Lab4 report

Detailed description of the implementation

Decoder.v

- list of control signals should be of each instruction
  - R-type
    - RegWrite = 1
    - ALUOp = 10
  - addi
    - RegWrite = 1
    - WriteBack1 = 1
    - ALUOp = 00
  - Load
    - RegWrite = 1
    - WriteBack0 = 1
    - MemRead = 1
    - ALUSrcB = 1
    - ALUOp = 00
  - Store
    - WriteBack0 = 1
    - MemWrite = 1
    - ALUSrcB = 1
    - ALUOp = 00
  - Branch

        - Branch = 1
        - ALUOp = 01
    - JAL
        - RegWrite = 1
        - Jump = 1
        - WriteBack1 = 1
        - ALUOp = 11
    - JALR
        - RegWrite = 1
        - Jump = 1
        - WriteBack1 = 1
        - ALUSrcA = 1
        - ALUOp = 11
- list of instructions of each control signal
    - RegWrite (set to 1)
        - R-type, addi, Load, JAL, JALR
    - Branch (set to 1)
        - Branch
    - Jump (set to 1)
        - JAL, JALR
    - WriteBack1 (set to 1)
        - JAL, JALR
    - WriteBack0 (set to 1)
        - Load, Store
    - MemRead (set to 1)
        - Load

- MemWrite (set to 1)
  - Store
- ALUSrcA (set to 1)
  - JALR
- ALUSrcB (set to 1)
  - addi, Load, Store
- ALUOp (set to 00, 01, 10, 11)
  - 00
    - addi, Load, Store
  - 01
    - Branch
  - 10
    - R-type
  - 11
    - JAL, JALR

```
assign RegWrite = (instr_i == 7'b0110011 || instr_i == 7'b0010011 ||
                   instr_i == 7'b0000011 || instr_i == 7'b1101111 ||
                   instr_i == 7'b1100111)? 1'b1: 1'b0;
assign Branch = (instr_i == 7'b1100011)? 1'b1:1'b0;
assign Jump = (instr_i == 7'b1101111 || instr_i == 7'b1100111)? 1'b1:1'b0;
assign WriteBack1 = (instr_i == 7'b1101111 || instr_i == 7'b1100111)? 1'b1:1'b0;
assign WriteBack0 = (instr_i == 7'b0000011 || instr_i == 7'b0100011)? 1'b1:1'b0;
assign MemRead = (instr_i == 7'b0000011 )?1'b1:1'b0;
assign MemWrite = (instr_i == 7'b0100011)?1'b1:1'b0;
assign ALUSrcA = (instr_i == 7'b1100111)?1'b1:1'b0; // jalr
assign ALUSrcB = (instr_i == 7'b0010011 || instr_i == 7'b0000011 ||
                  instr_i == 7'b0100011)?1'b1:1'b0;
assign ALUOp = (instr_i == 7'b0010011 || instr_i == 7'b0000011 ||
                instr_i == 7'b0100011)? 2'b00:
               (instr_i == 7'b1100011)?2'b01:
               (instr_i == 7'b0110011)?2'b10:2'b11;
```

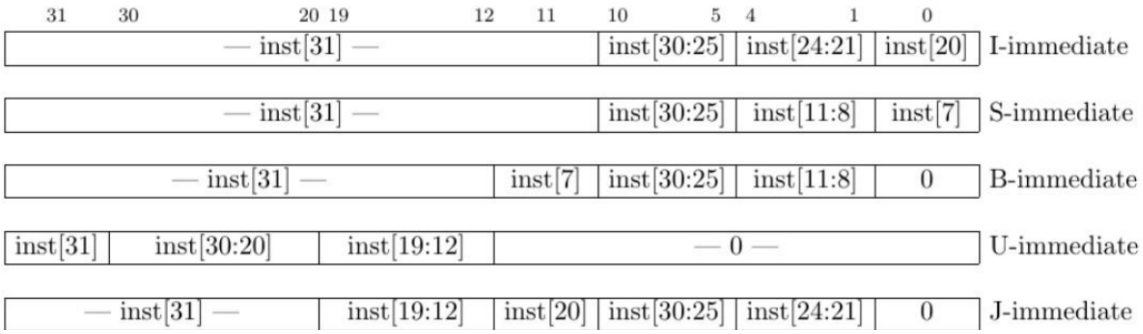
Imm_Gen.v

- The uppercase variable `I, S, B, J` are used to determine which type of immediate we should use for the instruction.

