**109 fall 計算機結構 PA3**

**電機 23 108011235 陳昭維**

## 1. Write an Assembly Program

The code is attached with the homework on eeclass, both 5 stage with and without supporting forwarding and hazard detection.

Example tested:

```
xx: .word  1, 200, 300, -40
yy: .word  5, 6, 7, 80
```

115 , 24

```
xx: .word  1, 2, 3, 4
yy: .word  5, 6, 7, 8
```

2 , 6

```
xx: .word  -93, 45, -33, -98
yy: .word  -11, -63, 58, -49
```

-44 , -16

Example tested (nop):

```
xx: .word  -93, 45, -33, -98
yy: .word  -11, -63, 58, -49
```

-44 , -16

```
xx: .word  -15, 44,-60,-68
yy: .word   71, -56, 31,-31
```

-24 , 3

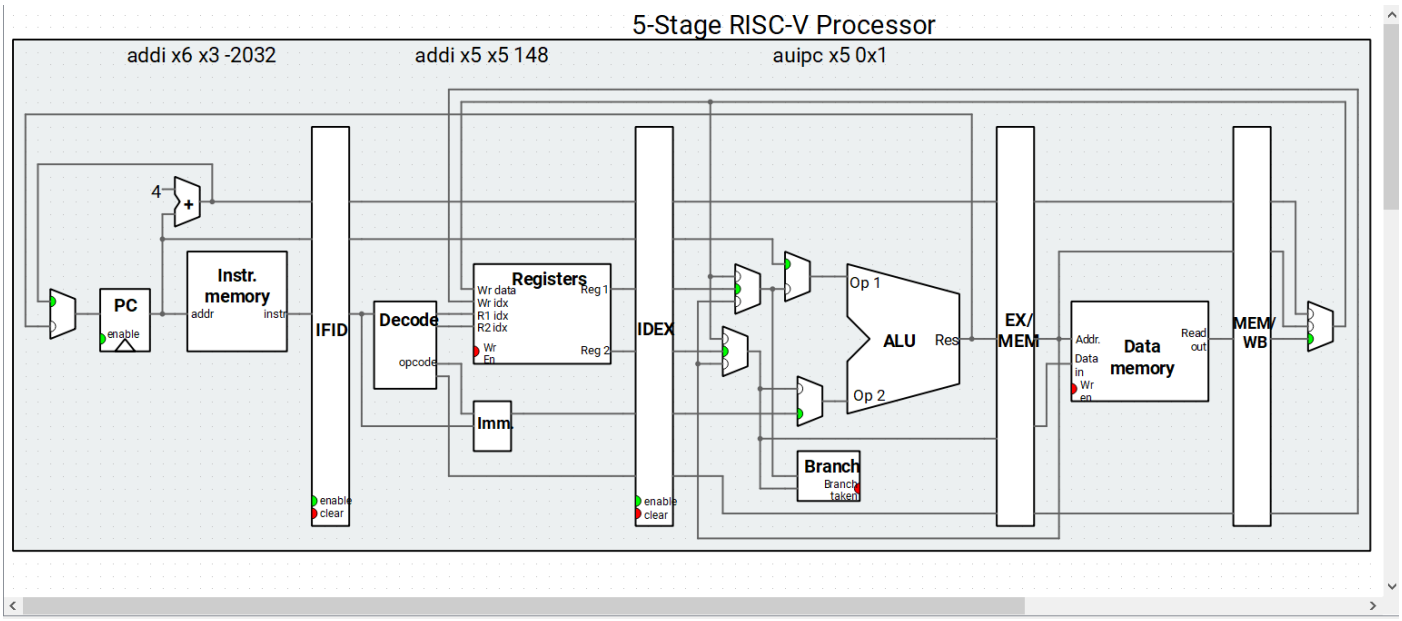
```
xx: .word  51,  71, 15, 90
yy: .word  40,  9,  96, -90
```

