Course Name: Computer Architecture Lab	**Course Code:** CMPE-421L
Assignment Type: Lab	**Dated:** 15th December 2023
Semester: 7th	**Session:** 2020
Lab/Project/Assignment #: 12	**CLOs to be covered: CLO 3**
Lab Title: Pipelined Architecture (Resolving Hazards)	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation

CLO 3	Understanding and implementation of pipelined RISC-V architecture and handling hazards associated with it.					
Levels (Marks)	**Level1**	**Level2**	**Level3**	**Level4**	**Level5**	**Level6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	Marks	Level	Rubric
Excellent	**9-10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**7-8**	L2	Submitted the lab tasks but have good understanding
Good	**5-6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**3-4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**1-2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

Lab # 12

Lab Goals

By reading this manual, students will be able to:

- Understand the Pipelined Architecture
- How to resolve Data Hazards
- How to resolve Control Hazards

Equipment Required

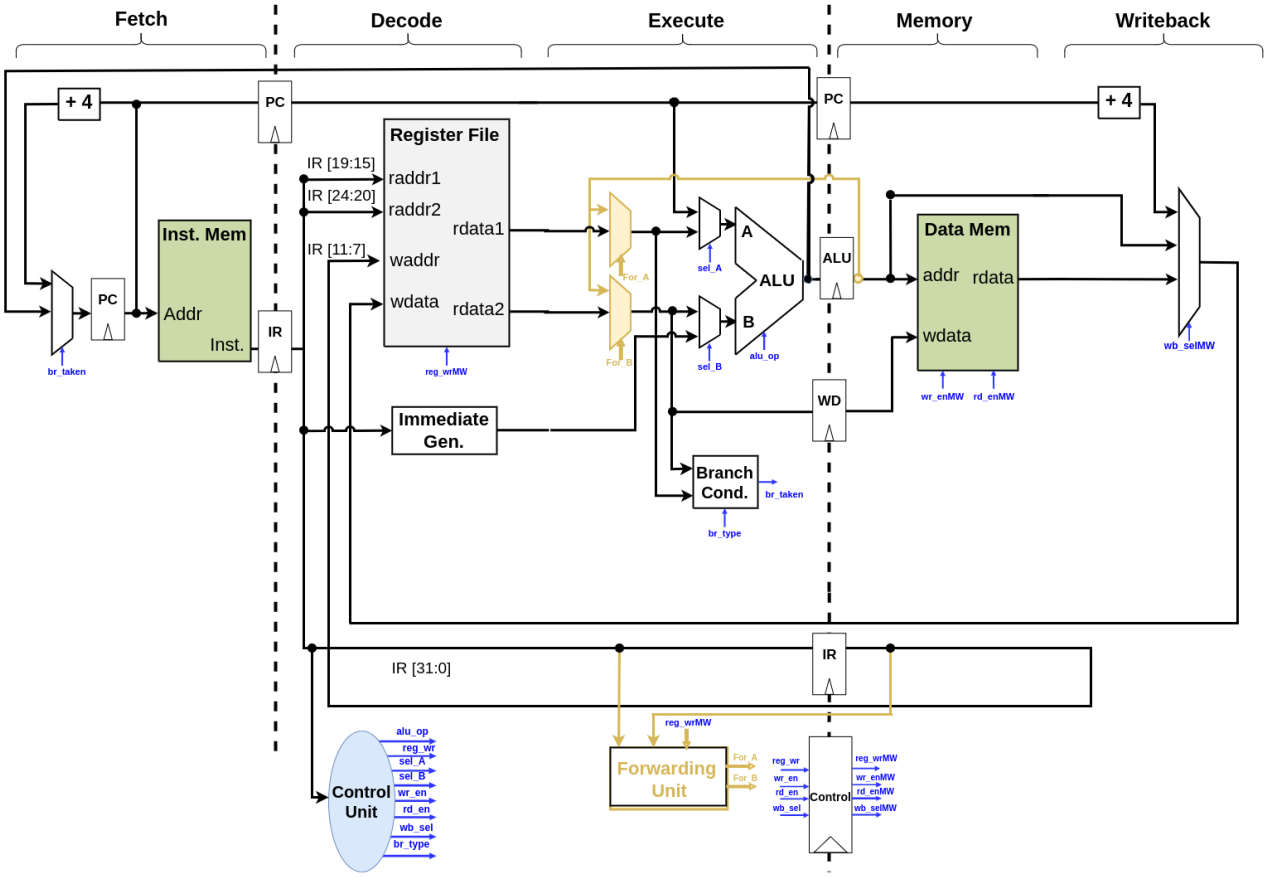
- Computer system with ModelSim or Xcelium, installed on it.

