CS F342 COMPUTER ARCHITECTURE

ASSIGNMENT 1

Implement a 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 IF unit is fetching the third instruction, 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, the 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).

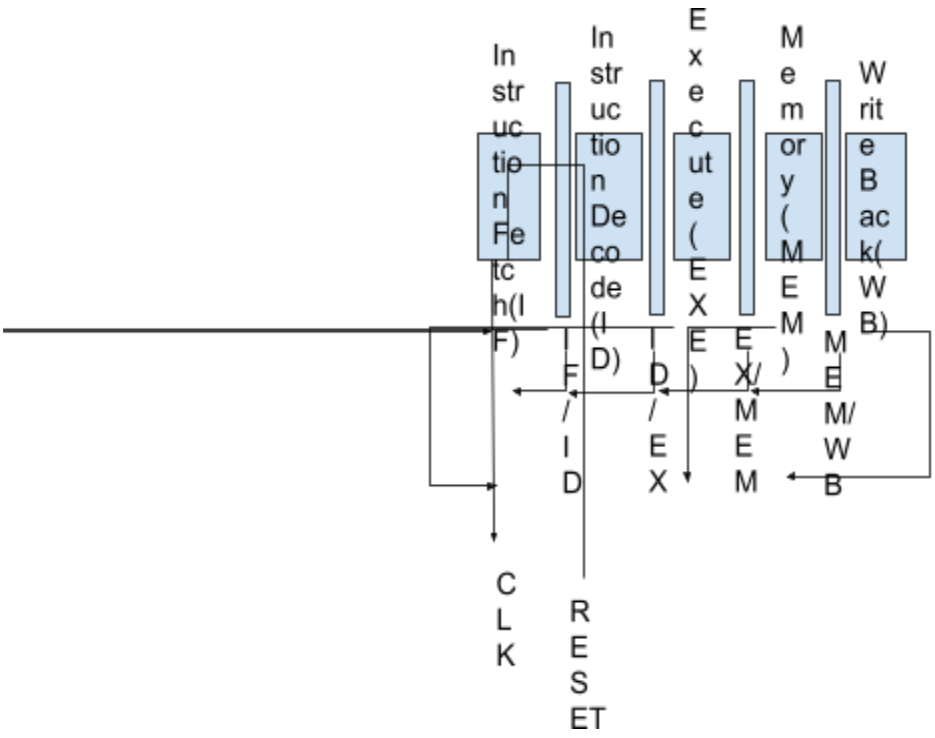
op	rs	rt	rd	shamt	funct
6 bits (31-26)	5-bits (25-21)	5-bits (20-16)	5-bits (15-11)	5-bits (10-6)	6-bits (5-0)

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 register file 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.

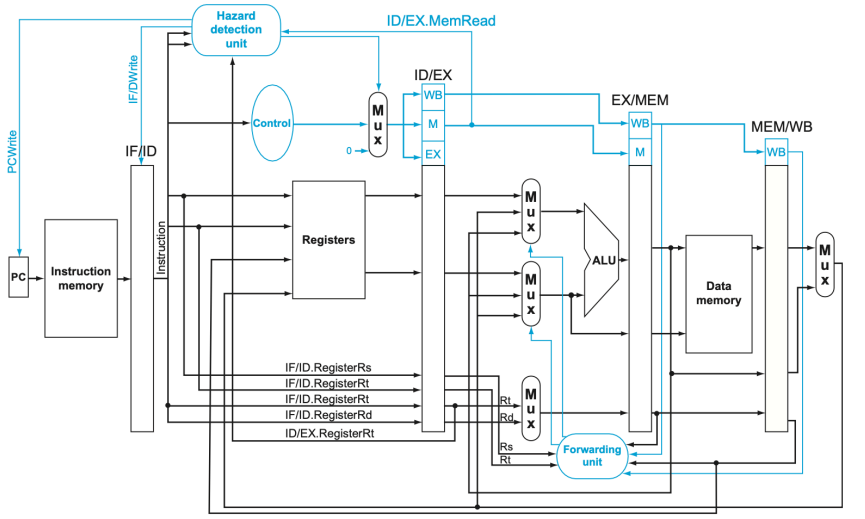
   The name of the zipped folder should be in the format IDNO_NAME.zip

