**Computer Organization**

**Lab5**

**Architecture diagrams:**

使用題目上的架構

buoae«aod 
لاا 
HAW(G 
OOBO•ruiSuI 
JOIIOOJOP' 

**Hardware module analysis:**

這次的主要目的是加了 Forwarding Unit 和 Hazard Detection Unit來解決data hazard.

此lab解決的hazard有三種

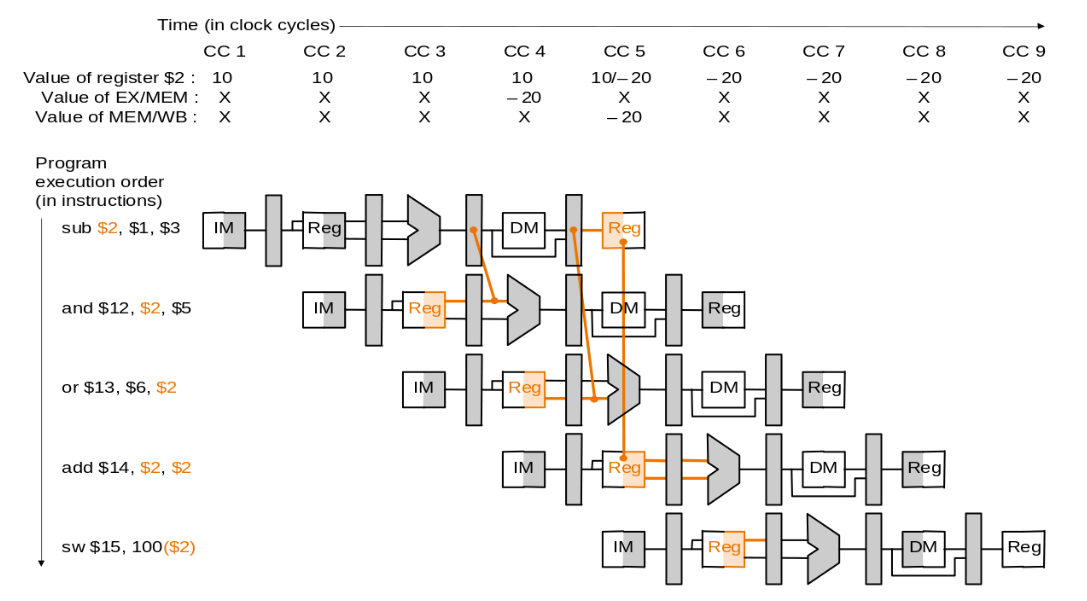
1. R-type data hazard
2. Load-use data hazard
3. Branch hazard

Forwarding Unit 主要進行data的forwarding(duh)

Hazrd Detection Unit則是進行stall 與 flush的動作

以下進行hazard的整理

1. R-type hazard



上圖來自網路大神的筆記

<https://hackmd.io/@8bFA57f7SRG-K7AAT8s62g/ryv1NT3S?type=view>

從圖清楚的看到R-type的兩種hazard

第一種的條件為IF/ID.RegRt(Rs) = EX/MEMRegRd

第二種為條件為IF/ID.RegRt(Rs) = MEM/WB.RegRd

總之原因是寫入的目標與下個指令讀取的來源相衝突

加入forwarding後可以不用bubble解決衝突，code的條件判斷如下

需要Forwarding

if( EXMEM\_WB[1] && EXMEM\_Rd != 0 && EXMEM\_Rd == IDEX\_Rs)

Rs\_forward <= 2'b01;

if( EXMEM\_WB[1] && EXMEM\_Rd != 0 && EXMEM\_Rd == IDEX\_Rt)

Rt\_forward <= 2'b01;

if(MEMWB\_WB[1] && MEMWB\_Rd != 0 && MEMWB\_Rd == IDEX\_Rs)

Rs\_forward <= 2'b10;

if(MEMWB\_WB[1] && MEMWB\_Rd != 0 && MEMWB\_Rd == IDEX\_Rt)

Rt\_forward <= 2'b10;

