CS F342 COMPUTER ARCHITECTURE
ASSIGNMENT 1

Implement a 5-stage MIPS pipelined processor in Verilog and a forwarding unit that resolves data hazards. This processor supports load word (lw), store word (sw), jump on zero (jz), addition (add), and subtraction (sub) instructions. The processor should implement forwarding and stalling to resolve data hazards. The processor has Reset CLK as inputs and no outputs. The processor has instruction Fetch, Decode, Execution, Memory, and Writeback units. The processor also contains four pipelined buffers 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**. The pipeline registers contain unknown values when the instruction unit fetches the first instruction. When the second instruction is fetched in the IF unit, the IF/ID registers 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 Address and Data size is 32-bits)
The processor will have static branch prediction, predicting branch not taken. In case of incorrect prediction,, at the end of EX stage, the relevant pipeline registers will be flushed.

The instruction and its **32-bit instruction format** are shown below:

lw destinationReg, offset(sourceReg)
Sign extends the offset to 32-bits, adds it to sourceReg1, it loads the data in this calculated address to sourceReg2. The opcode for lw is 100011.

opcode	rs(sourceReg1)	rt(sourceReg2)	imm(offset)
6 bits (31:26)	5 bits (25:21)	5 bits (20:16)	16 bits (15:0)

sw sourceReg1, offset(sourceReg2)
Sign extends the offset to 32-bits, adds it to sourceReg1, it stores the data in sourceReg2 to this calculated address. The opcode for sw is 101011.

opcode	rs(sourceReg1)	rt(sourceReg2)	imm(offset)
6 bits (31:26)	5 bits (25:21)	5 bits (20:16)	16 bits (15:0)

jz rs, address
Jumps to the address calculated by left shifting the specified address by 2 bits, if the contents of sourceReg is 0. The opcode for jz is 000010.

opcode	rs(sourceReg)	address
6 bits (31:26)	5 bits (25:21)	21 bits (20:0)

add destinationReg, sourceReg1, sourceReg2
Stores the addition of sourceReg1 and sourceReg2 in destinationReg. Opcode is 000000, shamt is 00000, and the funct is 100000.

opcode	rs(sourceReg1)	rt(sourceReg2)	rd(destinationReg)	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)

sub destinationReg, sourceReg1, sourceReg2

Stores the subtraction of sourceReg1 and sourceReg2 in destinationReg. Opcode is 000000, shamt is 00000, and the funct is 100010.

opcode	rs(sourceReg1)	rt(sourceReg2)	rd(destinationReg)	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. All registers in the register file should be set to zero on reset. 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. On reset, the first location in DMEM, DMEM[0], should be initialized to the decimal number 20. On reset, ensure that the instruction memory gets initialized with the following instructions, starting at address 0:

```
        lw r1, 0(r0)
        add r2, r1, r0
        sub r2, r2, r1
        jz r2 L
        add r3, r2, r1
L:      add r4, r2, r1
        sw r1, 0(r5)
```

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

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 in a pdf file name, **IDNO_NAME.pdf**.

