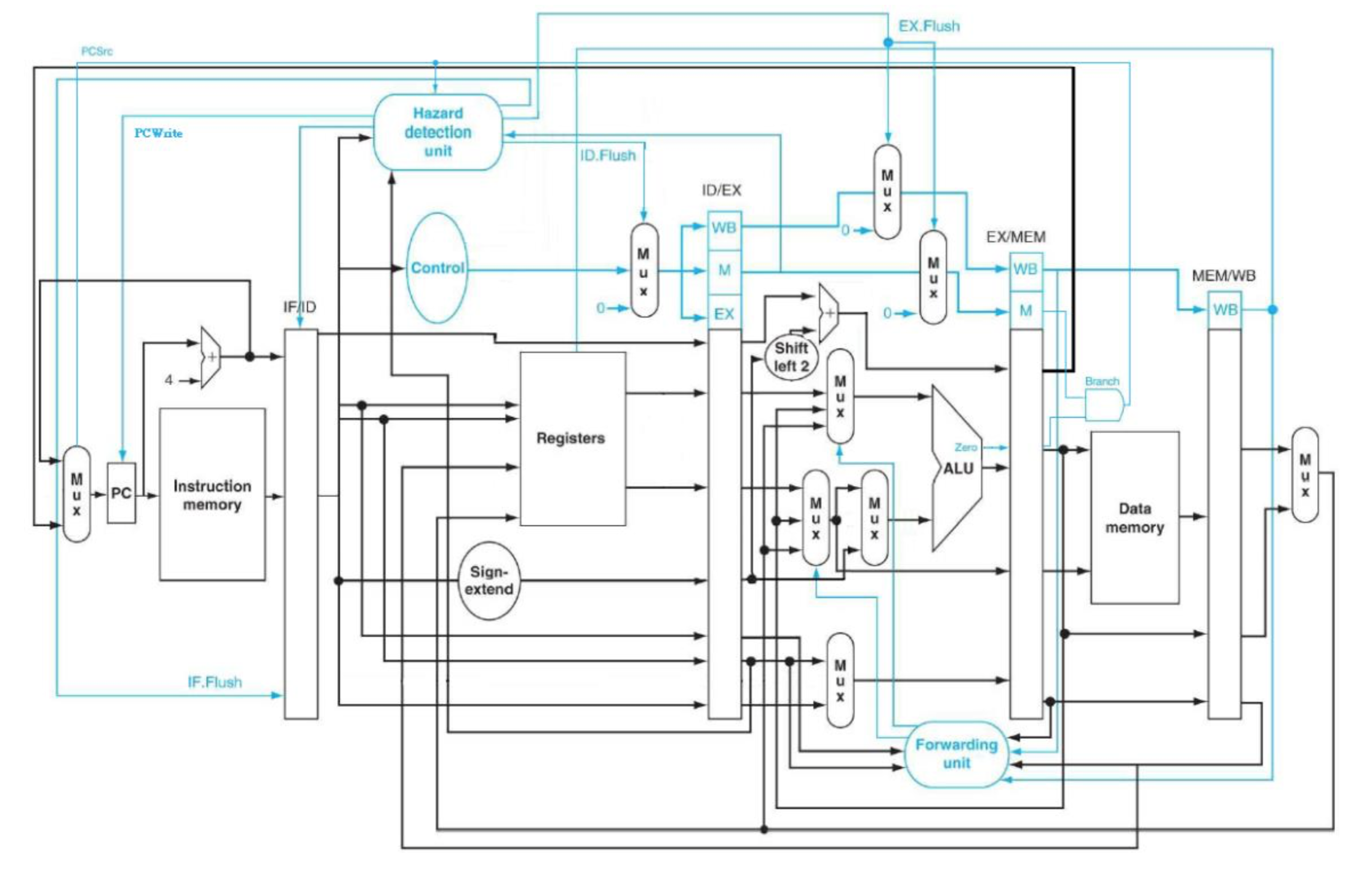
