**CIS 201 – Assembly Programming – Programming Project**

**Bit Flipper**

Overview

In this assignment, the student will write an assembly program that manipulates the individual bits in a BYTE of memory based on user provided inputs. Unique cycles of flipped bits will be shown in the program’s output based on a predefined color cycle.

When completing this project, the student should demonstrate mastery of the following concepts:

* Visual Studio - General Project Creation
* Visual Studio – ASM (Irvine) Project Creation
* Formatting & Transporting Assembly Programs
* Assembly Commands – TEST, CMP, jCOND
* Assembly Commands – ADD, SUB, INC, DEC
* Assembly Commands – AND, OR
* Assembly Labels
* Defining and Using Symbolic Constants
* Direct Array Address Indexing
* Indirect Array Address Indexing
* Defining Custom Procedures
* Call Stack Management
* Program Documentation
* Attractive Aesthetic UI Design

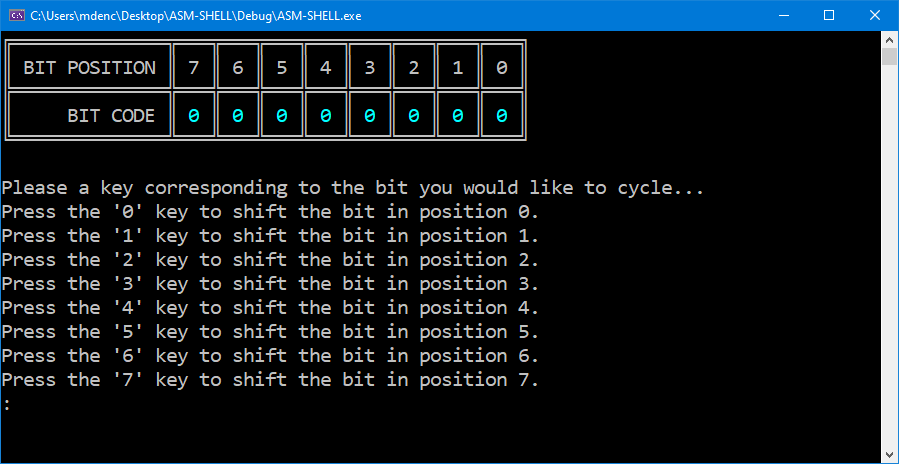
Assignment

In this assignment, the student will create a utility that allows for the casual inputting of a colored bit pattern (1 byte) by pressing numeric keys in sequence that correspond to the positions of the bits that should be moved along a fixed pattern or values and colors.

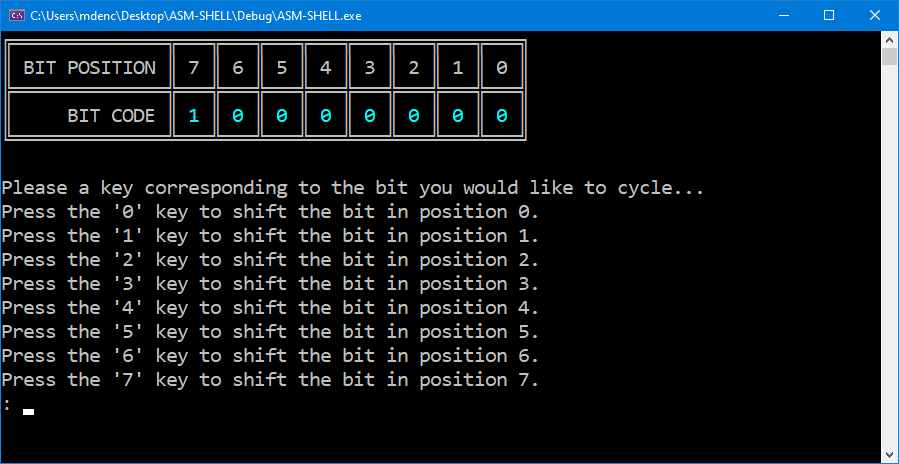
As the user interacts with the program, a common cycle should occur on a continuous basis. First, the screen should be cleared and the table containing the bit positions and colored bit pattern should be nearly drawn. Next, an informative prompt should request the user to enter a numeric digit between 0 and 7 (each input corresponds to a unique bit position). Once a selection has been made with a single keystroke (no enter keypress should be needed for confirmation) the bit in that position should cycle its value and color based on the following pattern:

* Each time a bit position is selected, the value of the bit will change. If the bit previously had a value of 0, the value should flip to 1. If the bit had a value of 1, it should be changed back to 0.
* Each time a bit is flipped from 0 to 1, and then 1 back to 0, the color should shift in the specific order LIGHTCYAN, LIGHTGREEN, LIGHTRED, YELLOW, and WHITE. If another color shift occurs after this point, the color of the bit should start back at the beginning of the cycle with the color LIGHTCYAN.
* FOR EXAMPLE: Let’s consider a single bit in position 7. By default, the starting value should be 0 and the bit should be colored LIGHTCYAN. We will denote this with notation (0:LIGHTCYAN). If continual requests to flip the bit in position 7 occur, the following pattern of bit values and colors should emerge:  
  + (0:LIGHTCYAN)
  + (1:LIGHTCYAN)
  + (0:LIGHTGREEN)
  + (1:LIGHTGREEN)
  + (0:LIGHTRED)
  + (1:LIGHTRED)
  + (0:YELLOW)
  + (1:YELLOW)
  + (0:WHITE)
  + (1:WHITE)
  + (0:LIGHTCYAN) ß At this point, the pattern starts to repeat.
  + (1:LIGHTCYAN)
  + ...

