Task - 3

Designing a 4-stage pipelined processor involves breaking down the processor's operations into distinct stages, each handling a part of an instruction cycle. The main stages usually are:

1. **Instruction Fetch (IF)**: Fetch the instruction from memory.
2. **Instruction Decode (ID)**: Decode the instruction and read register values.
3. **Execution (EX)**: Perform the operation (e.g., ADD, SUB, AND, LOAD).
4. **Write Back (WB)**: Write the result back to the register file or memory.

To design a 4-stage pipelined processor with basic instructions like ADD, SUB, AND, and LOAD, here's an outline of the architecture and how each instruction would work in each stage.

**Processor Architecture Overview**

1. **Registers**:
   * A set of registers (e.g., R0 to R7) where data is temporarily stored.
   * A Program Counter (PC) to keep track of the current instruction.
   * A Register File to store values being operated on.
2. **Control Unit**:
   * Generates control signals to manage the flow of data between the stages and the operations.
3. **ALU (Arithmetic Logic Unit)**:
   * Performs arithmetic operations (ADD, SUB) and logic operations (AND).
4. **Memory**:
   * For load and store operations.
5. **Pipeline Registers**:
   * Registers are placed between each stage to hold intermediate values during the pipeline's operation.

**Pipeline Stages and Operation Breakdown**

1. **Instruction Fetch (IF) Stage**:
   * **Operation**: The instruction is fetched from memory using the Program Counter (PC).
   * **Data Flow**: The instruction is passed to the ID stage, and the PC is incremented to point to the next instruction.
2. **Instruction Decode (ID) Stage**:
   * **Operation**: The instruction is decoded, and the required registers are read.
   * **Data Flow**: The operands for the instruction are fetched from the register file.
   * Control signals are generated to know whether the instruction is an ADD, SUB, AND, or LOAD.
3. **Execution (EX) Stage**:
   * **Operation**: The ALU performs the operation specified by the decoded instruction.
     + For **ADD** and **SUB**, it adds or subtracts the values.
     + For **AND**, it performs a bitwise AND operation.
     + For **LOAD**, the effective address is calculated.
   * **Data Flow**: The result of the operation (e.g., the sum, difference, logical result, or address for load) is passed to the next stage.
4. **Write Back (WB) Stage**:
   * **Operation**: The result from the EX stage is written back to the register file or memory (in the case of a LOAD).
   * **Data Flow**: For arithmetic operations, the result is written back into the destination register. For a LOAD instruction, data from memory is written to the register.

**Instruction Flow Example (ADD Instruction)**

Let's assume the instruction is ADD R1, R2, R3 (i.e., R1 = R2 + R3).

1. **IF Stage**: The instruction ADD R1, R2, R3 is fetched from memory.
2. **ID Stage**: The registers R2 and R3 are read, and the instruction is decoded as an ADD operation.
3. **EX Stage**: The ALU performs the addition of the values in registers R2 and R3.
4. **WB Stage**: The result of the addition is written back to register R1.

**Instruction Flow Example (LOAD Instruction)**

Let's assume the instruction is LOAD R1, 100(R2) (i.e., load the value from memory address 100 + R2 into R1).

1. **IF Stage**: The instruction LOAD R1, 100(R2) is fetched from memory.
2. **ID Stage**: The base address in register R2 is read, and the instruction is decoded as a LOAD operation.
3. **EX Stage**: The effective address is calculated (i.e., R2 + 100).
4. **WB Stage**: The data from the calculated address in memory is loaded into register R1.

**Control Unit and Pipeline Hazards**

The control unit generates control signals to handle data forwarding, register reading/writing, and operation types. Pipelining introduces several potential hazards, which can affect the execution order:

1. **Data Hazard**: When one instruction depends on the result of a previous instruction. This can be mitigated with forwarding and stall mechanisms.
2. **Control Hazard**: Caused by branches or jumps. Can be mitigated with branch prediction techniques or stalling until the branch outcome is determined.

Below is a simplified Verilog code example for a 4-stage pipelined processor.

module pipelined processor (clk, reset, mem data in, mem data out, pc in, pc out);

input clk, reset;

input [31:0] mem data in; // Memory data in

output [31:0] mem data out; // Memory data out

output [31:0] pc\_

in, pc out; // Program counter

// Registers for each stage

reg [31:0] IF\_ID\_IR, ID\_EX\_IR, EX\_WB\_IR; // Instruction Registers for each stage

reg [31:0] IF\_ID\_PC, ID\_EX\_PC, EX\_WB\_PC; // Program Counter values at each stage

// ALU Outputs

reg [31:0] EX ALUResult;

// Memory Output

reg [31:0] WB Data;

// Control signals

reg [2:0] ALU\_OP;

reg MemRead, MemWrite, RegWrite;

// Pipeline Stages

always @(posedge clk or posedge reset) begin

if (reset) begin

// Reset all registers

IF\_ID\_IR <= 32'b0;

ID\_EX\_IR <= 32'b0;

EX\_WB\_IR <= 32'b0;

