ECE 252 - Homework 8

Written (problems 1 & 2) Due in Discussion Wed Dec. 7th

Code (problem 3) Due Thurs. Dec 8th at Midnight (11:59 PM)

Instructions: You should do this homework in your group assigned to you in your 252 section. You should hand in ONE copy of the homework that lists the common section number and names and UW ID numbers of all students. You should **staple** multiple pages together.

Warning: Most homeworks will use questions from your textbook, Patt and Patel's *Introduction to Computing Systems*, which we abbreviate (*ItCS*).

First contact for questions is TA Dustin Kreft (dkreft@wisc.edu)

Problem 1 (4 points): What does the following program do?

Give a BRIEF description of what the program checks for, as well as, what I/O is occurring with in the system and what is that I/O?

```
.ORIG    x3000
         LD R0, ASCII
         LD R1, NEG

AGAIN    LDI   R2, DSR
         BRzp  AGAIN
         STI   R0, DDR
         ADD   R0, R0, #-1
         ADD   R3, R0, R1
         BRp   AGAIN
         HALT

ASCII    .FILL x0039
NEG      .FILL xFFD1
DSR      .FILL xFE04   ; Address of DSR
DDR      .FILL xFE06   ; Address of DDR

         .END
```

Solution: The program checks for the status of the display register to unsure the display is ready. The I/O that is occurring in the system is an output to the display/console which is the numbers 9876543210.

Problem 2 (4 points): Explain why the following program will not work and what the output should be. Correct the program so it works correctly.

```
.ORIG    x3000
         JSR   A
         OUT
         BRnzp DONE

A        AND   R0, R0, 0
         ADD   R0, R0, 5
         JSR   B
         RET

DONE     HALT

ASCII    .FILL x0030

B        LD    R1, ASCII
         ADD   R0, R0, R1
         RET
         .END
```

Solution: The program will not work because of the second jump, JSR B, this will over write the R7 register and cause an infinite loop. There are several ways to correct the problem. 1) Store R7 before making another jump then recall it. 2) Add the code inline for the B routine. Or any other form that works. The output should be '5'.

Problem 3 (17 points):

You will complete the code for your project to do the tasks listed below.

1. **Complete GET_CHAR routine (3 points):**
   This routine doesn't take any input but returns the ASCII value of the character received in register R0. Essentially, this task can be broken down into following parts:
   a. Check whether any input was received or not (i.e. using KBSR register).
   b. If yes, then read the value from KBDR register and write into R0.
   c. If no character was received, then write 0 into R0.

   KBSR and KBDR memory locations that contain the memory-mapped register addresses. Please read section 8.2 (p202) from ItCS.

2. At the start of SCROLL_LOOP, poll I/O using the above mentioned routine and do the following operations if input is received:
   a. Accept command (3 points): w and z are the valid inputs (both in lower and upper case). If no input is received, then branch to NO_KEY_PRESSED label. If 'w' or 'W' is received as input, then branch to W_PRESSED label. Else if 'z' or 'Z' is received, then branch to Z_PRESSED label. Else if any other character was received, then display the error message INPUT_ERROR_MSG, which is "\nWrong Input\n" and branch to NO_KEY_PRESSED. Also echo the character to the console.

   Perform the following actions for the given command:
   b. W or w (3 points): This should change the color in which the string is being displayed. Array COLOR_LIST has the values of colors to use. Every time this command is entered, the color of the string should be changed to the next color in the list. If the next color is zero, then the color should be reset to the first value (BLUE). Also, display the message COLOR_MSG, which is "\nColor changed\n", every time the color changes. This piece of code should be entered under label W_PRESSED. Unconditionally branch to NO_KEY_PRESSED label when done.

   There are 2 memory locations tracking the current color value and address: CURR_COLOR and CURR_COLOR_INDEX. CURR_COLOR is used in DRAW_STRING to determine the current color in which the string is drawn. CURR_COLOR_INDEX holds the offset of current color from the label COLOR_LIST and makes it easier to change to next color.
   Current contents of COLOR_LIST are given below. You are free to change the values of the colors as you please. For changing to your favorite color, refer to PennSim Manual.

```
COLOR_LIST:
   .FILL 0x001F; BLUE
```

```
.FILL 0x7FFF; WHITE
.FILL 0x7C00; RED
.FILL 0x03E0; GREEN
.FILL 0x3466; Puce
.FILL 25000 ; Random
.FILL 0x0000
```

c. **Z or z (8 points):** This command lets you change the string being displayed. Firstly, you should prompt for a string by displaying the message PROMPT_FOR_STRING, which is "\nEnter a string\n". The program should start with this. The input string from the user can contain the characters A-Z (ASCII 65-91), a-z (ASCII 97-112), enter (ASCII 13) and space (ASCII 32). You should receive input characters from console and store it at memory location labeled as STRING.

