CIT 593 Homework 11: LC-4 C, Assembly, & I/O

Due Date: Thursday 11/19/2014 (@11:59pm) via canvas upload ONLY

Note: With LC-4, we have diverged a bit from the book at this point; carefully read the lecture slides for this week to help you with this homework. *Please read **Chapter 10** of the book, they will refer to items in the LC-3 CPU, where we are using the LC-4 CPU. There are some differences, but it is still worth the read.*

Assigned Problems (to be done individually, NOT group work):

1) Calling LC-4 TRAPs from C

Overview: For this problem you will write a C program to interface with the Assembly TRAPs that you authored in HW #8 & #9.

Prerequisite knowledge: From lecture you learned that the LC-4, C-compiler (lcc), can read in C-code and translate it into LC-4 Assembly-code. You have learned that functions in C have a unique name (like `pow()`, `mult()`, `main`). The functions are translated into Assembly subroutines with corresponding labels to match the function names. We have seen that when one function calls another (EXAMPLE: `main()` calls: `pow`), the compiler translates this into a JSR <label> (EXAMPLE: inside `main.asm` we saw: `JSR pow`).

If we wish to write C-code that calls a TRAP (instead of a normal subroutine), there are two small problems that we must solve. The compiler is "dumb" and only knows to generate a JSR instruction, instead of a proper TRAP instruction to invoke a TRAP. We *could* manually intervene, open up the assembly file that the LCC compiler produces and replace the JSR <TRAP> with a TRAP x## to invoke the TRAP we are interested in calling. But this brings us to the second *problem*…

The second issue revolves around how the C-compiler uses the stack to pass arguments to functions that it calls, allocates local variables for functions as well as store return values (as we painfully covered in lecture). We would like to call the TRAPs that you wrote in HW 8 & 9, namely: `TRAP_GETC`, `TRAP_PUTC`, etc., from our C-code. But if you recall, we used registers in the register file and not the stack to pass arguments/handle local variables/return values to/from the caller. So we couldn't easily manually intervene and replace the JSR <TRAP> with a TRAP x##, because the C-compiler would use the stack to pass data back & fourth, but we used registers for this purpose!

If we would like to call our existing TRAPs from C-code, we need to write a "wrapper" subroutine in assembly. A wrapper (like how a chewing gum wrapper hiding delicious gum inside paper, not *Eminem*!) is a normal assembly subroutine we can use JSR to call. Inside the wrapper subroutine, we can then call the desired TRAP with the appropriate TRAP x## instruction while passing arguments through R0-R6. This way the call to the TRAP is "wrapped" in a normal assembly subroutine that the C-compiler can properly call using JSR.

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

Re-download and expand the file: **CIT593_HW10_Helper_Files.zip** from HW #10. It contains:

my_first_c_program.c - c-program that calls TRAP_PUTC through a wrapper called: lc4_putc()
lc4_stdio.asm - contains the actual "wrapper" subroutines: lc4_getc, lc4_putc
user.asm - as shown in lecture, partitions user data memory & calls main
PennSim.jar - same old PennSim, just put in for convenience
my_first_c_program_script.txt - a script that **LINKS**, my_first_c_program with lc4_stdio.asm

Copy your **os.asm** from HW #9 into the directory you expanded CIT593_HW10…zip
Copy the LCC compiler (from HW #10) into the same directory

Open the file: **`my_first_c_program.c`**

- In HW #10 you noticed the "declaration" of the function: `lc4_putc()` in this file
- What you don't see is the "definition" of the function: `lc4_putc()`
- The 'definition' is the "meat", aka: the code that implements it, so where is it…

Compile the file: **`my_first_c_program.c`**

- Open the output of the LCC compiler: my_first_c_program.asm
- Examine the file, notice the declaration of the local variables
- Scroll down until you see a JSR to lc4_putc()
- Look at how the arguments are passed to lc4_putc() on the stack
- ***Notice, there is NO subroutine labeled: lc4_putc() in this file***

Open the file: **`lc4_stdio.asm`**

- Scroll down to the label: `lc4_putc`
- This is the "definition" (aka the "meat") of the function: `lc4_putc()`
- This is a LC-4 assembly subroutine that "wraps" a call to the TRAP: TRAP_PUTC
- From lecture, the code should look familiar, notice: prologue, function body, & epilogue
- The prologue is identical to lecture:
  - It saves return address of calling function (aka main), from (R7)
  - Saves main's frame pointer (R5), updates the STACK pointer (R6), and updates the FRAME pointer (R5)
- The function body is custom to call the TRAP itself!
  - It gets the arguments passed from main() off the stack (e.g.: a single ASCII letter)
  - It then copies that argument into R0
  - Then it calls the TRAP
  - The TRAP_PUTC is the same one you've used before (in os.asm), all it wants is a character to write to the ASCII display in R0
- The epilogue is just like lecture as well: restores the stack!