不需要Forwarding

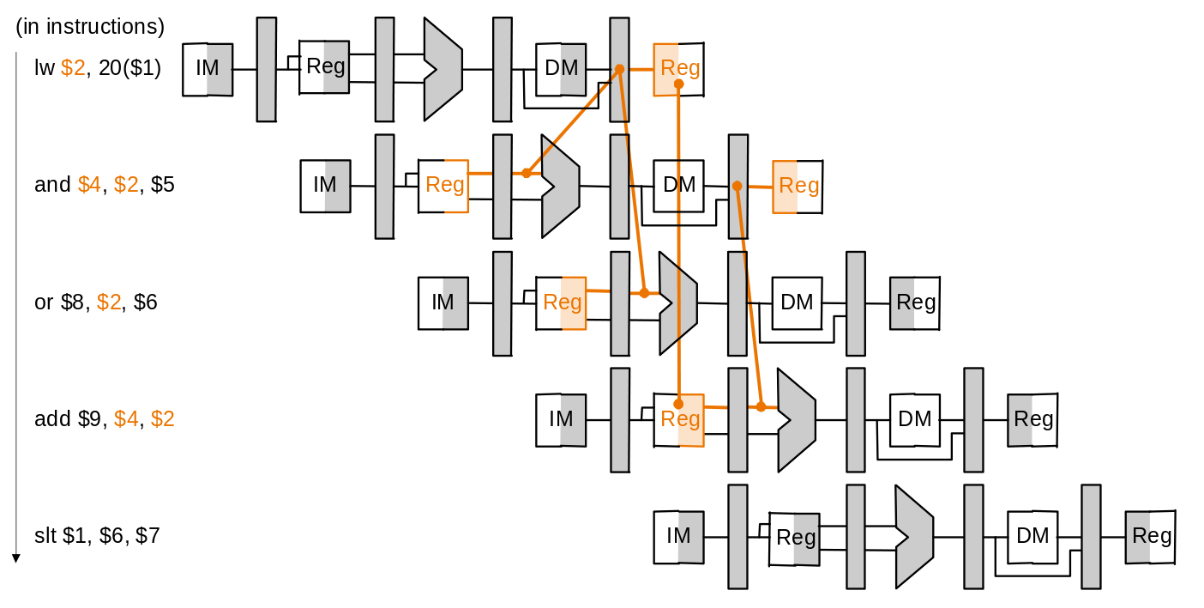
if(MEMWB\_Rd != IDEX\_Rs && EXMEM\_Rd != IDEX\_Rs)

Rs\_forward <= 2'b00;

if(MEMWB\_Rd != IDEX\_Rt && EXMEM\_Rd != IDEX\_Rt)

Rt\_forward <= 2'b00;

1. Load-use hazard



上圖可以看到就算使用forwarding也無法解決data hazard

所以load-use hazard無法避免的需要stall一個cycle

Stall這動作是由hazard detection unit進行的，有兩條控制線分別連到

Program Counter 和 IF\ID.Reg，在load-use發生時控制線會停止兩個reg的寫入達到stall的作用，以下為load-use發生的條件

if( (IFID\_RsRt[9:5] == IDEX\_Rt || IFID\_RsRt[4:0] == IDEX\_Rt) &&

IDEX\_Rt != 0 &&

M[1]

)

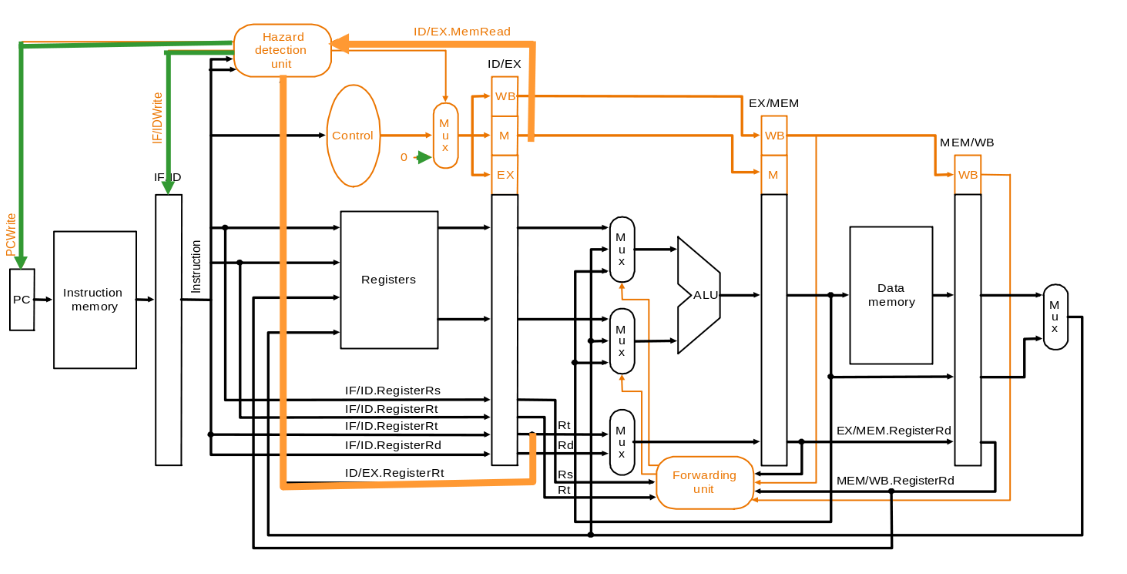
IDFlush <= 0; //load use 不需要flush

EXFlush <= 0;

