a. Report counters statistics for the 5 benchmark inputs: benchmarks/{fibi.c, fibm.c, fibr.c, mmmRV32IM.c} and benchmarksO3/mmmRV32IM.c. Comment on the behavior and performance differences across the 3 fib programs and across the -O and -O3 mmm programs.

The IPC is the highest for fibi.c despite the highest number of single issue stalls.

|  | Cycle Count | Instr Count | Instr Exec | ID\_2 | ID\_1 | ID\_1DataHaz | ID\_1  Non ALU | ID\_1  Single  Issue | ID\_0 | ID\_0  Data  Haz | ID\_0  Inval | Control Flow Exec | Mispredict Count |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| fibi.c | 16109 | 28876 | 26456 | 15647 | 5103 | 457 | 11 | 4641 | 1213 | 0 | 1213 | 5546 | 1211 |
| fibm.c | 30333 | 31684 | 30048 | 16291 | 11847 | 10824 | 5980 | 1018 | 4033 | 3213 | 820 | 2135 | 818 |
| fibr.c | 30733 | 40596 | 34772 | 22643 | 9079 | 8083 | 6806 | 991 | 2916 | 2 | 2914 | 989 | 2912 |
| mmmRV32IM.c | 41921 | 47670 | 46426 | 23926 | 14129 | 9511 | 4610 | 4613 | 9103 | 8479 | 624 | 8479 | 622 |
| mmmRV321M.c (O3) | 17120 | 20000 | 19846 | 10040 | 3229 | 2947 | 4466 | 277 | 4207 | 4128 | 79 | 294 | 77 |

b. Discuss and report additional counters you introduced.

We did not introduce any additional counters.

c. Describe the specific optimization techniques/mechanisms you employed beyond the

minimum requirement. Use counter data to support their effectiveness. (How does it

improve performance? What execution conditions or program behaviors are being

exploited? How common place is the exploited condition or behavior?)

We first optimized our critical path by moving calculations for values that could be calculated at earlier stages to the earliest stage possible and passing the value down in registers. This lowered our critical path timing. We then increased our IPC by increasing the size of the BTB so that our mispredicted amount of control flow instructions would be at its lowest. Both of these worked to increase our MIPS to the highest we could get it. After that, we made sure to remove any registers that were passing values no longer needed down to later stages. This optimized our area and power usage. When there is a data dependency between two ways, we will let the primary way go forward by 1 cycle, and only stall 1 cycle for the secondary way. Our cycle count decreased after we implemented these techniques. We found that the register values for mixed.c tend to change each time, so we update the BTB write data each time. Also, we count the total number of instructions for mixed.c. The common things are we must look at all disassembly code to get what’s going on in the program. And they have something to do with the register values.

d. Discuss if the observed behavior and performance agree with your intuition/expectations.

(Does your technique/mechanism work as well as you had hoped? What is the

performance bottleneck at the end?)

Our techniques for optimizing MIPS worked just as we expected. And optimizing that also allowed us to lower power consumption because we finished faster. One of our biggest performance bottlenecks is our mispredict rate which could only be improved by a better predictor for said tests. In addition to that, the stall rate also lowered our IPC.