> `I: addi, load, jalr`
>
> `S: store`
>
> `B: branch`
> `J: jal`

```
wire I, S, B, J;
assign I = (opcode == 7'b0010011 || opcode == 7'b0000011 ||
            opcode == 7'b1100111)?1'b1:1'b0;//addi//load//jalr
assign S = (opcode == 7'b0100011)?1'b1:1'b0;//store
assign B = (opcode == 7'b1100011)?1'b1:1'b0;//branch
assign J = (opcode == 7'b1101111)?1'b1:1'b0;//jal
```

- Generate the immediate value depends on opcode for each instruction base on the table:




```
wire [31:0]i,s,b,j;
assign i[11:0] = instr_i[31:20];
assign i[31:12] = {20{instr_i[31]}};

assign s[4:0] = instr_i[11:7];
assign s[11:5] = instr_i[31:25];
assign s[31:12] = {20{instr_i[31]}};

assign b[0] = 1'b0;
assign b[11:1] = {instr_i[7], instr_i[30:25], instr_i[11:8]};
assign b[31:12] = {20{instr_i[31]}};

assign j[19:0] = {instr_i[19:12], instr_i[20], instr_i[30:25], instr_i[24:21], 1'b0};
assign j[31:20] = {12{instr_i[31]}};
```

- Output immediate value by `I, S, B, J`.

```
always@(*)begin
  if(I==1'b1)begin
    Imm_Gen_o = i;
  end
  else if(S==1'b1)begin
    Imm_Gen_o = s;
  end
  else if(B==1'b1)begin
    Imm_Gen_o = b;
  end
  else if(J==1'b1)begin
    Imm_Gen_o = j;
  end
end
```

ALU_Ctrl.v

- instr = the 30th and 14th-12th bits(func3) in instruction
- (ALUOp == 10): R-type
  - (instr == 0000): add
  - (instr == 1000): sub
  - (instr == 0111): and
  - (instr == 0110): or
  - (instr == 0100): exclusive or
  - (instr == 0010): slt
  - (instr == 0001): shift left
  - (instr == 1101): shift right
- (ALUOp == 00): add
  - addi, Load, Store
- (ALUOp == 01): sub
  - Branch

- (ALUOp == 11): do not care
    - JAL, JALR

```
assign ALU_Ctrl_o = (ALUOp == 2'b10 & instr == 4'b0000)? 4'b0010: //add
                    (ALUOp == 2'b10 & instr == 4'b1000)? 4'b0110: //sub
                    (ALUOp == 2'b10 & instr == 4'b0111)? 4'b0000: //and
                    (ALUOp == 2'b10 & instr == 4'b0110)? 4'b0001: //or
                    (ALUOp == 2'b10 & instr == 4'b0100)? 4'b0100: //exclusive or
                    (ALUOp == 2'b10 & instr == 4'b0010)? 4'b0111: //slt
                    (ALUOp == 2'b10 & instr == 4'b0001)? 4'b0101: //shift left
                    (ALUOp == 2'b10 & instr == 4'b1101)? 4'b0011: //shift right
                    (ALUOp == 2'b00)? 4'b0010: //add
                     4'b0110; //sub
```

alu.v

- assign `zero` bit

```
assign Zero = ((ALU_control != 4'b0100)&& (ALU_control != 4'b0101) &&
               (ALU_control != 4'b0011) && (rst_n==1'b1))? ~(|result): 1'b0;
```

- other code is the same as lab3

Simple_Single_CPU.v

- we declare some wire as the following usage

> `RSdata_o`; rs1
> `RTdata_o`; rs2
> `PCReg_i`; store `pc` or rs1(for `jalr`), select by `MUX_ALUSrcA`
> `PCSrc_i`; `pc` source when branch == 1
> `ADD4_o`; store `pc` + 4
> `alu_i2`; second alu input, select by `MUX_ALUSrcB`
> `ALUresult`; alu result, as the input of `Data_Memory` and `MUX_WriteBack0`
> `Mem_o`; store the output of `Data_Memory`

> `WB0`; store the output of `MUX_WriteBack0` , and be used to `MUX_WriteBack1`

- PC
  - input
    - `clk_i`
    - `rst_i`
    - `pc_i`: the result from `MUX_PCSrc`, as the previous `PC` value
  - output
    - `pc_o`: be the input of `IM` and `Adder_PCPlus4`

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

- Adder_PCPlus4:
  - input:
    - `pc_o`: the current PC value
    - `imm_4`: an integer 4
  - output
    - `ADD4_o`: the result of `pc + 4`, be the input of `MUX_PCSrc` and `MUX_WriteBack1`

```
Adder Adder_PCPlus4(
    .src1_i(pc_o),
    .src2_i(Imm_4),
    .sum_o(ADD4_o)
);
```

