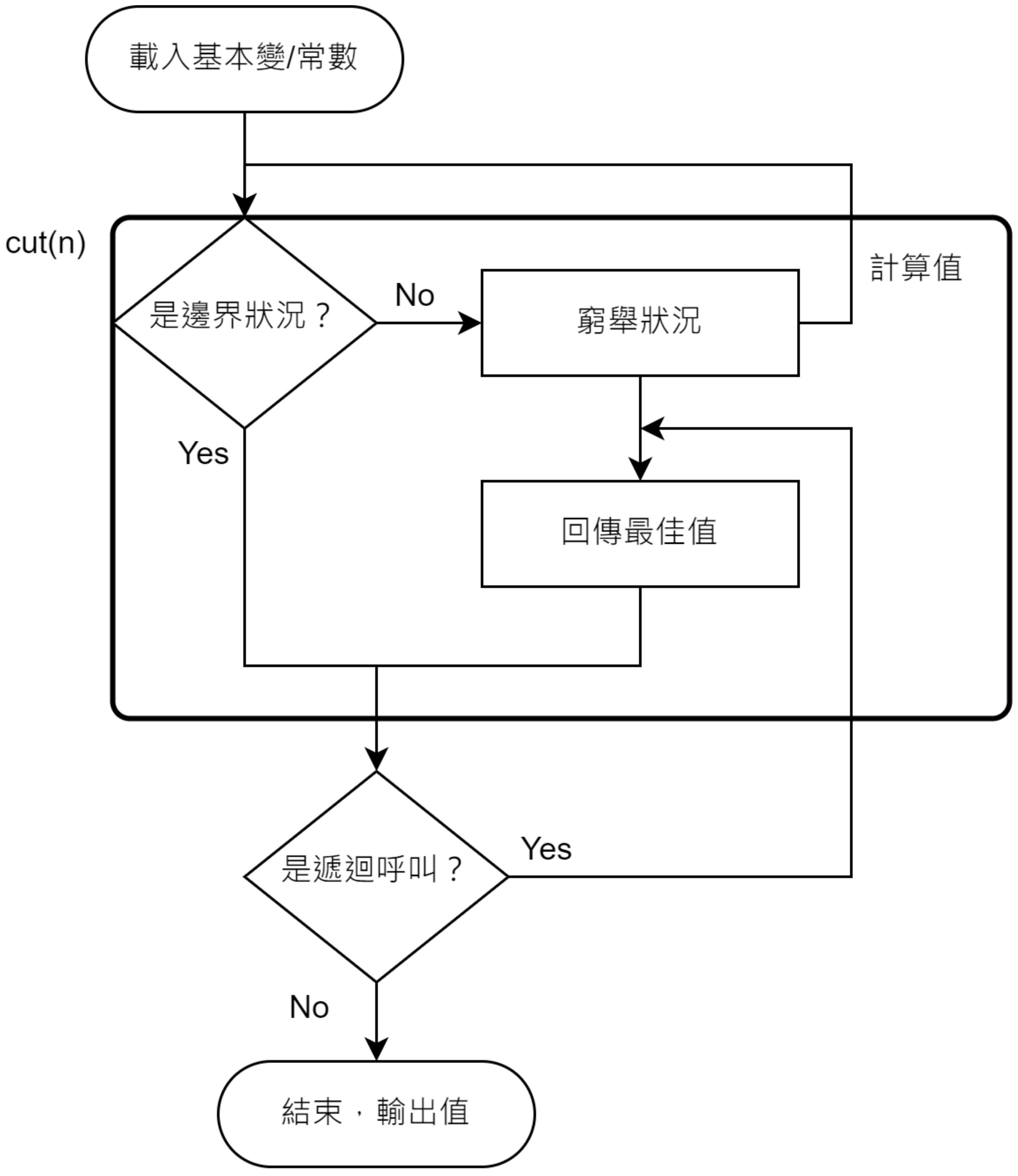
HW5

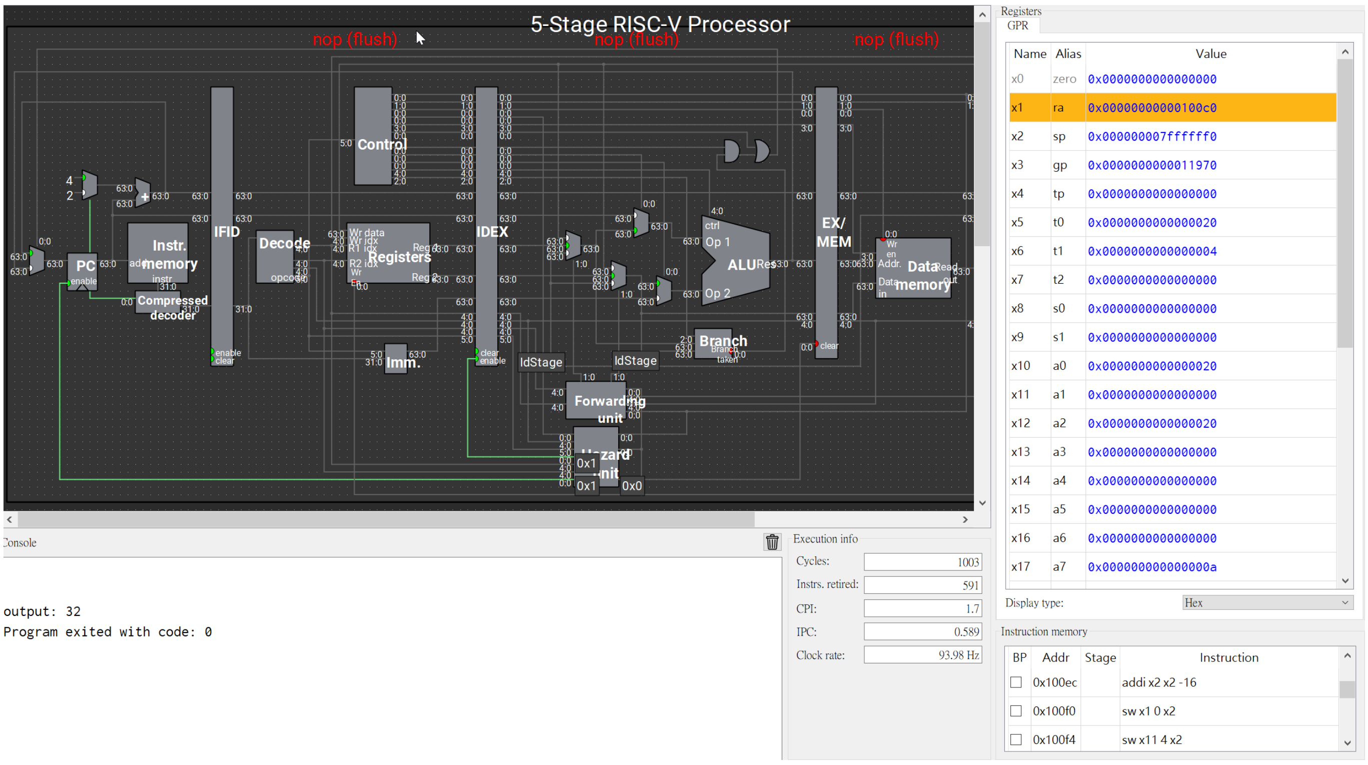
> author: 111062272 蕭登鴻

1

Flowchart of the main program:



testcase1:



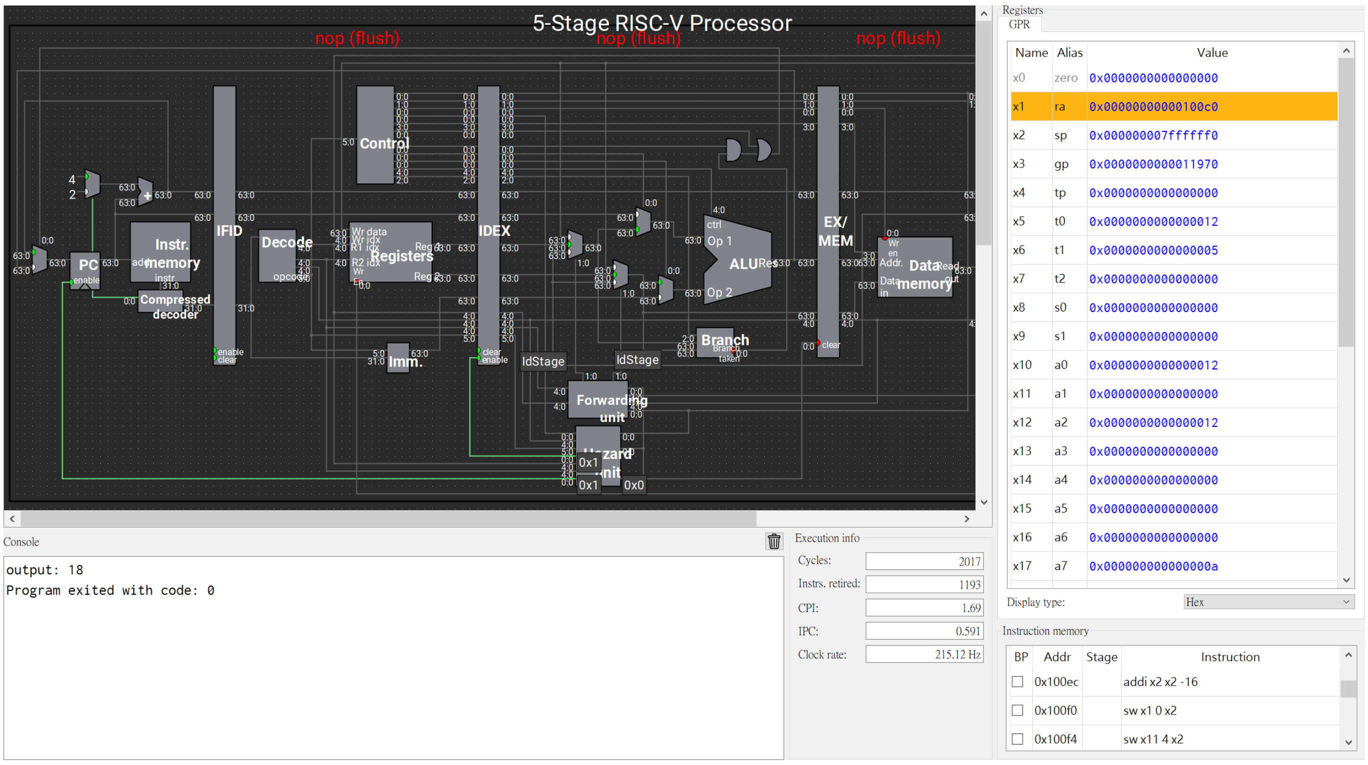
Name	Alias	Value
x0	zero	0x0000000000000000
x1	ra	0x00000000000100c0
x2	sp	0x000000007ffffff0
x3	gp	0x0000000000011970
x4	tp	0x0000000000000000
x5	t0	0x0000000000000020
x6	t1	0x0000000000000004
x7	t2	0x0000000000000000
x8	s0	0x0000000000000000
x9	s1	0x0000000000000000
x10	a0	0x0000000000000020
x11	a1	0x0000000000000000
x12	a2	0x0000000000000020
x13	a3	0x0000000000000000
x14	a4	0x0000000000000000
x15	a5	0x0000000000000000
x16	a6	0x0000000000000000
x17	a7	0x000000000000000a

```
output: 32
Program exited with code: 0
```

