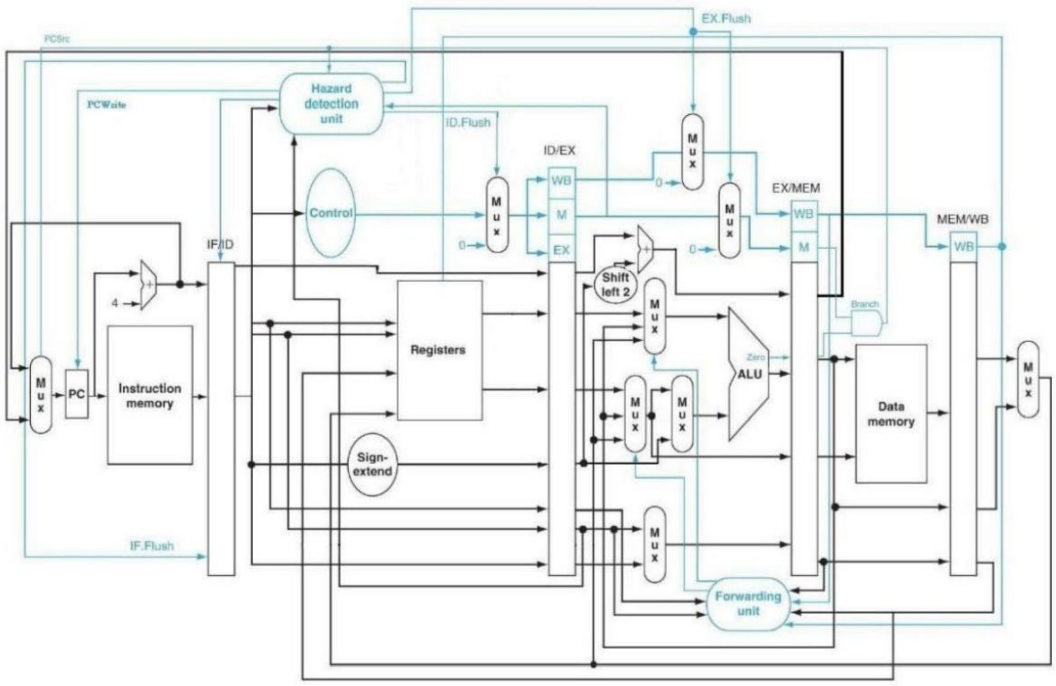
Computer Organization Lab5

Name: 陳宥安

ID: 109550073

Architecture diagrams:



由上圖可見，這次lab較上次相比多了Forwarding和Hazard Detection Unit，然後各個stage的Pipe_Reg多存了一些控制信號，其餘跟lab4都很相似，所以這次的lab我會用lab4的code下去，照著這個電路圖刻、做改良。

Hardware module analysis:

以下針對各個檔案（元件）做簡單說明：

Adder1, Adder2: 就是兩個加法器（a+b）。

MUX_2to1 * 8: 就是個選擇器，但這次有用在關於要不要Flush跟Forwarding的地方，根據特定信號來決定Pipe_Reg存的值。

Program_Counter: 存答案的地方counter。

Instruction_Memory: 讀取並解讀MIPS指令。

Data_Memory: 暫時存一些資料的地方。

Decoder: 判斷MIPS是哪種format、判定要做何種instruction，並decode出一些訊號供續使用。

Reg_File: 靠Decoder傳入判斷instruction是哪種format來決定ALU的輸入值。

Sign_Extend: 如果instruction為i-format要做extension。

ALU: 就是個ALU，加減和邏輯運算，跟上個Lab一樣。

ALU_Ctrl: 控制ALU的控制訊號。
Shift_Left_Two: 就是個shifter，向右shift兩位。
Pipe_Reg: Register for儲存在不同階段的指令。
MUX_3to1 * 2: 三選一的選擇器，以forwardA/forwardB作為控制信號，用來選擇ALU的input訊號。
Forwarding Unit: 處理Data Hazard，根據指令有不同Hazard的類別做不同的Forwarding。
Hazard Detection Unit: 偵測是否有Hazard，並根據種類決定是否要Flush掉已經跑入pipeline的指令及阻止ProgramCounter變動。

