Task 3 - 4-Stage Pipelined Processor (Verilog)

Objective: Design a simple 4-stage pipelined processor with ADD, SUB, LOAD instructions.

Verilog Code (pipelined_processor.v)

```
module pipelined_processor (
    input clk,
    input reset,
    output reg [7:0] result
);
reg [15:0] instruction_mem [0:7];
reg [7:0] register_file [0:7];
reg [15:0] IF_ID, ID_EX, EX_MEM, MEM_WB;
integer i;
always @(posedge clk or posedge reset) begin
    if (reset) begin
         IF_ID <= 16'h0000;
         instruction_mem[0] <= {8'd10, 8'd20};
         instruction_mem[1] <= {8'd30, 8'd15};
         instruction_mem[2] <= {8'd50, 8'd25};
         instruction_mem[3] <= {8'd5,  8'd3};
         for (i = 4; i < 8; i = i + 1)
             instruction_mem[i] <= 16'h0000;
    end else begin
         IF_ID <= instruction_mem[0];
    end
end
always @(posedge clk) begin
    if (!reset)
         ID_EX <= IF_ID;
end
always @(posedge clk) begin
    if (!reset) begin
        if (ID_EX[15:8] < ID_EX[7:0])
            EX_MEM <= {8'd0, ID_EX[15:8] + ID_EX[7:0]};
        else
            EX_MEM <= {8'd0, ID_EX[15:8] - ID_EX[7:0]};
    end
end
always @(posedge clk) begin
    if (!reset) begin
         MEM_WB <= EX_MEM;
         register_file[0] <= MEM_WB[7:0];
         result <= MEM_WB[7:0];
    end
end
endmodule
```

Testbench Code (pipelined_processor_tb.v)

```
module pipelined_processor_tb;
  reg clk;
  reg reset;
  wire [7:0] result;
  pipelined_processor uut (
      .clk(clk),
      .reset(reset),
      .result(result)
  );
  initial begin
      clk = 0;
      forever #5 clk = ~clk;
  end
  initial begin
      reset = 1;
      #10;
      reset = 0;
      #100;
      $finish;
  end
  initial begin
      $monitor("Time %t: result = %d", $time, result);
  end
endmodule
```