Left led ≠ word aligned, hence use store byte to store values at the left location.
led

# CSCI 260 – Project 2

Due: 11:59 PM November 19, 2020

The goal of this homework is to write a useful MIPS program that includes functions, using the MARS simulator. It is an **individual** assignment and no collaboration on coding is allowed (other than general discussions).



## 1 Assignment

For this project, you will draw on two seven-segment LED displays using MARS' "Digital Lab Sim" tool (instead of the bitmap tool from project 1). We will not use the hex keyboard part of the tool.

The output of this assignment is to count $a, a+1, \ldots, b$ in sequence on the LED displays. The numbers $a$ and $b$ are supplied as inputs in the data segment (make sure to use the labels a,b and put these at the beginning of the data segment). You are required to write at least the following functions though you may wish to write more:

- `main()`: This will sequence through $a, a+1, \ldots, b$, calling another function to display the numbers appropriately on the LEDs.
- `display(led,digit)`: This will draw a single digit to a single LED by lighting up the appropriate segments. The arguments are as follows:
  - led: 0 for the left LED, 1 for the right LED
  - digit: 0-9 for the digit being drawn

a0 a1

Left Adress: 11
Right Addr: 10

0 1

(0-9) (0-9)

so each number calls display twice.
eg.) 79 : display(0,7)
display(1,9)

**Main** This first checks that $0 \le a \le b \le 99$ and terminates if not. Then it calls `display` as needed. The number b should be displayed on termination. You will need to figure out how you want to represent the number being drawn in a convenient manner for calling `display`. Since the LEDs will sequence too quickly, you will also need to execute a delay loop between calls to `display`[1]. You can do this by writing a nonsensical loop (perhaps empty) with many iterations. I will leave it to you to experiment with how many iterations, but your delay should be 1-2 seconds. For full credit, you should minimize your use of divisions/mod, as these are particularly expensive (yes, this requirement might seem somewhat silly considering you are slowing the program in the delay loop).

one use of div, not in a loop. Seems like mult not needed

**Display** This one is straightforward after reading Section 1.1 below, though there are several ways of implementing it. To figure out which segments to draw for each digit, you may consult the wikipedia page `https://en.wikipedia.org/wiki/Seven-segment_display` (look under Decimal). Note that MARS (like some real memory systems) requires 32-bit memory accesses to be word-aligned.

[1] While the speed feature provided by the MARS interface may seem to serve the same purpose, please do not use it as that slows everything down.

**A Note on Functions** Since MARS is just a simulator, you will not be able to actually follow our function convention. It may also let you get away with doing terrible things like not popping soemething you pushed onto the stack – this would of course lead to a system crash in reality.

For this project, you should save/restore registers according to our convention, but do **not** use the frame pointer. Although MARS won't necessarily enforce this, part of your grade will be your adherence to the convention.

### 1.1 Using Digital Lab Sim

Both LEDs are implemented using memory mapped I/O – *i.e.*,writes to specific memory locations are interpreted by the hardware to send data to the LEDs instead of actual memory. The particular memory locations are 0xFFFF0011 for the left LED and 0xFFFF0010 for the right LED. The seven segments of the LEDs correspond to bits 0-6 of the data "written" to the appropriate LED. For example, writing the byte 0x24 (0b 0010 0100) to memory location 0xFFFF0010 would light up segments 2 and 5 of the right LED.

You will need to connect to the tool in the same way as project 1.

### 1.2 Debugging Using MARS

As you may have already realized in project 1, MARS includes a symbolic debugger that acts much like the ones you used in 135 for C++. In particular, you can set breakpoints (left column of the execute window text segment. Then, you can single step using the Run menu. Of course, the contents of all registers and memory cells in the data segments are also visible.

## 2 Submission Instructions

Please submit a single text file using blackboard (under the Projects menu). The file should contain your name in header comments, and follow normal MIPS conventions. You may use any of the instructions and pseudo-instructions we covered in class, along with any pseudo-ops supported by MARS.

Your program will be graded on both correctness (including function conventions) and style (meaningful comments on every line, register conventions, etc.).

See syllabus for late policy.