IEEE – Digital Electronics

Cairo University Branch

Workshop Spring 2025

Single Cycle RV-32I Processor

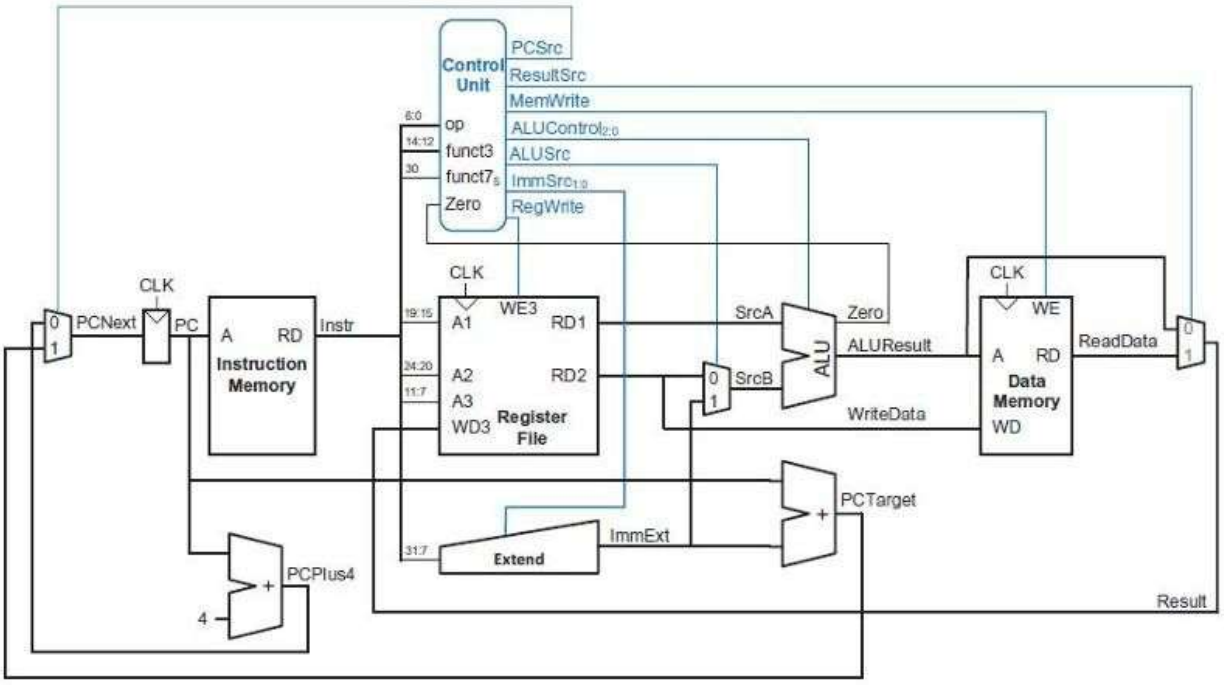


Table of Contents

Main Modules **3**

1. **ALU** **3**
2. **Program Counter** **4**
3. **Instruction Memory** **5**
4. **Register File** **6**
5. **Data Memory** **7**
6. **Control Unit** **8**
   - **6.1. ALU Decoder** **8**
   - **6.2. Main Decoder** **8**
   - **6.3. Control Unit** **11**

Small Modules **12**

1. **Sign Extender** **12**

Top Module and Simulations **13**

1. **Top Module : RISC_V** **13**
2. **Simulation Results** **15**

Main Modules

1. ALU

```
module ALU(
    input [31:0] SrcA,
    input [31:0] SrcB,
    input [2:0]  ALUControl,
    output reg [31:0]  ALUResult,
    output       Zero
);

assign Zero = ~|ALUResult; // zero flag by nor reduction

always @(*) begin
    case (ALUControl)
        0: ALUResult = SrcA + SrcB;
        1: ALUResult = SrcA << SrcB;
        2: ALUResult = SrcA - SrcB;
        3: ALUResult = 32'b0;
        4: ALUResult = SrcA ^ SrcB;
        5: ALUResult = SrcA >>> SrcB;
        6: ALUResult = SrcA | SrcB;
        7: ALUResult = SrcA & SrcB;
        default: ALUResult = 32'b0;
    endcase
end

endmodule
```

