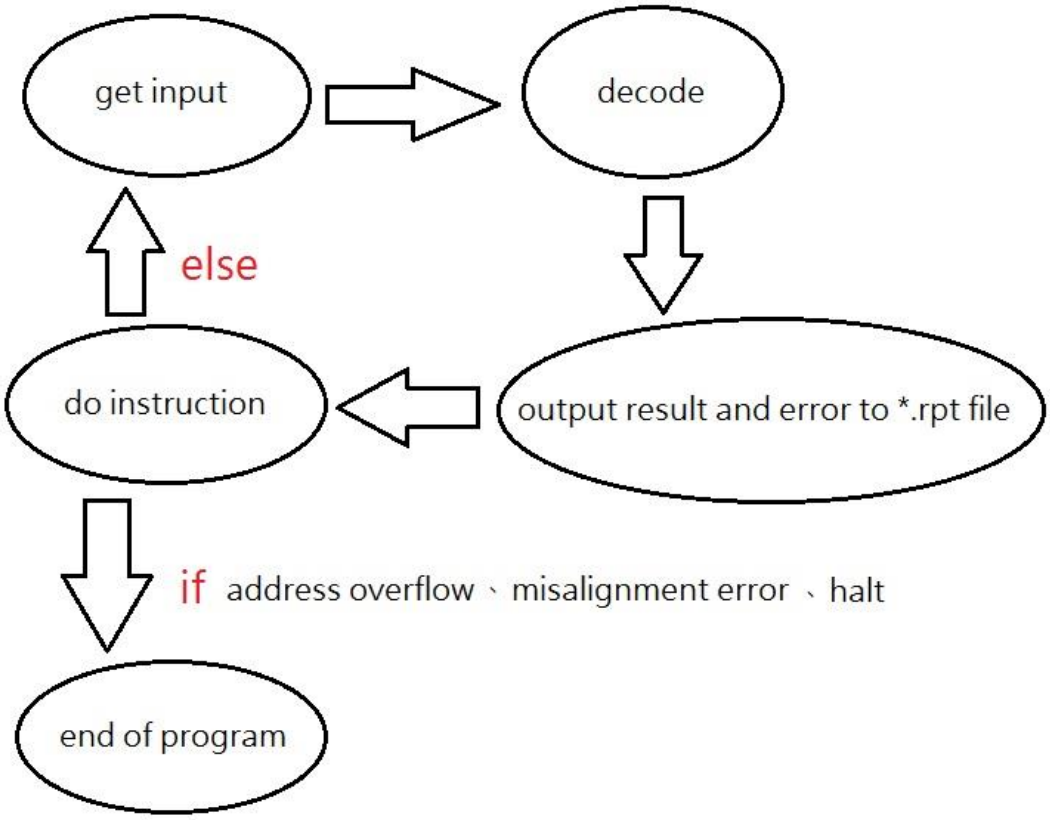
## 1) Project Description

### 1-1) Program Flow chart




### 1-2) Detailed Description

首先說明我是怎麼看 iMemory 和 dMemory 的，我是將 4 個 bytes 看成一行。
也就是說大小為 1024 個 bytes 的兩個記憶體，我將它們看成 256 行。
所以 iMemory 和 dMemory 我宣告成 int[256]，初始化為全部都是 0。

解碼部分：

由於直接讀進來的資料順序並不對，所以要先進行解碼。
以讀 4 個 bytes 為例，正常來說順序是 1234，但讀進來的是 4321。
所以只要將 14 互換，23 互換，就能得到正確的順序了。

讀檔部分：

先用 fopen 將 iimage 和 dimage 打開，
再用 fread 讀檔，我每次都只讀一行，也就是 4 個 bytes。
一開始先讀 iimage 和 dimage 的前兩行，
獲得 PC 和 SP，以及兩個檔案接下來有多少行要讀。
PC 拿到後先除以 4，因為 4 個 bytes 就是一行，
這樣 PC 是多少就代表是第幾行指令，jump 和 branch 會比較容易做。

SP 不用改。至於有多少資料要讀，我是以 iTotal 和 dTotal 來表示。

接著把 iimage 和 dimage 剩下的資料讀進 iMemory 和 dMemory。
dMemory 比較簡單，用迴圈跑 dTotal 次，每次讀一行放到 dMemory，
比較簡單是因為一定是從 dMemory[0]開始放。
iMemory 不同，雖然一樣是用迴圈跑 iTotal 次，但要從 iMemory[PC]開始放。
放完以後就可以開始跑指令了。
之所以要一次放完是因為要處理 jump 和 branch 的情況，
如果一次放一個，遇到要跳到後面的指令，就會出問題。

跑指令部分：
跑指令我是用一個 while(1)迴圈來跑，
結束方式是遇到 halt 或 address overflow 或 misalignment，就 break 終止。
每次進入迴圈時先將前一個 cycle 的結果 output 到 snapshot.rpt。

接著要先分析是哪種指令，我使用的方法是 shift，
例如要讀 opcode，就(unsigned int)iMemory[PC]>>26，即可得到 opcode。
使用相同的方法，就可以獲得指令中任何一段資訊，也可配合&使用。
例如想要 rs，就(iMemory[PC]>>21)&0x1F。
然後就用這個方法將每種指令用 if 判斷式列出來。
指令要做的事情就照 Appendix A 寫的就好了，我只說明某些特別要注意的地方。

首先是 srl 和 sra，其實 sra 只要直接 shift 就好，因為 int 的 shift 就是 sra。
反而是 srl 要特別處理，我是將型別強制轉換成 unsigned int 後再 shift。

再來是有關 PC 改動的部分。
一般來說每個 cycle，PC 只要加 1 就好(我是以行來看)，
但遇到 jump 或 branch 的話，就要依照指令規則來改動 PC 的值，
由於我已將 PC 除以 4，所以給值時也都要記得除以 4。

