Lab #4: TRAPs and Subroutines – Memory Dump
Computer Organization

PURPOSE

In this lab, students will gain familiarity with the use of low-level subroutines, caller/callee register saving policy, and TRAP calls.

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).

```
Enter starting memory address:
x3000
Enter ending memory address:
x3001
Memory contents x3000 to x3001:
x3000   xF030
x3001   xF025
```

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).

Note: The representation for output characters (ASCII) is different than the standard binary representation of the value. For example, you may find it useful to note that the ASCII representation for any single-digit value is #48 greater than the number itself. Thus, the ASCII representation of the character 0 has value #48 (x30) while the ASCII representation of the character 1 has value #49 (x31).

PROGRAM GRADING

Grades will be assigned out of 50 points as follows:

Input Routine (Trap x40):

- **10 points: 4-hex digit input:** Your project must include an input routine that reads a 4-hex digit input. It must properly translate the ASCII inputs into a 16-bit value returned in R0.
- **5 points: Error checking:** Your input routine should verify that the input consist of exactly 4 hexadecimal characters. It should accept uppercase ASCII characters for the hex digits A-F. If an error is detected, the routine must return the value x0000. *(You can earn 2 extra-credit points for correctly accepting either upper or lower case A-F digits.)*
- **5 points: TRAP implementation:** Your input routine should be implemented as a TRAP. Please locate this routine starting at address x4000 in memory. Please execute this routine as TRAP x40. *The TRAP routine should not have any unexpected side-effects!*

Output Routine (Trap x41):

- **10 points: 4-hex digit output:** Your project must include an output routine that properly displays for the user as a 4-hex digit value the contents of R0. The 4-digit value should have an 'x' preceding it to indicate that it is a hex value.
- **5 points: TRAP implementation:** Your output routine should be implemented as a TRAP. Please locate this routine starting at address x5000 in memory. Please execute this routine as TRAP x41. The *TRAP routine should not have any unexpected side-effects.*

Memory Dump (main) Routine:

- **5 points: Address input:** The main routine must prompt the user to input a starting and ending memory address. It should verify that the starting address is lower than the ending address, and prompt the user to reenter the values if this is not the case, or if either input value is not a proper 4-digit hex value.
- **5 points: Address output loop**: Your project must be able to properly output (to the console, in ASCII) the contents in the range of the specified start/end addresses. (See example under assignment description).

Overall:

- **5 points: Documentation**: Your code should be well documented and easy to follow. You do NOT have to comment every line of code, but you should have high-level comments for each 3-10 lines that represent a functional block of code. All major functional blocks should be commented. Symbols should be meaningful. Sample execution must demonstrate the functionality of your project.

DELIVERABLES

- You will need to implement and turn in **four** assembly language (.asm) files including: (1) **main.asm**: the "main" program which performs the overall memory dump functionality, (2) **input.asm**: the trap service routine for the input routine, (3) **output.asm**: the trap service routine for the output routine, (4) **tvp.asm**: contents for the trap vector table necessary to enable your routine. I will load assemble and load all four files before testing your routine. *No credit will be awarded for files that do not assemble.*