LAB04 Single Cycle CPU

109550164 徐聖哲 109550167 陳唯文

1. Detailed description of the implementation

Decoder.v

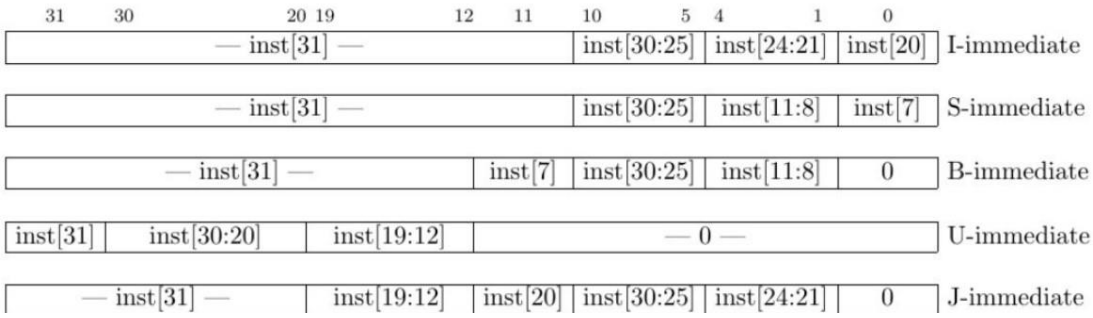
Instr	RW	B	J	WB	MR	MW	Src	Op	code
R-type	0	0	0	00	0	0	00	10	0110011
addi	1	0	0	00	0	0	01	11	0010011
Load	1	0	0	10	1	0	01	00	0000011
Store	1	0	0	00	0	1	01	00	0100011
Branch	0	1	0	00	0	0	00	10	1100011
JAL	1	0	1	01	0	0	00	10	1101111
JALR	1	0	1	01	0	0	10	10	1100111

```
assign RegWrite = (opcode[5:2] == 4'b1000) ? 1'b0 : 1'b1;
//only store and branch no need
assign Branch = (opcode == 7'b1100011) ? 1'b1 : 1'b0;
//only branch
assign Jump = (opcode[2:0] == 3'b111) ? 1'b1 : 1'b0;
//only jal jalr
assign WriteBack1 = (opcode[2:0] == 3'b111) ? 1'b1 : 1'b0;
//only jal jalr
assign WriteBack0 = (opcode[6:4] == 3'b000) ? 1'b1 : 1'b0;
//only lw
assign MemRead = (opcode == 7'b0000011) ? 1'b1 : 1'b0;
//only load need
assign MemWrite = (opcode == 7'b0100011) ? 1'b1 : 1'b0;
//only store need
assign ALUSrcA = (opcode == 7'b1100111) ? 1'b1 : 1'b0;
//only jalr
assign ALUSrcB = (opcode[6:4] == 3'b001 || opcode[6:4] == 3'b000 || opcode[6:4] == 3'b010) ? 1'b1 : 1'b0;
//choose imm
assign ALUOp = (opcode[6:4] == 3'b011) ? 2'b10 : (opcode[6:4] == 3'b001) ? 2'b11 : (opcode[6:4] == 3'b110) ? 2'b10 : 2'b00;

endmodule
```

每種 type 對應的 signal 如表格顯示。Writeback 是之前沒有學過的 signal，我們的思考方向是：在 WriteBack0 的部分，只有 lw 指令需用到 memory，所以我們把其他指令為都設成 0 (包含 don't care)，Writeback1 的部分則只有 jump instruction 需要設成 1

Imm_Gen.v




```
always @(*) begin
    case (opcode)
        7'b0110011 : Imm_Gen_o <= 0; //R-type
        7'b0010011 : Imm_Gen_o <= {{21{instr_i[31]}}, instr_i[30:20]}; //addi
        7'b0000011 : Imm_Gen_o <= {{21{instr_i[31]}}, instr_i[30:20]}; //lw
        7'b1100111 : Imm_Gen_o <= {{21{instr_i[31]}}, instr_i[30:20]}; //jalr
        7'b0100011 : Imm_Gen_o <= {{21{instr_i[31]}}, instr_i[30:25], instr_i[11:7]}; //sw
        7'b1100011 : Imm_Gen_o <= {{20{instr_i[31]}}, instr_i[7], instr_i[30:25], instr_i[11:8], 1'b0}; //beq
        7'b1101111 : Imm_Gen_o <= {{12{instr_i[31]}}, instr_i[19:12], instr_i[20] , instr_i[30:21], 1'b0}; //jal
        default: Imm_Gen_o <= 0;
    endcase
end
```

依 spec 提供的圖寫即可完成這一部份，並將前面補上的 bit 填入 imm 的最高位數，也就是 sign extension。需要注意的是在 B-type 跟 J-type 的指令內並不包含立即值最後一個的 bit，所以要補上 0。

