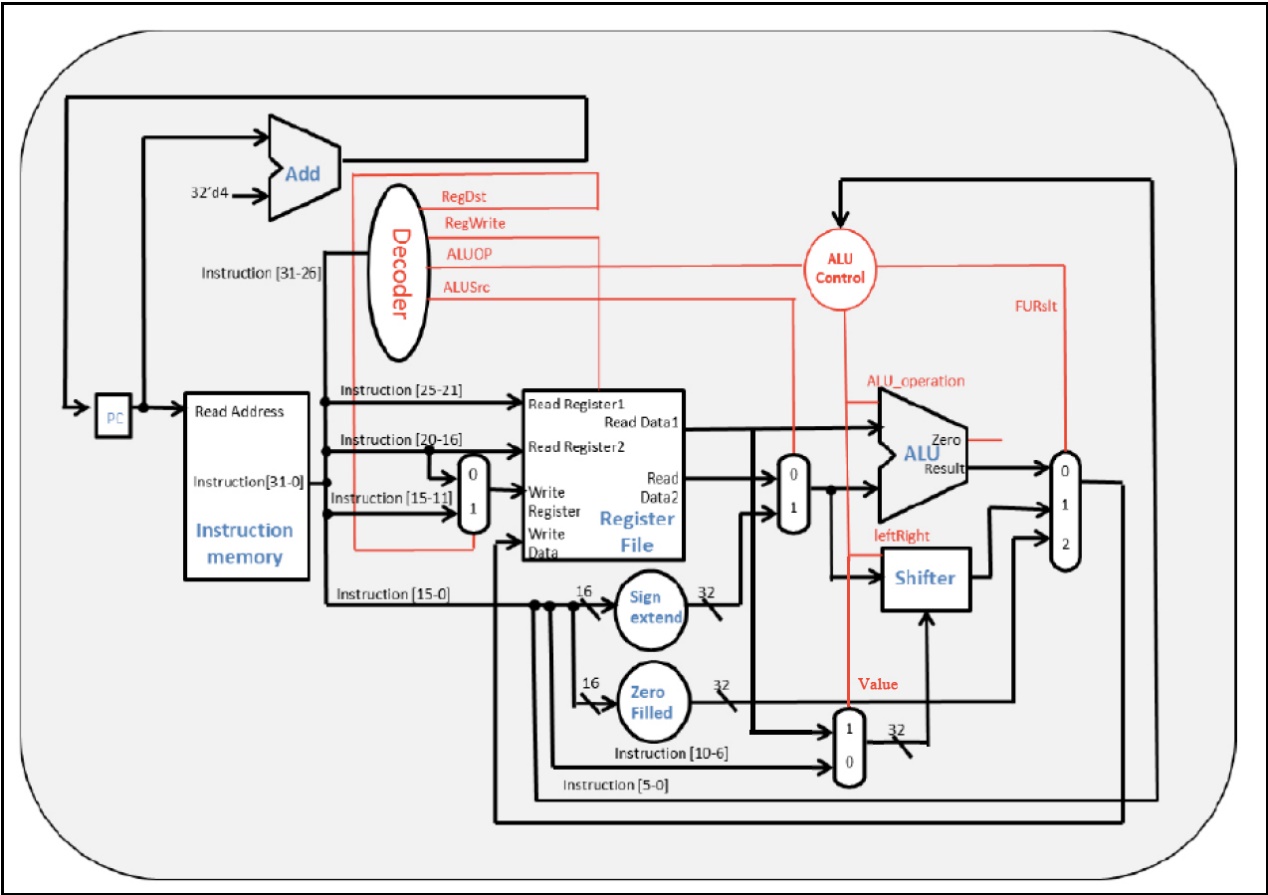
**Computer Organization**

**Architecture diagrams:**

****

**Hardware module analysis:**

1. Adder功能 : 計算下一個PC = PC+4 ，因為這次沒有branch指令，所以所有的指令完成後下一個指令的位置一律是PC+4。
2. Decoder 功能 : 讀取opcode並發送各種跟資料輸入有關控制訊號
   1. Rtype opcode:

write = 1; // 1 => 要儲存運算結果

op = ALUOP\_RTYPE;

src = 0; // 0 => 用Reg當第二個數入來源

Dst = 1; // 1 => 用Rd來儲存

* 1. Addi opcode:

write = 1; // 1 => 要儲存運算結果

op = ALUOP\_ADDI;

src = 1; // 1 => 用Imm當第二個輸入來源

Dst = 0; // 0 => 用Rt來儲存

* 1. Lui opcode:

write = 1; // 1 => 要儲存運算結果

op = ALUOP\_LUI;

