**Two Operand**

**Increment I**

1- in R1

2- in R2

3- in R3

4- in R4

5- IADD R3,R5,2

6- ADD R1,R4,R4

7- SUB R5,R4,R6 ***-- Hazard1***

8- AND R7,R6,R6 ***-- Hazard2***

9- OR R2,R1,R1

10- SHL R2,2

11- SHR R2,3 ***-- Hazard3***

12- SWAP R2, R5 ***– Hazard4***

13- ADD R5,R2,R2 ***–Hazard5***

Note In instruction 5:

No Data hazard is found, IADD instruction already needs 2 cycles to execute, First one is already a NOP and R3 is needed in the second one, thus: 2 cycles have passed since R3 was in execute stage(in instruction 3), so the actual value of R3 can be read safely, similarly in instruction 6, 2 cycles have passed so R4 can be read safely.

Hazard1:

Data hazard on both R4 and R5, R5 Needs 1 Nop only, R4 needs 2 Nops, so a total of 2 Nops is sufficient.

Hazard2:

Data hazard on R6, Needs 2 Nops, for the actual value of R6 is in Ex/Mem buffer. Note that it's **NOT** an actual hazard, because regardless the value of R6, the result would be zeros, as R7 is also zeros 🡺 (0 and X) = 0

Hazard3:

Data hazard on R2, Needs 1 Nop only as SHR executes in 2 cycles (for it has immediate value), first one is already a Nop.

Hazard4:

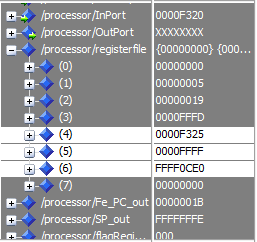
Data hazard on R2 only, Swap already takes 2 cycles, however, according to our design Rsrc1 is moved to Rsrc2 in the first cycle, so the instruction Needs 2 Nops. Note that if we considered moving Rsrc2 to Rsrc1 first in our design, it would need only 1 Nop. As R5 is hazard-free.

Hazard5:

Data hazard on both R2 and R5, R5 Needs 1 Nop only as it was written first in Swap as clarified above, R2 needs 2 Nops, so a total of 2 Nops is sufficient.

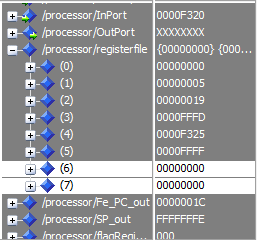
**Screenshots:**

When instruction 7 is at WB stage:



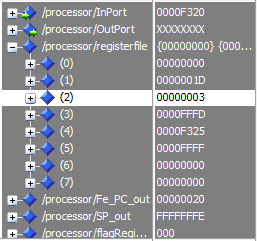
A false value of R4 and R5 was read, yielded 0 – F320 = FFFF0CE0

When instruction 8 is at WB stage:



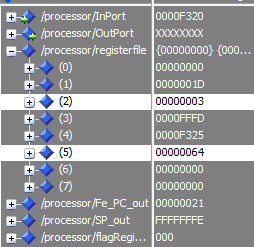
A false value of R6 was read, however, the result yielded a correct 0 as clarified previously.

When instruction 11 is at WB stage:



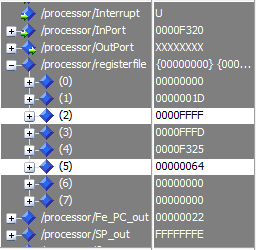
A false value of R2 (19) was used, yielded 3 on shifting right 3 times.

When instruction 12 is at WB stage (1st half):



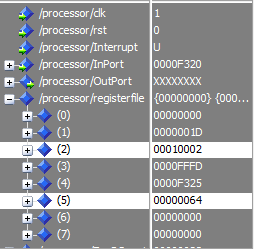
A false old value (64) of R2 is read and put in R5. Instead of the correct one (3)

2nd half:



Correct value(FFFF) was put in R2, as R5 was already hazard-free.

When instruction 13 is at WB stage:



False values or R2 wa used (3 instead of 0C), yielded the false

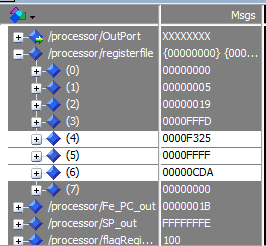
3 + FFFF = 10002

**Increment II**

All hazards are supposedly solved with Forwarding Unit

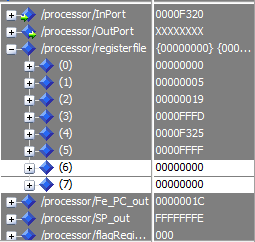
**Screenshots:**

When instruction 7 is at WB stage:



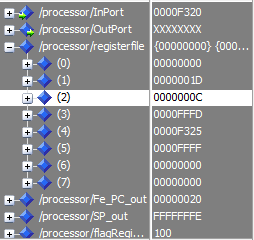
Proper value of R4 and R5 is read, so R6 is accurately calculated.

When instruction 8 is at WB stage:



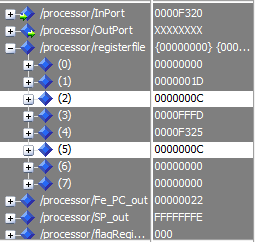
Whatever the value of R6 is, the result will yield 0

When instruction 11 is at WB stage:

****

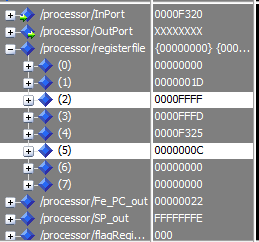
R2 is shifted properly, for the right value (64 was used)

When instruction 12 is at WB stage (1st half):



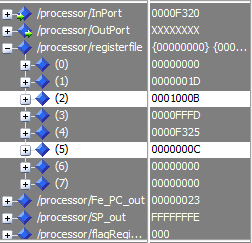
The actual value of R2 (0C) is read and directed to R5

2nd half:



The actual (old) value of R5 (FFFF) is read and directed to R2. Note that forwarding from Ex/Mem buffer is suspended here in order to get the proper value (in this case the old FFFF, not the new 0C one)

When instruction 13 is at WB stage:



A proper value for both R2 and R5 is read, yielded the correct 1000B value.

**Increment III and Increment IV**

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