Open the file: **`my_first_c_program_script.txt`**

- Examine the following line:
  `as my_first_c_program user lc4_stdio my_first_c_program`
- This asks the assembler to "assemble" three files: my_first_c_program, user, lc4_stdio
- It also asks the assembler to "LINK" them into a file called: my_first_c_program.obj
- This basically means: take the contents of each file and make it one big file
- Then, when your subroutine: **main** (in **my_first_c_program.asm**) performs a JSR to a subroutine lc4_putc (in lc4_stdio.asm) the linking step ensures it is found
- This allows the C-compiler to be "dumb"; to merely issue "JSR" instructions to functions that may not even exist yet!

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

Assigned Problems (to be done individually, NOT group work):

1) Invoking TRAP_GETC from your C-program

Make a copy of: my_first_c_program.c, call it: my_sixth_c_program.c
Make a copy of: my_first_c_program_script.txt call it: my_sixth_c_program_script.txt

- Update the script file to assemble and load your eventual my_sixth_c_program.asm

Hopefully you've read the first two pages well; the intention of this first problem is to help you connect the dots in terms of how C/assembly/the stack/and I/O interconnect. Your job is to update **my_sixth_c_program.c**, and **lc4_stdio.asm** to complete wrapper function: **`lc4_getc()`**. You must finish the "wrapper" in stdio.asm and declare (***not define***) it in **my_sixth_c_program.c**. You can use the wrapper for **lc4_putc()** and the lecture slides as a model. Realize that in your os.asm you have the complete TRAP_GETC.

*Realize, that lc4_putc() copies an argument passed to it from the stack to R0 and then calls TRAP_PUTC. The wrapper you will write for TRAP_GETC, does the opposite. It does not take in a parameter from the user, but it **returns** a character from the keyboard to the user in R0. You must determine how to copy that value from R0 back into the stack, so that main() can use it.*

To test your wrapper, you will need to update **my_sixth_c_program.c**. Call the function you have declared, lc4_getc(). A character should be returned, use lc4_putc() to print out the character you've received. **Compile** my_sixth_c_program.c, load it into PennSim and test it out. The program should read a character from the LC-4's keyboard, and write it out to the ASCII display.

Helpful tip: You can see the "stack" anytime you like while your program is running. Simply scroll down to memory address: x7FFF anytime and watch the stack grown and shrink at runtime.

For this part you should turn in 5 files: my_sixth_c_program.c , my_sixth_c_program_script.txt, lc4_stdio.asm, os.asm, user.asm

2) Wrapping the remaining traps for user in your C-programs

Make a copy of: my_sixth_c_program.c, call it: my_seventh_c_program.c
Make a copy of: my_sixth_c_program_script.txt call it: my_seventh_c_program_script.txt

- Update the script file to assemble and load your eventual my_seventh_c_program.asm

Modify lc4_stdio.asm, to define additional wrapper functions for TRAP_VIDEO_COLOR and TRAP_VIDEO_BOX. Name the wrapper function: **lc4_video_color** and **lc4_video_box**. Declare these wrapper functions at the top of my_seventh_c_program.c. Use the same procedure as you did earlier, except realize: these traps most likely modify registers R5 and R6. Recall that you may use the stack to save register values (you may store them in the "temporaries" section of the activation record).

Update my_seventh_c_program's main() to call each of these traps. First call lc4_video_color and paint the screen **green**. Next call lc4_video_box and place a red box at coordinates: 64, 64.

For this part you should turn in 5 files: my_seventh_c_program.c , my_seventh_c_program_script.txt, lc4_stdio.asm, os.asm, user.asm

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

3) Let's play a game!

Make a copy of: my_seventh_c_program.c, call it: my_eigth_c_program.c
Make a copy of: my_seventh_c_program_script.txt call it: my_eigth_c_program_script.txt

- Update the script file to assemble and load your eventual my_eigth_c_program.asm

You now have a complete set of I/O functions accessible through C. We can now operate at a new level of abstraction, simply calling functions to direct I/O. In this section we're going to create a simplified version of the popular video game called **Tetris (http://www.freetetris.org/game.php)**. In this game, we will simply have boxes scroll down from the top of the screen until they hit the bottom, that you can control using the keyboard.

