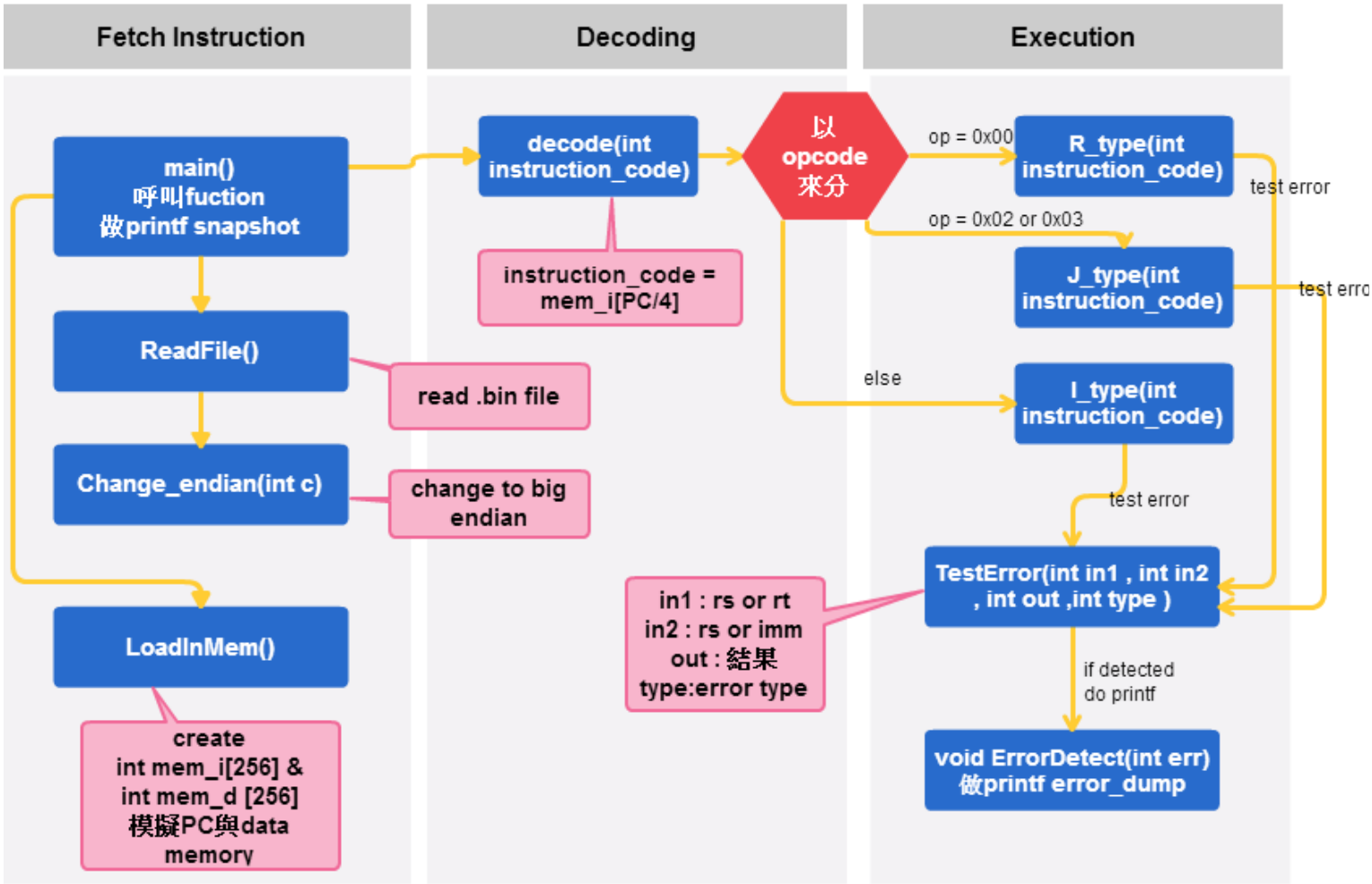
Project 1：single _ cycle Report

102062222 張心愉

● Simulator design & elaboration

Simulator 流程簡圖 ：

說明：最上面灰色方塊為 simulator 階段，藍色方塊為程式內 function 與簡略說明，粉紅色方塊為對該方塊的 comment，紅色方塊為如何 decode 出三種 type 指令，黃色線段為呼叫被指向的 function（以下會有詳細說明）。