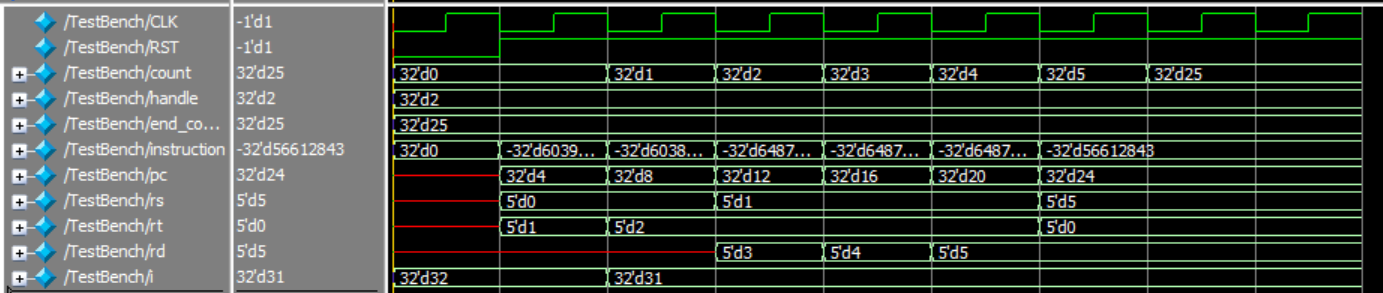
src = 1; // 1 => 用Imm當第二個輸入來源

Dst = 0; // 0 => 用Rt來儲存

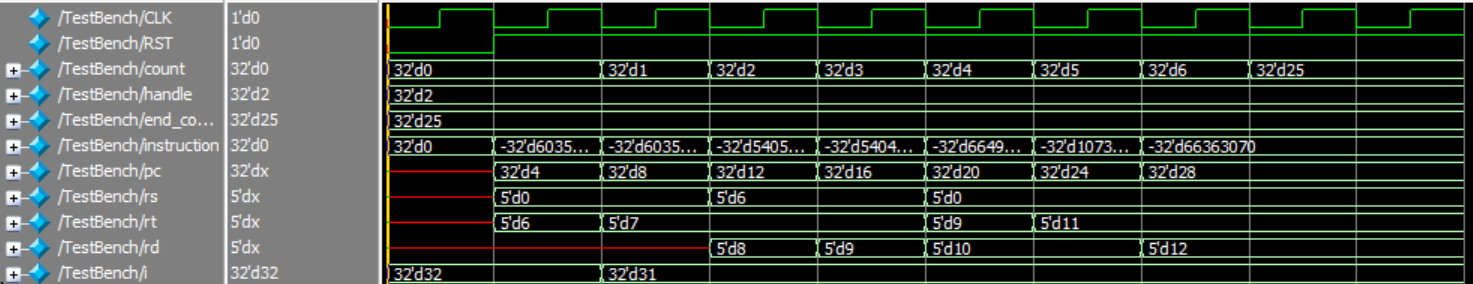
1. ALU Controle 功能 : 接收ALUOP控制訊號並進一步轉換成與資料運算有關的訊號
   1. Alu operation :給定每種指令一個對應的alu運算方式。
   2. LeftRight :1向左shift(sll,sllv)，0向右shift(srl,srlv)，設計用alu operation[0]來判斷
   3. Value :0用Imm來shift(sll,srl)，1用$rs來shift(sllv,srlv)，設計用alu operation[1]來判斷
2. Sign extended 功能:複製原訊號的最高位bit(MSB)到前16bit
3. Zero filled 功能 : 用於不需要計算的lui，把輸入訊號移至前16bit，並將後16bit填0
4. Shifter 功能 : 將$rt依照給定的位移量跟位移方向做位移

**Finished part:**

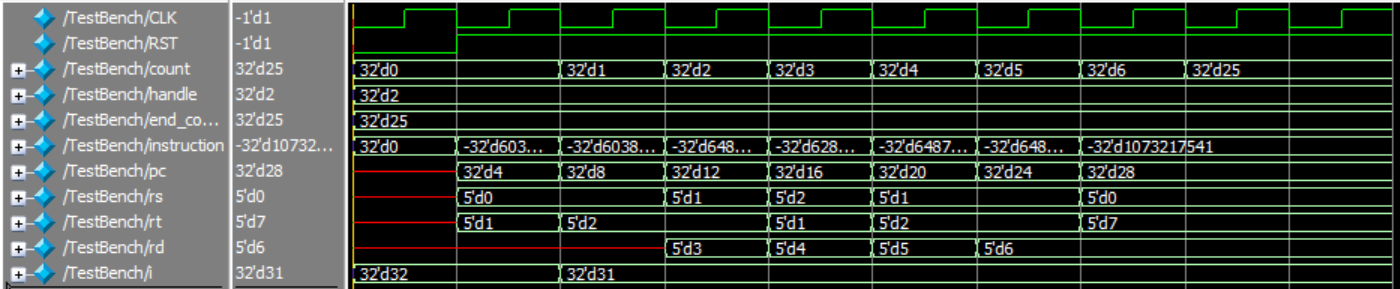
**Test data1**



**Test data2**



**Test data3**



**Problems you met and solutions:**

其實這次的作業寫verilog沒什麼很困難的地方，比較蠢的是我一開始做加減法沒有加上$signed()，所以算到負數就會錯，主要是要花很多時間搞清楚每個module負責的功能、cpu運作的每一個過程。從解讀指令到控制alu運算每一個控制訊號都要了解，花了很多時間。

**Summary:**

經過這次的作業我認識了一個簡易cpu完整的運作過程，從指令讀取到解碼，從發送控制訊號到運算，每一個步驟都有更深入的了解。