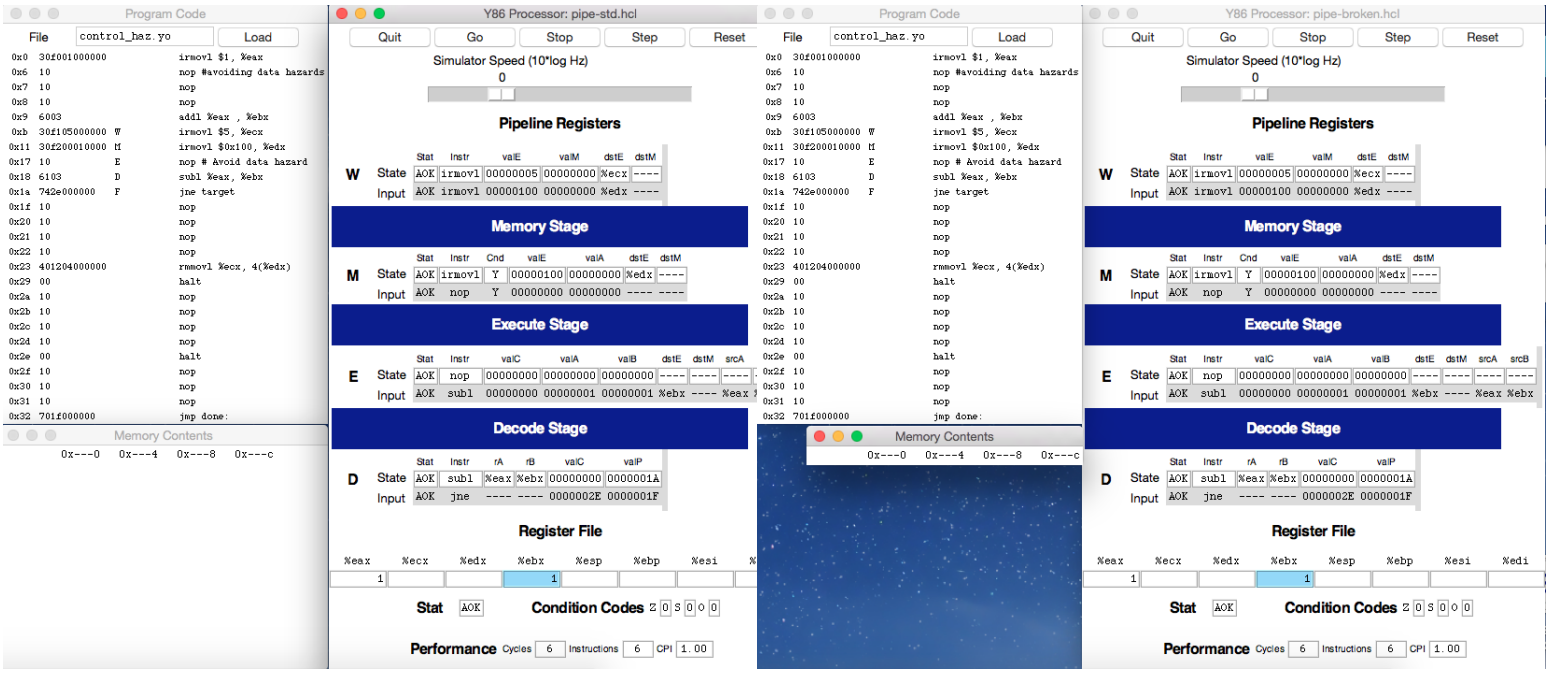
Control Hazard Demonstration

Carter J. Bastian *

May 2015

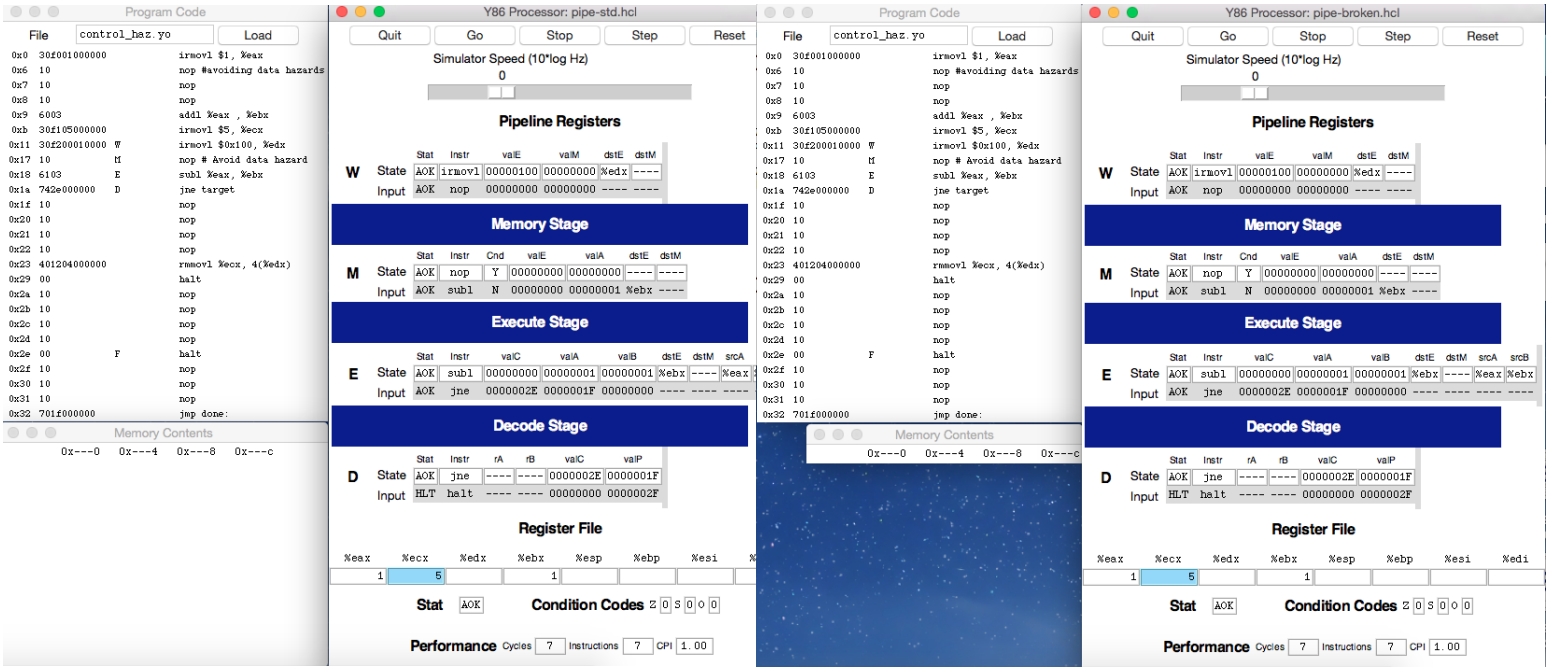


(a) Correct Implementation (std)

(b) Incorrect Implementation (broken)

Figure 1: In this step, the `jne` instruction has just been fetched by the two different versions.

