Sethu Senthil

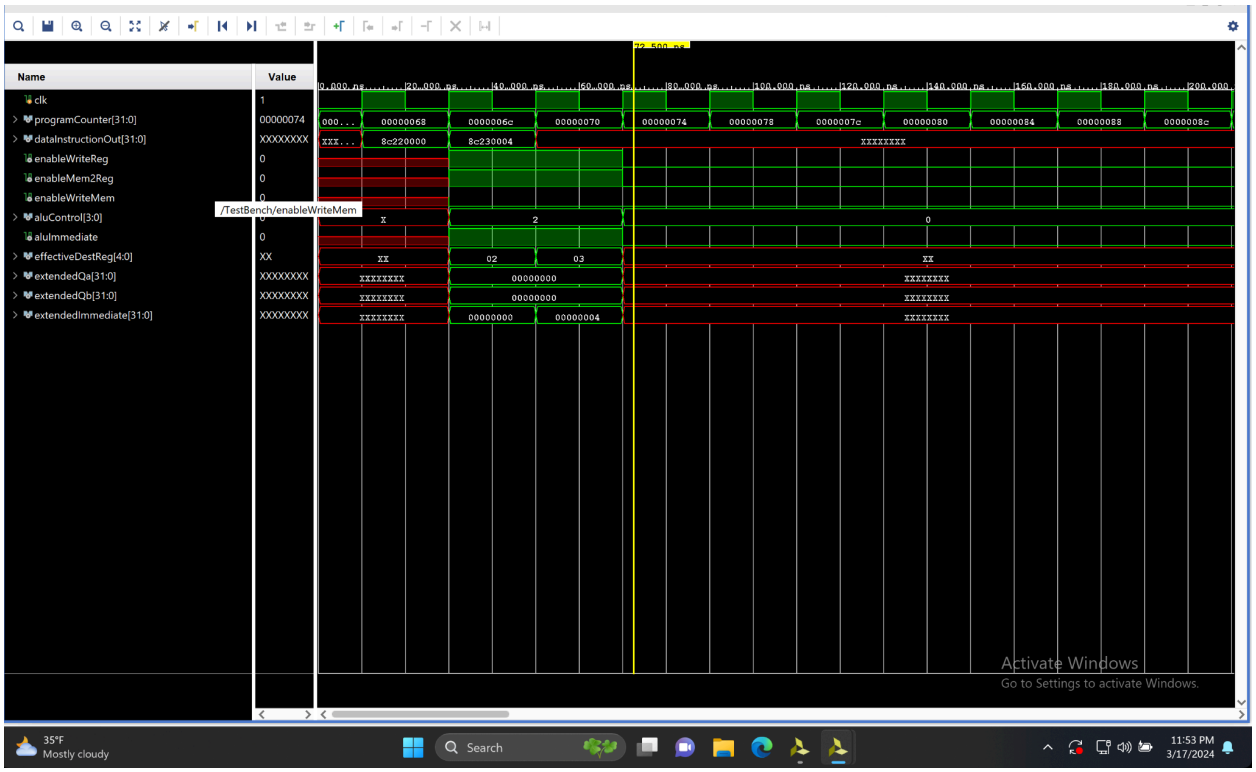
Lab 4

Section 1, CMPEN 331

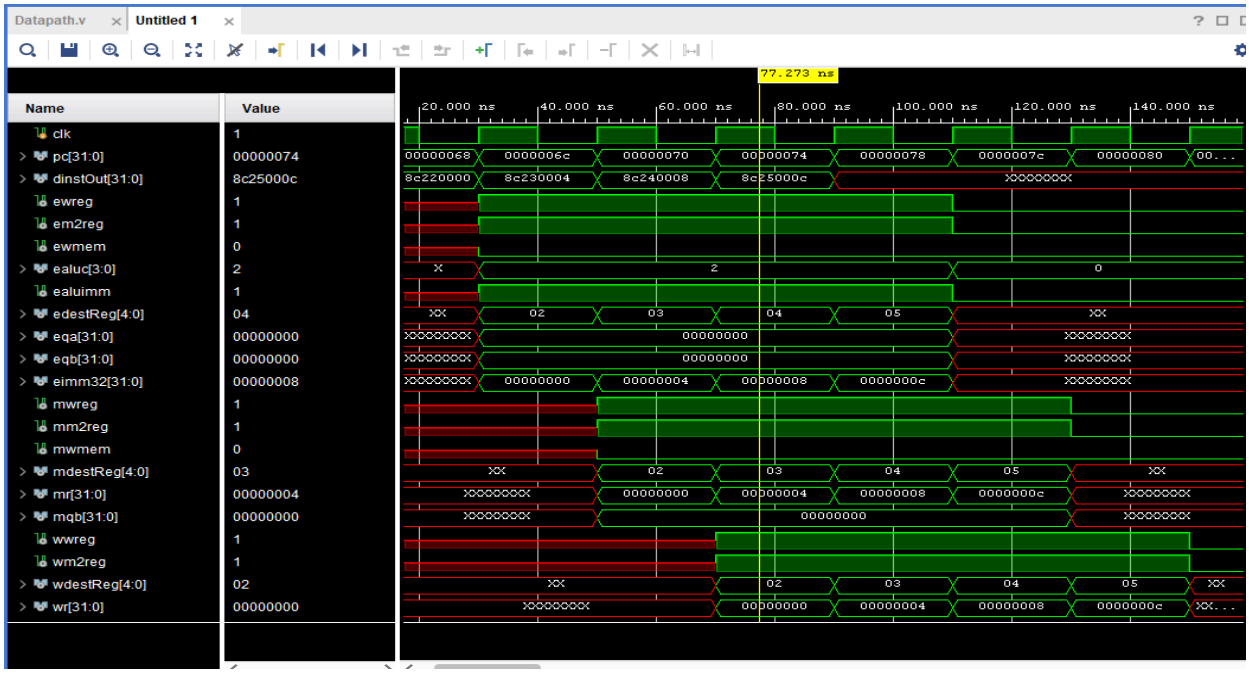
sns5787

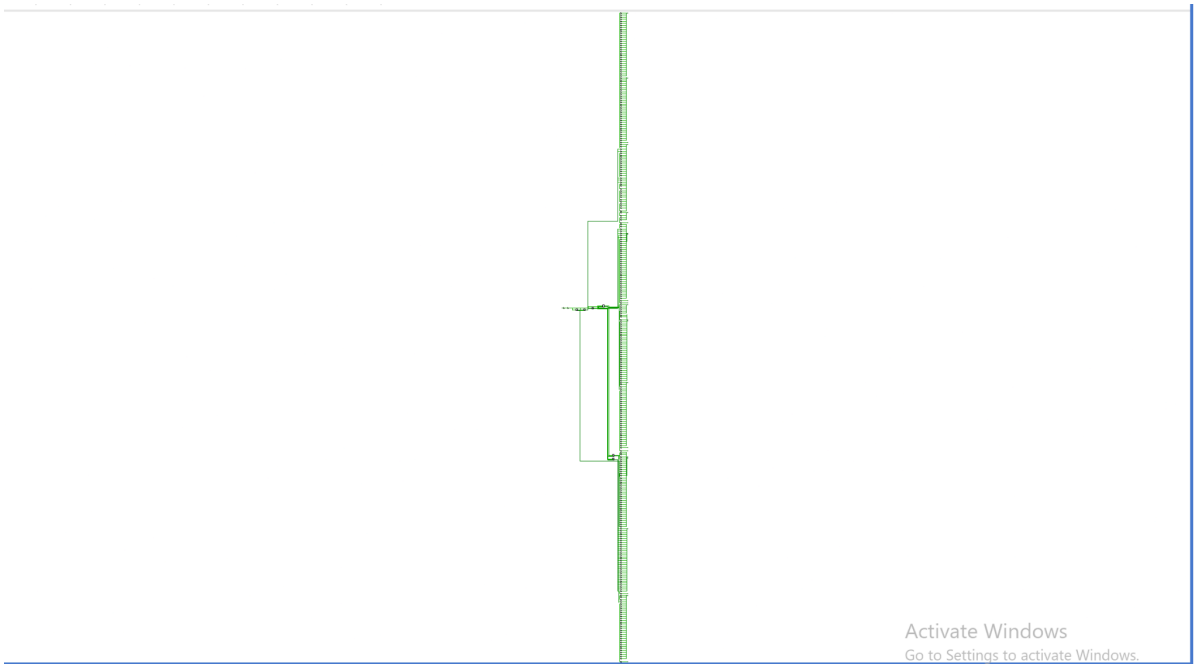
Images:

