Part 3: Performance Analysis B

Code Overview

The *_start* block of my program contains all the relevant setup required to initiate the game. The main block we are concerned with is *IDLE* where most of the user interaction and game state updates happen. My approach to registering keypresses is interrupt-based. However, *IDLE* polls the global *DATA* variable which holds the most recently fetched keyboard signal from the keyboard's data register. It tries to match the *DATA* variable against relevant make signals corresponding to the keys that are relevant in game logic and performs the appropriate updates to the game's state.

Performance Analysis

I will consider all instructions executed in *SERVICE_IRQ* and *PS2_ISR* as instructions attributed to I/O interactions or interrupts, and all instructions executed in *IDLE* to be instructions servicing user code or game logic. I am not attributing any instructions to polling as there is no consistent time frame between keystrokes. Therefore, code spent polling the *DATA* variable after a keystroke interrupt will be attributed to game logic as well. To get an idea of how many instructions and data memory accesses are executed in each category, I will go through two scenarios of updating the game board's state: from pressing the N key to seeing the updated game board and from pressing D to seeing the updated cursor. My methodology in measuring the relevant data will be to insert a breakpoint at the beginning and end of *SERVICE_IRQ* to measure the number of data memory accesses and executed instructions serviced to interrupts. I will also insert breakpoints at the beginning and end of the *update_GoL_board* and *move_cursor_right* subroutines to measure how many executed instructions and memory accesses are serviced to game logic in each case. With the way my ISR is set up, I will have to run measurements on independent runs. This is because my ISR empties the data register of the PS/2 keyboard, so once I reach the ISR's first breakpoint, the ISR will empty the make and break signal of the keypress (unless I hold the key until I exit the ISR, but that would be impractical) which won't match any of the conditions specified in the *IDLE* block.

Counter	Value	Counter	Value
Exec. instructions	4260985	Exec. instructions	4261815
Data loads	580038	Data loads	580214
Data stores	636971	Data stores	637093
Simulator run time (ms)	1295	Simulator run time (ms)	1296
Simulator core run time (ms)	1156	Simulator core run time (ms)	1157
Simulated MIPS	3.683	Simulated MIPS	3.680

Figure 1. Counter values before and after the interrupt service routine

Counter	Value	Counter	Value
Exec. instructions	5635945	Exec. instructions	7340895
Data loads	763250	Data loads	996984
Data stores	637061	Data stores	938393
Simulator run time (ms)	1688	Simulator run time (ms)	2184
Simulator core run time (ms)	1495	Simulator core run time (ms)	1970
Simulated MIPS	3.767	Simulated MIPS	3.724

Figure 2. Counter values before and after calling *update_GoL_board*

Counter	Value	Counter	Value
Exec. instructions	6067139	Exec. instructions	6077603
Data loads	820779	Data loads	822238
Data stores	637029	Data stores	638930
Simulator run time (ms)	1812	Simulator run time (ms)	1819
Simulator core run time (ms)	1597	Simulator core run time (ms)	1603
Simulated MIPS	3.797	Simulated MIPS	3.789

Figure 3. Counter values before and after calling *move_cursor_right*

From Figure 1, we can tell that running the ISR executes 830 instructions, performs 298 data memory accesses, and requires 1 ms of simulator runtime. Similarly, the *update_GoL_board* subroutine (Figure 2) executes 1704950 instructions, performs 535066 data memory accesses, and requires 496 ms of simulator runtime. Lastly, the *move_cursor_right* subroutine executes a total of 10464 instructions, performs 3360 data memory accesses, and requires 7 ms of simulator runtime. From these results, we realize that the bulk of the game logic is concentrated in *update_GoL_board* as it executes a total number of instructions and memory accesses that are orders of magnitudes greater than the ISR and moving the cursor (we can generalize that moving the cursor in all different directions will yield similar results to *move_cursor_right*). However, the takeaway here is that the ISR takes up minimal resources in terms of runtime and processing which is ideal as we would like to minimize the time servicing interrupts and maximize the time the processor spends on user code or game logic. From the gathered results, we can conclude that the time spent in the ISR is trivial and that the overwhelming majority of the program's runtime and resources will be spent processing game logic.