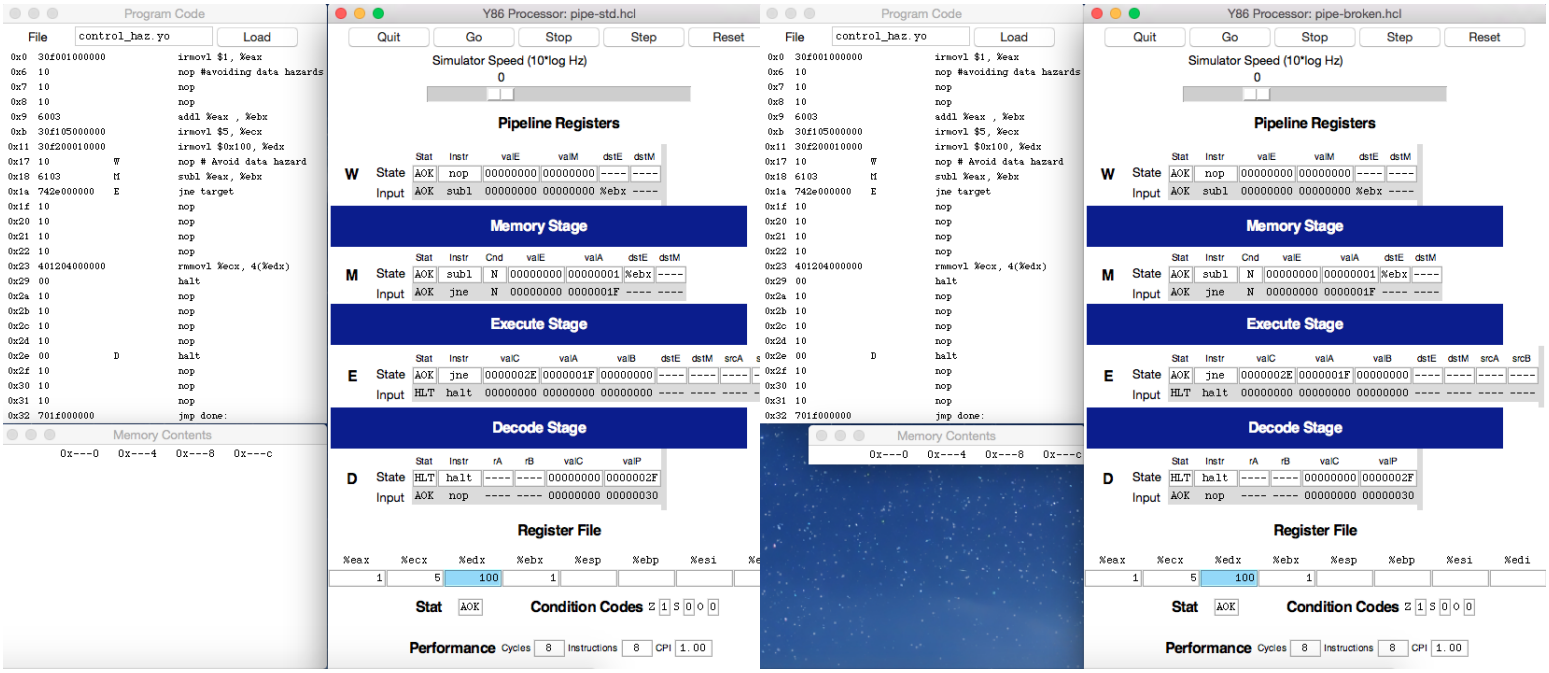
*COSC51 HW9 Q4



(a) Correct Implementation (std)

(b) Incorrect Implemenetation (broken)

