Lab 3: Subroutines and Input/output

ECE-3731

INTRODUCTION

A microprocessor, unlike a personal computer, is rarely used as a stand-alone machine. It is more often used to control or communicate with external hardware, for which its capabilities are well suited.

There are two main ways for a microprocessor to communicate with external hardware. Serial communication communicates one bit at a time, and as communication speeds increase, serial communication is becoming increasingly popular. It is cheaper, as it requires less hardware. Many PCs use serial links with the modem and mouse.

In parallel communications, several bits are sent simultaneously. Theoretically, this allows for faster transfer of data. However, parallel communications is more expensive; it requires nine lines for one-way communication, as opposed to two for serial ports. It is becoming less popular. It is still useful on the HCS12, though.

SUBROUTINES

Writing modular code is an important programming technique. When programs get very large and complicated, it is useful to break them down into smaller, more manageable sections. Working with smaller sections makes it far easier to organize, write, and debug the program. This is called modular programming – instead of tackling a huge problem all at once, we work on small modules, and then put the whole thing together at the end.

In a C program, each module is usually a function. A function does a particular task for us, and we can just call that function anytime we need to perform that task. In assembly programming on the HC12, we use subroutines (also called functions) to perform a task.

Subroutines are pieces of code that can be used again and again in a program. They are program modules that are logically separate and independent of the main program, and form the bulk of most commercial programs. You will use them a lot, so it is important that you understand them. Subroutines ALWAYS require the stack (for storing the PC at the very least), and the stack MUST be initialized.

A subroutine can be thought of as a program within a program. To use a subroutine, we have to do two things, call the subroutine from the main program, and write the subroutine itself.

Calling a subroutine from the main program is simple, the command is as follows:

```
                    JSR     ROUTINE1
```

where ROUTINE1 is the name of the subroutine.

Write the subroutine as a separate block of code. Subroutines should be placed after the main code in your program. (The main code of the program must be an endless loop or end in an endless loop waiting for reset.) The first line of the subroutine should be given a label. The subroutine should end with the command RTS or return from subroutine. Remember: "A subroutine begins with a label and ends with RTS".

This is what actually happens when you call a subroutine:

At the JSR (Jump to subroutine) instruction, the program counter gets saved to the stack, and the PC is then loaded with the subroutine address.
The subroutine executes.
At the RTS command, the PC is retrieved from the stack, and the program operation resumes where it left off before the subroutine.

SUBROUTINE BANNER COMMENTS

You must write a subroutine banner comments section above each subroutine that you use in your code. It should have the following information:

Name	Example
Purpose	To demonstrate a subroutine table
Input	A number in register A
Output	(Number + 5) in A
Registers affected	A
Memory Affected	Location $D002

At the RTS command, the PC is retrieved from the stack, **and the program operation resumes where it left off before the subroutine call**.
In this lab, the main program receives a single character input from the terminal that represents a menu selection corresponding to an action to be performed.
The required action will be performed using a subroutine. I am well aware that this can be done without subroutines, but the purpose of this lab is to teach you how to use subroutines.

Terminal, I/O:

The following functions are available for the terminal:

TermInit (initialize the terminal), putchar(character in B printed to terminal), getchar(character from keyboard (terminal) to B register), out2hex(print two ascii characters representing the hex digits in B), printf(print string of characters pointed to by D register)

Pre-Lab Quetions:

1) Sketch the flowcharts of the 7 cases which are described in Lab assignment.
2) Download the Sample Project “ParsingLoopASM.zip” from canvas (from the same webpage you downloaded this experiment), debug it and sketch the flowchart of this sample project.

LAB ASSIGNMENT

Subroutine calls – keyboard/terminal:

To be done in Assembly

In this program, you need write a program in which the user responds to a menu of choices by typing a single character.

Your program should, *in an infinite loop*, do the followingdepending on the user input:
It should begin by printing the prompt string:
"Make a selection: V, W,A,D,4,2, P: "
--Also, the following actions are performed in response to menu item selection:

(a) User presses V
The main program calls a subroutine to handle this action.

The subroutine prompts the user to enter an 8-bit integer value from the keyboard.
The value entered is stored in a memory location named OP1.
The value entered is also returned to the main program in register A.
The **main program** prints **the value returned in A register** to the terminal. e.g. if the value in A is $2F it should print:
OP1 = $2F

(b) User presses W
The main program calls a subroutine to handle this action.

The subroutine prompts the user to enter an 8-bit integer value from the keyboard.
The value entered is stored in a memory location named OP2.
OP2 is allocated in memory so that it immediately follows OP1.
The value entered is also returned to the main program in register A.
The **main program** prints **the valuereturned in A register** to the terminal. e.g. if the value in A is $2F it should print:
OP2 = $2F

(c) User presses A

The main program calls a subroutine to handle this action.

The subroutine adds the current value of OP1 to the current value of OP2.
The subroutine prints a line showing the addition result.
e.g. $2A + $52 = $7C
If the addition result is invalid assuming unsigned operands it also prints "invalid unsigned".
If the addition result is invalid assuming signed operands it also prints "invalid signed".

(d) User presses D:

The main program calls a subroutine to handle this action.

The subroutine obtains the current value in OP1 converts it to decimal and displays it on screen. The function should print the original value followed by the decimal value on the same line and return. If the value in OP1 is $1B, then the output should be,
$1B = 27

(e) User presses 4

The main program calls a subroutine to handle this action.

The subroutine prints the current value of the four hex digits in memory beginning at the address of OP1 to the terminal.
e.g. If OP1 is $2A and OP2 is $3B it prints
$2A3B

(f) User presses 2

The main program calls a subroutine to handle this action.

The subroutine assumes that the address of the memory location that it modifies is passed in X register.
The value is modified by doubling it.
In this case the main program will pass the address of OP1 in X register to the subroutine.
In this way the contents of OP1 will be replaced by double the value.
[**Note**: The memory address must be passed using X register.]

(g) User presses P
The main program calls a subroutine to handle this action.

The subroutine displays the value of OP1 on PORTB.

Post Lab Questions:

1) Write a program which prompts user to enter a character, if the user presses 'W', the programcalls a subroutine which loads a value $3D in register 'A', XOR's register A with $6F and displays the result in hexadecimal result on screen.

2) Write a program which prompts user to enter a character, if the user presses 'X', the program calls a subroutine which loads a value $3D in register 'A',logically shift left register A and displays the result in hexadecimal result on screen.

3) Write a program which prompts user to enter a character, if the user presses 'Y', the programcalls a subroutine which loads a value $3B in register 'A',arithmetically shifts right register A and displays the result in hexadecimal result on screen.

4) For post-lab question 1,2 and 3, show your working on paper. Also attach the screen shots after execution of the programs, do they match?

5) What is the difference between LSLA and ASLA? Explain with example.

6) Write a program which prompts user to enter a character, if the user presses 'Z', the program asks user to enter a number. In this case the program implements **5-input majority voting logic which is explained as follows**.
The value of the number represented by the five leftmost bits of Register-B is tested. If the number of logic '1' bits (in the 5 leftmost bits) is greater than the number of logic '0' bits, the program should output: "*The number of 1's in the 5 left most bits is greater than the number of 0's*". If the number of logic '0' bits (in the 5 leftmost bits) is greater than the number of logic '1' bits, the program should output: "*The number of 0's in the 5 left most bits is greater than the number of 1's*"