2. Program Counter

```
module ProgramCounter(
    input               PCsrc,
    input               clk,
    input               areset,
    input               load,
    input [31:0]        ImmExt,
    output reg [31:0] PC
);

reg [31:0] PCNext = 0;

always @(*) begin
    PCNext = PCsrc? (PC + ImmExt) : (PC + 4);
end

always @(posedge clk, negedge areset) begin
    if (!areset)
        PC <= 32'b0;
    else if (load)
        PC <= PCNext;
    else
        PC <= PC;
end

endmodule
```

3. Instruction Memory

```
module InstructionMemory #(parameter WIDTH = 32, DEPTH = 64)(
    input [31:0]  PC,
    output reg [31:0] Instr
);

reg [WIDTH - 1:0] InstrMem [0:DEPTH - 1];

initial begin
    $readmemh("program.txt", InstrMem);
end

always @(*) begin
    Instr = InstrMem[PC[31:2]];
end


endmodule
```

4. Register File

```
module RegisterFile #(parameter WIDTH = 32, DEPTH = 32)(
    input clk, rst_n,
    input [4:0] A1, A2, A3,
    input [WIDTH - 1:0] WD3,
    input WE3,
    output [WIDTH - 1:0] RD1,RD2
);

reg [WIDTH - 1:0] register [0:DEPTH - 1];

// read asynchronously
assign RD1 = register[A1];
assign RD2 = register[A2];

integer i;
always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
        for (i = 0;i < DEPTH;i = i + 1) begin
            register[i] <= {WIDTH{1'b0}};
        end
    end
    else if (WE3)
        register[A3] <= WD3;

end

endmodule
```

5. Data Memory

```
module DataMemory #(parameter WIDTH = 32, DEPTH = 64)(
    input [WIDTH - 1:0] A,WD,
    input clk,WE,
    output [WIDTH - 1:0] RD
);

reg [WIDTH - 1:0] dataMem [0:DEPTH - 1];

assign RD = dataMem[A];


always @(posedge clk) begin
    if (WE)
        dataMem[A] <= WD;
    else
        dataMem[A] <= dataMem[A];
end

endmodule
```

6. Control Unit

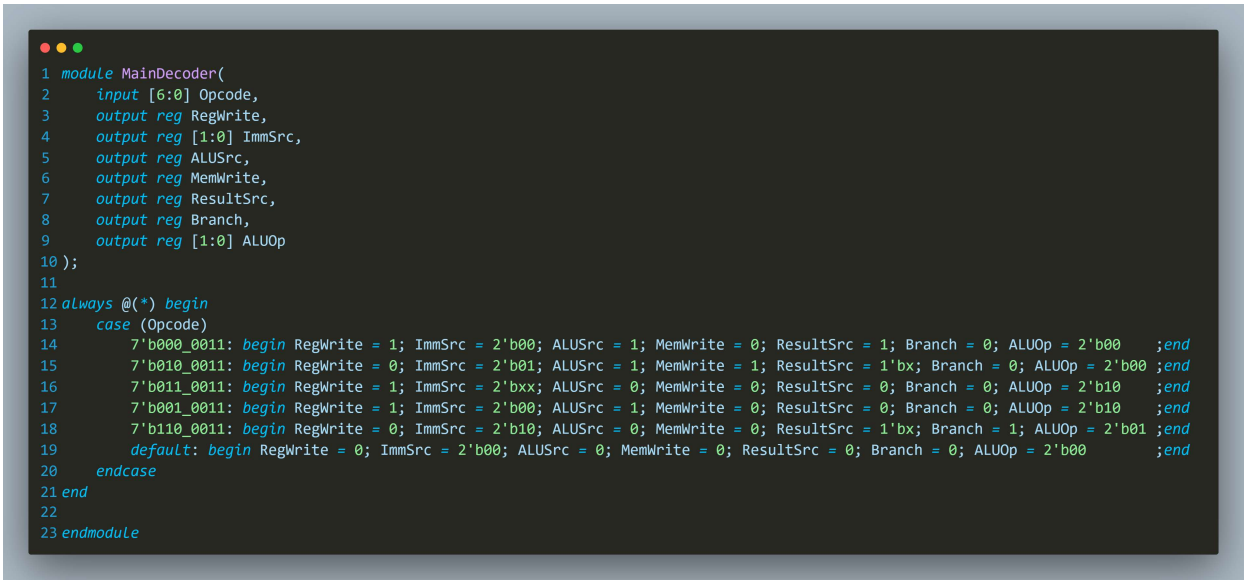
6.1. ALU Decoder

```
module ALUDecoder(
    input [1:0] ALUOp,
    input [2:0] funct3,
    input funct7_5, OP_5,
    output reg [2:0] ALUControl
);

always @(*) begin
    case (ALUOp)
        0: ALUControl = 3'b0;
        1: ALUControl = 3'b010;
        2: begin
            case (funct3)
                0: begin
                    case ({OP_5,funct7_5})
                        0,1,2: ALUControl = 3'b0;
                        3: ALUControl = 3'b010;
                        default: ALUControl = 0;
                    endcase
                end
                1: ALUControl = funct3;
                4: ALUControl = funct3;
                5: ALUControl = funct3;
                6: ALUControl = funct3;
                7: ALUControl = funct3;
                default: ALUControl = 0;
            endcase
        end
        default: ALUControl = 0;
    endcase
end

endmodule
```

