TRAPS and Multiple Files

Programming Lab #4
CEG 3310/5310: Computer Organization

PURPOSE

In this lab you will learn how you can use multiple files in assembly to implement more complicated functions.

ASSIGNMENT

Using the LC-3 simulator, you will construct an assembly-level program that prompts the user for a starting address (in hex) and an ending address (in hex). Your program will then output the contents of memory (in hex) between the provided ranges (inclusive).

This lab includes 4% of extra credit to your final class grade which is equivalent to over half of an entire lab grade. If you did not get the grade you were hoping for on Lab 3, this will definitely help.

```
Enter starting memory address:
x3000
Enter ending memory address:
x3003
Memory contents x3000 to x3003:
x3000   xF030
x3001   xB006
x3002   xF040
x3003   xB005
```

Example execution of the Memory Dump routine

I/O for this routine requires that we develop a routine to enable the input of a 4-digit hex value and a routine for displaying the contents of a 16-bit registers/memory location as a 4-digit hex value. We will implement each of these routines as TRAPs.

- Input (**Trap x40**): A Trap routine (invoked as TRAP x40) that reads a 4-digit hex from the keyboard and returns the value in R0. This trap may call other traps. You will develop this trap routine and locate it in memory at address x4000.
- Output (**Trap x41**): A Trap routine (invoked as TRAP x41) that displays the contents of R0 to the display as a 4-digit hex value. This routine should output exactly 5 characters: a leading "x" and the 4 hex digits. Do *not* display a carriage return/line feed/end-of-line as part of the trap call. This trap may also call other traps. You will develop this trap routine and locate it in memory at address x5000.

Develop these TRAP routines and the use them to implement a program to perform memory dumps (using the i/o format provided in the example above).

GETTING STARTED

You are provided 5 .asm files and you only have to modify 2 of them:

- main_lab4.asm
    - Do not modify this file
    - This file exists to run your trap calls x40 and x41
- trap_vector_input.asm
    - Do not modify this file
    - This file puts the address x4000 into the TRAP vector at location x40
- trap_vector_output.asm
    - Do not modify this file
    - This file puts the address x5000 into the TRAP vector at location x41
- trap_input.asm
    - This is where you write your PART 1 code
    - Use this file to write your hex character input
- trap_output.asm
    - This is where you write your PART 2 code
    - Use this file to write your memory dump output

To get started drag these .asm files into your simulator in this order:

1. trap_vector_input.asm
2. trap_vector_output.asm
3. trap_input.asm
4. trap_output.asm
5. main_lab4.asm

Without writing any code yet, you can continually press "Step" in your simulator to see how the unmodified works.

The first TRAP call (x40) takes your program to address x4000, once you implement code in x4000 you should have your starting address of your memory dump in R0.

The second TRAP call (x40) takes your program to address x4000, again, and this will put your ending address of your memory dump inside of R0.

The last TRAP call (x41) takes your program to address x5000, which will use addresses R0 for your starting address, and R1 for your ending address.

PART 1

To implement trap_input.asm you must do the following:

- You must read a users input of "x####" where the #'s are hex digits
- Once you have the 4 hex digits you must store the hex values in R0
- For example: if user types the ASCII characters "xA1B5" then at the end of your TRAP call R0 must contain the value "xA1B5"
- NOTE: This is very tricky! For example the ASCII value "A" will be read from the keyboard in HEX as "x0041", somehow you must convert this to "x000A". Also, the ASCII value "0" will be read from the keyboard in HEX as "x0030"
- So if the user enters "xA1B5" 5 different values will be read in this order:
    - x0078

    - x0041
    - x0031
    - x0042
    - x0035
    - x000A
- After all these inputs R0 must read:
    - xA1B5
- This TRAP call should function properly even after being called multiple times, you will need to call this TRAP function at least twice to complete this lab (to get your starting and ending addresses).
- EXTRA CREDIT: You must only implement reading UPPERCASE hex values. If you implement LOWERCASE letters as well, such as “xa1b5” then you will be given +1% extra credit to total class grade
- EXTRA CREDIT: You can assume that a user will always enter a valid input. You do not have to check for errors. For example if a user enters “xZ#5p” you can process it as if it was a valid input. If you do implement valid input checking you will be give +2% extra credit to total class grade

PART 2

To implement trap_output.asm you must do the following:

- Using R0 as your starting address and R1 as your ending address
- Cycle through memory addresses and display the contents to the user
- For example, if R0 contains x3000 and R1 contains x3003 then the display on the console should look as follows:

```
Memory contents x3000 to x3003:

x3000  xF030

x3001  xB006

x3002  xF040

x3003  xB005
```

- NOTE: This is tricky as well, you must convert an address/data inside register to a string. For example:
    - Address/Data in Register:
        - x3000
    - String that will get stored in memory:
        - x0078
        - x0033
        - x0030
        - x0030
        - x0030
        - x0000
- EXTRA CREDIT: You do not have to check if R1 > R0. If you do check R1 > R0 and display a message to the user “The starting address is smaller than the ending address!” then you will be given +1% extra credit to total class grade

GRADING

- Input TRAP implemented properly: 5 points

- Output TRAP implemented properly: 5 points
- Total: 10 points