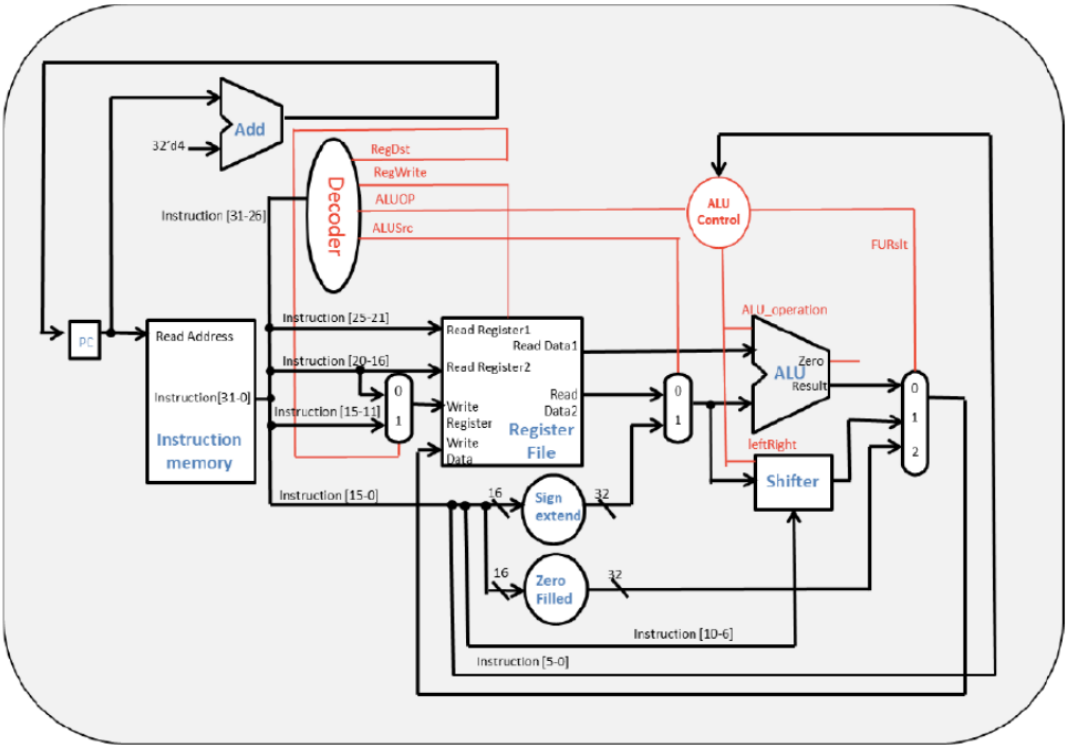
Computer Organization – HW3

109350008 張詠哲

Architecture diagrams:

接線的方式和助教 spec 中的示意圖一樣，但是因為 Zero Filled 這次沒有要實作，因此 zero filled 雖然在 simple_single_CPU 有接，但其 module 實際上沒有東西。



Hardware module analysis:

- **Adder:**
  加法器，用於處理 program counter 的加法 (+4)。
- **ALU:**
  處理 and、or、nor、slt、add、sub...的算術電路。
- **ALU_Ctrl:**
  產出 ALU 所需的 control signals，像是 ALU operation and FURslt。
- **Decoder:**
  decode instruction operation 後產出 control signal，像是 RegWrite、ALUOp、ALUSrc、RegDst。
- **Instr_Memory:**
  指令存放的記憶體。

- **Mux2to1:**
  二選一選擇器，在這次的實作中會使用在 rd、rt 的選擇以及 ALU 的第二個資料是從 rd 或是 sign extension 的選擇。
- **Mux3to1:**
  三選一選擇器，原本會使用在最後寫回 register 是要用 shifter 或是 ALU 或是 zero filled 的結果，但是因為這次沒有需要實作 zero filled，所以這次只有 ALU 或是 shifter。
- **Program_Counter:**
  控制指令執行 by clock。
- **Reg_File:**
  看要使用那些 register，且會輸出 rs 或是 rd 的資料給 ALU。
- **Shifter:**
  做資料的 shift，可以 shift left 或是 right。
- **Sign_Extend:**
  把資料的最高位元做 sign extension。
- **Simple_Single_CPU:**
  最後的 module，用於把所有小 module 串接起來

Finished part:

自己實作部分，完成了:

1. Adder

```
module Adder( src1_i, src2_i, sum_o );

//I/O ports
input    [32-1:0] src1_i;
input    [32-1:0] src2_i;
output   [32-1:0] sum_o;

//Internal Signals
wire [32-1:0] sum_o;

//Main function
/*your code here*/
assign sum_o = src1_i + src2_i;
```

