HW 5

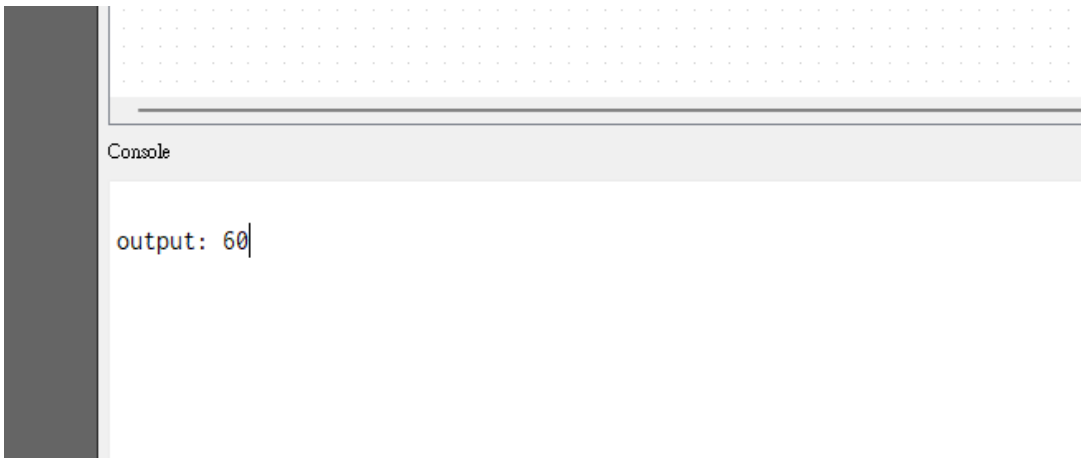
Computer Architecture

學號：111062107

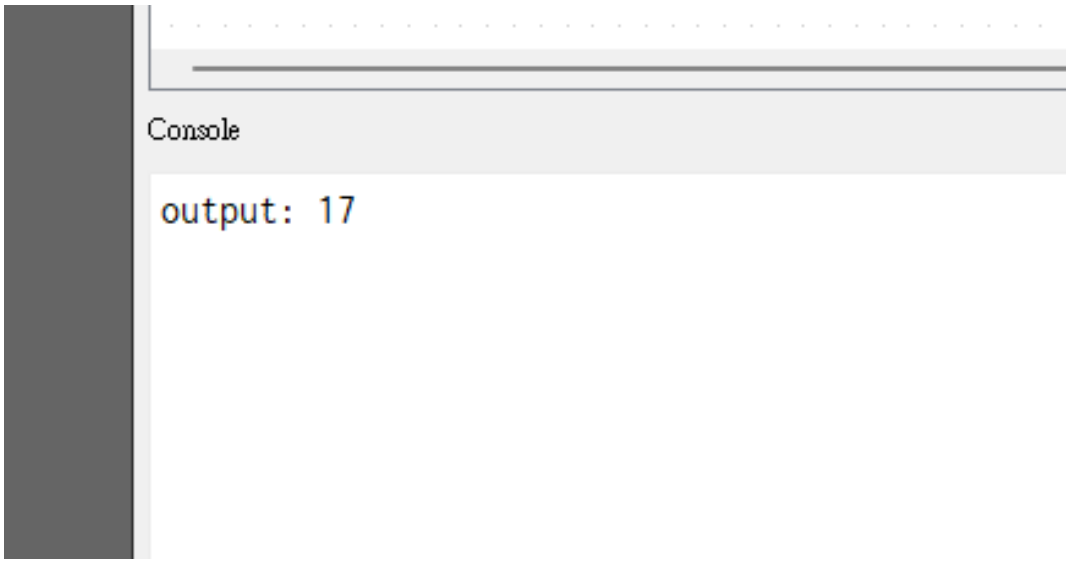
姓名：鄧弘利

1. Assembly Coding

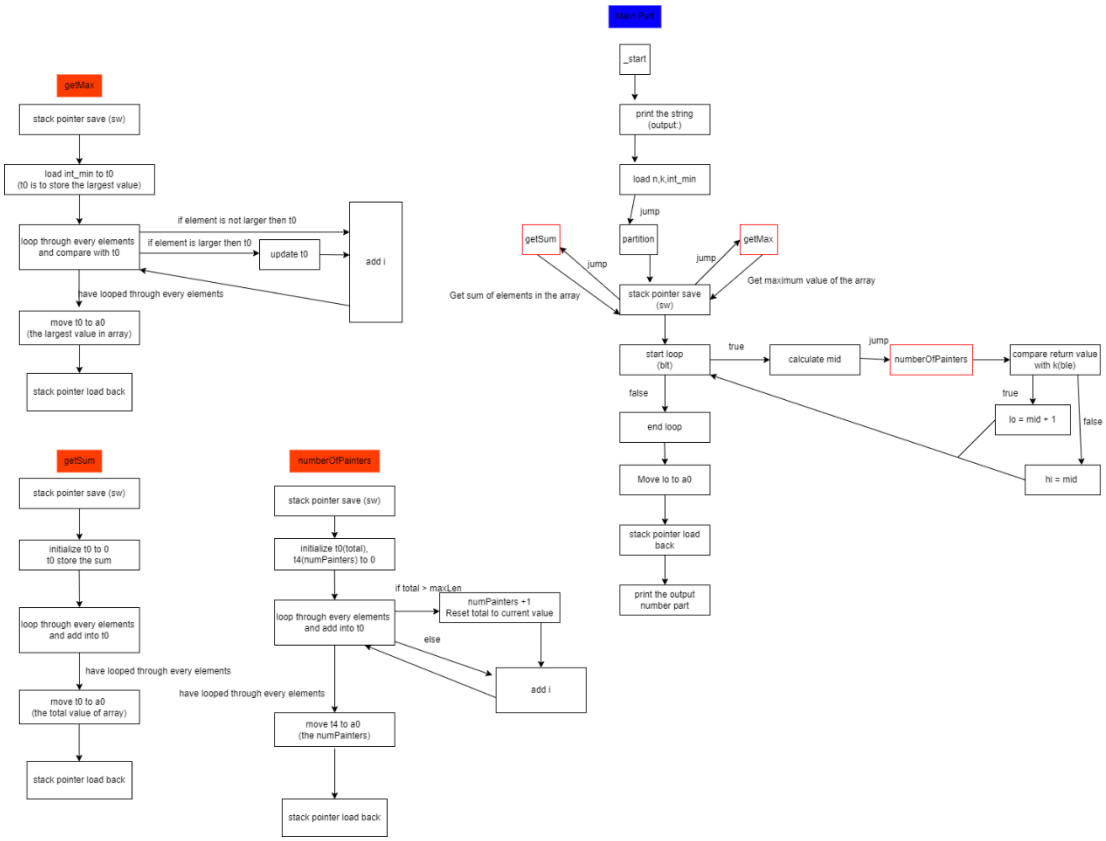
For testcase 1



For testcase 2



Flowchart



主要的流程是藍色為底下面的部份。主要就是一開始先印出 str: .string "output: "的部分，再來 load n,k,INT_MIN 的值，就開始跑最主要的 partition。