

計算機組織 Lab3 Report

109550025謝翔丞

Detailed description of the implementation

1. Simple_Single_CPU

- 這個part其實就是根據每個module的input/output給定該改的東西，而這部分也尤為重要，因為當後面遇到bug時除了本身扣寫錯之外，也有可能是這裡給錯東西。
- 不確定該如何詳細的描述這部份所以直接放上截圖

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

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

Reg_File RF(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .RSaddr_i(instr[20-1:16-1]),
    .RTaddr_i(instr[25-1:21-1]),
    .RDaddr_i(instr[12-1:8-1]),
    .RDdata_i(ALUresult[32-1:0]),
    .RegWrite_i(RegWrite),
    .RSdata_o(RSdata_o),
    .RTdata_o(RTdata_o)
);

Decoder Decoder(
    .instr_i(instr),
    .ALUSrc(ALUSrc),
    .RegWrite(RegWrite),
    .Branch(Branch),
    .ALUOp(ALUOp)
);
```

```
Adder PC_plus_4_Adder(
    .src1_i(pc_o[32-1:0]),
    .src2_i(imm_4[32-1:0]),
    .sum_o(pc_i[32-1:0])
);

ALU_Ctrl ALU_Ctrl(
    .instr({instr[30],instr[14:12]}),
    .ALUOp(ALUOp[2-1:0]),
    .ALU_Ctrl_o(ALU_control[4-1:0])
);

alu alu(
    .rst_n(rst_n),
    .src1(RSdata_o[32-1:0]),
    .src2(RTdata_o[32-1:0]),
    .ALU_control(ALU_control[4-1:0]),
    .result(ALUresult[32-1:0]),
    .zero(zero),
    .cout(cout),
    .overflow(overflow)
);
```

c.

2. alu.v

- 這部分我延用先前的32bit ALU並且加以多做修改，我增加了一個判斷 `ALU_control[1:0] == 2'b11` (也就是 `opcode == 2'b11`) 的情況下判斷2、3 bit的值，其實就是根據整個ALU_control來判斷我這次新加的三種操作，然後把運算答案給result; 而除這三種以外的運算都能用之前的ALU code 解決，所以不用更動。

```
always @(*) begin
    if(~rst_n)begin
        result <= 0;
        zero <= 0;
        cout <= 0;
    end

    else if (ALU_control[2-1:1] == 2'b11 && ALU_control[4-1:3-1] != 2'b01) begin
        case(ALU_control[4-1:3-1])
            2'b11://sra
                result <= src1 >>> src2;
            2'b10://sll
                result <= src1 << src2;
            2'b00://xor
                result <= src1 ^ src2;
        endcase

        zero <= 0;
        cout <= 0;
        overflow <= 0;
    end
end
```

```
else begin
    result <= mux_end;
    zero <= ~(|result);
    cout <= tmp_cout_reg[32-1];
    overflow <= ^tmp_cout_reg[32-1:32-2];
end
```

b.

3. ALU_Ctrl.v

a. 這裡因為我在decode的時候有將指令利用Opcode做分類, 所以用if切割成 4 個區塊:

i. ALUOp == 2'b00

1. 這部份的指令都是s或是l type,
所以ALU_Ctrl_o直接給 4b0010。

ii. ALUOp == 2'b01

1. 這部份的指令是branch的,
所以ALU_Ctrl_o直接給 4b0110。

iii. ALUOp == 2'b10

1. 這部份的指令都是R type,
所以ALU_Ctrl_o就對照slide的圖表給定
add : 4'b0010
sub : 4'b0110
and : 4'b0000
or : 4'b0001
slt : 4'b0111

iv. ALUOp == 2'b11

1. 這部份的指令是這次額外增加的,
所以ALU_Ctrl_o 我給定
sra : 4'b1111
sll : 4'b0111
xor : 4'b0011