2. ALU_Ctrl

```
parameter addi = 3'b011;

assign ALU_operation_o = ({ALUOp_i,funct_i} == 9'b010010010 || ALUOp_i == addi) ? 4'b0010: //add
                         ({ALUOp_i,funct_i} == 9'b010010000) ? 4'b0110: //sub
                         ({ALUOp_i,funct_i} == 9'b010010100) ? 4'b0001: //and
                         ({ALUOp_i,funct_i} == 9'b010010110) ? 4'b0000: //or
                         ({ALUOp_i,funct_i} == 9'b010010101) ? 4'b1101: //nor
                         ({ALUOp_i,funct_i} == 9'b010100000) ? 4'b0111: //slt
                         ({ALUOp_i,funct_i} == 9'b010000000) ? 4'b0001: //sll
                         ({ALUOp_i,funct_i} == 9'b010000010) ? 4'b0000: 4'b0000; //srl

assign FURslt_o = ({ALUOp_i,funct_i} == 9'b010010010 || ALUOp_i == addi) ? 2'b00: //add
                  ({ALUOp_i,funct_i} == 9'b010010000) ? 2'b00: //sub
                  ({ALUOp_i,funct_i} == 9'b010010100) ? 2'b00: //and
                  ({ALUOp_i,funct_i} == 9'b010010110) ? 2'b00: //or
                  ({ALUOp_i,funct_i} == 9'b010010101) ? 2'b00: //nor
                  ({ALUOp_i,funct_i} == 9'b010100000) ? 2'b00: //slt
                  ({ALUOp_i,funct_i} == 9'b010000000) ? 2'b01: //sll
                  ({ALUOp_i,funct_i} == 9'b010000010) ? 2'b01: 2'b00; //srl
```

3. Decoder

```
/*your code here*/
parameter R = 6'b000000;
parameter addi = 6'b001000;

assign RegWrite_o = (instr_op_i == R) ? 1'b1 :
                    (instr_op_i == addi) ? 1'b1 : 1'b0;

assign ALUOp_o = (instr_op_i == R) ? 3'b010 :
                 (instr_op_i == addi) ? 3'b011 : 3'b000;

assign ALUSrc_o = (instr_op_i == R) ? 1'b0 :
                  (instr_op_i == addi) ? 1'b1 : 1'b0;

assign RegDst_o = (instr_op_i == R) ? 1'b1 :
                  (instr_op_i == addi) ? 1'b0 : 1'b0;
```

4. Mux2to1

```
module Mux2to1( data0_i, data1_i, select_i, data_o );

parameter size = 0;

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

//Main function
/*your code here*/
assign data_o = select_i ? data1_i : data0_i;
```

5. Mux3to1

```
module Mux3to1( data0_i, data1_i, data2_i, select_i, data_o );

parameter size = 0;

//I/O ports
input  wire    [size-1:0] data0_i;
input  wire    [size-1:0] data1_i;
input  wire    [size-1:0] data2_i;
input  wire    [2-1:0] select_i;
output wire    [size-1:0] data_o;

//Main function
/*your code here*/
assign data_o = select_i[1] ? data2_i :
                select_i[0] ? data1_i : data0_i;
```

6. Sign_Extend

```
module Sign_Extend( data_i, data_o );

//I/O ports
input     [16-1:0] data_i;
output    [32-1:0] data_o;

//Internal Signals
wire [32-1:0] data_o;

//Sign extended
/*your code here*/
assign data_o[32-1:0] = {{16{data_i[16-1]}},data_i[16-1:0]};
```

7. Simple_Single_CPU

而以下是由助教提供的 module:

ALU

Instr_Memory

Program_Counter

Reg_File

Shifter

測資部分:

3 個測資都通過(data1、data2、data3)

```
# run 1000ns
==========================================
   Congratulation. You pass  TA's pattern
==========================================
$stop called at time : 90 ns : File "C:/Users/admin/OneDrive/桌面/hw3/TestBench.v" Line 263
ERROR: [Common 17-39] 'xsim' failed due to earlier errors.
```

Problems you met and solutions:

最後要串接 simple_single_CPU 的時候要綜合所有的 module 有點複雜，要慢慢對照。

最後一個是不知道為什麼下載 vivado 之後我的電腦磁碟就怪怪的，最後還壞軌，最後只好再買一個。不知道是因為 vivado 的關係還是磁碟本身就時候到了。

Summary:

這次學會如何接一個簡易型的 single cycle CPU，其中包含: Adder、ALU_Ctrl、Decoder、Mux2to1、Mux3to1、Sign_Extend、Simple_Single_CPU、ALU、Instr_Memory、Program_Counter、Reg_File、Shifter。而這也是我第一次實作那麼大型的 verilog。