# Computer Organization
# Lab4: Pipelined CPU

110550093 蔡師睿

## 1 Architecture Diagrams

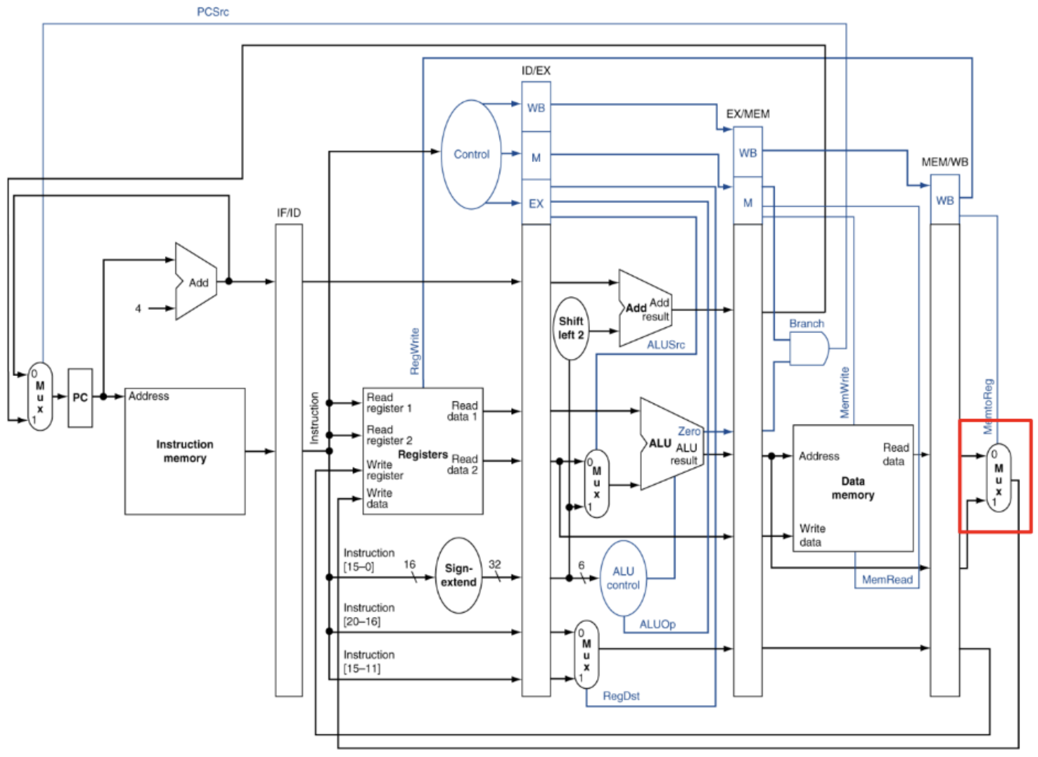


Figure 1: Pipelined CPU

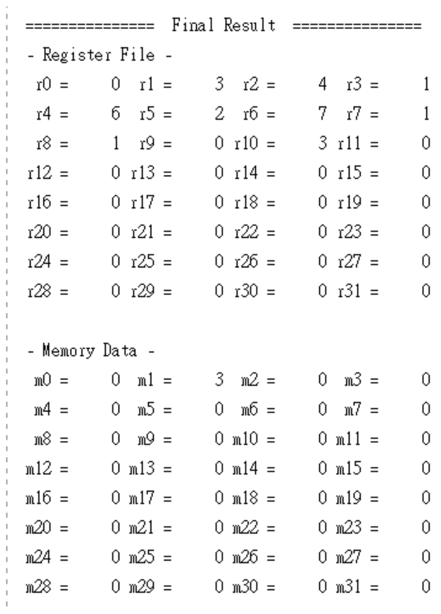
## 2 Hardware Module Analysis

與 Lab3 的不同之處在於拿掉了 jump, jr, jal 等指令，但使用 pipeline 的方式完成 CPU，使 CPU 可一次處理多個指令，且不影響結果，在提升 CPU 效能的同時，保證了其準確性。