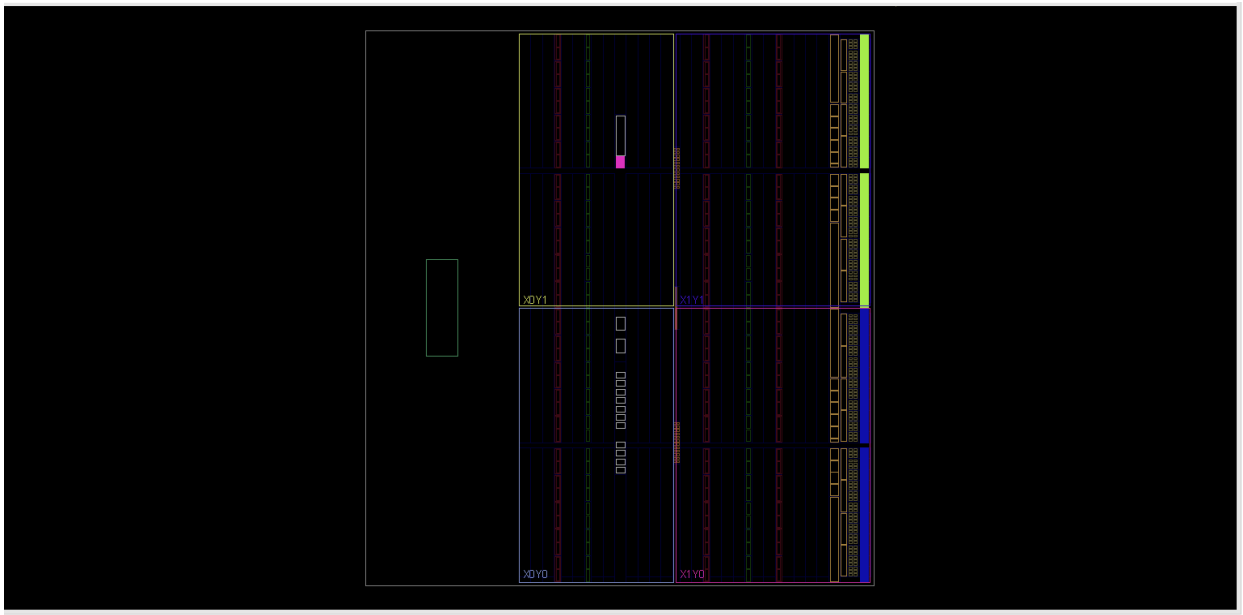
Lab 3 Waveform:

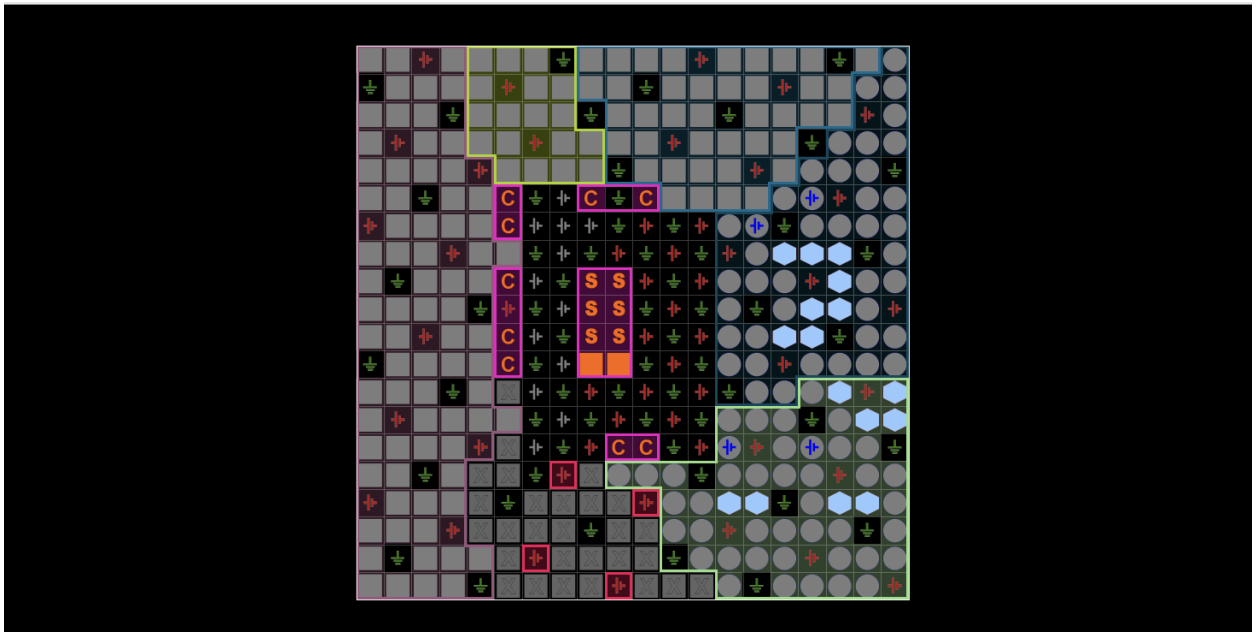


Lab 4 Waveform:




Activate Windows
Go to Settings to activate Windows.


