1. 背景

這學期的計算機組織為先前數位邏輯設計和組合語言&嵌入式系統的課程內容之延伸，以教導MIPS處理器的運作模式、電路設計架構為主要方向，同時帶入了以硬體描述語言 Verilog 來實作邏輯電路設計的教學內容。

本次作業為實作一個管線化的MIPS處理器，可以處理算術、邏輯比較、移位運算；載出、存入、分支；以及跳躍指令。作業要求中的16道指令可分為 R-Format (add, sub, and, or, sll, slt, multu, mfhi, mflo, jr)、I-Format (andi, lw, sw, beq)、J-Format (j) 三種格式的指令，以及nop。MIPS 的特色為所有指令統一為 32 位元寬度，讓執行上更有效率，統一的格式也能讓元件設計更加便利。

管線化的設計可讓原本只能一次運行一道指令的性質，變成依照設計原理去切成多個執行階段，多道指令可以依序同時在處理器上運行，增加執行速度。

2. 方法

處理器需要遵照五大指令執行原則 : 將記憶體內的指令取出，透過控制單元與檔案暫存器進行解碼後，視指令執行需求運用 ALU 做運算，或只運用其餘加法器元件做位址計算。若有讀出或寫入操作，則需要存取資料記憶體。最後將執行結果寫回檔案暫存器與更新pc incrementor。

設計重點說明：

- 參考課本的兩個流程圖 (4-2 p21、4-4 p9) 畫出流程圖。
- 透過流程圖一筆一筆的對應切pipelined。
- 將 jr、jump、branch 指令移到第二層 (ID)，讓 pc 流暢的更新。
- 以 nop 指令解決相關 Hazards

3. 結果

測試資料：

instr_mem：

```
// beq  $s3, $s4, 3
03
40
98
12
// add  $s2, $s0, $s2
20
88
30
02
// lw   $s1, $t7, 0
00
00
2F
8E
// sub  $s5, $s2, $s5
22
50
35
02
// j    5
05
00
00
08
// sll $t0, $t1, 4
00
20
09
00
// or   $s3, $s0, $s3
25
80
33
02
// sw   $zero, $s2, 24
18
00
12
AC
// andi $t0, $t1, 15
0F
00
28
31
// jr  $ra
08
00
E0
03
```

模擬結果：

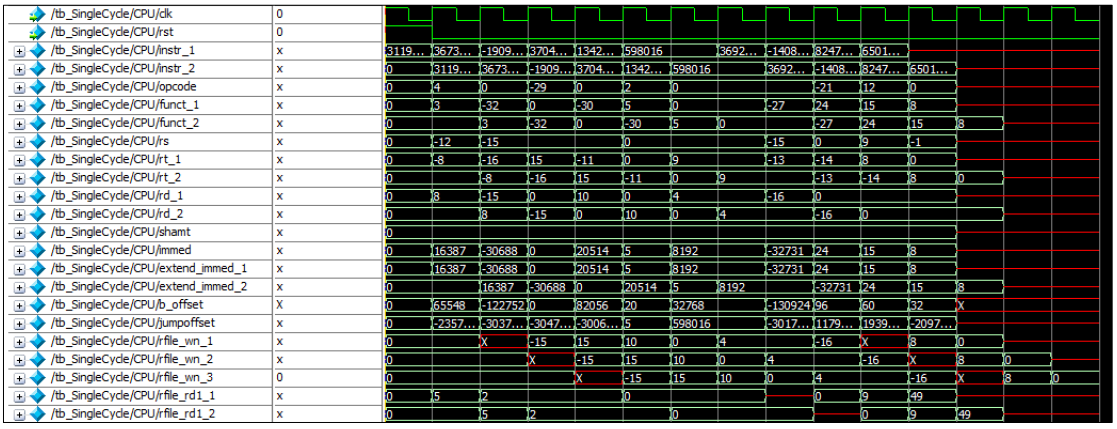
```
#                    0, reading data: Mem[          x] =>          x
# 18446744073709551615, PC:          x
#                    0, reading data: Mem[          0] =>  311967747
#                    0, reg_file[ 0] =>          0 (Port 2)
#                    0, reg_file[ 0] =>          0 (Port 1)
#                    0, PC:          0
#                    0, wd:          0
#                    0, NOP
#
#                    1, reading data: Mem[          4] =>   36735008
#                    1, reg_file[24] =>         36 (Port 2)
#                    1, reg_file[20] =>          5 (Port 1)
#                    1, PC:          4
#                    1, BEQ
#
#                    2, reading data: Mem[          8] => 2385444864
#                    2, reg_file[16] =>          1 (Port 2)
#                    2, reg_file[17] =>          2 (Port 1)
#                    2, PC:          8
#                    2, wd:          x
#                    2, ADD
#
#                    3, reading data: Mem[         12] =>   37048354
#                    3, reg_file[15] =>         21 (Port 2)
#                    3, PC:         12
#                    3, LW
#
```

