COMPUTER ORGANIZATION LAB4

REPORT

109550083 楊皓宇 109550121 温柏萱

Decoder

```
`timescale 1ns/1ps

module Decoder(
    input   [7-1:0]     instr_i,
    output reg              RegWrite,
    output reg              Branch,
    output reg              Jump,
    output reg              WriteBack1,
    output reg              WriteBack0,
    output reg              MemRead,
    output reg              MemWrite,
    output reg              ALUSrcA,
    output reg              ALUSrcB,
    output reg [2-1:0]      ALUOp
);

/* Write your code HERE */

always @(instr_i) begin
    case (instr_i)
        7'b0110011: begin // R-type
            RegWrite = 1'b1;
            Branch = 1'b0;
            Jump = 1'b0;
            WriteBack1 = 1'b0;
            WriteBack0 = 1'b0;
            MemRead = 1'b0;
```

```
        MemWrite = 1'b0;
        ALUSrcA = 1'b0;
        ALUSrcB = 1'b0;
        ALUOp = 2'b10;
        end
    7'b0010011: begin // I-type
        RegWrite = 1'b1;
        Branch = 1'b0;
        Jump = 1'b0;
        WriteBack1 = 1'b0;
        WriteBack0 = 1'b0;
        MemRead = 1'b0;
        MemWrite = 1'b0;
        ALUSrcA = 1'b0;
        ALUSrcB = 1'b1;
        ALUOp = 2'b10; // IDK
    end
    7'b0000011: begin // Load
        RegWrite = 1'b1;
        Branch = 1'b0;
        Jump = 1'b0;
        WriteBack1 = 1'b0;
        WriteBack0 = 1'b1;
        MemRead = 1'b1;
        MemWrite = 1'b0;
        ALUSrcA = 1'b0;
        ALUSrcB = 1'b1;
        ALUOp = 2'b00; // IDK
    end
    7'b0100011: begin // Store
        RegWrite = 1'b0;
        Branch = 1'b0;
        Jump = 1'b0;
        WriteBack1 = 1'b0;
        WriteBack0 = 1'b0;
        MemRead = 1'b0;
        MemWrite = 1'b1;
        ALUSrcA = 1'b0;
        ALUSrcB = 1'b1;
        ALUOp = 2'b00; // IDK
    end
    7'b1100011: begin // Branch (BEQ)
        RegWrite = 1'b0;
```

```
            Branch = 1'b1;
            Jump = 1'b0;
            WriteBack1 = 1'b0;
            WriteBack0 = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            ALUSrcA = 1'b0;
            ALUSrcB = 1'b0;
            ALUOp = 2'b01; // IDK
        end
        7'b1101111: begin // JAL
            RegWrite = 1'b1;
            Branch = 1'b0;
            Jump = 1'b1;
            WriteBack1 = 1'b1;
            WriteBack0 = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            ALUSrcA = 1'b0;
            ALUSrcB = 1'b0;
            ALUOp = 2'b00; // IDK
        end
        7'b1100111: begin // JALR
            RegWrite = 1'b1;
            Branch = 1'b0;
            Jump = 1'b1;
            WriteBack1 = 1'b1;
            WriteBack0 = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            ALUSrcA = 1'b1;
            ALUSrcB = 1'b0;
            ALUOp = 2'b00; // IDK
        end
    endcase
end

endmodule
```

Based on the slides, we can determine the values of each signal as the following table.

instructions	R-type	I-type	Load	Store	Branch	JAL	JALR
RegWrite	1	1	1	0	0	1	1
Branch	0	0	0	0	1	0	0
Jump	0	0	0	0	0	1	1
WriteBack1	0	0	0	0	0	1	1
WriteBack0	0	0	1	0	0	0	0
MemRead	0	0	1	0	0	0	0
MemWrite	0	0	0	1	0	0	0
ALUSrcA	0	0	0	0	0	0	1
ALUSrcB	0	1	1	1	0	0	0
ALUOp	10	10	00	00	01	00	00