2. Design Verilog Files for all the sub-modules.

3. Design a Verilog file for the main processor.

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

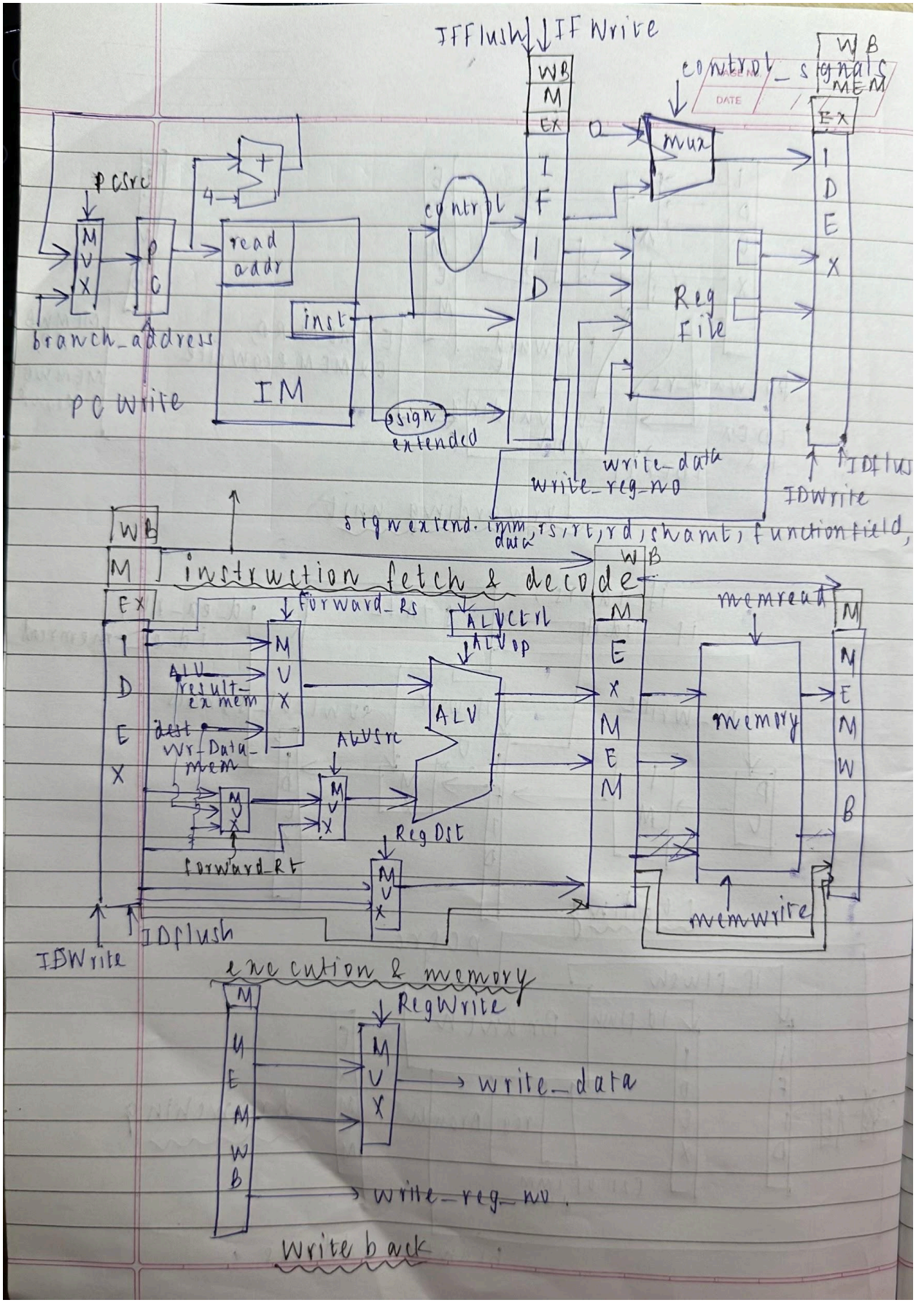
---

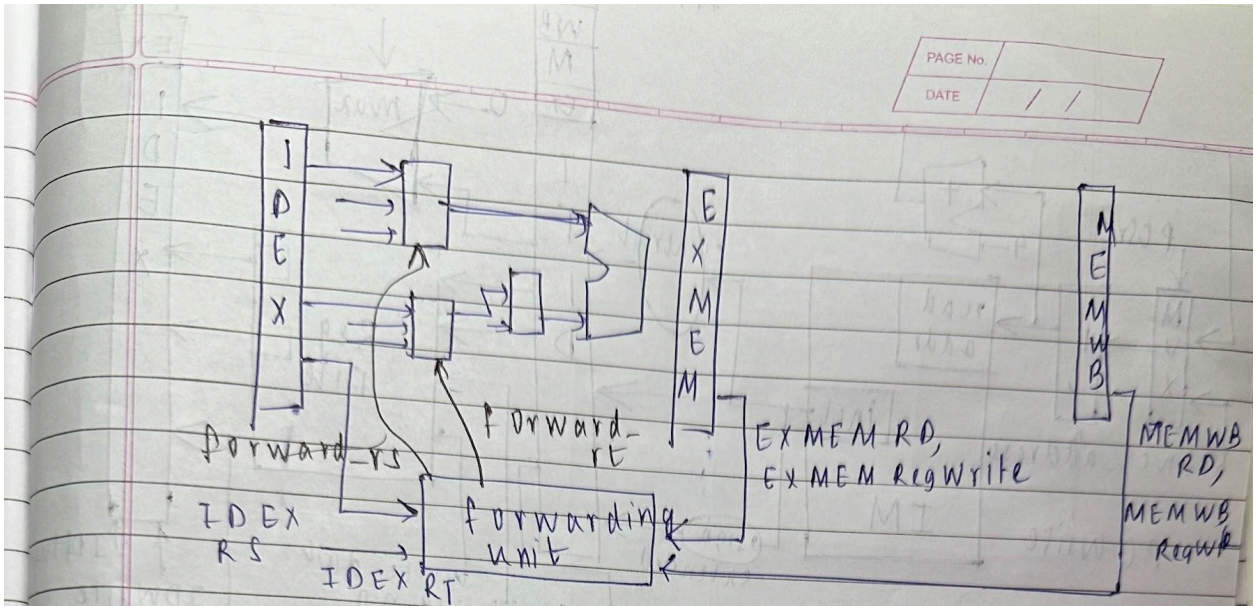
Name: Aishwarya Nag **ID No:** 2021AAPS1988G

