

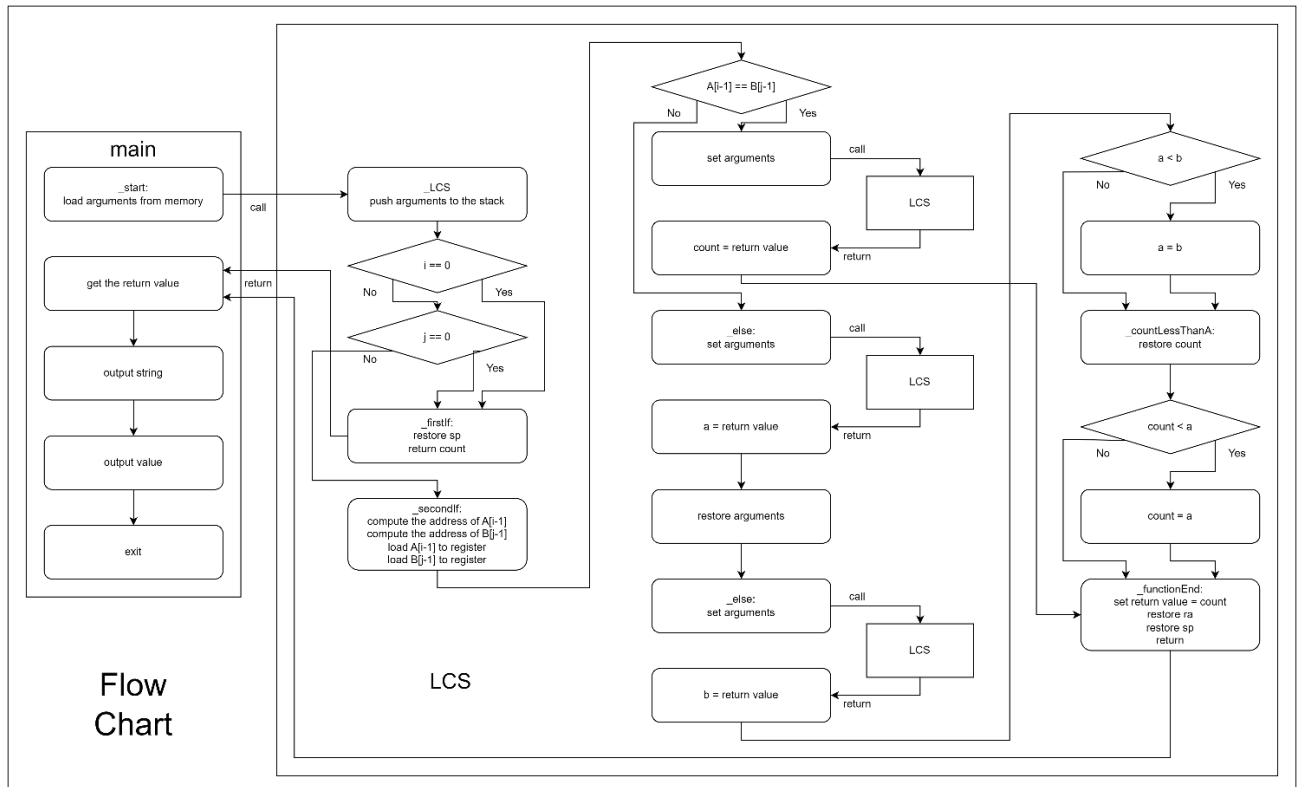
Department of Computer Science
National Tsing Hua University
CS4100 Computer Architecture
 Spring, 2024, Homework 5
 Due date: 5/26/2024 23:59

