Implementing a three function calculator

Objectives
The purpose of this lab is to write a program to implement a simple calculator with add, subtract and multiply functions. This lab builds on the knowledge gained from the previous lab to read the keypad and output characters to the LCD screen.

Turn-In Requirements:
1. Pages 2, 3, 4 and answers to the 5 questions as specified in the lab policies handout.

References

Equipment for this lab:
• 68HC11 trainer kit, to include 68HC11 EVBU and prototyping strips
• IBM compatible PC to connect to the trainer kit via an RS-232 serial cable
Laboratory Exercise

Name: _______________________________ Signature: ___________________________

Name: _______________________________ Signature: ___________________________

Notes

- In this lab, all numbers such as addresses and data are given in hexadecimal format (“hex”) unless otherwise indicated. In completing the lab, record all information in hexadecimal unless directed differently.

- Make sure your development EVBU board is connected to power (green LED on board ON) and the serial port of the EVBU is connected to the serial port of your development PC containing the AXIDE software, configured to the correct port at the correct baud rate, etc.

- Three additional subroutines available on the EE-333301 Web site under RESOURCES. The file name is Lab2.rtf. Copy these files to the end of the echo_1.RTF file used in Lab#1, and rename the file to calculator.RTF.

Three function calculator program

In this lab you will write a program to implement a simple 3 function calculator. The keys corresponding to the functions are as indicated below.

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Add</td>
</tr>
<tr>
<td>B</td>
<td>Subtract</td>
</tr>
<tr>
<td>C</td>
<td>Multiply</td>
</tr>
</tbody>
</table>

Use ‘#’ key as the ‘=’ key in a standard calculator. The input numbers to the program are DECIMAL numbers. Following are few examples.

- 123A48# is equivalent to 123 + 48 =
- 253B129# is equivalent to 253 – 129 =
- 59C33# is equivalent to 59 * 33 =

All the input numbers must be less than 255, and the output numbers will be in the range $0 – 65535$. Your program should accept inputs from the keypad and when ‘#’ key is pressed, print out in the next line the equivalent input (as shown above) and the result after the ‘=’ sign. Use ‘*’ key to clear the LCD display. Following are a few examples.

- 134A193# input to your program
  $34 + 193 = 327$ output of your program

- 200B253# input to your program
  $200 – 253 = -53$ output of your program

- 204B30# input to your program
  $204 – 30 = 174$ output of your program
The keypad input and LCD output program used in the previous lab can be used to read and display inputs. For your convenience, 3 additional subroutines are provided for this class. They are:

1. **ascii_to_dec**: This subroutine converts a ascii string pointed by register X to a decimal number. The subroutine will exit when the first non numeric ascii character (i.e. not in the range ‘0’ – ‘9’) is encountered. The result of the conversion is in acc A. The number converted will be in the range 0-255\textsubscript{10}. If the first character is a non numeric, the acc A will still return 0. When the subroutine exits, register X points to the non numeric character.

2. **dec_to_lcd**: This subroutine sends the number contained in acc D as an unsigned decimal number to the LCD display. The range of the number is 0 – 65535\textsubscript{10}.

3. **is_numeric**: This subroutine checks to see if a given ascii character is in the range ‘0’ – ‘9’.

The following algorithm is suggested for your program. However, you are free to use any other algorithm.

The first phase is reading and storing the input and recognizing the ‘#’.

- Allocate a memory location to store a string, or an array of characters (one character after the other).
- As characters are read form the keypad, store them in the reserved location as a string.
- While you are reading characters, if the ‘*’ is detected, then clear the screen and start over.
- If the character ‘#’ is detected then the input phase is over.

The next phase is the processing of the input string.

- Scan the input string from left to right until a non numeric character is encountered (this is the operator ‘A’, ‘B’ or ‘C’). Store this character in a location called “operator”.
- While you are doing that, it is a good idea to note the location of the start of the second operand (or the number).
- Use ‘ascii_to_dec’ subroutine to convert the two operands (numbers) before and after the ‘operator’ and store in locations ‘number1’ and ‘number2’.
- Perform the appropriate operation on the two numbers depending on the operator.
- In the next line, output the properly formatted result using ‘dec_to_lcd’ subroutine.

**Requirements of the program:**

1. It should produce correct output. The ‘subtract’ operation should be able to produce ‘negative’ results.
2. If the ‘*’ key is pressed at any time, the screen should be cleared, and the program should be ready to accept the next input.
3. Demonstrate to the T. A. that you program is running. T. A. Initials:______________________
**Do the Following to document your program** (Note: comments, structure, and readability counts!!!)

1. Hand in a complete *Flowchart* and for the entire program (*calculator*.RTF).

2. Hand in the listing of the complete, well-commented source-code that *matches the flowchart*. Remember, if the TA can not read your code with “reasonable” effort, then it is wrong.

When you turn in the lab report, please answer the following questions. Provide flow charts and/or assembly language programs as necessary to explain your answers.

Questions

1. Where is the ‘reset’ vector of the 68HC11 located? Suppose your program is located at address $675f$, what do you need to do to get the 68HC11 to execute your program?

2. The calculator program you wrote doesn’t detect any syntax errors. For example the following input is incorrect.

   123A34B56#

   How could you modify the algorithm to detect such an error?

3. How do you modify ‘ascii_to_dec’ subroutine to return an error if the first character it encounters is non numeric?

4. If the input to ‘ascii_to_dec’ subroutine is $312_{10}$, what will acc A contain when the program exits the subroutine?

5. If the ‘dec_to_lcd’ subroutine interprets the number in acc D as a signed 16 bit number, what will it print if acc D has the following:

   - $10675_{10}$
   - $32768_{10}$
   - $62453_{10}$