Project Instructions

Introduction

In the first four labs, you wrote small pieces of RISC-V assembly code to implement conditionals (if-statements), loops, functions, and arrays. In the project, we're asking you to use those skills together to build a game based on Sokoban.

To complete t[…]ment:

- The base re[…]ay the game.
- Two enhanc[…] suggest or suggest your own.
- User docum[…]n and play the game.

Your project i[…]0 p.m. Submit both your .s (assembly program) and .pdf (user documentation) to their respective assignments[…]

Backgr[…]

Sokoban is a puzzle game. The original version is played on a grid representing a warehouse, where every square is a floor or wall. The character occupies a floor space, and boxes are scattered across the grid, also on the floor space. Targets (also called storage locations) are scattered on the floor. Neither the character nor the boxes can enter a wall, and the character can push one box at a time. The goal is to push all of the boxes onto the targets.

We will mimic the core game-play. You will use a grid of LEDs to represent the board. The character, boxes, walls, and targets will each be represented by specific colors of LED (that you may choose). We will use a D-pad to control the movement of the player, and we will take a reset input on the console. Messages, such as congratulations sent on success, will also be sent to the player via the console.

Starter Code

We are providing starter code that provides examples of how to generate random numbers and how to handle I/O. Your first task is to read this code to see what facilities it provides.

- **main** is not implemented. It contains a sequence of TODO instructions that will guide you through the base functionality.
- Above **main**, there are several memory locations reserved to store the location of the player, box, and target location. You may wish to update these declarations later, but you should be efficient about your use of memory. Use bytes, for example, instead of words unless you need the additional range provided by words.
- The **rand** function provides a (not really) random number that you can use to set initial locations for the player, box, and target.
- The **setLED** and **pollDpad** functions handle memory-mapped IO. They will allow you to turn the LEDs on and off and to receive input from the user via the d-pad.

The last two functions are particularly interesting, and I encourage you to look at them to see how they work. We'll discuss ways to handle communication later in this course, and mapping devices into memory is a common technique. In short, input from the device is written into a specific location in memory, where your processor can request it using the normal memory interface, instead of needing to deal with the device directly. Similarly, output to the device is written to a specific location in memory, and the device will check that location to accept and process your output (its input).

Running the Starter Code

Before you run your code, you will need to configure the I/O devices. Open Ripes and select the I/O tab.

Double-click on the "LED Matrix" device to get an LED matrix if there is not already one. To the right, you should see a panel "LED Matrix 0" with parameters "Height", "Width", and "Size". Set the height and width to 8 and 8. Set the size to a value that makes it easy for you to distinguish the LEDs.

Next, double-click on the "D-Pad" device to get a d-pad if there is not already one. To the right, you should see a panel "D-Pad 0". Select it and read the instructions. No values need to be set here; they should be okay as-is.

Finally, to make sure the program runs quickly enough, use the single-cycle processor and use fast execution (the ">>" symbol). That's how we'll mark, and you should configure your processor this way as you develop the code.

Base Requirements

Successfully implementing the base requirements will give you a score of 6 / 10. The base requirements are intended to allow someone to play the game. Your program must:

- Generate a random location for the character, 1 box, and 1 target.
- Light up the correct elements of an 8x8 LED grid to represent the character, boxes, and targets. The perimeter of the grid should be colored to represent walls.
- Read input from the d-pad and react to each of the 4 directions of the d-pad being pressed by moving the "character" and any "box" in front of them corresponding to the direction selected.
  - The LED grid should update to match the corresponding state.
  - Movement should be blocked by walls following the normal sokoban rules.
- Restart (by either providing a new randomly generated grid or resetting to the original) if the user requests it. You may choose -- and should document -- how the user signals that they want a restart.
- Inform the user on the console when the box has successfully been placed on the target.
- **The game should be solvable. (Updated: Sep 26)**

Your program must be written in assembly and will be submitted to Quercus as a .s file. It should not be written in C and compiled to assembly.

Enhancements

To obtain full marks on the project, you must implement two enhancements -- one from each category below. Each successfully implemented enhancement will add up to 2 points to your score, to a maximum of 10 / 10. You may receive less than the full 2 points for a partially completed enhancement.

Important: At the top of your sourcecode file, add a comment (a) stating precisely which enhancements you are implementing, (b) where we can find them (line number or label name), and (c) how they are implemented. This will help the TAs find your enhancements and to gauge whether they are well designed. If they are unable to find your enhancement, it'll be marked as not being present.

The ideas below are our suggestions. If you'd like to implement a different enhancement, please post your suggested enhancement to this thread on piazza. Do so at least one week prior to the deadline. We'll respond with "Yes, it fits into category [A/B]" or "No, sorry." If a feature has been approved, you may use it, whether or not you are the one who proposed it.