X0Y1
X1Y1
X0Y0
X1Y0



Datapath Code:

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/07/2024 09:24:34 PM

// Design Name:

// Module Name: Datapath

// Project Name:

// Target Devices:
```

```
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module Datapath(
    // Lab 3 ------ IDEXEpipeline
    input clk,              // Clock signal
    output wire [31:0] pc,   // Program Counter
    output wire [31:0] dinstOut, // Data from the Instruction Memory
    output wire ewreg,        // Control signal for write enable in the EX stage
    output wire em2reg,       // Control signal for write enable in the Memory stage
    output wire ewmem,        // Control signal for write enable in the MEM stage
    output wire [3:0] ealuc,  // ALU control signal in the EX stage
    output wire ealuimm,      // ALU immediate in the EX stage
    output wire [4:0] edestReg, // Destination register address in the EX stage
    output wire [31:0] eqa,   // Data from source register A in the EX stage
    output wire [31:0] eqb,   // Data from source register B in the EX stage
```

```
    output wire [31:0] eimm32 // 32-bit immediate in the EX stage

   // Lab 4 ------ EXEMEM
    ,output wire mwreg,        // Control signal for write enable in the MEM stage
    output wire mm2reg,        // Control signal for write enable in the M2 stage (MEM-WB)
    output wire mwmem,         // Control signal for memory write enable in the MEM stage
    output wire [4:0] mdestReg, // Destination register address in the MEM stage
    output wire [31:0] mr,     // Data read from memory in the MEM stage
    output wire [31:0] mqb     // Data from source register B in the MEM stage

    // Lab 4 ------ MEMWB
    ,output wire wwreg,        // Control signal for write enable in the WB stage
    output wire wm2reg,        // Control signal for write enable in the M2 stage (WB)
    output wire [4:0] wdestReg, // Destination register address in the WB stage
    output wire [31:0] wr,     // Result to be written to the register file in the WB stage
    output wire [31:0] wdo     // Data read from memory or ALU result to be written to the
register file
);

    // Lab 3 ------ wires
    wire wreg;              // Control signal for write enable
    wire m2reg;             // Control signal for write enable (Memory stage)
    wire wmem;              // Control signal for memory write enable
```

```
wire aluimm;            // ALU immediate value
wire regrt;             // Control signal for selecting a register
wire [4:0] destReg;     // Destination register address
wire [31:0] qa;         // Data from source register A
wire [31:0] qb;         // Data from source register B
wire [31:0] imm32;      // 32-bit immediate value
wire [31:0] nextPc;     // Next program counter value
wire [31:0] instOut;    // Data from the Instruction Memory
wire [3:0] aluc;        // ALU control signal
wire [15:0] imm;        // Immediate value
wire [4:0] rs;          // Source register address
wire [4:0] rt;          // Source register address
wire [4:0] rd;          // Destination register address
wire [5:0] op;          // Operation code
wire [5:0] func;        // Function code

// Lab 4 ------ wires
wire [31:0] b;          // Data input to the ALU
wire [31:0] r;          // Result from the ALU
wire [31:0] mdo;        // Data read from memory or result from ALU


//connect components
```

```
    ProgramCounter IF_ProgramCounter_datapath(.clk(clk), .nextPc(nextPc), .pc(pc));
    pcAdder IF_pcAdder_datapath(.pc(pc), .nextPc(nextPc));
    InstructionMemory IF_InstructionMemory_dp(.pc(pc), .instOut(instOut));
    IFIDpipelineReg IFIDpipelineReg_datapath(.clk(clk), .instOut(instOut),
.dinstOut(dinstOut));
    ControlUnit ID_controlUnit_datapath(.op(op), .func(func), .wreg(wreg),
.m2reg(m2reg), .wmem(wmem), .aluc(aluc), .aluimm(aluimm), .regrt(regrt));
    RegrtMultiplexer ID_RegrtMultiplexer_datapath(.rt(rt), .rd(rd), .regrt(regrt),
.destReg(destReg));
    RegisterFile ID_RegisterFile_datapath(.rs(rs), .rt(rt), .qa(qa), .qb(qb));
    ImmediateExtender ID_ImmediateExtender_datapath(.imm(imm), .imm32(imm32));
    IDEXEpipeline IDEXEpipeline_datapath(
        .wreg(wreg),
        .m2reg(m2reg),
        .wmem(wmem),
        .aluc(aluc),
        .aluimm(aluimm),
        .destReg(destReg),
        .qa(qa),
        .qb(qb),
        .imm32(imm32),
        .clk(clk),
        .ewreg(ewreg),
```