6.2. Main Decoder



```
module MainDecoder(
    input [6:0] Opcode,
    output reg RegWrite,
    output reg [1:0] ImmSrc,
    output reg ALUSrc,
    output reg MemWrite,
    output reg ResultSrc,
    output reg Branch,
    output reg [1:0] ALUOp
);

always @(*) begin
    case (Opcode)
        7'b000_0011: begin RegWrite = 1; ImmSrc = 2'b00; ALUSrc = 1; MemWrite = 0; ResultSrc = 1; Branch = 0; ALUOp = 2'b00    ;end
        7'b010_0011: begin RegWrite = 0; ImmSrc = 2'b01; ALUSrc = 1; MemWrite = 1; ResultSrc = 1'bx; Branch = 0; ALUOp = 2'b00 ;end
        7'b011_0011: begin RegWrite = 1; ImmSrc = 2'bxx; ALUSrc = 0; MemWrite = 0; ResultSrc = 0; Branch = 0; ALUOp = 2'b10    ;end
        7'b001_0011: begin RegWrite = 1; ImmSrc = 2'b00; ALUSrc = 1; MemWrite = 0; ResultSrc = 0; Branch = 0; ALUOp = 2'b10    ;end
        7'b110_0011: begin RegWrite = 0; ImmSrc = 2'b10; ALUSrc = 0; MemWrite = 0; ResultSrc = 1'bx; Branch = 1; ALUOp = 2'b01 ;end
        default: begin RegWrite = 0; ImmSrc = 2'b00; ALUSrc = 0; MemWrite = 0; ResultSrc = 0; Branch = 0; ALUOp = 2'b00        ;end
    endcase
end

endmodule
```

6.3. Control Unit

```
module ControlUnit(
    input [6:0] OP,
    input [2:0] funct3,
    input funct7_5,
    input Zero,
    input Sign,
    output reg PCSrc,
    output ResultSrc, MemWrite,
    output [2:0] ALUControl,
    output ALUSrc,
    output [1:0] ImmSrc,
    output RegWrite
);

wire Branch;
wire [1:0] ALUOp;
wire beq,bnq,blt;


MainDecoder mainDecoder (
    OP, RegWrite, ImmSrc, ALUSrc, MemWrite, ResultSrc, Branch, ALUOp
);

ALUDecoder aluDecoder (
    ALUOp,funct3, funct7_5, OP[5], ALUControl
);

assign beq = Zero & Branch;
assign bnq = ~Zero & Branch;
assign blt = Sign & Branch;

always @(*) begin
    case (funct3)
        3'b000: PCSrc = beq;
        3'b001: PCSrc = bnq;
        3'b100: PCSrc = blt;
        default: PCSrc = 0;
    endcase
end


endmodule
```

