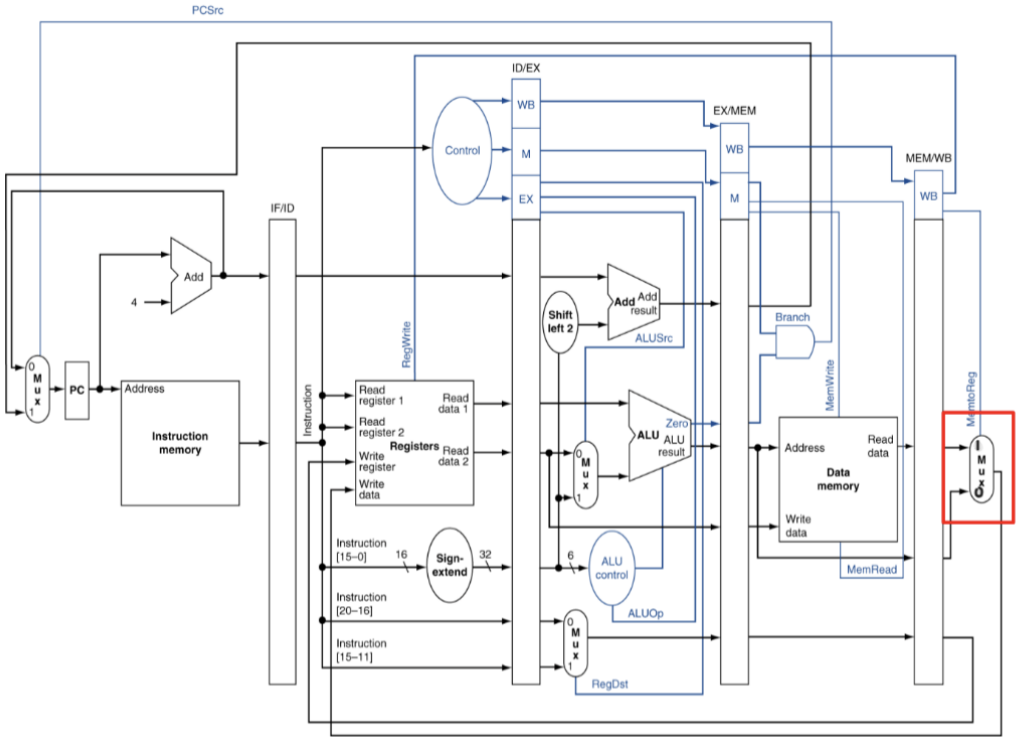
Computer Organization Lab4

ID: 110550108 Name: 施柏江

Architecture diagrams:



Hardware module analysis:

Lab4 與 Lab3 相比，多了 mult 和 xor 這兩個指令，並移除了 jump、jal、jr 指令，因此在 ALU.v 和 ALU_Ctrl.v 中添加了另外的處理，並在 decoder.v 中將多餘的 control signal 刪除。此次的 CPU 多了 pipeline 的功能，因此在設計上多了 pipe_register 儲存各 stage 所需要的 data。

Adder: 負責處理 immediate 指令及計算記憶體位置。

ALU_Ctrl: 根據 opcode 和 ALU_op 來決定要讓 ALU 執行何種運算。

ALU: 負責處理邏輯與加減乘運算。

Data_Memory: 負責處理記憶體的讀寫。

Decoder: 為電路中最重要的核心，負責處理各種 control signal。

Instruction_Memory: 將 address 轉成對應的 instruction。

ProgramCounter: 指向要執行指令的 address。

Reg_File: 輸出此 instruction 需要用到的 register 的資料。

Shift_Left_Two_32: 將輸入的值左移 2 位。

Sign_Extend: 藉由把 sign bit 延伸到第 17~32 位，將 16bit 的數值延伸到 32bit。

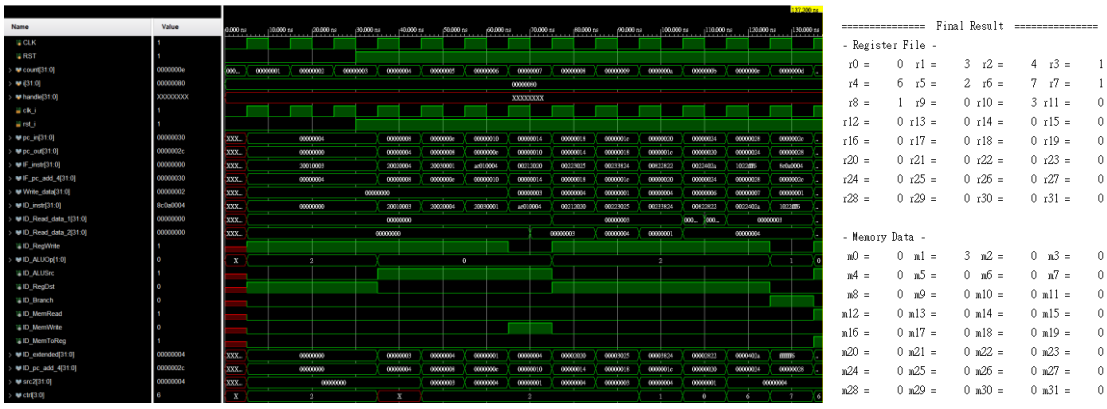
MUX_2to1: 上圖有 4 個。一個判斷是否需要 branch 到其他 address；一個判斷是 r-format 還是 i-format；一個判斷 destination 是哪個 register；一個判斷寫入 register 的 data 為何。