```
        .em2reg(em2reg),
        .ewmem(ewmem),
        .ealuc(ealuc),
        .ealuimm(ealuimm),
        .edestReg(edestReg),
        .eqa(eqa),
        .eqb(eqb),
        .eimm32(eimm32)
    );

    ALU EXE_ALU_datapath(.eqa(eqa), .b(b), .ealuc(ealuc), .r(r));
    ALUMux EXE_ALUMux_datapath(.eqb(eqb), .eimm32(eimm32), .ealuimm(ealuimm),
.b(b));
    EXEMEMpipeline EXEMempipeline_datapath(
        .ewreg(ewreg),
        .em2reg(em2reg),
        .ewmem(ewmem),
        .edestReg(edestReg),
        .r(r),
        .eqb(eqb),
        .clk(clk),
        .mwreg(mwreg),
        .mm2reg(mm2reg),
```

```
        .mwmem(mwmem),
        .mdestReg(mdestReg),
        .mr(mr),
        .mqb(mqb)
    );
    DataMemory MEM_DataMemory_datapath(.mr(mr), .mqb(mqb), .mwmem(mwmem),
.clk(clk), .mdo(mdo));


    MEMWBpipeline MEMWBpipeline_datapath(
        .mwreg(mwreg),
        .mm2reg(mm2reg),
        .mdestReg(mdestReg),
        .mr(mr),
        .mdo(mdo),
        .clk(clk),
        .wwreg(wwreg),
        .wm2reg(wm2reg),
        .wdestReg(wdestReg),
        .wr(wr),
        .wdo(wdo)
    );
```

```
    assign op = dinstOut[31:26];
    assign func = dinstOut[5:0];
    assign rs = dinstOut[25:21];
    assign rt = dinstOut[20:16];
    assign rd = dinstOut[15:11];
    assign imm = dinstOut[15:0];
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:31:23 PM
// Design Name:
// Module Name: IF_ProgramCounter_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ProgramCounter(
    input clk,            // Input for clock signal, triggering positive edge updates.
    input [31:0] nextPc, // Input representing the next program counter value.
    output reg [31:0] pc // Output representing the current program counter value.
);

initial
    begin
        pc = 32'd100;      // Initializing the program counter to start at 100.
    end

always @(posedge clk)
    begin
        pc = nextPc;      // Updating the program counter on the positive clock edge.
    end
```

```
endmodule



`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:32:38 PM
// Design Name:
// Module Name: IF_pcAdder_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module pcAdder( // Module defining the program counter adder for CPU operation.
    input [31:0] pc, // Input representing the current program counter, 32 bits wide.
    output reg [31:0] nextPc // Output register representing the next program counter,
also 32 bits wide.
);

    always @(*) begin // Continuous updating of nextPc whenever any input signal
changes.
        nextPc <= pc + 32'b00000000000000000000000000000100; // Updating nextPc by
adding 4 to the input program counter.
    end // End of always block

