Digital System Design using HDL Lab Report

Experiment 8d

Instruction decoder and control unit

Submitted by

Madhu Krishnan A P

M.Tech VLSI and Embedded Systems

Department of Electronics

Cochin University of Science and Technology

Contents

1 Module description

The instruction decoder and control units are responsible for generating control signals based on the opcode, funct3, and funct7 fields of the instruction and the Zero signal from the ALU. The unit produces various control outputs, including signals for branching (Branch), memory operations (MemWrite), register writes (RegWrite), immediate value selection (ImmSrc), and ALU control (ALUControl). It consists of a Main Decoder, which determines high-level control signals based on the opcode, and an ALU Decoder, which refines the ALU operation based on ALUOp, funct3, and funct7. The unit supports different instruction types like R-type, I-type, Load, Store, and Branch, with appropriate configurations for each. The PCSrc signal ensures conditional branching by combining the Branch and Zero signals. This design enables the processor to execute instructions effectively by orchestrating its components.

2 Block diagram

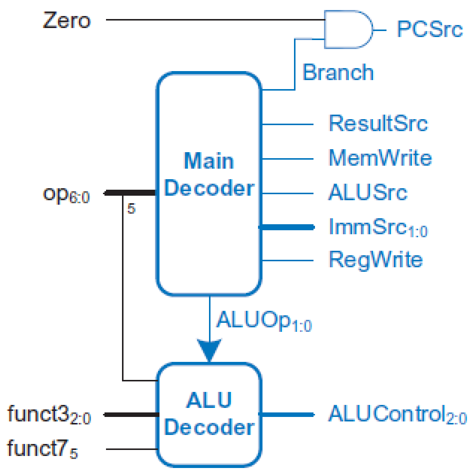

Figure 1: Instruction decoder and control unit

3 Inputs

- **opcode (7 bits):** Specifies the type of instruction (e.g., R-type, I-type, Load, Store, Branch).
- **funct3 (3 bits):** Used for additional decoding within the instruction type (e.g., determining ALU operations like ADD, SUB, AND, etc.).
- **funct7 (7 bits):** Provides further instruction-specific information (e.g., distinguishing between ADD and SUB).
- **Zero (1 bit):** Indicates if the result of the ALU operation is zero (used for conditional branching).

4 Outputs

- **Branch (1 bit):** Enables branching logic when set.
- **MemWrite (1 bit):** Enables writing to memory.
- **ALUSrc (1 bit):** Selects between immediate data or register data as the second operand for the ALU.
- **ImmSrc (2 bits):** Determines the format of the immediate value (e.g., I-type, S-type, or B-type).

- **RegWrite (1 bit):** Enables writing to a register.
- **ResultSrc (2 bits):** Selects the source of data to write back to the register (e.g., ALU result, memory data).
- **ALUControl (3 bits):** Specifies the operation to be performed by the ALU (e.g., ADD, SUB, AND, OR, etc.).
- **PCSrc (1 bit):** Determines the next program counter (PC) value for branching.

5 Internal Signal

- **ALUOp (2 bits):** Intermediate signal from the main decoder that helps determine the ALU operation in the ALU Decoder.

6 Functionality

6.1 Main Decoder

The main decoder generates the control signals based on the **opcode**:

6.1.1 R-type (opcode = 7'b0110011):

- Register-to-register operations.
- Enables **RegWrite**.
- ALU operations determined by **funct3** and **funct7**.
- **ALUOp = 2'b10** for R-type.

6.1.2 I-type (opcode = 7'b0010011):

- Immediate ALU operations.
- Enables **RegWrite** and **ALUSrc**.
- **ALUOp = 2'b10**.

6.1.3 Load (opcode = 7'b0000011):

- Loads data from memory into a register.
- Enables **RegWrite**, **ALUSrc**, and selects memory data as the result (**ResultSrc = 2'b01**).

6.1.4 Store (opcode = 7'b0100011):

- Stores data from a register into memory.
- Enables **MemWrite** and **ALUSrc**.
- Immediate value is of S-type (**ImmSrc = 2'b01**).

6.1.5 Branch (opcode = 7'b1100011):

- Conditional branching.
- Enables **Branch**.
- Immediate value is of B-type (**ImmSrc = 2'b10**).
- **ALUOp = 2'b01** to perform subtraction (used for branch comparison).