Questions Related to Assignment

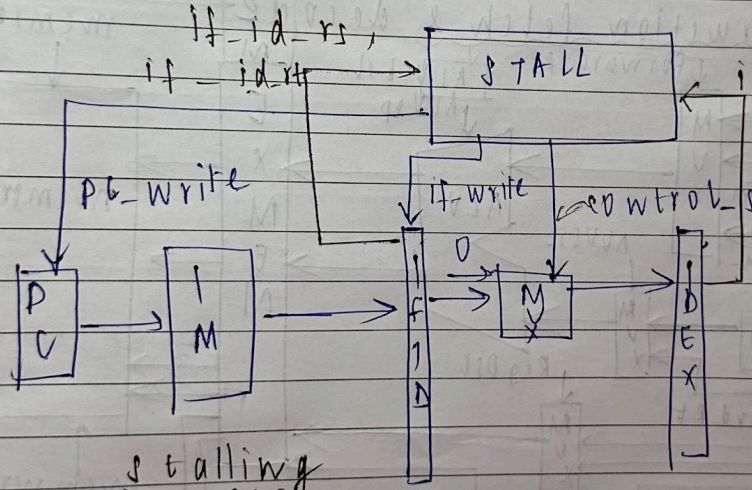
1. **Draw the complete Datapath and show control signals of the 5-stage pipelined processor.**

Answer:

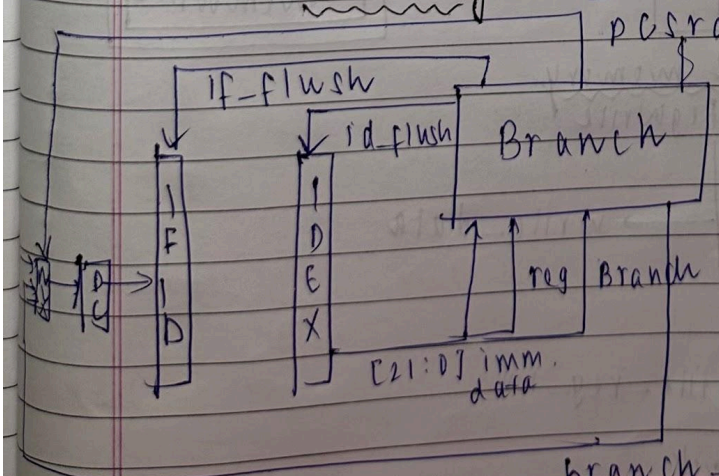

IFFlush
IF Write
WB
M
EX
control_signals
MEM
WB
EX
control
mux
PCSrc
M
U
X
P
C
+
4
read addr
inst
IM
branch_address
PC Write
I
F
I
D
Reg File
I
D
E
X
sign extended
write_data
write_reg_no
IDflush
IDWrite
sign extend. imm, rs, rt, rd, shamt, function field,
data
instruction fetch & decode
forward_RS
ALUCtrl
ALUop
ALU result ex mem
dest
Wr_Data_mem
ALUSrc
ALU
RegDst
forward_RT
IDflush
IEWrite
memread
memory
memwrite
execution & memory
RegWrite
write_data
write_reg_no
Write back



Forwarding unit



stalling



Branching

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

Answer:

Instructions	**Control Signals**							
	RegWrite	MemtoReg	MemRead	MemWrite	Branch	RegDst	ALUop	ALUSrc
lw	1	1	1	0	0	0	11	1
sw	0	X	0	1	0	X	10	1
jz	0	X	0	0	1	X	01	0
add	1	0	0	0	0	1	00	0
sub	1	0	0	0	0	1	00	0

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

Answer

```
module inst_fetch_da(
    input clk,
    input reset,
    input PCSrc,
    input pc_write,
    input [31:0] branch_address,
    output [31:0] Instruction_Code
    );

reg [31:0] PC;
initial PC = 0;

inst_mem_da mem(.PC(PC), .reset(reset), .Instruction_Code(Instruction_Code));

always @(posedge clk, negedge reset)
begin
if (reset==0)
PC <= 0;
else
begin
    if (pc_write==1)
    begin
    if (PCSrc==0)
    PC <= PC + 4;
    else
    PC <= branch_address;

    end

    end

end

always @(PCSrc)
begin
    if(PCSrc)
        PC<=branch_address;
end

endmodule
```

