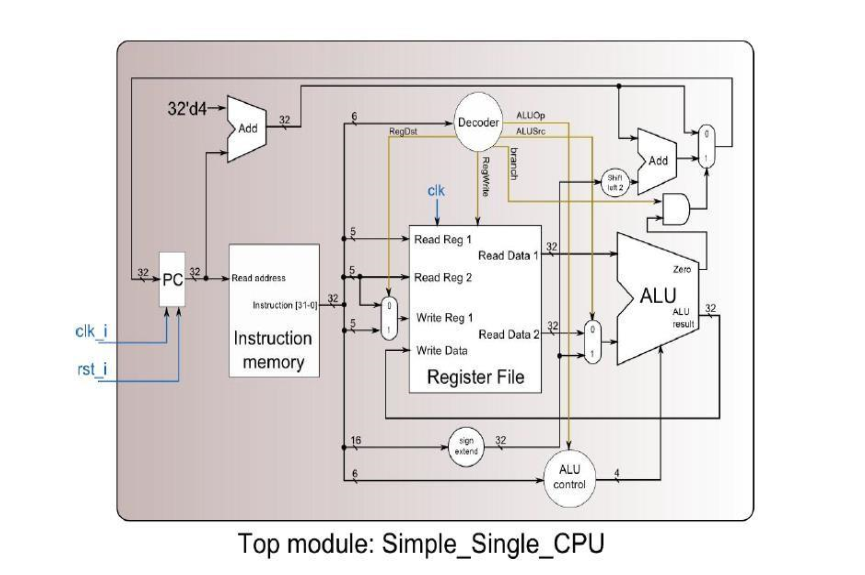
**Computer Organization**

**0516303 余采彧**

**0516208 黃郁恬**

**Architecture diagrams:**

****

**Hardware module analysis:**

* **Mux\_2to1:**

**assign data\_o= (select\_i ? data1\_i : data0\_i) ;**

**根據輸入的select\_i判斷:**

**🡪select\_i 為1，輸出就等於輸入的data1\_i**

**🡪select\_i 為0，輸出就等於輸入的data0\_i**

**(p.s. 共有三個mux，輸入和輸出的東西並不一樣)**

* **Shift\_Left\_Two\_32:**

**assign data\_o = data\_i<<2;**

**把輸入想左移兩個bit(<<)，賦值給輸出。**

* **Sign\_Extend:**

**data\_o = {{16{data\_i[15]}}, data\_i};**

**把輸入的前16個bit(0~15)複製給輸出的前16個bit，並把輸入的sign bit(也就是第15個bit: msb)複製給輸出的後16個bit。**

* **Adder:**

**assign sum\_o = src1\_i + src2\_i;**

**把輸入的兩個運算元做相加。**

* **ALU\_Ctrl:**

**case(ALUOp\_i)**

**0: ALUCtrl\_o=4'd2;**

**1: ALUCtrl\_o=4'd6;**

**2: begin**

**ALUCtrl\_o[0]= funct\_i[0] || funct\_i[3];**

**ALUCtrl\_o[1]= (!funct\_i[2]);**

**ALUCtrl\_o[2]= funct\_i[1];**

**ALUCtrl\_o[3]= 1'b0;**

**end**

**3: ALUCtrl\_o=4'd7;**

**endcase**

**根據Decoder的ALU\_op\_o:**

**ALU\_op\_o為0時:**

**ADDI，ALUCtrl\_o設為2，也就是add(0010)。**

**ALU\_op\_o為1時:**

**BEQ，ALUCtrl\_o設為6，也就是sub(0110)，因為beq就是要做相減的動作，看結果是否為0來判斷兩運算元是否相等。**

**ALU\_op\_o為2時:**

**R-format，ALUCtrl\_o進階透過function判斷(畫真值表)。**

**ALU\_op\_o為3時:**

**SLTI，ALUCtrl\_o設為7，也就是做slt(0111)。**

* **Decoder:**