IF\_ID\_PC <= 32'b0;

ID\_EX\_PC <= 32'b0;

EX\_WB\_PC <= 32'b0;

end else begin

// Instruction Fetch Stage

IF\_ID\_IR <= mem\_data\_in; // Fetch instruction from memory

IF\_ID\_PC <= pc\_in; // Set PC value

// Instruction Decode Stage

ID\_EX\_IR <= IF\_ID\_IR; // Pass instruction to the next stage

ID\_EX\_PC <= IF\_ID\_PC; // Pass PC value to the next stage

// Execution Stage

EX\_WB\_IR <= ID\_EX\_IR; // Pass instruction to the next stage

EX\_ALUResult <= ALU\_OP == 3'b000 ? (ID\_EX\_IR[15:0] + ID\_EX\_IR[31:16]) : 0; // ADD operation example

EX\_WB\_PC <= ID\_EX\_PC + 4; // Increment PC for next instruction

// Write Back Stage (Write result to register or memory)

WB\_Data <= EX\_ALUResult;

end

end

assign pc\_out = EX\_WB\_PC;

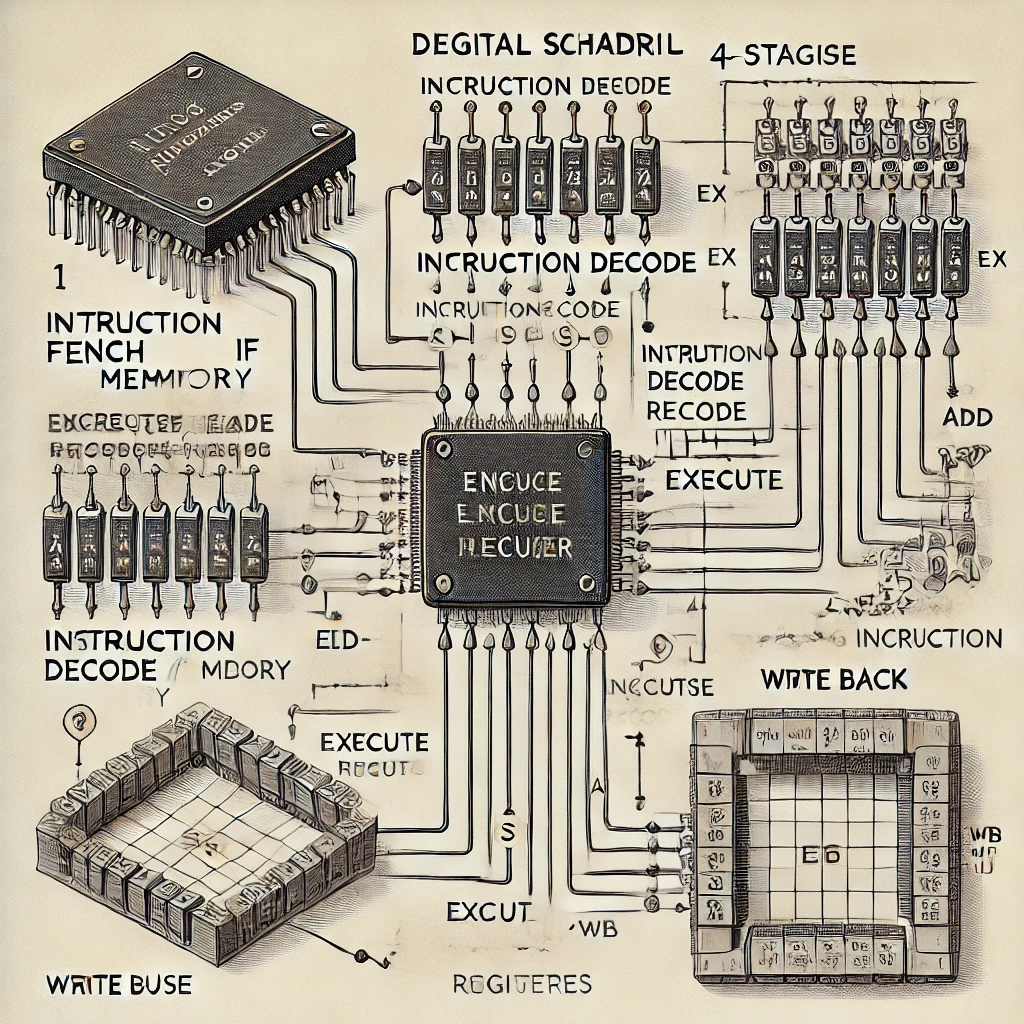
assign mem\_data\_out = WB\_Data;

endmodule

**Simulation and Testing**

To test the processor, you would simulate the execution of basic instructions, showing the pipeline's operation across each stage. The simulation would show:

1. How the instruction moves through each stage (IF, ID, EX, WB).
2. Any hazards or delays introduced during execution (e.g., stalls or forwarding).
3. The values in registers and memory after each instruction completes.



This is a basic outline of a 4-stage pipelined processor.