Project 2: Basic Input/Output

This project is due on **March 6, 2026** at **6 p.m.** and counts for 6.25% 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/YcSRAgc_`), which is also where you will turn in the project by committing and pushing to the repository you create when you accept the assignment.
If you receive a permission error while accepting the assignment, check your email associated with your Github account for an invitation link to your repository.
Github Classroom also provides autograding for you. Click on the `Actions` tab in your repository, and select a commit to see the autograding report for that version of your repository. Autograding is run each time you commit.

Late days If you wish to use one or more late days on this assignment, fill out this form: `https://forms.gle/ZJdYxYEQgan9zc1P6`

Introduction

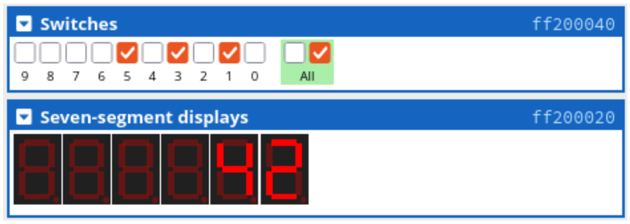
In this project, you will program the DE10-Lite to perform basic input/output (I/O) using the JTAG UART port.

Objectives:

- Understand how memory-mapped I/O (MMIO) is used to interact with peripherals.
- Understand memory alignment and how to access unaligned words.
- Apperciate the simplicity (and downsides!) of busy-wait loops for I/O and timing.

Part 1. Binary to decimal

In this section you will convert the binary value on the DE-10 Lite **switches** to a decimal number displayed on the **Seven-segment display**.
For example, if you set the switches to `0000101010` (42 in binary), then the seven-segment display should show the decimal value `42`:



Your code should run in a loop, so that if the value of switches changes, it will update the decimal value shown on the seven-segment.

Hints

We have given you an array of bytes (labelled `NUMS`) that corresponds to what each seven-segment digit should be set to in order to display a particular number, and the starter code uses that to display a single digit. **How would you display a two or three digit number?** Try to modify the code to display the number `2360`.

Once you can display multiple digits, think about **how to get the individual digits from a number in a register**. For example, if `r5` had the value `42` (`0x2a`), how would you determine that the first digit was a 4, and the second was a 2? Consider how you could compute the remainder of a number after dividing by 10 using a combination of `divu`, `mul`, and `sub` instructions.

What to submit A `sw-to-decimal.s` file that converts the switches binary value to a decimal value displayed on the seven-segment.

Part 2. UART input/output

In this part, you will write a program to read and write to the JTAG UART terminal.
Your program should start by printing a prompt that says `Enter number:`
It should then wait for the user to enter a decimal number, echoing back whatever is entered so it displays in the terminal, until the user presses `enter` (ASCII: `0x0a`)
The program should keep a running total of the numbers entered so far. After each number, the program should print `Total:` and the total of the numbers entered so far.
For example, an example run where a user entered these numbers would look like this:

```
Enter number:12
Total:12
Enter number:3
Total:15
Enter number:10
Total:25
Enter number:
```

Your code should ignore non-numbers entered. For example if a user enters `hello`, then these characters should be echoed back, but ignored by the total.

Hints

Be sure to get the spacing and line returns right, and to not output extra null-terminating characters. The autograder will print out the hex of what has been read.

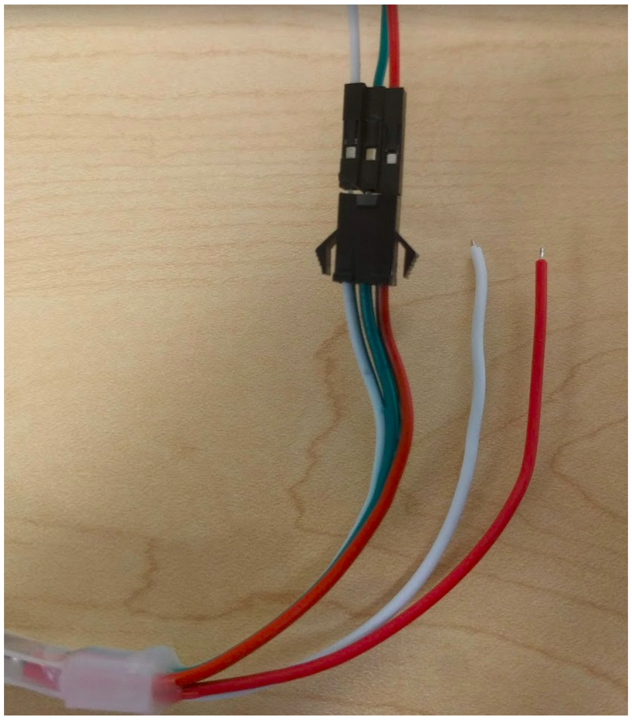
The ASCII codes for the numbers are conveniently in order: `0` is ASCII `0x30`, `1` is `0x31`, and `9` is `0x39`. Therefore, reading an ASCII character for a digit makes it easy to convert that digit to a number by subtracting `0x30` from the character's ASCII value.

What to submit A `jtag.s` file that prints the running total of numbers entered over JTAG UART.

