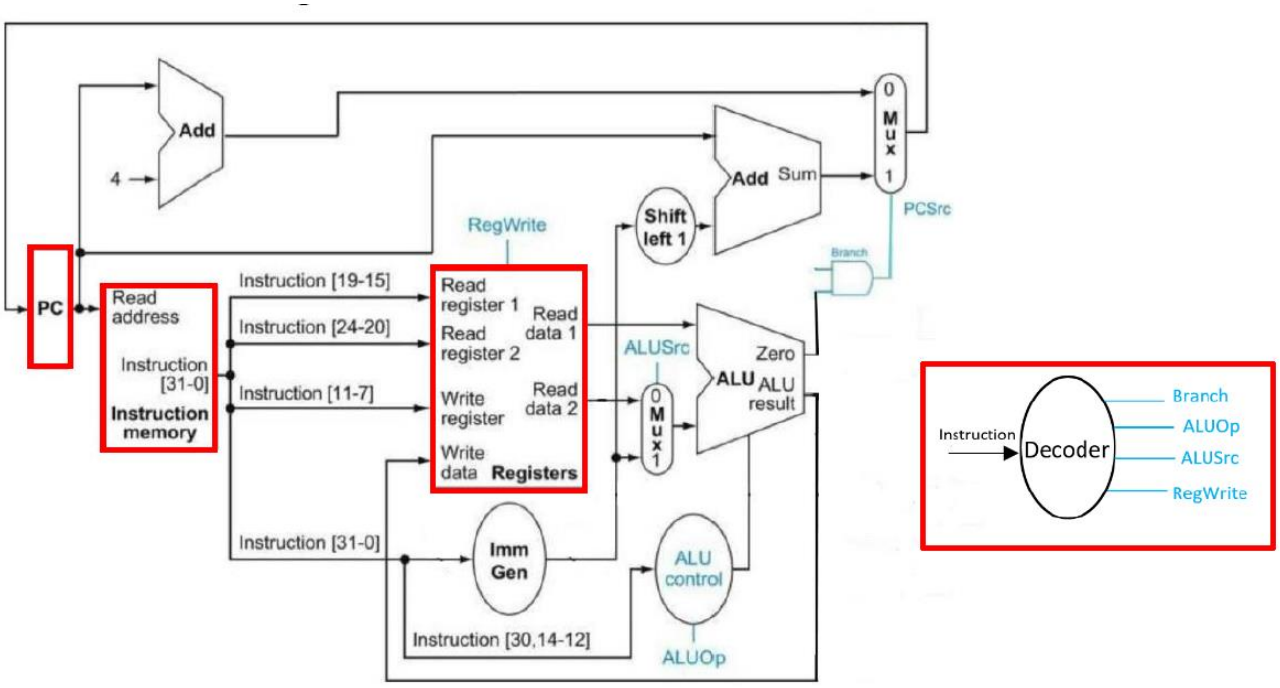
Computer Organization

Architecture diagram:



Detailed description of the implementation:

1. ALU

利用 input 的 4-bit ALU_control 訊號，完成不同控制。

除了 Lab2 的 7 個基本指令，這次 ALU 還新增了 4 種新指令如下；

4’b0011: result = a – b　　// branch not equal 會進行減法

4’b1110: result = a ^ b　　// exclusive or

4’b1111: result = a << b // a shift left b-bit

4’b1001: result = a>>>b // a shift right arithmetic b-bit

Zero detection 輸出訊號則較麻煩，當 result == 0 時，bne 要把 Zero 設為 0，其他都設為 1；當 result != 0 時，bne 的 Zero 要設為 1，其餘設為 0。我們這組利用 ALU_Ctrl == 4'b0011 來判斷是否為 bne。

補充說明，我們把 input 的 src1 和 src2 轉成 signed number，這樣才能利用 behavioral 的方式進行 shift left logical 和 shift right arithmetic 的指令。