Cycles:	1003
Instrs. retired:	591
CPI:	1.7
IPC:	0.589
Clock rate:	93.98 Hz

BP	Addr	Stage	Instruction
☐	0x100ec		addi x2 x2 -16
☐	0x100f0		sw x1 0 x2
☐	0x100f4		sw x11 4 x2

testcase2:



Name	Alias	Value
x0	zero	0x0000000000000000
x1	ra	0x00000000000100c0
x2	sp	0x000000007ffffff0
x3	gp	0x0000000000011970
x4	tp	0x0000000000000000
x5	t0	0x0000000000000012
x6	t1	0x0000000000000005
x7	t2	0x0000000000000000
x8	s0	0x0000000000000000
x9	s1	0x0000000000000000
x10	a0	0x0000000000000012
x11	a1	0x0000000000000000
x12	a2	0x0000000000000012
x13	a3	0x0000000000000000
x14	a4	0x0000000000000000
x15	a5	0x0000000000000000
x16	a6	0x0000000000000000
x17	a7	0x000000000000000a

```
output: 18
Program exited with code: 0
```

Cycles:	2017
Instrs. retired:	1193
CPI:	1.69
IPC:	0.591
Clock rate:	215.12 Hz

BP	Addr	Stage	Instruction
☐	0x100ec		addi x2 x2 -16
☐	0x100f0		sw x1 0 x2
☐	0x100f4		sw x11 4 x2