6.2 ALU Decoder

The ALU Decoder generates the **ALUControl** signal based on **ALUOp**, **funct3**, and **funct7**:

- **ALUOp = 2'b00:** Default operation (**ADD**) for load/store instructions.
- **ALUOp = 2'b01:** Subtraction (used for branch comparison).
- **ALUOp = 2'b10:** R-type or I-type operations. The specific operation is determined by **funct3** and **funct7**:
  - **funct3 = 3'b000:** ADD (default) or SUB (if **funct7[5] = 1**).
  - **funct3 = 3'b111:** AND.
  - **funct3 = 3'b110:** OR.
  - **funct3 = 3'b100:** XOR.
  - **funct3 = 3'b001:** SLL (Shift Left Logical).
  - **funct3 = 3'b101:** SRL (logical shift right) or SRA (arithmetic shift right) depending on **funct7[5]**.

6.3 PCSrc Logic

The **PCSrc** signal is set to **1** when both the **Branch** signal and **Zero** signal are high, indicating a successful branch condition.

7 Verilog description

```
// Control Unit Module
module ControlUnit (
    input  logic [6:0]  opcode,      // 7-bit opcode
    input  logic [2:0]  funct3,      // 3-bit funct3
    input  logic [6:0]  funct7,      // 7-bit funct7
    input  logic        Zero,        // ALU Zero signal
    output logic        Branch,      // Branch signal
    output logic        MemWrite,    // Memory write signal
    output logic        ALUSrc,      // ALU source selection
    output logic [1:0]  ImmSrc,      // Immediate source selection
    output logic        RegWrite,    // Register write enable
    output logic [1:0]  ResultSrc,   // Result source selection
    output logic [2:0]  ALUControl,  // ALU control signal
    output logic        PCSrc        // PC source for branching
);

    // Internal signal for ALUOp from Main Decoder to ALU Decoder
    logic [1:0] ALUOp;

    // Main Decoder
    always_comb begin
        // Default values
        Branch     = 0;
        MemWrite   = 0;
        ALUSrc     = 0;
        RegWrite   = 0;
        ImmSrc     = 2'b00;
        ResultSrc  = 2'b00;
        ALUOp      = 2'b00;
        case (opcode)
            7'b0110011: begin // R-type
                RegWrite  = 1;
                ALUOp     = 2'b10;
            end
            7'b0010011: begin // I-type (ALU Immediate)
                RegWrite  = 1;
                ALUSrc    = 1;
                ALUOp     = 2'b10;
            end
            7'b0000011: begin // Load
                RegWrite  = 1;
                ALUSrc    = 1;
                ResultSrc = 2'b01;
            end
            7'b0100011: begin // Store
                MemWrite  = 1;
                ALUSrc    = 1;
                ImmSrc    = 2'b01;
            end
            7'b1100011: begin // Branch
                Branch    = 1;
                ALUOp     = 2'b01;
                ImmSrc    = 2'b10;
            end
            default: ; // Default case does nothing
        endcase
    end

    // ALU Decoder
    always_comb begin
        case (ALUOp)
            2'b00: ALUControl = 3'b000; // Default: Add (e.g., for
                load/store)
            2'b01: ALUControl = 3'b001; // Subtract (for branches)
            2'b10: begin
                case (funct3)
                    3'b000: ALUControl = (funct7[5]) ? 3'b001 :
                        3'b000; // SUB if funct7[5]=1, ADD otherwise
                    3'b111: ALUControl = 3'b111; // AND
                    3'b110: ALUControl = 3'b110; // OR
                    3'b100: ALUControl = 3'b100; // XOR
                    3'b001: ALUControl = 3'b010; // SLL (Shift Left
                        Logical)
                    3'b101: ALUControl = (funct7[5]) ? 3'b011 :
                        3'b101; // SRL/SRA
                    default: ALUControl = 3'b000; // Default to ADD
                endcase
            end
            default: ALUControl = 3'b000; // Default to ADD
        endcase
    end
    // PCSrc logic (Branch AND Zero)
    assign PCSrc = Branch & Zero;
endmodule
```

