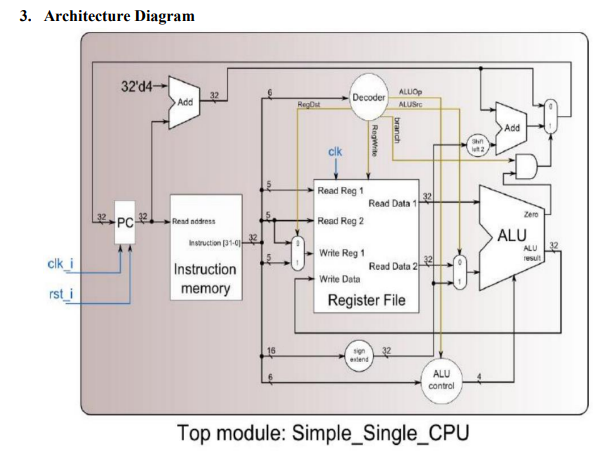
**Computer Organization**

**0616018林哲宇, 0616032 張哲銓**

**Architecture diagram:**



**Detailed description of the implementation:**

主幹是 Simple\_Single\_CPU，裡面寫著每一個模組要做的事情。

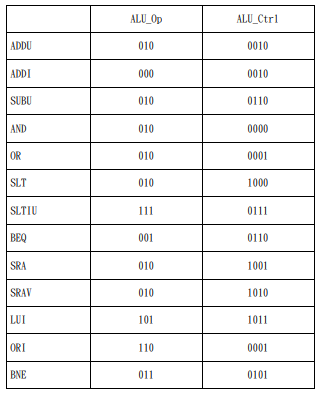
最一開始是 Instruction memory 的部分，它會看目前的 PC 來決定現在要給哪一個 instruction

接著是 decoder，在這裡，會把輸入的 32 位元的 instruction 做分析的動作，根據最高的六位 opcode 決定 RegDst, RegWrite, Branch, ALUSrc，剩餘的26位要看是哪個 operation 才知道代表什麼。

　　再來是有關 PC 的部分，首先每執行一個 instruction 後都要 PC+4，接著要看 Branch 和 alu 輸出的 zero 是否都為 1，如果是的話還要把 PC+4 再加上 instruction 的最低十六位的 sign extended 後的 shift left 的值；如果是0就 PC+4 就對了。

　　然後關於 register 的部分，首先是要知道它是 R-type 還是 Itype 來判斷 RegDst 是 rd 還是 rt，接著就把 rs, rt 分別取出來準備給 ALU 使用，在 ALU 運算完之後，如果 RegWrite 是 1，就要寫進去。

最後是 ALU，藉由 ALUCtrl 來判斷要執行的是哪種運算，在這裡實作運算的結果以及判斷 zero。下圖是ALU\_Ctrl 對應的 ALU\_Op



**Problems encountered and solutions:**

1. 要知道每個 opcode 分別要讓 RegDst, RegWrite, Branch, ALUSrc 等於 0 還是 1

1-sol. RegDst 就看是不是 R-type，是的話 RegDst 就是 1，否則為0

只有 bne, be 的 RegWrite 是 0，其餘都是1

只有 bne, be 的 Branch 為1，其餘為0

只有 sltiu, addi, lui, ori 的 ALUSrc 是1，其餘為0

1. sign extended 要補 0 還是 1

2-sol. 基本上 unsigned 的都是補0，比較特別的是 ori 也是補0

1. SRA 比較特別，因為它的 src1 不是 rs，而是後面的 shamt

3-sol. 特例化，判斷如果是 SRA 的話，就讓 src1 等於 instruction [10:6]

**Lesson learnt (if any):**

詳細的了解 CPU 內部實作原理，從讀取指令、分析指令到運算出結果並且回傳至暫存器，期望之後能實作memory 的部分。