ALU_Ctrl.v

opcode	ALUOp	Operation	I_{30}+fun3	ALU function	ALU control
ld	00	load register	XXXX	add	0010
sd	00	store register	XXXX	add	0010
beq	01	branch on equal	XXXX	subtract	0110
R-type	10	add	0000	add	0010
		subtract	1000	subtract	0110
		AND	0111	AND	0000
		OR	0110	OR	0001

```
reg [3:0] result;
assign ALU_Ctrl_o = result;

parameter aluadd = 4'b0010;
parameter alusub = 4'b0110;
parameter aluand = 4'b0000;
parameter aluor  = 4'b0001;
parameter aluslt = 4'b0111;

reg [3:0] alufunc;

always @(*) begin
        case (instr)
                4'b0000 : alufunc <= aluadd;
                4'b1000 : alufunc <= alusub;
                4'b0111 : alufunc <= aluand;
                4'b0110 : alufunc <= aluor;
                4'b0010 : alufunc <= aluslt;
                default: alufunc <= aluadd;
        endcase
end

always @(*) begin
        case (ALUOp)
                2'b00: result <= aluadd;
                2'b01: result <= alusub;
                2'b10: result <= alufunc;
                2'b11: result <= aluadd;
                default: result <= 4'b1111;
        endcase
end

endmodule
```

由於只有 R-type 指令除了 ALUOp 外，還需要 ALU control 的判斷，所以這裡拆成兩部分寫。第一個 always 的 alufunc 即為在 R-type 指令下，對於不同 ALU control 的判斷和相對應的動作，第二個 always 寫的則是 ALUOp 對應到不同 type 的指令需要輸出的動作。

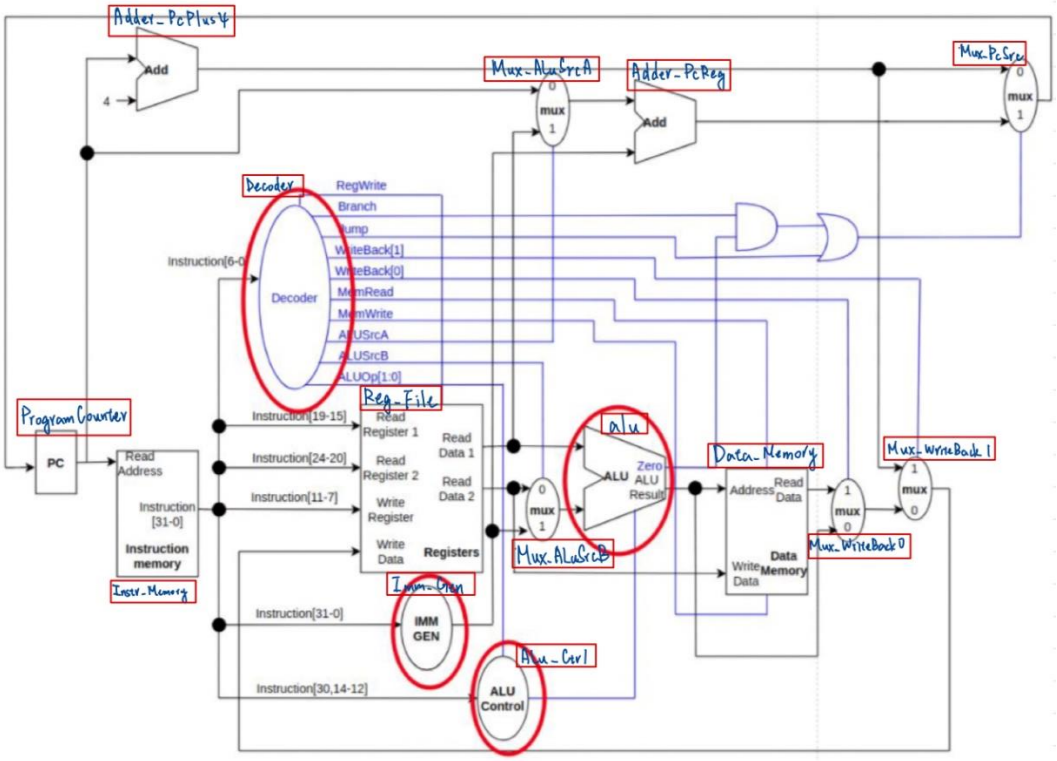
Alu.v

```
assign Zero = (result == 32'b0);

always @(*) begin
        case (ALU_control)
            4'b0010: result <= src1 + src2;
            4'b0110: result <= src1 - src2;
            4'b0000: result <= src1 & src2;
            4'b0001: result <= src1 | src2;
            4'b0111: result <= {31'b0, (src1 < src2)};
            default: result <= result;
        endcase
end
```

直接使用 operator 完成 and, or, add, sub, slt 的指令

Simple_Single_Cpu.v




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

```
Adder Adder_PCPlus4(
    .src1_i(pc_o),
    .src2_i(Imm_4),
    .sum_o(pcplus4)
);
```

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

```
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)
);
```

```
MUX_2to1 MUX_ALUSrcA(
    .data0_i(pc_o),
    .data1_i(RSdata_o),
    .select_i(ALUSrcA),
    .data_o(MUX_ALUSrcA_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(pcplus4),
    .data1_i(Adder_PCReg_o),
    .select_i(PCSrc),
    .data_o(pc_i)
);
```