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

Answer:

```
//asumption: control signals calculated and updated in IFID in first cycle
inst_fetch_da instf (.clk(clk), .reset(reset), .PCSrc(PCSrc), .pc_write(pc_write),  .branch_address(branch_address), .Instruction_Code(Instruction_Code));
control_da ctrl (.Instruction_Code(Instruction_Code),.RegDst(RegDst),.RegWrite(RegWrite),.ALUSrc(ALUSrc),.ALUop(ALUop), .MemRead(MemRead),
.MemWrite(MemWrite), .MemtoReg(MemtoReg), .Branch(Branch));
```

```
module reg_file_da(
    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 clk,
    input reset
    );

    reg [31:0] registerFile [31:0];

    assign Read_Data_1 = registerFile [Read_Reg_Num_1];
    assign Read_Data_2 = registerFile [Read_Reg_Num_2];

    always @ (reset) //asynchronous reset
    begin
        if (reset == 0)
        begin
            registerFile [0] = 32'h0; registerFile [1] = 32'h0; registerFile [2] = 32'h0; registerFile [3] = 32'h0;
            registerFile [4]= 32'h0; registerFile [5] =32'h0; registerFile [6] = 32'h0; registerFile [7] = 32'h0;
            registerFile [8] = 32'h0; registerFile [9] = 32'h0; registerFile [10] = 32'h0; registerFile [11] = 32'h0;
            registerFile [12]= 32'h0; registerFile [13] =32'h0; registerFile [14] = 32'h0; registerFile [15] = 32'h0;
            registerFile [16] = 32'h0; registerFile [17] = 32'h0; registerFile [18] = 32'h0; registerFile [19] = 32'h0;
            registerFile [20]= 32'h0; registerFile [21] =32'h0; registerFile [22] = 32'h0; registerFile [23] = 32'h0;
            registerFile [24] = 32'h0; registerFile [25] = 32'h0; registerFile [26] = 32'h0; registerFile [27] = 32'h0;
            registerFile [28]= 32'h0; registerFile [29] =32'h0; registerFile [30] = 32'h0; registerFile [31] = 32'h0;
```

```
        end

    end

    always @ (*)
    begin
    if (clk)
    begin
        if (RegWrite == 1)
        begin
            registerFile [Write_Reg_Num] = Write_Data;
            registerFile [0] = 32'h0; //register 0 is the $zero register
        end
    end
    end

endmodule
```

5. **Implement the Register File and copy the <u>image</u> of the Verilog code of the Register file unit here.**

Answer:

```
module reg_file_da(
    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 clk,
    input reset
    );

    reg [31:0] registerFile [31:0];

    assign Read_Data_1 = registerFile [Read_Reg_Num_1];
    assign Read_Data_2 = registerFile [Read_Reg_Num_2];

    always @ (reset) //asynchronous reset
    begin
        if (reset == 0)
        begin
            registerFile [0] = 32'h0; registerFile [1] = 32'h0; registerFile [2] = 32'h0; registerFile [3] = 32'h0;
            registerFile [4]= 32'h0; registerFile [5] =32'h0; registerFile [6] = 32'h0; registerFile [7] = 32'h0;
            registerFile [8] = 32'h0; registerFile [9] = 32'h0; registerFile [10] = 32'h0; registerFile [11] = 32'h0;
            registerFile [12]= 32'h0; registerFile [13] =32'h0; registerFile [14] = 32'h0; registerFile [15] = 32'h0;
            registerFile [16] = 32'h0; registerFile [17] = 32'h0; registerFile [18] = 32'h0; registerFile [19] = 32'h0;
            registerFile [20]= 32'h0; registerFile [21] =32'h0; registerFile [22] = 32'h0; registerFile [23] = 32'h0;
            registerFile [24] = 32'h0; registerFile [25] = 32'h0; registerFile [26] = 32'h0; registerFile [27] = 32'h0;
            registerFile [28]= 32'h0; registerFile [29] =32'h0; registerFile [30] = 32'h0; registerFile [31] = 32'h0;


        end

    end

    always @ (*)
    begin
    if (clk)
    begin
        if (RegWrite == 1)
        begin
            registerFile [Write_Reg_Num] = Write_Data;
            registerFile [0] = 32'h0; //register 0 is the $zero register
        end
    end
    end

endmodule
```

6. **What is the data hazard in the given instruction snippet? Determine the condition that can detect the data hazard in the given instruction snippet.**

Answer: There is Read After Write data hazard in the code snippet.

i. lw r1, 0(r0) ; add r2, r1, r0 : MEMWB_regWrite=1 & MEMWB_RD=IDEX_RS is used to check the hazard. It is resolved using stalling followed by forwarding from MEMWB stage to RS.

ii. add r2, r1, r0, sub r2, r2, r1 : EXMEM_regWrite=1 & EXMEM_RD=IDEX_RS are used to check hazards. It is resolved using forwarding from EXMEM stage to RS.

iii. sub r2, r2, r1; jz r2 L : EXMEM_regWrite=1 & EXMEM_RD=IDEX_RS are used to check hazards. It is resolved using forwarding from EXMEM stage to RS.

