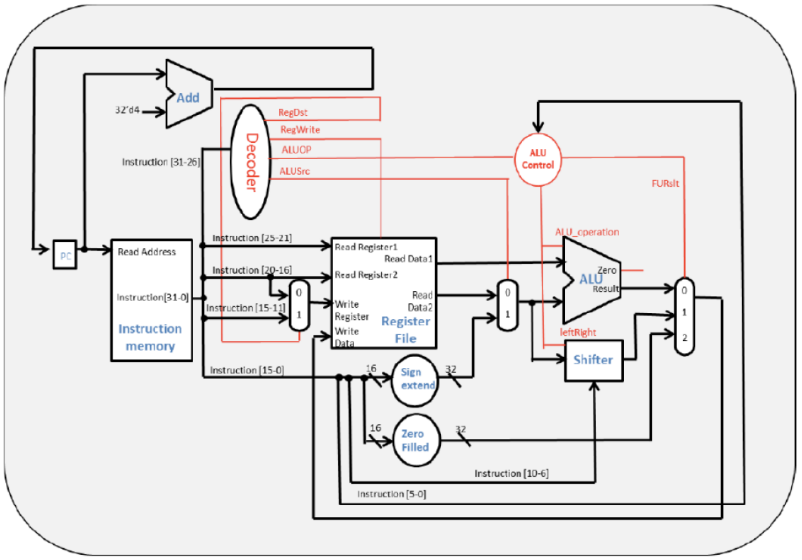
Computer Organization

1. Architecture diagrams:

這次作業我是以右圖的線路流程來完成程式碼。
由於本次作業可以忽略**Zero Filled** 的實作，所以那段線路可以不用參考。



ALU Control 那邊有拉一條 **ALU_operation** 和 **leftRight** 實作方式是以**ALU_operation** 的第**3**個**bits**當作**leftRight** 的判斷條件。

2. Hardware module analysis:

1. Adder.v :

此**module**為實行**PC + 4** 的功能，實作方法即為將兩個**input(32bits)**的數值做相加。

2. MUX2to1 :

此**module**為**2**擇**1**多工器，實作方法為利用**1 bit** 的線路當作**select line**，如果**select line** 的值為**0**，則輸出**input0**的值，反之，**select line** 為**1**的話，則輸出**input1**的值。

3. Decoder :

此**module**的實作方法為根據**instruction[31:26]**的**bit**去決定各個**control line(RegDst, RegWrite, ALUOP, ALUSrc)** 的值為何。

4. ALU_Ctrl：

此**module**的實作原理為根據**ALUOP**的值和**instrution[5:0]**的**bit**去決定出**ALU**需要作甚麼運算。比較特別的是，和課堂講義不同在需要多輸出**1**條**FURslt**去當作接下來**3to1MUX**的**select line**。

5. Sign_Extend：

此**module**的實作原理為讓**I-format**的**address**值**(16bit)**去**extend**成**32bit**，因為**sign integer** 的特性，利用最高位元去判斷正負性。

6. ALU：

此**module**是以助教的**template**為模板，因此不多加闡述。

7. Shifter：

此**module**是以**leftRight**的值去判斷要左移或右移，並以**instruction[10:6]**的值**(shamt)**去做出要移幾個位元。

8. MUX3to1：

此**module**為**3**擇**1**多工器，實作方法為利用**2 bit** 的線路當作**select line**，如果**select line** 的值為**00**，則輸出**input0**的值；**select line** 為**01**的話，則輸出**input1**的值；**select line** 為**10**，則輸出**input2**的值。

9. Simple_Single_CPU：

此**module**為將所以會用到的**wire**都寫入到上述的**module**裡。

3. Finished part:

本次作業完成了所有**module**，並且通過了所有測資。

4. Problems you met and solutions:

1.

本次作業最大的問題還是對於**verilog**的使用不是很熟練，但是透過上網查詢得以成功解決，也因此學到更多的技巧。

2.

這次執行**simulation** 的時候，經常出現上述圖片的錯誤訊息，一開始檢查各個**.v**檔是否有寫錯，後來重新啟動**vivado**後就能成功執行。

5. Summary:

本次作業讓我更加熟悉**CPU**的運作，然而這次實作的**CPU**仍有侷限性，例如不能執行**lw, sw ,beq**等功能，但就結果來講，讓我更認識**CPU**了。