Assembler #5

Re-submit Assignment

Due Aug 9 by 11:59pm **Points** 50 **Submitting** a file upload **File Types** asm and exe

At the end of this assignment, upload both your assembler source: **CS2810-Assembler-Template.asm** and the executable file **CS2810-Assembler-Template.exe.**
(note that the .asm file is in your project folder and the .exe in the debug folder within the project)

a. Download the assignment project template zip file from 2015 project template: **https://weberstate.box.com/v/2810-asm-vs2015 (https://weberstate.box.com/v/2810-asm-vs2015)**
   2017 project template: **https://weberstate.box.com/v/2810-VS2017-template (https://weberstate.box.com/v/2810-VS2017-template)**
b. Unzip the project folder to a location on your hard drive.
c. Start Visual Studio and open the VS solution contained in the project folder.
d. When the project has started, open the assembler source code file, CS2810-Assembler-Template.asm
e. Modify the program source code to do the following:
   (Basically, you are writing a program to play the higher/lower guessing game)

1. Clear the display.
2. Display the string "CS2810 Fall Semester 2017" on row 4 column 0 of the display.
3. Display the string "Assembler #5" on row 5, column 0 .
4. Display your name on row 6 column 0 of the display.
5. Begin displaying the guessing game on row 8, column 0. Display steps on individual rows as shown in the sample run section.
6. Generate a random number between 0 and 100. (see Irvine notes below)
7. Prompt the user to guess a number between 0 and 100.
8. Based on your randomly generated number, and tell the user whether the real number is higher, lower, or correct. (“100 is too high” or “50 is too low”)
9. Continue to allow the user to guess until they get the correct number. Once they have the correct value, ask them if they would like to play again. If yes, clear the display and start over. If no, exit the program.

Sample Program Run

Sample Run

```
CS 2810 Fall Semester 2017
 Assembler #5
 Josh Jensen

Guess a number between 0 and 100: 57
 57 is too high
 Guess again: 50
 50 is too low
 Guess again: 55
 55 is correct!
 Would you like to play again? (1 for yes, 0 for no)
 0
```

New to this assignment, you will need to write a carriage return to the console (this is what happens when you press “enter”.)

Store the following as a string and write it out to the console as normal anywhere you want to move the cursor down a single row.

```
vCarriageReturn byte 13,10
```

The Irvine library contains a number of Procs and Macros that simplify I/O in MASM.

Irvine Procedures:

- **Clrscr** - clears the command window.
- **Gotoxy** - positions the cursor at the row and column specified in register sections DH and DL
- **WriteString** - displays the string at the address specified in the EDX register. The string must be 0 terminated.
- **ReadDec** - allows the user to enter a decimal value which is converted to binary and placed in EAX
- **Randomize** - Seeds the pesudo-random number generator with the current system time accurate to 1/100th of a second. This only needs to be called once for your entire program.
- **RandomRange** – generates an unsigned pseudo-random 32-bit number, between o and (n-1) and stores it in eax. Specify the maximum

number (n) by placing the max into eax. i.e. (eax=100, will generate and store a number between 0 and 99.) This should be used after a call to Randomize has been issued.

- **WriteDec** – displays the unsigned integer equivalent of the number stored in the eax register to the console

The sample program shows how to call the Procs.
Note that with any Procs that require values to be in registers, the registers **MUST** be set **PRIOR** to calling the Proc.

The following are some helpfup reminders to various MASM instructions we have used this semester.

CALL - the CALL instruction pushes a return address on the stack and then jumps to the specified label.
Example: CALL DisplayName

RET - The RET instruction jumps to the address on the top of the stack and pops the stack. It is used to provide a return from a CALL.
Example: RET

MOV - the MOV instruction copies from register to register, register to memory and memory to register.
Sample formats are:
MOV ECX, EAX
MOV word ptr [displayTime+0], ax

SHL - The Shift Left command shifts the bits of a register to the left. As bits are shifted out to the left, zeros are brought in on the right. The effect of the command SHL AX,1 is that of multiplying the AX register by 2.

AND - This instruction performs a bit-by-bit logical AND between a register and a literal with the result of the AND placed in the specified register. It is used to mask (ie. strip out) bits from a register.
Examples:
AND AX,1111100000000000b
AND AX,F800h

CMP - performs an implied subtraction of the source operand from the destination operand.
Example:
CMP EAX,3

JZ, JNZ, JG, JL, JGE, JLE - are conditional jumps where program execution is transferred to the specified label based on the state of both the zero flag and the sign flag.
Example:
JZ newLocation

Explanations of the letter codes are as follows:

JZ = Jump Zero.
JNZ = Jump Not Zero
JG = Jump Greater Than
JL = Jump Less Than
JGE = Jump Greater Than or Equal To
JLE = Jump Less Than or Equal To

JMP - is an unconditional jump. Program execution is transferred to the specified label.
Example:
JMP newLocation

MUL - multiplies the contents of the specified register by the contents of the EAX register and places the result in the EAX register.
Example: MUL BL

DIV - The Divide instruction allows for 8, 16 and 32 bit division. In this assignment you will use AX register as the dividend and use an 8-bit register half as the divisor. After the division, the quotient will be in AL and the remainder in AH for 8-bit division. For 32-bit division, the quotient is in AX and the remainder in DX.
Examples:
DIV BL
DIV BX
Note that the dividend is not specified. It is always AX for 8-bit divisors and DX:AX for 16-bit divisors

The command window is typically 80 columns (characters) wide and 25 rows (lines) deep.