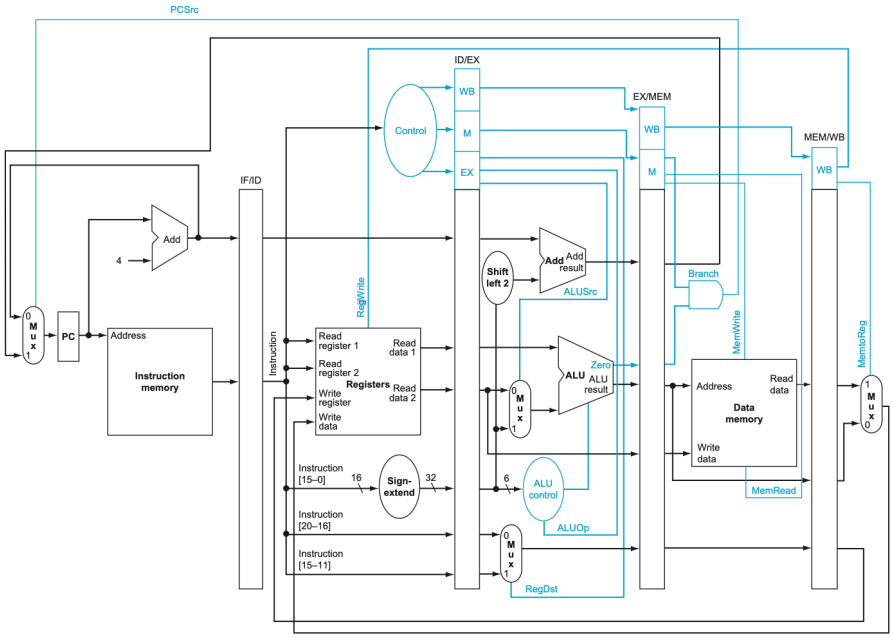
NAME: ABHIMANYU MAGAPU ID No: 2021A3PS2516G

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:

Answer:

Instruction	Control Signals							
	RegDst	AluSrc	Branch	Memread	Memwrite	RegWrite	Memtoreg	Aluop
lw	0	1	0	1	0	1	1	01
sw	0	1	0	0	1	0	0	01
lui	0	1	0	0	0	1	0	11
ori	0	1	0	0	0	1	0	10
jr	0	0	1	0	0	0	0	00
mul	1	0	0	0	0	1	0	00

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 branches, 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 the 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: A = 5, B = 1, C = 6. x= (5%8) + 1 = 6, y = ((1+2)%8) + 1) = 4,
z = ((6+3)%8)+1 = 2.

LW - cpi = 0.25*(⅝*1 + ⅛*2), SW = 0.1*1, Rtype = 0.5*1, Branch = 0.1*(¼*3+¾*1),
Jump = 0.05(3)
Total Cpi = 1.191
Time = Cpi * Clock TP * Number of Instr = 0.119s

4. Implement the Instruction Fetch block. Copy the image of Verilog code of the Instruction fetch block here

Answer:

```
24
25      module instr_fetch3(
26          output [31:0] instruction_code,
27          output [31:0] pipl_pc,
28          input [31:0] pc_pip,
29          input pc_sel,
30          input clock,
31          input reset,
32          input stall);
33          reg [31:0] pc;
34          wire [31:0] next_pc, muxed_pc;
35      always @(posedge clock) begin
36        if (!reset) begin
37          pc <= 32'b0;
38        end else if (!stall) begin
39          if (pc_sel) begin
40            pc <= pc_pip;
41          end else begin
42            pc <= muxed_pc;
43          end
44        end
45      end
46
47
48          assign pipl_pc = pc + 4;
49          instr_mem3 m1(instruction_code, reset, pc);
50          mux1x2 i1(next_pc, pipl_pc, pc_pip, pc_sel);
51          mux1x2 i2(muxed_pc, next_pc, pc, stall);
52      endmodule
```