**case(instr\_op\_i)**

**6'b000000: //R-format**

**begin**

**assign ALU\_op\_o = 3'b010; //2**

**assign ALUSrc\_o = 1'b0;**

**assign RegWrite\_o = 1'b1;**

**assign RegDst\_o = 1'b1;**

**assign Branch\_o = 1'b0;**

**end**

**6'b000100: //BEQ**

**begin**

**assign ALU\_op\_o = 3'b001; //1**

**assign ALUSrc\_o = 1'b0;**

**assign RegWrite\_o = 1'b0;**

**//RegDst\_o don't care**

**assign Branch\_o = 1'b1;**

**end**

**6'b001000: //ADDI**

**begin**

**assign ALU\_op\_o = 3'b000; //0**

**assign ALUSrc\_o = 1'b1;**

**assign RegWrite\_o = 1'b1;**

**assign RegDst\_o = 1'b0;**

**assign Branch\_o = 1'b0;**

**end**

**6'b001010: //SLTI**

**begin**

**assign ALU\_op\_o = 3'b011; //3**

**assign ALUSrc\_o = 1'b1;**

**assign RegWrite\_o = 1'b1;**

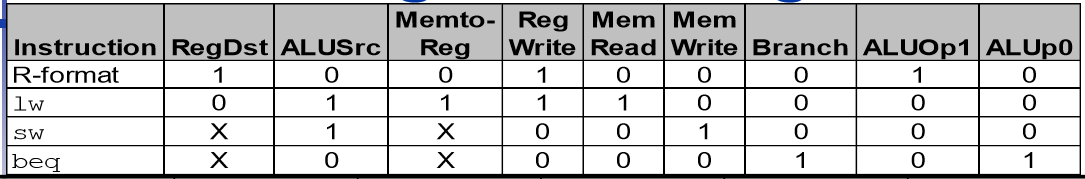
**assign RegDst\_o = 1'b0;**

**assign Branch\_o = 1'b0;**

**end**

**endcase**

**對照下表去做賦值:**

****

* **ALU:**

**同上次作業，只是參數的地方稍微調整成這次lab的格式。**

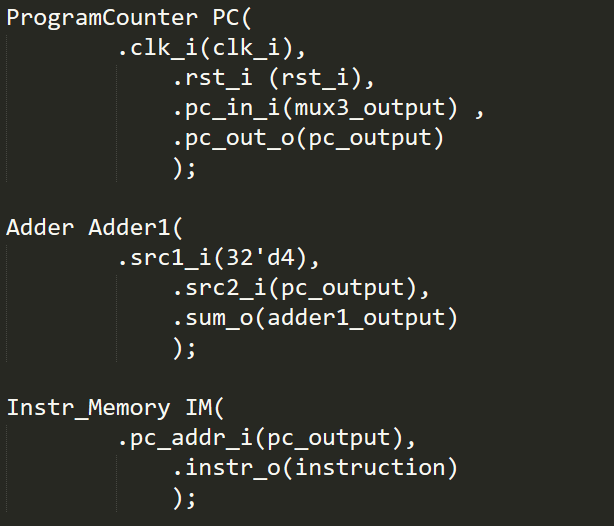
**(因此還必須引入alu\_top)**

**Finished part:**

1. **ALU(包含alu\_top)**
2. **Decoder**
3. **ALU\_Ctrl**
4. **Adder**
5. **Sign\_Extend**
6. **Shift\_Left\_Two\_32**
7. **Mux\_2to1**
8. **Simple\_Single\_CPU**

