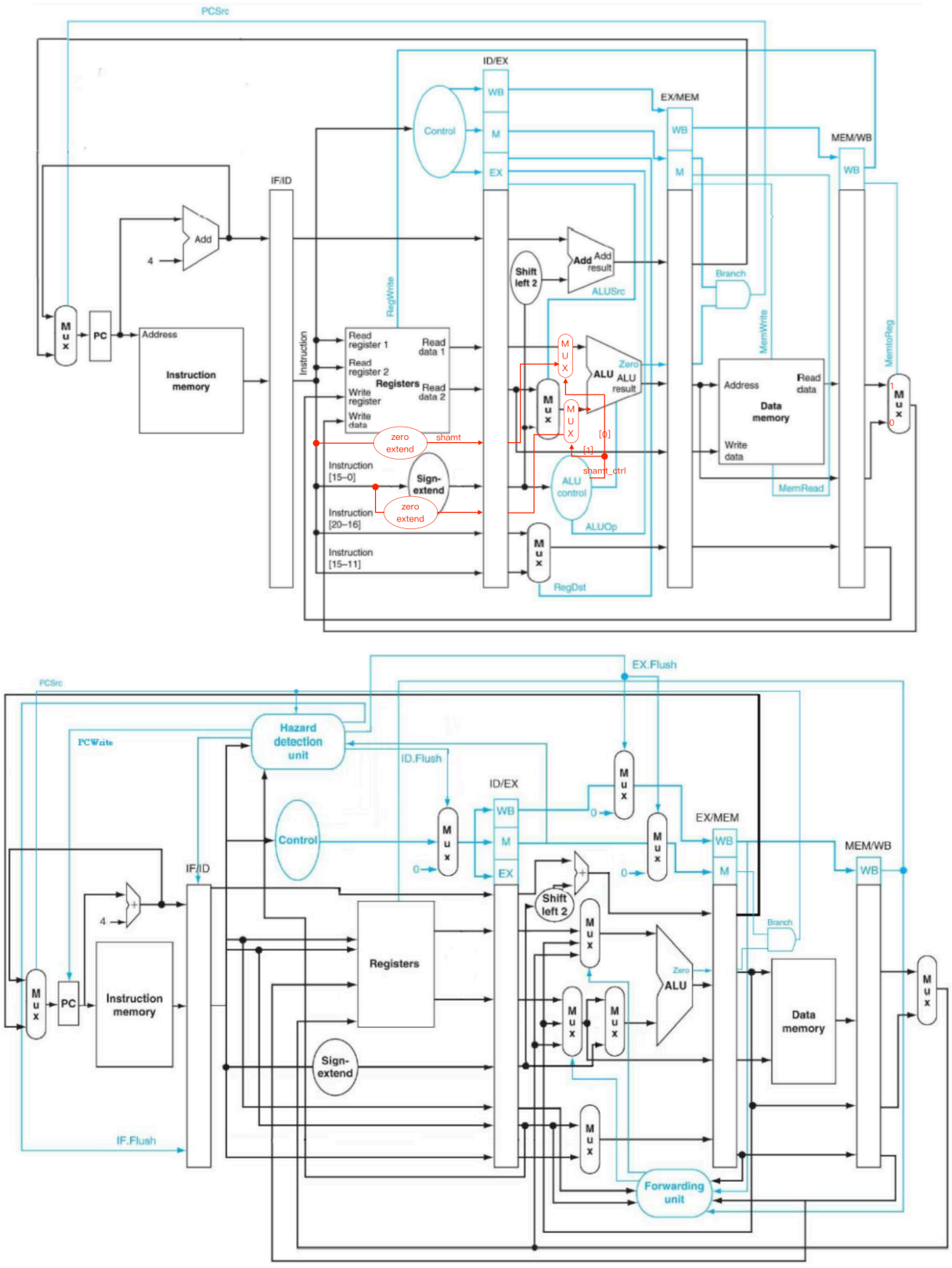
Computer Organization

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。

2. 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。

3. 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。

4. 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。