MP2 LC3-Your-Own-Adventure (part 1 of 2)

Due Date: **Wednesday, September 10, 2014 at 10:00 p.m.**

Overview

In the next two weeks, you will apply your knowledge of I/O and subroutines to develop an interactive program involving data manipulation, text output, and user input. The program simulates a popular type of book in which the reader encounters specific choices while reading through the story. The reader must decide on a particular choice, then flip to the page corresponding to that choice to continue the adventure. In our version of the game, the number of choices possible for any decision is limited to three. The point of this program is to give you hands-on experience with translating one type of data into another, to expose you to direct use of I/O device interfaces, and to help you make the connection between finite state machines and software.

You may want to refer back to first assignment for information about the weekly routine, the meaning of the challenge problems, instructions for checking out a copy, suggestions for tools, and instructions for handing in. Only details specific to this program are included in this document.

The Pieces

This week, you start with only a short piece of code that prints the low eight bits (one byte) of R3 as a two-digit hexadecimal number. You will modify and add to the given code to complete the assignment.

In programming studio, we will develop a set of subroutines that write a set of four hexadecimal numbers to the monitor based on direct interaction with the LC-3's DSR and DDR registers. Our goals will be threefold: first, to develop the necessary PRINT_CHAR subroutine; second, to wrap the PRINT_HEX_DIGIT code into a properly working subroutine; and, third, to write a subroutine that uses it to print four space-separated values to the monitor. This final subroutine may be called whatever you wish.

Outside of programming studio, you must write code to translate a database into a more useful format for use while playing the adventure game. (Precise details are given in the Details section below.) A given database consists of some number of records, which correspond to the pages of the virtual book. Each page (record) consists of a string and three page numbers corresponding to up to three possible choices allowed on that page. After you have performed the translation, you must then validate your processing by using the subroutines developed in programming studio to dump the contents of the array that you have built to the monitor.

Details

You will be given a text database starting at x5000. The database has a specific format, which is detailed below. You will need to write LC-3 assembly code to parse the text database to create an array that will help you to access the database for our adventure game.

The format of the database is as follows: Each "page" of the story is made up of 4 parts. The first part is a string, followed by 3 possible choices for the reader to make. Each page will include all four of these elements. The first string in the database starts at x5000 and holds ASCII characters in the usual format (8 MSB's are 0) in succession until a NUL value (ASCII x00) is found. The strings are of arbitrary non-zero length and NUL-terminated. The next three values immediately after the NUL are record numbers (stored as 16-bit two's complement format) for choices 1, 2 and 3 respectively. The next text record in the database is located directly after the three choice memory locations. The end of the database is marked by a single NUL character, so if you find that the memory location after the three choice values for a record holds NUL, you should stop processing. (As a result, records are not allowed to use empty strings so keep that fact in mind when you create your own tests.)

The array that you build up from the database must store four pieces of information for each text record in the database in sequential order, namely the address of the first character of the string, and then choice 1, 2 and 3 values for that string. The array that you build must start at location x4000, and should terminate when data for the last string of the text database has been stored.

Finally, your program should print the values stored in the array out to the console in the format shown below and halt. The values that we print are the page number (starting at 0), choice 1, choice 2 and choice 3 respectively. Example output is given below.

Example Database

Location	Data	Description
x5000	'H'	
x5001	'i'	
x5002	NUL	Signifies end of string
x5003	1	Choice 1
x5004	4	Choice 2
x5005	3	Choice 3
x5006	'D'	Beginning of next text record
...	...	
x5031	'M'	
x5032	'o'	
x5033	'm'	
x5034	NUL	Signifies end of string
x5035	2	Choice A
x5036	6	Choice B
x5037	-1	Choice C
x5038	NUL	End of database

Array Format

Location	Data	Description
x4000	x5000	Address of text string for record 0
x4001	1	Choice 1 for record 0
x4002	4	Choice 2 for record 0
x4003	3	Choice 3 for record 0
x4004	x5006	Address of text string for record 1
...	...	...
x4023	-1	Choice 3 for record 8

Example Output

```
00 01 04 03
01 05 02 08
...
08 02 06 FF
```

Testing

You should test your program thoroughly before handing in your solution. Remember, when testing your program, you need to set the relevant memory contents appropriately in the simulator.

We have given you two sample test inputs, test.asm and surreal.obj. Note that you do not have the ASM version of surreal! ("Surreal" is a fairly extensive game in which you play a 198 student trying to finish your programming assignment, but to get to play the game, you have to finish your code...) You will need to assemble the test.asm. For each test, we have provided a script file to execute your program with the test input—runtest and runsurreal—and a correct version of the output: testout and surrealout.

After successfully assembling all necessary files, run the following commands to test your code:

```
lc3sim -s runtest > mytestout
diff mytestout testout
```

Remember to debug your code before trying the tests! And remember that your final register values need not match those of the output that we provide, but all other outputs must match exactly.

