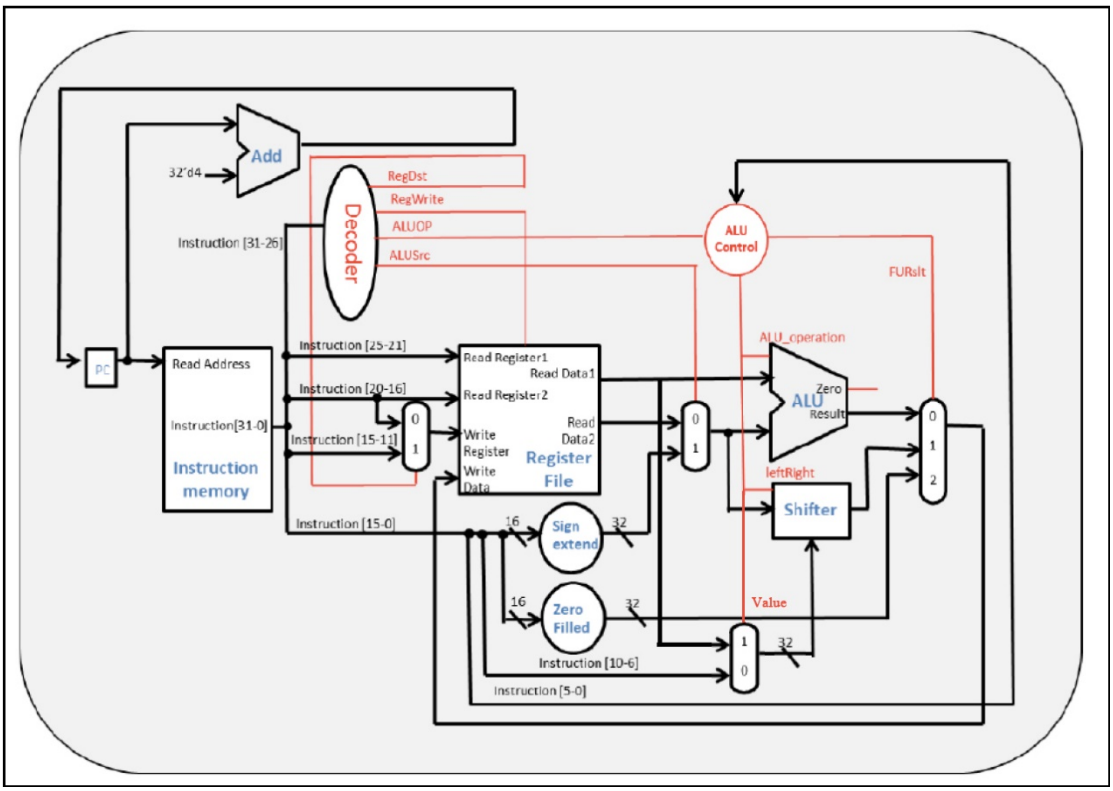
Computer Organization

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
   I. Rtype opcode:
      i. write = 1; // 1 => 要儲存運算結果
      ii. op = ALUOP_RTYPE;
      iii. src = 0; // 0 => 用 Reg 當第二個數入來源
      iv. Dst = 1; // 1 => 用 Rd 來儲存
   II. Addi opcode:
      i. write = 1; // 1 => 要儲存運算結果

ii. op = ALUOP_ADDI;

iii. src = 1; // 1 => 用 Imm 當第二個輸入來源

iv. Dst = 0; // 0 => 用 Rt 來儲存

III. Lui opcode:

i. write = 1; // 1 => 要儲存運算結果

ii. op = ALUOP_LUI;

iii. src = 1; // 1 => 用 Imm 當第二個輸入來源

iv. Dst = 0; // 0 => 用 Rt 來儲存

3. 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

I. Alu operation :給定每種指令一個對應的 alu 運算方式。

II. LeftRight :1 向左 shift(sll,sllv)，0 向右 shift(srl,srlv)，設計用 alu operation[0]來判斷

III. Value :0 用 Imm 來 shift(sll,srl)，1 用$rs 來 shift(sllv,srlv)，設計用 alu operation[1]來判斷

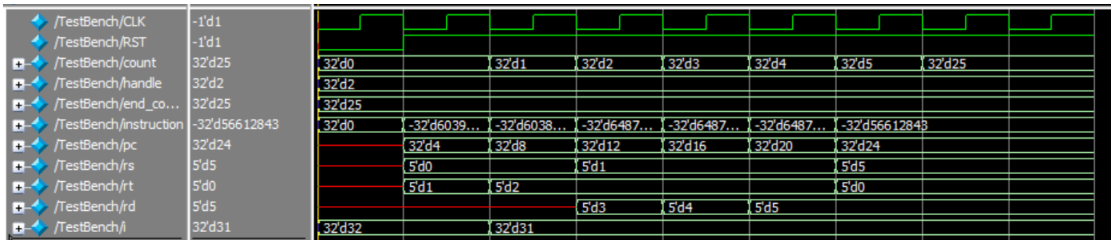
4. Sign extended 功能:複製原訊號的最高位 bit(MSB)到前 16bit

5. 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

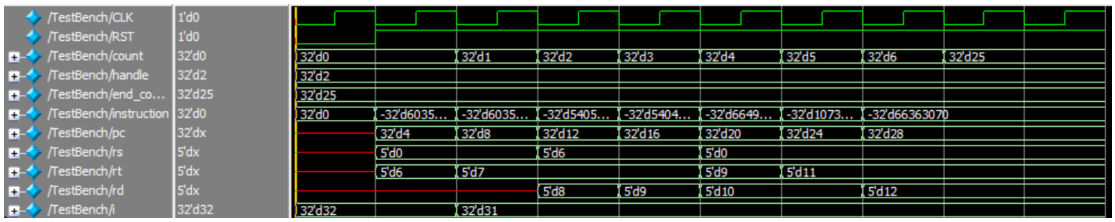
6. Shifter 功能 : 將$rt 依照給定的位移量跟位移方向做位移

Finished part:

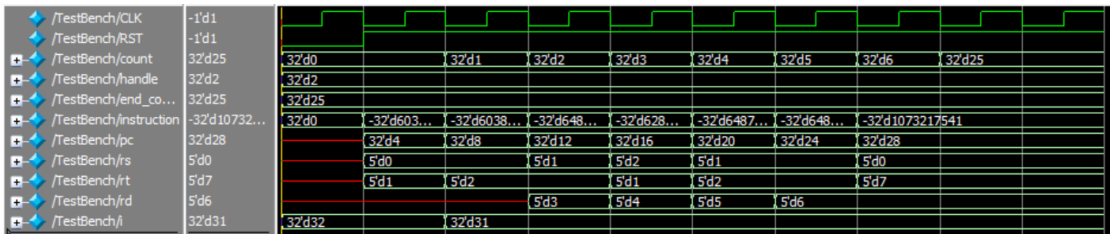
Test data1



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。