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:43:21 PM
// Design Name:
// Module Name: IF_InstructionMemory_dp
```

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module InstructionMemory( // Module defining the instruction memory component within the CPU.
    input [31:0] pc, // Input representing the program counter.
    output reg [31:0] instOut // Output representing the instruction fetched from memory.
    );
       reg [31:0] memory [0:63]; // 32x64 array used to store instructions in memory.


       initial begin
        // Initializing memory with instructions
```

```
        // lw $v0, 00($at)
        memory[25] = {6'b100011, 5'b00001, 5'b00010, 5'b00000, 5'b00000, 6'b000000};
        // lw $v1, 04($at)
        memory[26] = {6'b100011, 5'b00001, 5'b00011, 5'b00000, 5'b00000, 6'b000100};
        // lw $a0, 08($at)
        memory[27] = {6'b100011, 5'b00001, 5'b00100, 5'b00000, 5'b00000, 6'b001000};
        // lw $a1, 12($at)
        memory[28] = {6'b100011, 5'b00001, 5'b00101, 5'b00000, 5'b00000, 6'b001100};
       end

    always @ (*) // Continuous updating of instruction output based on the program
counter bits 7 to 2.
        begin
            instOut <= memory[pc[7:2]];
        end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:46:22 PM
```

```
// Design Name:
// Module Name: IFIDpipelineReg_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module IFIDpipelineReg( //IFID pipeline
    input clk, //clock input needed as dinstOut only updates on the positive edge of clock.
    input [31:0] instOut, //input
    output reg [31:0] dinstOut //output
    );
```

```
    always @ (posedge clk) //always block that will only update dinstOut on the positive edge of the clock. dinstOut is to the instOut input of this module.
        begin
            dinstOut <= instOut;
        end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:48:40 PM
// Design Name:
// Module Name: ID_controlUnit_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ControlUnit( // Module defining the control unit of the CPU.
    // Inputs
    input [5:0] op, // Opcode input.
    input [5:0] func, // Function input.

    // Outputs
    output reg wreg, // Write to register file.
    output reg m2reg, // Memory to register.
    output reg wmem, // Write to memory.
    output reg [3:0] aluc, // ALU control.
    output reg aluimm, // ALU immediate source.
    output reg regrt // Destination register address.
);

    always @ (*) begin // Continuous updating of control signals based on op and func
values.
```

```
case (op) // Checking the opcode portion.
    6'b000000: // R-type instructions
    begin
        case (func) // Checking specific function values.
            6'b100000: // ADD instruction
            begin
                // Setting control signals for ADD instruction
                wreg = 1'b1; // Write to the register file
                m2reg = 1'b0; // Do not write to memory
                wmem = 1'b0; // Do not write to memory
                aluc = 4'b0010; // ALU operation for addition
                aluimm = 1'b0; // ALU source from registers
                regrt = 1'b0; // Destination register address
            end

            default: // Default behavior for unspecified func values
            begin
                // Set default control signals here
                // For example, you can set all signals to 0.
                wreg = 1'b0;
                m2reg = 1'b0;
                wmem = 1'b0;
                aluc = 4'b0000;
```

```
            aluimm = 1'b0;
            regrt = 1'b0;
        end
    endcase
end

6'b100011: // LW instruction
begin
    // Setting control signals for LW instruction
    wreg = 1'b1; // Write to the register file
    m2reg = 1'b1; // Write to memory
    wmem = 1'b0; // Do not write to memory
    aluc = 4'b0010; // ALU operation for addition
    aluimm = 1'b1; // ALU source from registers
    regrt = 1'b1; // Destination register address
end

default: // Default behavior for unspecified op values
begin
    // Set default control signals here for unspecified op values
    // For example, you can set all signals to 0.
    wreg = 1'b0;
    m2reg = 1'b0;
```

```
            wmem = 1'b0;
            aluc = 4'b0000;
            aluimm = 1'b0;
            regrt = 1'b0;
        end
    endcase
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:50:07 PM
// Design Name:
// Module Name: ID_RegrtMultiplexer_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RegrtMultiplexer(
    // Inputs
    input [4:0] rt,     // Input register value rt
    input [4:0] rd,     // Input register value rd
    input regrt,        // Control signal to select the output (0 for rd, 1 for rt)

    // Output
    output reg [4:0] destReg  // Output register value (selected based on the control
signal)
);

always @(*)
begin
    if (regrt == 0)
```

```
        destReg = rd;    // Select rd as the output when regrt is 0.
    else
        destReg = rt;    // Select rt as the output when regrt is 1.
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:54:38 PM