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

Answer:

```
module forw_unit_da(
    input [4:0] ex_mem_rd,
    input [4:0] mem_wb_rd,
    input [4:0] id_ex_rs,
    input [4:0] id_ex_rt,
    input ex_mem_regWrite,
    input mem_wb_regWrite,
    output reg [1:0] forward_rs,
    output reg [1:0] forward_rt
    );
    //register 0 is the $zero register

assign condition_1_a = ( (ex_mem_regWrite==1) && (ex_mem_rd!=0) && (ex_mem_rd==id_ex_rs) );
assign condition_1_b = ( (ex_mem_regWrite==1) && (ex_mem_rd!=0) && (ex_mem_rd==id_ex_rt) );
assign condition_2_a = ( (mem_wb_regWrite==1) && (mem_wb_rd!=0) && (mem_wb_rd==id_ex_rs) && ~(condition_1_a) );
assign condition_2_b = ( (mem_wb_regWrite==1) && (mem_wb_rd!=0) && (mem_wb_rd==id_ex_rt) && ~(condition_1_b) );



always @ (*)
begin

    if (condition_1_a)
    begin
        forward_rs = 01;
    end

    if (condition_1_b)
    begin
        forward_rt = 01;
    end


    if (condition_2_a)
    begin
        forward_rs = 10;
    end

    if (condition_2_b)
    begin
        forward_rt = 10;
    end

    else
    begin
        forward_rs = 00;
        forward_rt = 00;
    end

end

endmodule
```

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

Answer:

```
module datapath_da(
    input clk,
    input reset
    );

    wire pc_write, if_write, control_signals, id_ex_clk, if_flush, id_flush;
    wire [31:0] m4_output; //all control signals
    wire [31:0] branch_address;
    wire PCSrc;
    wire [31:0] Instruction_Code;
    wire ALUSrc, RegDst, Branch, MemWrite, MemRead, MemtoReg, RegWrite;
    wire [1:0] ALUop;
    wire [31:0] Read_Data_1, Read_Data_2, Write_Data;
    wire [31:0] B, result;
    wire zerof, ALUcontrol;
    wire [31:0] m1_output; //detination register
    wire [31:0] read_data, write_data;
    wire [1:0] forward_rs, forward_rt;
    wire [31:0] fwd_rs_mux_out, fwd_rt_mux_out;

    reg [40:0] IFID;
    reg [130:0]IDEX;
    reg [74:0]EXMEM;
    reg [70:0]MEMWB;

    //assumption: branch calculation at the end of execution stage to reduce CPI
    branch_address_da BA (.address_field({ IDEX[13:9], IDEX[18:14], IDEX[130:126], IDEX[120:115]}), .branch_register_data(IDEX[114:83]), .Branch(IDEX[4]),
    .if_flush(if_flush), .id_flush(id_flush), .branch_address(branch_address), .PCSrc(PCSrc));

    stall_da stall (.id_ex_memread(IDEX[2]), .if_id_rs(IFID[25:21]), .if_id_rt(IFID[20:16]), .id_ex_rt(IDEX[13:9]), .pc_write(pc_write), .if_write(if_write),
    .id_ex_clk(id_ex_clk), .control_signals(control_signals));
    mux_da control_signals_stall_mux (.in1(32'b0), .in2({23'b0, IFID [40:32]}), .control(control_signals), .out(m4_output));

    //asumption: control signals calculated and updated in IFID in first cycle
    inst_fetch_da instf (.clk(clk), .reset(reset), .PCSrc(PCSrc), .pc_write(pc_write),  .branch_address(branch_address), .Instruction_Code(Instruction_Code));
    control_da ctrl (.Instruction_Code(Instruction_Code),.RegDst(RegDst),.RegWrite(RegWrite),.ALUSrc(ALUSrc),.ALUop(ALUop), .MemRead(MemRead),
    .MemWrite(MemWrite), .MemtoReg(MemtoReg), .Branch(Branch));

    reg_file_da rf(.Read_Reg_Num_1(IFID[25:21]),.Read_Reg_Num_2(IFID[20:16]), .Write_Reg_Num(MEMWB[6:2]),.Write_Data(Write_Data),.Read_Data_1(Read_Data_1),
    .Read_Data_2(Read_Data_2),.RegWrite(MEMWB[0]),.clk(clk),.reset(reset));

    mux_da dest_reg_mux (.in1({{27'b0},IDEX[13:9]}),.in2({{27'b0},IDEX[18:14]}),.control(IDEX[5]),.out(m1_output)); //choice between rd & rt for destination reg
    ALUCtrlUnit alu_ctrl (.ALUop(IDEX[7:6]), .function_field(IDEX[120:115]), .ALUcontrol(ALUcontrol));
    forw_unit_da fw (.ex_mem_rd(EXMEM[9:5]),.mem_wb_rd(MEMWB[6:2]), .id_ex_rs(IDEX[125:121]), .id_ex_rt(IDEX[13:9]), .ex_mem_regWrite(EXMEM[0]),
     .mem_wb_regWrite(MEMWB[0]), .forward_rs(forward_rs), .forward_rt(forward_rt));
    mux_da_41 fwd_rs_mux (.in1(IDEX[114:83]), .in2(EXMEM[73:42]), .in3(Write_Data), .in4(0), .out(fwd_rs_mux_out), .control(forward_rs)); //choice of forwarding
    mux_da_41 fwd_rt_mux (.in1(IDEX[82:51]), .in2(EXMEM[73:42]), .in3(Write_Data), .in4(0), .out(fwd_rt_mux_out), .control(forward_rt)) ; //choice of forwarding
    mux_da alu_input_mux (.in1(fwd_rt_mux_out),.in2(IDEX[50:19]),.control(IDEX[8]),.out(B)); //choice for input B to ALU
    alu_da eu (.A(fwd_rs_mux_out) , .B(B) , .control (ALUcontrol) , .result (result) , .zerof(zerof));


    data_mem_da dm (.data_address(EXMEM[73:42]),.write_data(EXMEM[41:10]), .read_data(read_data), .MemRead(EXMEM[2]), .MemWrite(EXMEM[3]), .reset(reset));

    mux_da reg_write_mux (.in1(MEMWB[38:7]), .in2(MEMWB[70:39]), .control(MEMWB[1]), .out(Write_Data));

    always @ (posedge clk)
    begin
        if (reset==1)
        begin
        EXMEM[4:0] <= IDEX[4:0]; // assigning WB and MEM control signals
        EXMEM[74:5] <= {zerof, result, IDEX[82:51], m1_output [4:0] }; // zerof, alu_result, read data 2, m1 output
```

```
        MEMWB[1:0] <= EXMEM[1:0]; //assigning WB control signals
        MEMWB[70:2] <= {read_data, EXMEM[73:42], EXMEM[9:5]};//read data from DMEM, ALU result, m1 output
        end

        else
        begin
        IFID <= 41'b0;
        IDEX <= 131'b0;
        EXMEM <= 75'b0;
        MEMWB <= 71'b0;
        end
    end

    always @ (posedge clk)
    begin
        if (reset==1 & if_write==1 & if_flush==1)
        IFID <= {ALUSrc, ALUop, RegDst, Branch, MemWrite, MemRead, MemtoReg, RegWrite, Instruction_Code};
        else if (reset==1 & if_flush==0)
        IFID <= 41'b0;
    end


    always @ (posedge clk)
    begin
        if (reset==1 & id_ex_clk==1 & id_flush==1)
        begin
        IDEX[8:0] <= m4_output [8:0]; //assigning all control signals to IDEX
        IDEX[114:9] <= {Read_Data_1, Read_Data_2, {16{IFID[15]}}, IFID[15:0], IFID[15:11], IFID[20:16] }; //data1, data2, sign extension of imm data, rd, rt
        IDEX[120:115] <= IFID [5:0]; //function field
        IDEX[125:121] <= IFID [25:21]; //rs
        IDEX[130:126] <= IFID [10:6]; //shamt
        end

        else if (reset==1 & (id_ex_clk==0 | id_flush==0) )
        begin
        IDEX[130:0] <= 0;
        end
    end
endmodule
```

