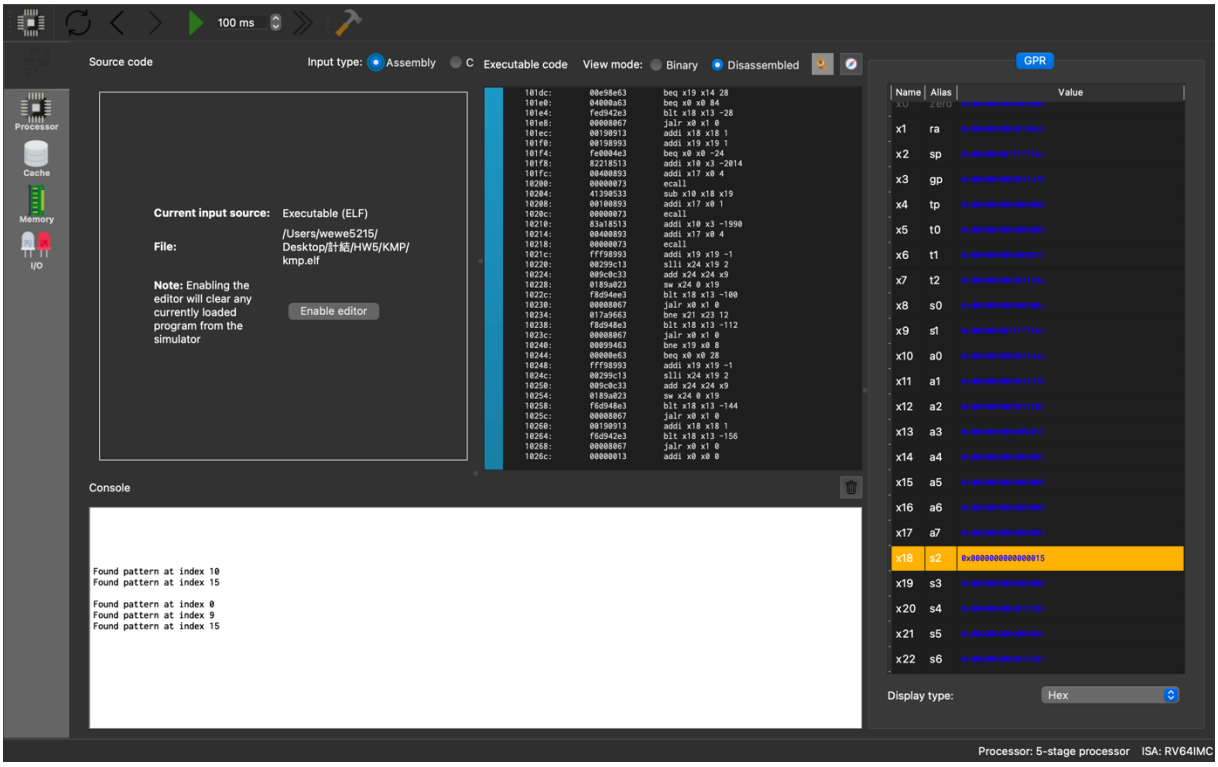
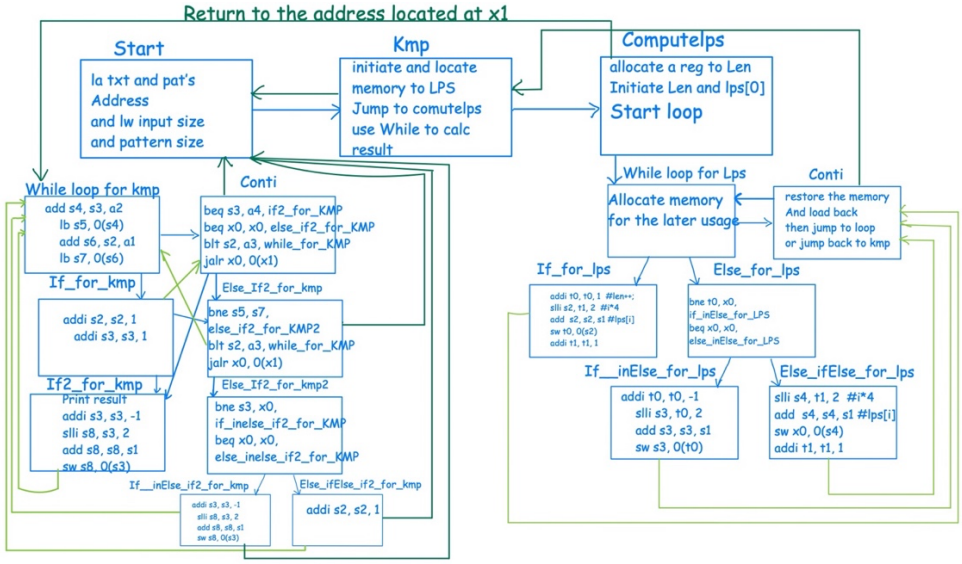
1. Assembly coding
   a. Evidence of correct results