Extra credit: LED strip

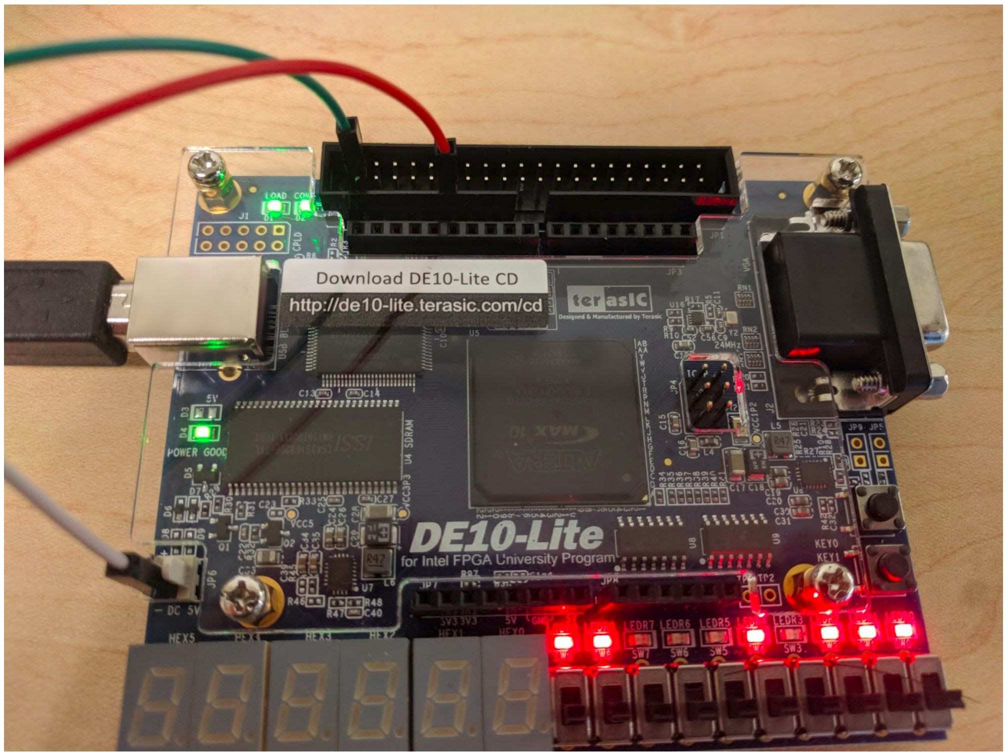
This part requires having a physical DE-10 Lite and LED strip.
Your task is to write a busy-wait assembly program to control a WS2812 LED strip.
You can buy a WS2812-compatible LED strip from Amazon for $10: `https://a.co/d/0gqli8MI`
You are welcome to buy other LED strip variants as well, just ensure it is 5V and WS2812 (or SK6812)-compatible. More options (e.g. 144 LED per meter) can be purchased from Amazon, Adafruit, or Sparkfun.

You'll also need 3 jumper connectors to connect your LED strip to the DE10-Lite. You can either purchase your own, borrow some from an embedded systems lab, or I have a few extras. You'll need three male-to-female connectors (you can also chain two male-male and female-female connectors). Ideally, you can match the colors of the LED strip (red, green, white), but this is not required.



Double check that your strand wires are white for ground (-), red for 5V (+), green for data. The red wire should be connected to +5V on your DE10 (JP1 header pin 11), and the white wire connected to ground. **It is important you do not mix these up**, as getting this backward may short out the LED strip and break it. I recommend connecting the white wire (ground) to the (-) (bottom) side of the DC 5V header to make it easy to double check. Make sure you connected ground to ground!

The green (data) wire should be connected to the JP1 header on pin 2 (upper left-most pin). See the DE10-Lite manual for a GPIO (JP1) pinout: `https://ecen2360.org/static/DE10_Lite_User_Manual.pdf`
Your connections should look like this:


Download DE10-Lite CD
http://de10-lite.terasic.com/cd
DE10-Lite
for Intel FPGA University Program

Ensure that the red (5V) and white (GND) are connected correctly

Take a look at the WS2812 datasheet `https://cdn-shop.adafruit.com/datasheets/WS2812.pdf`, which describes how to drive the LEDs using carefully-timed signals. For example, to send a 0-bit, you must hold the data line high for 0.35 microseconds, and low for 0.8 microseconds. To send a 1-bit, hold the data line high for 0.7 microseconds and low for 0.6 microseconds. Setting the color of an LED takes 24-bits sent in this way, with 8-bits each for green, red, and blue values (0-255 brightness). The entire strand acts as a shift register, where writing a new 24-bit value to the first LED will shift its previous value to the next LED in the chain.

As a note/hint: The DE10-Lite computer has (approximately) a 50 MHz clock, and each instruction takes about 1 cycle to complete. Make sure you enable the *output* direction on whatever GPIO header pin you're using (see the DE10-Lite manual for direction bit MMIO address).

You will write **two** patterns for the LED strip. The first pattern will simply demonstrate that you can control the LED strip, and the second one is up to you!

Part 3.1 Static Pattern

For the first pattern, you will create a static repeating pattern of red, green, and blue across the length of the LED strip:

Blue, red, green repeating pattern

Note: if you have trouble distinguishing between these colors (e.g. red-green color-blind), you may choose any 3 unique colors for your pattern.

Part 3.2 Dynamic Pattern

For the second pattern, you will create any pattern that you like, as long as it changes over time. Here's a basic example for inspiration: `https://youtu.be/hFDNiV1Kmrc`.

Make it pretty! Imagine something you would bring to Burning Man or as a light show at Red Rocks.

What to submit

1. A `led-static.s` file that displays a static pattern with different colors.
2. A `led-dynamic.s` file that displays a dynamic pattern of your choosing.

Submission Checklist

1. Part 1: `sw-to-decimal.s`
2. Part 2: `jtag.s`
3. Part 3: (optional) `led-static.s` and `led-dynamic.s`