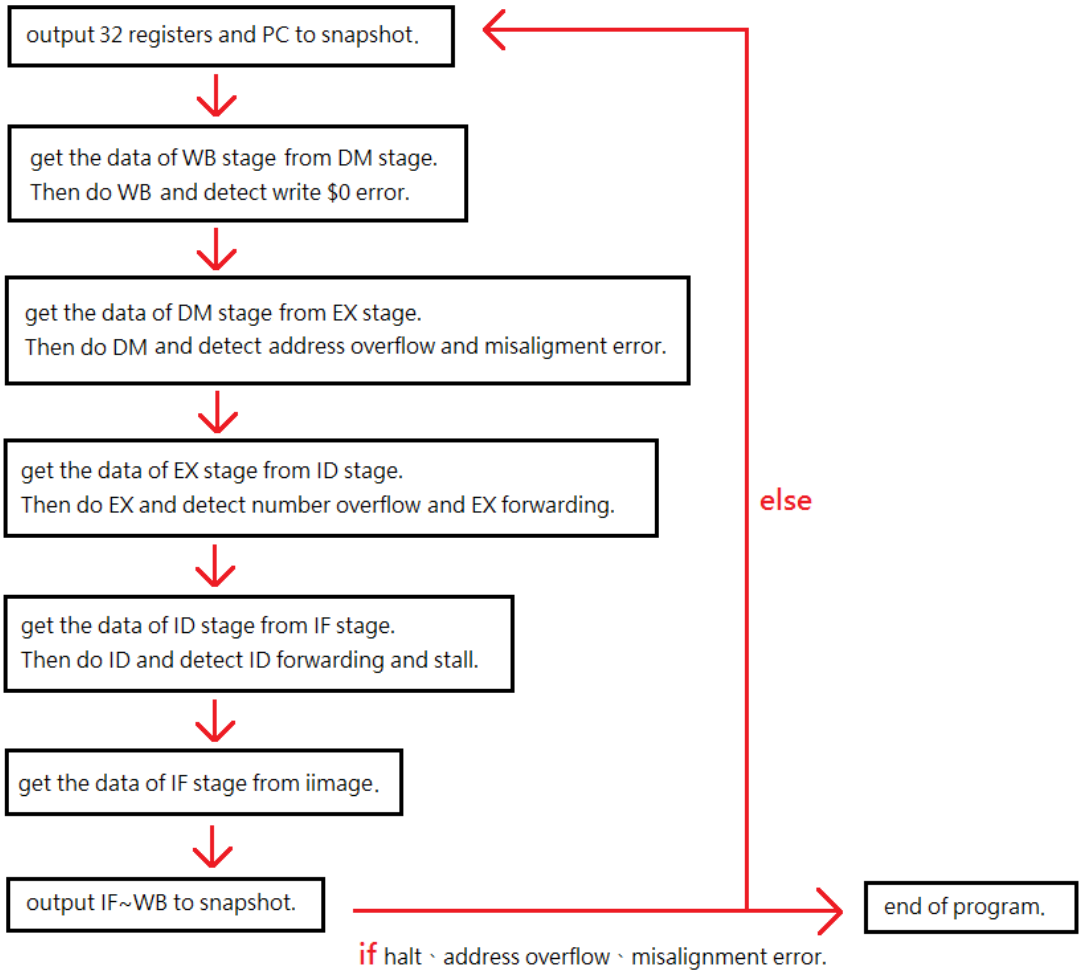
1) Project Description

1-1) Program Flow chart



1-2) Detailed Description

一開始的部分和上次一樣，先把 iimage 和 dimage 都讀進來並 decode。
進到迴圈後，一開始先 output 32 個 register 和 PC 就好。
因為要偵測 stall 和 forwarding，所以等做完 IF~WB 後，才 output IF~WB。
以下會分別說明 IF~WB，每個部分都是先取需要的 data 後才做該做的事。

WB:

先從 DM 取得需要的 data，得知是否要 WB，
以及 WB_Reg，還有代表指令的字串。
如果 WB_Reg 是 0，就 output error 並且不做 WB。
如果不是 0，就直接 WB 即可。 (reg[WB_Reg] = WB_Value)

DM:

先從 EX 取得需要的 data，
得知是否要 WB、是否要 write memory、是否要 read memory，

以及 writeData 和 WB_Reg，還有代表指令的字串。
接下來分為三種情況，read memory、write memory、else。
Read 就是 load 系列的指令，write 就是 save 系列的指令。
Address 就是 EX 的 ALU result。
Load 將 address 對應的值傳給 WB_Value。
Save 則是將 writeData 存入 address 的位置。
Else 的話，只要將 EX 的 ALU result 傳給 WB_Value 就好。
雖然 Else 的部分也有不會 WB 的指令，但是直接給值也沒關係，
因為有 is_WB 這個變數，所以就算亂給值，最後到 WB 的時候也不會有動作。