Pipe_Reg: 負責儲存各個 stage 的 data，為 pipelined CPU 最重要的一部分。

Pipelined_CPU: 負責將所有 module 統整再一起，組合成一個 pipelined CPU。

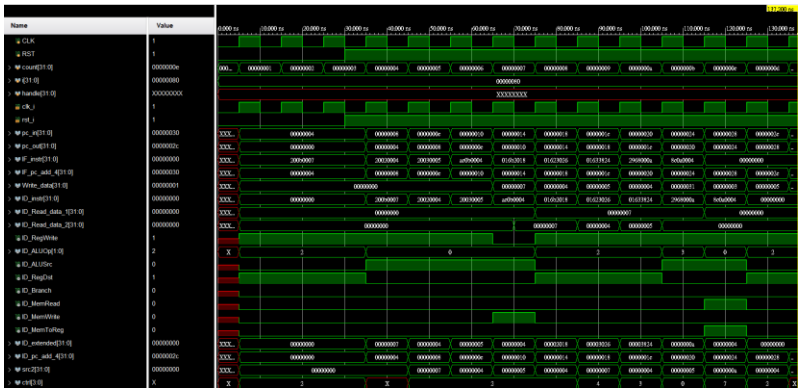
Simulation results:

Test1



各個 register 的值與 MIPS code 的結果一致，而 m1 是透過 sw 指令寫入的

Test2



```
=============== Final Result ===============
- Register File -
r0 =    0  r1 =    0  r2 =    4  r3 =    5
r4 =   49  r5 =    0  r6 =    3  r7 =    5
r8 =    1  r9 =    0 r10 =    7 r11 =    7
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 Data -
m0 =    0  m1 =    7  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
```

Test3(Bonus)

I1 和 I2 因為都使用到了$1，需間隔 2 operations，故將不會受到$1 影響的 I3 和 I10 提前到 I1 和 I2 之間。I5 和 I6 因為都使用到了$4，需間隔 2 operations，故從 I7、I8、I9 之間選 2 個提前到 I5 和 I6 之間，且因 I8 和 I9 都使用到了$7，需間隔 2 operations，所以只能選擇將 I7 和 I9 提前到 I5 和 I6 之間，並將 I8 提前到 I5 之前。

```
I1:    addi   $1, $0, 16
I3:    addi   $3, $0, 8
I10:   addi   $9, $0, 100
I2:    addi   $2, $1, 4
I4:    sw     $1, 4($0)
I8:    addi   $7, $1, 10
I5:    lw     $4, 4($0)
I7:    add    $6, $3, $1
I9:    and    $8, $7, $3
I6:    sub    $5, $4, $3
```

```
1   00100000000000010000000000010000
2   00100000000000110000000000001000
3   00100000000010010000000001100100
4   00100000001000100000000000000100
5   10101100000000010000000000000100
6   00100000001001110000000000001010
7   10001100000001000000000000000100
8   00000000011000010011000000100000
9   00000000111000110100000000100100
10  00000000100000110010100000100010
```

```
=============== Final Result ===============
- Register File -
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 Data -
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
```

Problems you met and solutions:

一開始我並不明白助教在 spec 裡電路圖的註解是什麼意思，就直接照著電路圖設計電路，結果 register 和 memory 跑出來的值錯得離譜(全為 0)，於是我又重新仔細理解紅框框裡的 MUX 的每個 input，發現當 MemToReg 的值為 1 時，所需要的 data 為 0 的 data，而非 1 的 data，於是我把 0 和 1 的 data 交換，終於完成了這次的 lab。

Summary:

這次的 lab 雖然沒有設計到 pipeline register 最厲害的部分(forwarding)，需要將指令重排以避免 data hazard，但也讓我對如何設計 pipeline 有了更深刻的印象。CPU 的功能越來越完整，但也越來越難 debug 了。