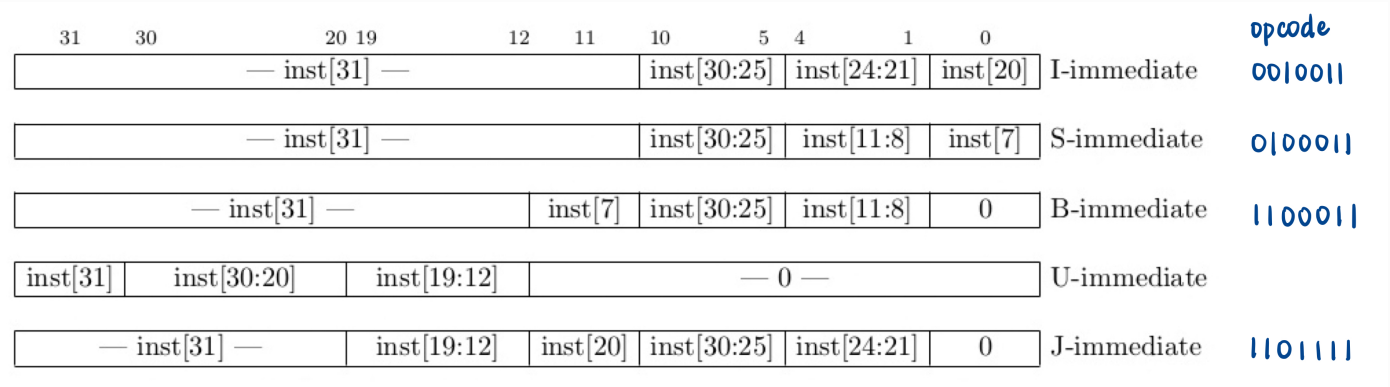
Immediate Generator

```
`timescale 1ns/1ps

module Imm_Gen(
    input  [31:0] instr_i,
    output reg[31:0] Imm_Gen_o
);

//Internal Signals
wire    [7-1:0] opcode;
wire    [2:0]   func3;
wire    [3-1:0] Instr_field;

assign opcode = instr_i[6:0];
assign func3  = instr_i[14:12];
integer i;

always @(*) begin
    // convert to case
```

```
    case (opcode)
        7'b0110011: begin // R
            Imm_Gen_o = 32'b0;
        end
        7'b0010011: begin // I
            Imm_Gen_o[10:0] = 32'b0;
            for (i = 11; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
        7'b0000011: begin // Load
            Imm_Gen_o[10:0] = instr_i[30:20];
            for (i = 11; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
        7'b0100011: begin // Store
            Imm_Gen_o[4:0] = instr_i[11:7];
            Imm_Gen_o[10:5] = instr_i[30:25];
            for (i = 11; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
        7'b1100011: begin // Branch
            Imm_Gen_o[0] = 1'b0;
            Imm_Gen_o[4:1] = instr_i[11:8];
            Imm_Gen_o[10:5] = instr_i[30:25];
            Imm_Gen_o[11] = instr_i[7];
            for (i = 12; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
        7'b1101111: begin // JAL
            Imm_Gen_o[0] = 1'b0;
            Imm_Gen_o[4:1] = instr_i[24:21];
            Imm_Gen_o[10:5] = instr_i[30:25];
            Imm_Gen_o[11] = instr_i[20];
            Imm_Gen_o[19:12] = instr_i[19:12];
            for (i = 20; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
        7'b1100111: begin // JALR
            Imm_Gen_o[10:0] = instr_i[30:20];
            for (i = 11; i <= 31; i = i + 1) Imm_Gen_o[i] =
instr_i[31];
        end
    endcase
```

```
end

endmodule
```

Based on the given information, I've filled the immediate values based on the table below.



ALU Control

```
`timescale 1ns/1ps
/*instr[30,14:12]*/
module ALU_Ctrl(
    input       [4-1:0] instr,
    input       [2-1:0] ALUOp,
    output reg     [4-1:0] ALU_Ctrl_o
);
wire [2:0] func3;
assign func3 = instr[2:0];

    wire [6-1:0] allInputs;
  assign allInputs = {ALUOp, instr};

  always @(*) begin
    casez (allInputs)
      6'b100000: ALU_Ctrl_o = 4'b0010; //ADD
      6'b100010: ALU_Ctrl_o = 4'b0111; //SLT
      6'b10?000: ALU_Ctrl_o = 4'b0010; //ADDI
      6'b00?010: ALU_Ctrl_o = 4'b0010; //LW, SW
      6'b01?000: ALU_Ctrl_o = 4'b0111; //BEQ
      default: ALU_Ctrl_o = 4'b0000; //NOP
    endcase
  end
```

```
endmodule
```

Determine the output of each operation by switch case.

