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

Source code — Input type: Assembly

```
# Reference: https://www.geeksforgeeks.org/longest-common
.data
.align 4

# ========testcase1==========
A: .word 1, 2, 3, 2, 1
i: .word 5
B: .word 8, 7, 6, 4
j: .word 4
strOutput: .string "Max length of common subarray: "
# output: Max length of common subarray: 0
# ============================

# ========testcase2==========
# A: .word 1, 2, 8, 2, 1
# i: .word 5
```

Console

```
Max length of common subarray: 0
```

Testcase 2

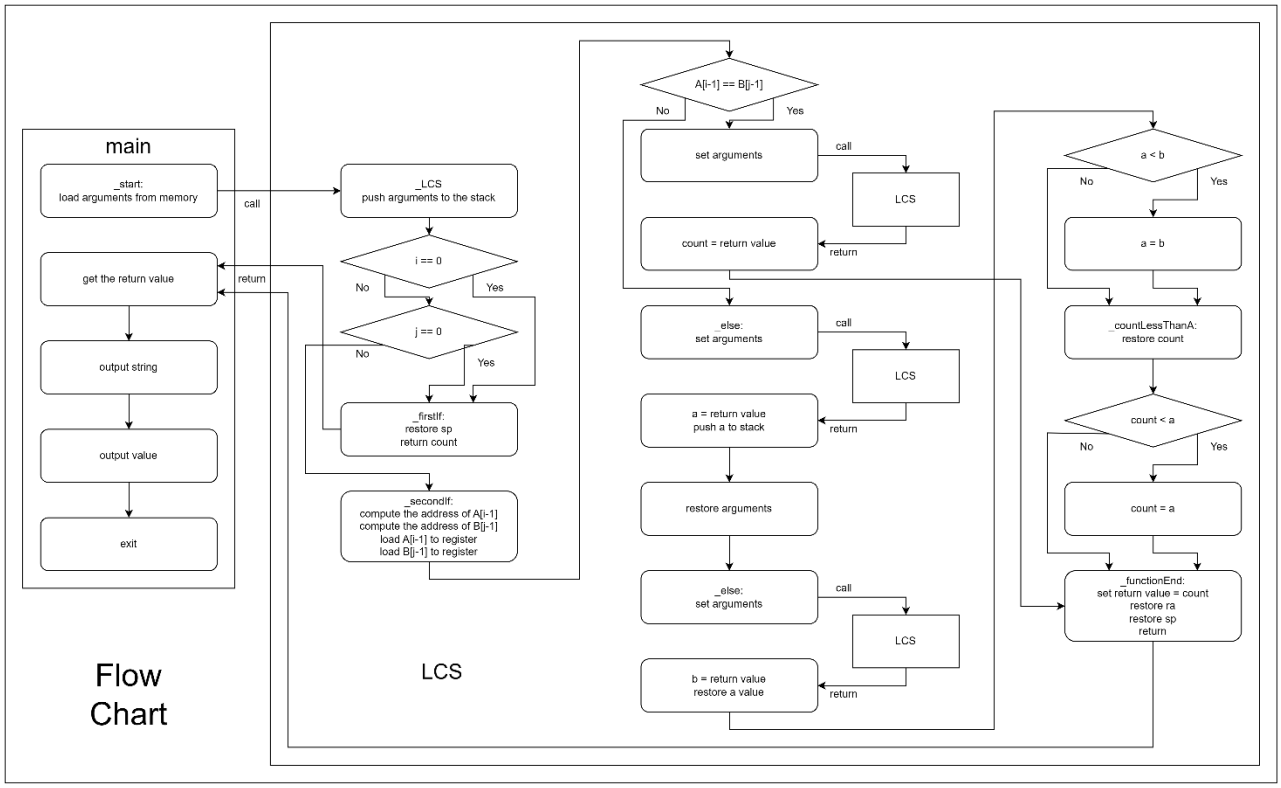
Source code — Input type: Assembly

```
# strOutput: .string "Max length of common subarray: "
# output: Max length of common subarray: 0
# ============================

# ========testcase2==========
A: .word 1, 2, 8, 2, 1
i: .word 5
B: .word 8, 2, 1, 4, 7
j: .word 5
strOutput: .string "Max length of common subarray: "
# output: Max length of common subarray: 3
# ============================

.text
.global _start
```