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

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

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

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

```
MUX_2to1 MUX_WriteBack1(
    .data0_i(MUX_WriteBack0_o),
    .data1_i(pcplus4),
    .select_i(WriteBack1),
    .data_o(RegWriteData)
);
```

我們在圖上標好了每一個 unit 的位置，能更清楚每一個 funtion 的 input 和 output 應該要怎麼接才對。

2. Implementation results

```
iverilog *.v -o Simple_CPU.vvp
vvp Simple_CPU.vvp -fst -sdf-verbose -lxt2
WARNING: Instr_Memory.v:15: $readmemb(Instruction.txt): Not enough words in the file for the requested range [0:64].
LXT2 info: dumpfile Simple_CPU.lxt opened for output.
CONGRATULATION!!
MMMMMMMMMMMMMMMMMMMMWXKOOkkOO0XWMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMW0dc:::ccllllllcc:::coONMMMMMMMMMMMMMM
MMMMMMMMMMWk:;lk0NNNNNNNNNNNNNNNKko;:dNMMMMMMMMMM
MMMMMMMMMO'cONNNNNNNNNNNNNNNNNNNNNNKl'xWMMMMMMMM
MMMMMMMNc'ONNNNNNNKXNNNNNNNNXXNNNNNN0;;XMMMMMMM
MMMMMMW;,KNNNNNNN0.xNNNNNNNNl;NNNNNNNXc'NMMMMMM
MMMMWKl.;dXNN0lNNNd.KNNNNNNNNx.KNNdkNNNxc.:KWMMMM
MMXl,cl...0NNd.:::;,::::::::::;,:::.cNNX'..cc;cKMM
Wo'oXNNk.,NNNd'WWWWWWWWWWWWWWWWWWWWc:NNNc.dNNNx'cN
O;;.ONX,.:NNNo,Wk:NWWWWWWWWWWWWWWooWl;NNNo.'0NK.,;k
MM0.0x,c.cNNNo,Wl.XWWWWWWWWWWWWW,,Wl,KNNx.:'oK'xMM
MMX.,.:0.lNNNo,Wo'XWWWWWWWWWWWW::Wl'ONN0.do.,.0MM
MMMNN.ox.oXNNo,WWWWWWWWWWWWWWWWWWWWl,kKNX.lk.0NMMM
MMMM0.kl.xKNNo'WWWWWWWNcoocKWWWWWWWl,kONN:;K.dMMMM
MMMMd.K:,xoKNd'WWWWWWWXOkKWWWWWWWc,kxxKo'N;:MMMM
MMMM;:N'.','0d.;codxxkkkOOOOkkxdoc;.;d'.,..Kd.WMMM
MMMN.xNkOXk,.'..kOkxd..looo'.oxkO0'...;xN0xX0.0MMM
MMMx'XNNNNNO....,:oxOO:.dx';kOxo:,..'.kNNNNNN:lMMM
MMW;lNXOK0x..:..:;,,''.......',,;:'.;,.o0KOXNx.NMM
MMO.cc'.ox'.:;.,::::::::::::::::::;.,:'.dx..:l.dMM
MMN0OKl'x,.;:,.;::::::::::::::::::::.':'.d:;X0ONMM
MMMMMW';;.Oko,.:::::::::::::::::::''ox0;':.KMMMMM
MMMMMWxoc.,lx,.:::::::::::::::::ccc,.xo;.;odXMMMMM
MMMMMMMMk.0k..,:::::::::::::::ccccc:..dK,lMMMMMMMM
MMMMMMMMX'....,:::::::::::::::ccccccc.....OMMMMMMMM
```

3.Problems encountered and solutions

1.The first problem I first encountered is in Decoder.v . In the previous lecture, I didn't leran about writebackA, writebackB. So, I spent some time understand the spec, we find out that only lw will use writebackB, thus we simplify our code and functioned it well.

2.The second problem is found when I was browsing my result.txt and answer.txt. Line 1695 in result.txt is different from answer.txt, which correspond to "slt". I first check my decoder.v and imm.v. But I finally find the problem in alu.v, which is the signed problem when I use (src1 < src2). As soon as, I changed the code, I immediately receive congratulation.

4.experience

109550164 徐聖哲

這次的作業的難度跟上次差不多，只要搞懂 spec 裡面的圖就可以寫出 decoder 和 imm.v。透過這次的 code 我更了解 instruction 在 cpu 中跑的過程以及各個 mux 的作用，終於有把計算機組織知識實作的感覺。

109550167 陳唯文

在寫這次 hw 的過程中，最需要的應該就是要先讀懂整個 single cycle cpu 的 datapath，才能把 decoder.v 在不同 type 的指令下每個輸出和 single cycle cpu.v 的線接好。而這也是我們第一次的小組作業，在兩個人互相討論作業的情況下，感覺的確能完成的更有效率。