Operation	ALU_Ctrl
add	0010
slt	0111
addi	0010
lw/sw	0010
nop	0000

ALU

```
`timescale 1ns/1ps

module alu(
    input                   rst_n,         // negative reset
   (input)
    input signed       [32-1:0]   src1,          // 32 bits source 1
         (input)
    input signed       [32-1:0]   src2,          // 32 bits source 2
         (input)
    input        [ 4-1:0]   ALU_control,   // 4 bits ALU control
input  (input)
    output reg   [32-1:0]   result,        // 32 bits result
   (output)
    output reg              Zero          // 1 bit when the output is
0, zero must be set (output)
);

  always @(*) begin
    Zero = src1 == src2;

    case (ALU_control)
      4'b0010: result = src1 + src2;
      4'b0111: result = src1 < src2;
      default: result = 32'b0;
```

```
    endcase
  end

endmodule
```

Since we are allowed to use operators directly, we rewrote the entire ALU part and the code was loads shorter than the original one.

Single Cycle CPU

```
`timescale 1ns/1ps
module Simple_Single_CPU(
    input clk_i,
    input rst_i
);

//Internal Signales
//Control Signales
wire RegWrite;
wire Branch;
wire Jump;
wire WriteBack1;
wire WriteBack0;
wire MemRead;
wire MemWrite;
wire ALUSrcA;
wire ALUSrcB;
wire [2-1:0] ALUOp;
wire PCSrc;
//ALU Flag
wire Zero;

//Datapath
wire [32-1:0] pc_i;
wire [32-1:0] pc_o;
wire [32-1:0] instr;
wire [32-1:0] RegWriteData;
wire [32-1:0] Imm_Gen_o;
wire [32-1:0] Imm_4 = 4;
wire [4-1:0] ALUControlOut;
wire [4-1:0] ALUControlIn;
```