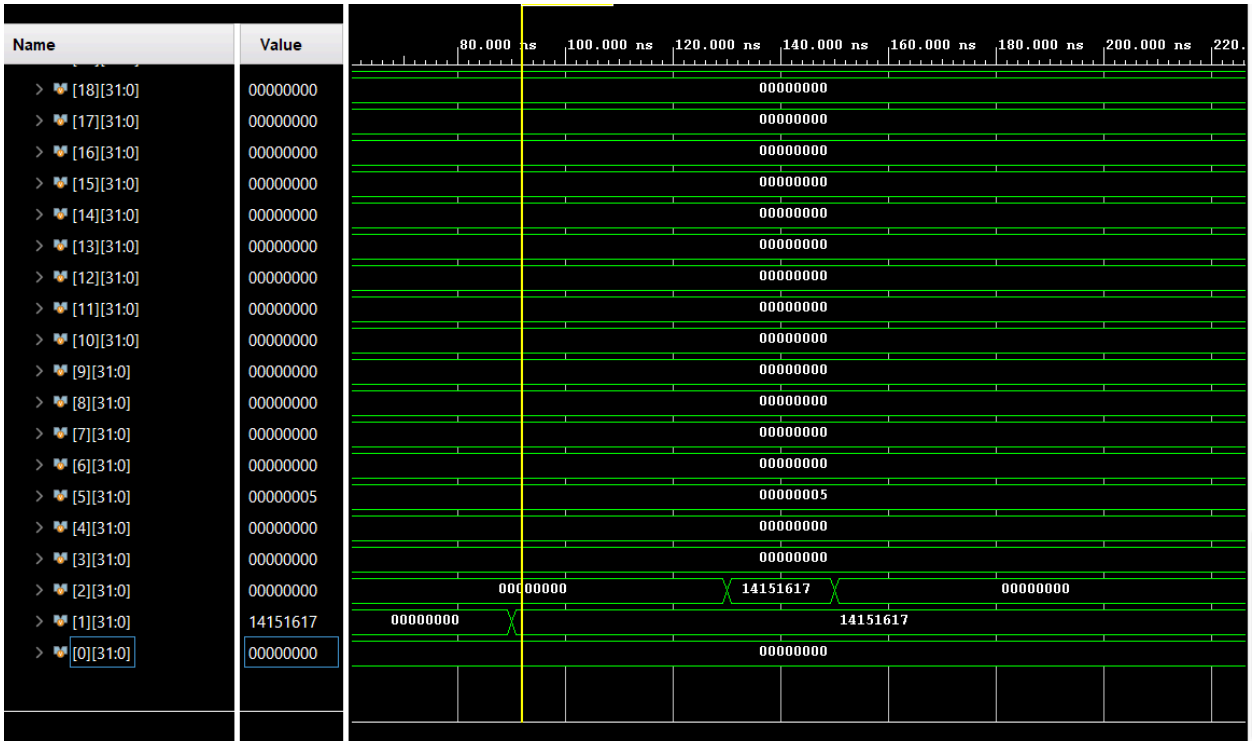
9. **Test the processor design by generating the appropriate clock and reset. Copy the <u>image</u> of your testbench code here.**