   b. Flow chat



2. Hazards in my code

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

```
51    KMP:
52    slli s0, a4, 2 #patternSize
53    sub sp, sp, s0
54    add s1, sp, x0 #lps's addr
55
```

image:a-1

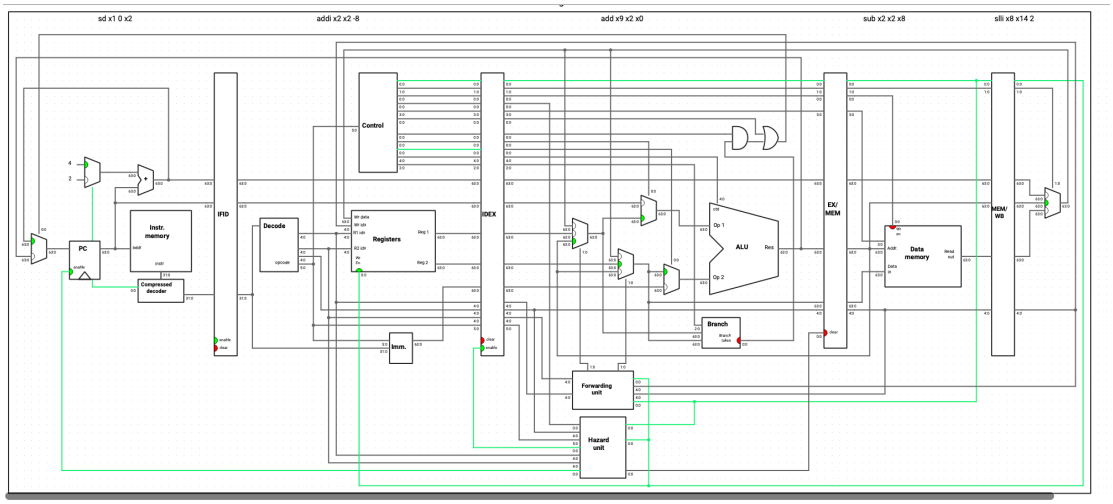


image:a-2

By image a-1, we can see that sp in line 54 is used after the subtraction of sp and s0 in line 53, and by image a-2, we can see

ID/EX.RegisterRs1 = EX/MEM.RegWrite(there's a dependency on MEM and EX stage).