Console

```
Max length of common subarray: 3
```

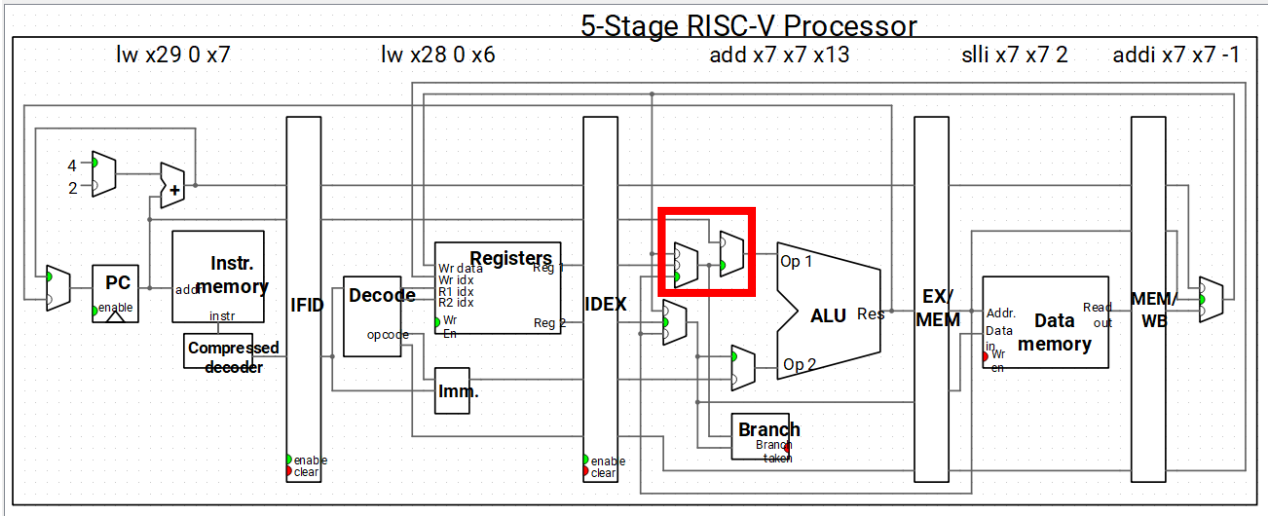
b. Flow Chart



2. Hazard in Your Code

- Type 1:

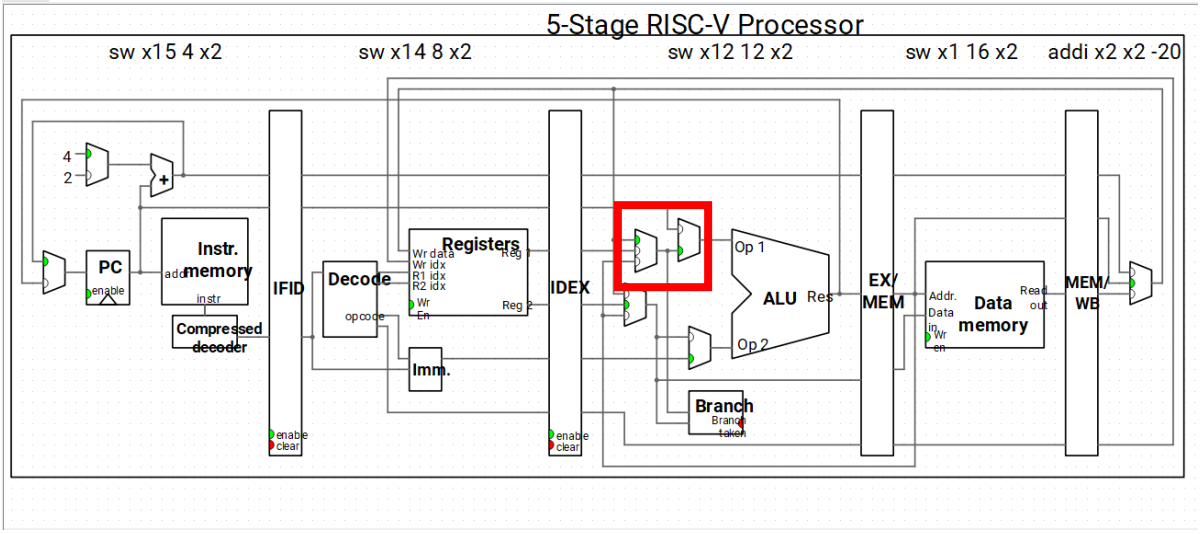
```
81      # count B[i-1] address
82      mv t2, a4
83      addi t2, t2, -1
84      slli t2, t2, 2
85      add t2, t2, a3
```