2

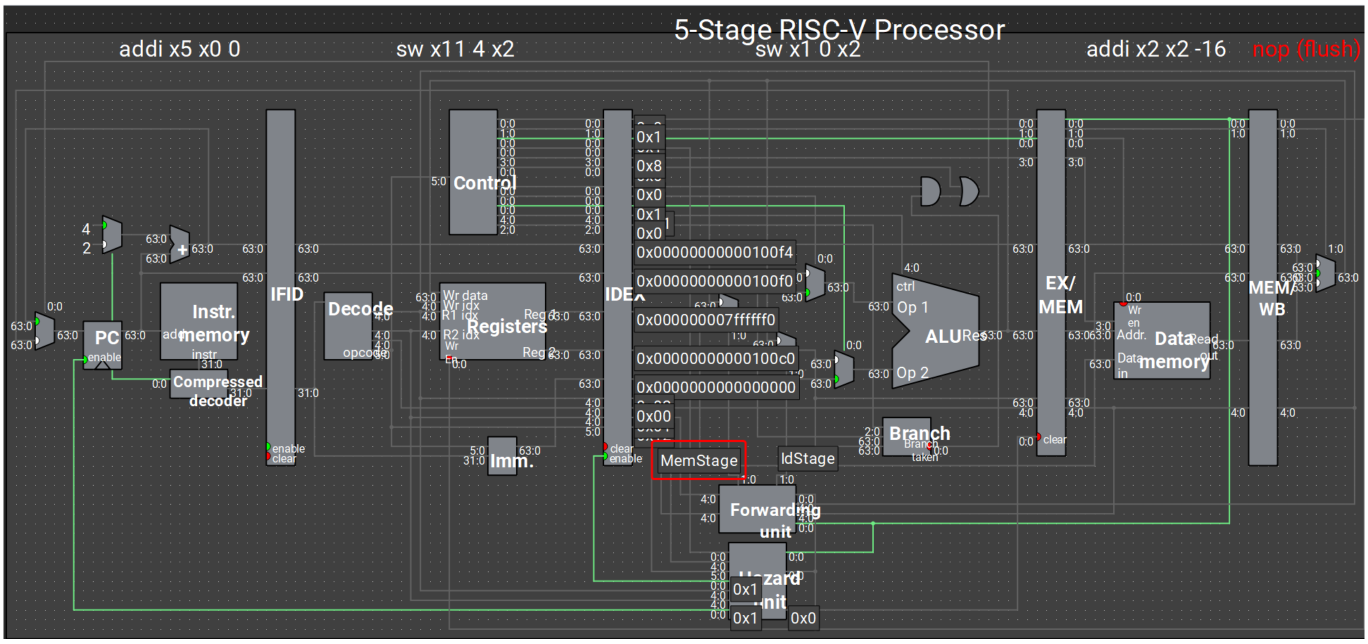
Type 1 & 2

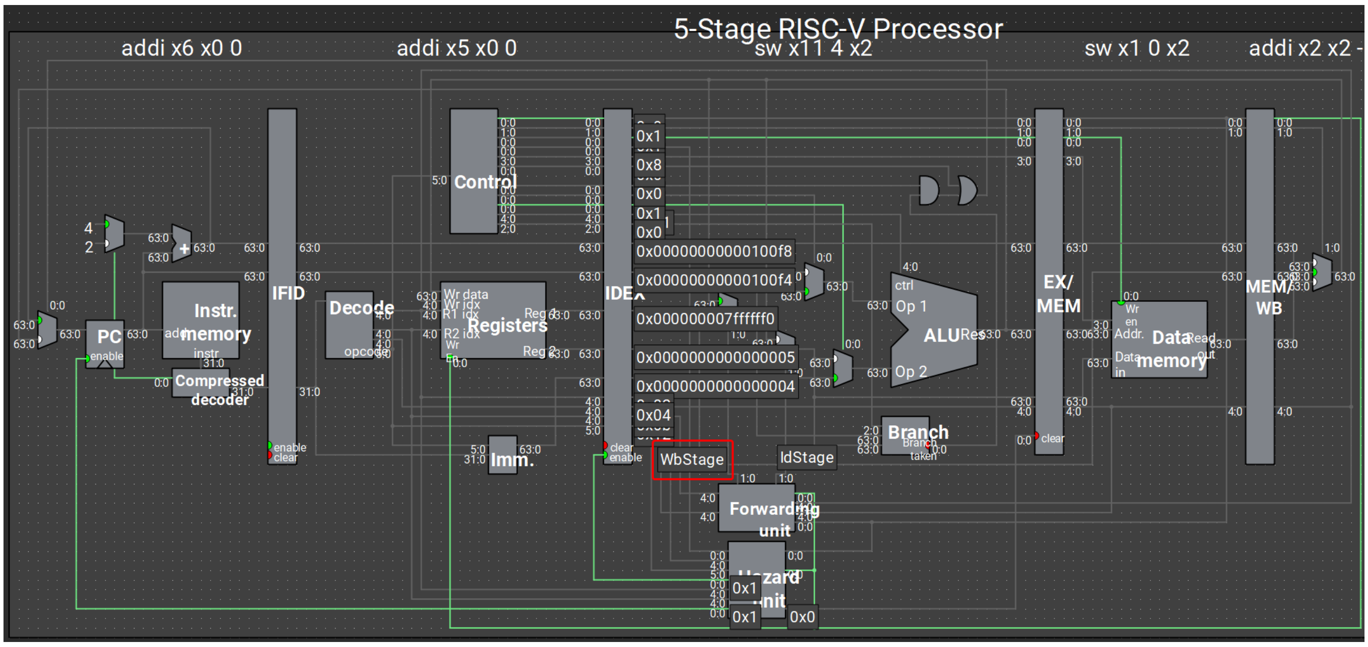
The code segment (line 58~60)

```
addi sp, sp, -16
sw ra, 0(sp)
sw a1, 4(sp)
```

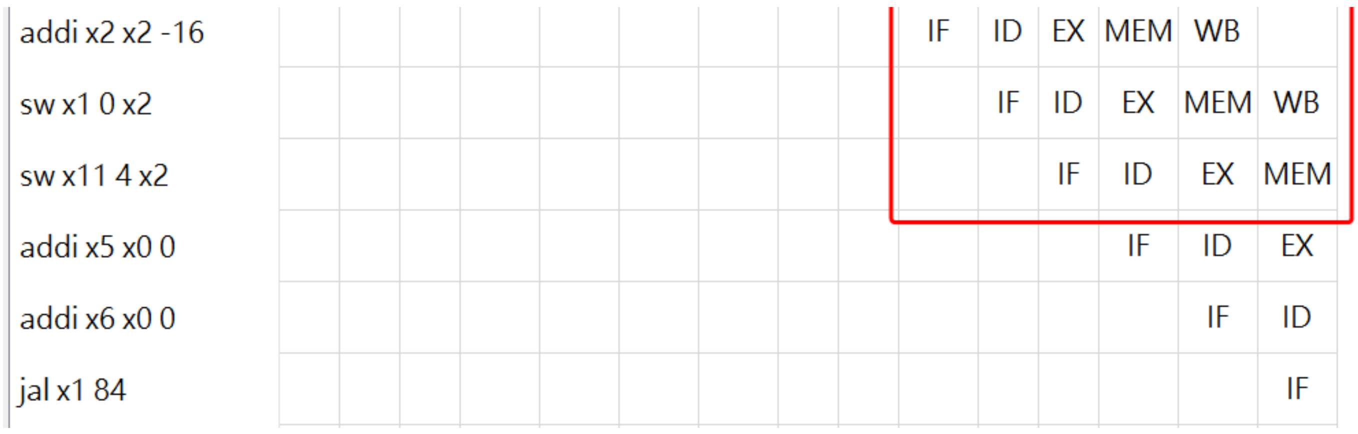
contains both type 1 & 2 dependency (sp).

Ripes solve them by forwarding in the `MEM` & `WB` stage, as illustrated:





Pipeline result:



addi x2 x2 -16										IF	ID	EX	MEM	WB	
sw x1 0 x2											IF	ID	EX	MEM	WB
sw x11 4 x2												IF	ID	EX	MEM
addi x5 x0 0													IF	ID	EX
addi x6 x0 0														IF	ID
jal x1 84															IF