   As the characters are being entered, check if they are valid or not. If yes, then keep storing the characters until the enter key is received. You should not store enter, instead terminate the string with a null character. If a valid character is not received, then display the error message STRING_ERR_MSG, which is "\nInvalid character in string\n", and again prompt for a string.

   Check for the number of characters being entered by the user, and if it exceeds a MAX_STRING_LEN value then stop accepting input and prompt the user saying MAX_LEN_WARN_MSG which is "\nString length limit reached\n" and continue to display the string entered. (MAX_LEN_WARN_MSG + 1) memory locations have been reserved for string starting at memory location labeled as STRING.

   Once a valid string is received, echo the string being displayed to the console and jump to NO_KEY_PRESSED label to display the string. Since we can display only A-Z and space, characters a-z should be changed to upper case before being displayed. Don't forget to change the case before storing it to memory or else the character won't be displayed (why?).

   Before start of scroll loop, you should jump to Z_PRESSED label in order to prompt user for the string to be displayed. And then at the beginning of the Scroll loop, you should check if any input was received or not and then take proper action.

Use "TRAP x21" (or OUT) to display a character on console. It takes in ASCII value of the char to be drawn as input in R0. To display a string on a console, use TRAP x22 (or PUTS). It takes an address from where the null-terminated string starts as

input in R0. Both of these traps do not give any output. For more information on character output read section 8.3 (p204) of ItCS.

The complete Scroll Loop has been provided. It uses the DRAW_STRING subroutine to draw the string stored starting at memory location labeled as STRING. It doesn't take any inputs in registers but from memory locations LOCAL_X, LOCAL_Y, CURR_COLOR and STRING. The wrap logic in Scroll Loop has also been provided to you.

Note that the GET_CHAR routine is non-blocking, i.e. it'll poll only once. Hence while receiving the string characters as input, you need to keep polling (or in other words, repeatedly call GET_CHAR) till you receive a character and then only proceed to the error check and storing in memory. If you are not careful, you'll end up getting invalid characters since the input would be 0x0.

Summarizing, CURR_COLOR tracks the current color of string and can be changed by command 'w' or 'W'. STRING holds the actual string being displayed. The characters in the string can be changed on pressing 'z' or 'Z'. Maximum length of string being entered is available in MAX_STRING_LEN. Also, the string needs to be entered at the start of the program.

Also, note that lc3os has been further modified for Homework 8 to incorporate color. Hence, it is different from the previous homeworks. Make sure to load the homework 8 version (lc3os_hw8.asm) or else your code for color change might not work. You should also load the bitmap file (hw8_bitmap.asm) before testing your program. The bitmap file is same as homework 7 though.

Sample console output has been provided below for your assistance. The Strings in yellow are info messages, in sky blue are input received echoed back to console, in red are the error messages and in green are the strings being currently displayed.

```
Enter a string
testing the limi
String length limit reached
TESTING THE LIMw
Color changed
w
Color changed
y
Wrong Input
x
Wrong Input
a
Wrong Input
z
```

```
Enter a string
A Valid String
A VALID STRINGx
Wrong Input
w
Color changed
z
Enter a string
invalid 1
Invalid character in string

Enter a string
invalid again 2
Invalid character in string

Enter a string
invalid =
Invalid character in string

Enter a string
valid this time
VALID THIS TIME
```

Extra Credit (5 points):

For extra credit, you can implement some features on your own. Some of the other features may include:

- Incrementing y by 1 and wrapping (i.e. start from top) once you've reached at bottom
- Decrementing y by 1 and wrapping (i.e. start from bottom) once you've reached on top
- Change the speed of scrolling
- Reverse scrolling
- User can enter a "?" and console will display the list of valid commands
- Etc.

A write up should define and explain the feature(s) clearly and what character is being used to invoke the feature.