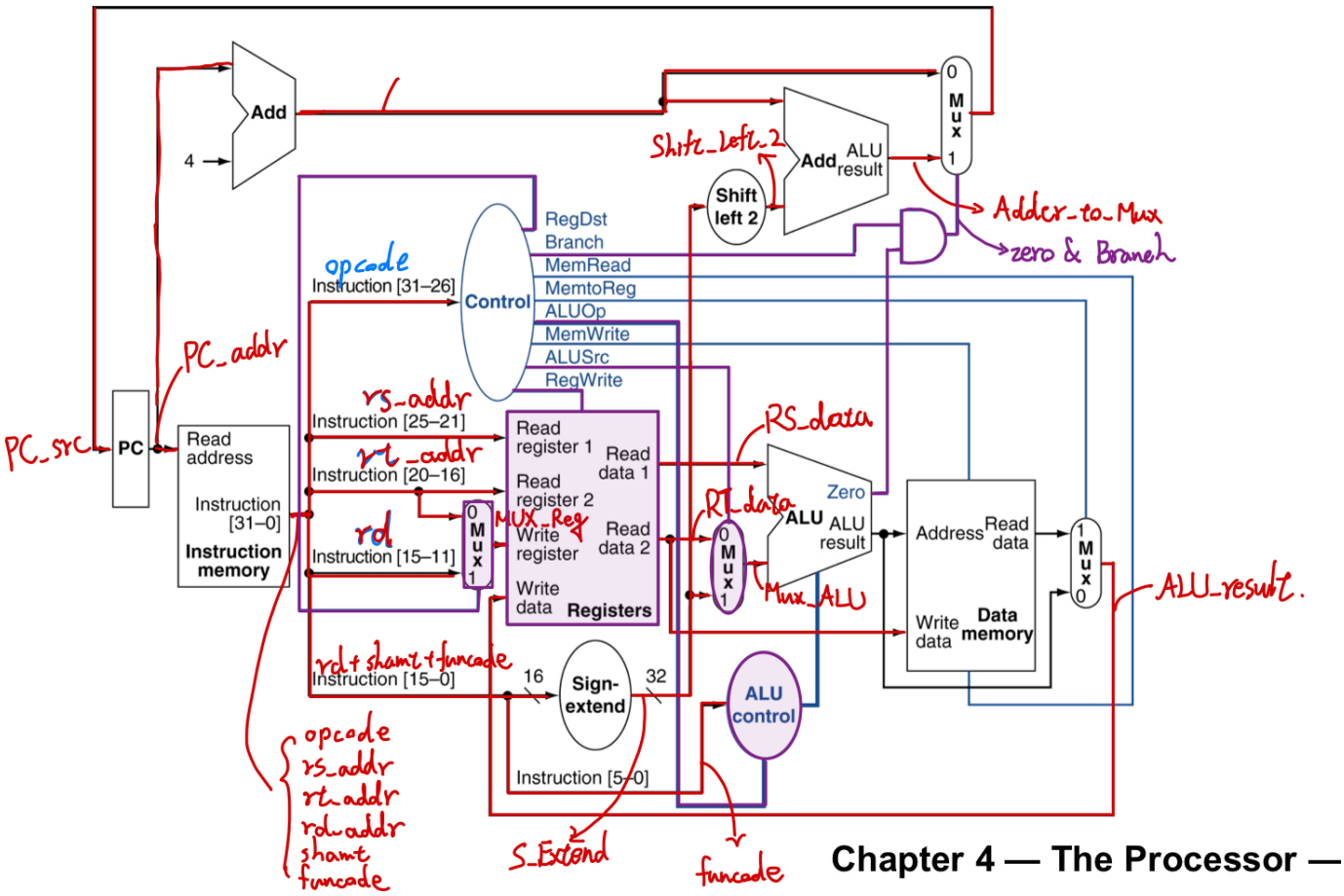
Computer Organization

Architecture diagrams:



這次的Single_Simple_CPU我是參照上圖接線，紅字紅線為wire name和線路結構，紫線則是Control的selector。

Hardware module analysis:

Adder：如左圖，就是很簡單的把input的src1和src2相加。

```
19  input  [32-1:0]  src1_i;
20  input  [32-1:0]  src2_i;
21  output [32-1:0]  sum_o;
22
23  //Internal Signals
24  wire    [32-1:0]     sum_o;
25
26  //Parameter
27  assign sum_o = src1_i + src2_i;
28  //Main function
```

ALU：參照上次Lab1的講義，將input的ctrl_i當作是ALU_operation，由上而下是add, or, add, sub, nor, slt，default則將result值直接設為0，但執行後發現有無default這行範測的執行結果不會變，最後zero的判斷則以輸出result是否為0做依據。

```
//I/O ports
input signed [32-1:0]  src1_i;
input signed [32-1:0]  src2_i;
input  [4-1:0]   ctrl_i;

output [32-1:0]  result_o;
output           zero_o;

//Internal signals
reg    [32-1:0]  result_o;
wire             zero_o;

//Parameter
assign zero_o = (result_o == 0);

always @(*) begin
    case(ctrl_i)
        4'b0000 : result_o <= src1_i & src2_i;      //and
        4'b0001 : result_o <= src1_i | src2_i;      //or
        4'b0010 : result_o <= src1_i + src2_i;      //add
        4'b0110 : result_o <= src1_i - src2_i;      //sub
        4'b1100 : result_o <= ~(src1_i | src2_i);   //nor
        4'b0111 : result_o <= (src1_i < src2_i);    //slt
        default : result_o <= 0;
    endcase
end
//Main function
```

