# ECE 375 LAB 5

Large Number Arithmetic

**Lab Time: Wednesday 5-7pm**

*Rhea Mae V. Edwards*

*Jackson Neff*

## Introduction

The purpose of this lab is to help us further understand the concept and doing of manipulating and handling large (greater the 8-bit) number through the use of arithmetic/ALU instructions, with the process of creating and handling functions and subroutines. Then we are able to verify the correctness of our program with simulations.

We are instructed to learn and understand these concepts by creating a 16-bit addition, 16-bit subtraction, and a 24-bit multiplication between various pairs of numbers for each operation.

## Program Overview

… //explain the overall general organization of the program

Besides the standard INIT and MAIN routines within the program, six additional routines were created and used. Along with the MUL16 subroutine being provided in the skeleton file, the ADD16, SUB16, and MUL24 routines all preform various arithmetic operations on a pair numbers throughout the program. The MUL16 subroutine multiples two 16-bit numbers, whereas the MUL24 subroutine multiples two 24-bit numbers. The ADD16 subroutine adds two 16-bit numbers together, and the SUB16 subroutine subtracts two 16-bit numbers from one and then the other.

The COMPOUND subroutine … //explain

The FUNC subroutine … //explain

### Initialization Routine

The initialization routine provides a one-time initialization of key registers that allow the BumpBot program to execute correctly. First the Stack Pointer is initialized, allowing the proper use of function and subroutine calls.

… //finish explaining the initialization routine here, please ☺

### Main Routine

The Main routine … //explain

### Add16 Routine

The ADD16 routine … //explain

### Sub16 Routine

The SUB16 routine … //explain

### Mul24 Routine

The MUL24 routine … //explain

### Compound Routine

The Compound routine … //explain

## MUL16 Routine

The MUL16 routine … //explain

## FUNC Routine

The FUNC routine … //explain

## Additional Questions

1) The V flag is a status register flag that is set when an arithmetic operation's result exceeds the number of bits, and it can also be used in a similar way to a carry bit when doing signed arithmetic. For example, if we add the numbers 125 and 125 and get 250. In binary, this looks like 01111101 + 01111101 = 11111010. The problem is, if the result is a signed 8 bit number, the 1 in the 8th bit makes the entire number negative, giving us 123 – 128 = -5. If we use the overflow flag to indicate that 250 doesn't fit in as a signed value in an 8-bit register, we can compensate for this error.

2) .BYTE and .EQU are similar, but they have some important differences. .EQU assigns a value to a label. This value can be memory, but doesn't have to be. .BYTE always defines memory, and it is memory that can automatically allocate without overlapping. With .EQU, a programmer needs to calculate in their head the correct addresses to use, and if they want to change one, they may need to change all the ones declared after that one. With .Byte though, a programmer can change the amount of bytes allocated to a certain label, and all the subsequent allocations automatically account for it. One final advantage of .BYTE, more than one byte can be allocated to a single label. With .EQU, only one address can be assigned to a label.

## Difficulties

We faced a couple of difficulties. One of the difficulties that we face was simply just to understand how to use variable within the program, for example, how to move them around different types of memory. Once we got that figured out, another difficulty was fully understanding a workable solution on how to setup the multiplication subroutine, being a little bit more complex than the addition and subtraction subroutines.

## Conclusion

… //write up a simple, detailed conclusion

## Source Code

```
//just paste all the code here, text only, and it will copy the formatting
//if not, highlight this part of the text, and use the paintbrush icon up top (format painter)

;*********************************************************** //remove this when have the code in
```