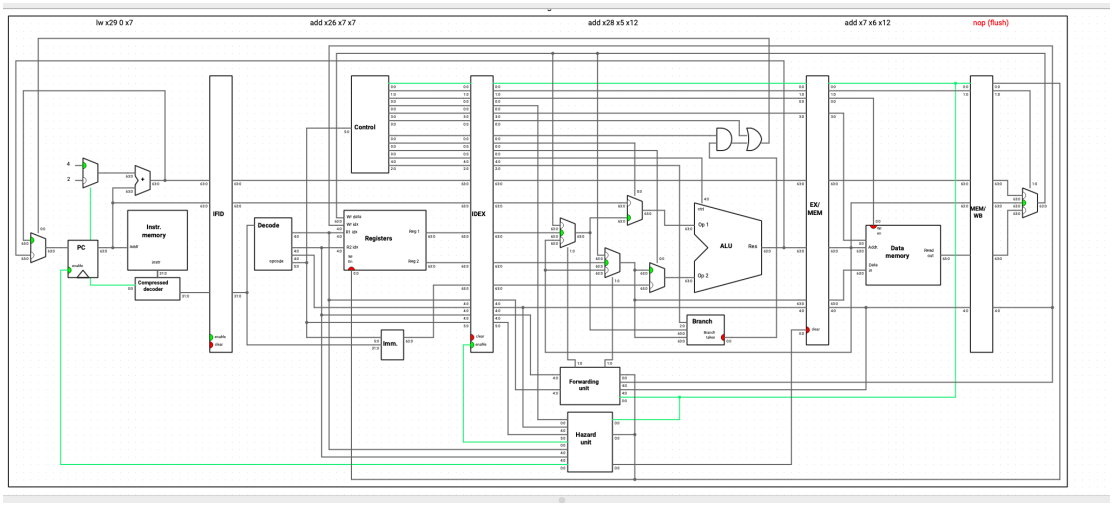
As a result, we can say that it is an EX Hazard with R-type RAW, and the result calculated by ALU may be sent to EX/MEM pipeline reg and the value of x2 may be set by forwarding so that add x9, x2, x0 will not use the old value.

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

Image:b-1

```
75    while_for_LPS:
76        add t2, t1, a2
77        add t3, t0, a2
78        #create type2 hazard
79        add s10, t2, t2
```

Image:b-2



By image b-1, we can see that t2 in line 79 is used after the t2 in line 76, and by image b-2 ,we can see

ID/EX.RegisterRs1 = MEM/WB.RegWrite(there's a dependency on WB and EX stage).

As a result, in imageb-2, we can say that it is an MEM Hazard with R-type RAW, and the value in MEM/WB reg, which stores the result of add x7, x6, x12.

Besides, it forwards to the forwarding unit and we can make use of it when executing add x26, x7, x7 to prevent from using old value.

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

```
134     add s11, s5, s5
135     #type 3
136     lb s7, 0(s6)
137     beq s5, s7, if_for_KMP
```