// Design Name:
// Module Name: ID_RegisterFile_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RegisterFile(
    // Inputs
    input [4:0] rs,     // Input for the source register (rs)
    input [4:0] rt,     // Input for the target register (rt)

    // Outputs
    output reg [31:0] qa,  // Output for the value stored in the source register
    output reg [31:0] qb   // Output for the value stored in the target register
);

reg [31:0] register [0:31]; // 32x32 array for registers (register file)

// Initialize all registers to 0
integer r;
initial begin
    for (r = 0; r <= 31; r = r + 1) begin
        register[r] = 0;  // Initialize each register to 0.
```

```
    end
end

always @ (*) // Continuous updating of qa and qb based on rs and rt inputs.
begin
    qa = register[rs];  // Output qa corresponds to the value stored in the source register
(rs).
    qb = register[rt];  // Output qb corresponds to the value stored in the target register
(rt).
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:56:22 PM
// Design Name:
// Module Name: ID_ImmediateExtender_dp
// Project Name:
// Target Devices:
```

```
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ImmediateExtender( // Module for immediate value extension.
    input [15:0] imm, // Input immediate value.
    output reg [31:0] imm32 // Output extended immediate value.
    );

    always @ (*) // Continuous update of imm32 value.
        begin
            imm32 = {{16{imm[15]}}, imm}; // Sign extension: Replicating the sign bit to fill
the upper 16 bits.
        end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 09:57:11 PM
// Design Name:
// Module Name: IDEXEpipeline_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module IDEXEpipeline(
```

```
    // Inputs
    input wreg,           // Control signal for write enable to the register file
    input m2reg,          // Control signal for write enable to the register file (M2 stage)
    input wmem,           // Control signal for memory write enable
    input [3:0] aluc,     // ALU control signal
    input aluimm,         // ALU immediate value
    input [4:0] destReg,  // Destination register address
    input [31:0] qa,      // Value from source register A
    input [31:0] qb,      // Value from source register B
    input [31:0] imm32,   // 32-bit immediate value
    input clk,            // Clock signal

    // Outputs
    output reg ewreg,     // Output for write enable signal
    output reg em2reg,    // Output for write enable signal (M2 stage)
    output reg ewmem,     // Output for memory write enable signal
    output reg [3:0] ealuc,   // Output for ALU control signal
    output reg ealuimm,   // Output for ALU immediate value
    output reg [4:0] edestReg,   // Output for destination register address
    output reg [31:0] eqa,  // Output for source register A value
    output reg [31:0] eqb,  // Output for source register B value
    output reg [31:0] eimm32 // Output for 32-bit immediate value
);
```

```
// On the positive edge of the clock, update the output signals with the input values.
always @ (posedge clk)
begin
    ewreg = wreg;       // Update write enable signal for register file writing.
    em2reg = m2reg;     // Update write enable signal for register file writing in M2 stage.
    ewmem = wmem;       // Update memory write enable signal.
    ealuc = aluc;       // Update ALU control signal.
    ealuimm = aluimm;   // Update ALU immediate value.
    edestReg = destReg; // Update destination register address.
    eqa = qa;           // Update source register A value.
    eqb = qb;           // Update source register B value.
    eimm32 = imm32;     // Update 32-bit immediate value.
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 10:04:27 PM
```

```
// Design Name:
// Module Name: EXE_ALU_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ALU(
    input [31:0] eqa,     // Input A for the ALU
    input [31:0] b,       // Input B for the ALU
    input [3:0] ealuc,    // ALU control signal
    output reg [31:0] r   // Output of the ALU
);
```

```
// ALU operation codes
// 0000 - AND
// 0001 - OR
// 0010 - ADD
// 0110 - SUBTRACT
// 0111 - SET LESS THAN
// 1100 - NOR
// 1111 - XOR

always @ (*)
begin
    case(ealuc)
        4'b0000: // AND
            begin
                r = eqa & b; // Perform AND operation between eqa and b.
            end

        4'b0001: // OR
            begin
                r = eqa | b; // Perform OR operation between eqa and b.
            end

        4'b0010: // ADD
```

```
        begin
            r = eqa + b; // Perform ADD operation between eqa and b.
        end

    4'b0110: // SUBTRACT
        begin
            r = eqa - b; // Perform SUBTRACT operation between eqa and b.
        end

    4'b1100: // NOR
        begin
            r = ~(eqa | b); // Perform NOR operation between eqa and b.
        end

    4'b1111: // XOR
        begin
            r = eqa ^ b; // Perform XOR operation between eqa and b.
        end
  endcase
end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 10:06:38 PM
// Design Name:
// Module Name: EXE_ALUMux_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module ALUMux(
    input [31:0] eqb,    // Input B data from the ALU
    input [31:0] eimm32, // Immediate value from the pipeline
    input ealuimm,       // Mux control signal
    output reg [31:0] b  // Output data selected by the Mux
);

always @(*) begin
    case(ealuimm)
        1'b0: // Select eqb as the output
            begin
                b <= eqb; // Select eqb when ealuimm is 0.
            end
        1'b1: // Select eimm32 as the output
            begin
                b <= eimm32; // Select eimm32 when ealuimm is 1.
            end
    endcase
end

endmodule

`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 10:08:06 PM
// Design Name:
// Module Name: EXEMempipeline_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module EXEMEMpipeline(
    input ewreg,            // Control signal for writing to the register file
```

```
    input em2reg,           // Control signal for writing to the register file (Memory stage)
    input ewmem,            // Control signal for writing to memory
    input [4:0] edestReg,   // Destination register address
    input [31:0] r,         // Result from the ALU
    input [31:0] eqb,       // Value from source register B
    input clk,              // Clock signal

    output reg mwreg,       // Output for write enable signal
    output reg mm2reg,      // Output for write enable signal (M2 stage)
    output reg mwmem,       // Output for memory write enable signal
    output reg [4:0] mdestReg, // Output for destination register address
    output reg [31:0] mr,   // Output for result from the ALU
    output reg [31:0] mqb   // Output for value from source register B
);

always @ (posedge clk)
begin
    mwreg = ewreg;          // Update write enable signal for register file writing.
    mm2reg = em2reg;        // Update write enable signal for register file writing in M2
stage.
    mwmem = ewmem;          // Update memory write enable signal.
    mdestReg = edestReg;    // Update destination register address.
    mr = r;                 // Update result from the ALU.
```

```
    mqb = eqb;              // Update value from source register B.
end


endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2024 10:09:58 PM
// Design Name:
// Module Name: MEM_DataMemory_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
//////////////////////////////////////////////////////////////////////////////////


module DataMemory(
    input [31:0] mr,     // Memory read address
    input [31:0] mqb,    // Data to be written to memory
    input mwmem,         // Memory write control signal
    input clk,           // Clock signal
    output reg [31:0] mdo // Data read from memory
);

// Define a data memory array with 64 words
reg [31:0] dataMemory [0:63];

// Initialize data memory with some values (words 0-9)
initial begin
    dataMemory[0] = 32'hA00000AA;
    dataMemory[1] = 32'h10000011;
    dataMemory[2] = 32'h20000022;
    dataMemory[3] = 32'h30000033;
    dataMemory[4] = 32'h40000044;
    dataMemory[5] = 32'h50000055;
```

```
    dataMemory[6] = 32'h60000066;
    dataMemory[7] = 32'h70000077;
    dataMemory[8] = 32'h80000088;
    dataMemory[9] = 32'h90000099;
end

always @(*) begin
    //  Set mdo to the value at the memory read address (bits 7:2 of mr)
    mdo = dataMemory[mr[7:2]]; // Read data from memory at the specified address.
end

always @(negedge clk) begin
  //If mwmem is 1, write the value in mqb to the memory at the read address
    if (mwmem == 1) begin
        dataMemory[mr[7:2]] <= mqb; // Write data to memory at the specified address.
    end
end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
```

```
// Engineer:
//
// Create Date: 04/07/2024 10:09:58 PM
// Design Name:
// Module Name: MEM_DataMemory_dp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module DataMemory(
    input [31:0] mr,     // Memory read address
    input [31:0] mqb,    // Data to be written to memory
    input mwmem,         // Memory write control signal
```

```
    input clk,          // Clock signal
    output reg [31:0] mdo // Data read from memory
);

// Define a data memory array with 64 words
reg [31:0] dataMemory [0:63];

// Initialize data memory with some values (words 0-9)
initial begin
    dataMemory[0] = 32'hA00000AA;
    dataMemory[1] = 32'h10000011;
    dataMemory[2] = 32'h20000022;
    dataMemory[3] = 32'h30000033;
    dataMemory[4] = 32'h40000044;
    dataMemory[5] = 32'h50000055;
    dataMemory[6] = 32'h60000066;
    dataMemory[7] = 32'h70000077;
    dataMemory[8] = 32'h80000088;
    dataMemory[9] = 32'h90000099;
end

always @(*) begin
    //  Set mdo to the value at the memory read address (bits 7:2 of mr)
```

```
    mdo = dataMemory[mr[7:2]]; // Read data from memory at the specified address.
end

always @(negedge clk) begin
  //If mwmem is 1, write the value in mqb to the memory at the read address
    if (mwmem == 1) begin
        dataMemory[mr[7:2]] <= mqb; // Write data to memory at the specified address.
    end
end

endmodule
```