5. Implement the Instruction Decode block. Copy the image of the Verilog code of the Instruction decode block here

Answer:

```
61      //decode
62      control_unit2 c1(reg_write, alusrc, reg_dst, branch, memwrite, memread, memtoreg, aluop, out_if[31:0]);
63      reg_file2 m2(read_data1, read_data2, write_data , out_mem[4:0], out_if[25:21], out_if[20:16], out_mem[70] , clock
64      pp_reg2 #(.WIDTH(152)) id_ex(out_id,
65        {out_if[25:21], reg_write, alusrc, reg_dst, branch, memwrite, memread, memtoreg, aluop,
66         out_if[63:32], read_data1, read_data2, {{16{out_if[15]}}, out_if[15:0]}, out_if[20:16], out_if[15:11]},
67          clock, reset, stall, out_ex[105]);
68      //rs, control signals, pc, rd1, rd2, immdata signextended, rt, rd
```

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

Answer:

```
stall = id_ex_memread && ((id_ex_rt == if_id_rs) || (id_ex_rt == if_id_rt));
```

```
// Default no forwarding
forward_rs = 2'b00;
forward_rt = 2'b00;

// EX Hazard Detection for Rs
if (ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rs)) begin
    forward_rs = 2'b01;
end

// MEM Hazard Detection for Rs
if (mem_wb_regwrite && (mem_wb_rd != 0) && (mem_wb_rd == id_ex_rs) &&
    !(ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rs))) begin
    forward_rs = 2'b10;
end

// EX Hazard Detection for Rt
if (ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rt)) begin
    forward_rt = 2'b01;
end

// MEM Hazard Detection for Rt
if (mem_wb_regwrite && (mem_wb_rd != 0) && (mem_wb_rd == id_ex_rt) &&
    !(ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rt))) begin
    forward_rt = 2'b10;
end
```

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

Answer:

```
23  module reg_file2(rd_data1, rd_data2, wr_data, wr_reg_num, rd_reg_num1, rd_reg_num2, reg_write, clock, reset);
24  input [4:0] rd_reg_num1, rd_reg_num2, wr_reg_num;
25  input clock, reg_write, reset;
26  input [31:0] wr_data;
27  output reg [31:0] rd_data1, rd_data2;
28
29  reg [31:0] regmemory [31:0];
30  integer i;
31  always @(*) begin
32      if(clock)begin
33          if(reg_write)begin
34              if ( wr_reg_num == 0 ) begin
35                  regmemory[wr_reg_num] = 0; end
36              else regmemory[wr_reg_num] = wr_data;
37          end
38          else regmemory[wr_reg_num] = regmemory[wr_reg_num];
39       end
40  end
41  always @(negedge clock) begin
42      rd_data1 = regmemory[rd_reg_num1];
43      rd_data2 = regmemory[rd_reg_num2];
44  end
45  always @ (clock, reset)begin
46      if (!reset) begin
47          for (i = 0; i<32; i = i+1) begin
48              regmemory[i] <= i;
49              end
50      end
51  end
52  endmodule
```