Answer:

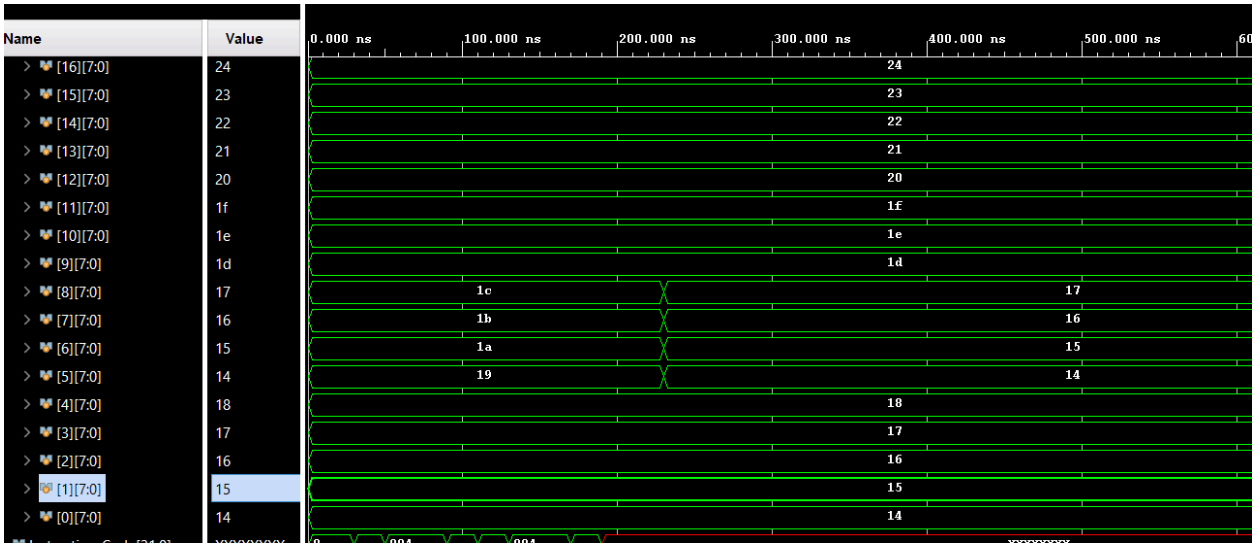
```
module datapath_da_tb(

    );

    reg clk, reset;

    datapath_da dut (.clk(clk), .reset(reset));

    initial begin
    clk = 0;
    repeat (75) #10 clk = ~clk;
    #10 $finish;
    end

    initial begin
    reset = 0;
    #25 reset = 1;
    end
endmodule
```

10. **Verify if data memory is updated according to the set of instructions (mentioned earlier). Copy the <u>image</u> of the waveform of DMEM, CLK, and RESET. Show only those DMEM addresses that are updated in those instructions.**

Answer:



Updated register file after the execution of the instructions (in Hexadecimal).



Updated data memory after execution of the instruction. DMEM[5] is updated.

Data memory does not change as R5 has 0 and R1 has 14151617H. SW R1, 0(R5) adds 0 to R5 content and moves R1 data to that location. Since as per the given question DMEM[4:0] content is already 14151617H there is no change made. Thus, to show SW function R5is initialized to 5.