1. Assembly Coding

a. Test Result

Testcase 1	Testcase 2
<div>Source code</div> <div>Input type: <input checked="" type="radio"/> Assembly <input type="radio"/></div> <pre> 1 # Reference: https://www.geeksforgeeks.org/longest-common 2 .data 3 .align 4 4 5 # =====testcase1===== 6 A: .word 1, 2, 3, 2, 1 7 i: .word 5 8 B: .word 8, 7, 6, 4 9 j: .word 4 10 strOutput: .string "Max length of common subarray: " 11 # output: Max length of common subarray: 0 12 # ===== 13 14 # =====testcase2===== 15 # A: .word 1, 2, 8, 2, 1 16 # i: .word 5 17 # B: .word 8, 2, 1, 4, 7 18 # j: .word 5 19 strOutput: .string "Max length of common subarray: " 20 # output: Max length of common subarray: 3 21 # ===== 22 23 .text 24 .global _start 25 26 # Start your coding below, don't change anything upper ex </pre> <div>Console</div> <div>Max length of common subarray: 0</div>	<div>Source code</div> <div>Input type: <input checked="" type="radio"/> Assembly <input type="radio"/></div> <pre> 10 # strOutput: .string "Max length of common subarray: " 11 # output: Max length of common subarray: 0 12 # ===== 13 14 # =====testcase2===== 15 A: .word 1, 2, 8, 2, 1 16 i: .word 5 17 B: .word 8, 2, 1, 4, 7 18 j: .word 5 19 strOutput: .string "Max length of common subarray: " 20 # output: Max length of common subarray: 3 21 # ===== 22 23 .text 24 .global _start 25 26 # Start your coding below, don't change anything upper ex </pre> <div>Console</div> <div>Max length of common subarray: 3</div>

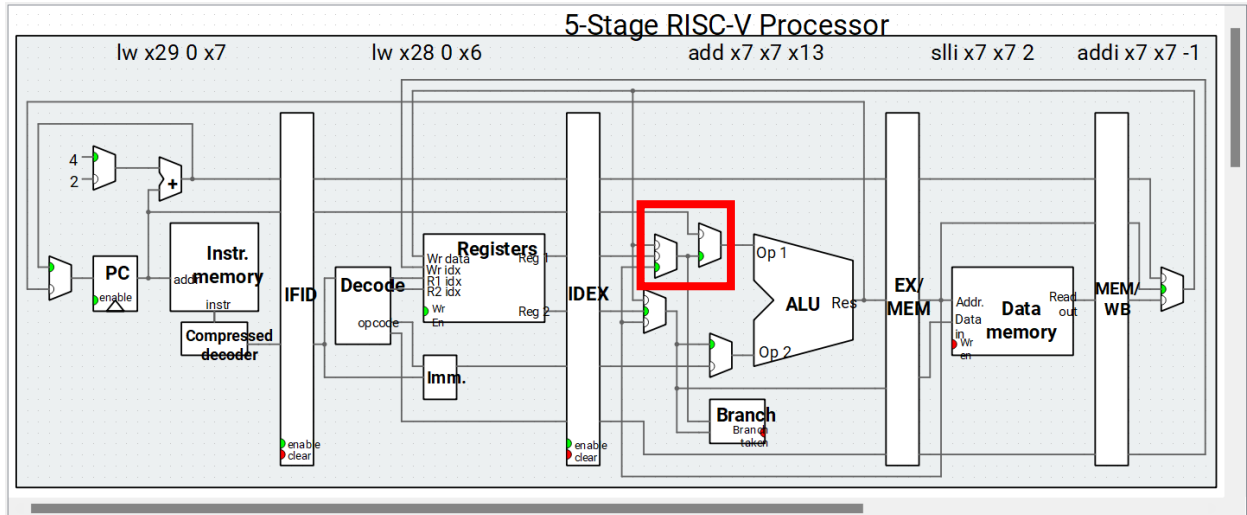
b. Flow Chart



2. Hazard in Your Code

- Type 1:

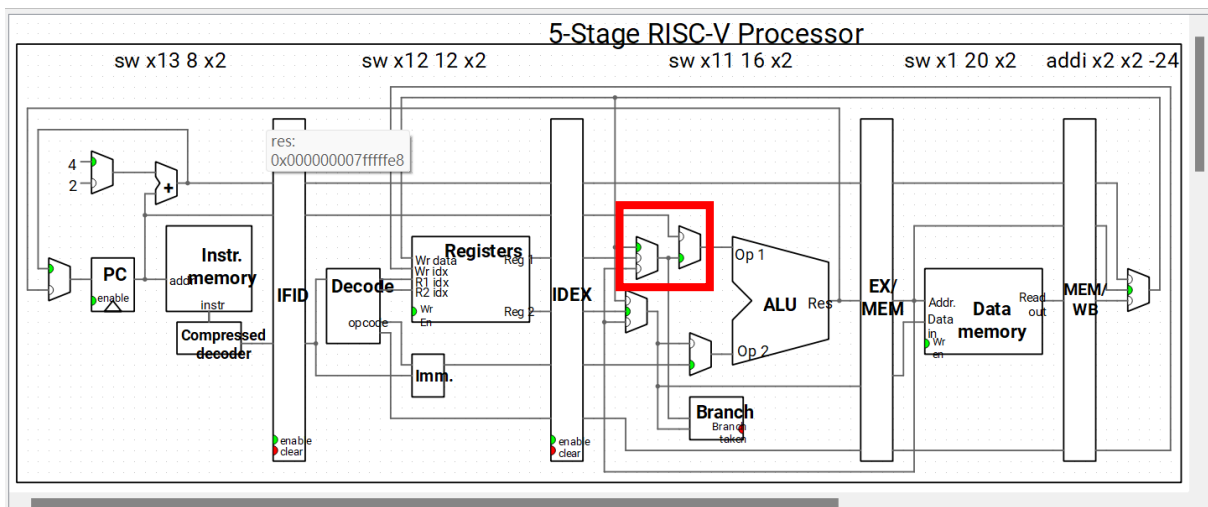
```
83 # count B[i-1] address
84 mv t2, a4
85 addi t2, t2, -1
86 slli t2, t2, 2
87 add t2, t2, a3
```



t2 是 slli t2, t2, 2 的 rd，又被 add t2, t2, a3 當作 rs1，因此會產生 Type 1 hazard，紅色框框圈起來的 mux 顯示了為了解決這個 hazard，把 EX/MEM 存的值(上一個 ALU 結果) forward 給 ALU，這樣就可以拿到正確的值做計算。

- Type 2:

```
55 _LCS:
56 # save data
57 addi sp, sp, -24
58 sw ra, 20(sp)
59 sw a1, 16(sp)
```