Here are the new rules of play:

1) Draw a **RED** 10x10 box starting in row 0, but centered on a black screen
2) Check if the user presses keys: j (left), k (right), or m (down)
    a. If they press a key, move the box left, right, or down by 10 pixels
        i. Do not let the user move the box off the screen
        ii. After responding to their key, move the box down 10 pixels (no matter what key they've pressed)
    b. If they don't press a key, move the box down by 10 pixels
3) Keep track of the boxes coordinates on the screen
    a. When the box reaches the bottom 10 pixels of the screen, leave it on the screen
4) Start all over again with step 1) above:
    a. When the second box reaches the bottom of the screen it is all right to allow it to "overwrite" the previous box
    b. Stop the game when you have allowed 10 boxes to reach the bottom of the screen (no matter where the boxes have landed)
    c. Print "GAME OVER!" to the ASCII screen the game is complete

For this part you should turn in 5 files: my_eigth_c_program.c , my_eight_c_program_script.txt, lc4_stdio.asm, os.asm, user.asm

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

1) Extra Credit: Let's enhance our game!

Make a copy of: my_eigth_c_program.c, call it: my_ninth_c_program.c
Make a copy of: my_eigth_c_program_script.txt call it: my_ninth_c_program_script.txt

- Update the script file to assemble and load your eventual my_ninth_c_program.asm

Enhance the game you made in part 3 above by adding these features:

1) Change the colors of the boxes that fall (red, green, blue, would be nice)
2) When a box falls to the bottom of the screen, do not allow it to overwrite existing boxes
3) When a "row" of 10x10 boxes is formed on the bottom row of the screen, erase it!

Steps 1-3 will get you the extra credit, these steps are optional, but I'll be impressed:

4) Show a "score" to the user on the ASCII display. Give them 10 points for each row that gets erased.
5) Prompt the user for their name and print their name with their score after they enter it!

For this part you should turn in 5 files: my_ninth_c_program.c , my_ninth_c_program_script.txt, lc4_stdio.asm, os.asm, user.asm

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

2) Extra Credit: Working with Pointers and TRAPS

You have noticed that lc4_putc() is quite limiting. You can only output 1 character at a time. You created a function alled: lc4_put_str() in HW 10, but that just calls lc4_putc() in a loop. Create a new TRAP in os.asm called: TRAP_PUTS that will take as input a "string" of characters terminated by a 0, and output them to the ASCII display. To do this, TRAP_PUTS should take in as input: R0, which should contain the address in data memory of the first character in the string. Each subsequent character should be stored in consecutive rows in data memory. The last row of data memory after the characters in the string, should be filled with a 0. The 0 tell us "where" the string actually ends, we call it NULL termination.

Next, update lc4_stdio.asm to create wrapper for this trap, call it lc4_puts. Your wrapper for this function should take as input a pointer to an array of chars. Pass string1 and string2 to your new function and have them displayed on the screen. Yes, we will examine the contents of your extra credit work to ensure you properly created lc4_puts()!

For this part you should turn in 5 files: my_tenth_c_program.c , my_tenth_c_program_script.txt, lc4_stdio.asm, os.asm, user.asm

CIT 593 Homework 11: LC-4 C, Assembly, & I/O

Directions on how to submit your work:

- Create a single zip file called: **LAST_FIRST_HW11.zip**
- The zip file should contain at least 9 files, at most 13 files named in this assignment.
- There should not be any sub-directories within your zip file.

As an example, I would turn in the following SINGLE zip file:

FARMER_THOMAS_HW11.zip

This single zip file would contain at minimum **9 files**, at maximum *13 files (if you do e.c.):*

os.asm
user.asm
lc4_stdio.asm
my_sixth_c_program.c
my_sixth_c_program_script.txt
my_seventh_c_program.c
my_seventh_c_program_script.txt
my_eigth_c_program.c
my_eigth_c_program_script.txt
my_ninth_c_program.c
my_ninth_c_program_script.txt
my_tenth_c_program.c
my_tenth_c_program_script.txt

You will then upload ONLY 1 file to canvas: **FARMER_THOMAS_HW11.zip**

- DO NOT TURN IN ANY **.obj**, **PennSim.jar** , **or the LCC compiler files!!**
- Make certain that you submit the latest versions of your code.
- Submitting using any other compression type (.RAR, TAR, GZIP) will be rejected.

Paper/Email submissions will not be accepted for this assignment.