進到 partition 之後，先將會用到的變數、ra 等等先儲存進 stack，接下來個跑一遍 getSum 還有 getMax，再用這兩個 return value，進行 loop，當符合條件的時候，計算 mid 後跑 numberOfPainters，再用其 return 的值，根據條件更新 lo 或是 hi。當跑完之後，就將結果放在 a0，並在跳回去之後，印出來。

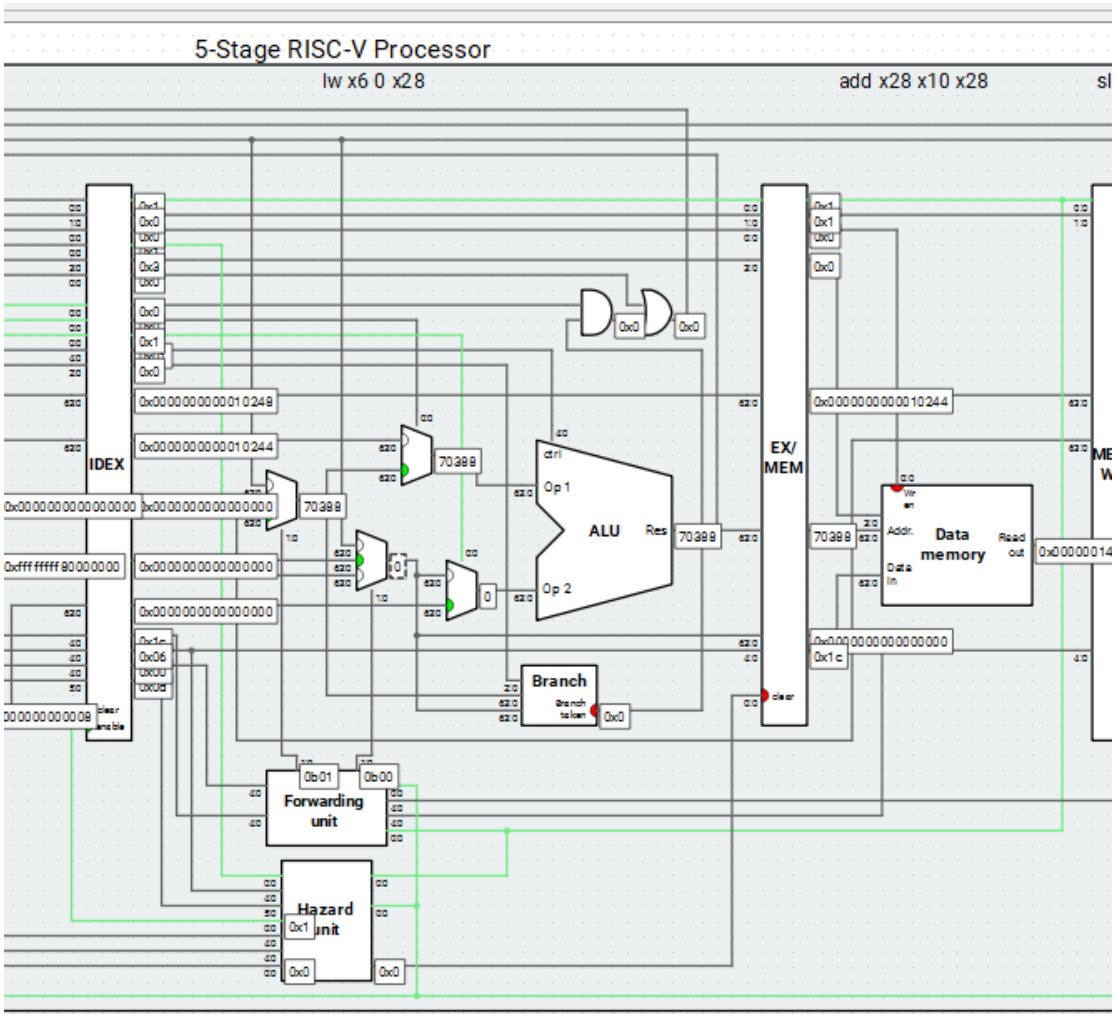
getMax、 getSum 以及 numberOfPainters 都是將每一個 element 都跑過一遍，根據不同條件，更新值，並在最後將值放進 a0，作為 return value。