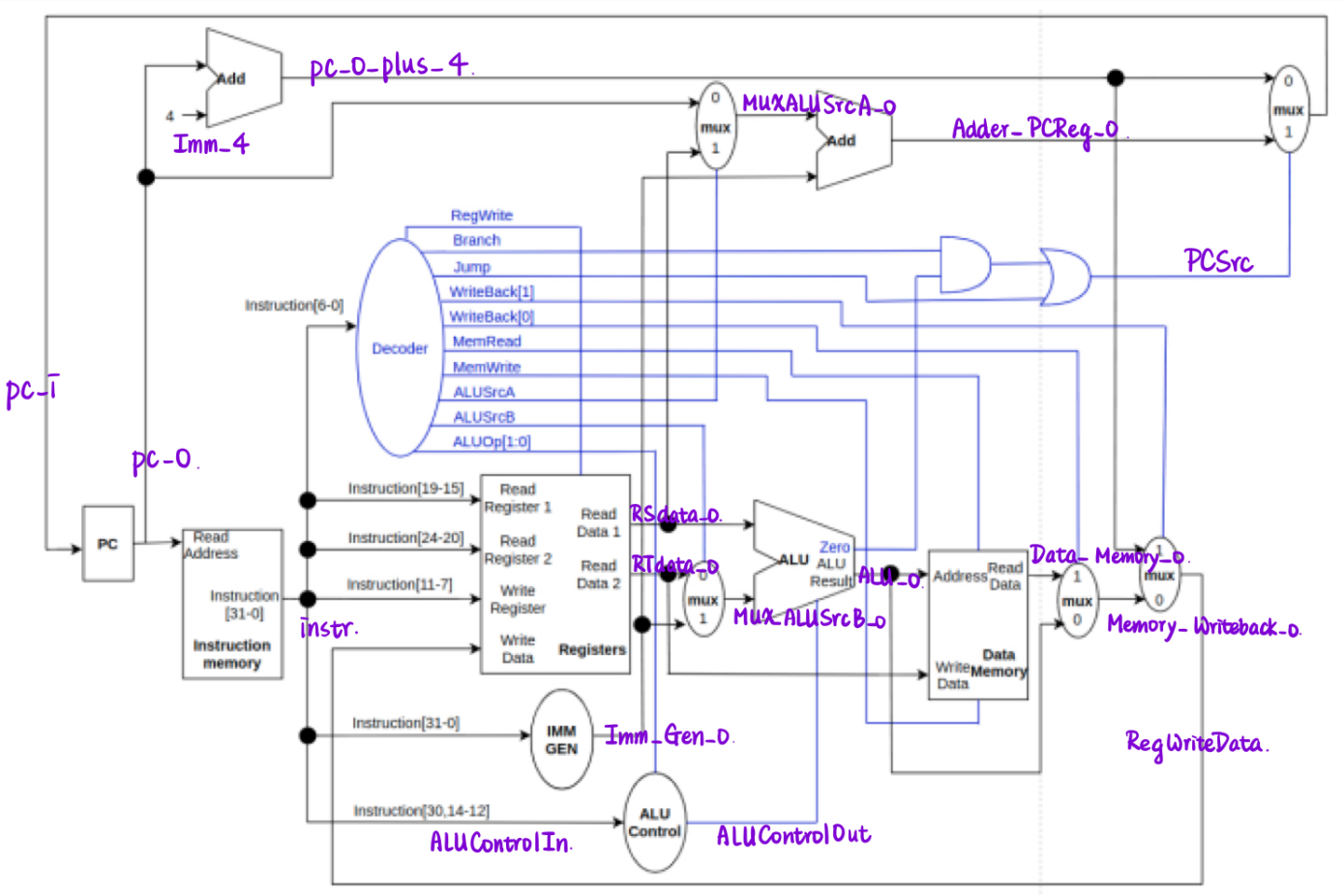
```
assign ALUControlIn[3] = instr[30];
assign ALUControlIn[2:0] = instr[14:12];
assign PCSrc = (Branch & Zero) | Jump;

ProgramCounter PC(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .pc_i(pc_i),
    .pc_o(pc_o)
);

wire [32-1:0] pc_o_plus_4;

Adder Adder_PCPlus4(
    .src1_i(pc_o),
    .src2_i(Imm_4),
    .sum_o(pc_o_plus_4)
);

Instr_Memory IM(
    .addr_i(pc_o),
    .instr_o(instr)
);

wire [32-1:0] RSdata_o;
wire [32-1:0] RTdata_o;

Reg_File RF(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .RSaddr_i(instr[19:15]),
    .RTaddr_i(instr[24:20]),
    .RDaddr_i(instr[11:7]),
    .RDdata_i(RegWriteData),
    .RegWrite_i(RegWrite),
    .RSdata_o(RSdata_o),
    .RTdata_o(RTdata_o)
);

Decoder Decoder(
    .instr_i(instr[6:0]),
    .RegWrite(RegWrite),
    .Branch(Branch),
```

```
    .Jump(Jump),
    .WriteBack1(WriteBack1),
    .WriteBack0(WriteBack0),
    .MemRead(MemRead),
    .MemWrite(MemWrite),
    .ALUSrcA(ALUSrcA),
    .ALUSrcB(ALUSrcB),
    .ALUOp(ALUOp)
);

Imm_Gen ImmGen(
    .instr_i(instr),
    .Imm_Gen_o(Imm_Gen_o)
);

ALU_Ctrl ALU_Ctrl(
    .instr(ALUControlIn),
    .ALUOp(ALUOp),
    .ALU_Ctrl_o(ALUControlOut)
);

wire [32-1:0] MUX_ALUSrcA_o;

MUX_2to1 MUX_ALUSrcA(
    .data0_i(pc_o),
    .data1_i(RSdata_o),
    .select_i(ALUSrcA),
    .data_o(MUX_ALUSrcA_o)
);

wire [32-1:0] Adder_PCReg_o;

Adder Adder_PCReg(
    .src1_i(MUX_ALUSrcA_o),
    .src2_i(Imm_Gen_o),
    .sum_o(Adder_PCReg_o)
);

MUX_2to1 MUX_PCSrc(
    .data0_i(pc_o_plus_4),
    .data1_i(Adder_PCReg_o),
    .select_i(PCSrc),
    .data_o(pc_i)
```

