

Lab 09: LC-3 Simulator and Programs II

(Due 11:59 PM, Mon Oct 26)

Note: The programs are getting a longer, so let's hand in your lab using Blackboard, using the digital dropbox. **Be sure to Add and Send your file.** Include comments that include your name and the assignment (Lab 9).

A. Why?

Reading and writing numbers is good.

B. Outcomes

By the end of the activity you should

- Have more practice using the LC-3 simulator.
- Be able to write instructions in LC-3 assembler.
- Be able to read a sequence of digits as ASCII characters and convert them to the equivalent nonnegative binary number.
- Be able to convert a nonnegative binary number to a string of ASCII characters, for printing.

C. In-Lab Practice

1. What are the assembler representations for the following operations?
 - (a) Make my program start at location x3000. [Hint: Read up on the .ORIG pseudo-instruction]
 - (b) Use AND to set R7 to 0.
 - (c) Use ADD to copy R7 to R6.
 - (d) Branch to the label TOP if the Z condition code is set.
 - (e) Define a label EIGHT for a word that contains "8".
 - (f) Read a character.
 - (g) Print a string.
 - (h) Halt execution.
 - (i) Hey, this is the end of my program file. [Hint: Read up on the .END pseudo-instruction.] (Note: This is not the same as "Halt execution" — halt is like returning to the operating system. "End of program" says this is the last line of your program.)
2. Here's some pseudocode for reading a sequence of ASCII digits and calculating the equivalent binary number:
 - Initialize the resulting number N to 0.

- Get the next character C (an ASCII digit, “0”, ..., “9”).
- While C existed
 - Calculate the binary number B that C represents: $B = C - \text{“0”}$.
 - Calculate $N = 10 * N + B$
 - Get the next character C

Hand-execute this algorithm on some sample inputs. (Say, “0”, “42”, and “13579”).

3. Here’s some pseudocode for taking a positive binary number N and creating the equivalent string S in memory:
- Let’s assume S is going to be stored in memory locations L – L’ (where L’ contains the terminating x00 character). Let P = L’.
 - Copy a terminating x00 character to *P (the location pointed to by P).
 - Decrement P.
 - Let X = N // X holds the part of N yet to be converted.
 - While X > 0
 - Calculate $X = 10 * Q + R$ where $0 < R < 10$
 - Calculate the ASCII character C that represents R: $C = R + \text{“0”}$.
 - Copy C to *P; decrement P.
 - Set X = Q
 - // X = 0 and P points to the first character of the string for N

Hand-execute this algorithm on some sample inputs. (Say “1”, “9”, “10”, and “2468”). Also, answer this question: Why did we assume $N > 0$? (What happens if $N < 0$ or $N = 0$?)

D. Questions to Hand In

You can write your program in binary or assembler, your choice. (Me personally, I’d write it in assembler, but your mileage may vary.) In any case, comment your program so that the TAs can understand it. (6 of the 24 points for this lab will be for readability.) You do not have to check for or handle bad input.

1. Write a program that reads in a nonnegative decimal number from the keyboard and calculates the equivalent binary number. Stop reading when you see a space. (A sample input: 0_ (where _ indicates a space).
2. Write a program that loads an value into R0 and calculates and prints its ASCII representation. The value may be zero or positive; assume it won’t be negative. Calculate the whole string and then use a single TRAP instruction to print it out.

One way to test your program is by hard-coding instructions to load a particular value into R0. E.g.

```
AND R0, R0, #0    ; R0 ← 0
ADD R0, R0, #15   ; R0 ← 15
... fall into the rest of your program ...
```

(To test your program with a different value, you edit your program, change the “add 15” to something else, then assemble and run your program.)

Another way to test your program is to combine it with the program from Question 1: You wind up with a program that reads a nonnegative integer and calculates the equivalent binary number, then does the exact opposite: it takes the binary number, calculates the equivalent string, and prints it out.