2. Hazards in Your Code

A. Type (1) : R-types RAW (read after write) at the following 1st instruction

```
134     add t3, a0, t3      # Calculate address of arr[i]
135     lw t1, 0(t3)        # Load current value into t1
```

The register t3 is used to store the result of a0+t3 in the first instruction. However, t3 is also used in the next instruction lw as rs1.



In this example, t3(x28) = 70399.In Ripes, the processor detects dependency on MEM and EX stages for Type (1) . Then, the Forwarding unit sets the control signal (2b'01) of MUX before rs1 to select the t3 value forwarded from the MEM stage to the EX stage. As a result, the rs1 of ALU will be the result of the preceding instruction。

B. Type (2): R-types RAW at the following 2nd instruction

I don’t have this type.

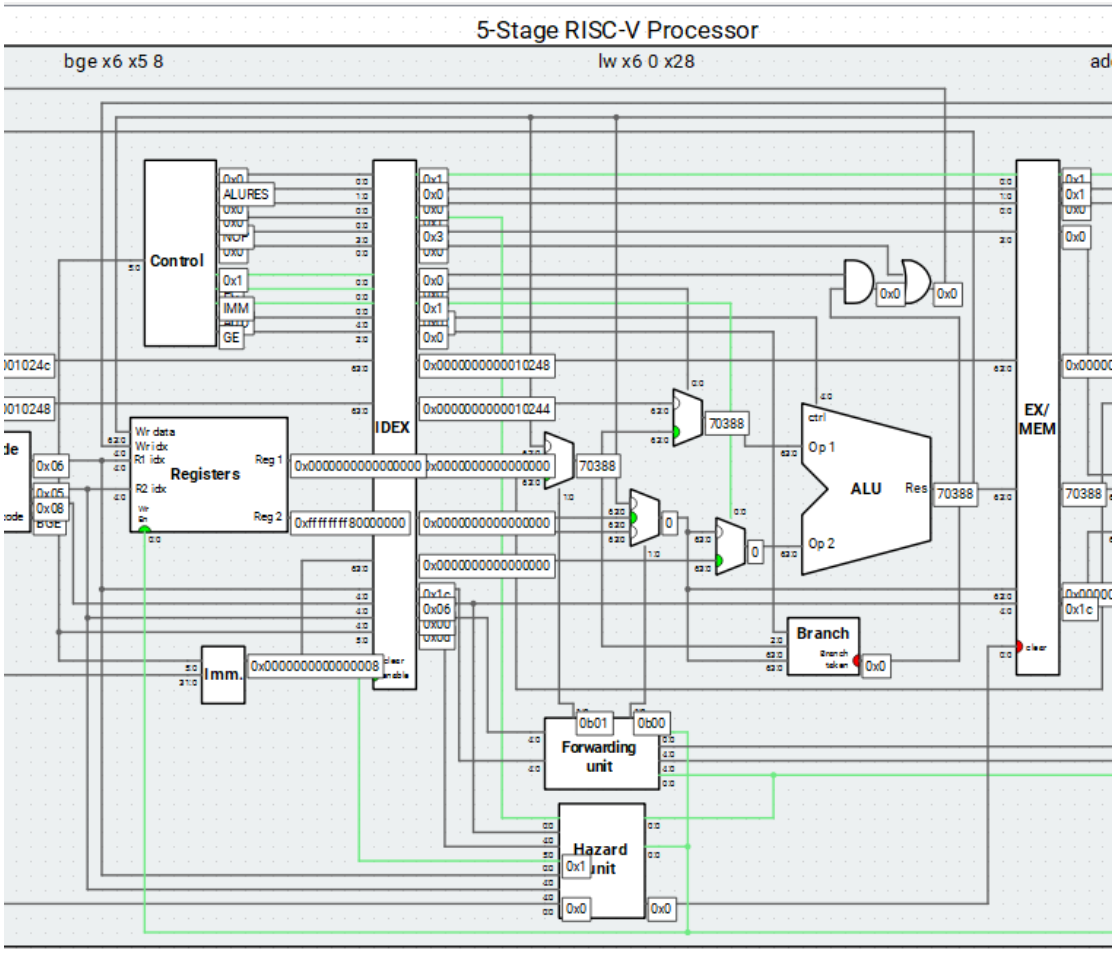
```
234     add t0, zero, zero
235     sub t1, zero, zero
236     add t2, t0, s0
```