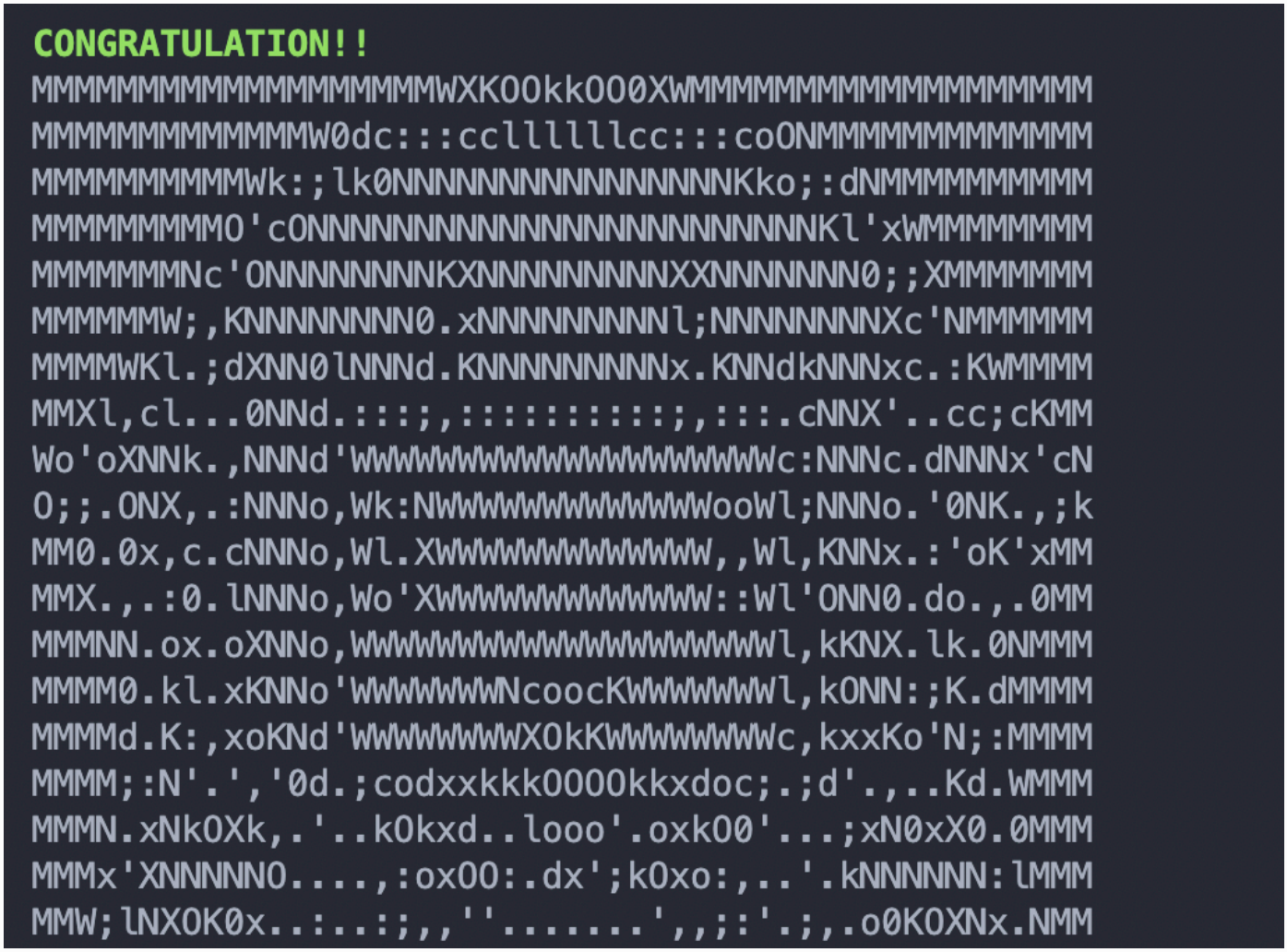
```
);

wire [32-1:0] MUX_ALUSrcB_o;

MUX_2to1 MUX_ALUSrcB(
    .data0_i(RTdata_o),
    .data1_i(Imm_Gen_o),
    .select_i(ALUSrcB),
    .data_o(MUX_ALUSrcB_o)
);

wire [32-1:0] alu_o;

alu alu(
    .rst_n(rst_i),
    .src1(RSdata_o),
    .src2(MUX_ALUSrcB_o),
    .ALU_control(ALUControlOut),
    .Zero(Zero),
    .result(alu_o)
);

wire [32-1:0] Data_Memory_o;

Data_Memory Data_Memory(
    .clk_i(clk_i),
    .addr_i(alu_o),
    .data_i(RTdata_o),
    .MemRead_i(MemRead),
    .MemWrite_i(MemWrite),
    .data_o(Data_Memory_o)
);

wire [32-1:0] MUX_WriteBack0_o;

MUX_2to1 MUX_WriteBack0(
    .data0_i(alu_o),
    .data1_i(Data_Memory_o),
    .select_i(WriteBack0),
    .data_o(MUX_WriteBack0_o)
);

MUX_2to1 MUX_WriteBack1(
```

```
    .data0_i(MUX_WriteBack0_o),
    .data1_i(pc_o_plus_4),
    .select_i(WriteBack1),
    .data_o(RegWriteData)
);

endmodule
```

Since some wires are not defined in the given input/ output, we added some wires to make sure it worked. The added wires look something like this.



Results



Encountered Problems

Since in the previous lab, we were given `RSdata[31:0]` and `RTdata[31:0]` when filling in the blanks of the single cycle CPU, but in this lab the two sources disappeared, thus at first we couldn't figure out what to put into the space.

Then we realized we are free to add wires, thus we defined a some wires that can help us connect each output and it worked.