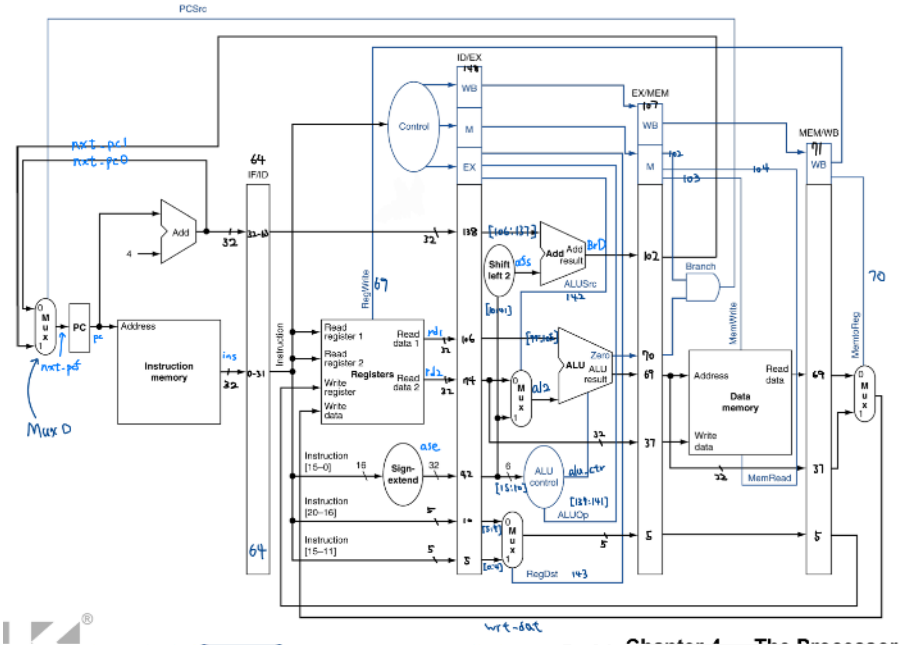
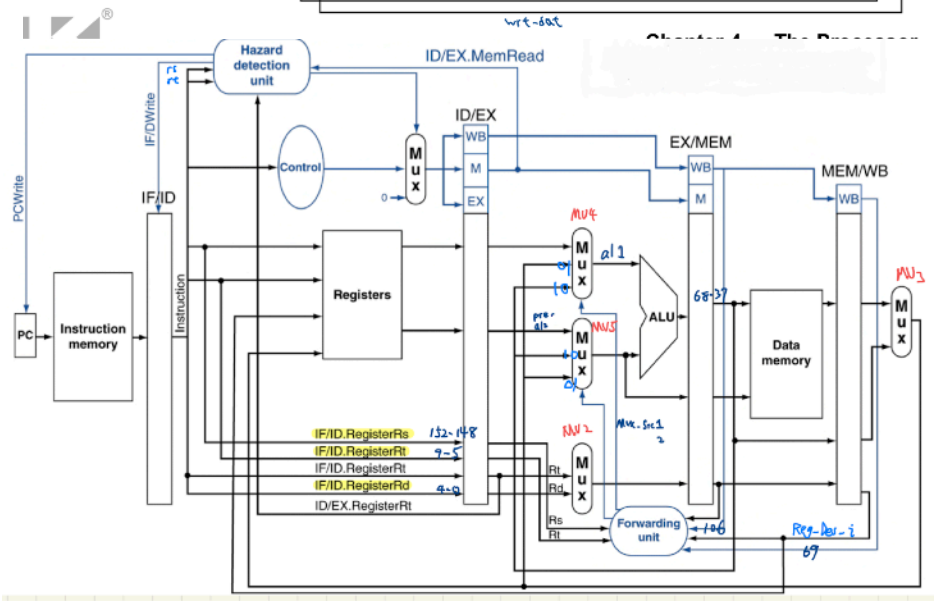
Computer Organization
Lab5 Advanced Pipelined CPU

Architecture diagrams





這次的 Pipelined CPU 我採用與上次相同，如左圖的電路圖，每個 Stage 中間的黑色數字為資料往下傳需要的 bits 數量，深藍色為它所在那個 Stage 的哪一個或那一區的 bits，而控制元件 Control 跟 ALU control 採用與前一次大致相同。

另外，再增加兩個 module: Hazard_Detection_Unit 以及 Forwarding_Unit。向左圖一樣增加這兩個 module ，調整 pipe line reg 的 size，增加 MUX。

還有將 Pipe Reg 增加 Write 訊號，控制可不可以寫。

Hardware Module Analysis

這次有增加幾個 Branch ，我將一一對應到不同的 ALU_OP 訊號給 ALU_Control ，讓個別控制 ALU 的訊號，之後再將 ALU 的輸出訊號抓出來，做布林運算算是不是要 Branch ，如果是 Branch 的話，就交由 Hazard_Detection_Unit 將 Pipe Reg flush 掉，以及把 ID/EX 的控制訊號設為 0 。

- beq : 與第一張電路圖相同
- bne : 相減不為 0 (zero 訊號為 0)
- bge : a – b 後 sign bit result 為 0
- bgt : a – b 後 sign bit result 為 0 且非 0 (zero 訊號不為 1)

```
// 1 -> next is branch
assign PCSrc =
    ( otEXMEM[102] &  otEXMEM[69] ) |
    ( otEXMEM[102] & ~otEXMEM[69] & ~otEXMEM[141] & otEXMEM[140] &  otEXMEM[139] ) | // bne 011
    ( otEXMEM[102] & ~toEXMEM[ 68] & ~otEXMEM[69] &  otEXMEM[141] & otEXMEM[140] &  otEXMEM[139] ) | // bge 111
    ( otEXMEM[102] & ~toEXMEM[68] & ~otEXMEM[ 69] & otEXMEM[141] & ~otEXMEM[140] & ~otEXMEM[139]);  // bgt 100
```

如上圖，若 PCSrc 為 1 的話，即為要 branch ，而這個訊號也傳給 HDU Hazard_Detection_Unit ，做 Branch 的 stall operation。

Problem Met & Solution

這次寫沒有遇到什麼問題，可能是因為期末考已經考完，準備期末考這部分都已經搞懂，且在寫的時候運氣不錯，測試時沒有遇到太多問題，所以還算順利。

Result

Testcase 1

```
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
r16=   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
```

這次的唯一一筆測試資料結果如上圖，可以看到與預期相符。

Testcase 2

```
Register==============================================================
 r0=   0,  r1=  16,  r2=  20,  r3=   8,  r4=  16,  r5=   8,  r6=  24,  r7=  26
 r8=   8,  r9= 100, 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
r16=   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
```

另外，我將 Lab4 測試資料 2 來用，看看同樣具有 Data Hazard 的資料會不會對，結果與答案相符。

Testcase 3

```
Register==============================================================
 r0=   0,  r1=   3,  r2=   4,  r3=   1,  r4=   6,  r5=   2,  r6=   7,  r7=   1
 r8=   1,  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=   0,  m1=   3,  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
```

此外，我將 Lab4 具有 branch 的測試資料 1 也拿來做測試，結果與答案相符，所以 branch 部分應該正確。

Summary

這次的 Lab 應該是這學期計組的最後一次 Verilog 作業，相當初學期剛開始可能連寫加法器都有問題，現在已經可以寫出基本的 CPU ，真的覺得很神奇，但也代表自己有學到東西。