ALU_Ctrl：參照教授給的CO Lab 2的講義，需進行的指令有除了Rformat指令之外的addi, slti, beq，一開始不太清楚ALUOp_i的3個bits要怎麼樣才可執行這額外三個指令，最後是自己設計四個值分別為R-type, addi, slti, beq做執行。

```
//I/O ports
input      [6-1:0] funct_i;
input      [3-1:0] ALUOp_i;

output     [4-1:0] ALUCtrl_o;

//Internal Signals
reg        [4-1:0] ALUCtrl_o;

//Parameter
always @(*) begin
  case(ALUOp_i)
    3'b000 : begin // R-type
              case(funct_i)
                      6'b100000:  ALUCtrl_o <= 4'b0010; // add
                      6'b100010:  ALUCtrl_o <= 4'b0110; // sub
                      6'b100100:  ALUCtrl_o <= 4'b0000; // and
                      6'b100101:  ALUCtrl_o <= 4'b0001; // or
                      6'b101010:  ALUCtrl_o <= 4'b0111; // slt
              endcase
        end
        3'b010 : ALUCtrl_o <= 4'b0010; // addi
        3'b011 : ALUCtrl_o <= 4'b0111; // slti
        3'b100 : ALUCtrl_o <= 4'b0110; // beq
  endcase
end
```

MUX_2to1：如左圖，selector為0就輸出data1，為1就輸出data2。

```
//I/O ports
input   [size-1:0] data0_i;
input   [size-1:0] data1_i;
input              select_i;
output  [size-1:0] data_o;

//Internal Signals
reg     [size-1:0] data_o;

always @(*) begin
    data_o <= (select_i == 0) ? data0_i : data1_i;
end
//Main function
```

Decoder：參照講義ch4 page.32，分成R-type, I-type, beq，再參照Architecture Diagram的圖，將各個指令的各個Control Signal賦值。R-type和I-type不同處就在寫入哪個register(rt or rd)，beq則是記得在Branch填入1。

```
37  //Parameter
38  always @(*) begin
39      case(instr_op_i)
40          6'b000000 : begin // R-type
41              RegDst_o <= 1;
42              ALUSrc_o <= 0;
43              Branch_o <= 0;
44              RegWrite_o <= 1;
45              ALU_op_o <= 3'b000;
46          end
47
48          6'b001000 : begin // addi
49              RegDst_o <= 0;
50              ALUSrc_o <= 1;
51              Branch_o <= 0;
52              RegWrite_o <= 1;
53              ALU_op_o <= 3'b010;
54          end
55
56          6'b001010 : begin // slti
57              RegDst_o <= 0;
58              ALUSrc_o <= 1;
59              Branch_o <= 0;
60              RegWrite_o <= 1;
61              ALU_op_o <= 3'b011;
62          end
63
64          6'b000100 : begin // beq
65              RegDst_o <= 1;
66              ALUSrc_o <= 0;
67              Branch_o <= 1;
68              RegWrite_o <= 0;
69              ALU_op_o <= 3'b100;
70          end
71
72      endcase
```

Shift_Left_Two_32：就是input左移2bits後輸出。

```
13  //I/O ports
14  input [32-1:0] data_i;
15  output [32-1:0] data_o;
16
17  //shift left 2
18  assign data_o = (data_i << 2);
```

Sign_Extend：如下圖，這個寫法是參照網路上的寫法，因為覺得自己本來寫得好醜又看到別人有別的寫法就改了，也確認過範測輸出都一樣，就是將第16bit延伸至32bit。

```
24  //Sign extended
25  always@(data_i)
26      data_o = { {16{data_i[15]} } , data_i};
27      // data_o = {data_i[15], data_i[15], data_i[15], data_i[15], data_i[15],
28      //           data_i[15], data_i[15], data_i[15], data_i[15], data_i[15],
29      //               data_i[15], data_i[15], data_i[15], data_i[15], data_i[15],
30      //                   data_i[15], data_i};
```

Finished part:

看到有人發問這段該寫什麼，而助教回答就寫我完成了什麼指令就好，所以我決定把top modulo丟在這。

https://github.com/Exfruit/verilog/blob/610d37f7e126a4e0b6239024116f6dc0f999727b/Lab_2/code/Simple_Single_CPU.v

總之就是照著Architecture Diagram的圖做接線。

Problems you met and solutions:

在top modulo 的線一直接錯，解決方法是把圖畫好再去洗個臉讓眼睛看清楚一點；還有就是ALU_op有3個bits一直讓我想不通，因為在講義上都只找到2bits的說明，結果原來是要自己做設計。

Summary:

這次的作業比起上次的ALU我覺得要簡單許多，最麻煩的地方應該是在ALU_op設計I-type和beq，還有最後top modulo的接線，因為接錯線還找了一段時間才知道錯在哪，而設wire時也是耗費一段時間，因為線路實在太多看了眼睛都痛了，我覺得在接線時一定要把像上面的Architecture Diagram的圖畫清楚，那條線要接哪，還有wire name不要搞錯，除了這些問題外這次作業並沒有什麼太大的問題，而在寫這次作業的過程中我也更了解Control是如何運作，以及Decoder的MUX的實作，在上課時聽不太懂的地方經過這次實作有了更深的了解。