**實作方法如Hardware module analysis所敘述**

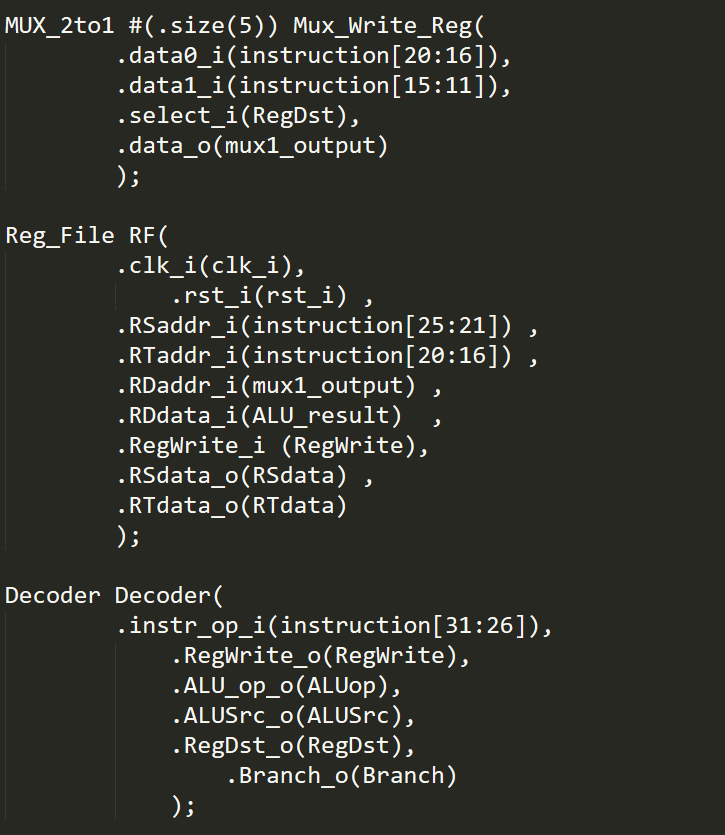
**如何接線:**

****

**PC的input接第三個mux的output，也就是Mux\_PC\_Source的output。**

**第1個adder的input分別是4以及PC的output。**

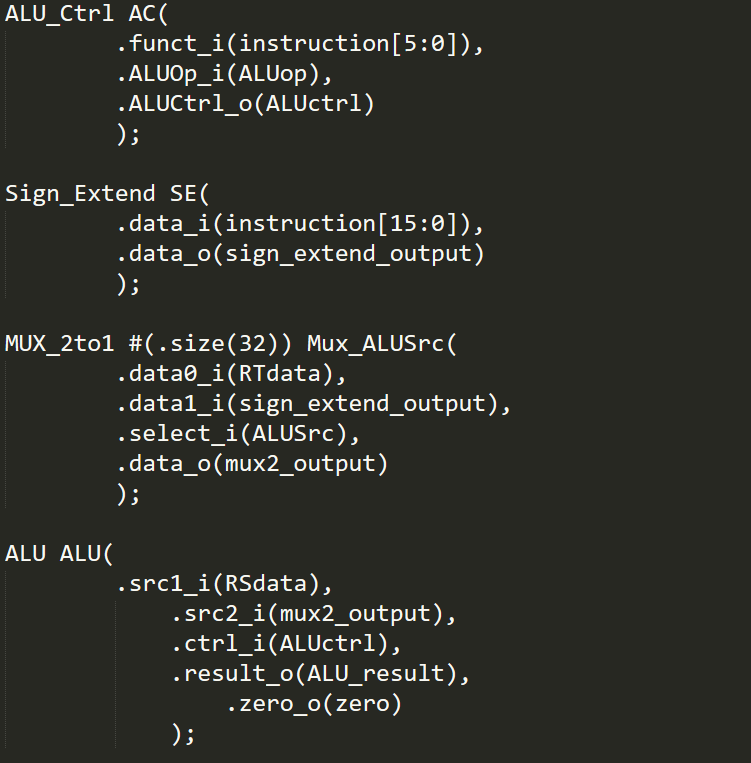
**Instruction Memory 的input是 PC的output，output是指令。**