IFFlush <= 0;

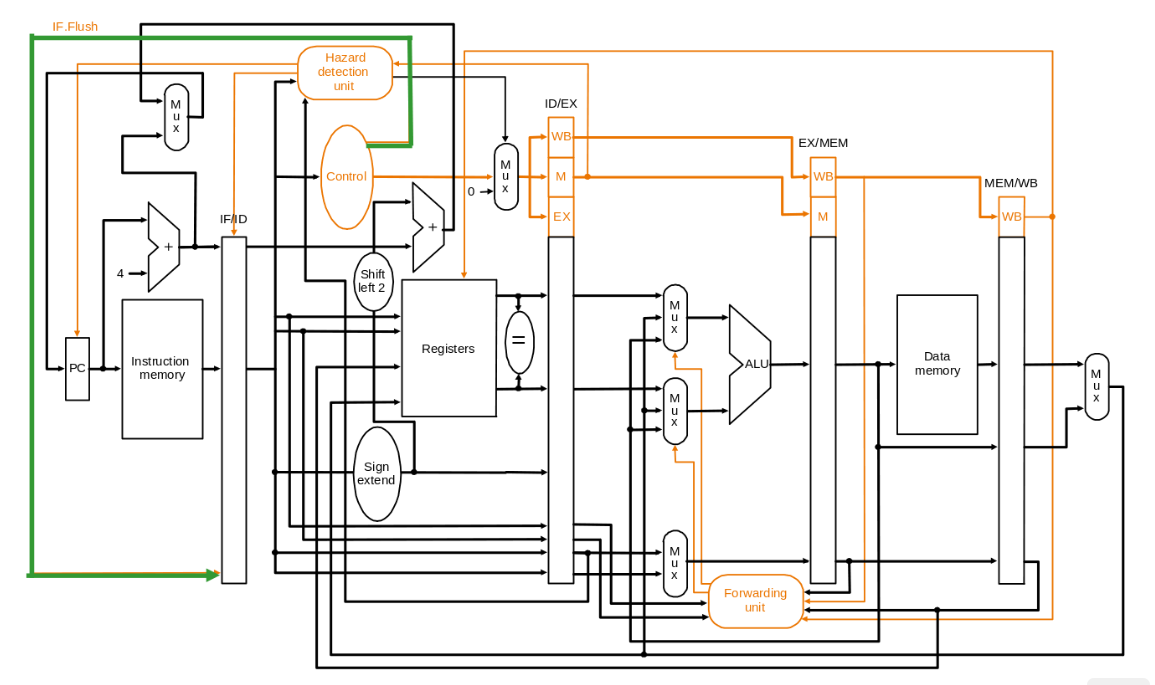
PCWrite <= 0; // 暫停PC寫入

IFIDWrite <=0;



1. Branch Hazard

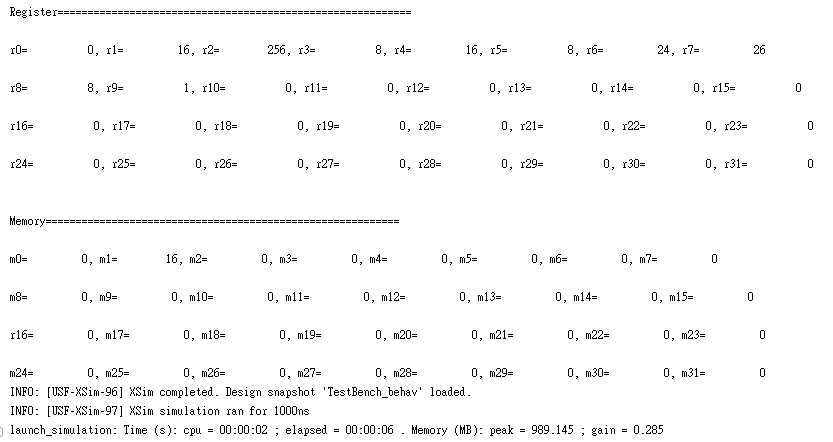
因為要等比較結果才能知道要不要branch所以真到要branch時會讀到不需要的指令，hazrd detection unit可以flush調不需要的指令



