Brown University
School of Engineering
EN164 Computing System Design
Professor Sherief Reda
LAB 02 (100 points).
Due in lab to TAs by Friday, Mar 2nd.

Please download and use the MARS simulator for the first four exercises.

1. [10 points] Write a MIPS program to reverse (not inverse!) the bits in a register (e.g., 10111→11101). Use as few instructions as possible. Assume the register of interest is `$t3`. Use the MIPS simulator to test your code.

2. [15 points] Write a MIPS assembly code to test whether overflow occurs when $t2 and $t3 are added. Use as fewest instructions as possible. Use the MIPS simulator to test your code.

3. [20 points] Write a MIPS procedure that tests whether a given input string is a palindrome. The procedure should return 1 if the string is palindrome; otherwise, 0. (Recall that a palindrome is a word that is the same forward and backward. For example, the words "wow" and "racecar" are palindromes). Write an assembly program that test the procedure by calling it three times using the strings "wow", "racecar", and "sunshine" as inputs. Use the MIPS simulator to test your code.

4. [20 points] Each number in the Fibonacci series is the sum of the previous two numbers:

```
n:    1 2 3 4 5 6  7 ...
f(n): 1 1 2 3 5 8 13 ...
```

By definition the first two numbers in the series are equal to 1.

(1) Write a ***recursive*** procedure called `fib` in a high-level language that you are familiar with (e.g., MATLAB, C, Java or whatever you like) that returns the Fibonacci number for any nonnegative value of `n`. Note: there are a number of ways to implement Fibonacci series; this question asks you to write a recursive method.
(2) Write the same ***recursive*** procedure but in MIPS assembly language. Use the MIPS simulator to test your code.

The team(s) that gets the MIPS procedure working correctly using the minimum number of instructions will get a credit of 1 absolute point that is usable towards the total 100 points that determine the grade of this course.

5. [35 points] Your objective in this problem is to design and implement a “dancing lights” machine, which can execute instructions that let LED light on the DE2 board dance. The machine reads and executes instructions in sequence from ROM. There are three instructions that the machine should handle:

1. ***Light dance move:*** an instruction should specify which of the first six green LEDs is turned on or off.
2. ***Pause time:*** an instruction should specify a pause time in units of 100s of milliseconds.
3. ***Halt***: this instruction will cause the machine to execute no further new instructions

(1) Design an 8-bit instruction set format of this machine specifying the operation mnemonic and operands.
(2) Write an assembly program that consists of at least 6 interesting dance moves that are “well timed” before the machine comes to halt.
(3) Translate the assembly program into machine language using the format you created in part (a)
(4) Write the Verilog code (+ schematic if applicable) for this machine and implement it on the DE2 board. To create the storage for your program instructions, you have two options. You can use either a ROM or a custom array of registers. For the ROM, you will need to instantiate a ROM from the megafunction wizard and initialize its content from a .mif file. The .mif file can be created and edited by choosing File -> New -> Memory Files -> Memory Initialization File. For the register array option, you can declare an array that fits your program and initialize it directly with your instructions inside the `initial` statement.
(5) In addition to reporting the results of the previous 4 items, please report (i) the total logic and routing resources using by your circuit; (4) RTL circuit view; the propagation delay of your circuit critical path.

The team(s) that get their design working correctly while using the minimum number of LEs will get a credit of 2 absolute points that are usable towards the total 100 points that determine the grade of this course.