2. ALU_Control

所有指令可以利用 ALUop 訊號，分成 3 種：load/store、Branch、ALU 運算

(a) load/store：**I-type(load) & S-type**

input：ALUop = 00，output：ALU_Ctrl = 0010，因為 load 和 store 都在 ALU 裡面進行加法。

(b) Branch：**beq & bne**

input：ALUop = 01，output：ALU_Ctrl = 0010，因為所有的 Branch 指令在 ALU 裡面都進行減法。

(c) 算術運算：**R-tpye & I-type**

input：ALUop = 10 && Funct3，利用 ALUop 和 Funct3 (instr[2:0])，依據不同指令對 ALU_Ctrl 給予不同訊號。

補充：instr[3] = instruction[30]；instr[2:0] = instruction[14:12] (=Funct3)

3. Imm_Gen

面對的指令可分為 I-type 和 B-type，我們在元件內部建立一個內部訊號 Type，當指令是 I-type 時，Type 設為 1，其他設為 0。

I-type 指令只要把輸入的 12-bit 依序放到輸出 Lsb 的 12-bit 即可；

B-type 因為指令的 12-bit 是拆散的，需要依照指令格式對應回來比較麻煩。

最後再把 immd[31:12]依正負號判斷(instr[31])用 sign-bit 補滿即可。

Implementation results:

Test_data_1

```
#    Time: 0 ps  Iteration: 0  Instance: /testbench/cpu/IM
# r0 =   0, r1 =  21, r2 =   9, r3 =   1,
# r4 =  20, r5 =   1, r6 =   0, r7 =   0,
# r8 =   0, r9 =   0, r10 =  0, r11 =  0
# ** Note: $stop    : C:/Users/Heyward/Desktop/lab3/testbench.v(35)
#    Time: 1010 ns  Iteration: 0  Instance: /testbench
# Break in Module testbench at C:/Users/Heyward/Desktop/lab3/testbench.v line 35
```

Test_data_2

```
VSIM 2> run -all
# r0 =   0, r1 =   0, r2 =   0, r3 =   0,
# r4 =   0, r5 =   0, r6 =   2, r7 =   5,
# r8 =   7, r9 =   9, r10 =  0, r11 =  0
# ** Note: $stop    : C:/Users/Heyward/Desktop/lab3/testbench.v(35)
#    Time: 1010 ns  Iteration: 0  Instance: /testbench
# Break in Module testbench at C:/Users/Heyward/Desktop/lab3/testbench.v line 35
```

Test_data_3

```
VSIM 2> run -all
# r0 =   0, r1 =   0, r2 =   0, r3 =   0,
# r4 =   0, r5 =   0, r6 =   0, r7 =   0,
# r8 =   0, r9 =   0, r10 =  2, r11 =  2
# ** Note: $stop    : C:/Users/Heyward/Desktop/lab3/testbench.v(35)
#    Time: 1010 ns  Iteration: 0  Instance: /testbench
# Break in Module testbench at C:/Users/Heyward/Desktop/lab3/testbench.v line 35
```

Problems encountered and solutions:

一開始照著講義寫，忘記把 xor, sll, sra 等新的指令寫進 ALU_Ctrl 裡面，測試半小時才發現忘記寫進去；Immediate Generator 有兩種定址模式，我們也想了很久，才想到把進來的種類分成 I-type 和 B-type，再依據不同情況把 Immediate 產生，至於正負號我們也爭論了一下到底要不要考慮，因為兩種 type 正負號的 Sign-bit 在 instruction 的不同位置，最後才想到先處理 LSB 的 12-bit，再一口氣處理 Sign-bit 的問題。

最後測試時發現 Decoder 會把 slti 的 ALUop 設成 00，幸好助教即時發布更新的 Decoder 讓我們免於 Debug 的地獄中。

Comment:

這次作業其實比上次簡單，因為助教幫我們把最難的 Decoder 寫完了(灑花)，我們只要把 ALU 新增一些指令，再把線接一接就大致完成了，而且在實作的過程中把 Datapath 和 Controlpath 弄得更清楚了，也對 ALUop、ALU_ctrl 和 ALU 之間的關係更加明瞭。