Project 1: Basic Assembly Programming

This project is due on **February 6, 2026** at **6 p.m.** and counts for 5% of your course grade.

The code and other answers you submit must be entirely your own work, and you are bound by the Honor Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with others to develop a solution. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

There is starter code on **Github Classroom** (`https://classroom.github.com/a/zkTsTT9a`), which is also where you will turn in the project, by committing/pushing to the repository created for you when you accept the assignment.
Github Classroom also provides autograding for you. If you click on the `Actions` tab in your repository, you can select a commit and see the autograding report for that version of your repository. Autograding is run each time you commit, and feedback is provided for missed test cases.

Introduction

In this project, you will program a simple assembly program, and run it on the DE10-Lite board or simulator. We will extend the Simple Program from the DE10-Lite setup tutorial. If you have not already done so, we recommend you run this program on your board or simulator, and verify that toggling the switches toggles the corresponding LED above the switch.

Objectives:

- Understand how to create assembly projects and use the DE10-Lite simulator.
- Use Nios II assembly to do basic bit manipulation of registers.
- Understand how a CPU interprets and executes instructions.

Part 0. Installing Software

For this course, you may use either:

- **Online simulator** of the DE10-Lite: `https://cpulator.01xz.net/?sys=nios-de10-lite` (*recommended*) or
- **Physical board** of the DE10-Lite, available online (`http://de10-lite.terasic.com/`) or in the E-store.

If you are using the **online simulator**, you do not need to install the software below. Simply use the website directly.
Note: We recommend saving your progress from the website to your repository frequently. This will help in the event that you close the tab, or your browser (or the site) crashes.

Software installation instructions

If you'd like to use the **physical board**, be warned: the software needed is somewhat buggy, and the setup can be difficult. Install instructions for Linux and Windows can be found here: `https://docs.google.com/document/d/1weECJy2RoGCv4STBf7dwq5mauFFMwJk9HlPW6h1XtqA/edit?usp=sharing`

If you are using a Mac, you will need to install a virtual machine that is capable of running x86_64, such as UTM. Inside this, install either a Windows or Linux (recommended Ubuntu) operating system, then follow the instructions above.

Part 1. Basic Adder

The DE10-Lite has 10 switches along the bottom right of the board (on the simulator, these are checkboxes in the right pane). The provided `sw-led.s` file (shown below, and in your Github Classroom starter repository) is a simple program that reads the state of the switches and writes it to the LEDs. Try running this program on the DE10-Lite (or emulator) and toggling the switches. In this project, you will extend this program to implement an *add* operation.
We will split the switches in half: the leftmost 5 switches will represent a binary number, while the rightmost 5 switches represent a second binary number. The LEDs will display a binary representation of the *sum* of these two numbers.

For example, if we wanted to add the numbers 3 and 5, the board should look as follows:

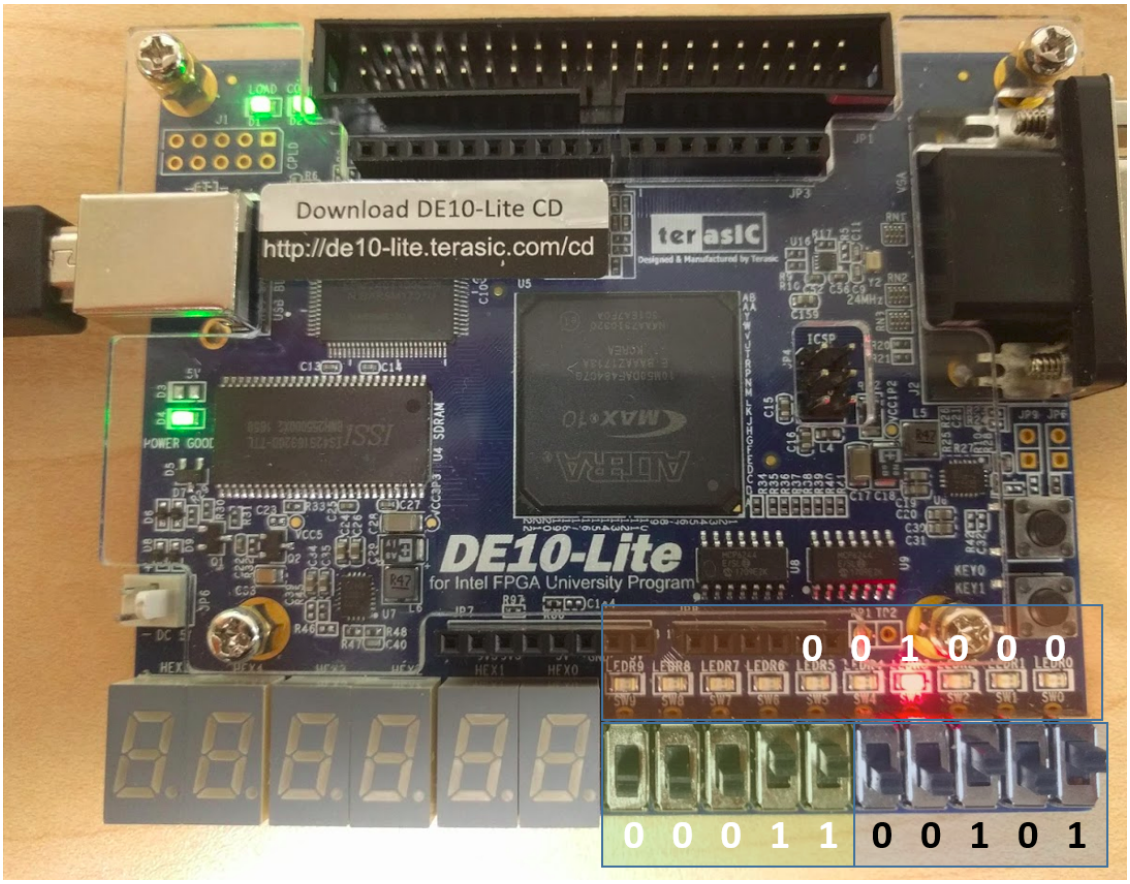

Figure 1: **Adding 3+5** — The leftmost five switches encode the binary number 3 (0b00011) and the rightmost five encode the number 5 (0b00101). The sum of these numbers (8) is displayed in binary on the LEDs: 0b001000.

You should write this program in assembly. You may use the `sw-led.s` file as a starting point - don't worry about how the program is reading the state of the switches for now (or writing the LEDs), that code is provided for you.

You should assume `r4` contains the value of the switches (in a single register), and you must modify it accordingly to treat it as two 5-bit numbers that you then add together. Interpret the switches and LEDs in big endian, with the most significant bit on the left.
Try different switch configurations to verify that your interpretation of the switches and adding is correct.

You can base your code on the following example (from sw-led.s):

```
    .text
    .equ    LEDs,       0xFF200000
    .equ    SWITCHES,   0xFF200040
    .global _start
_start:
    movia   r2, LEDs        # Address of LEDs
    movia   r3, SWITCHES    # Address of switches

LOOP:
    ldwio   r4, (r3)        # Read the state of switches

    # <Your code to modify r4 here>

    stwio   r4, (r2)        # Display the state on LEDs
    br      LOOP
```

Historical fact: The first fully electronic commercially-available calculator was released in 1961. The ANITA (A New Inspiration To Arithmetic) weighed 33 pounds and used vacuum tubes to add, subtract, multiply and divide.

What to submit A `sw-led.s` file that:

1. Takes two 5-bit binary numbers from the switches and adds them
2. Displays the resulting binary number on the LEDs

Part 2. Loops in assembly

We will give you code that provides an integer `N`. Your task is to write assembly that computes the sum of integers from 1–`N` (inclusive). Your result should be stored in `r2` when the `break` instruction is called.
Starter code is provided in `sum-n.s`. We encourage you to write this out in C or a similar high-level language, and then think through how you would translate this into assembly.
Recall that a loop has 3 main parts to it: an **initializer** (what happens before the loop), a **condition** (that breaks the loop), and an **increment** (something that updates each loop).

What to submit A `sum-n.s` file that sums integers 1–`N`.

Part 3. If/else in assembly

For this part, we will provide 3 signed integers in `r4`, `r5`, and `r6`. Write assembly code that finds the maximum of these numbers, and stores it in `r2`.

What to submit A `if-else.s` file that puts the largest of `r4`, `r5`, and `r6` into `r2`.

Part 4. Understanding assembly

4.1 Decoding Instructions

Translate the following hex value to Nios II assembly code.

`993ff915`

4.2 Encoding Instructions

Convert the following assembly into machine code. Your answer should be an 8-digit hex number.

`divu r14,r5,r22`

What to submit A plain text file (e.g. not a word document) named `writeup.txt` containing your answers to part 4. You should format this file using this template (your file should have 3 lines total):

```
1. decoded instruction (e.g. addi r3, r4, 8)
---
2. encoded instruction (e.g. 0xDEADBEEF)
```

Submission Checklist

Commit to your Github Classroom repository the following files:

Part 1

An assembly file named `sw-led.s` that adds the left/right switches and outputs the result (in binary) to the LEDs.

Part 2

An assembly file named `sum-n.s` that sums integers 1–N.

Part 3

An assembly file named `if-else.s` that finds the maximum of 3 numbers.

Part 4

A text file named `writeup.txt` with your answers to the short questions, using the template described above.