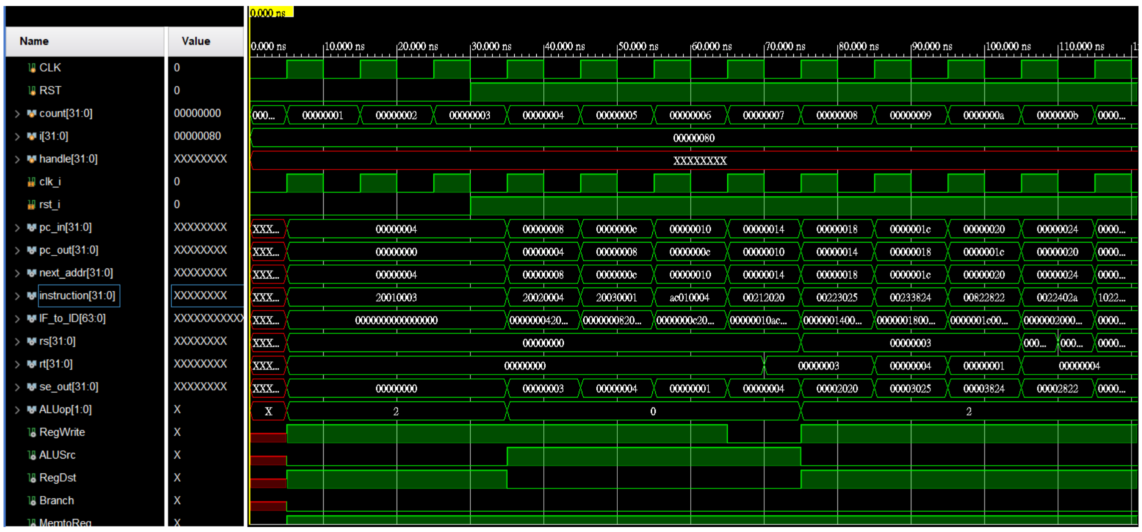
Pipeline 的實作方式是將 CPU 先分爲 5 個 stages，並在各 stage 中間新增 pipeline register，暫存要傳到下個階段的值，因此 CPU 便能同時處理數個指令。(參考 Figure 1)

## 3 Simulation Results

### Test 1



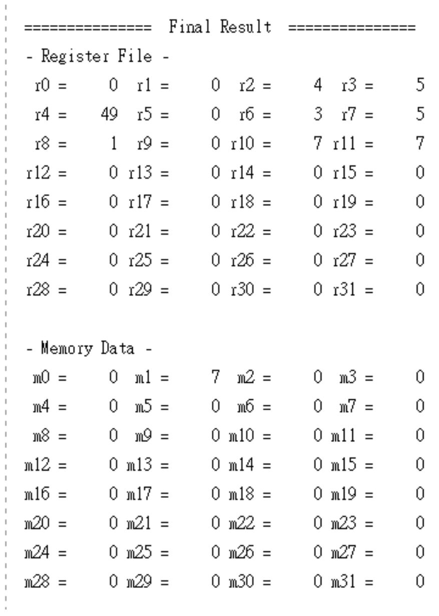
```
===============  Final Result  ===============
- Register File -
 r0 =     0  r1 =     3  r2 =     4  r3 =     1
 r4 =     6  r5 =     2  r6 =     7  r7 =     1
 r8 =     1  r9 =     0 r10 =     3 r11 =     0
r12 =     0 r13 =     0 r14 =     0 r15 =     0
r16 =     0 r17 =     0 r18 =     0 r19 =     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 =     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
```