Category A: Memory enhancements
Option 1: Increase difficulty by increasing the number of boxes and targets.
  - This requires a change to how the boxes and targets are stored.
  - The number of boxes and targets should be (theoretically) unbounded. That is: you may not assume there is a maximum of, say, 64.
Option 2: Provide a multi-player (competitive) mode.
  - For full credit, the program should prompt for the number of players prior to starting the game.
  - Each player should be given a chance to play the same puzzle each round.
  - The program should track how many moves each player needs to solve the puzzle and should display cumulative standings (number of moves required) after the round.

Category B: Code enhancements
Option 1: Improve the random number generator by implementing (and citing in a comment) a formal pseudo-random generation function.
Option 2: Increase difficulty by adding a time limit on the game. Display this countdown in the console or via an LED interface.
Option 3: Generate internal walls in addition to the character, box, and target. To get full credit for this enhancement, multiple walls must be randomly generated but there must always be a path from the box to the target.
  - If you are very ambitious, also generate additional boxes and targets. The puzzle should still be solvable!
  - Note: This enhancement is far more challenging than the other two suggestions!

User Documentation

This project allows you to make important decisions regarding the user interface and how to interact with the program. For example:

- How will the game represent walls or the character?
- How will the game indicate that a d-pad key has been pressed but that no movement can occur?
- How does the user indicate that they want to restart?
- How will the game celebrate the player successfully solving the puzzle?

Your enhancements also need to be described. For example:

- If multi-player mode is implemented, how will players know that it is their turn?
- How many walls will be placed?

Finally, we provided some instructions for running your starter code -- but your users will not be reading this assignment handout. They will have Ripes installed but have never opened it, so you will need to tell them how to configure their system and how to start your game.

This documentation needs to be written with a *user* in mind. The user wants to play the game, not understand how your code is written. The user will not be familiar with Ripes. And the user wants to quickly find the information they need, if there is a problem.

The user documentation should be submitted as a PDF.

To get you started, here are some tips for writing clear rules (albeit, for a card game). And here are some examples of game manuals: Oregon Trail, Zork, and Lode Runner.

Grading

The assignment will be graded in two pieces: the program and the user documentation.

The Program

The assembly project is graded out of 10 marks:

  10/10: All base features are implemented correctly, and two enhancement features have been fully implemented.
  8/10: All base features are implemented correctly, and one enhancement feature has been fully implemented.
  6/10: All base features are implemented correctly.
  3/10: The program can light up the LED map but does not implement sokoban movement.
  0/10: No submission or unable to illuminate LEDs.

User Documentation

In addition to submitting your code (a .s file), you must submit a PDF that contains the *user documentation* for your game. The documentation will be marked out of 6 marks. We will be focusing on three qualities of your writing, each worth 2 marks.

Since you may do well in some aspects of one quality and less well than others, your grade may fall between levels. For example, consider the "Structure and Organization" quality. Your document may be well organized into sections (2/2), and the text may be organized appropriately into paragraphs and lists (2/2). However, your visual aids may not be referenced in the text at all (0/2). The grader may, depending on the importance of the various criteria in your particular writing, assign a grade of 1.5, 1.0, or even 0.5.

Structure and Organization

  2/2: The document is organized into appropriate sections presented in a logical order (e.g., "How to Start" is before "Gameplay" and "Ending the Game".) Visual aids are placed near text that references them. The text is organized into paragraphs or lists, as appropriate.
  1/2: The document is mostly organized into appropriate, logically ordered sections, but some sections could be reordered or long sections could be split. Some visual aids are placed far from where they are referenced or are not referenced. The text sometimes uses a list where a paragraph would be more appropriate.
  0/2: The document is not logically organized and should be split into sections. Visual aids are not placed near relevant text, are not referenced, or aren't present when they would be useful. The text frequently uses lists instead of organizing the text into paragraphs.

Writing Mechanics

  2/2: Sentences are concise (typically < 35 words), clear (easy to understand), and complete (containing a subject, verb, and appropriate phrases or clauses). Pronoun antecedents are clear. Verbs used are precise (e.g., "Click on a d-pad button" instead "Use the d-pad button").
  1/2: Some sentences are overly long, complex, or incomplete. Pronoun antecedents are sometimes unclear. Some verbs are imprecise.
  0/2: Sentences are long, complex, and/or incomplete. Pronoun antecedents are unclear. Verbs are imprecise.

Understanding Audience Expectations

  2/2: Terms are defined when first used. Only essential information is presented. All of the information a user needs is presented. Section headings are used to make relevant information easy to find. Gender-neutral language is used consistently.
  1/2: Some terms are not defined. Some inessential information is presented. Some information a user needs is missing. Some section headings are unclear or do not accurately present the material in the section. Some gendered language is used.
  0/2: Terms are introduced without definition. Essential information is difficult to find or missing. Section headings do not help identify where to find relevant information. Gendered language is used.