**One Operand**

**Increment I**

1-Not R1

2-NOP

3-INC R1 ***-- Hazard1***

4-IN R1

5-IN R2

6-Not R2 ***--*** ***Hazard2***

7-INC R1

8-Dec R2 ***-- Hazard3***

9-Out R1 ***-- Hazard4***

10-Out R2 ***-- Hazard5***

Hazard1:

Data hazard, actual value of R1 has not been written yet. Needs 1 Nop instruction before it to be executed properly.

Hazard2:

Data hazard, Needs 2 NOPs before it

Hazard3:

Data hazard, Needs 1 Nop only

Hazard4:

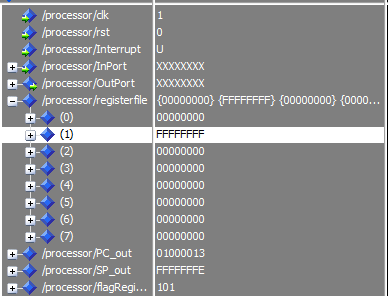
Data hazard, Needs 1 Nop only

Hazard5:

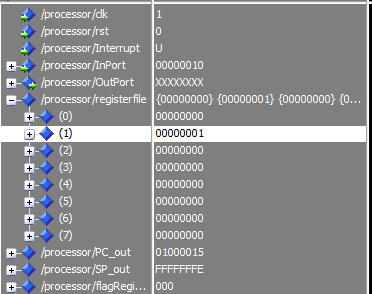
Data hazard, Needs 1 Nop only

Screenshots:

When instruction 1 is at WB stage (No hazards):

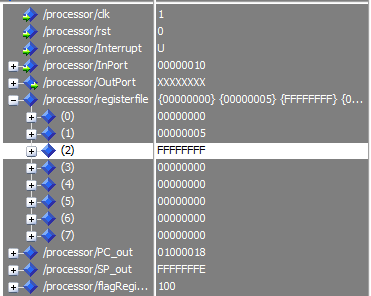


When instruction 3 is at WB stage:



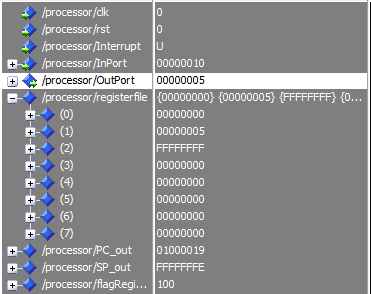
R1 has false value!, joined the instruction as 0 as the value FFFFFFFF has not been written back.

When instruction 6 is at WB



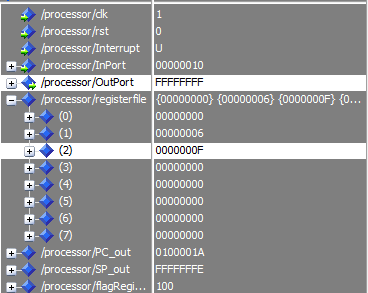
R2 joined as 0, input port value 10 has not been written back yet!

When instruction 9 is at execute



Output port has a value of 5 instead of 6!, as that value has not been written in R1 yet.

When instruction 8 is at WB and instruction 10 is at execute



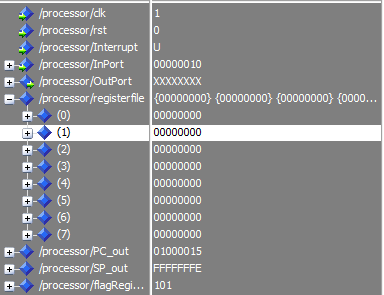
Similarly, output port has false value of R2, value written in R2 used value of 10 not FFFFFEF, thus yielded 0000000F, not EEEEEEEF

**Increment II**

All hazards is supposedly solved with Forwarding Unit

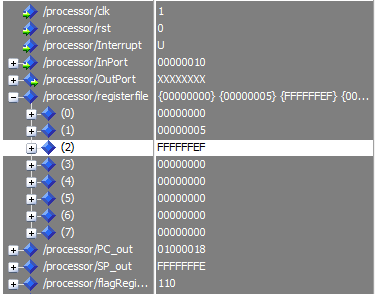
Screenshots:

When instruction 3 is at WB stage:



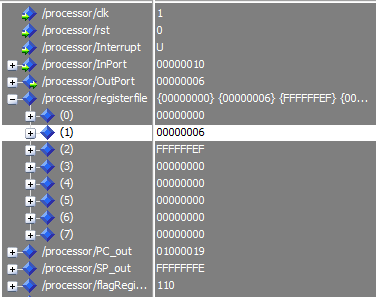
R1 is read correctly as FFFFFFFF, so yielded 00000000

When instruction 6 is at WB



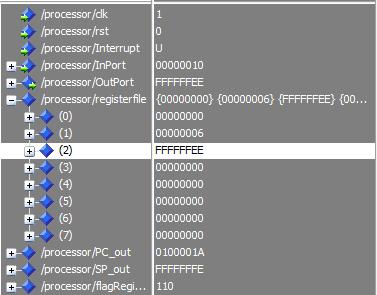
Correct value of R2 (10) was read, so yielded FFFFFFEF

When instruction 9 is at execute



Correct value of R1 is read and put at the outport

When instruction 8 is at WB and instruction 10 is at execute



Both the output port and R2 has correct values as they read the correct FFFFFFEF value of R2

**Increment III and Increment IV**

Exactly the same as II, It was already hazard-free, no more hazards were to be handled by stalling or flushing needed.