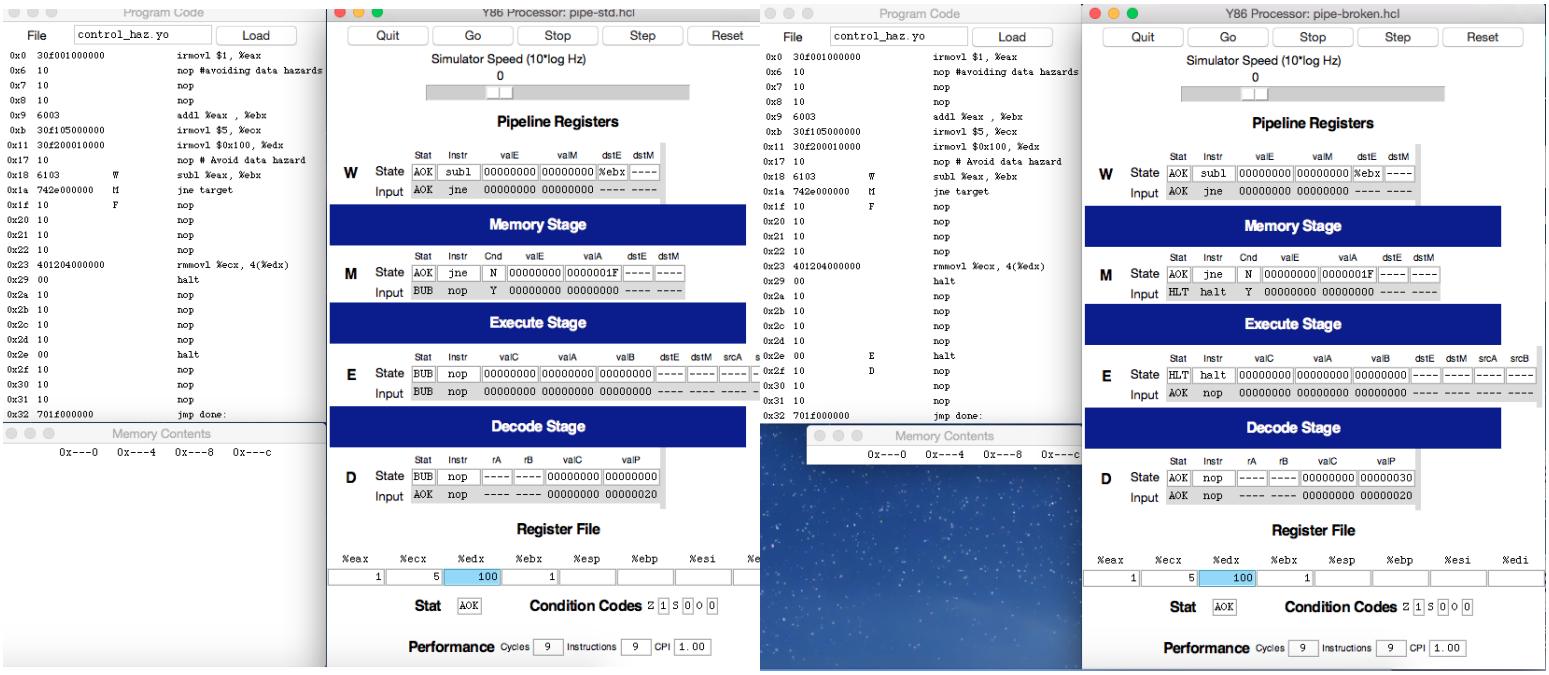
Figure 2: In this step, both implementations predict the conditional jump to "target" and fetch the `halt` instruction which should not be executed.



(a) Correct Implementation (std)

(b) Incorrect Implementation (broken)

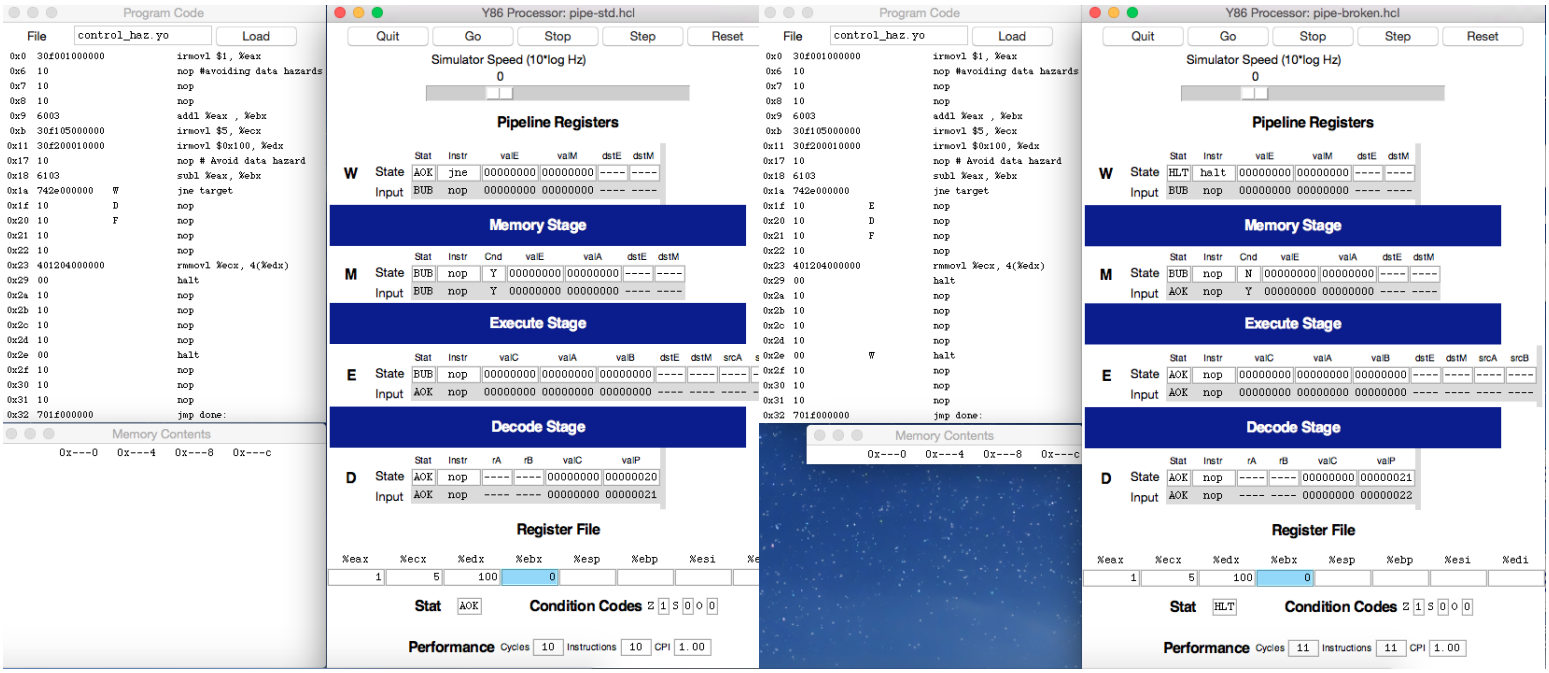
Figure 3: The `jne` instruction reaches the Execute Pipeline Register. It's now clear that the conditional jump should not have been taken. However, the "target's" `nop` instruction has just been fetched, and the `halt` instruction has been loaded into the Decode Pipeline Register.