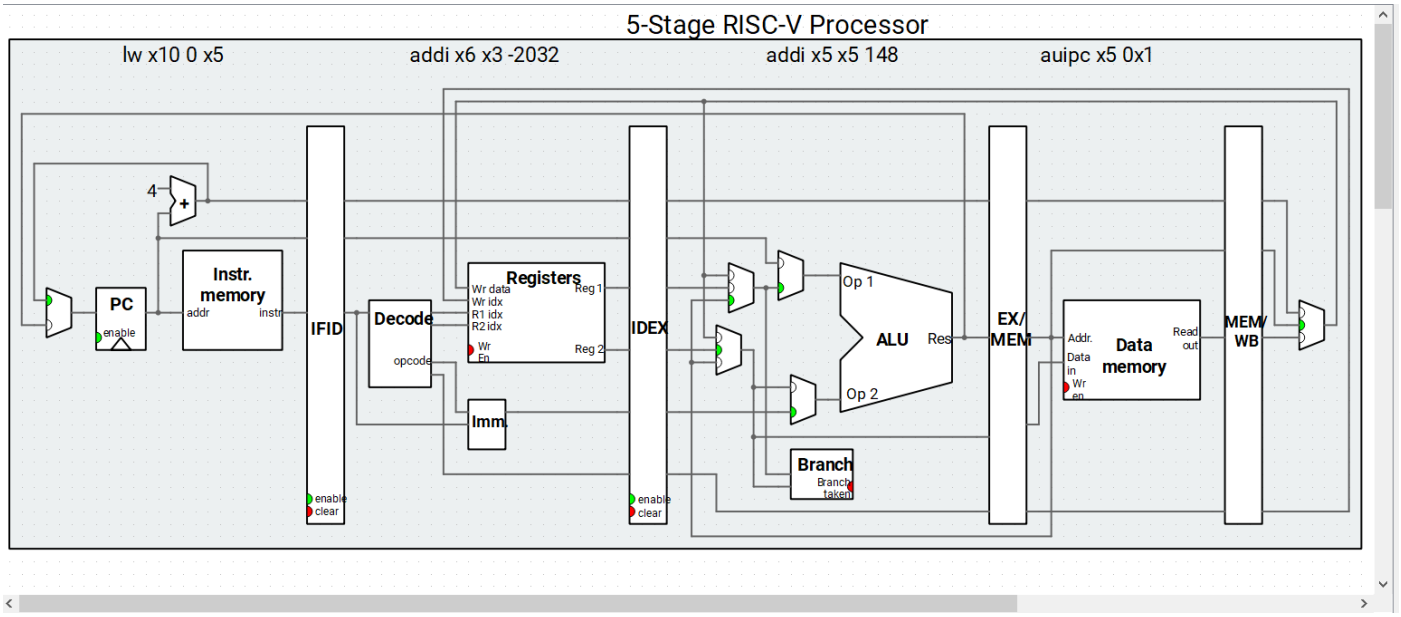
56 , 13

## 2. Simulate Pipeline

### *I type forwarding at following 1st instruction*

```
10074:      00001297        auipc x5 0x1                    ID
10078:      09428293        addi x5 x5 148                  IF
```