Challenges for Program 2

The weekly MP's will be graded out of 100 points. Most weeks extra credit will be offered to give students an extra challenge and an opportunity to make up points lost elsewhere. Since these are extra credit, students are expected to complete these challenges without the assistance of TA's. Completing the extra credit should have no change on the functionality of the original assignment. If you choose to attempt the extra credit, please be sure to submit only one file but be sure that the extra credit portion does not somehow inadvertently alter the functionality of base submission. If you are going to attempt the extra credit, please list which sections of the extra credit you attempted in the introductory paragraph so we will know which extra credit to test for. Here are a few challenges for this week with their associated awards.

(1 point)
We implicitly assumed that page numbers fit in a byte. If a database contains more than 256 pages, print the following error message (exactly!) and do not print the list of page choices:
"Database invalid: contains more than 256 rooms!\n"

(2 points)
Given a database containing N pages, a choice C is valid if (and only if) C < N. Verify that all choices in the database are valid, and print the following error message (exactly, other than for the hex values) for each invalid choice: "Page 1A contains invalid choice 8C.\n" Do not print the normal list of page choices if the database contains invalid choices.

(10 points)
Even if the choices in a database are all valid, some of the pages may be unreachable. The game starts from page 0. Use the search ideas that we discussed in class to find all reachable pages (you'll want another array to keep track of which pages are reachable). If some of the pages in the database are NOT reachable, print an error message (exactly, other than for the hex value) for each unreachable page: "Page 3C is not reachable.\n" Do not print the normal list of page choices if the database contains unreachable pages.

(7 points) (REQUIRES PREVIOUS CHALLENGE)
If all choices are valid and all pages are reachable, print path information after the normal output. For each page, print a shortest path starting from page 00 and ending at that page, using the following format as an example for page 05: "00:13:2C:0F:05\n"

Checking Out a Copy:

A copy of the starting code has been placed in your Subversion repository. To check out a copy, *cd* to the directory in which you want your copy stored, then type:

```
svn co https://subversion.ews.illinois.edu/svn/fa14-ece198kl/[netid]/MP2
```

Replace "[netid]" with your NetID. Recall that the Subversion ("svn") checkout command ("co") makes a copy of the files stored in your repository in the directory in which you execute the command. The copy will be called "MP2" – type *cd MP2* to enter it.

Use your working copy of the lab to develop the code. Commit changes as you like, and make sure that you do a final commit once you have gotten everything working. **Commit a working copy and make a note of the version number before you try any of the challenges!**

With your assembly code, you must include a paragraph describing your approach as well as a table of registers and their contents. Avoid using R7 if possible, as any TRAP instructions (such as the OUT trap you will need to print to the monitor) will overwrite its contents and may confuse you.

Specifics:

- Your program must be written in LC-3 assembly language, and must be called prog2.asm, we will NOT grade files with any other name.
- Your code must begin at memory location x3000 (just don't change the code you're given in that sense).
- The last instruction executed by your program must be a HALT (TRAP x25).
- You must use the PRINT_HEX_DIGIT subroutine to print hex values to the monitor, and must not use any TRAPs in the PRINT_CHAR subroutine that PRINT_HEX_DIGIT must call. Note that the code given to you does not meet these constraints.
- You must use a subroutine to print four space-separated hex values from registers to the monitor by calling PRINT_HEX_DIGIT, the choice of subroutine name is yours; this code is to be developed during programming studio.
- All subroutine interfaces must be documented clearly, and the code must obey the documentation.
- You may assume that the database is valid for the purposes of the regular parts of the assignment. For the challenges, your code must handle those potential problems mentioned in that section.
- The text database's records are implicitly indexed from 0 onwards. Since your array should be in the same sequential order as the text database, they have the same implicit indices. The values in the choice 1,2 and 3 memory locations refer to a record index.
- Certain choice values are -1. These do not point to any record, but rather indicate that the choice is invalid. You do not need to treat these values in any special way for Program 2.
- You must end your output text lines by outputting a linefeed (ASCII x0A) character to the monitor using the PRINT_CHAR subroutine.
- Your array must begin at x4000, and the text database begins at x5000.
- Your output must match the desired format exactly, as shown in this specification and in the test files provided.
- You may not make assumptions about the initial contents of any register.
- Your code must be well-commented. Comments begin with a semicolon. Follow the commenting style of the code examples provided in class and in the textbook.

Grading Rubric:

Functionality (60%)

- 30% - program builds the correct array

- 30% - program prints the array values correctly

Style (25%)

- 5% - subroutine PRINT_CHAR defined and written properly
- 5% - subroutine PRINT_HEX_DIGIT defined and written properly
- 5% - subroutine to print four hex values defined and written properly
- 5% - code uses PRINT_CHAR and PRINT_HEX_DIGIT subroutines rather than OUT traps
- 5% - clear code structure used for parsing of text database

Comments, clarity, and write-up (15%)

- 5% - introductory paragraph clearly explaining program's purpose and approach used
- 5% - each subroutine specifies an interface definition and a table of registers
- 5% - code is clear and well-commented (every line)

Note that correct output (and the points awarded for it) also depends on building the array correctly.