(a) Correct Implementation (std)

(b) Incorrect Implementation (broken)

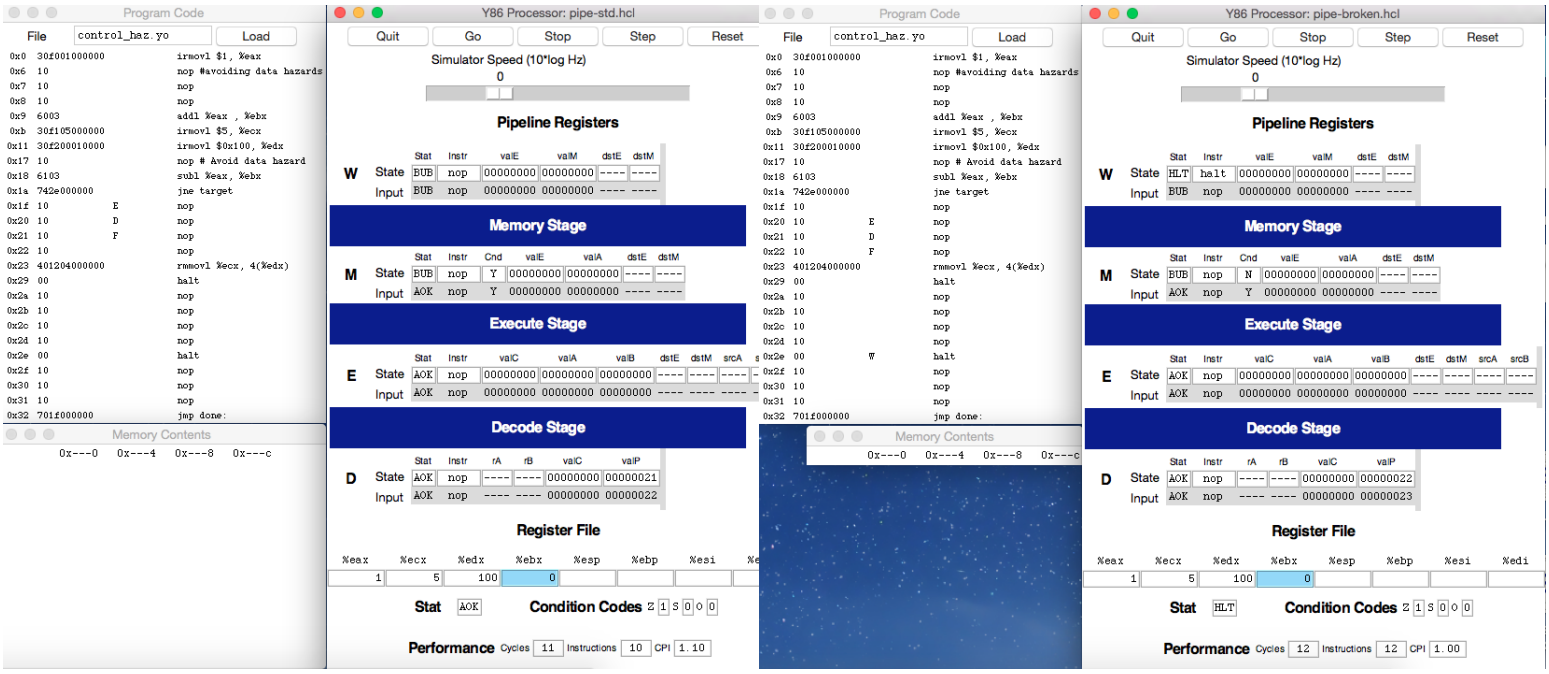
Figure 4: This step best illustrates the difference between the two versions. The correct implementation cancels the two incorrectly-fetched instructions (by converting them into `nop bubbles`) and then fetches the next instructions from the correct location – the "done" tag. By doing so, the correct version mitigates the control hazard.
On the other hand, the incorrect version simply fetches the next instruction from the correct location without cancelling the `halt` instruction or the `nop` instruction, simply progressing them to the Execute and Decode stages respectively.