Pipelined Architecture (Resolving Hazards)

In the previous lab, the single cycle RISC-V processor was converted to a pipelined processor by the help of pipelining registers. The pipelined processor is going to handle multiple instructions concurrently and due to dependency of the result of an instruction on another. These hazards can be classified as data or control hazards. Data hazards take place when an instruction tries to read a register that has not been updated by the previous instructions. On the other hand, the control hazards take place when the decision of fetching the next instruction has not been during the decode stages. The control hazards in case of jumps and taken branches. This is due to the fact that when jump/branch is resolved in the execution phase, the subsequent instruction is being fetched simultaneously.

Resolving Data Hazards

In the case of the three-stage pipeline, some data hazards can be resolved by forwarding the result of the Memory-Writeback stage to the Decode-Execute stage which is performed by adding forwarding multiplexers. Forwarding is used when the destination register in the Memory-Writeback stage matches either of the source registers in the Decode-Execute stage. This leads to the addition of two forwarding multiplexers and a forwarding unit which takes the whole instruction in the two pipeline stages as the inputs and the selection of the two muxes becomes the outputs.



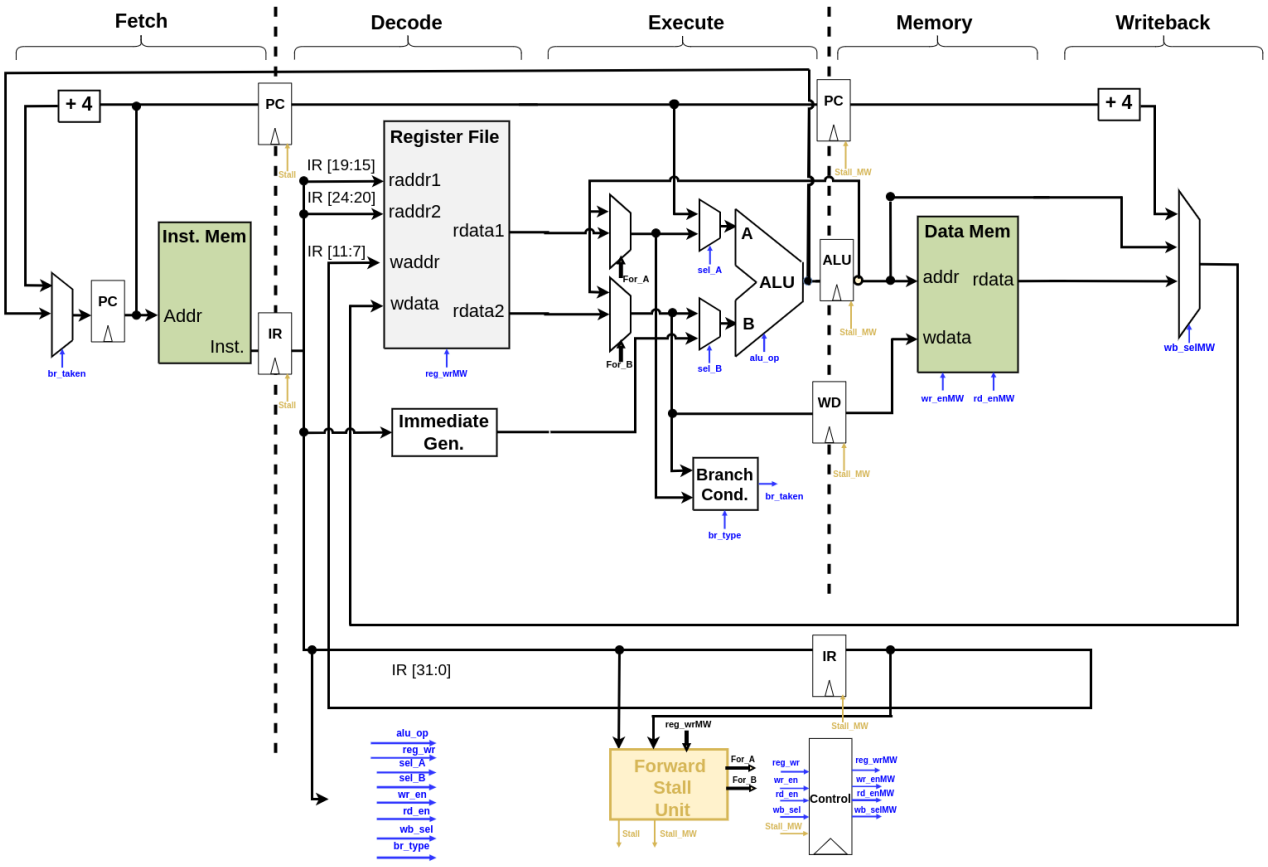
Pipelined processor microarchitecture with forwarding.

```
// Check the validity of the source operands from EXE stage
assign rs1_valid = |exe2fwd.rs1_addr;
assign rs2_valid = |exe2fwd.rs2_addr;

// Hazard detection
assign lsu2rs1_hazard = ((exe2fwd.rs1_addr == lsu2fwd.rd_addr) & lsu2fwd.rd_wr_req) &
rs1_valid;
assign lsu2rs2_hazard = ((exe2fwd.rs2_addr == lsu2fwd.rd_addr) & lsu2fwd.rd_wr_req) &
rs2_valid;

// Generate the forwarding signals
assign fwd2exe.fwd_lsu_rs1 = lsu2rs1_hazard;
assign fwd2exe.fwd_lsu_rs2 = lsu2rs2_hazard;
```

Forwarding is not sufficient in case of load instructions which can have multi-cycle latency due to which the results can not be forwarded. The only solution left would be to stall the pipeline until the result has been written to the register file. When a stage is stalled, all the previous stages must also be stalled in order to avoid instruction loss. For this purpose, we add the stalling capability to the forwarding to make it the forward stall unit. This adds the stall signals to all the pipeline registers which has been illustrated in Figure