Code design 設計說明：

進入 main 之前，宣告所有會用到的 function 與全域變數，像是存 register 的 reg[32]，計算 program counter 的 PC，還有存 iimage 與 dimage 的 data_i [256], data_d[256]，此為 int array，所以共可以存 1024 byte，符合題目要求，並在 main 一開始就初始為零。

main()

在 main 的一開始，就直接 call **ReadFile()**。

ReadFile()——利用 fread 讀出.bin 檔，並檢測是否有讀到檔案，若檔案指標是 null 則 exit，將讀進的資料以 4 byte 為一單位存入 int 指標（此時還未存進 data_i [256], data_d[256]）。接著會偵測該系統是 little endian or big endian，若不是 big endian 就 call Change_endian(int c) 轉換成 big endian，最後 fclose 讀進來的.bin 檔。

讀檔結束跳回 main 後將 data_i [256], data_d[256] , cycle_count , PC 等會用來計算的值初始為零。並呼叫 LoadInMem()。

LoadInMem()——先找出共要讀幾個指令，檢查指令數量是否大於 1k，若是合法測資則用 for loop 將資料一個一個存進 data_i [256], data_d[256]。

確定讀入檔案的資料都存進來之後，就進入 while loop 無窮回圈，（op 為 halt 會 break），因為題目要求再做第一個指令之前就要先印 cycle 0，所以先做 printf 到 snapshot.rpt。接著檢查 cycle 與 PC 是否合法，然後進入 decode(data_i[PC/4])。

decode——對讀進來的 instruction_code 做 shift 找出 opcode，然後分辨出是哪種型態的 MIPS 指令，並分別傳入做 execute。

```
op = (instruction_code >> 26) & 0x3f;
if(op == 0x3F){ exit(0); }
PC = PC + 4 ; /*在執行這行的時候 ＰＣ已經跳到下一個 */
      if(op == 0x00){ R_type(instruction_code);}
      else if(op == 0x02 || op == 0x03){ J_type(instruction_code);}
      else{ I_type(instruction_code); }
}
```

R-type , I-type , J-type——對各種型態的 MIPS 指令分別 parse 出對應的 rs , rt , rd ,imm 並在執行指令前先偵測是否 write 0 error ，同時若是 write 0 error ，還要對特定指令多偵測 number overflow 與 memory overflow , Misalignment。而在 write 0 error 的情況下就不會進入 switch case.

進入 switch case 則執行該指令，並偵測 number overflow 與 memory overflow , Misalignment，如果是需要 halt 的 error，在印完 error_dump 後會直接 exit 結束程式。

對於偵測 error 的部分，我有多寫 TestError & ErrorDetect 的 function。其中 TestError 是將兩個 input 與指令對應的 result 傳入，檢查是否 number overflow 或 memory overflow。ErrorDetect 則是確定已經偵測到 error 了，就做 fprintf 到 error_dump.rpt。

該 PC 指令執行完，跳回 main()，回到 while loop 並印出該 cycle register 的值，接著繼續做下一個指令……，直到遇到要 halt 的 error 結束程式，或者是 data_i 的指令正常執行完，跳出 while loop，fclose 寫入的檔案，整個程式結束。

● Testcase design & elaboration

基本上我 testcase 設計的概念就是跑所有指令跑過一遍，還有對 error detect，sub，sb lb sh lh，做詳細的測試，以下是我的 testcase 簡略介紹。

1. Write 0 error & NOP detect

```
sll $0, $0, 0        # can't detect write $0 error
lw   $0, 1($sp)      # write to $0 error
```

這是測試是否有實作到助教要求的 NOP 指令不會有 error。

2.Number overflow

```
lw     $1,   1($sp)   # $1=0x7FFFFFFF       data at 4
addi $0,    $1, 1     # write to $0 error
addi $1,    $1, 1     # number overflow
```

第二行指令除了 write 0 error 外也要 number overflow ，有些人 write 0 error 會忘了檢查 overflow，也就是測試大家的細心度。

3. D-Memory access

```
sb $2, -1($1)            # store to 1023
lw $4, 1020($0)   # $4 = 0x00000033
sh $2, -4($1)            # store to 1020
lw $5, 1020($0)          # $5 = 0xD4330033
sb $2, -2($1)            # store to 1022
lw $3, 1020($0)          # $03 = 0xD4333333
sh $2, -8($1)            # store to 1016
lw $7, 1016($0)          # $07 = 0xD4330000
sw $2, -8($1)            # store to 1016
lw $6, 1016($0)          # $06 = 0xFFFFD433
sb $2, -10($1)           # store to 1014
lw $9, -12($1)           # $9 = 0x00003300
```

```
sh $2, 1014($0)          # store to 1014
lw $8, -12($1)           # $8 = 0x0000D433
lbu $14, 1023($0)        # $14 = 0x00000033
addi $14, $14, 0x80      # $14 = 0x000000B3
lb $15, 1023($0)         # $15 = 0xFFFFFFB3
```