Image:c-1

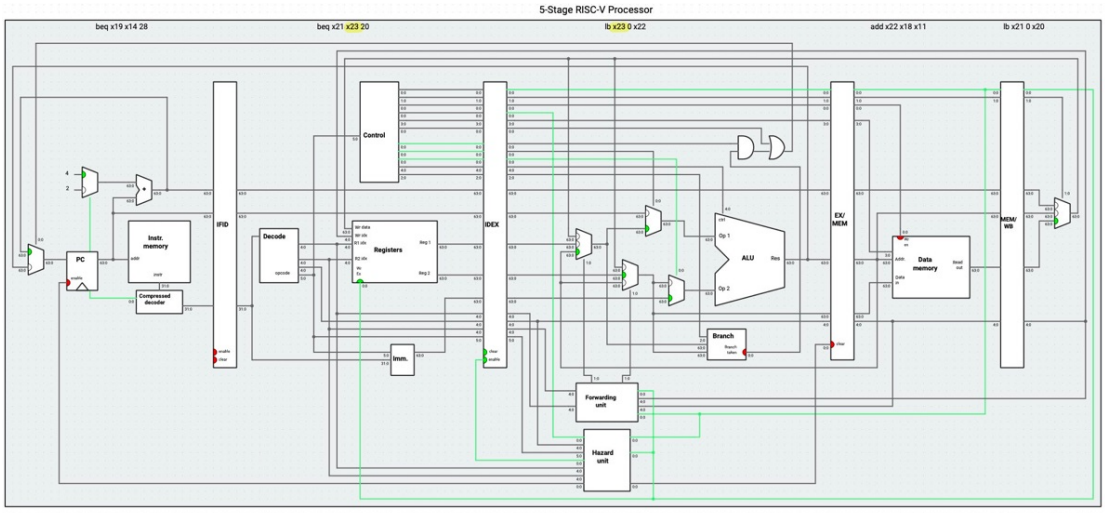


Image:c-2→ ID/EX.RegisterRd = IF/ID.RegisterRs1

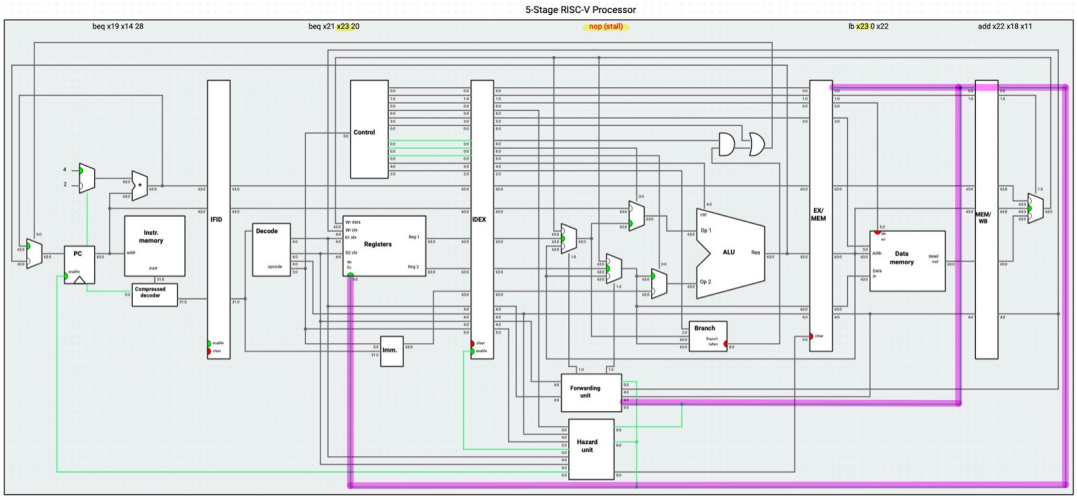


Image:c-3

By image c-1, we can see that s7 in line 137 is used after the load instruction(lb s7, 0(s6)→load the value in s6 to s7) in line 136, and by image c-2, we can see ID/EX.RegisterRd = IF/ID.RegisterRs1 (there's a dependency on ID and EX stage).

As a result, we can say that it is an Load-use Hazard , and in order to get the correct value, we have to stall for a cycle, just as what image c-3 shows.

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

```
130     add s4, s3, a2
131     lb s5, 0(s4)
132     add s6, s2, a1
133     #type4
134     add s11, s5, s5
```

