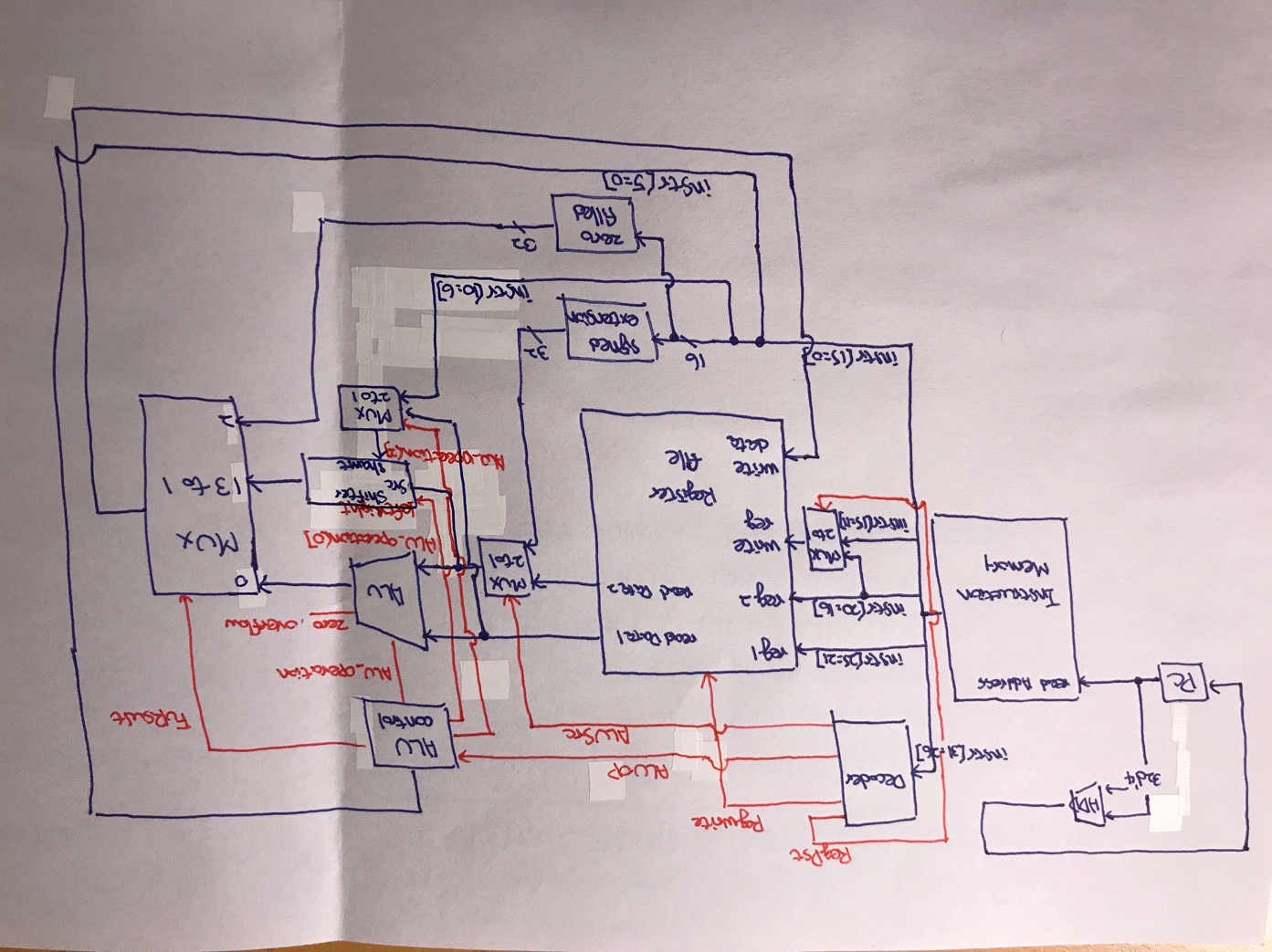
**Computer Organization**

**Architecture diagrams:**

****

**Hardware module analysis:**

**PC: Program counter, 用來指向instruction memory裡面指令的位置，從0開始每次加4。**

**Instruction Memory: 用來記憶cpu指令，測資會從此讀入cpu。**

**Register File: 儲存cpu內的reg值，模擬register的讀出跟寫入。**

**ALU: 執行從register file內讀出或是從指令讀出的數值，對其進行運算。**

**Shiftrer: 將拿到的數值進行Left 或是Right Shift並傳給Mux 3 to 1。**

**Signed Extension: 看輸入的數值最高位為1或0，而對higer 16 bits補1或0。**

**Zero Filled: 將輸入的數值往左推16個bits，而lower 16 bits補0。**

**Decoder: 將讀到的指令解析判斷是哪一種形式的指令（R type, lui 或是其他），進而得知後面的module（register file , mux ,alu等）該做什麼。**

**ALU Control: 根據decoder解析出來的ALUOP與instr[5:0] (function code)，來判斷該讓ALU進行什麼樣的功能運作。**

**MUX 2 TO 1: 總共用了三個此MUX ，從左到右的功能分別是，第一個為根據R type 或 I type決定要從instruction的哪一段指令拿直。第二個為決定ALUsrc2的值該從Reg read data 2 或 instruction的指令拿值。第三個為決定Shifter Shamt 該從Reg提出還是從instruction的指令拿值。**

**MUX 3 TO 1: 根據ALU Control 解析出來的值，告訴MUX 此指令為R type或是sll, srl或是lui等，並將正確的output給予Write Data。**

**Finished part:**

**Adder.v , ALU.v , ALU\_Ctrl.v , Decoder.v , Mux3to1.v , Mux2to1.v , Simple\_Single\_CPU.v , Shifter.v , Sign\_Extend.v , Zero\_Filled.v**

**Problems you met and solutions:**

**一開始在設計Decoder的時候，因為講義上沒有給Lui, Shifter, ADDI的指令規格該如何給ALUOP 的code，所以有點不知所措。**

**但後來再仔細看，發現原來只要稍微自己設計一下（例如Shifter, Lui的ALUOP code其實是don’t care），就可以設計一套Decoder系統出來。**

**Summary:**

**第一眼看到Lab 3其實有點害怕，雖然知道每一個小電路在做什麼，但是到Verilog上之後如何用module完美的模擬出來其實是需要一個縝密的計畫，所幸助教已經將一些需要Verilog技術性Coding的東西與架構完成，因此只需要專注在設計CPU上，看到自己設計的CPU在過了一個又一個的Testbench的時候真的備感欣慰。**