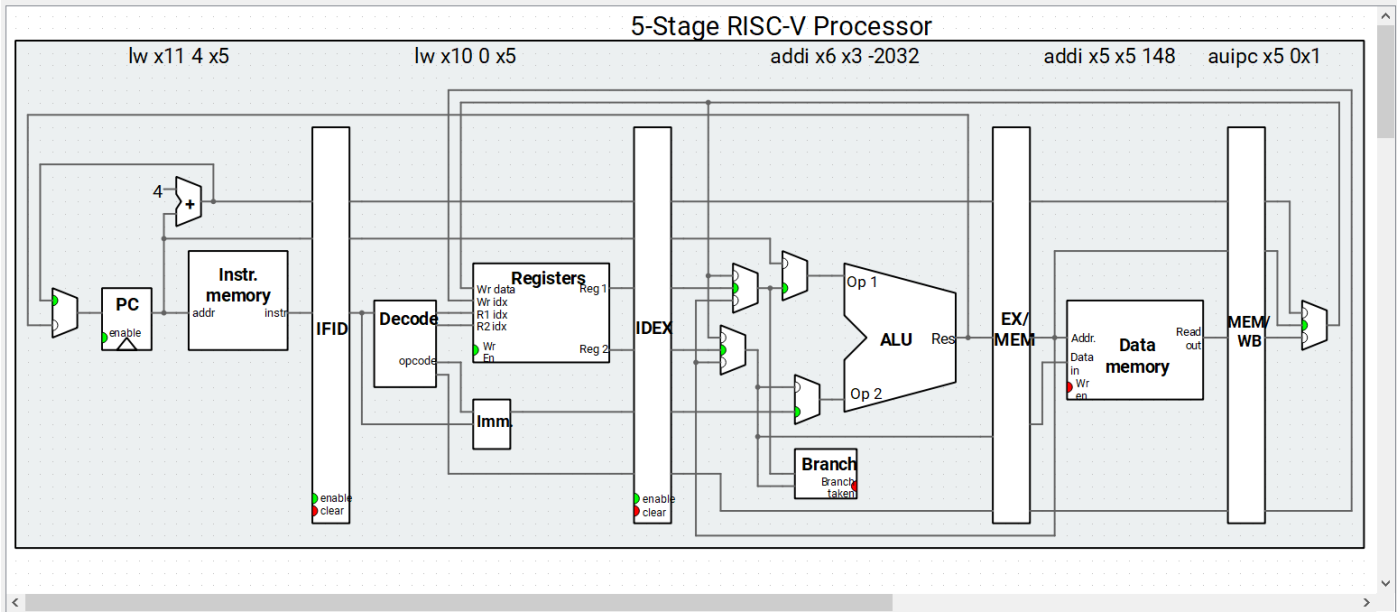

**Comment:**

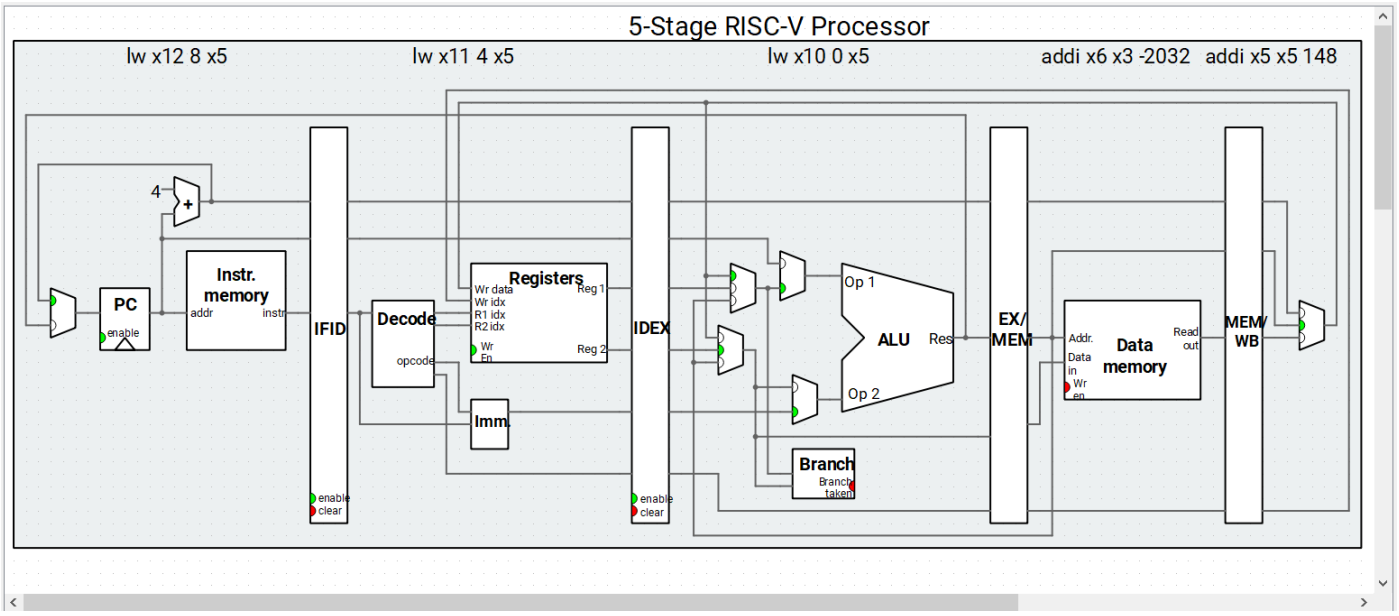
Forwarding happens between the cycle for the above stage, there is an EX/MEM forwarding for new value in register x5 to EX stage.

Forwarding is needed to let addition for the following addi instruction to us the correct value in x5.