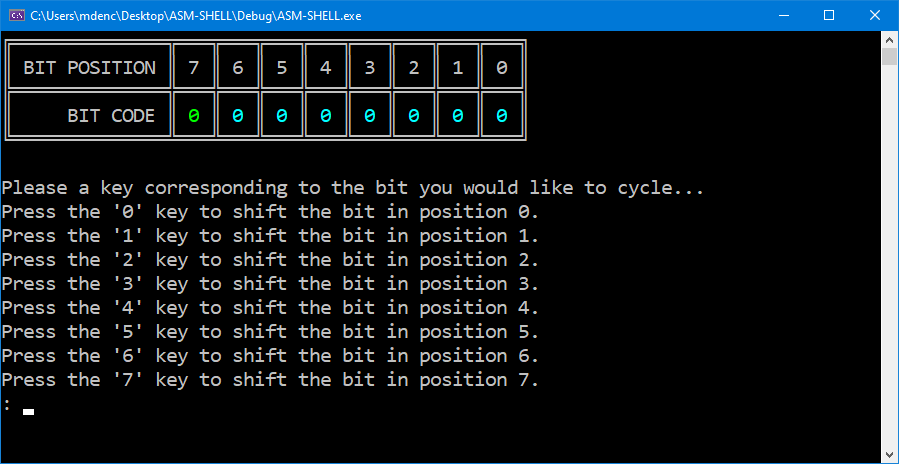
To get an idea about how your program should behave, consider the following sequent of screen shots.



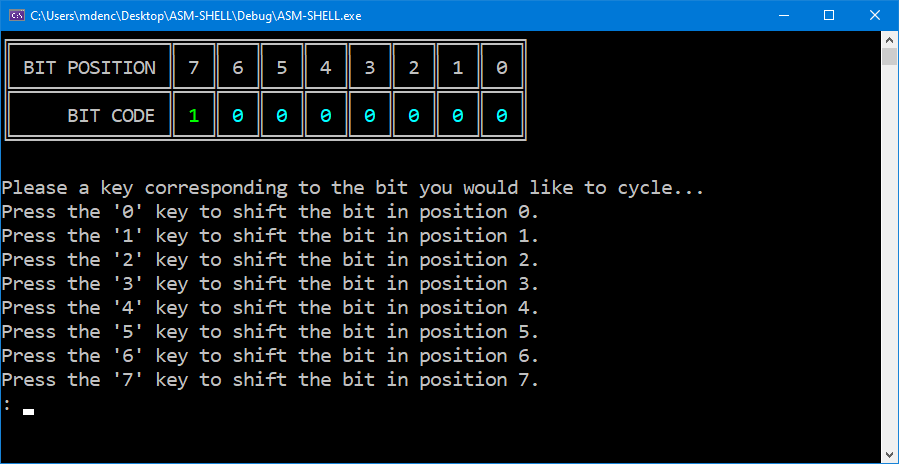
**\* The program just started; the shown bit pattern contains all (0:LIGHTCYAN)**



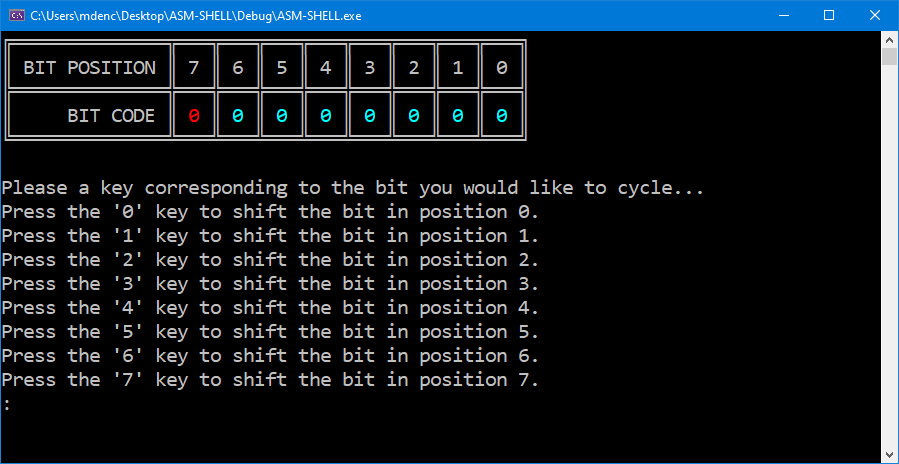
**\* The user pressed ‘7’ to cause the bit in position 7 to change from a 0 to a 1.**