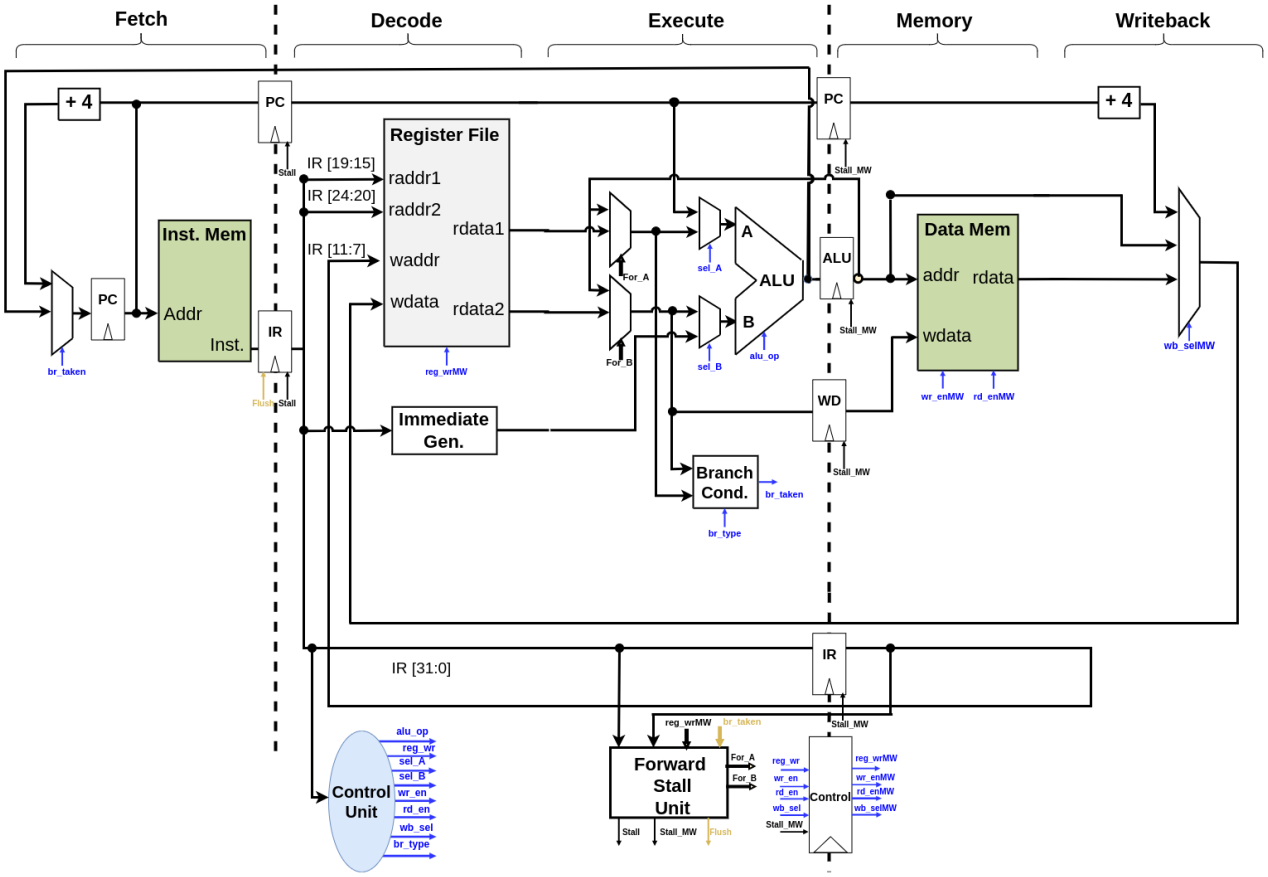


Pipelined processor microarchitecture with stalling.

.

Resolving Control Hazards

For taken branches as well as jumps the following instruction (which has been fetched) should not be executed. Rather it should be flushed from the pipeline, while the program counter is updated to the new address. For this purpose we need to flush the Decode-Execute stage which is done by setting the instruction pipeline register between the Fetch stage and the Decode-Execute to nop. For this purpose, we need to modify our forward stall module to add the br_taken flag as its input and the flush signal as the output. These changes can be observed in given Figure.



Pipelined processor microarchitecture with flushing.

Implementation for PC updating and fetch stage flushing.

```
// PC update state machine
always_ff @(posedge clk) begin
    if (rst_n) begin
        pc_ff <= '0;
    end else begin
        pc_ff <= pc_next;
    end
end

assign pc_next = exe2if_fb.jump_br_taken ? exe2if_fb.alu_pc
               : id2if_fb_rdy            ? (pc_ff + 32'd4)
               : pc_ff;
```

PC update state machine incorporating jump/branch related PC updating.

```
`ifdef IF2ID_PIPELINE_STAGE
    assign if2id_data.instr = exe2if_fb.jump_br_taken
                            ? `INSTR_NOP       // Insert NOP for jump or branch taken
                            : imem2if_rdata_i;
`else
    assign if2id_data.instr = imem2if_rdata_i;
`endif
```

Instruction flushing during fetch phase for jump/branch related control hazard.

Tasks

- Implement the proposed stall/forwarding/flush strategy to resolve as many hazards as possible and implement the proposed strategy.
- Write an assembly program to test some of these hazards and verify the implementation.