |          0 |         8 |  0.00 |
| BUFHCE     |    0 |     0 |          0 |        72 |  0.00 |
| BUFR       |    0 |     0 |          0 |        16 |  0.00 |
+------------+------+-------+------------+-----------+-------+


7. Specific Feature
-------------------

+-------------+------+-------+------------+-----------+-------+
|  Site Type  | Used | Fixed | Prohibited | Available | Util% |
+-------------+------+-------+------------+-----------+-------+
| BSCANE2     |    0 |     0 |          0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |          0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |          0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |          0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |          0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |          0 |         2 |  0.00 |
| STARTUPE2   |    0 |     0 |          0 |         1 |  0.00 |
| XADC        |    0 |     0 |          0 |         1 |  0.00 |
+-------------+------+-------+------------+-----------+-------+


8. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+


9. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


10. Instantiated Netlists
-------------------------
```

Unrelated Questions

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

Answer: I faced problem in stalling and forwarding , store after load created problem for me but sorted it.

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:

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: Sanket Lonkar **Date:** 14/04/2024
ID No.: 2020B1A82099G