```
always @(*) begin

    if(ALUOp == 2'b00)begin//ld sd
        ALU_Ctrl_o <= 4'b0010;
    end

    else if(ALUOp == 2'b01)begin// beq
        ALU_Ctrl_o <= 4'b0110;
    end

    else if(ALUOp == 2'b10)begin//R-type
        case(instr)
            4'b0000://add
                ALU_Ctrl_o <= 4'b0010;
            4'b1000://sub
                ALU_Ctrl_o <= 4'b0110;
            4'b0111://and
                ALU_Ctrl_o <= 4'b0000;
            4'b0110://or
                ALU_Ctrl_o <= 4'b0001;
            4'b0010://slt
                ALU_Ctrl_o <= 4'b0111;
        endcase
    end

    else if(ALUOp == 2'b11)begin//self-define
        case(instr)
            4'b1101://sra
                ALU_Ctrl_o <= 4'b1111;
            4'b0001://sll
                ALU_Ctrl_o <= 4'b0111;
            4'b0100://xor
                ALU_Ctrl_o <= 4'b0011;
        endcase
    end

end
```

4. Decoder.v

- a. 這裡的輸出有4個部份, 我先將instr_i的0到6bit assign給opcode, 12到14bit給func3, 接下來有幾個細節:
 - i. Regwrite 我是判斷opcode的4-6bit是否為010, 如果是的話Regwrite就給0, 反之為1。這是我觀察到的結果, 因為只有s-type的指令才需要寫入register(雖然這次都是R-type, 所以或許也可以直接給1?)但我仍然將他一次寫到位喇。
 - ii. ALUOp的部份較複雜, 我先將不是r-type的指令的Opcode都給2'b00, 接著判斷除了add,sub,and,or 和 slt 以外的指令, 也就是這次的 sra,xor, sll 都將他們的Opcode給 2'b11, 上述5個則是2'b10
 - iii. Branch 這次沒有使用到所以直接給0。

```
assign opcode = instr_i[6:0];
assign funct3 = instr_i[14:12];
assign RegWrite = (opcode[6:4] == 3'b010) ? 0 : 1;

assign ALUOp = (opcode[6:4] != 3'b011) ? 2'b00 : (funct3 != 3'b000 && funct3 != 3'b010 && funct3 != 3'b111 & funct3 != 3'b110) ? 2'b11 : 2'b10;

assign Branch = (opcode[6:4] == 3'b110) ? 1 : 0;
```

iv.

5. Adder.v

- a. 這部份很單純, 我的作法是將src1_i 跟 src2_i 加起來然後assign 給ouput的sum_o.
 - i. `assign sum_o = src1_i + src2_i;`

Implementation results

```
hsianchengfun@hsianchengfun-Swift:~/110second/Computer Architecture/LAB03$ chmod +x ./lab3TestScript.sh && ./lab3TestScript.sh
***** CASE 1 *****
Testcase 1 PASS
***** CASE 2 *****
Testcase 2 PASS
***** CASE 3 *****
Testcase 3 PASS
***** CASE 4 *****
Testcase 4 PASS
***** CASE 5 *****
Testcase 5 PASS
***** CASE 6 *****
Testcase 6 PASS
***** CASE 7 *****
Testcase 7 PASS
***** CASE 8 *****
Testcase 8 PASS
***** CASE 9 *****
Testcase 9 PASS
***** CASE 10 *****
Testcase 10 PASS
=====
Total Score:100
hsianchengfun@hsianchengfun-Swift:~/110second/Computer Architecture/LAB03$
```

1.

Problems encountered and solutions

1. 這次的single_cycle_CPU基本上我沒有遇到太多的問題, 主要是因為建立在上次完成好的ALU上, 所以輕鬆許多, 唯一遇到花了我一些時間處理的是slt的部份, 上次的lab明明沒有問題, 但是這次在有關slt的幾筆測資一開始都出現小問題, 所幸後來一步一步追蹤找到是因為這次有另外處理opcode, 但我不小心給錯值, 因此修正後馬上全過。