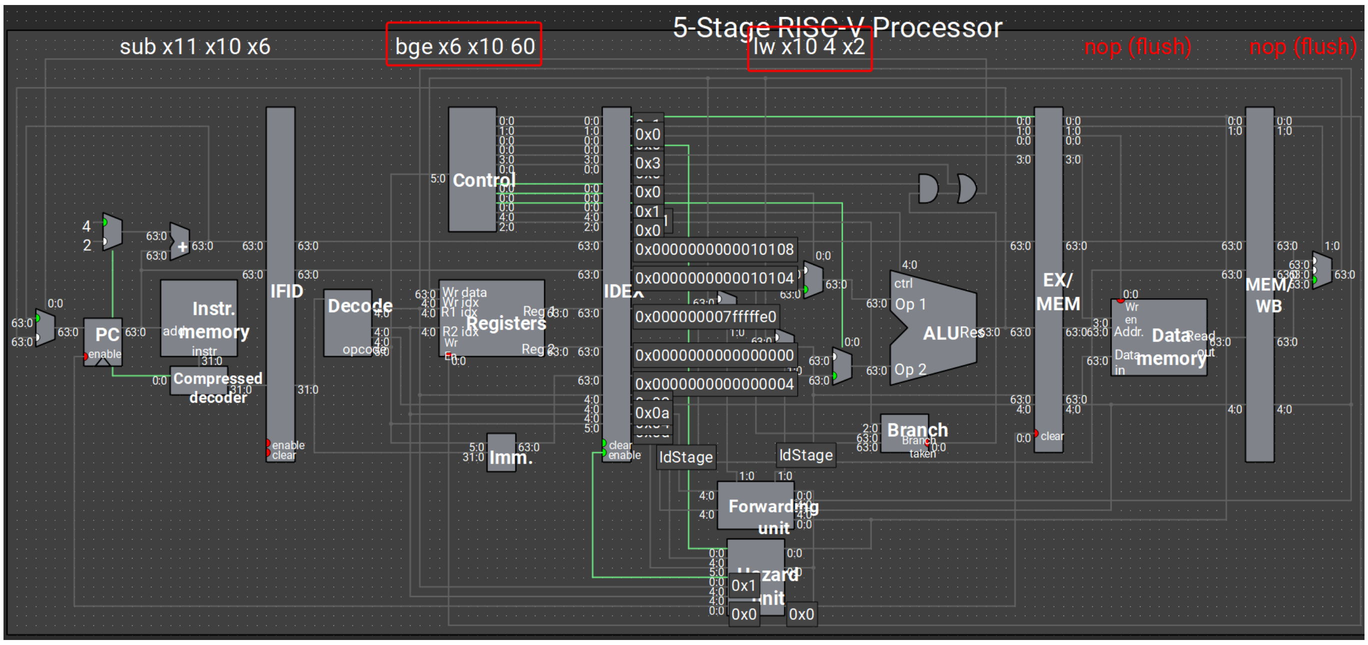
Type 3

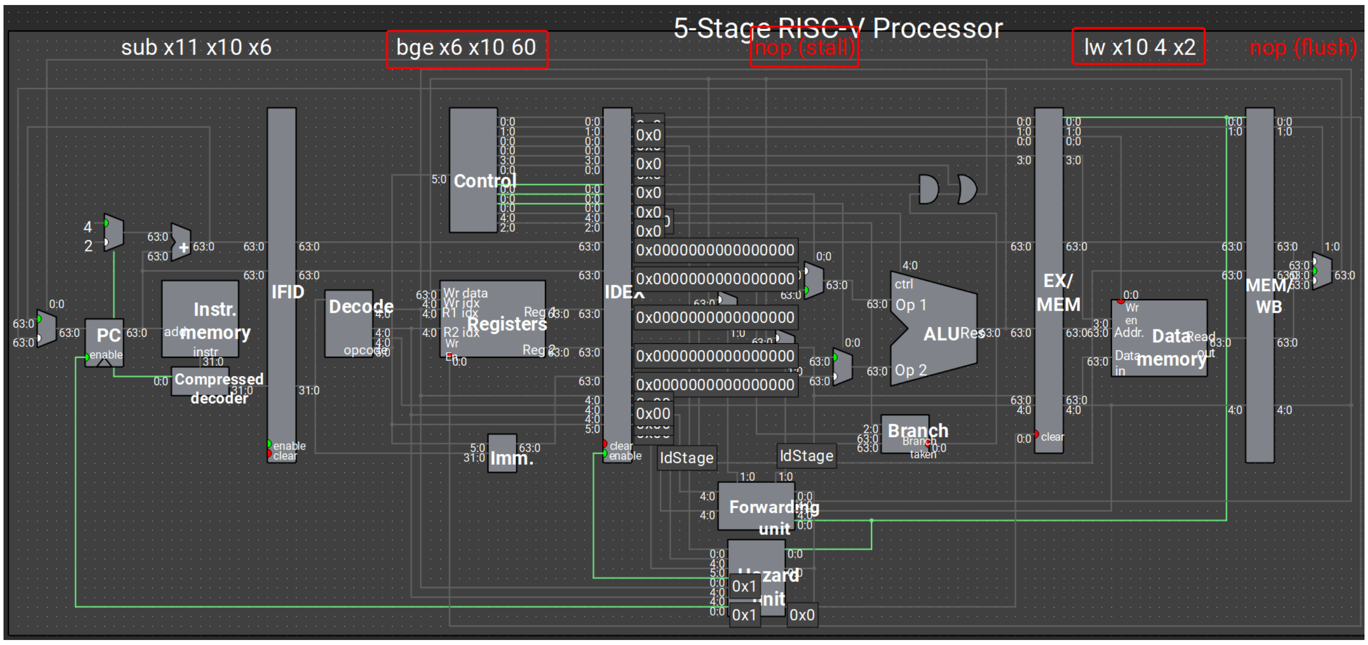
The code segment (line 67, 68)