Finished part:

a. CO_P5_test_1.txt

```
################################ clk_count = 15####################################
=======================================Register=======================================
r0 =    0, r1 =   16, r2 =  256, r3 =    8, r4 =   16, r5 =    8, r6 =   24, r7 =   26

r8 =    8, r9 =    1, r10=    0, r11=    0, r12=    0, r13=    0, r14=    0, r15=    0

r16=    0, r17=    0, r18=    0, r19=    0, r20=    0, r21=    0, r22=    0, r23=    0

r24=    0, r25=    0, r26=    0, r27=    0, r28=    0, r29=    0, r30=    0, r31=    0

=======================================Memory=========================================
m0 =    0, m1 =   16, m2 =    0, m3 =    0, m4 =    0, m5 =    0, m6 =    0, m7 =    0

m8 =    0, m9 =    0, m10=    0, m11=    0, m12=    0, m13=    0, m14=    0, m15=    0

m16=    0, m17=    0, m18=    0, m19=    0, m20=    0, m21=    0, m22=    0, m23=    0

m24=    0, m25=    0, m26=    0, m27=    0, m28=    0, m29=    0, m30=    0, m31=    0
```

b. CO_P5_test_2.txt

```
################################ clk_count = 64####################################
=======================================Register=======================================
r0 =    0, r1 =    0, r2 =   16, r3 =    6, r4 =    0, r5 =   16, r6 =    0, r7 =    0

r8 =    2, r9 =    0, r10=    0, r11=    0, r12=    0, r13=    0, r14=    0, r15=    0

r16=    0, r17=    0, r18=    0, r19=    0, r20=    0, r21=    0, r22=    0, r23=    0

r24=    0, r25=    0, r26=    0, r27=    0, r28=    0, r29=    0, r30=    0, r31=    0

=======================================Memory=========================================
m0 =    4, m1 =    1, m2 =    0, m3 =    6, m4 =    0, m5 =    0, m6 =    0, m7 =    0

m8 =    0, m9 =    0, m10=    0, m11=    0, m12=    0, m13=    0, m14=    0, m15=    0

m16=    0, m17=    0, m18=    0, m19=    0, m20=    0, m21=    0, m22=    0, m23=    0

m24=    0, m25=    0, m26=    0, m27=    0, m28=    0, m29=    0, m30=    0, m31=    0
```

如圖，輸出正確，MIPS的部份我會在summary作簡單的解釋。

Problems you met and solutions:

接線瘋狂打錯字==，為了要能比較輕易的分變出不同stage的訊號，幫wire命名時有刻意分大小寫。結果沒想到造成我極大的困擾。vivado不會報大小寫錯==，超級討厭==，害我debug超久==，再來是decoder, ALUctrl,

ALU這次要能夠多判斷bne/bgt/bge，但我在decoder的branch信號忘記加到bne/bgt/bge，讓我第2份測資一開始跑錯，還一度以為又是接線問題==，除此之外都還好。

Summary:

我們來分析兩份Machine code的MIPS:

a. CO_P5_test_1.txt

```
I1:    addi    $1,$0,16
I2:    mult    $2,$1,$1
I3:    addi    $3,$0,8
I4:    sw      $1,4($0)
I5:    lw      $4,4($0)
I6:    sub     $5,$4,$3
I7:    add     $6,$3,$1
I8:    addi    $7,$1,10
I9:    and     $8,$7,$3
I10:   slt     $9,$8,$7
```

這份測資跟上次的加分題測資蠻像的，而且有各種Hazard，所以八成是為了測試我們的Code能不能偵測出Hazard，要寫好Forwarding Unit和HazardDetection Unit才能跑出正確答案，而最後跑出來的結果跟PDF上提供的答案一樣，所以成功解決了Hazard。

b. CO_P5_test_2.txt

```
I1:    addi    $2, $0, 3
I2:    sw      $2, 0($0)
I3:    addi    $2, $0, 1
I4:    sw      $2, 4($0)
I5:    sw      $0, 8($0)
I6:    addi    $2, $0, 5
I7:    sw      $2, 12($0)
I8:    addi    $2, $0, 0
I9:    addi    $5, $0, 16
I10:   addi    $8, $0, 2
I11:   beq     $0, $0, 2
I12:   addi    $2, $2, 4
I13:   bge     $2, $5, 6
I14:   lw      $3, 0($2)
I15:   bgt     $3, $8, 1
I16:   beq     $0, $0, -5
I17:   addi    $3, $3, 1
I18:   sw      $2, 0($3)
I19:   beq     $0, $0, -8
```

這個測資加入了beq/bge/bgt，應該是要看我們可不可以偵測出正確的Branch type，由於他這段Code的判斷有點過度繁瑣所以這裡我略過，至少最後我跑出的答案和PDF上提供的答案一樣（一開始

以為怎麼只有r2不一樣嚇死我了，幸好是答案有錯），所以我可以偵測出對的Branch Type。

這是我這學期最後一份作業，而且我是考試後做的，對Hazard處理還算有印象，所以寫的還可以，只是接線過於複雜才讓我寫lab的時間創新高，總結而言，算是為這堂課畫下一個還可以的句點拉，但是我真的不想再碰verilog了……。