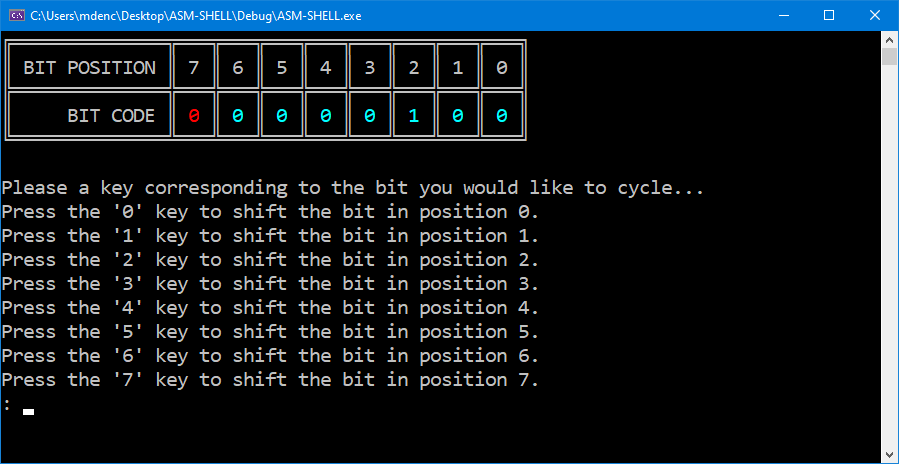
**\* The user pressed ‘7’ again. This reset the value back to 0, but caused the color to cycle to LIGHTGREEN.**



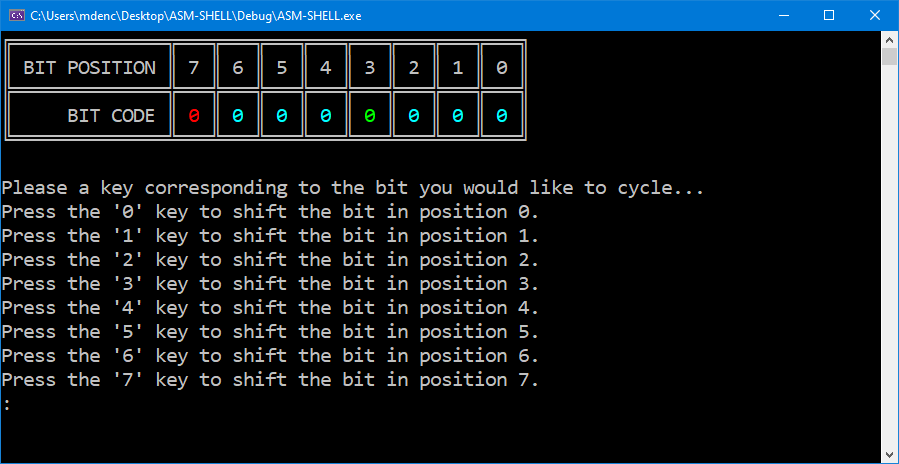
**\* When the user presses ‘7’ again, the bit in position 7 now goes to (1:LIGHTGREEN).**



**\* Another press to ‘7’ causes a change to (0:LIGHTRED).**



**\* This time, the user pressed ‘2’, which began the cycle of transitions on the bit in position 2. It changes from (0:LIGHTCYAN) to (1:LIGHTCYAN).**



**\* A final press of ‘2’ shows this bit operates on the same color pattern. The bit in that position changes from (1:LIGHTCYAN) to (0:LIGHTGREEN).**

**THIS IS DEFINITELY THE WRONG SCREEN SHOT PASTED IN HERE... I WILL FIX THIS WHEN I GET HOME TONIGHT**

When you are writing your solution, please keep the following requirements/guidelines in-mind:

* You should break down your core strategy into a series of smaller problems that are properly expressed with well defined custom procedures. You have creative freedom in terms of how you manage the call stack within your procedures, but a method must be chosen that preserves the contents of relevant register values on behalf of the calling procedure.
* You do not have to provide a keyboard input entry to exit the program, but one can be introduced if you would like. Minimally, your program must operate on an infinite loop where the screen is cleared and re-drawn each time the user chooses to cycle the value of a specific bit position. If you wish to cause a premature termination, this can be accomplished by sending the Windows console the “kill” signal with the “Control + C” keyboard combination.
* Your solution must be fully documented to receive full credit. In particular, regions of your program that contain clever/creative/sophisticated calculations should have the strategy clearly explained in the program documentation.
* You are not required to use all of the commands and skills mentioned in the skill mastery list at the beginning of these instructions. Solutions to this problem can be developed with a subset of these options. However, if you decide to use techniques that were not explicitly reviewed in class at some point, clearly document the commands and explain how it is been incorporated into your strategy.

Assessment

This assignment will be assessed based on the provided grading rubric.