****

**第二個mux(也就是Mux\_Write\_Reg)的input接指令的20~16 bit，和指令的15~11bit, 用decoder的regdst作為二選一的選擇依據。**

**Register file的input用指令的25~21bit和20~16和Mux\_PC\_Source的輸出，並依據decoder傳入的regwrite來判斷是否要寫入值，並接alu的結果來存結果值，輸出rsdata，rtdata分別是兩個要做運算的operand。**

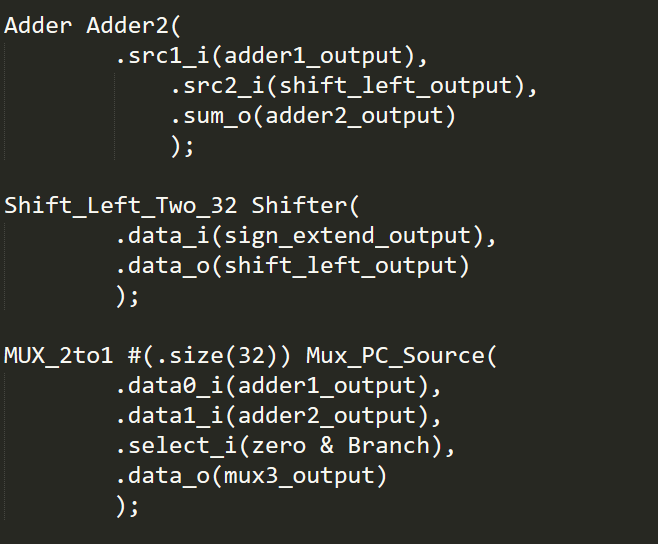
**Decoder就根據op code把判定好的值分別給定好**

****

**Alu\_ctrl的input是指令的5~0bit。它是第二層，把第一層decoder設好的alu\_op作為input來設r\_format和i\_format的值**

**Sign\_extend: 把指令的15~0bit(也就是i\_format的immediate)部分從原本的16位元擴增到32位元。**

**Alu要做運算的兩位元分別是rsdata和Mux\_Write\_Reg的output(判斷是r-format抑或i-format)，然後作運算把結果存入alu\_result，並設值zero作為是否branch的依據。**

****

**Problems you met and solutions:**

1. **接線的時候，不知道誰要連誰(誰的output要給誰的input之類的~)**

* **Sol: 對照作業以及上課投影片**

1. **Alu\_control的設值出錯(因為沒有考慮好所有狀況)**

* **Sol: 畫真值表找不同處作為設值的依據**

1. **都寫好後，發現值有傳錯， 該寫值的地方沒有寫，接線接錯導致答案錯誤**

* **Sol: 把欲檢驗的參數(input和output)分別印出，跟手寫的指令輸出一一對應，找出錯誤點，和partner討論應該更改哪裡**

1. **把所有的檢查覺得應該都沒錯誤時，印出來的結果仍舊錯誤，發現第一個alu的輸出並沒有正確的寫進register file的RDdata\_i裡面(但往後的都有)**

* **Sol: 經討論之後，認為是clock時間點的問題，但又不知道該怎麼辦，**

**只好試試把ALU換成上次作業寫好的alu跟alu\_top(本來是用ppt appendix提供的那個ALU)，結果就過了(到現在還是不明白為甚麼!!!)**

**Summary:**

**心得**

**經過兩次作業的洗禮之後，發現寫作業有五分之四的時間是用來找錯誤的，就算很快就寫好了，但結果往往是錯的，沒考慮到的地方一一浮現，這時候團隊合作就很重要，大家一起討論想出問題點，一個人負責印值看錯誤，一個人負責解碼用紙筆trace每個指令，然後問題便一個一個解決，這是很珍貴的體驗。在lab2時，當初lab1遇到的問題沒有再出現了，但為何錯仍舊是找了好久好久，為了找出錯誤，幾乎每個檔案都display好多東西，然後一一去對照(超痛苦，而且每要display不一樣的東西，整個又要重新simulate一次)，一個晚上就寫好了的code，卻花了兩天來debug(而且是兩個人喔!)，只能說團隊合作真心重要!!!**