Image:d-1

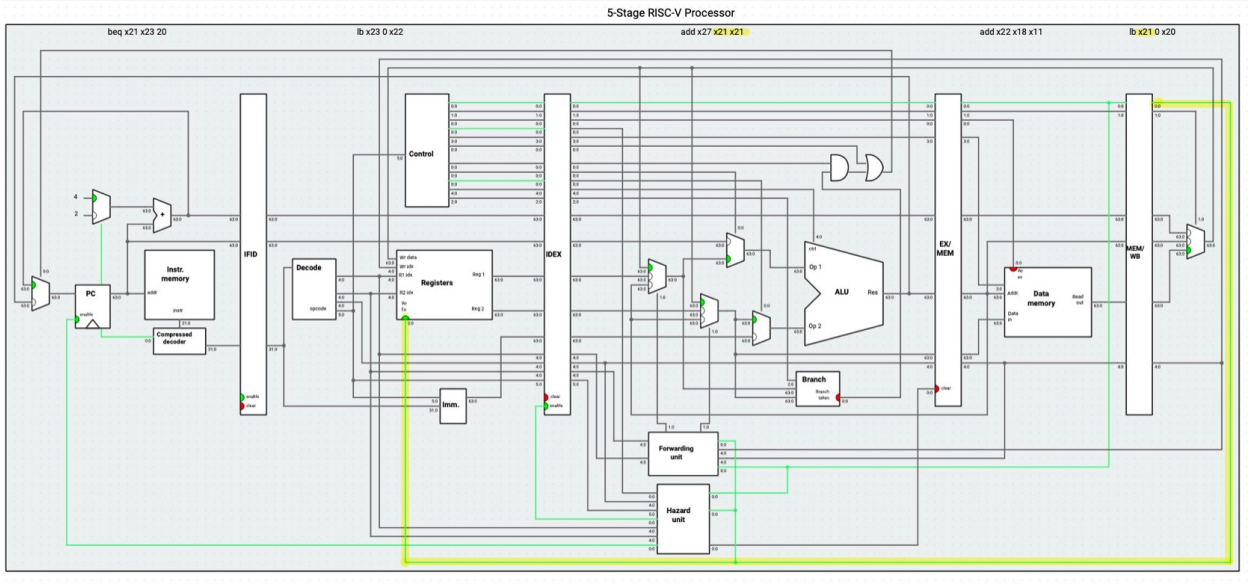


Image:d-2→ ID/EX.RegisterRs1 = MEM/WB.RegWrite

By image d-1, we can see that after s5 load byte from s4 in line 131, an addition instruction s11, s5, s5 use the value of s5 in line134.

And by image d-2, we can say that it is a MEM Hazard, and the value of new s5 forwards to the unit and the value of s5 is updated so that the result of calculation will be true.

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

```
continue:
    beq s3, a4, if2_for_KMP
    beq x0, x0, else_if2_for_KMP
    blt s2, a3, while_for_KMP
    jalr x0, 0(x1)
if_for_KMP:
   addi s2, s2, 1
```