EX:
一開始分為兩種情況，有 stall 和沒 stall。
有 stall 的話，將所有要取得的 data 都設為 0，並將代表指令的字串設為 NOP。
沒 stall 的話就正常取得 data。
從 ID 取得的 data 和 DM 取 EX 的部份一樣，這裡就不再重複說明。
接下來依據 ALUOP 的值決定要做什麼事，ALUOP 是在 ID 分配的，
分配的原則是運算方式相同的分成同一類。以下是 ALUOP 的值所代表的指令:

```
0 nop / jr / beq / bne / bgtz / j
1 add / addi / lw /lh /lhu / lb / lbu / sw / sh / sb
2 addu / addiu / jal
3 sub
4 and / andi
5 or / ori
6 xor
7 nor / nori
8 nand
9 slt / slti
10 sll / lui
11 srl
12 sra
```

不過在做指令之前，
要先檢查 data1 和 data2 的 register 有沒有和 DM 的 WB_Reg 衝到，
有的話就要 forward。($0 衝到不用 forward)
然後就做指令即可，將結果傳給 ALU result。
要注意 ALUOP 等於 1 和 3 的時候要檢查是否有 number overflow。

ID:
首先如果要 stall 的話，就不要從 IF 那裡取 instruction。沒有的話就正常取值。
接下來要檢查 forward 和 stall，主要分為兩種情況: branch、else。
branch: (jr、beq、bne、bgtz、j、jal)
先檢查要讀取的 register 有沒有撞到 EX 的 WB_Reg，

有的話就 stall，沒有的話就再檢查有沒有撞到 DM 的 WB_Reg，
有的話又分為 load 和非 load 的指令，如果是 load 指令就要 stall，
如果是非 load 指令就 forward。(如果是$0 就不用 stall 和 forward)
由於不會 WB 的指令，我會將 WB_Reg 設為 0，所以不會因為那些指令而 stall 或 forward。

Else:
先檢查要讀取的 register 有沒有撞到 EX 的 WB_Reg，
如果有且 EX 又是 load 指令的話，就要 stall。
如果不符合上述條件，就再檢查有沒有撞到 DM 的 WB_Reg，
有的話再看有沒有衝到 EX 的 WB_Reg，沒有的話就要 stall，有的話就沒事。
可以簡單的用以下兩行來表示:
if(rs && WB_Reg_EX==rs && isRD_EX) stall = 1;
else if(rs && WB_Reg_DM==rs && WB_Reg_EX!=rs) stall = 1;

剩下只要將一些變數決定好就好:
data1、data2、data1_Reg、data2_Reg、isWB、isRD、isWD、WB_Reg、writeData。
我把以上變數值相似的指令放在一起，比較方便。
最後再分配 ALUOP 和代表指令的字串就好了。

IF:
如果 stall 的話，就不從 iimage 取 instruction。沒有的話則正常取值就好。
IF 較為簡單，取得 instrucion 就好。

做完以上這些就可以 output IF~WB，然後進入下一個 cycle 了。
如果有 halt 或會導致結束的錯誤，則結束程式，不進入下一個 cycle。

2) Test case Design

2-1) Detail Description of Test case

我的 PC 起始值是 8，dimage 則是空的 (也就是說都是預設值)。

```
bne   $0, $1, 0x2
lui   $1, 0x7FFF
jal   0x0
jal   0x6
addi  $30, $31, 0x8
jr    $30
jal   0x9
jal   0x10
sw    $30, 0($0)
sw    $31, 4($0)
lw    $31, 0($0)
nop
add   $30, $0, $31
add   $30, $0, $0
add   $30, $30, $30
add   $29, $0, $0
bne   $29, $30, 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
halt
halt
halt
halt
```

基本上和上次的 testcase 差不多，我只多加了幾個指令。
主要是測以下幾點:

1. jal 會寫回$31，如果有人和他衝到要 stall。
2. jr 要讀 rs，如果衝到要 stall 或 forward。
3. 連續兩個 jal，寫回$31 的值不能被蓋掉。
4. 連續兩個 sw，writeData 不能被蓋掉。
5. 非 branch 的 ID 同時和 EX 和 DM 衝到，且 EX 不是 load 指令，不需要 stall。
6. beq 兩個讀取的 register 分別撞到 EX 和 DM，要 stall，不需要 forward。