**4-Stage Pipeline Stages**

1. **IF (Instruction Fetch)**
   * Fetch instruction from memory.
2. **ID (Instruction Decode & Register Fetch)**
   * Decode instruction.
   * Read registers.
3. **EX (Execute / ALU Operation / Address Calculation)**
   * Perform arithmetic or address calculation.
4. **MEM/WB (Memory Access or Write Back)**
   * Load instruction: access memory.
   * Arithmetic instructions: write result to register.

**Supported Instructions**

* ADD R1, R2, R3 → R1 = R2 + R3
* SUB R1, R2, R3 → R1 = R2 - R3
* LOAD R1, 100(R2) → R1 = MEM[R2 + 100]

**Verilog CPU Core (pipeline.v)**

module cpu4 (

input clk, reset

);

// Pipeline registers

reg [31:0] IFID\_instr;

reg [31:0] IDEX\_instr;

reg [31:0] EXMEM\_instr;

reg [31:0] MEMWB\_instr;

reg [31:0] IDEX\_rs1, IDEX\_rs2;

reg [31:0] EXMEM\_alu\_out;

reg [31:0] MEMWB\_mem\_data;

**// Register file and memory**

reg [31:0] regs [0:7];

reg [31:0] mem [0:255];

initial begin

regs[0]=0; regs[1]=0; regs[2]=0; regs[3]=0;

regs[4]=0; regs[5]=0; regs[6]=0; regs[7]=0;

integer i; for (i = 0; i < 256; i = i+1) mem[i] = i + 100;

end

**// Simple program memory**

reg [31:0] imem [0:15];

initial begin

// Instructions: opcode[31:24], rd[23:16], rs1[15:8], rs2/immed[7:0]

imem[0] = {8'h02,8'd1,8'd0,8'd10}; // LOAD R1, 10(R0)

imem[1] = {8'h00,8'd2,8'd1,8'd1}; // ADD R2, R1, R1

imem[2] = {8'h01,8'd3,8'd2,8'd1}; // SUB R3, R2, R1

imem[3] = {8'h00,8'd4,8'd3,8'd2}; // ADD R4, R3, R2

imem[4] = 32'hxxxxxxxx;

end

reg [3:0] pc;

**// Pipeline stages**

always @(posedge clk or posedge reset) begin

if (reset) begin

pc <= 0;

IFID\_instr <= 0; IDEX\_instr <= 0;

EXMEM\_instr <= 0; MEMWB\_instr <= 0;

end else begin

**// 4. MEM/WB: write results**

if (MEMWB\_instr[31:24] == 8'h00) regs[MEMWB\_instr[23:16]] <= MEMWB\_mem\_data;

else if (MEMWB\_instr[31:24] == 8'h01) regs[MEMWB\_instr[23:16]] <= MEMWB\_mem\_data;

else if (MEMWB\_instr[31:24] == 8'h02) regs[MEMWB\_instr[23:16]] <= MEMWB\_mem\_data;

**// Shift pipeline regs**

MEMWB\_instr <= EXMEM\_instr;

EXMEM\_instr <= IDEX\_instr;

IDEX\_instr <= IFID\_instr;

IFID\_instr <= imem[pc];

**// Register reads**

IDEX\_rs1 <= regs[IFID\_instr[15:8]];

IDEX\_rs2 <= (IFID\_instr[31:24]==8'h02)

? {{24{IFID\_instr[7]}}, IFID\_instr[7:0]} // sign-extend immediate

: regs[IFID\_instr[7:0]];

**// ALU/MEM**

if (EXMEM\_instr[31:24]==8'h00) EXMEM\_alu\_out <= regs[EXMEM\_instr[15:8]] + regs[EXMEM\_instr[7:0]];

else if (EXMEM\_instr[31:24]==8'h01) EXMEM\_alu\_out <= regs[EXMEM\_instr[15:8]] - regs[EXMEM\_instr[7:0]];

else if (EXMEM\_instr[31:24]==8'h02) EXMEM\_alu\_out <= regs[EXMEM\_instr[15:8]] + {{24{EXMEM\_instr[7]}},EXMEM\_instr[7:0]};

**// Memory access value passed to WB**

if (MEMWB\_instr[31:24]==8'h02) MEMWB\_mem\_data <= mem[MEMWB\_instr[15:8] + {{24{MEMWB\_instr[7]}},MEMWB\_instr[7:0]}];

else MEMWB\_mem\_data <= EXMEM\_alu\_out;

**// PC update**

pc <= pc + 1;

end

end

endmodule

**Testbench (tb\_cpu4.v)**

`timescale 1ns/1ps

module tb\_cpu4;

reg clk = 0, reset = 1;

cpu4 uut (.clk(clk), .reset(reset));

always #5 clk = ~clk; // 10 ns clock period

initial begin

$dumpfile("cpu4.vcd");

$dumpvars(0, tb\_cpu4);

#10 reset = 0;

#200 $finish;

end

endmodule

**Simulation**

iverilog -o cpu4.vvp pipeline.v tb\_cpu4.v

vvp cpu4.vvp

* **IFID\_instr → IDEX\_instr → EXMEM\_instr → MEMWB\_instr** shift each cycle.
* regs[1] updated after LOAD in stage 4.
* Subsequent ADD/SUB using newly-written register values in later cycles.
* Overlap of instructions with no pipeline stalls (hazards ignored for simplicity).