## *Load forwarding at following 2nd instruction*

```
10078:    09428293    addi x5 x5 148        EX
1007c:    81018313    addi x6 x3 -2032      ID
10080:    0002a503    lw x10 0 x5           IF
```



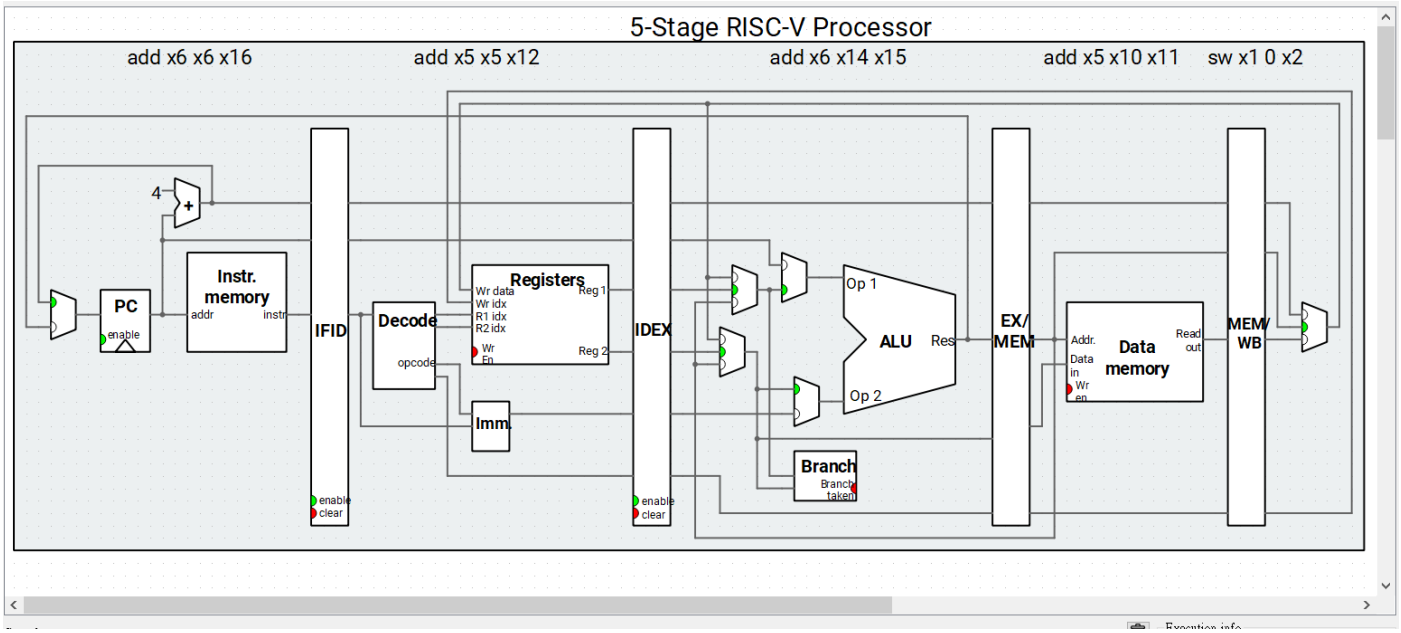

**Comment:**

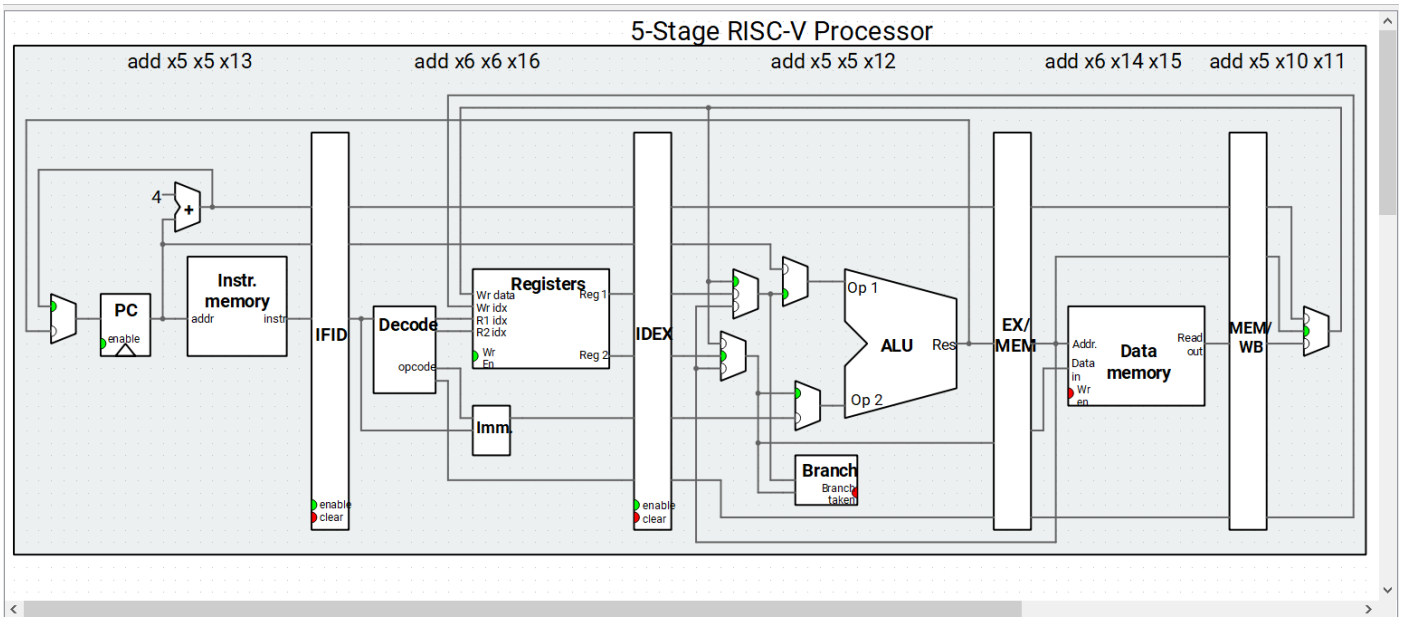
Forwarding happens between the cycle for the above stage, there is a MEM/WB forwarding of new value in register x5 to EX stage.

Forwarding is needed to let the 2nd following lw instruction to us the correct value in x5.

## *R type forwarding at following 2nd instruction*

```
100d4:    00b502b3    add x5 x10 x11    EX
100d8:    00f70333    add x6 x14 x15    ID
100dc:    00c282b3    add x5 x5 x12     IF
```