這是一個很容易錯的地方，store and load 的部分，還有順便檢測邊界（1024 附近）有沒有初始好 0，如果哪個小地方不小心寫錯，這邊就會測出來。這裡有一個小陷阱是我把 imm 故意設計為負的，如果他忘記 imm 要做正負號處理就會出錯，而這塊都是沒有 error 的，也就是單純檢查 store and load 有沒有寫好。

4. Jump and loop

```
Try:  nor $12, $10, $11
      sra $11, $11, 1
      bne $12, $0, Try       # Try = -3
```

這是檢查 bne 與 PC 計算是否正確，同時也檢查 sra 與 nor 有沒有問題，順帶增加 cycle 數目，有些沒寫好的就會當掉。

5.number overflow !!!

```
lw $1, 0($0)             # $1 = 0x80000000          data at 0
lw $20, 0($0)            # $20 = 0x80000000          data at 0
add $0,$20,$1            # write 0 & overflow
add $20,$20,$1           # number overflow
sub $1, $0, $1           # 0+(-0x8000000) = 0x8000000 , no number overflow
sub $25, $8, $1          # D433+(-0x80000000) = 0x8000D433
addi $12, $1, -1         # 0x80000000 - 1 = $12 0x7FFFFFFF, number overflow,
sub $12, $12, $1         # 0x7FFFFFFF +(-0x80000000)=$12 0xffffffff
addi $1, $1, -1          # 0x80000000 +(-1) = $1 0x7FFFFFFF, overflow,
addi $1, $1, 1           # 0x7FFFFFFF+1 = $1 0x80000000, number overflow
```

這可以說是我這筆測資中的精華部分，各種 number overflow 檢測，將 sub,add,addi 全部測過一遍，正負號沒有判斷好，或者是 sub 這個指令沒寫好，error 沒做好偵測，都會錯。主要是在檢測數值在 0x80000000 附近的情況，因為這種情況下最容易寫錯。

6. four errors

```
add $30, $30, $1          # $30 => 0x80000000
```

```
lh $0, 0xffff($30)          # Write $0 error & number overflow & Address overflow &
                            Misalignment error
```

最後是用一個 four error 的錯誤並 halt，整筆測資結束，考驗 simulator 細心度，還有 error detect 的順序。

除了上述講的主要檢測地方之外，我也將所有的指令都跑過一遍，以此判斷是否有正確將每個指令實作。

生產測資最困難的地方在於，要產生出對應的.bin 檔案，無論是用 c 模擬一個完整的 assembler 或者是寫一個小的 MIPS 轉換器，再用別的程式，如 notepad++一個一個打進去產生.bin 檔，這都是很麻煩又很累的一件事。而我，是選擇後者的方法，寫一個小的 MIPS 轉換器，生產測資的同時順便有了正確，經過我手算算出來的答案，也就是確定 snapshot.rpt 的正確性，是比較簡單又實用的方法。但如果是已經確定正確性的 simulator，寫一個完整的 assembler 轉確實也快多了，但一個完整的 assembler 卻要考慮到很多測資合法性的問題，並沒有那麼簡單。

雖然生產測資很累人，但看到完整的測資出來，還有能測爆別人的時候，還是覺得辛苦是值得的，也終於知道助教們要生測資有多辛苦。

Repo Conclusion :

對於我而言，這個 project1 最困難的部分，就是 error 的部分了。該怎麼偵測才能確定偵測到所有 error，還有如何檢測 overflow 才不會有問題？還有該生怎麼樣的測資才能測出別人所有 error？因為一開始考慮不周，導致我 error detect 的部分改了非常多次，（還被自己的測資測爆過，不過也因此找出 bug）。現在的 error detect 基本上已經沒有問題了。還有就是 sb lb 部分，big endian 與 little endian 的不同讓我搞混了很多次，最後是詢問助教才確定不用考慮 little endian 的讀法。

寫完這份 project，不僅學到如何正確實作 MIPS CODE，也終於學會使用工作站，還有寫測資。雖然有時候還是要上網查指令才會用，不過對我而言，這個 project 真的讓我學到了很多東西，也更了解 MIPS 與工作站。