Small Modules

1. Sign Extender

```
module sign_extend(
    input [31:7]      Instr,
    input [1:0]       ImmSrc,
    output reg [31:0] ImmExt
);

always @(*) begin
    case (ImmSrc)
        0: ImmExt = {{20{Instr[31]}}, Instr[31:20]};
        1: ImmExt = {{20{Instr[31]}}, Instr[31:25], Instr[11:7]};
        2: ImmExt = {{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};
        3: ImmExt = 32'b0;
        default: ImmExt = 32'b0;
    endcase
end

endmodule
```

2. MUX

```
module mux_2to1(
    input  [31:0] In1, In2,
    input         Sel,
    output [31:0] Out
);

assign Out = Sel? In2: In1;

endmodule
```

Top Module and Simulations

1. Top Module : RISC_V

```
module RISCV(
    input clk, rst_n
);


wire [31:0] SrcA, SrcB, Instr, ReadData, ALUresult, RD2, PC, ImmExt, Result;
wire [2:0] ALUcontrol;
wire [1:0] ImmSrc;
wire Zero, ALUSrc, sign , load , PCSrc, RegWrite, MemWrite, Resultsrc;

assign SrcB = ALUSrc? ImmExt : RD2;
assign Result = Resultsrc? ReadData : ALUresult;
assign sign = ALUresult[31];


ProgramCounter pc (
    PCSrc, clk, rst_n, 1'b1, ImmExt, PC
);

InstructionMemory inst_mem (
    PC, Instr
);

RegisterFile reg_file (
    clk ,rst_n,Instr[19:15], Instr[24:20], Instr[11:7], Result, RegWrite,
    SrcA, RD2
);

ALU alu (
    SrcA, SrcB, ALUcontrol, ALUresult, Zero
);


DataMemory data_mem (
    ALUresult, RD2, clk, MemWrite, ReadData
);

ControlUnit CU (
    Instr[6:0] , Instr[14:12], Instr[30], Zero, sign, PCSrc, Resultsrc,
    MemWrite, ALUcontrol, ALUSrc, ImmSrc, RegWrite
);
```

```
42
43 sign_extend sign_extender (
44     Instr[31:7], ImmSrc, ImmExt
45 );
46
47
48
49 endmodule
```

2. Simulation Results

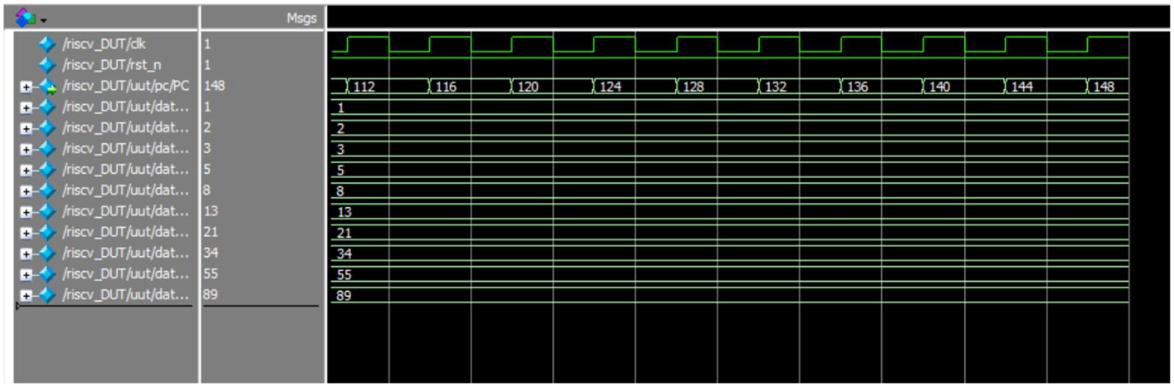


Figure 1 - FIBONACCI series