CS061 – Lab 6

More subroutines!

1 High Level Description

The purpose of this lab is to train you to juggle multiple subroutines at a time!

2 Our Objectives for This Week

1. Exercise 1 ~ Re-package your assignment 3 code as a subroutine
2. Exercise 2 ~ Write a subroutine to capture a 16-bit binary number from the user
3.
4. Exercise 3 ~ Refine your ex. 2 subroutine with with input validation

3.1 Exercises

Exercise 1

Recall that exercise 4 from lab **4** created an array of the first 10 powers of 2 {2^0, 2^1, 2^2, …, 2^9}, and then attempted to print these values to console as though they were ascii characters - with interesting results! *(If you don't remember that long-ago exercise, go back & run lab4_ex4.asm again!)*
But then in programming assignment 3, you figured out how to take the 16-bit value in a register and print to console the corresponding 16 ascii 1's and 0's, with correct spacing.

Your first lab 6 exercise is to take that assn 3 code and turn it into a subroutine:

Copy your lab4 ex. 4 code into lab6_ex1.asm
Now take your assn 3 code and modify it as needed to transform it into a subroutine.

> *The main difference between your assn 3 and your new subroutine is that the sub will NOT be storing the binary value to be printed - that's the point of making it a subroutine!*
> *Instead, when you invoke the sub, pass in via R6 the address of the value you want printed.*

Use your lab 5 subroutine headers as a guide for your new sub header, making sure you clearly specify all sub details - name, "parameter" register, post-condition, "return" register(s).
Add the new subroutine into your lab6_ex1.asm file, with its own .orig
(use the suggested locations for subs: x3200, x3400, etc)

Finally, inside the display loop of the original lab4_ex4 code (which is now your "main"), modify the output steps by:

- getting the *address* of each power of 2 in R6 (instead of getting the *value* into R0)
- replace OUT with an invocation of your new subroutine.

The end result will be 10 lines of output *(because the subroutine will be invoked 10 times, once per iteration)*, representing the first 10 powers of 2 as 16-bit binary numbers displayed as ascii 1's and 0's.

Exercise 2

Write the *converse* of a binary printing subroutine - that is, write a binary reading subroutine:
First, the subroutine will prompt the user to enter a 16-bit 2's complement binary number from the keyboard: the user will enter 'b' followed by exactly sixteen 1's and 0's: e.g. "`b0001001000110100`" *(no spaces on input)*
Your subroutine will do the following:

1. User enters a binary number as a sequence of 17 ascii characters: `b0010010001101000`
2. This input is transformed into a single 16-bit value, which is stored in R2 (the return register).

Again, make sure your subroutine header is properly constructed.

Your "main" can now:

- store the value in R2 to a labelled address in memory (can be local or remote, as you wish). Capture that address in R6.
- invoke the subroutine you built in exercise 1 to print the value stored at the address in R6 to the console to check your work.

"Binary read" Algorithm:

```
R2          <-- 0 (set to 0 - DO NOT load a hard-coded zero from memory!!)
counter     <-- 16
R0          <--  get input from user (the initial 'b') and ignore it
do
{
        ; that's your job :)
        ; HINT: the 4-bit binary number b1011 is (b101 * #2 + 1)
} while ( counter > 0 );
```

Once again, you might want to consult the AL control structure tutorials

Exercise 3

Enhance exercise 2 so that it now performs *input validation*:

- If the first character entered is not 'b', the program should output an error message and start over.
- After that *(i.e. inside the loop)*, if a SPACE is ever entered, the program should simply echo it and continue - i.e. spaces are accepted but ignored in the conversion algorithm.
- If *any* character other than '1', '0' or SPACE is entered inside the loop, the program should output an error message and ask for a valid character - i.e. it should keep all 1's & 0's received so far, and keep looping until it gets a valid '0', '1' or space, and then continue.

Exercise 4

Create a subroutine that turns a single zeros into one and one into zero. This should process one at a time. Do this for all elements

Example

1001 1110 1101 1101 -> 0110 0001 0010 0010

The subroutine should only take one ELEMENT

Example

flip_bit(0) -> 1

3.2 Submission

Demo your lab exercises to your TA ***before you leave lab***.

If you are unable to complete all exercises in lab, show your TA how far you got, and request permission to complete it after lab.

Your TA will usually give you partial credit for what you have done, and allow you to complete & demo the rest later for full credit, so long as you have worked at it seriously for the full 3 hours.

When you're done, demo it to any of the TAs in office hours ***before your next lab***.

Office hours are posted on Piazza, under the "Staff" tab.