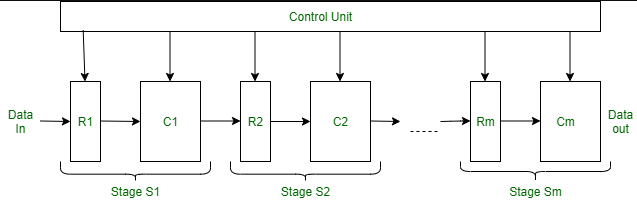
**PIPELINE PROCESSOR DESIGN**

* **Diagram**

****

**CODE**

module pipelined\_processor (

input wire clk,

input wire reset,

output reg [7:0] result

);

// Define pipeline registers

reg [15:0] IF\_ID; // Instruction fetched

reg [15:0] ID\_EX; // Decoded instruction

reg [15:0] EX\_MEM; // Execution result or address

reg [7:0] MEM\_WB; // Final result

// Memory and registers

reg [15:0] instruction\_memory [0:15];

reg [7:0] data\_memory [0:15];

reg [7:0] register\_file [0:7];

// Instruction breakdown

wire [3:0] opcode;

wire [2:0] rs, rt, rd;

wire [7:0] immediate;

assign opcode = ID\_EX[15:12];

assign rs = ID\_EX[11:9];

assign rt = ID\_EX[8:6];

assign rd = ID\_EX[5:3];

assign immediate = ID\_EX[7:0];

// Program Counter

reg [3:0] pc;

// Initialize memory and registers

initial begin

pc = 0;

result = 0;

IF\_ID = 0; ID\_EX = 0; EX\_MEM = 0; MEM\_WB = 0;

// Load instructions

instruction\_memory[0] = 16'b0001\_000\_001\_010\_00; // ADD R2 = R0 + R1

instruction\_memory[1] = 16'b0010\_010\_001\_011\_00; // SUB R3 = R2 - R1

instruction\_memory[2] = 16'b0011\_000\_011\_000001; // LOAD R0 = MEM[R3 + 1]

// Initialize registers

register\_file[0] = 8'h05; // R0 = 5

register\_file[1] = 8'h03; // R1 = 3

register\_file[2] = 8'h00; // R2

register\_file[3] = 8'h00; // R3

end

// Clock-driven operation

always @(posedge clk or posedge reset) begin

if (reset) begin

pc <= 0;

IF\_ID <= 0; ID\_EX <= 0; EX\_MEM <= 0; MEM\_WB <= 0;

end else begin

// Instruction Fetch (IF)

IF\_ID <= instruction\_memory[pc];

pc <= pc + 1;

// Instruction Decode (ID)

ID\_EX <= IF\_ID;

// Execute (EX)

case (opcode)

4'b0001: EX\_MEM <= {8'b0, register\_file[rs] + register\_file[rt]}; // ADD

4'b0010: EX\_MEM <= {8'b0, register\_file[rs] - register\_file[rt]}; // SUB

4'b0011: EX\_MEM <= {register\_file[rs] + immediate, 8'b0}; // LOAD Address

default: EX\_MEM <= 16'b0;

endcase

// Memory Access (MEM)

if (opcode == 4'b0011) begin

MEM\_WB <= data\_memory[EX\_MEM[15:8]]; // LOAD

end else begin

MEM\_WB <= EX\_MEM[7:0]; // Pass result for ADD/SUB

end

// Write Back

if (opcode != 4'b0011) begin

register\_file[rd] <= MEM\_WB;

end

result <= MEM\_WB;

end

end

endmodule

module testbench;

reg clk;

reg reset;

wire [7:0] result;

// Instantiate the processor

pipelined\_processor uut (

.clk(clk),

.reset(reset),

.result(result)

);

// Clock generation

initial begin

clk = 0;

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

end

// Simulation sequence

initial begin

reset = 1;

#10 reset = 0; // Release reset

#100 $finish; // End simulation after 100 ns

end

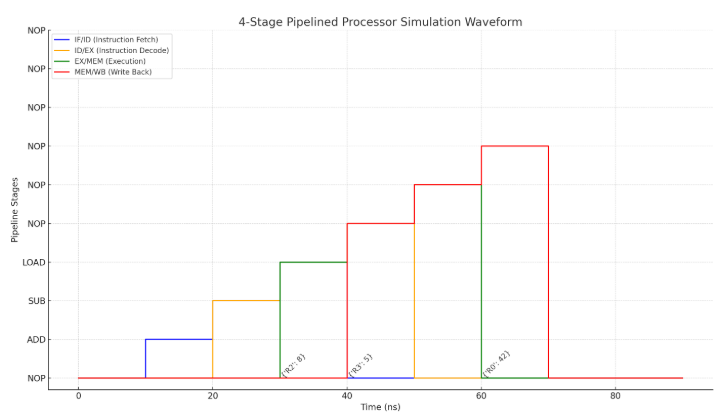
endmodule

**Simulation**

The simulation will show the following sequence

| **Cycle** | **Stage** | **Instruction** | **Registers Updated** |
| --- | --- | --- | --- |
| 1 | IF | Fetch ADD R2, R0, R1 | - |
| 2 | ID | Decode ADD R2, R0, R1 | - |
| 3 | EX | Execute R2 = R0 + R1 | - |
| 4 | MEM | Write R2 = 8 | R2 updated |
| 5 | IF | Fetch SUB R3, R2, R1 | - |

* **WAVEFORM**



**Details**

1. **Stages:**
   * IF/ID: Fetching instructions.
   * ID/EX: Decoding and preparing operands.
   * EX/MEM: Executing arithmetic operations or calculating addresses.
   * MEM/WB: Accessing memory or writing back results.
2. **Register Updates:**
   * At specific time steps, the updated register values are annotated on the timeline:
     + Cycle 30 ns: R2 updated to 8 (result of ADD).
     + Cycle 40 ns: R3 updated to 5 (result of SUB).
     + Cycle 60 ns: R0 updated to 42 (result of LOAD).