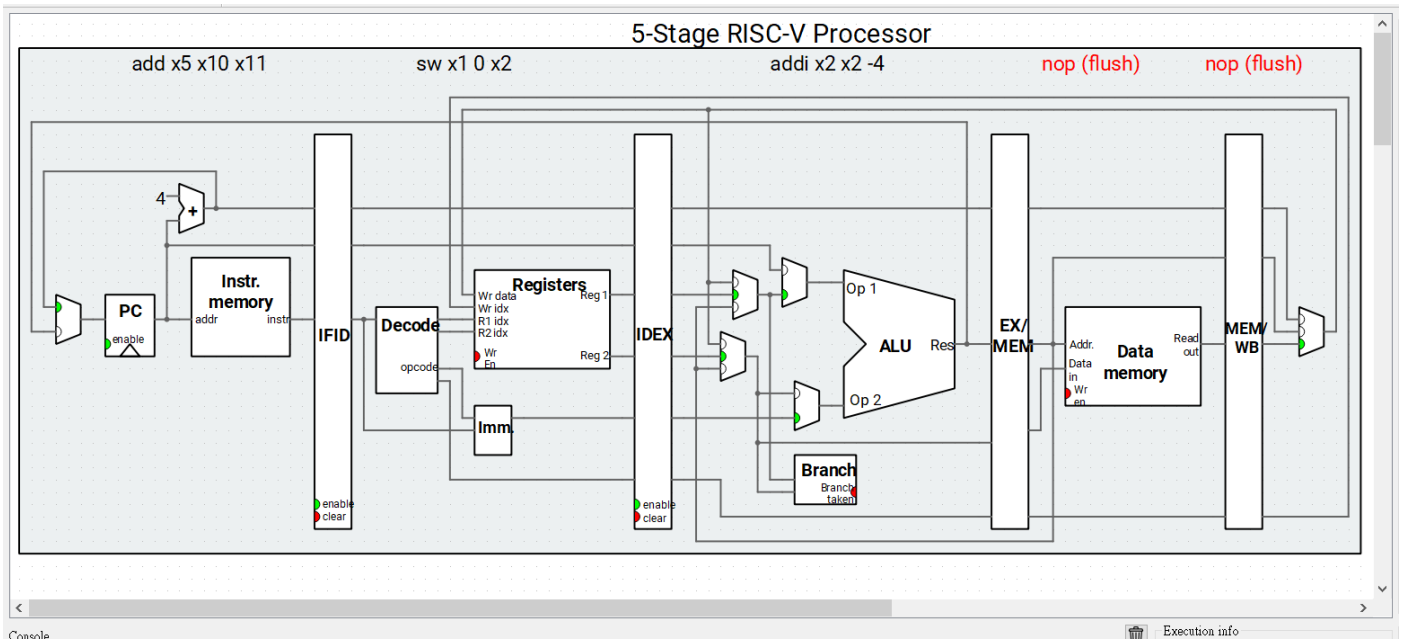

**Comment:**

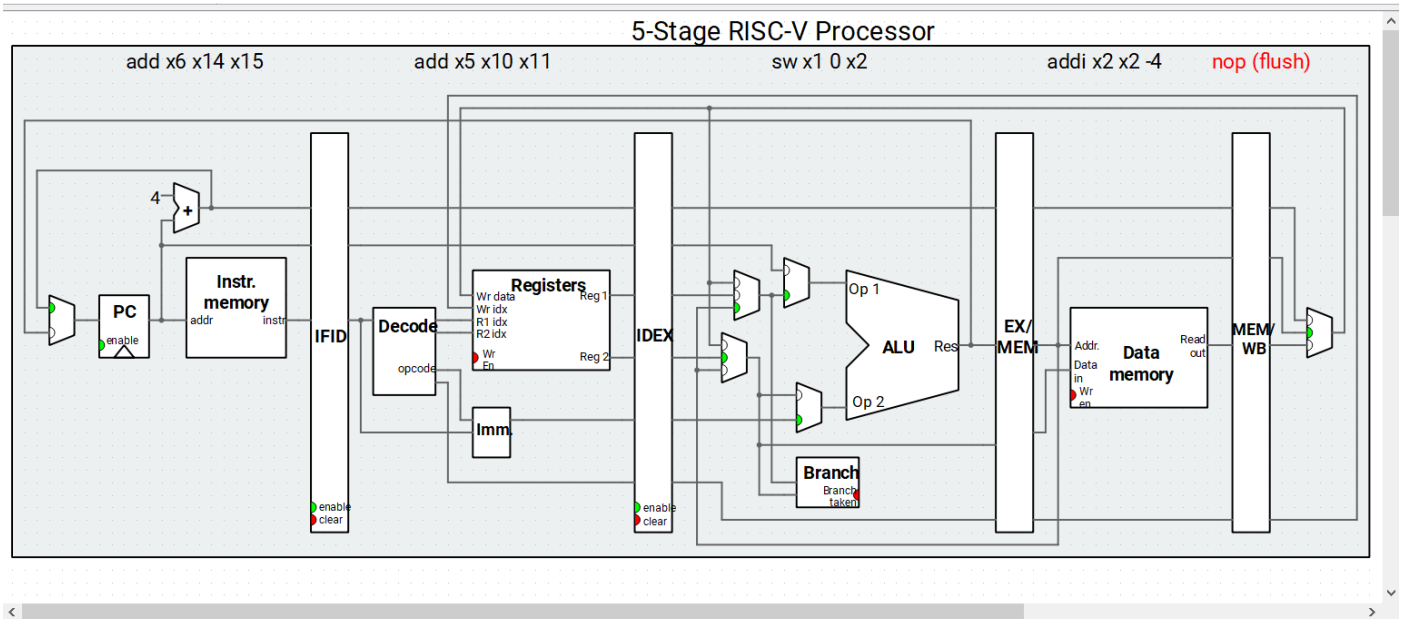
Forwarding happens between the cycle for the above stage, there is a MEM/WB forwarding of new value in register x5 to EX stage.

The forwarding is needed for the 2nd following instruction add to use the correct value in x5.