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 顯示了為了要解決這個 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:

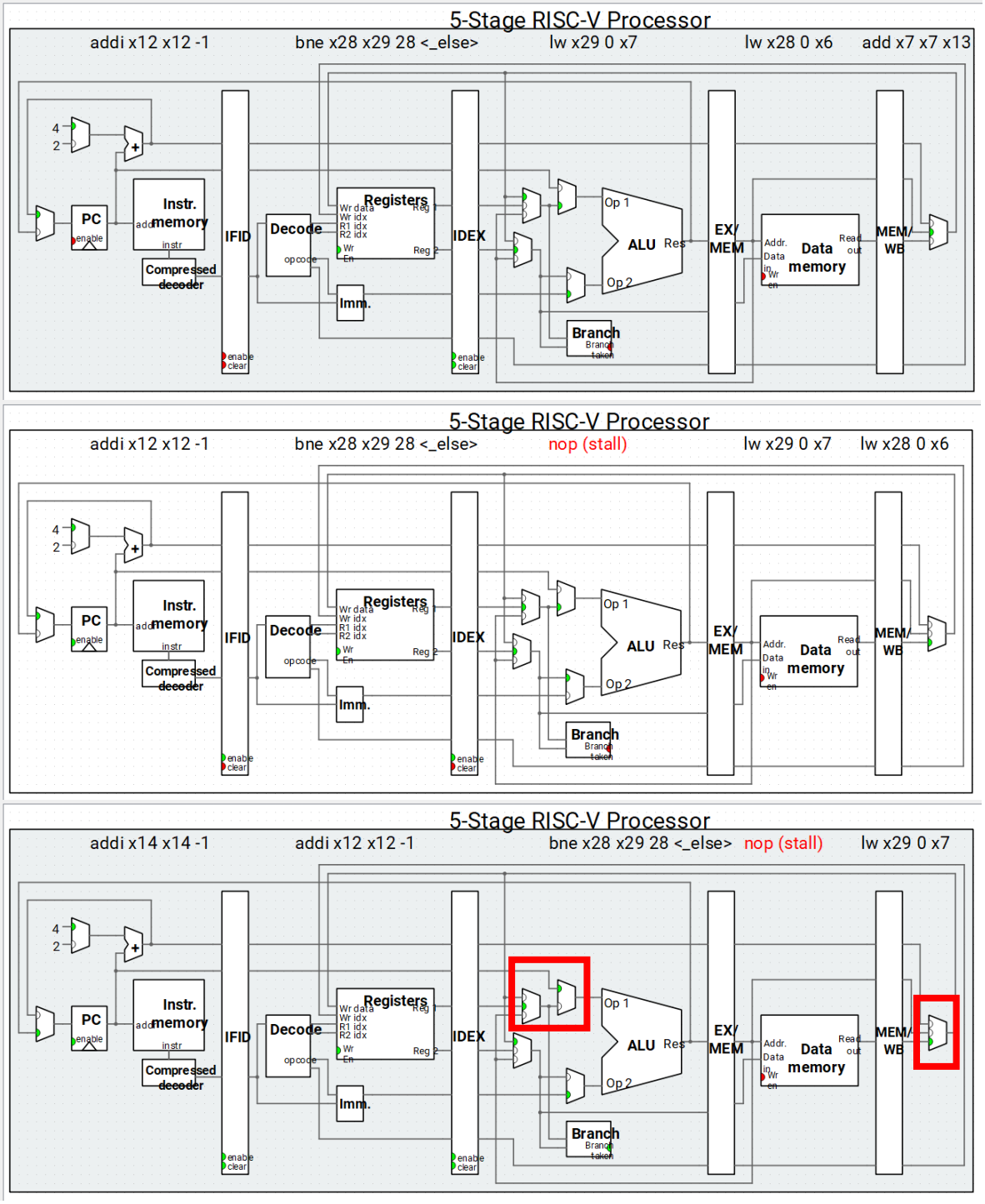
```
55  _LCS:
56      # save data
57      addi sp, sp, -20
58      sw ra, 16(sp) # save return address
59      sw a2, 12(sp) # save i
```



