# **Microprocessors (662-133) NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Lab #7- Subroutines.

Description

This lab will introduce the student to the methods of programs containing subroutines. Call and Ret instructions and will be explained and practiced in the lab.

Learning objectives

1. Create the proper file structure needed by Eclipse.
2. Download an assembly language program into the processor.
3. Use the Nios II debugger to edit and view the program.
4. Use the stepping functions to debug a program.
5. Insert a “break” point to help debug a program.
6. Use pseudo codes to help develop a program.
7. Use the branching instructions to make program decisions.
8. Use branching instructions to create loops.
9. Use the “for” loop to execute a loop a set number of times.
10. Use the Call instruction.
11. Use the Ret instruction.
12. Use the Stack proper to store and restore values in registers for efficiency.
13. Create a lookup table to be used as a seven segment display decoder.

Procedure

For this lab, we will be reviewing a previous lab, and expanding its functionality using subroutines.

To start with, take the instructions from lab 6 that display a number from 0 to 9999 (starts right before the division & remainder calculations, and ends after writing the combined value to the seven segment display), and convert it into a subroutine to use in lab 7

Remember to **save any registers you use** (there are probably a lot of them! Look very closely) on the stack, and restore them before calling 'ret'. The subroutine will have no return value, so it should not modify r2 or r3, but it should follow the convention of taking its one parameter (the number to display) on r4. If your lab 6 did not work reliably, a sample solution will be provided.

Name the subroutine's label whatever you prefer, and be sure to provide a comment or two on operation. You should be able to insert this subroutine into almost any program, simply by adding the subroutine, the .EQU statement for the HEX display address, and the .byte array in the .data section. If you save the registers properly the only modification you should have to make to the program's code is to put the value you want to display in r4 and call the subroutine.

***NOTE: If you want to single step thru the subroutines you must enable “instruction stepping mode”.***

Test your subroutine by adding it to any one of the simpler labs (displaying the switches + 1 in lab1, displaying the answer to the equation in lab 2, the counter in the mini lab 4, or the mini lab 7 to display the factorial.

Once your subroutine is working, you can start on lab 7 proper. You may wish to proceed step by step, and ensure each step is working before proceeding:

Initialize the program, and pick some register (r10-r20) as the current value, initialize it at zero

Construct a main program loop that

a) Waits for any pushbutton to be pressed

b) (You can make use of the chkbtn subroutine in lecture 7 notes to greatly simplify this part!)

If button 0 was pressed, increase stored value by the value on the slider switches.

If button 1 was pressed, decrease the stored value by the value on the slider switches.

If button 2 was pressed, reset the stored value to zero.

If button 3 was pressed, reset the stored value to 3300.

After performing the operations, if the value is below zero, reset it to zero.

After performing the operations, if the value is above 3300, reset it to 3300.

c) Display the current value on the seven-segment display using your subroutine.

Check that this main loop is functioning properly before proceeding. You should be able to add to or subtract from the value on the seven segment display, without being able to go below zero, or above 3300. Review lab 4's structure, as it is similar, but use the chkbtn subroutine instead of masking in the main program loop.

Lastly, review lab 3, and the setup and use of the PWM modules. Modify your program to output the current value to PWM0. Use a clock divider of 3300. If you setup the output filter as before, you have created a programmable voltage source.