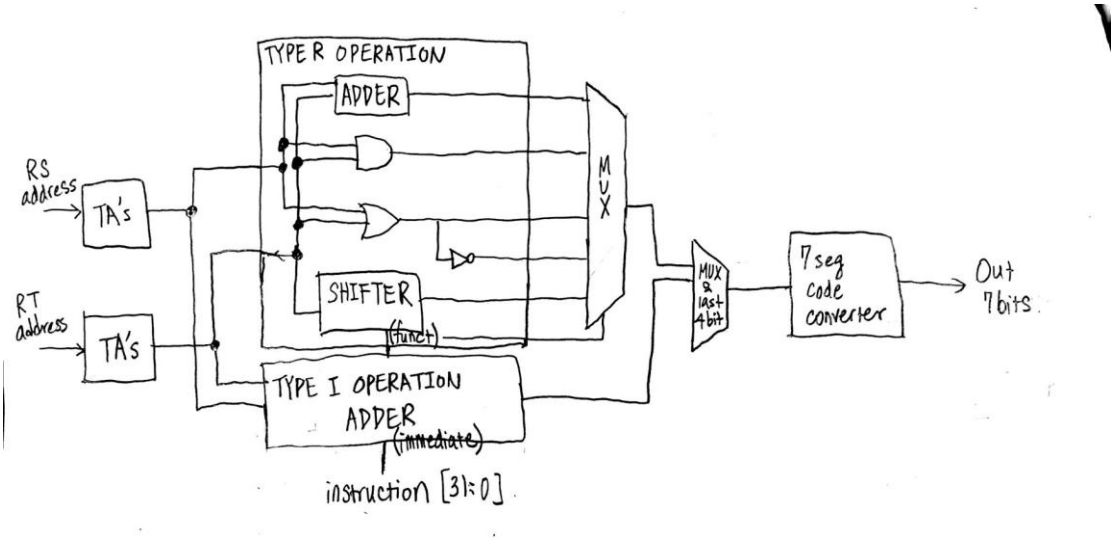
hw01: MIPS CPU + Seven-Segment Display

Block Diagram



設計概述

依照這次的設計所畫出的架構圖大致上長這樣。(如上圖)

這次的設計我從七段顯示器的轉換開始設計，接下來則是 operation 還有 funct 的 multiplexer，再來才是 Type R 裏頭各個 case 的設計。雖然這次的規模或許用不到這樣的流程，但將來在做更大的設計時我覺得除了畫架構圖外，在自製 module 的時候，從尾到頭、大到小設計會比較能讓自己搞清楚當下所進行到的設計階段。

另外，因為進來的 instruction 有長長的 32bit，在運用的時候又要分段進行指令和數值的讀取，若每次都直接用 instruction[幾 bit:幾 bit]的方式很容易眼花撩亂而且出錯率很高。

```
assign opcode = instruction[31:26];
assign rs = instruction[25:21];
assign rt = instruction[20:16];
assign shamt = instruction[10:6];
assign funct = instruction[5:0];
assign immediate = instruction[15:0];
```

所以(如上圖)我就新宣告不同名字的 logic 對 instruction 分段 assign 再利用這些新的 logic 去做運算，這樣不但易於判讀，而且在詢問老師後也確認是不會增加額外的面積或影響效率的。

在設計 shift 功能的時候，我本來是用兩個 module 分別寫 shift right 和 shift left，而且裡面還寫了 17 個 case 分別代表 shift 一個 bit、兩個 bit、……，包括 default 全部清為零(shift 超過 15 bit 就等於輸出 0)，但後來發現其實有更簡潔的寫法，而原本以為這樣的寫法 verilog 不會接受，沒想到是沒問題的，就是(如下圖)

```
always_comb begin
    case(shift_direc)
    2'b00: out_value = in_value << shift_pos;
    2'b10: out_value = in_value >> shift_pos;
    default: out_value = in_value;
    endcase
end
```

直接將 shift 幾個 bit 的變數寫在 << 和 >> 旁就可以了，這樣不但可以全部寫在一個 module 裡面分兩個 case 判斷是 shift right 或 shift left，也省了好幾列的 code，但至於實際有沒有縮小電路的面積我沒有求證，當時沒有想到拿舊的寫法去跑跑看合成，所以也沒做面積上的比對，但和原本的寫法比較，推測應該是可以增加效率的，因為就原本的兩層 case 和現在的一個 case 比，就少了一個 mux gate 要過。

合成面積

```
Combinational area:               2864.030450
Buf/Inv area:                      349.272013
Noncombinational area:               0.000000
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  2864.030450
Total area:                 undefined
1
report_timing
```

設計過程遇到的問題

這次的作業進行 run 的時候有遇到如下圖的 error: “ordered and named port list mixed syntax” 這是因為裏頭在做 name mapping 的時候所做的命名和模組的名字一樣所導致的錯誤，對於這點我本來還不知道也沒有注意，在未來設計中會更加小心。

```
ncvlog: *E,PLMIXU (mips.sv,40|86): ordered and named port list mixed syntax [12.3.3][12.3.4(IEEE)].
(`include file: mips.sv line 40, file: TESTBENCH.sv line 5)
opcode_mux opcode(.typeR(R_out), .typeI(I_out), .opcode, .last_four_bits(four_bit_output);
```

心得

對於這次的作業經驗，我歸納了幾點讓自己在未來進行設計時能多加注意和參考:

1. 設計前先搞清楚設計要求，才不會誤打誤撞
2. 設計前先畫架構圖和決定 code 順序
3. 在不影響效率的前提下，盡可能地用 high level 的寫法，比較明瞭清楚
4. 命名的時候注意不要有撞名或是有 bit 不對的狀況