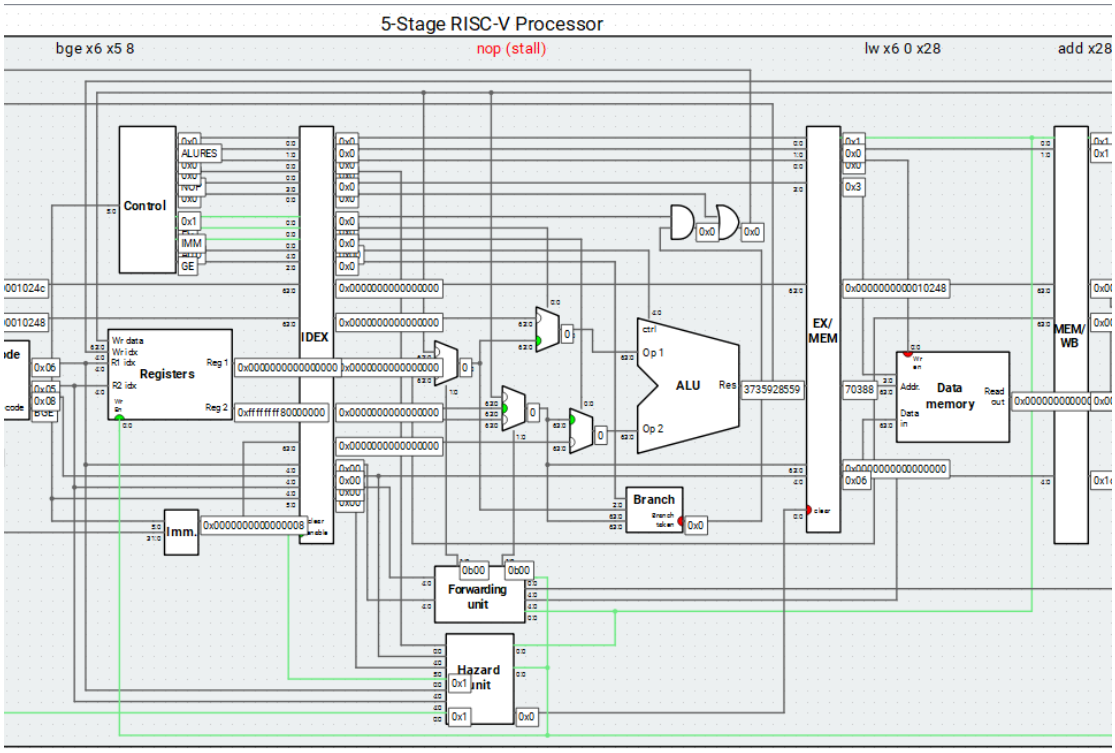
The register t0 is used to store the result of x0 + x0 in the first instruction. However, t0 is also used in the third instruction add as rs1.So it will happen the type 2 dependency.