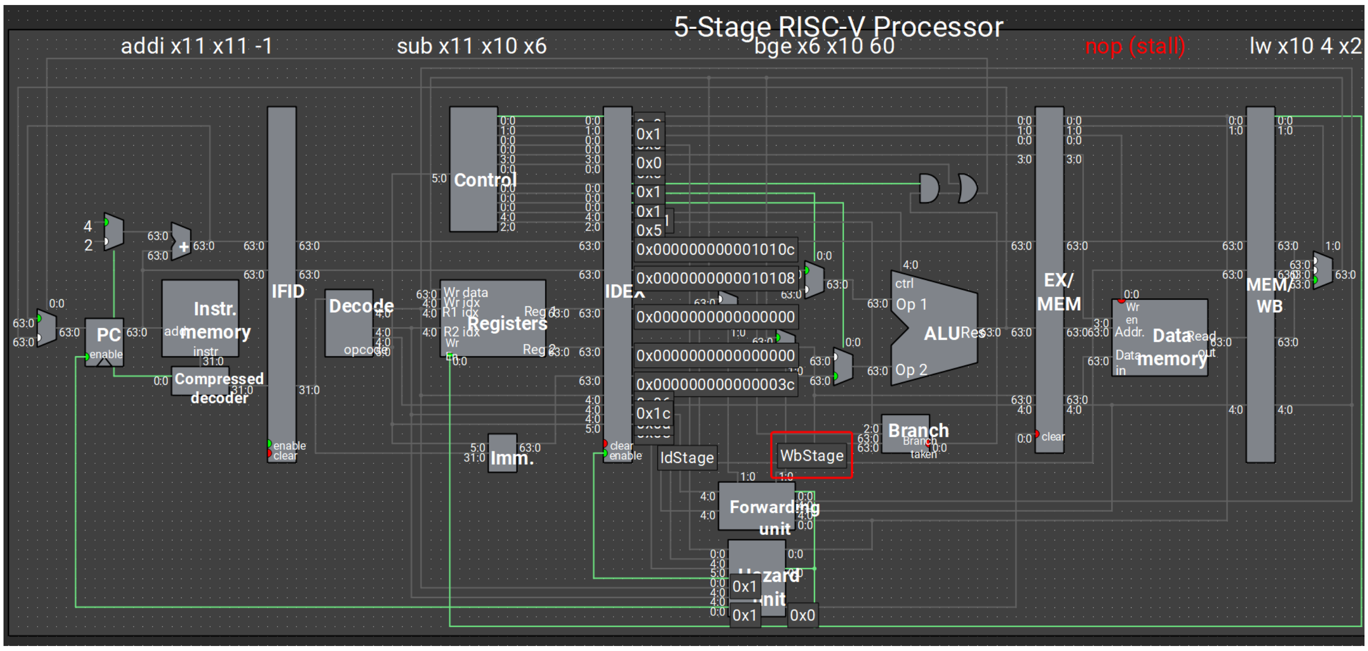
```
lw a0, 4(sp)
bge t1, a0, loop_end
```

contains a type 3 dependency (a0).

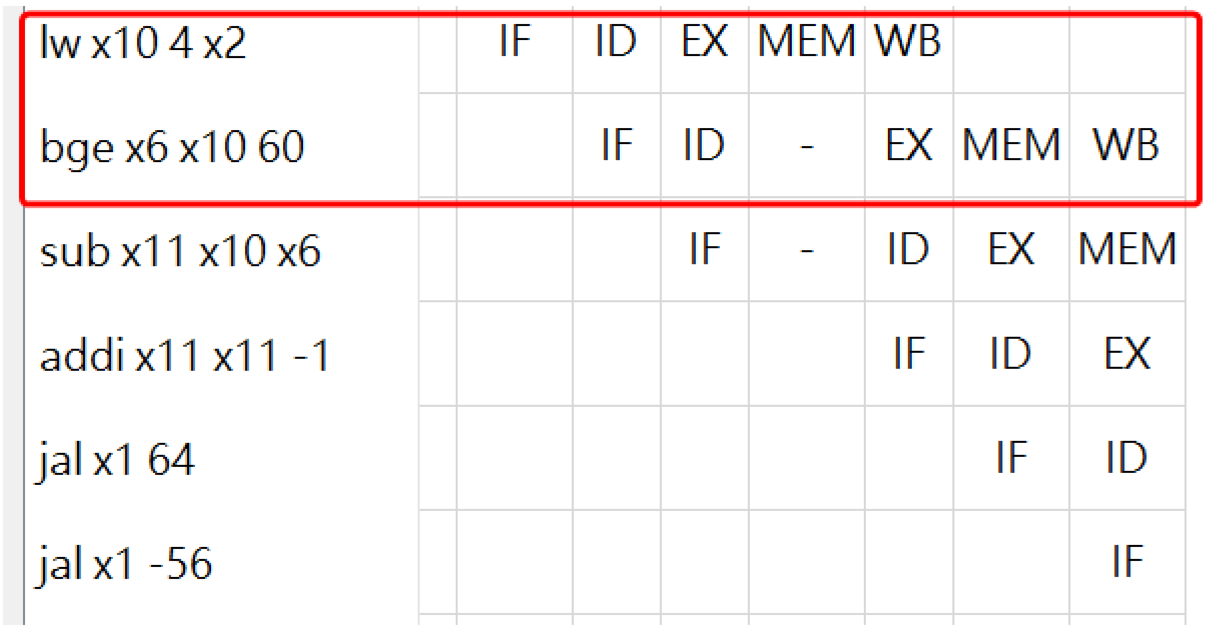
Ripes solve it by first inserting a nop(stall), so as to execute a `WB` forwarding in the next cycle.