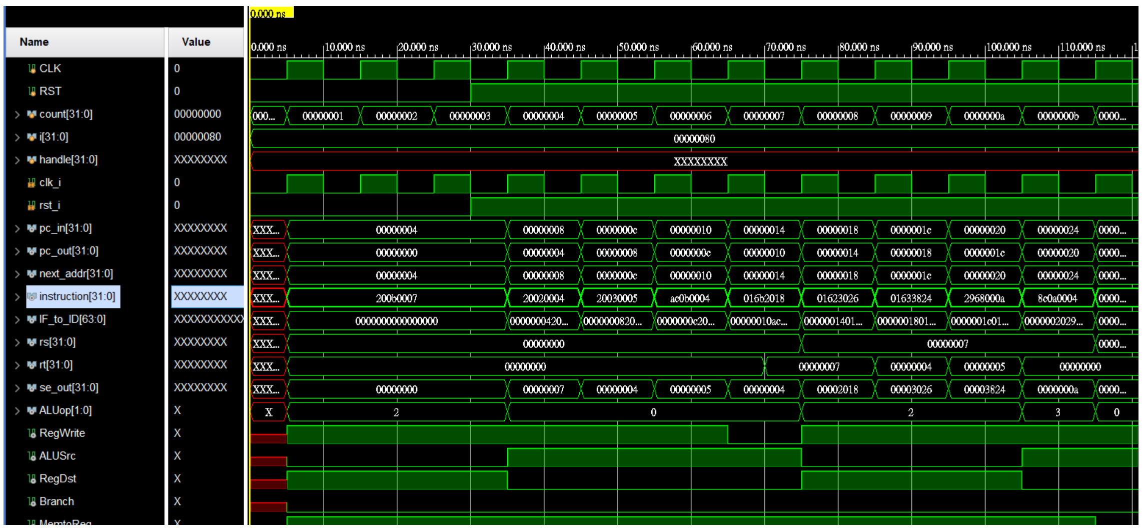
| Name | Value |
|---|---|
| CLK | 0 |
| RST | 0 |
| count[31:0] | 00000000 |
| i[31:0] | 00000080 |
| handle[31:0] | XXXXXXXX |
| clk_i | 0 |
| rst_i | 0 |
| pc_in[31:0] | XXXXXXXX |
| pc_out[31:0] | XXXXXXXX |
| next_addr[31:0] | XXXXXXXX |
| instruction[31:0] | XXXXXXXX |
| IF_to_ID[63:0] | XXXXXXXXXX |
| rs[31:0] | XXXXXXXX |
| rt[31:0] | XXXXXXXX |
| se_out[31:0] | XXXXXXXX |
| ALUop[1:0] | X |
| RegWrite | X |
| ALUSrc | X |
| RegDst | X |
| Branch | X |
| MemtoReg | X |

0.000 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
000... 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000a 0000000b 0000...
00000080
XXXXXXXX
XXX... 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 00000024 0000...
XXX... 00000000 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 0000...
XXX... 00000004 00000008 0000000c 00000010 00000014 00000018 0000001c 00000020 00000024 0000...
XXX... 20010003 20020004 20030001 ac010004 00212020 00223025 00233824 00822822 0022402a 1022...
XXX... 0000000000000000 0000000420... 0000000820... 0000000c20... 00000010ac... 0000001400... 0000001800... 0000001c00... 0000002000... 0000...
XXX... 00000000 00000003 000... 000... 0000...
XXX... 00000000 00000003 00000004 00000001 00000004
XXX... 00000000 00000003 00000004 00000001 00000004 00002020 00003025 00003824 00002822 0000...
X 2 0 2

## Test 2



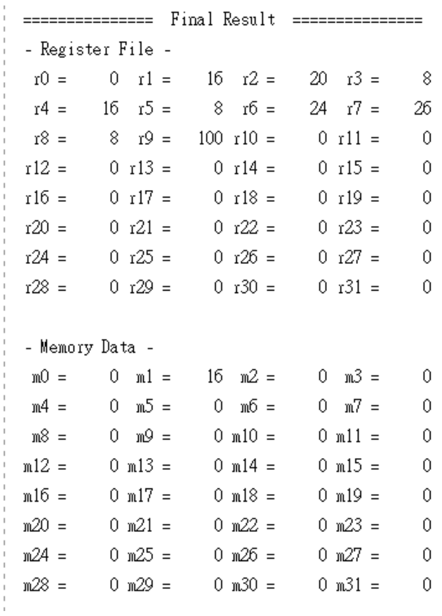
```
=============== Final Result ===============
- Register File -
 r0 =      0  r1 =      0  r2 =      4  r3 =      5
 r4 =     49  r5 =      0  r6 =      3  r7 =      5
 r8 =      1  r9 =      0 r10 =      7 r11 =      7
r12 =      0 r13 =      0 r14 =      0 r15 =      0
r16 =      0 r17 =      0 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
```