## *Store type forwarding at following 2nd instruction*

```
100cc:    ffc10113    addi x2 x2 -4    ID
100d0:    00112023    sw x1 0 x2       IF
```



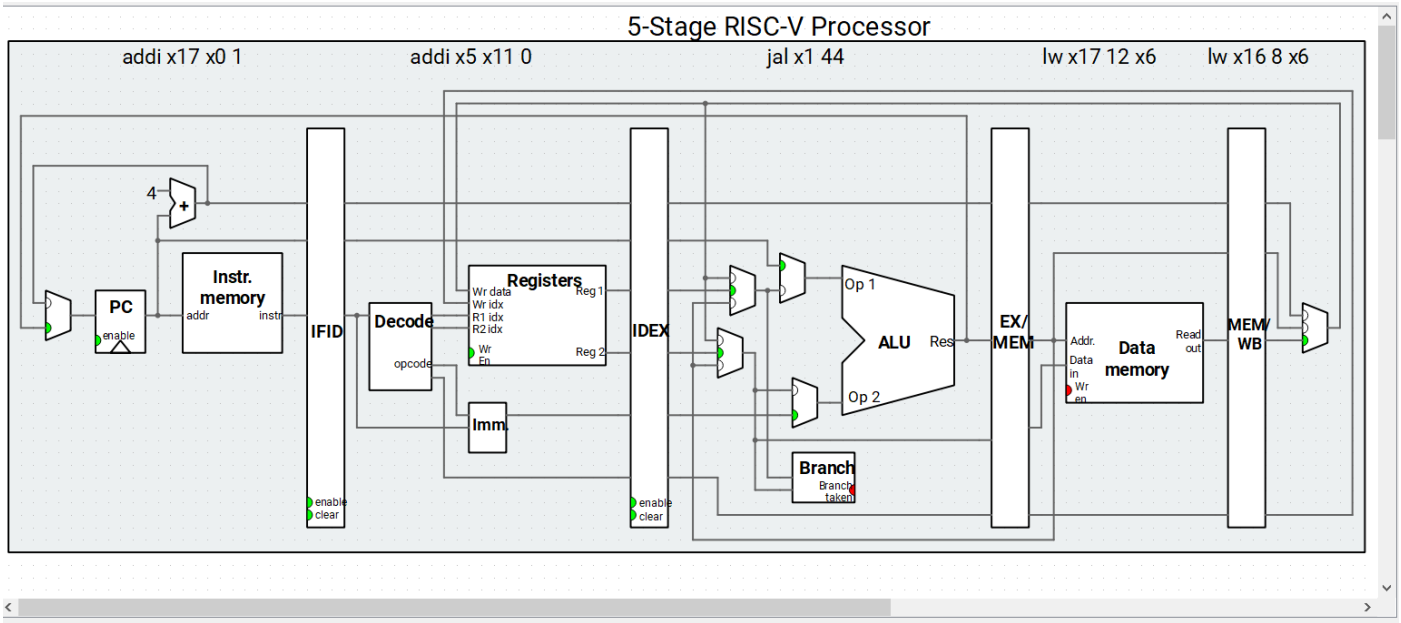

**Comment:**

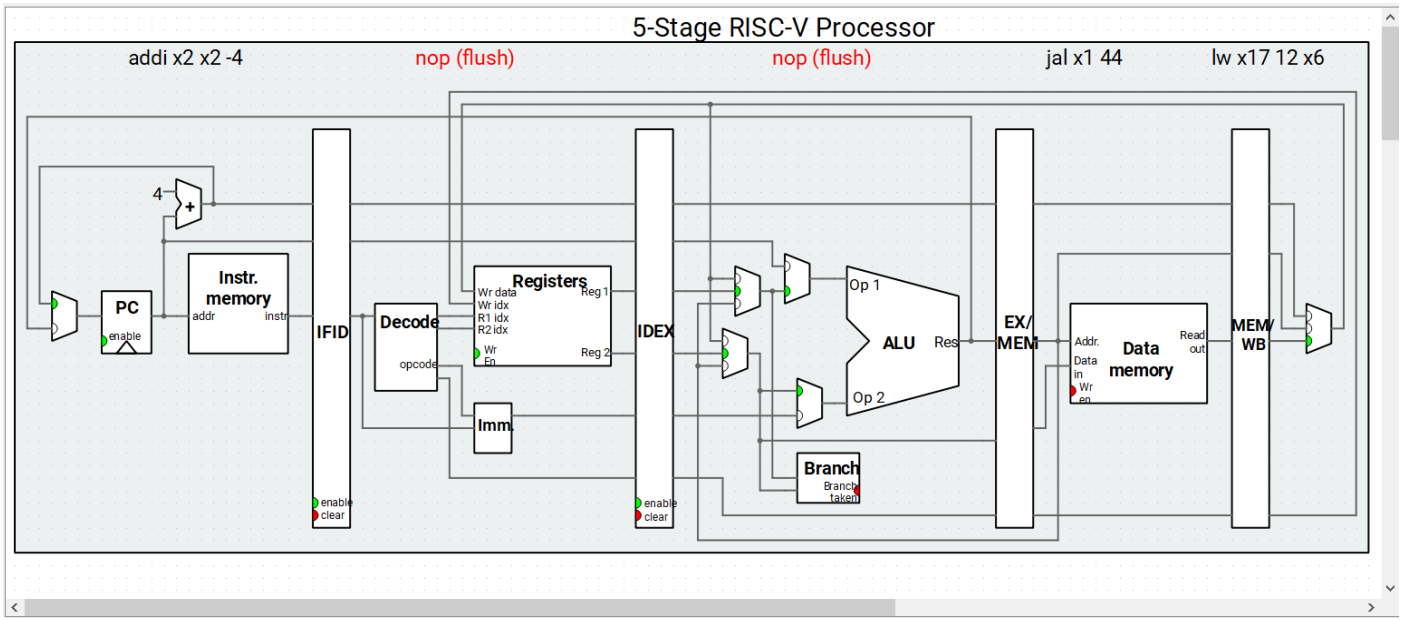
The forwarding is between the next cycle for above stage, there is an EX/MEM forwarding of new address of the stack pointer to EX stage.

The forwarding is needed for the following instruction sw to use the correct stack pointer address.

## *Jump and link Flush*

```
100a0:      02c000ef        jal x1 44                    EX
100a4:      00058293        addi x5 x11 0                ID
100a8:      00100893        addi x17 x0 1                IF
```




**Comment:**

The flush is needed because the jump and link jumps to the new function, thus the loaded instruction before entering the new function needs flush.

## 3. 5-stage pipeline without forwarding and or hazard detection

*The left code is the original code and the right code is the modified code*

```
la t0, xx
la t1, yy
lw a0, 0(t0)
```

```
auipc t0, 0x1
nop
nop
# 2 nop for the co
addi t0, t0, 212
# 1 nop is needed
#   after the "add
nop
la t1, yy
lw a0, 0(t0)
```

The load address instruction is break downed into auipc and addi instruction since there is a load instruction at following 2nd instruction, two nop needs to be put in between. And after that the instruction lw a0, 0(t0) needs the new value of t0 after the instruction t0, t0, 212, so one more nop need to be put in between.