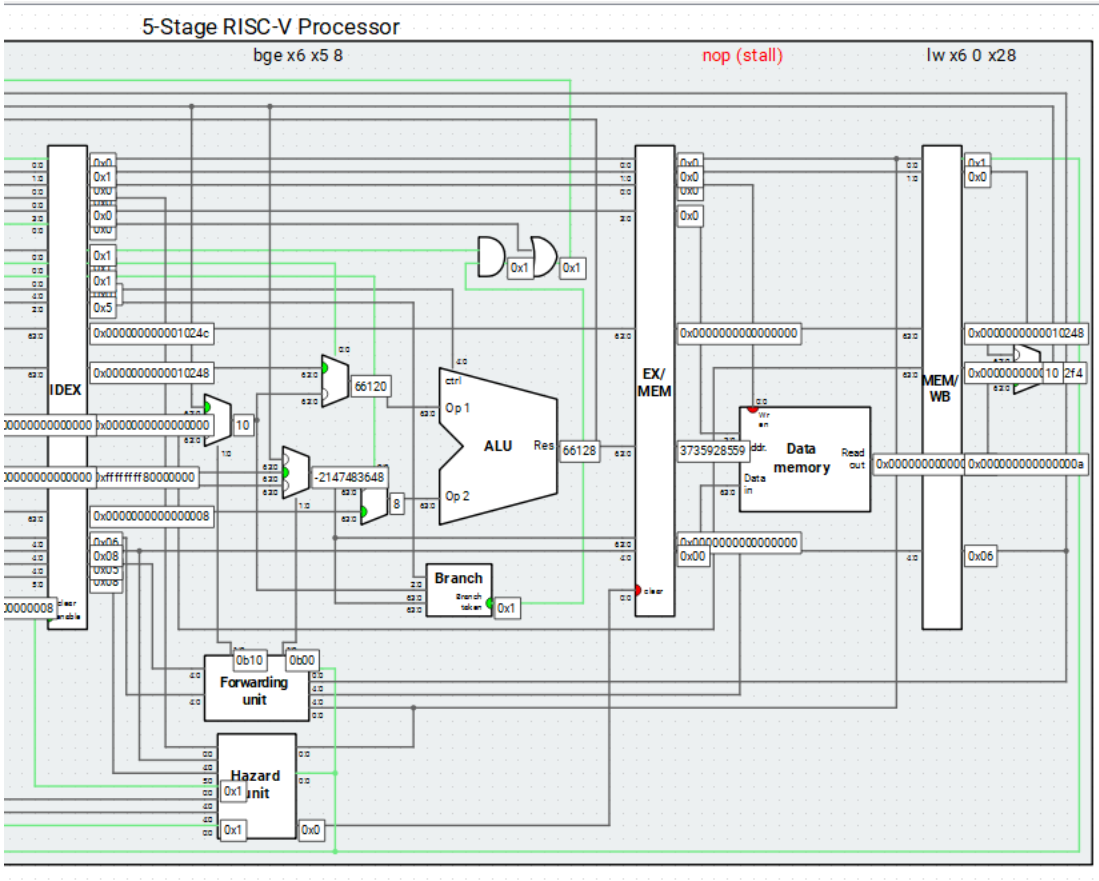
C. Type (3): Load RAW at the following 1st instruction

```
183     lw t1, 0(t3)        # Load current value into t1
184     bge t1, t0, Update      # Compare current value with max value
```

The register t1 is used to store the value at address (value in t3 + 0) in the first instruction. However, t1 is also used in the next instruction bge as rs1.







In this example, t1(x6) = 10,t0(x5) = INT_MIN.In Ripes, the processor first detects dependency on ID and EX stages for Type (3). Then, the Forwarding unit inserts a

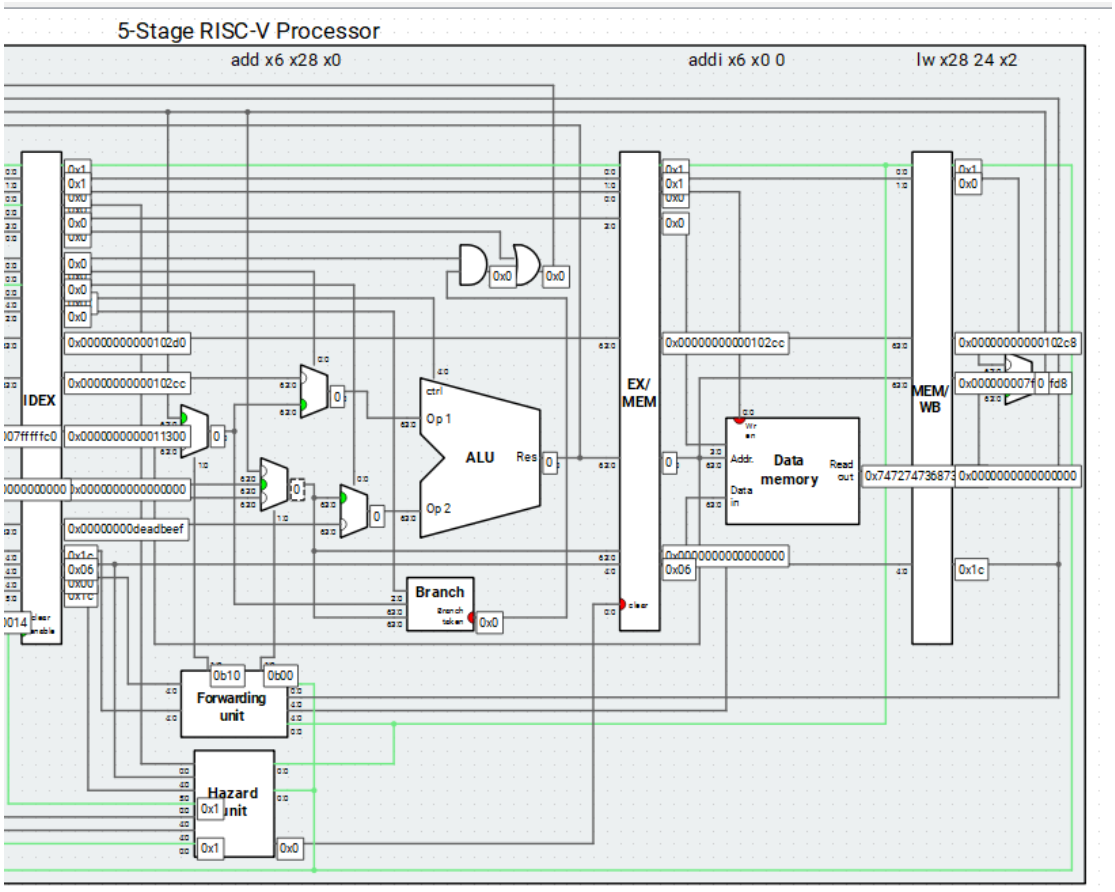
nop between these 2 instructions.It makes all control signals in ID/EX register to be 0, so a nop is created.Then,in third screenshot, the Forwarding unit sets the control signal (2b’10) of MUX before rs1 to select the t1(x6) value forwarded from the WB stage to the EX stage. As a result, the rs1 of Branch will be the result of the preceding instruction.