最後是 load 和 store 的部分。
我是先將 addr = rs + C，
此時 addr/4 即為第幾行，addr%4 即為那行的第幾個 byte。
如此一來就知道要 load 和 store 哪裡了。
例如 addr = 3 的話，就是第零行的第三個 byte(從零開始數)。
addr = 6 的話，就是第一行的第二個 byte(從零開始數)。
另外 store 的時候要記得先把要存的地方清空成 0，
再和要存的數字做 or，才不會存錯數字。

錯誤判斷部分：

判斷錯誤的部分依照 Appendix D 寫就好了，
如果想要寫入$0，又不是 NOP 的話，就是 write $0 error。
如果運算結果不符合預期，就是 number overflow。
如果想要 load 或 store 超過 dMemory 範圍的內容，就是 address overflow。
如果想要 load 或 store 錯位的 dMemory 內容，就是 misalignment error。

比較麻煩的是如果一次出現多種錯誤，要全部印出來。
要注意的是後面兩種錯誤發生時要直接結束程式，
但要全部偵測，所以在發生 address overflow 的時候，
要 break 前還是要偵測 misalignment，所以會變成有點類似洋蔥式的包法。

由於除了 number overflow 以外，另外三個的判斷都不難，
所以我只說明 number overflow 的部分。

首先只有兩種情況會 number overflow，正數加正數、負數加負數。
正數加正數結果小於等於 0，或負數加負數結果大於等於 0，就 overflow。
要注意的是加的時候要先用一個 temp 存結果，否則判斷可能會出問題。
例如 add $1, $1, $1，直接加到$1 的話，
判斷的時候會因為$1 的值已經改變了，而導致判斷錯誤。

比較特別的是 sub 的 number overflow 判斷，為了避免麻煩，
我將 sub 看成加法，即 rd = rs + (-rt)。
因此要先將 rt 做 2's complement，再與 rs 相加。
如此一來就能用跟加法一樣的方式來判斷是否 overflow 了。

如果不把 sub 當加法，會遇到的麻煩就是，
當一個負數和 0x80000000 相減的時候，其實是 overflow 的，
但當成減法的話會偵測不出來，除非針對這個 case 特別做處理。

會發生這個 case 的原因是，MIPS 一開始就是設定把 sub 當成加法。
所以以加法來看，先將 0x80000000 做 2's complement，結果還是 0x80000000，
然後負數和最大的負數相加，結果當然就是 overflow 了。
進一步地說，以 32bit 的整數來說，負數比正數多一個，
而多的那個就是 0x80000000，這會導致它無法轉換成正數，
所以才會做 2's complement 後還是它自己，然後就產生這個特殊 case 了。

## 2) Test case Design

### 2-1)Detail Description of Test case

在看我的 testcase 的內容之前，先提一下初始值的部分，
PC 的初始值是 8，SP 是 0，沒有改動 dMemory 的值(還是全部都是 0)。

```
bne    $0, $1, 0x2
lui    $1, 0x7FFF
jal    0x0
ori    $1, $1, 0xFFFF
add    $0, $1, $1
sll    $2, $1, 0x1
srl    $3, $2, 0x4
sra    $4, $2, 0x4
slti   $5, $2, 0xF000
sw     $1, (1020)$0
lh     $6, (1022)$0
lhu    $7, (1022)$0
lb     $8, (1021)$0
lbu    $9, (1021)$0
addiu $1, $0, 0xABCD
sb     $1, (1022)$0
lw     $1, (1020)$0
add    $1, $0, $0
lui    $1, 0x8000
sub    $2, $1, $1
add    $1, $1, $1
lh     $0, (1023)$0
halt
```

以上是我的 testcase，總共 23 個指令，我會一步一步說明。
首先第一個指令是，如果$0 和$1 不同就跳到第四個指令，但現在不會跳。
第二個指令是將$1 改為 0x7FFF0000，沒什麼特別的。
第三個指令是跳回 PC=0，由於在 PC=8 之前都是 0，所以會空轉兩個 cycle。
這部份是要測試同學是否是從 iMemory[PC]開始放資料的，
如果不是，那就會少兩個 cycle，而導致答案錯誤。
接著會回到第一個指令，這時候才是真正用上它的時候，
由於$1 剛剛被改過了，所以會跳到第四個指令，
如果不這麼做，會變成一個無限迴圈，最後 cycle 超過 500000，變成非法測資。

第四個指令是是將$1 變為 0x7FFFFFFF，沒什麼特別的。
第五個指令是測試 write $0 error 的同時發生 number overflow 的情況。
第六到第八個指令是測試三種 shift 的正確性。
第九個指令是比較 0xFFFFFFFE 是否小於 0xFFFFF000，
這個主要是測試有沒有將 0xFFF0 擴張成 0xFFFFFFF0。
第十到第十四個指令只是單純測試 load 和 store 的正確性。
第十五個指令將$1 變為 0xFFFFABCD。
第十六和十七個指令也是測試 load 和 store 的正確性。
第十八和十九個指令則是將$1 變為 0x80000000。
第二十個指令是測試 sub 的特殊 case(負數減 0x80000000)，會 overflow。
第二十一個指令是測試 rs=rt=rd 的 case，順便測試 number overflow。
第二十二個是測試三個錯誤一起發生時的情況，
分別是 write $0 error、address overflow、misalignment error，
主要是想測 lh 在 1023 的時候也是屬於 address overflow(至少要讀兩個 bytes)，
由於發生了 address overflow 和 misalignment，所以程式會在這裡結束，
第二十三個指令就不會執行了，以上是我的 testcase 的說明。