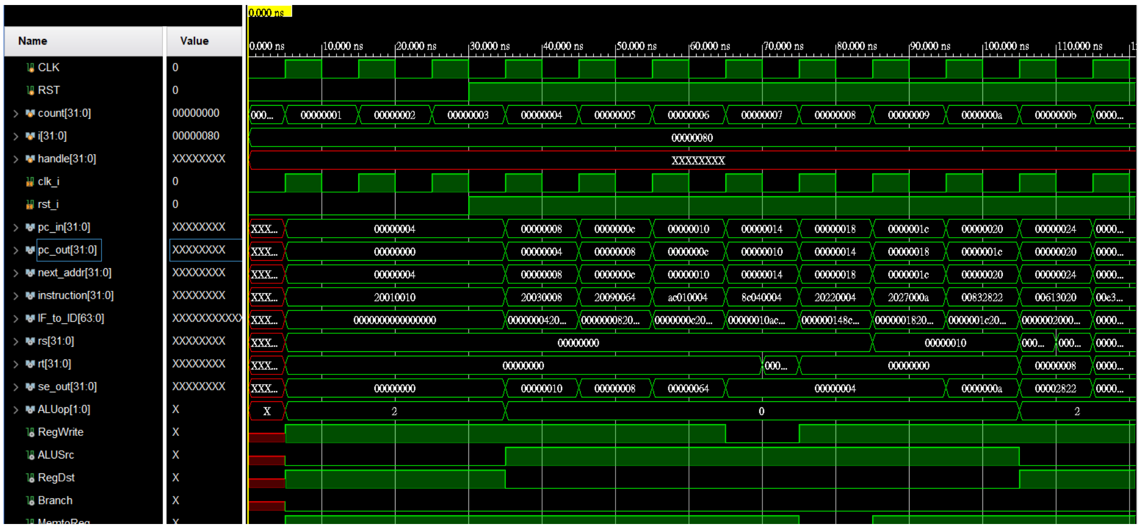
## Test 3

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

| Origin | After reorder |
|---|---|
| 00100000000000010000000000010000 | 00100000000000010000000000010000 |
| 00100000001000100000000000000100 | 00100000000000110000000000001000 |
| 00100000000000110000000000001000 | 00100000000010010000000001100100 |
| 10101100000000010000000000000100 | 10101100000000010000000000000100 |
| 10001100000001000000000000000100 | 10001100000001000000000000000100 |
| 00000000100000110010100000100010 | 00100000001000100000000000000100 |
| 00000000011000010011000000100000 | 00100000001001110000000000001010 |
| 00100000001001110000000000001010 | 00000000100000110010100000100010 |
| 00000000111000110100000000100100 | 00000000011000010011000000100000 |
| 00100000000010010000000001100100 | 00000000111000110100000000100100 |



```
==============  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
```



| Name | Value |
|---|---|
| CLK | 0 |
| RST | 0 |
| count[31:0] | 00000000 |
| i[31:0] | 00000080 |
| handle[31:0] | XXXXXXXX |
| clk_i | 0 |
| rst_i | 0 |
| pc_in[31:0] | XXXXXXXX |
| pc_out[31:0] | XXXXXXXX |
| next_addr[31:0] | XXXXXXXX |
| instruction[31:0] | XXXXXXXX |
| IF_to_ID[63:0] | XXXXXXXXXXX |
| rs[31:0] | XXXXXXXX |
| rt[31:0] | XXXXXXXX |
| se_out[31:0] | XXXXXXXX |
| ALUop[1:0] | X |
| RegWrite | X |
| ALUSrc | X |
| RegDst | X |
| Branch | X |
| MemtoReg | X |

## 4 Problems and Solutions

接線時因為我的 pipline register 是用一大坨的 array wire 去存，導致到下一個 stage 拉線出去時，常因為 index 值算錯整組 CPU 爆掉，造成 debug 不易，比較好的做法應該是 pipeline register 用好幾組 wire 去組成，這樣程式碼的可讀性也較高，不過最後還是成功 de 出了 bug，所以就沒改寫法了。

## 5 Summary

在了解 pipelined CPU 的運作之前，認為 pipelined CPU 的結構一定很複雜難以實作，不過後來發現原來只需加上數個 pipline registers 便能實現 pipelined CPU 的功能，不過遇到 hazard 時，只能調動機器碼的順序，因此功能的完整度還略顯不足。