- IM
  - input

        - `pc_o`: the current PC value
    - output
        - `instr`: instruction

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

- RF
    - input
        - `clk_i`
        - `rst_i`
        - `instr[19:15]`: rs1
        - `instr[24:20]`: rs2
        - `instr[11:7]`: rd
        - `RegWriteData`: the selected result from `MUX_WriteBack1`
        - `RegWrite`: control signal from `Decoder`
    - output
        - `RSdata_o`: rs1 output
        - `RTdata_o`: rs2 output

```
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
  - input
    - `instr[6:0]`: opcode of the instruction
  - output
    - `RegWrite`: control signal generated by `Decoder`
    - `Branch`: control signal generated by `Decoder`
    - `Jump`: control signal generated by `Decoder`
    - `WriteBack1`: control signal generated by `Decoder`
    - `WriteBack0`: control signal generated by `Decoder`
    - `MemRead`: control signal generated by `Decoder`
    - `MemWrite`: control signal generated by `Decoder`
    - `ALUSrcA`: control signal generated by `Decoder`
    - `ALUSrcB`: control signal generated by `Decoder`
    - `ALUOp`: control signal generated by `Decoder`

```
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)
);
```

- ImmGen
  - input
    - `instr`: instruction
  - output

    - `Imm_Gen_o`: immediate value generated by `Imm_Gen`.

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

- ALU_Ctrl
    - input
        - `ALUControlIn`: the 30th and 14th-12th bits in instruction
        - `ALUOp`: control signal from `Decoder`
    - output
        - `ALUControlOut`: output of alu control, used in `alu`.

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

- MUX_ALUSrcA
    - input
        - `pc_o`: the current PC value
        - `RSdata_o`: rs1
        - `ALUSrcA`: control signal from `Decoder`
    - output
        - `PCReg_i`: selected result, used in `Adder_PCReg`

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

- Adder_PCReg
  - input
    - `PCReg_i`: result from `MUX_ALUSrcA`, determine which one should be add
    - `Imm_Gen_o`: immediate value generated by `Imm_Gen`
  - output
    - `PCSrc_i`: PC value generated by adder, used in the input of `MUX_PCSrc`

```
Adder Adder_PCReg(
    .src1_i(PCReg_i),
    .src2_i(Imm_Gen_o),
    .sum_o(PCSrc_i)
);
```

- MUX_PCSrc
  - input
    - `ADD4_o`: the result of `pc + 4`, generated by `Adder_PCPlus4`
    - `PCSrc_i`: : PC value generated by `Adder_PCReg`
    - `PCSrc`: control signal from `Decoder`
  - output
    - `pc_i`: the the previous `PC` value in next instruction

```
MUX_2to1 MUX_PCSrc(
    .data0_i(ADD4_o),
    .data1_i(PCSrc_i),
    .select_i(PCSrc),
    .data_o(pc_i)
);
```

- MUX_ALUSrcB
  - input
    - `RTdata_o`: rs2

        - `Imm_Gen_o`: immediate value generated by `Imm_Gen`
        - `ALUSrcB`: control signal from `Decoder`
    - output
        - `alu_i2`: the selected result, used as one of the `alu` input

```
MUX_2to1 MUX_ALUSrcB(
    .data0_i(RTdata_o),
    .data1_i(Imm_Gen_o),
    .select_i(ALUSrcB),
    .data_o(alu_i2)
);
```

- alu
    - input
        - `rst_i`
        - `RSdata_o`: first input of `alu`, rs1
        - `alu_i2`: second input of `alu`, generated from `MUX_ALUSrcB`
        - `ALUControlOut`: control signal from `ALU_Ctrl`
    - output
        - `Zero`: zero bit
        - `ALUresult`: result of `alu`

```
alu alu(
    .rst_n(rst_i),
    .src1(RSdata_o),
    .src2(alu_i2),
    .ALU_control(ALUControlOut),
    .Zero(Zero),
    .result(ALUresult)
);
```

- Data_Memory
    - input

        - `clk_i`
        - `ALUresult`: the result of alu from `alu`
        - `RTdata_o`: rs2
        - `MemRead`: control signal from `Decoder`
        - `MemWrite`: control signal from `Decoder`
    - output
        - `Mem_o`: data read