再次附上網路大神的精美示意圖(不過他沒有flush後面的東西)

**Finished part:**

CO\_P5\_test\_1.txt 測試結果



自己測試beq的結果

我用lab4的測資去改的

執行的指令如下

00100000000000010000000000000101 (addi, $1 = 5)

00100000000000100000000000000101 (addi, $2 = 5)

00010000001000100000000000000001 (beq, $1 =$2?)

10001100000010100000000000000111

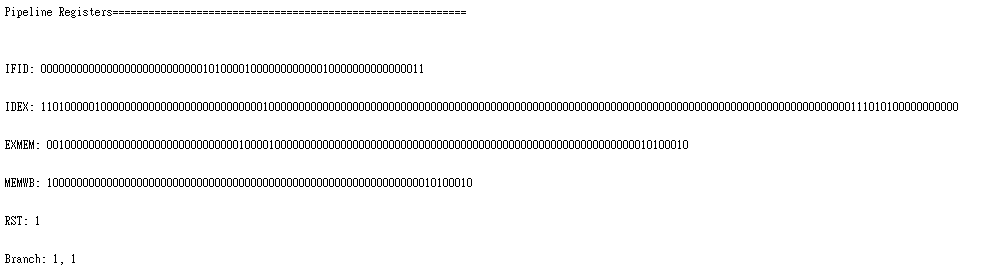
00100000000000010000000000000011

00100000000000100000000000000100

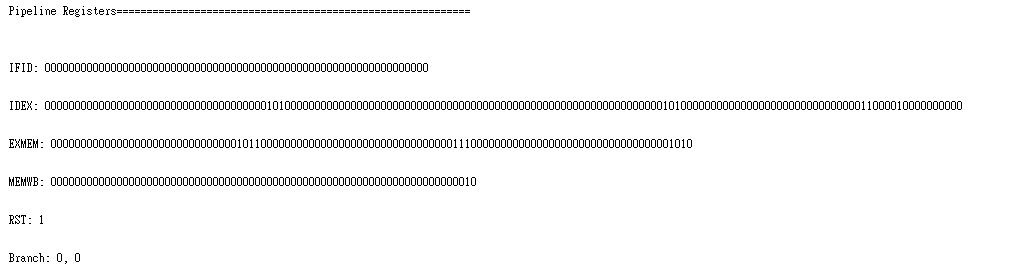
00100000000000100000000000000101

00100000000000100000000000001111

即將branch



執行Flush 可以看到IFID變成0



**Problems you met and solutions:**

1. **Forwarding Hazard條件問題**

原本以剩下條件用else就好，但其實這樣會造成問題，所以我還是把不會發生hazard的條件列出來，詳細條件如前面hardware analyses說的

1. **Hazard Detection Unit 條件問題**

((IFID\_RsRt[9:5] == IDEX\_Rt || IFID\_RsRt[4:0] == IDEX\_Rt) && IDEX\_Rt != 0 && M[1])前面那段要刮起來不然會判斷錯誤

1. **Load-use hazard stall**

因為需要pipeline register可以暫停寫入，所以稍微修改了pipline register，加了條write enable控制線

1. **Forwarding Unit造成Don’t Care Term**

在用Lab4的測資測試時，發現addi指令會造成Don’t care term，結果會不正確，原本以為是Forwarding Unit的條件判斷錯誤造成的，結果是眼殘接錯WB register，往前抓到前一個step的(不知為改正前Lab5結果還是對的@@)

1. **Beq指令結果不對，flush之後reg呈現Don’t care**

原本以為是Flush的問題，後來將相關線路的數值印出來才發現ALUSrc MUX的第二個source (branch pc)接錯線，接上之後beq正常運行但flush會錯誤，後來發現rst\_i的值平常是1，所以要flush的話pipreg的rst值是

(rst\_i & !IFFlush)

而不是

(rst\_i | IFFlush)

**Summary:**

好險有網路大神的筆記，不然可能寫不太出來，雖然期末考爆掉了，不過藉由這次的lab終於把課堂上所教的hazard都搞清楚了，雖然寫lab很花時間但很好玩，不過還是希望明年不要再寫一次rrr