```
# addition of 4 values
# R-type write and read at following 2nd instruction
add t0, a0, a1
add t1, a4, a5
# R-type write and read at following 2nd instrucion
add t0, t0, a2
add t1, t1, a6
add t0, t0, a3
add t1, t1, a7
```

```
# addition of 4 values
add t0, a0, a1
add t1, a4, a5

nop  #newly added
add t0, t0, a2
add t1, t1, a6
nop  #newly added
add t0, t0, a3
add t1, t1, a7
```

Since there is data dependency of add instruction after at 2nd following of the first/second and the third/forth line. So nop first is inserted to the third line and fifth line, then for add t1, a4, a5 and add t1, t1, a6 nop doesn't need to be inserted because the previous nop inserted makes it able to catch up write back then read.

```
addi t2, x0, 4
div a0, t0, t2
div a1, t1, t2
```

```
addi t2, x0, 4
nop #newly added
nop #newly added
div a0, t0, t2
div a1, t1, t2
```

Since there is data dependency of the div instruction after the first line addi instruction, so 2 nop has to be inserted to catch up the write back then read.

```
li a7, 1
nop
nop
nop
ecall
```

Also, for my code, ecall needs 3 nop after the instruction li, a7, 1 for it to work without hazard.

```
# store returned address
addi sp, sp, -4
sw ra, 0(sp)
```

The sw after the addi instruction doesn’t need nop but still can works fine.