Pipeline result:



lw x10 4 x2		IF	ID	EX	MEM	WB		
bge x6 x10 60			IF	ID	-	EX	MEM	WB
sub x11 x10 x6				IF	-	ID	EX	MEM
addi x11 x11 -1						IF	ID	EX
jal x1 64							IF	ID
jal x1 -56								IF

Type 4

After thorough search, example of type 4 dependency can't be identified in my assembly code. However, we can use the same code segment in type 3 with little modification for illustration:

Consider the code segment (67, 68, 71, with comments omitted), and replace line 68 with code irrelevant to register `a0`:

```
lw a0, 4(sp)
... # some code irrelevant to a0
sub a1, a0, t1
```

This is a type 4 dependency (a0).

The way of solving it is similar in type 3, with no `nop` inserted. A `WB` forwarding can be executed directly.

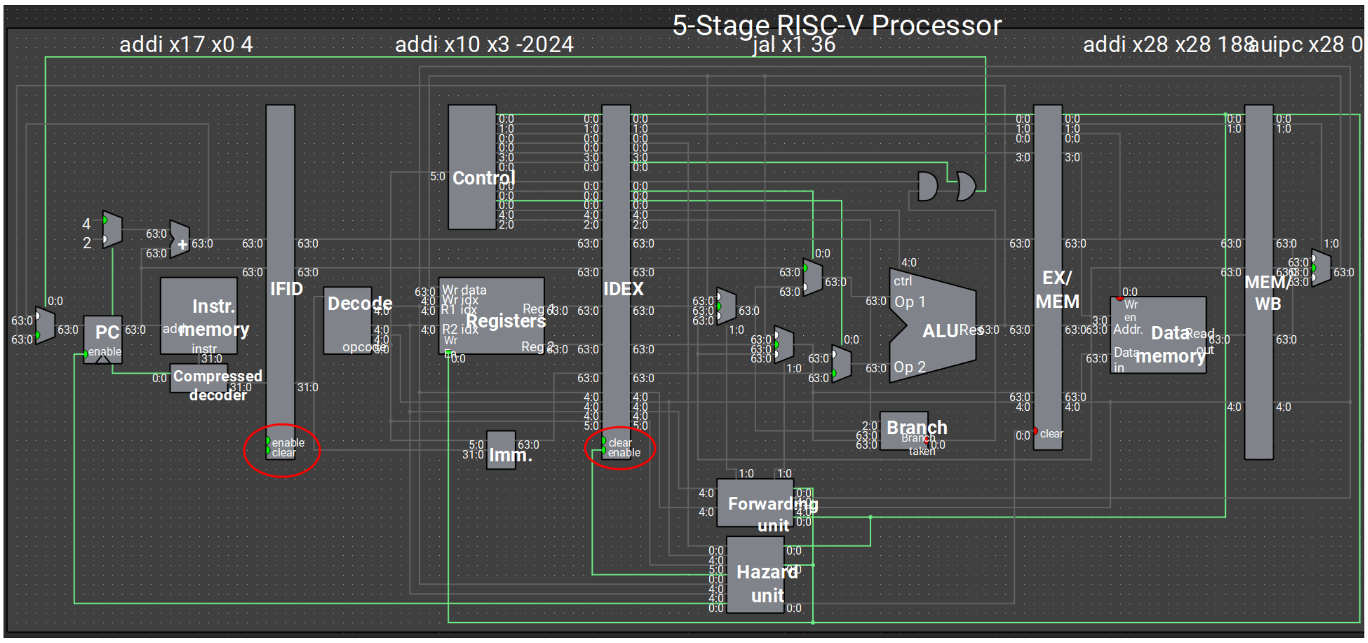
Type 5

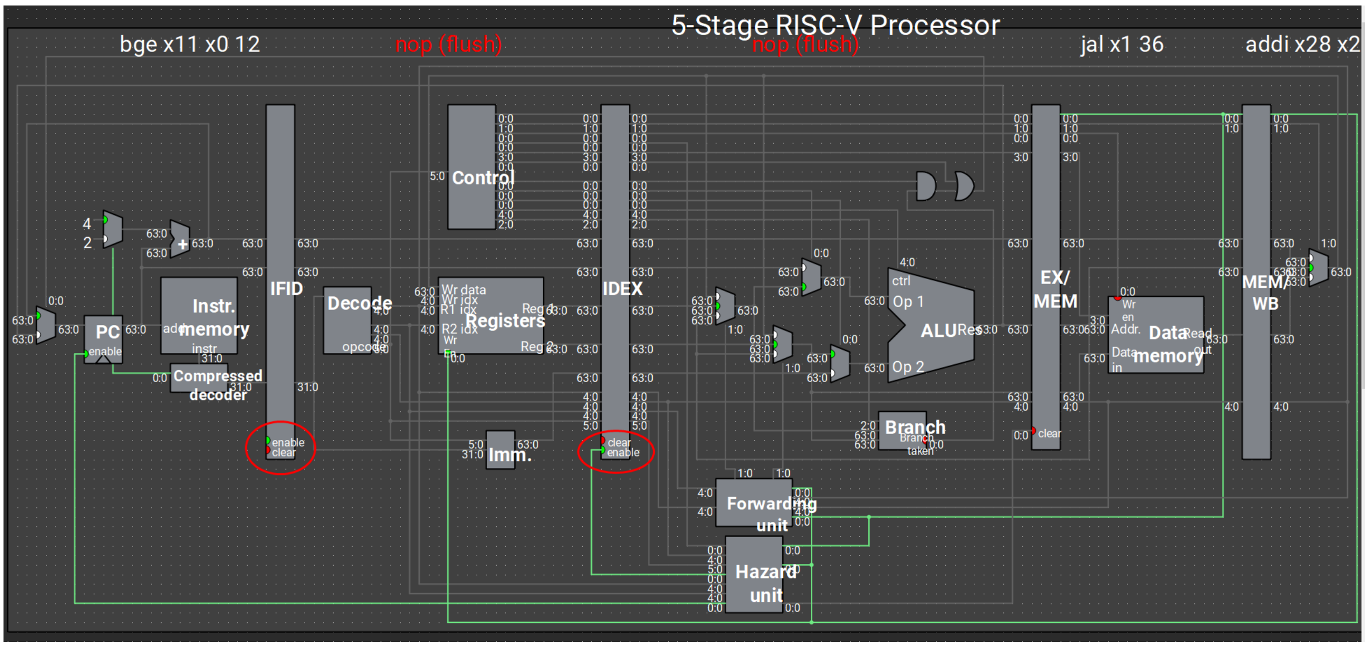
The code segment (line 76, 79, with comments omitted)

```
jal ra, loadSS
mv t4, t1
```

