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

Lab #4- Looping and Branching

Description

This lab will introduce the student to the methods of programming used with looping and branching instructions 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.

Procedure

As in Lab 3, you will need to setup the project again.

Remember, to get started:

Close any projects you have open.

Create a new project and chose the name 'lab4'.

Create a new source file lab4.s, and copy the contents of the provided lab4.s

Remember to setup the debugger the first time you wish to run the program, create a new debug entry called 'lab4 dbg'.

For this program, we will be creating a simple calculator that can add and multiply

In the main loop, the steps will be as follows:

If KEY 0 is pressed on the push buttons...

1. Read in the slider switches to a register (that is, store the value)
2. After doing this, you should wait until all the keys have been released. (This will involve a small loop that repeatedly reads the buttons)

if KEY1 is pressed on the pushbuttons...

1. Multiply the stored value and the current value on the switches and display the result on the red LEDs

if KEY2 is pressed on the push buttons...

1. Add the stored value and the current value on the switches and display the result on the red LEDs

if no key was pressed, just display the current stored value (r10) on the red LEDs

When the program is done, you should be able to

1. Set a number on the slider switches in binary
2. Tap KEY 0 momentarily to store it
3. Set another number on the slider switches in binary
4. Press one of the other keys to see either the product or the sum result on the LEDs

Also, when pressing KEY0 to store a value, the LEDs should not change to the new value UNTIL you *release* KEY0 so the main loop can continue.

Hint: Start with just a main loop and a single test whether KEY0 is pressed or not, storing the slider switches in a register (say r10) if so. If KEY0 is not pressed, display the current value of r10 on the red LEDS. Then once this is working, chain together the tests for KEY1 and KEY2. Review the if (condition)... else if (condition)... else if (condition)... structure from the lecture.

Challenge:

Choose an operation for KEY3 (subtract? Divide? a bitwise operator?) then implement it.

BITMASKING

In the tests in this lab you will need to check individual bits (buttons) to see if they are set (pressed). Bitmasking is the action of selecting which bits you care about and want to check with a compare or branch instruction. As we've seen on the DE1 board, each input switch is either a 1 or a 0 for the corresponding bit when reading the switch's IO register, and all the sliding switches are on one port (and thus are all read at once), and the same is true for the 4 pushbuttons. So far we've been reading the sliding switches as a single binary number, but if you want to examine the 1/0 on/off state of a single bit you will need to mask it.

By performing a bitwise AND operation on the number of interest (the number containing the bit we care about) along with a binary number containing a single 1 in that same position, we will end up with a number with a value of either 0, or some power of 2, depending on which bit:

AND

01100010

00100000

=

00100000 (The number 32)

AND

01110100

00001000

=

00000000 (Zero, since the same bit wasn't set)

Thus we can see if you want to mask out a bit or bits, AND them with the bit or bits that you care about. Then the result will either be those bits, or zero.

This is why the most common branch statements you see in RISC assembly are beq (branch if equal) or bne (branch if not equal) comparing a register to r0 (the zero register).

Since the result of a mask could be any number that's a power of two, rather than test against a specific number, we just test against zero (r0), and either branch on equal to zero (bits were not present) or branch on not equal to zero (bits were present)

For instance, if we want to test:

if (bit 3 is 1 in r12) then (do something)

Pseudo code:

1) Bitwise AND r12 with 0b0100 and store in r11

2) if r11 is equal to r0 (zero), jump to 4 (button was not pressed)

3) Do whatever we want when button is pressed

4) Continue with rest of program here

Instructions:

andi r11,r12,0b0100 # AND r12 with 0b0100 and store in r11

beq r11,r0,skip # If r11 is equal to r0 (zero), skip past the instructions

nop # Replace nop with instructions to execute

skip: # rest of program continues after skip label