Testbench:

```
module TestBench;

    reg clk;

    wire [31:0] pc;
    wire [31:0] dinstOut;
    wire ewreg;
    wire em2reg;
    wire ewmem;
    wire [3:0] ealuc;
```

```
wire ealuimm;
wire [4:0] edestReg;
wire [31:0] eqa;
wire [31:0] eqb;
wire [31:0] eimm32;


 wire mwreg;
 wire mm2reg;
 wire mwmem;
 wire [4:0] mdestReg;
 wire [31:0] mr;
 wire [31:0] mqb;

 wire wwreg;
 wire wm2reg;
 wire [4:0] wdestReg;
 wire [31:0] wr;
 wire [31:0] wdo;


initial begin
    clk <= 1'b0;
```

```
end

Datapath datapath(
    .clk(clk),
    .pc(pc),
    .dinstOut(dinstOut),
    .ewreg(ewreg),
    .em2reg(em2reg),
    .ewmem(ewmem),
    .ealuc(ealuc),
    .ealuimm(ealuimm),
    .edestReg(edestReg),
    .eqa(eqa),
    .eqb(eqb),
    .eimm32(eimm32),
    .mwreg(mwreg),
    .mm2reg(mm2reg),
    .mwmem(mwmem),
    .mdestReg(mdestReg),
    .mr(mr),
    .mqb(mqb),
        .wwreg(wwreg),
    .wm2reg(wm2reg),
```

```
        .wdestReg(wdestReg),
        .wr(wr),
        .wdo(wdo)
    );



    always begin
        #10;
        clk = ~clk;
    end

endmodule
```