CS311 - Computer Architecture Lab

Assignment 4 - Report

- Sourabh Bhosale (200010004)
- Surzith Raj Pandi (200010049)

1. Statistics Table:

- Total no. of cycles taken by each benchmark program.
- The number of times the OF stage needed to stall because of a data hazards.
- The number of times an instruction on a wrong branch path entered the pipeline.
  All the calculations for the lab assignment are tabulated below.

	Total No. of Clock Cycles	Total No. of OF Stalls	Total No. of Wrong Branches	Total No. of Instructions
descending	571	114	168	365
evenorodd	16	6	0	6
fibonacci	148	34	24	94
palindrome	123	56	14	56
prime	58	15	10	34

2. Interpretation and Comments:

For odd-even, code has no control hazards because the conditional branch was not taken as the value checked was odd. There are a few RAW hazards due to which we see a small number of data hazards.

For fibonacci, there is a control hazard in the loop. Every time it goes in the loop, it is a wrong branch, till the count reaches 10. So, we see a few control hazards. Similarly, there are a few data hazards outside the loop and a few inside. Hence, we also see some stalls in the pipeline.

For descending, code has a lot of branching, 3 nested loops and data hazards. Hence, we see that the total number of hazards increases by a large amount and so does the cycles executed.

For Prime, the loop runs from n=2 to n=5, i.e. 4 times. All these 4 times, we get a wrong branch taken for the jmp statements. Then it takes a wrong branch when it is supposed to enter prime/not prime. Hence, a total of 10. Similarly, we are getting a few data hazards.

We may conclude that that every time the code misses the jump to loop, there is a wrong branch taken, thus we get a control hazard. But we have more data hazards per loop. Hence, we see a larger number of data hazards than control hazards. We additionally add NOP instructions to avoid hazards which in turn increase the total number of cycles, for control hazards, we sometimes even use delay slot, i.e. reordering the code in such a way that two independent instructions above the branch are executed after the branch (transferring them to delay slots).