8. Implement the forwarding unit and copy the image of Verilog code of forwarding unit here.

Answer:

```
module forwarding_unit(
    output reg [1:0] forward_rs, // ForwardRs
    output reg [1:0] forward_rt,  // ForwardRt
    input mem_wb_regwrite,      // MEM/WB.RegWrite
    input [4:0] mem_wb_rd,      // MEM/WB.RegisterRd
    input ex_mem_regwrite,      // EX/MEM.RegWrite
    input [4:0] ex_mem_rd,      // EX/MEM.RegisterRd
    input [4:0] id_ex_rs,       // ID/EX.RegisterRs
    input [4:0] id_ex_rt       // ID/EX.RegisterRt
);
always @(*) begin
    // Default no forwarding
    forward_rs = 2'b00;
    forward_rt = 2'b00;
    // EX Hazard Detection for Rs
    if (ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rs)) begin
        forward_rs = 2'b01;
    end
    // MEM Hazard Detection for Rs
    if (mem_wb_regwrite && (mem_wb_rd != 0) && (mem_wb_rd == id_ex_rs) &&
        !(ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rs))) begin
        forward_rs = 2'b10;
    end
    // EX Hazard Detection for Rt
    if (ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rt)) begin
        forward_rt = 2'b01;
    end
    // MEM Hazard Detection for Rt
    if (mem_wb_regwrite && (mem_wb_rd != 0) && (mem_wb_rd == id_ex_rt) &&
        !(ex_mem_regwrite && (ex_mem_rd != 0) && (ex_mem_rd == id_ex_rt))) begin
        forward_rt = 2'b10;
    end
```

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

```
23      module fproc(clock, reset);
24      input clock, reset;
25
26      wire [31:0] instruction, pipl_pc;
27
28      wire [31:0] read_data1, read_data2;
29      wire reg_write, alusrc, reg_dst, branch, memwrite, memread, memtoreg;
30      wire [1:0] aluop;
31
32      wire [31:0] alu_src2, alu_out;
33      wire [31:0] jump_add, imdata;
34      wire [3:0] alusel;
35      wire zeroflag;
36      wire [4:0] write_reg;
37
38      wire [31:0] data_mem_read_data;
39
40      wire [63:0] out_if;
41      wire [151:0] out_id;
42      wire [106:0] out_ex;
43      wire [70:0] out_mem;
44
45      wire [31:0] write_data;
46
47      wire [1:0] forward_rs, forward_rt;
48      wire [31:0] alu_forward_1, alu_forward_2;
49
50      wire stall;
```

```
56      //stall
57      stall_unit m7(stall, out_id[141], out_id[9:5], out_if[25:21], out_if[20:16]);
58
59
60      // if
61      instr_fetch3 m1(instruction, pipl_pc, out_ex[101:70], out_ex[105], clock, reset, stall);
62      pp_reg3 #(.WIDTH(64)) if_id(out_if, {pipl_pc, instruction}, clock, reset, stall, out_ex[105]); //pc, instruction
63
64      //decode
65      control_unit2 c1(reg_write, alusrc, reg_dst, branch, memwrite, memread, memtoreg, aluop, out_if[31:26]);
66      reg_file2 m2(read_data1, read_data2, write_data , out_mem[4:0], out_if[25:21], out_if[20:16], out_mem[70] , clock, reset);
67      pp_reg2 #(.WIDTH(152)) id_ex(out_id,
68       {out_if[25:21], reg_write, alusrc, reg_dst, branch, memwrite, memread, memtoreg, aluop,
69        out_if[63:32], read_data1, read_data2, {{16{out_if[15]}}, out_if[15:0]}, out_if[20:16], out_if[15:11]},
70         clock, reset, stall, out_ex[105]);
71      //rs, control signals, pc, rd1, rd2, immdata signextended, rt, rd
72
73
74      //execute
75      assign imdata = alu_forward_1  << 2;
76      assign jump_add = out_id[137:106] + imdata;
77      mux1x2_5 t1(write_reg, out_id[9:5], out_id[4:0], out_id[144]);
78      mux1x2 t2(alu_src2, alu_forward_2, out_id[41:10], out_id[145]);
79      alu_control2 m3(alusel,out_id[139:138],out_id[15:10]);
80      alu m4(zeroflag, alu_out, alu_forward_1, alu_src2, alusel);
81      pp_reg4 #(.WIDTH(107)) ex_mem(out_ex,
82      {out_id[146], out_id[143:140], jump_add, zeroflag, alu_out, out_id[73:42], write_reg}, clock, reset, out_ex[105]);
83      // regwrite, branch, memwrite, memread, memtoreg, jump add, zerod, alu out, rd2,  write reg
```

```
    //mem
    data_mem m5(data_mem_read_data, out_ex[36:5], out_ex[68:37], out_ex[103], out_ex[104], reset);
    pp_reg #(.WIDTH(71)) mem_wb(out_mem,
    {out_ex[106], out_ex[102], data_mem_read_data, out_ex[68:37], out_ex[4:0]}, clock, reset);
    // regwrite, memtoreg, datamem output, alu out, write reg

    //wb
    mux1x2 t3(write_data, out_mem[36:5], out_mem[68:37], out_mem[69]);

    //forwarding
    forwarding_unit m6(forward_rs, forward_rt, out_mem[70], out_mem[4:0], out_ex[106], out_ex[4:0], out_id[151:147], out_id[9:5]);
    mux2x4 l1(alu_forward_1, out_id[105:74], out_ex[68:37], out_mem[68:37], {32{1'b0}}, forward_rs);
    mux2x4 l2(alu_forward_2, out_id[73:42], out_ex[68:37], out_mem[68:37], {32{1'b0}}, forward_rt);




    endmodule
```