Image:e-1

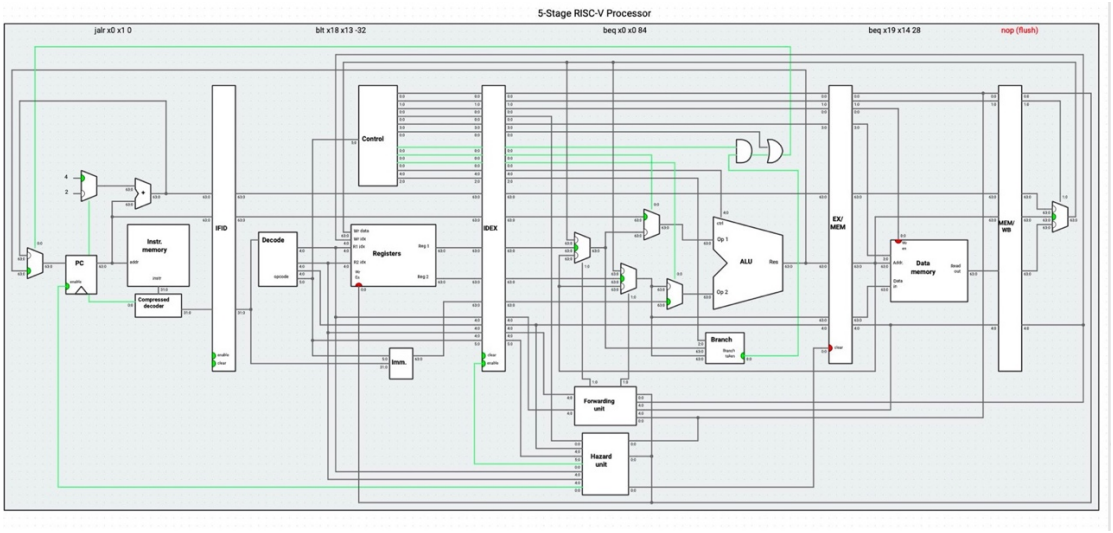


Image:e-2

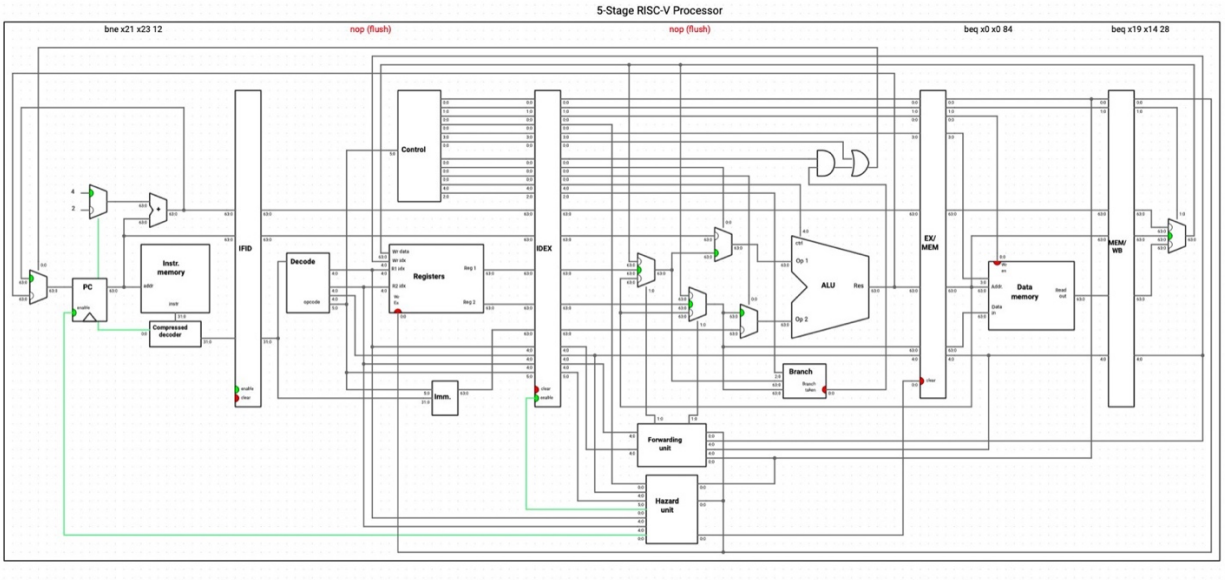


Image:e-3

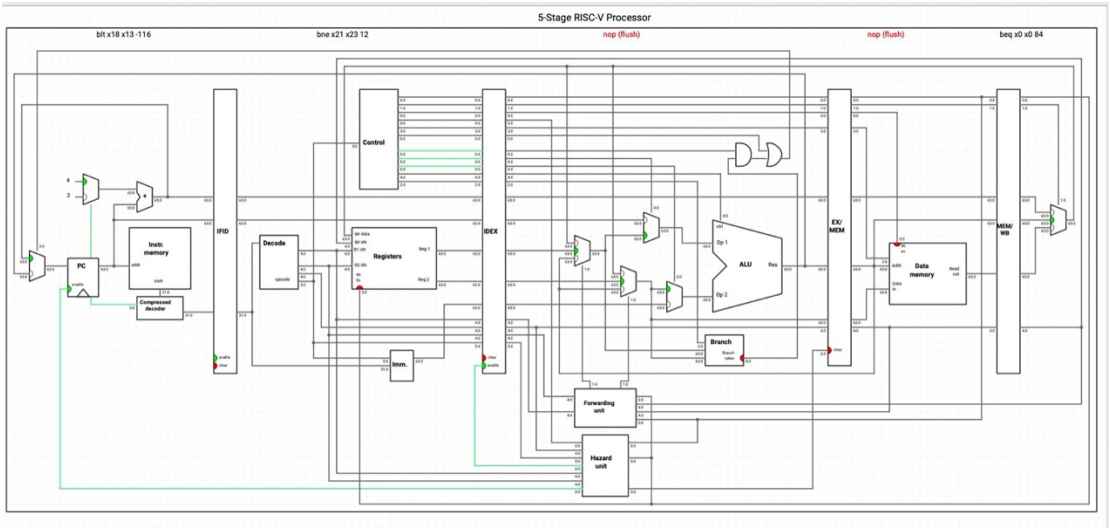


Image:e-4

We can see that there an instruction beq x0, x0, 84 and the control Hazard happens so that the next two instructions that have been fetched will be flushed.

The reason to this hazard is that the outcome of branch is determined in MEM, but since we use pipelining to help us, we have to predict the condition and fetch the incoming instructions.

When the prediction is wrong, we need to flush the wrong instructions and fetch the correct one ,and since we predict the equal doesn't stands, the hazard happens.