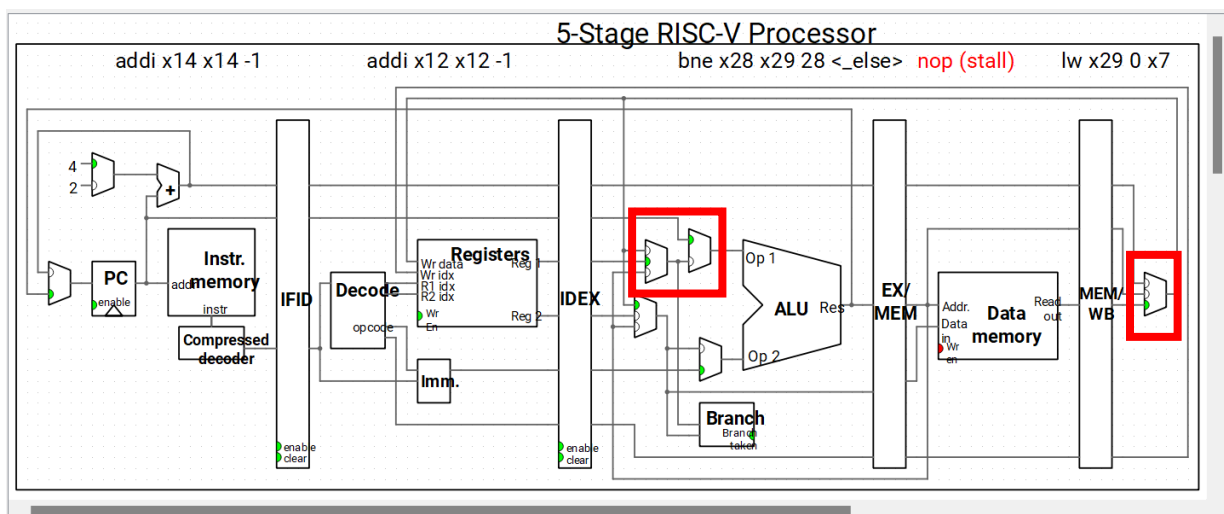
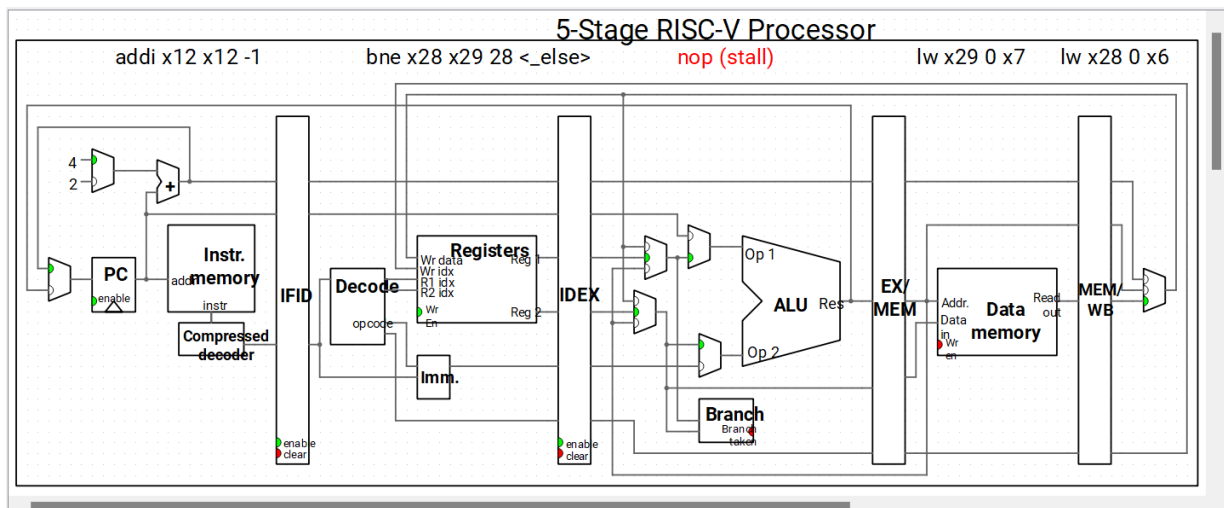
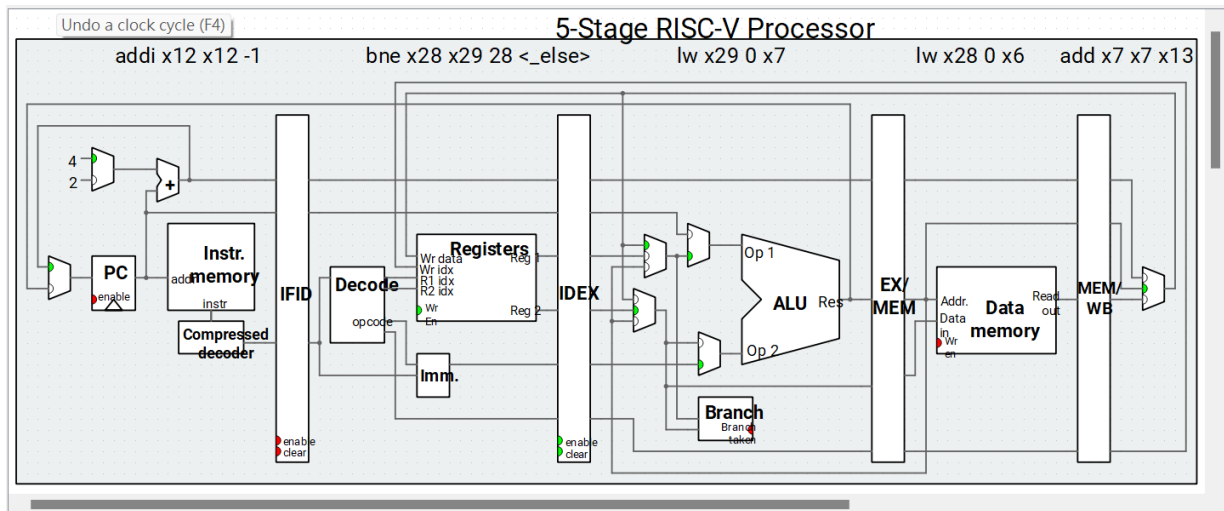
sp 是 addi sp, sp, -24 的 rd，也是 sw a1, 16(sp) 的 rs1，因此會有 Type 2 hazard，但因為結果早就算出來了，所以只需要從 MEM/WB 將正確的值 forward 給 ALU 就可以解決這個 hazard 了。

