Project 2: Terminators

Youssef Amin, Zhuo Li, Yue Hu

November 2025

1 Task 6 - Performance results

This task compares the performance of our Project 1 compiler (which uses memory slots for all variables) with our Project 2 compiler (which uses register-based variable storage). The main goal is to see how moving from memory loads/stores to dedicated registers affects runtime. Since memory access is slower, we expect that reducing the number of `ld`/`sd` instructions will improve performance, especially for workloads that involve many arithmetic operations or loops.

The test files includes stress tests from Project 1, as well as several new programs that contain dead code (unused branches, loops, and assignments).

Table 1 summarizes the timing results for the stress tests on `risc-machine-2.cs.unm.edu`, More details are available in our project `README.md` under the `tests` folder:

Workload	**Input**	**Project1**	**Project2**
Factorial (Addition)	12	0m2.960s	0m1.482s
Factorial (Addition)	13	0m38.434s	0m19.218s
GCD (Mod)	(102334155,102334156)	0m2.205s	0m0.321s
GCD (Sub)	(1,100000000)	0m0.959s	0m0.505s
Prime Count	10000	0m3.321s	0m1.573s

Table 1: Wall-clock Time Comparision for Workloads between Project 1 and Project 2.

Across all workloads, Project 2 consistently runs faster than Project 1. Table 1 shows that the speedup is especially large for programs with heavy arithmetic workload, such as factorial (addition), and prime counting. These programs perform many repeated operations, so removing the large number of memory loads and stores from Project 1 makes a clear difference in execution time.

Table 2 summarizes the total number of memory operations(`ld` and `sd`) generated by each compiler on `risc-machine-2.cs.unm.edu`. This gives a clear comparison of how much each compiler depends on memory.

Program	Project 1	Project 2
abs_val	5	4
counter_while	8	4
complex_dead	19	16
constant_prop	14	8
dead_updates	12	8
dead_while	8	6
redundant_arith	14	10
example1_factorial	12	8
collatz	21	12
countTriangularNumbers	28	14
stress_factorial	21	14
stress_gcd_mod	18	12
stress_gcd_sub	15	10
stress_prime_count	30	18
stress_square	17	12
sumfactors	25	12
test_branching	17	8

Table 2: Memory Operations Comparison between Project 1 and Project 2.

Table 2 shows that Project 2 also produces fewer memory operations for almost every test file. This includes both normal programs and programs that contain dead code. For the dead-code tests (`dead_updates`, `dead_while`, `complex_dead`, and `constant_prop`), Project 2 still reduces memory activity even though some commands never affect the final result.

Overall, these results show that our compiler in Project 2 has greatly improved efficiency in both runtime and memory behavior so far.