Ripe 會在發生這個 dependency type 的時候，插入 nop。bge x6 x5 8 不會進入下一個 stage，lw x6 0 x28 則會進到下一個 stage，中間就出現了 nop。

D. Type (4): Load RAW at the following 2nd instruction

```
231     lw t3, 24(sp)
232     addi t1, zero, 0
233     add t1, t3, zero
```

The register t3 is used to store the value at address which is value in sp + 24 in the first instruction. However, t3 is also used in the third instruction add as rs1.

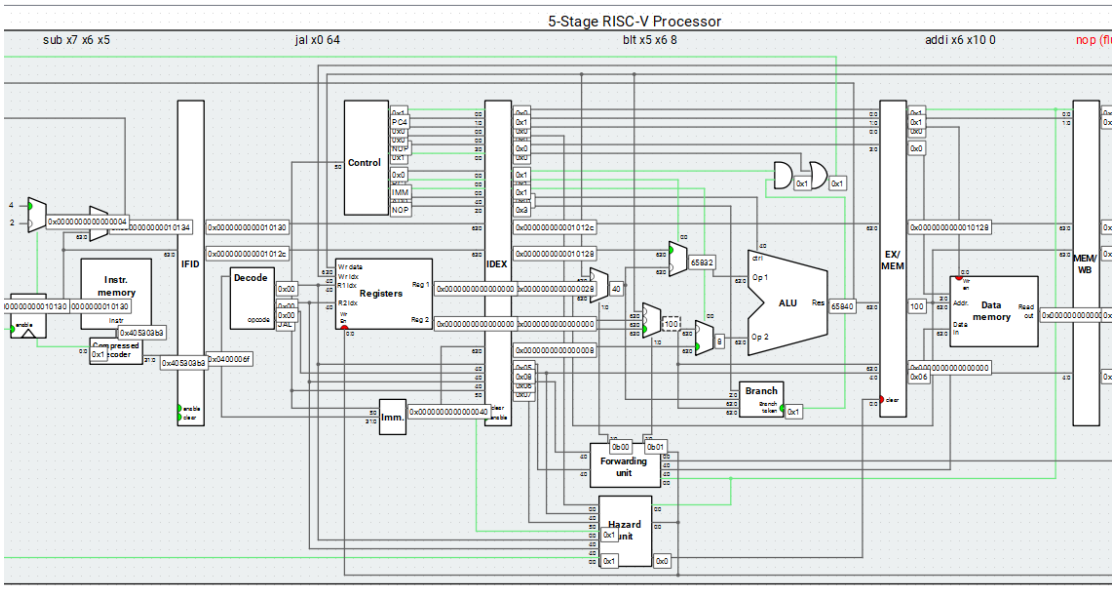


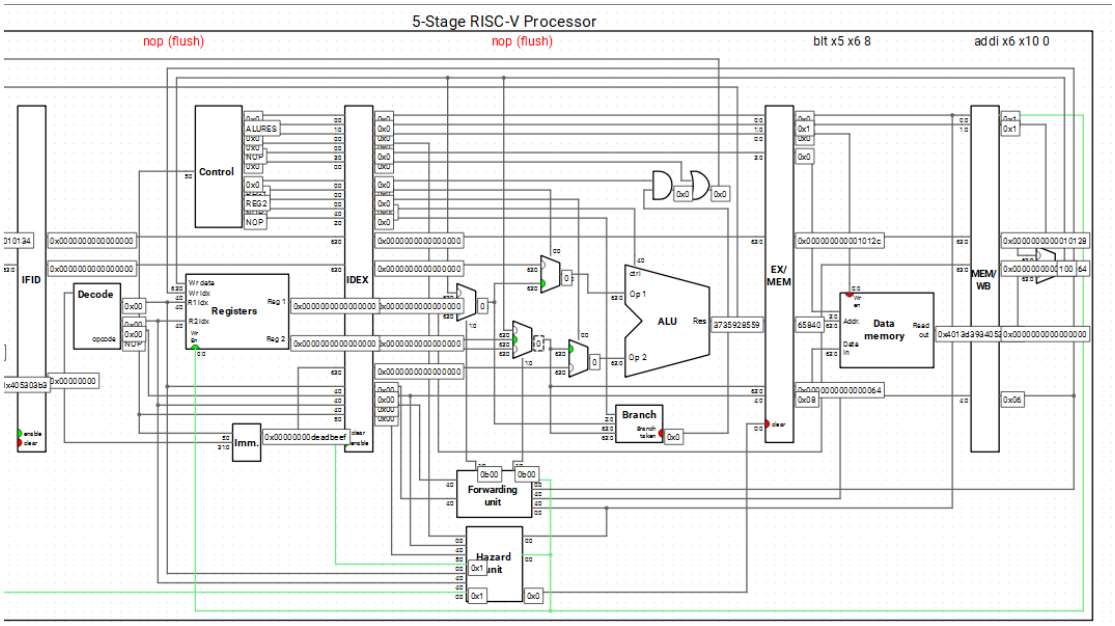
In this example, t1(x6) = 0,t3(x28) = 0.In Ripes, the processor detects dependency on WB and EX stages for Type (4). Then, the Forwarding unit sets the control signal (0b10) of MUX before rs1 to select the t3 value forwarded from the WB stage to the EX stage.

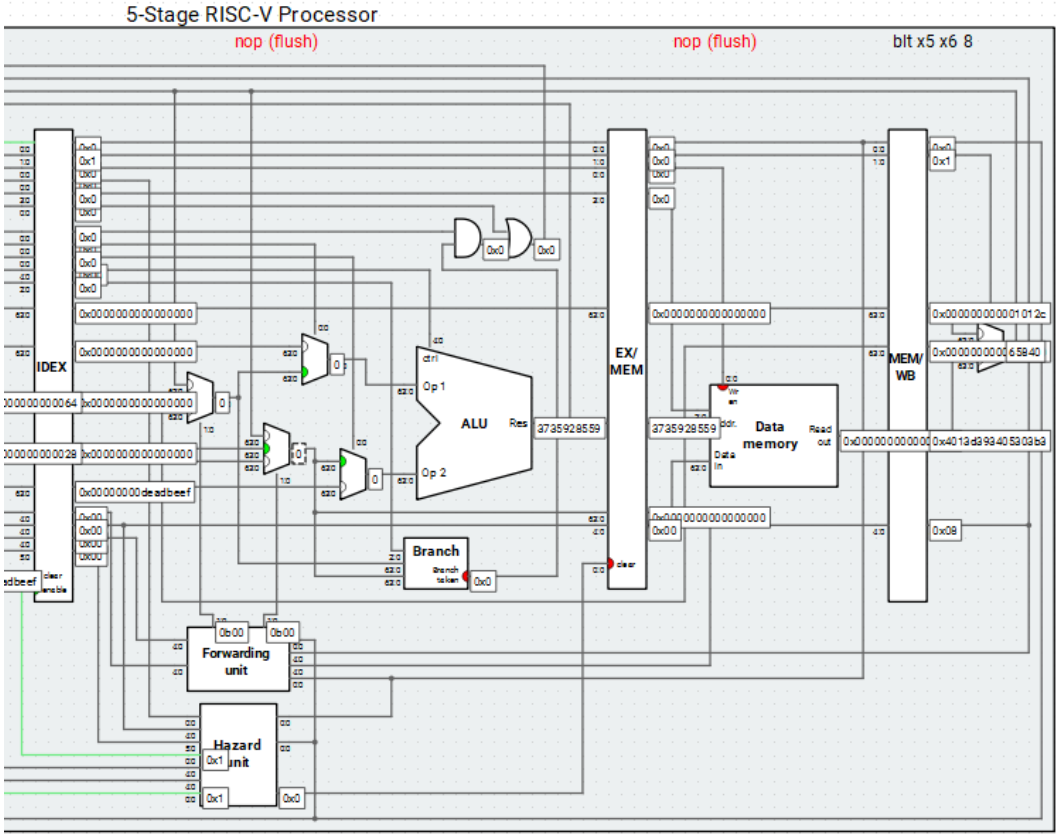
E. Type (5): Branch instruction (control hazard)