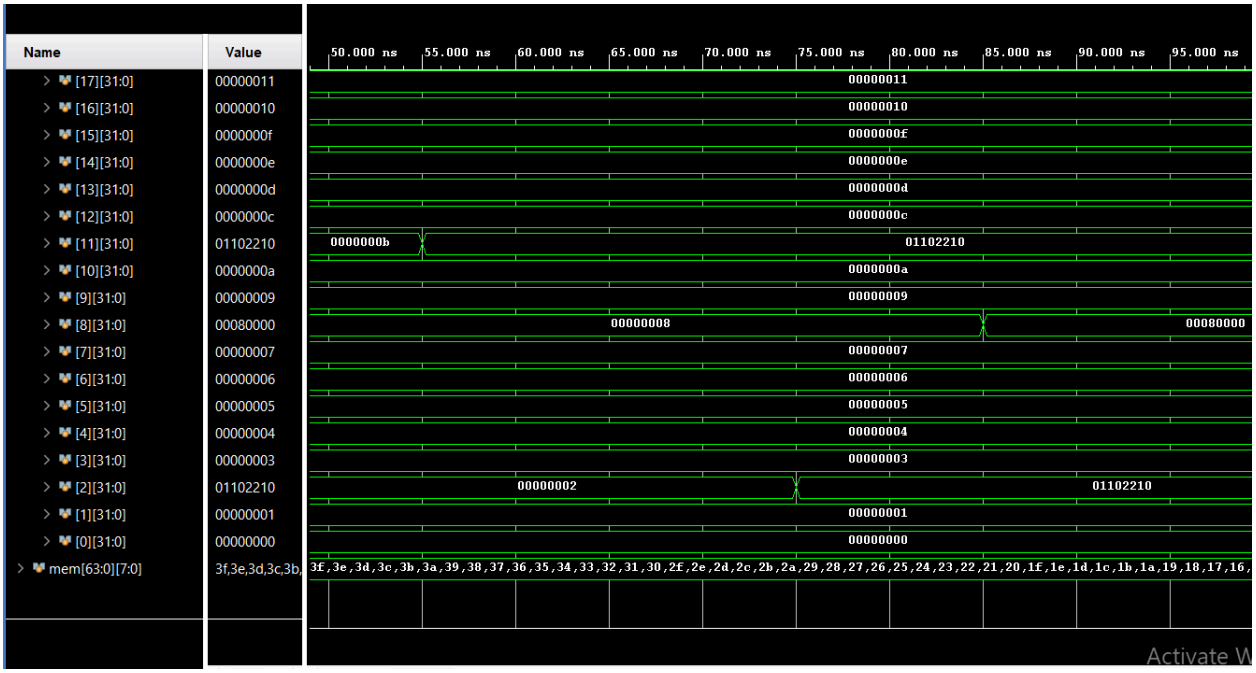
10. Test the processor design by generating the appropriate clock and reset. Copy the image of your testbench code here.