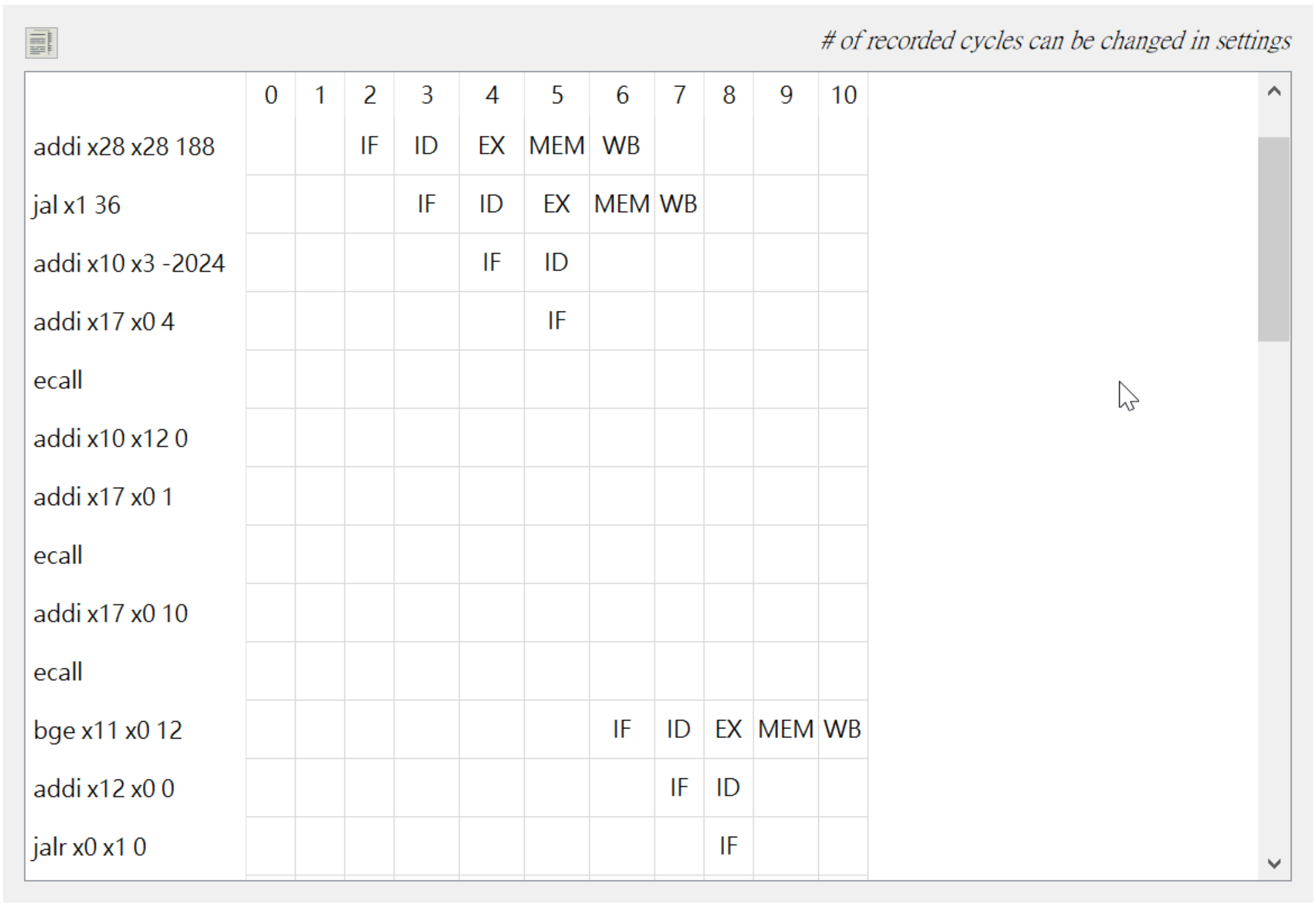
contains a type 5 dependency.

Ripes solve it by activating the `clear` signal of the `IF/ID` & `ID/EX` register file, which resulted in two `nop` (s).





Pipeline result:



# of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10
addi x28 x28 188			IF	ID	EX	MEM	WB				
jal x1 36				IF	ID	EX	MEM	WB			
addi x10 x3 -2024					IF	ID					
addi x17 x0 4						IF					
ecall											
addi x10 x12 0											
addi x17 x0 1											
ecall											
addi x17 x0 10											
ecall											
bge x11 x0 12							IF	ID	EX	MEM	WB
addi x12 x0 0								IF	ID		
jalr x0 x1 0									IF		