8 Testbench

```
// Testbench for Control Unit
module ControlUnit_tb;

    // Testbench signals
    logic [6:0]  opcode;
    logic [2:0]  funct3;
    logic [6:0]  funct7;
    logic        Zero;
    logic        Branch;
    logic        MemWrite;
    logic        ALUSrc;
    logic [1:0]  ImmSrc;
    logic        RegWrite;
    logic [1:0]  ResultSrc;
    logic [2:0]  ALUControl;
    logic        PCSrc;

    // Instantiate the ControlUnit
    ControlUnit uut (
        .opcode(opcode),
        .funct3(funct3),
        .funct7(funct7),
        .Zero(Zero),
        .Branch(Branch),
        .MemWrite(MemWrite),
        .ALUSrc(ALUSrc),
        .ImmSrc(ImmSrc),
        .RegWrite(RegWrite),
        .ResultSrc(ResultSrc),
        .ALUControl(ALUControl),
        .PCSrc(PCSrc)
    );

    // Test procedure
    initial begin

        // Test 1: R-type instruction (ADD)
        opcode = 7'b0110011; funct3 = 3'b000; funct7 = 7'b0000000;
            Zero = 0;
        #10;

        // Test 2: R-type instruction (SUB)
        opcode = 7'b0110011; funct3 = 3'b000; funct7 = 7'b0100000;
            Zero = 0;
        #10;

        // Test 3: I-type ALU immediate (ADDI)
        opcode = 7'b0010011; funct3 = 3'b000; funct7 = 7'b0000000;
            Zero = 0;
        #10;

        // Test 4: Load instruction (LW)
        opcode = 7'b0000011; funct3 = 3'b010; funct7 = 7'b0000000;
            Zero = 0;
        #10;

        // Test 5: Store instruction (SW)
        opcode = 7'b0100011; funct3 = 3'b010; funct7 = 7'b0000000;
            Zero = 0;
        #10;

        // Test 6: Branch instruction (BEQ) with Zero=1
        opcode = 7'b1100011; funct3 = 3'b000; funct7 = 7'b0000000;
            Zero = 1;
        #10;

        // Test 7: Branch instruction (BEQ) with Zero=0
        opcode = 7'b1100011; funct3 = 3'b000; funct7 = 7'b0000000;
            Zero = 0;
        #10;

        $finish;
    end

endmodule
```

The testbench for the ControlUnit module is designed to verify the functionality of the control signals generated for various instruction types. It applies different values to the opcode, funct3, funct7, and Zero inputs to simulate a range of instructions such as R-type (ADD, SUB), I-type (ADDI), Load, Store, and Branch instructions. Each test case checks the expected behaviour of the control signals like ALUControl, RegWrite, Branch, MemWrite, PCSrc, and others based on the instruction type. The results of each test are displayed, and the simulation concludes after all tests are completed.

9 Simulation results

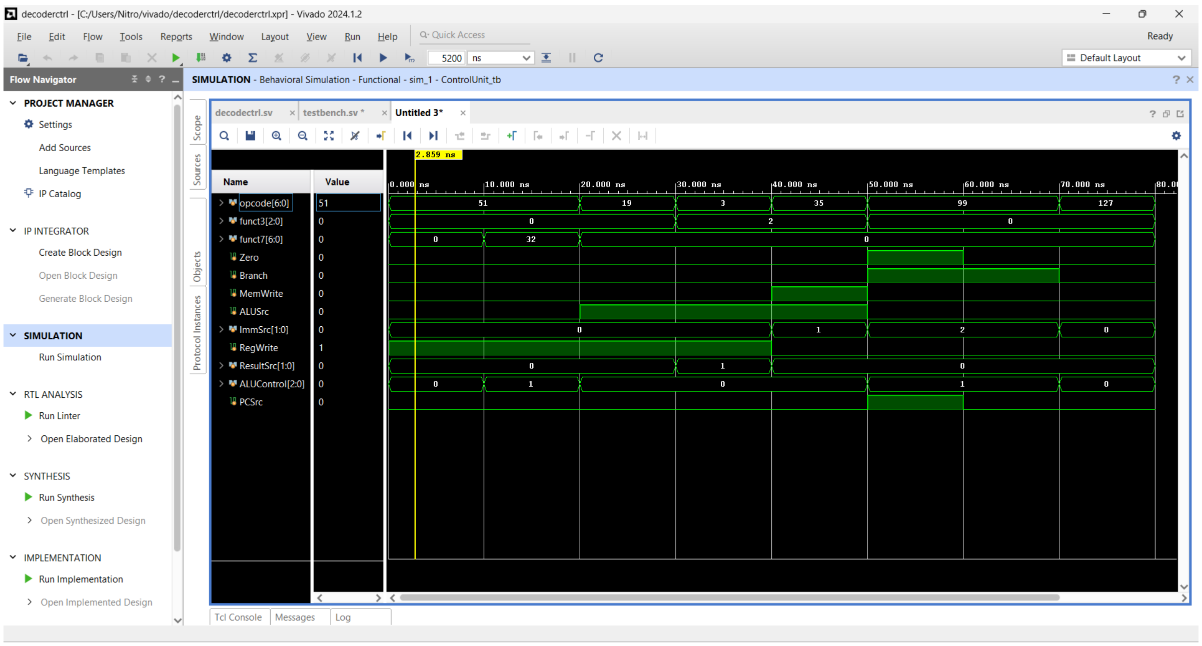

Figure 2: Decoder and control unit: Simulation result