    Answer:

```
    module fproc_tb();

    reg clock, reset;

    fproc q1(clock, reset);

    parameter stop_time = 900;

    initial #stop_time $finish;

    initial begin
        clock = 0;
        forever #5 clock = ~clock;

    end

    initial begin
        reset = 0;
        #1 reset = 1;
        #1 reset = 0;
        #8 reset = 1;

    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):




```
// lw r11, 12[r1]
// mul r2, r1, r11
// lui r8, r2, 8
// jr r4
// mul r9, r8, r1
// mul r6, r6, r6
// lui r21, r23, 89
// ori r7, r15, 67
// sw r4, 4[r5]
// mul r9, r8, r1
```

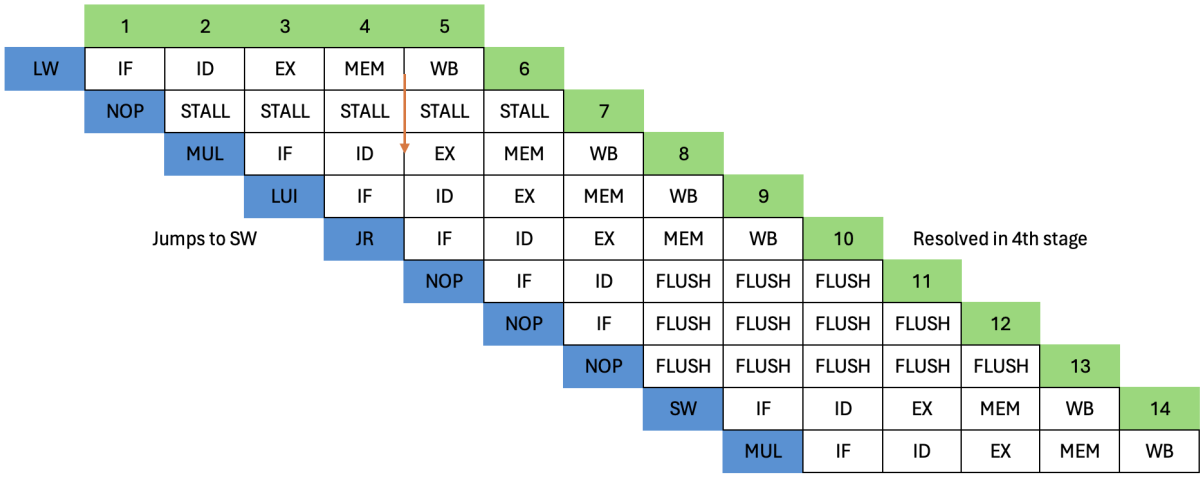
The jump instruction jumps to the location of SW.

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: 10 cycles needed to fetch the 6 instructions. 14 cycles to execute 6 instructions - CPI - 2.334

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

    Answer:



14. Is your design synthesizable? Which target FPGA was used for synthesis?

    Answer: Yes, the design in synthesizable

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

    Answer:

Power

Total On-Chip Power:	**0.079 W**
Junction Temperature:	**25.1 °C**
Thermal Margin:	59.9 °C (31.6 W)
Effective ϑJA:	1.9 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	High

Implemented Power Report

Unrelated Questions

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

Answer: Problem in making the flush unit and the right jump address.

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: I did not take any 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 : Abhimanyu Magapu Date: 23 March 2024
ID No. : 2021A3PS2516G