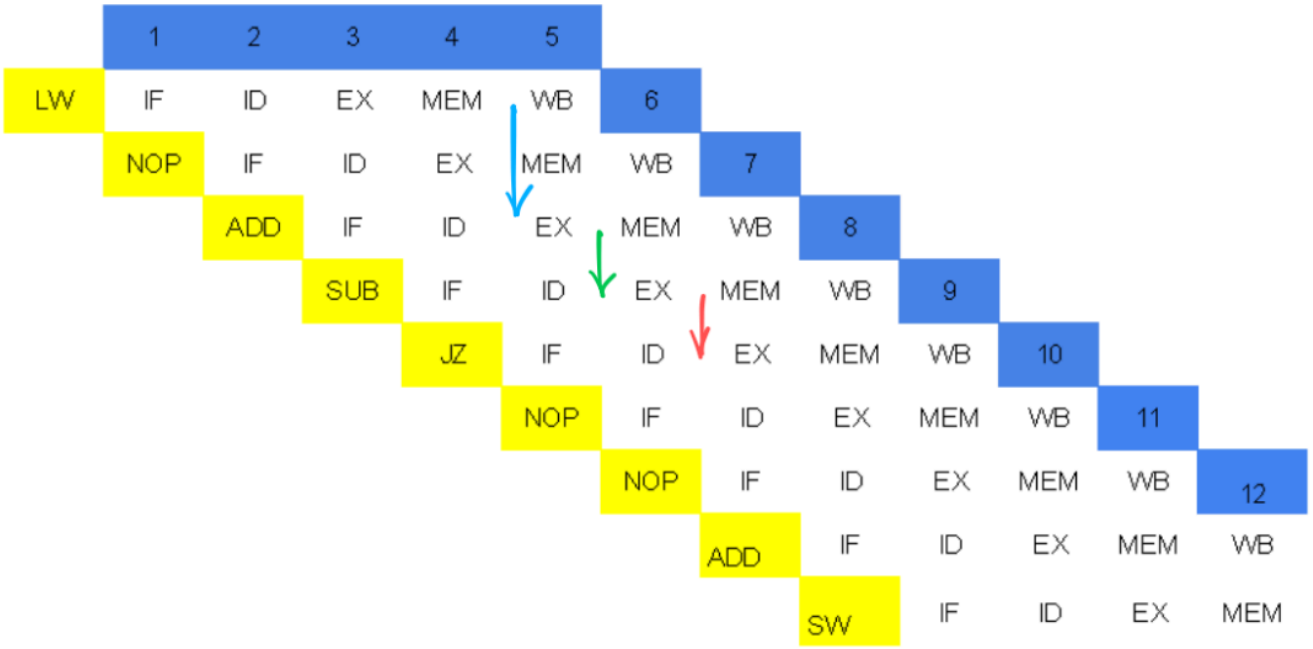
11. What is the total number of cycles needed to issue the program given above on the pipelined MIPS Processor? What is the CPI of the program?

Answer: 9 cycles needed to issue all the instructions. 12 clock cycles needed to execute the programme. CPI = 12/6 = 2.

.

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

Answer: Arrows indicate forwarding and NOPs indicate stalling & flushing. The first NOP indicates a stall while the second and third indicate flush.



SW instruction gets over in the EXMEM stage.

13. 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:

x = 2, y = 3, z = 4

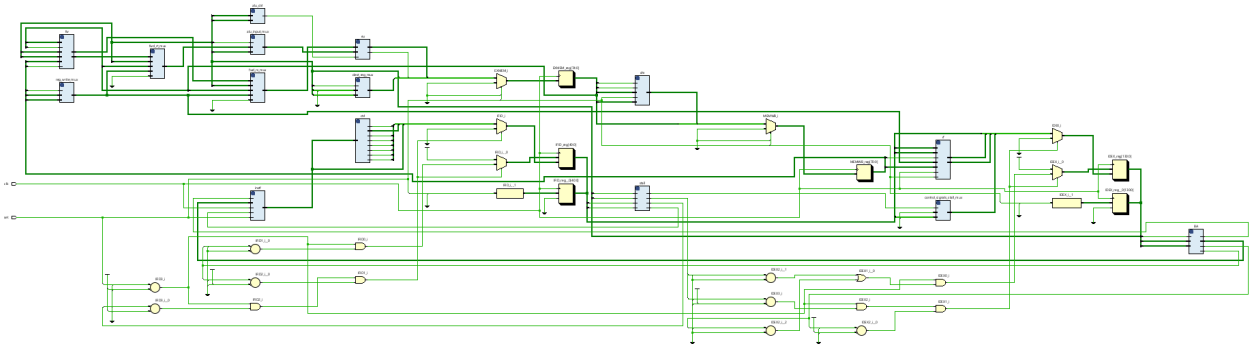
CPI = (0.25*0.5*2 + 0.25*0.5*1) + 0.6*1 + (0.1*(1/3)*3 + 0.1*(2/3)*1) + 0.05*3 = 1.292

Assumption: forwarding present & branch resolution in third stage

CPU Time = (10^9) * 1.292 * (100*10^(-12)) = 0.1292 seconds

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

Answer: Yes it is synthesizable. An additional y output is added and initialized to zero to make it synthesizable. Target FPGA used is xc7a100tcsg324-1 from the Artix-7 family.



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

Answer:

Utilization	1%
Total On-Chip Power	0.084W
Junction Temperature	25.4℃
Power supplied to off-chip devices	0W

Unrelated Questions

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

Answer: Update of the register in the correct clock edge. Update of PC after a taken branch in the correct clock edge. Displaying memory addresses for store word till 14151617H together. Thus, I updated the instructions to check the working of store word correctly.

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 it on my own.

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 others' code/answers to improve my results. (I might have got some doubts cleared by other students).

Name: Aishwarya Nag **Date:** 27th March, 2024
ID No.: 2021AAPS1988G