CS 3432 Spring 2022

Lab 3 Sudoku in RISC-V

The goals of this lab are 1) to learn how to use the RISCV toolchain to compile, assemble, link, and execute C and RISC-V assembly programs for simulated RISCV hardware, and 2) to trace and understand in detail how the stack is used for procedure calls.

For this lab, we will use a Sudoku solver that uses the backtracking algorithm. The backtracking algorithm is a depth-first search for all possible solutions. A Sudoku puzzle contains 81 cells in a 9x9 grid divided into 9 boxes. Each cell may contain a digit from 1 to 9, and each digit may occur only once in each row, column, and box. A Sudoku puzzle starts with some cells containing digits (called clues), and the goal is to fill in the remaining cells. A proper Sudoku puzzle has one and only one solution. A typical Sudoku puzzle is shown below. See the Wikipedia article at <https://en.wikipedia.org/wiki/Sudoku_solving_algorithms> for more details about Sudoku puzzles and the algorithms used to solve them.

A picture containing text, crossword puzzle

Description automatically generated

The backtracking algorithm tries filling an empty cell with a digit, and if the addition of that digit does not cause the grid to become invalid, it moves on to the next empty cell. If the new digit causes the grid to become invalid, the algorithm backtracks to a previous cell and tries a number it has not tried there before.

A C solution is provided for you in the files sudoku.c and trycell.c. You should compile and link these files using your RISC-V toolchain compiler as follows:

$ riscv64-unknown-elf-gcc -o sudoku sudoku.c trycell.c returnsps.s

[Note: The exact name of your RISCV compiler may be different depending on your version of the toolchain.]

You can then run your RISC-V program using the Spike simulator as follows:

$ spike pk ./sudoku > output.txt

You should see the following output:

Puzzle to solve

5 0 0 0 7 0 0 0 0

6 0 0 1 9 5 0 0 0

0 9 8 0 0 0 0 6 0

8 0 0 0 6 0 0 0 3

4 0 0 8 0 3 0 0 1

7 0 0 0 2 0 0 0 6

0 6 0 0 0 0 2 8 0

0 0 0 4 1 9 0 0 5

0 0 0 0 8 0 0 7 9

Solution:

5 3 4 6 7 8 9 1 2

6 7 2 1 9 5 3 4 8

1 9 8 3 4 2 5 6 7

8 5 9 7 6 1 4 2 3

4 2 6 8 5 3 7 9 1

7 1 3 9 2 4 8 5 6

9 6 1 5 3 7 2 8 4

2 8 7 4 1 9 6 3 5

3 4 5 2 8 6 1 7 9

You can run your program in debug mode by invoking spike as follows:

spike -d pk ./sudoku

Running in debug mode will allow you to step through the execution and examine the contents of registers. You will be asked to do this during your demo and answer questions about the execution. We will provide a set of questions you will be asked to answer a week prior to the due date.

You should also implement a stack trace facility that outputs a message about allocation of the stack frame including the value of the sp register after stack frame allocation and deallocation. One possible way to implement your stack trace facility is to write an assembly language procedure called returnsp that returns the current value of the sp register, put it in a file called returnsp.s, and link that file with your .c files. You can call then call the returnsp() function from your C code. The output of your stack trace will be long, so it would be best to redirect it to a file when you run your completed program.