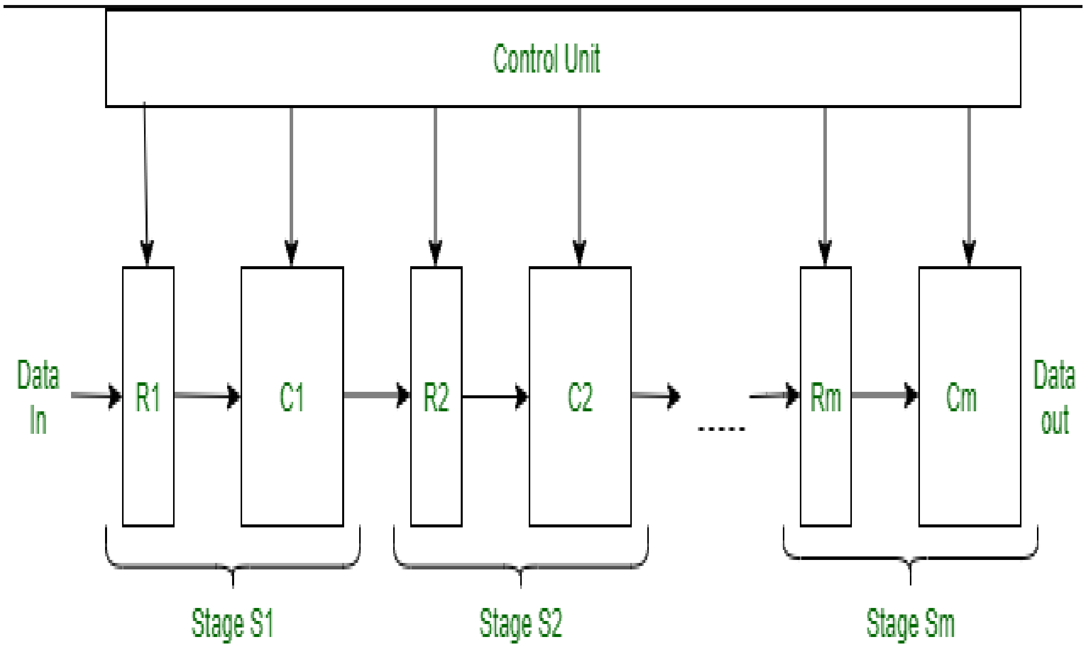
PIPELINE PROCESSOR DESIGN

- **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
```

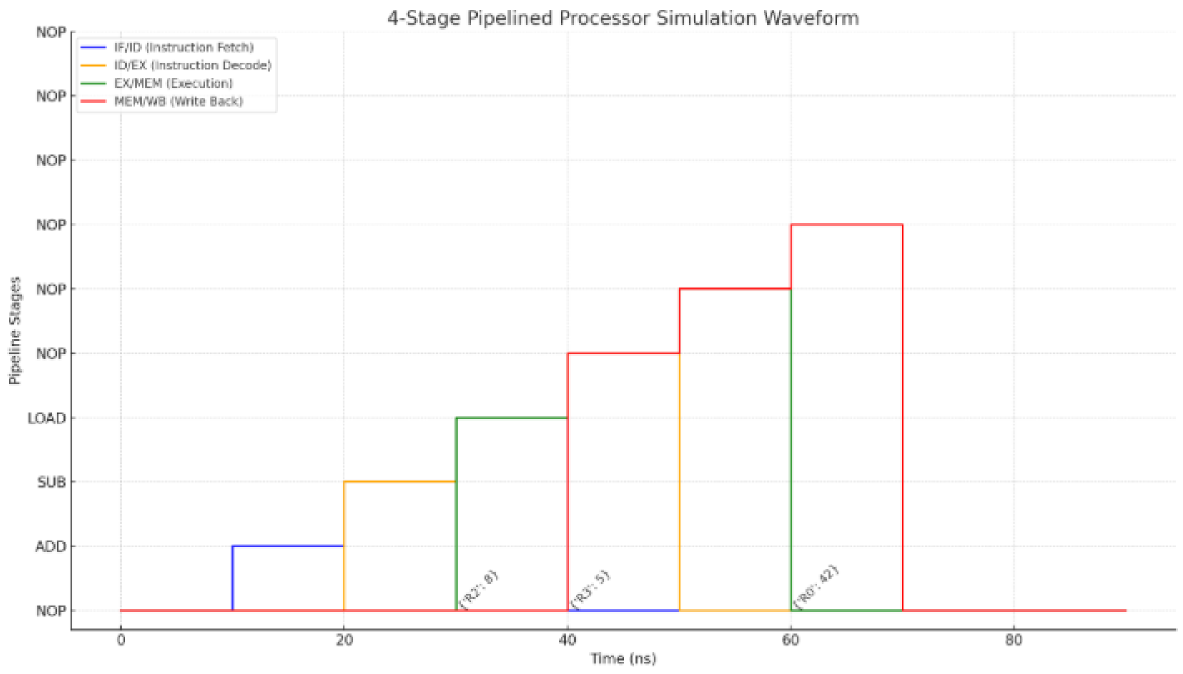
```
module testbench;

  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).