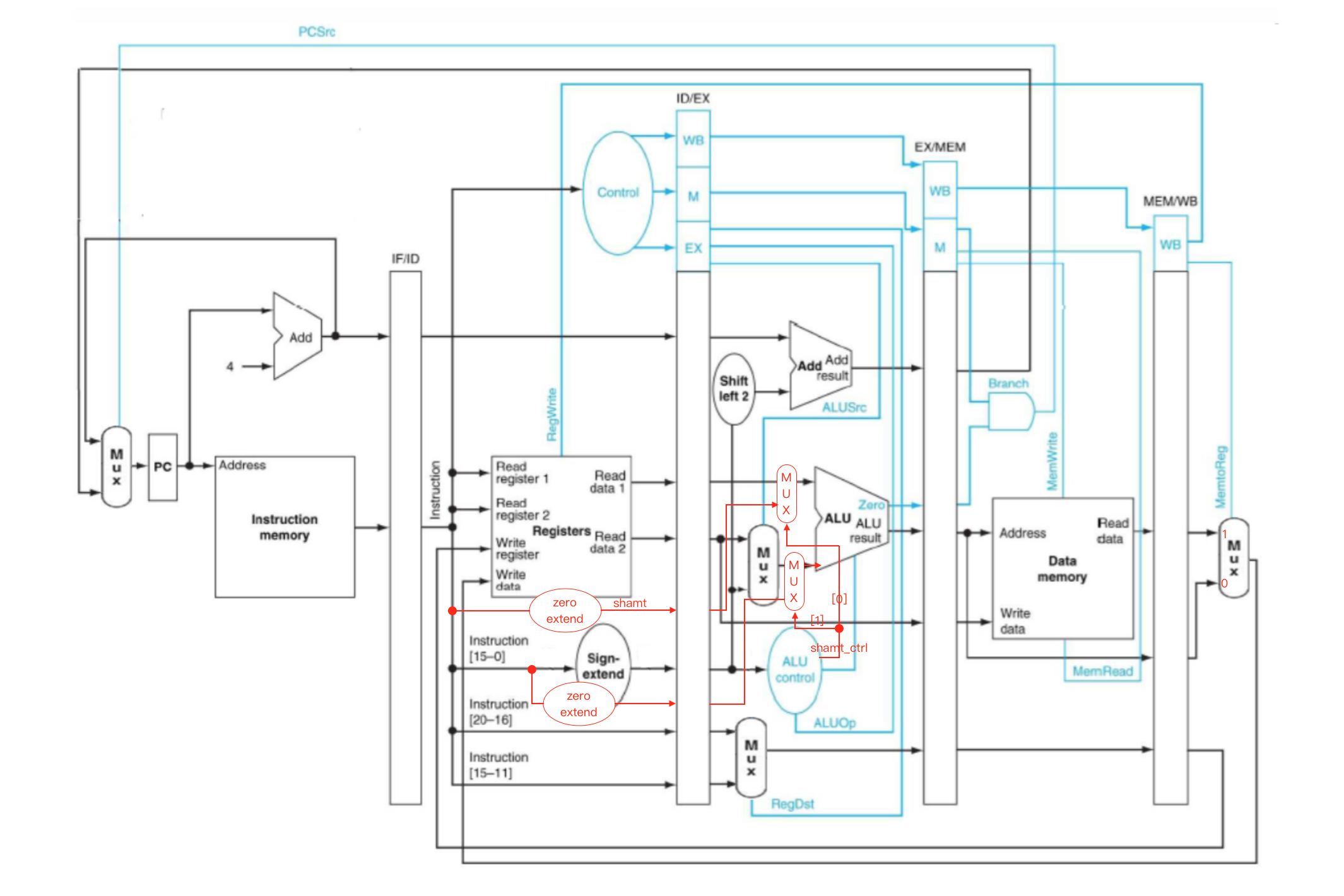
**Computer Organization**