```
Data_Memory Data_Memory(
    .clk_i(clk_i),
    .addr_i(ALUresult),
    .data_i(RTdata_o),
    .MemRead_i(MemRead),
    .MemWrite_i(MemWrite),
    .data_o(Mem_o)
);
```

- MUX_WriteBack0
    - input
        - `ALUresult`: the result of alu from `alu`
        - `Mem_o`: data read from `Data_Memory`
        - `WriteBack0`: control signal from `Decoder`
    - output
        - `WB0`: selected by mux, used in `MUX_WriteBack1`

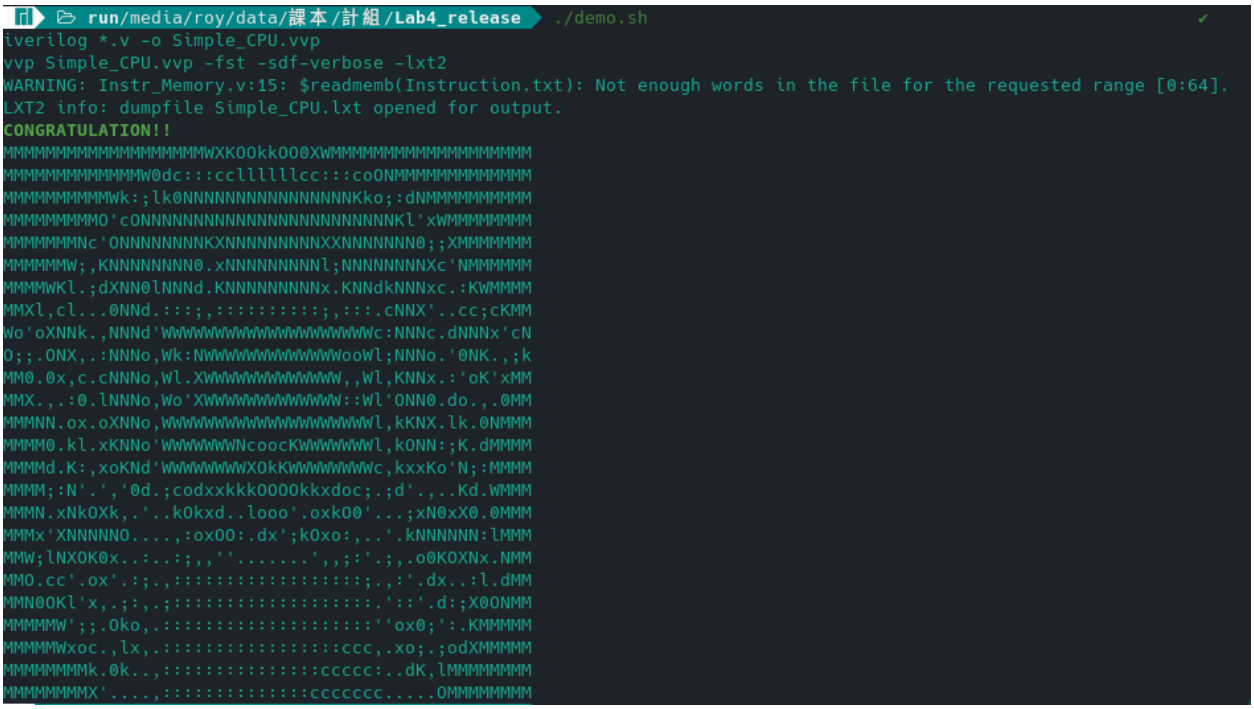
```
MUX_2to1 MUX_WriteBack0(
    .data0_i(ALUresult),
    .data1_i(Mem_o),
    .select_i(WriteBack0),
    .data_o(WB0)
);
```

- MUX_WriteBack1

- input
  - `WB0`: selected result by `MUX_WriteBack0`
  - `ADD4_o`: the result of `pc + 4`, generated by `Adder_PCPlus4`
  - `WriteBack1`: control signal from `Decoder`
- output
  - `RegWriteData`: the data selected by mux, used in `RF`

```
MUX_2to1 MUX_WriteBack1(
    .data0_i(WB0),
    .data1_i(ADD4_o),
    .select_i(WriteBack1),
    .data_o(RegWriteData)
);
```

Implementation results



Problems encountered and solutions

1. 在實做 JAL 的部份時，不小心跟 I-type 的 JALR 搞混，debug 很久後發現是 Imm_Gen 的部份寫爛了。
2. 一開始 alu_ctrl 忘了考慮 slt，於是就爛掉了。找後問題且修正後就成功通過 testbench 了。
3. 寫 Simple_Single_CPU 時，接線接到眼睛快脫窗ㄌqq，好在仔細檢查後，最後有成功動起來。
4. 一個一個看 decoder 要接去哪個 MUX，在哪個指令要設哪些 control signal，有點累。

Group 33
組員：109550168 林慧旻，109550042 林律穎