sp 是 addi sp, sp, -20 的 rd，也是 sw a2, 12(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:

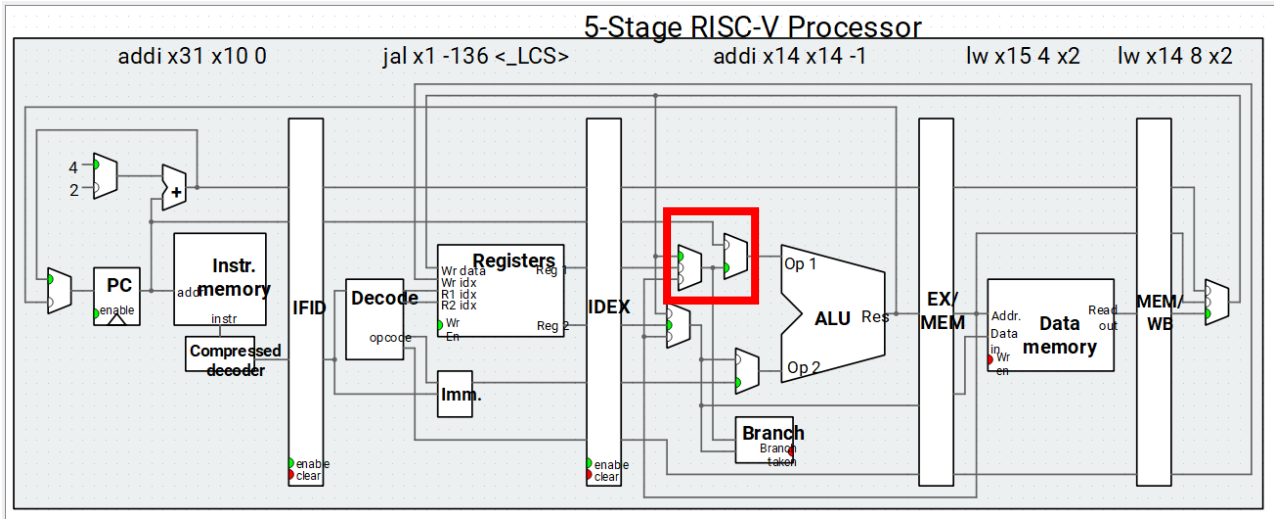
```
87      # load A[i-1] to t3
88      lw t3, 0(t1)
89
90      # load B[j-1] to t4
91      lw t4, 0(t2)
92
93      # (A[i-1] != B[j-1]) => _else
94      bne t3, t4, _else
```



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 foward 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:

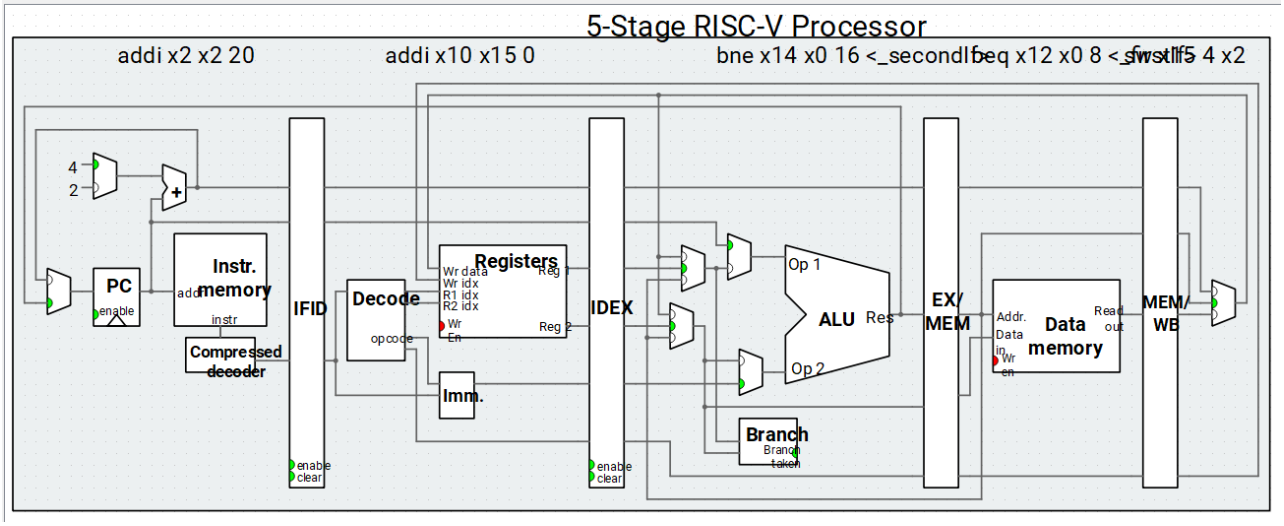
```
114     # restore the arguments
115     lw a2, 12(sp)
116     lw a4, 8(sp)
117     lw a5, 4(sp)
118
119     # b(t6) = LCS(i, j - 1, A, B, 0)
120     addi a4, a4, -1
```

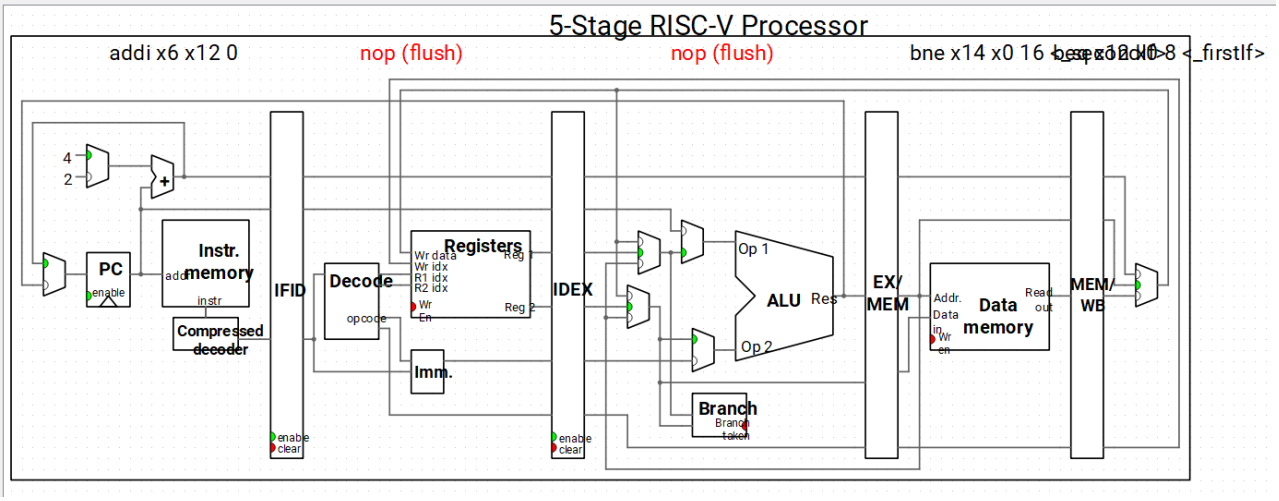


lw a4, 8(sp)與 addi a4, a4, -1 會產生 Type 4 hazard，與 Type 3 不同的是，由於正確的值已經被算出來了，因此不用 stall，只需要將 MEM/WB 的值 forward 給 ALU，就可以解決這個 hazard。

- Type 5:

```
63      # (i == 0 || j == 0)
64      beq a2, zero, _firstIf # c
65      bne a4, zero, _secondIf #
66
67  _firstIf:
68      # return count
69      mv a0, a5
70      addi sp, sp, 20
71      jr ra
72
73
74  _secondIf:
75      # count A[i-1] address
76      mv t1, a2
77      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。