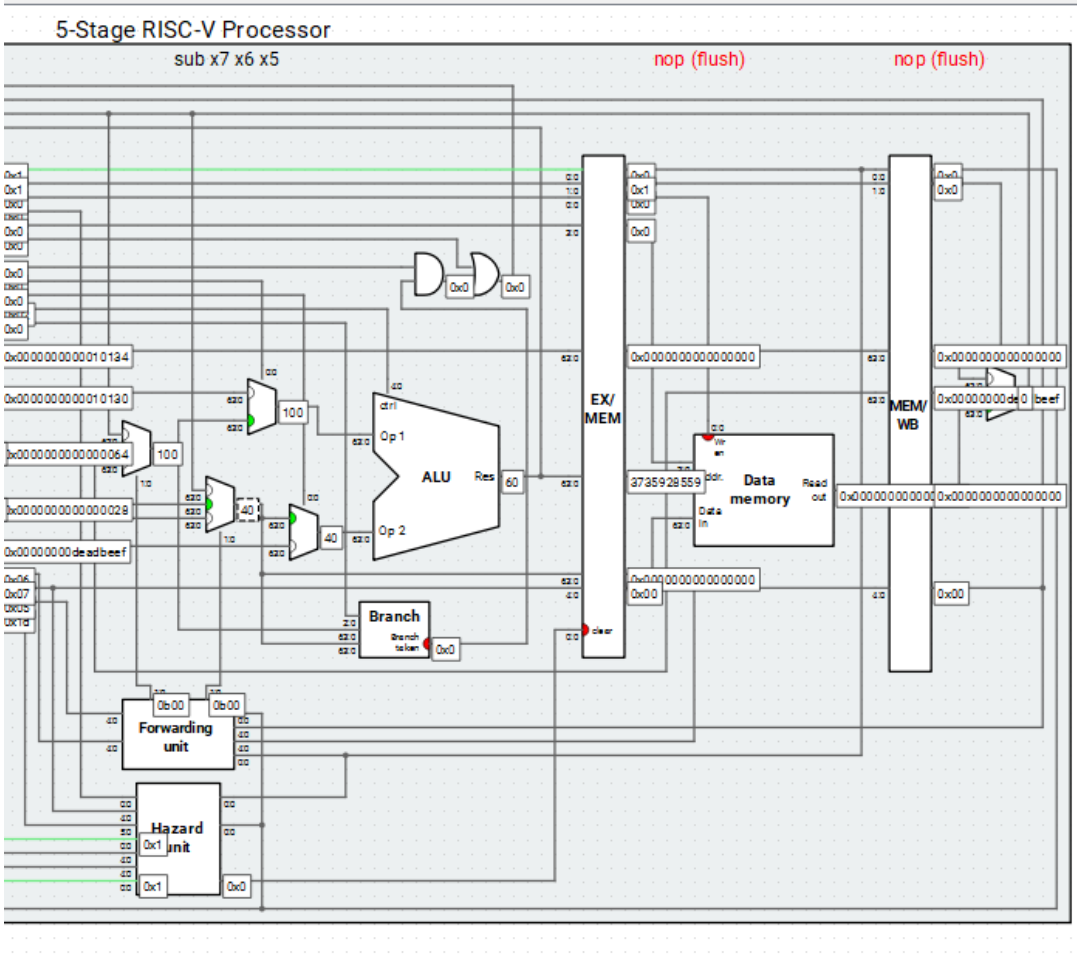
```
73  Loop:
74      blt t0, t1, 1f      # Break if lo < hi
75      j 2f                # Jump to EndLoop if lo >= hi
76
77  1:
78      # Use arithmetic shift right to divide by 2
79      # Calculate mid = lo + (hi - lo) / 2
80      sub t2, t1, t0
```

When blt is confirmed, and the branch is needed, the following two instructions must be nops and Ripes sets all control signals in ID/EX and IF/ID to 0 to achieve this. In this example, t0(x5) = 40,t1(x6) = 0.But t1 is actually 100 so that branch will happen, since addi x6 x10 0 and blt x5 x6 8 has RAW dependency, so a forwarding from MEM to EX is needed.




5-Stage RISC-V Processor
nop (flush)
nop (flush)
blt x5 x6 8
addi x6 x10 0
IFID
Control
Decode
Registers
IDEX
Imm.
ALU
Branch
EX/
MEM
Data
memory
MEM/
WB
Forwarding
unit
Hazard
unit


5-Stage RISC-V Processor
nop (flush)
nop (flush)
blt x5 x6 8
IDEX
ALU
Branch
EX/
MEM
Data
memory
MEM/
WB
Forwarding
unit
Hazard
unit


5-Stage RISC-V Processor
sub x7 x6 x5
nop (flush)
nop (flush)
EX/
MEM
MEM/
WB
ALU
Op 1
Op 2
Res
ctrl
Data
memory
Read
out
Data
in
Branch
Forwarding
unit
Hazard
unit
0x0
0x1
0x00
0b00
100
40
60
beef