0316055許庭嫣 / 0316313張逸群

**Architecture diagram:**

****

**Detailed description of the implementation:**

1. Pipeline Register：

將原本的CPU分割成五個stage，這五個stage的線都必須要用不同的名字命名(以開頭辨別是哪個stage)，如此不同stage重複出現的wire (control wires signal、address written back等)才不會搞混。每次做完一個stage的事後會將資料存在pipeline register，這裡要仔細算好全部儲存資料的bit總數；每經過一個clock cycle，下一個stage會從pipeline register拿出上一個stage的結果繼續做下一步。

將所有的要儲存在pipeline register的wire連至他的input，並將總bit數傳進他的size。每過一個clock cycle才改變他的值。

1. Forwarding Unit：

當要運算完成要寫入的data在被存進Register更新前下個指令就要拿它來使用時，必須作出修正才能使得答案符合所需。因為在EX stage做運算前改變好ALU的source data即可，因此需比對MEM和WB stage的write address是否與兩個source address一樣。

如果MEM\_RegWrite 且 MEM\_write\_address!=0 且MEM\_write\_address和EX\_RS\_ address相等，則讓source data為MEM stage的alu result；WB stage的判斷方式也相同，惟須注意的是必須先判斷MEM stage再判斷WB stage的資料，因為所需的值應該與較近的指令運算結果相同。而RT和RS異同。

1. Hazard Detection：

當做load的指令在EX stage被發現（MemRead=1）且write address和下個指令要使用的其中一個source address相同時，必須在兩個指令中安插nop，讓下個指令延後一個clock cycle做事。此時會洗掉ID stage的control signal全部設置為0，並不改變IF/ID register的值，而PCWrite=0使得PC的位址不改變。

當做branch的指令在MEM stage被確定（PCSrc=1）時，必須把預先以為會做的但結果不會做的事清掉。因此會將IF、ID、EX的control signal都洗掉設置為0，不改變Register和Memory的資料；而PC的位址將因做branch而變成label所在的位址，執行那個位址的指令。

1. CO\_P4\_test\_3：

001000 00000 00001 00000 00000000001 / addi $1, $0, 1

001000 00000 01111 11111 11111111111 / addi $15, $0, -1

001000 00000 01000 00000 00000001010 / addi $8, $0, 10

001000 00000 01001 00000 00000000100 / addi $9, $0, 4

000000 01000 01001 01100 00000011000 / mul $12, $8, $9

001000 00000 01110 00000 00000001000 / addi $14, $0, 8

000000 01100 01110 01000 00000100010 / sub $8, $12, $14

001000 00000 01001 00000 00000000000 / addi $9, $0, 0

100011 01000 01010 00000 00000000100 / lw $10, 4($8)

100011 01000 01011 00000 00000000000 / lw $11, 0($8)

000000 01010 01011 00010 00000101010 / slt $2, $10, $11

000100 00010 00001 00000 00000000011 / beq $2, $1, 3

101011 01000 01010 00000 00000000000 / sw $10, 0($8)

101011 01000 01011 00000 00000000100 / sw $11, 4($8)

001000 00000 01001 00000 00000000001 / addi $9, $0, 1

001000 00000 01101 00000 00000000100 / addi $13, $0, 4

000000 01000 01101 01000 00000100010 / sub $8, $8, $13

000000 00000 01000 00010 00000101010 / slt $2, $0, $8

000100 00010 00001 11111 11111110101 / beq $2, $1, -11

000100 01000 00000 11111 11111110100 / beq $8, $0, -12

000100 01001 00000 00000 00000000001 / beq $9, $0, 1

000100 01001 01001 11111 11111101111 / beq $9, $9, -17

000000 00000 00000 00000 00000000000

**Problems encountered and solutions:**

因為是做pipeline，一個clock會對應很多個instruction，看模擬器debug會很困難。在經過漫長的debug後，才發現可能只是wire的名字寫錯或是testbench給的counter不夠大。其他在理解過pipeline的實作方式及forwarding和hazard detection的寫法後，就沒有太大的問題了。

Microsoft Word是很爛的東西，打到一半會跳通知說有些東西沒有顯示資料有存在檔案中，頁面顯示的東西就全部空白看不到。明明有儲存檔案，重新開啟後打好的報告不見了一半。抵制Microsoft Word，以後盡量不使用，並向外推廣其他文字編輯器。

**Lesson learnt (if any):**

更能理解pipeline的運作方式及實作方法，還有做forwarding跟hazard detection的判斷依據及因此需要改變的地方。另外，雖然大多的code是使用前幾次的結果，但包括testbench等module可能都會有需要修改的時候。