(a) Correct Implementation (std)

(b) Incorrect Implementation (broken)

Figure 5: You can see the invalid instructions progress through the pipeline stages. You can also see the Correct implementation's `nop bubbles` move through the stages and the correct instructions getting fetched.



(a) Correct Implementation (std)

(b) Incorrect Implementation (broken)

Figure 6: Both programs have finished processing the `jne` instruction. You can see that the correct implementation has successfully continued on to the right part of the program without allowing the instructions from "target" to change the programmer-visible state. On the other hand, the invalid instruction is halted and will no longer process instructions. It will never even get to the rest of the correct instructions in the "done" section.

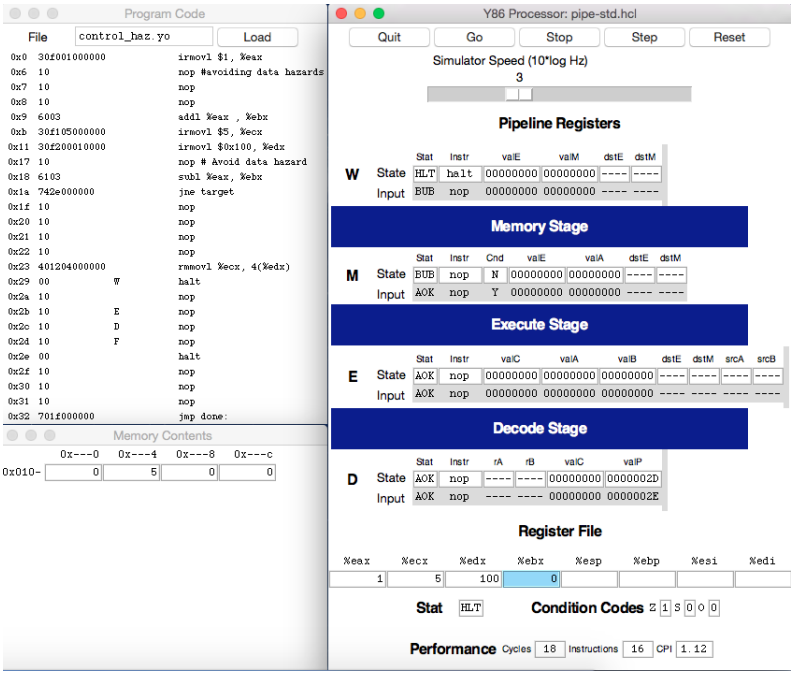


Figure 7: Here you can see the programmer-visible state from when the Correct Implementation (std) halts. Compare this to the programmer-visible state from Figure 6: (b) and you will see that there are problematic differences caused by the control hazard from control_haz.ys