● Type 3:

```

89 # load A[i-1] to t3
90 lw t3, 0(t1)
91
92 # load B[j-1] to t4
93 lw t4, 0(t2)
94
95 # (A[i-1] != B[j-1]) => _else
96 bne t3, t4, _else

```



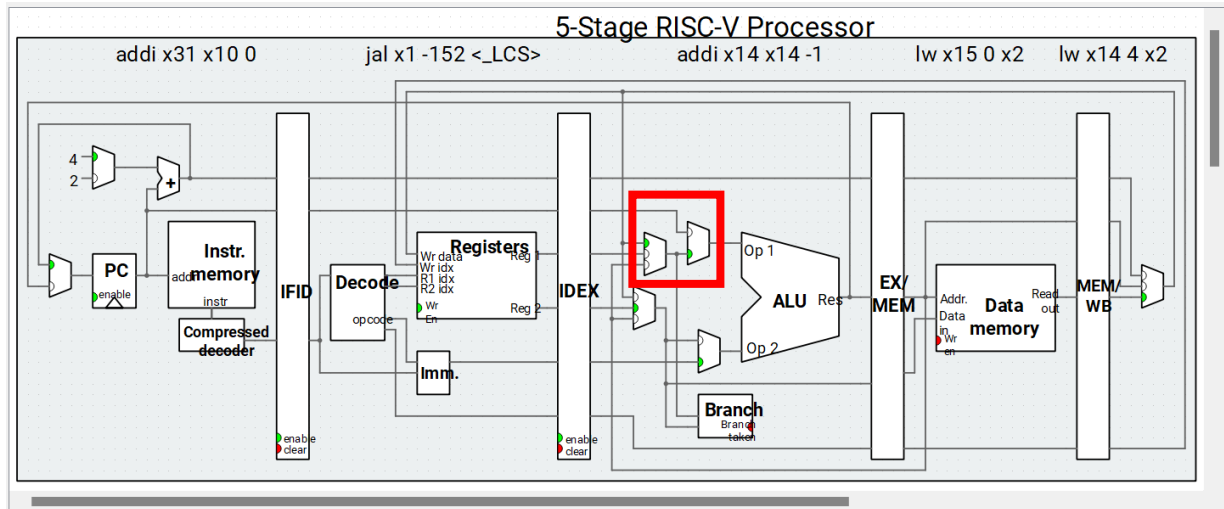
lw t4, 0(t2)與 bne t3, t4, _else 會有 Type 3 hazard，ID/EX 的 clear 訊號被設成 1，IF/ID 的 enable 訊號被設成 0，PC 的 enable 訊號被設成 0，使處理器 stall 一個 cycle，產生一個 bubble 等待 load 指令從 memory 拿到正確的值，然後透過 forward 的方式(紅色框框)，將正確的值 forward 給 ALU 的 input，以解決 hazard 的問題。

- Type 4:

```

120    lw a4, 4(sp)
121    lw a5, 0(sp)
122
123    # b(t6) = LCS(i,
124    addi a4, a4, -1