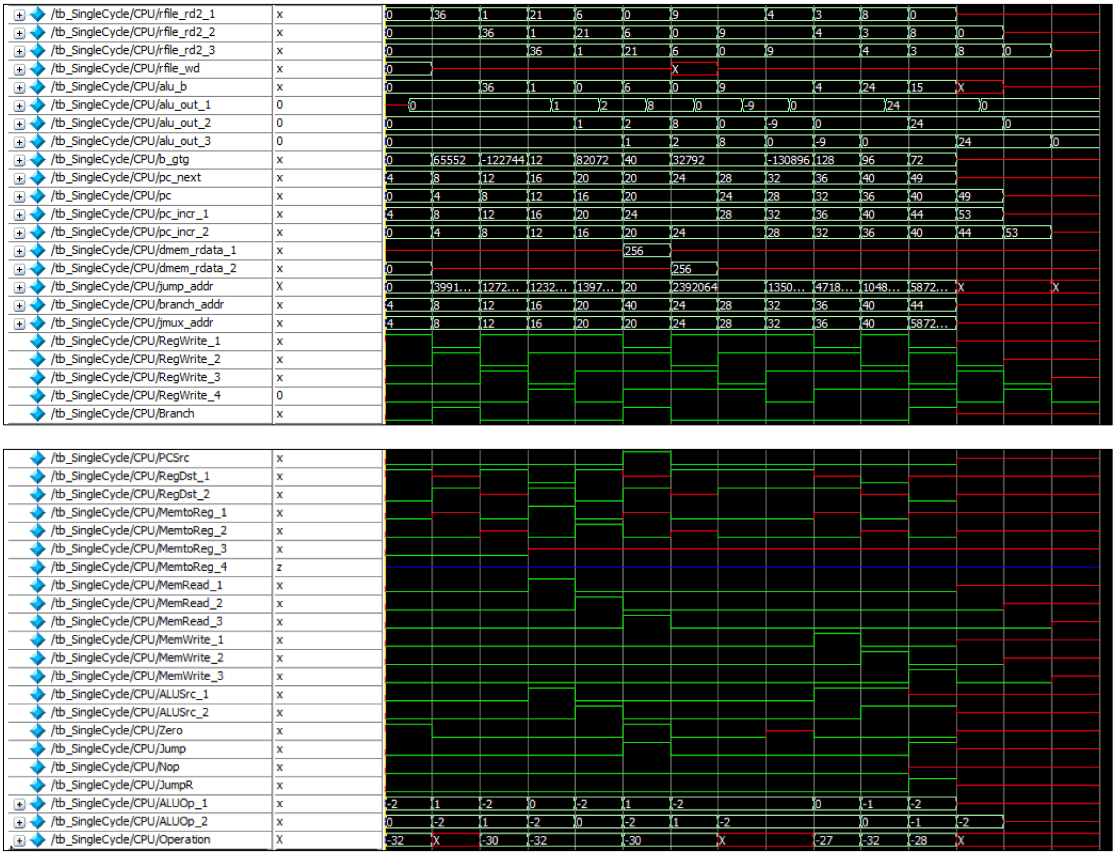
```
#                    4, reading data: Mem[         16] =>  134217733
#                    4, reg_file[21] =>          6 (Port 2)
#                    4, PC:         16
#                    4, wd:          x
#                    4, SUB
#
#                    5, reading data: Mem[         20] =>     598016
#                    5, reg_file[ 0] =>          0 (Port 2)
#                    5, reg_file[ 0] =>          0 (Port 1)
#                    5, reading data: Mem[          2] =>        256
#                    6, reg_file[17] <=          x (Write)
#                    5, PC:         20
#                    5, J
#
#                    6, reg_file[ 9] =>          9 (Port 2)
#                    7, reg_file[15] <=          X (Write)
#                    6, PC:         20
#                    6, wd:          X
#                    6, NOP
#
#                    7, reading data: Mem[         24] =>   36929573
#                    8, reg_file[10] <=          x (Write)
#                    7, PC:         24
#                    7, wd:          x
#                    7, NOP
#
```

```
#                    8, reading data: Mem[        28] => 2886860824
#                    8, reg_file[17] =>          x (Port 1)
#                    8, reg_file[19] =>          4 (Port 2)
#                    8, PC:        28
#                    8, wd:          x
#                    8, OR
#
#                    9, reading data: Mem[        32] =>  824705039
#                    9, reg_file[18] =>          3 (Port 2)
#                    9, reg_file[ 0] =>          0 (Port 1)
#                   10, reg_file[ 4] <=          x (Write)
#                    9, PC:        32
#                    9, SW
#
#                   10, reading data: Mem[        36] =>   65011720
#                   10, reg_file[ 9] =>          9 (Port 1)
#                   10, reg_file[ 8] =>          8 (Port 2)
#                   11, reg_file[ 4] <=          x (Write)
#                   10, PC:        36
#                   10, ANDI
#
#                   11, reading data: Mem[        40] =>          x
#                   11, reg_file[ 0] =>          0 (Port 2)
#                   11, reg_file[31] =>         49 (Port 1)
#                   12, reg_file[16] <=          x (Write)
#                   12, writing data: Mem[        24] <=          3
#                   11, PC:        40
#                   11, wd:          x
#                   11, JR
#
```

```
# control_single unimplemented opcode  x
#                   12, reg_file[ x] =>          x (Port 1)
#                   12, reg_file[ x] =>          x (Port 2)
#                   12, PC:        49
#                   14, reg_file[ 8] <=          x (Write)
#                   13, PC:         x
```