```



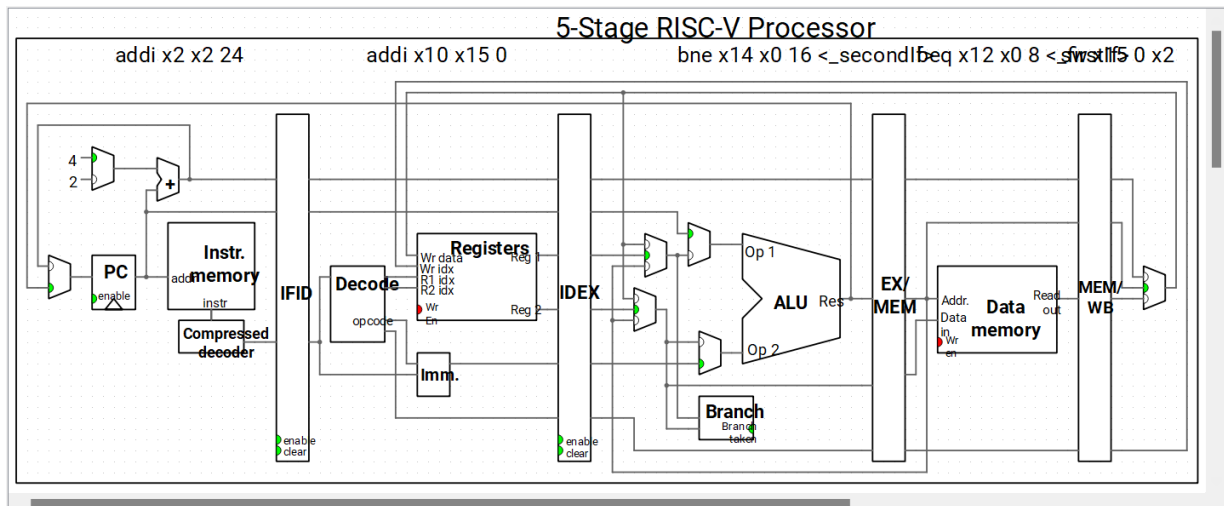
lw a4, 0(sp)與 addi a4, a4, -1 會產生 Type 4 hazard，與 Type 3 不同的是，由於正確的值已經被算出來了，因此不用 stall，只需要將 MEM/WB 的值 forward 給 ALU，就可以解決這個 hazard。

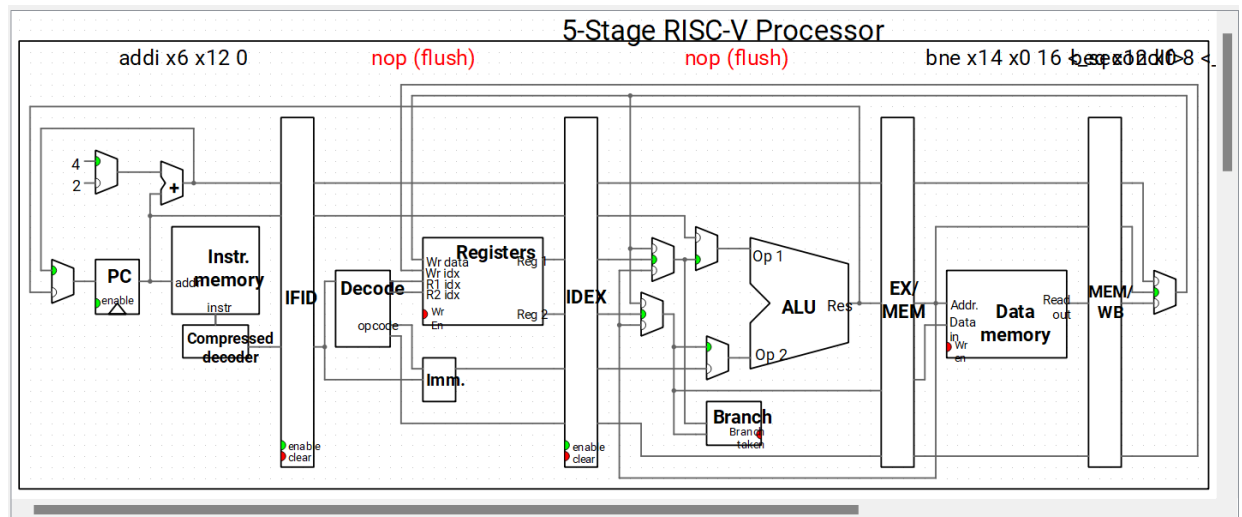
- Type 5:

```

65    # (i == 0 || j == 0)
66    beq a2, zero, _firstIf # c
67    bne a4, zero, _secondIf # c
68
69    _firstIf:
70    # return count
71    mv a0, a5
72    addi sp, sp, 24
73    jr ra
74
75
76    _secondIf:
77    # count A[i-1] address
78    mv t1, a2
79    addi t1, t1, -1

```





branch taken 後若 branch 指令後面還有指令執行的話需要 flush 掉，當偵測到 Type 5 hazard 後，IF/IF 和 ID/EX 的 clear 訊號都被設成 1，以 flush 掉不需要執行的指令，並從已經算好的新 PC 位址拿到正確的指令，以解決這個 hazard。