Waveform：





4. 討論

設計上的挑戰：

作業上對 Verilog 程式的編寫方式有特別規範，因此會需要將老師原本提供的範本給大幅修改，寫出的成品在功能上會與範例原版大同小異，但實際上架構相差甚遠。為了思考出新的編寫方式，以及整合大幅修改後的檔案，就會耗費額外的心力來完成。

舉例來說，原版 MIPS 處理器中的架構是 single cycle；作業則需要運用暫存器隨著時脈訊號去控管指令執行順序，達到管線化設計，因此電路的結構與設計上有更多細節，資料傳輸、運算的過程也會步驟更多。

程式驗證和修正：

除了運用編譯器做為第一關的程式碼錯誤檢查，也檢查模擬的執行結果。檢視波型圖、訊號數值與 transcript 上的執行輸出訊息，以此評估是否有其他編譯器無法指出的邏輯錯誤，或是元件訊號沒有在 modules 之間

正常傳輸的錯誤。

5. 結論

此次作業除了原先範例提供的 module，加上期中作業與管線化所設計的元件，總共使用了 25 種 module，每個元件該具有的功能必須都齊全。

除此之外，題目、講義沒提到的部分也不該加。規範編寫方式的用意，是為了能以課堂強調的觀念去理解 MIPS 處理器的結構設計。另外，也可以訓練學生以不同之電路描述方式，實作出具有原先基礎功能，並且還有額外功能的處理器。

程式範例大略展示了原始 single cycle 處理器內部的運作流程、元件輸出結果和用途。而完成作業的思考迴路應是只把理念拿來參考，實際作品會依照講義或題目說明指定的概念為準。