***Assembly Language 2014***

***Assignment #1***

***General notes:***

1. Assignment deadline is Thursday 20th November 2014 11:59 PM.
2. Make a separate .asm file for each problem.
3. Make sure your code is running, we **won’t fix** any syntax errors.

Problems (Chapter 4: Programming Exercises):

1. **Carry Flag**

Write a program that uses addition and subtraction to set and clear the Carry flag. After each instruction, insert the **call DumpRegs** statement to display the registers and flags. Using comments, explain how (and why) the Carry flag was affected by each instruction.

You are required to write 4 statements as follows:

1 Addition that sets the carry flag

1 Subtraction that sets the carry flag

1 Addition that clears the carry flag

1 Subtraction that clears the carry flag

1. **Zero and Sign Flags**

Write a program that uses addition and subtraction to set and clear the Zero and Sign flags. After each addition or subtraction instruction, insert the **call DumpRegs** statement (see Section 3.2) to display the registers and flags. Using comments, explain how (and why) the Zero and Sign flags were affected by each instruction.

You are required to write 4 statements as follows:

1 Addition that sets the zero flag & clears the sign flag

1 Addition that clears the zero flag & sets the sign flag

1 Subtraction that sets the zero flag & clears the sign flag​

1 Subtraction that clears the zero flag & sets the sign flag

1. **Overflow Flag**

Write a program that uses addition and subtraction to set and clear the Overflow flag. After each addition or subtraction instruction, insert the **call DumpRegs** statement (see Section 3.2) to display the registers and flags. Using comments, explain how (and why) the Overflow flag was affected by each instruction. Include an ADD instruction that sets both the Carry and Overflow flags.

You are required to write 4 statements as follows:

1 Addition that sets the overflow flag

1 Subtraction that sets the overflow flag

1 Addition that clears the overflow flag

1 Subtraction that clears the overflow flag

1. **Direct-Offset Addressing**

Insert the following variables in your program:  
.data  
Uarray WORD 1000h,2000h,3000h,4000h  
Sarray SWORD -1,-2,-3,-4

Write instructions that use direct-offset addressing to move the four values in **Uarray** to the EAX, EBX, ECX, and EDX registers. When you follow this with a **call DumpRegs** statement (see Section 3.2), the following register values should display:

**EAX=00001000 EBX=00002000 ECX=00003000 EDX=00004000**

Next, write instructions that use direct-offset addressing to move the four values in **Sarray** to the EAX, EBX, ECX, and EDX registers. When you follow this with a **call DumpRegs** statement, the following register values should display:

**EAX=FFFFFFFF EBX=FFFFFFFE ECX=FFFFFFFD EDX=FFFFFFFC**

1. **Reverse an Array**

Use a loop with indirect or indexed addressing to reverse the elements of an integer array **in place**. Do not copy the elements to any other array. Use the SIZEOF, TYPE, and LENGTHOF operators to make the program as flexible as possible if the array size and type should be changed in the future. Optionally, you may display the modified array by calling the DumpMem method from the Irvine32 library. See Chapter 5 for details. *(A VideoNote for this exercise is posted on the Web site.)*

Hint: If you need to divide by 2 use SHR (shift right mnemonic)

SHR EAX, 1 ; this shifts the EAX 1 step right (i.e. divides by 2)

1. **Fibonacci Numbers**

Write a program that uses a loop to calculate the first seven values of the *Fibonacci* number sequence, described by the following formula: Fib(1) = 1, Fib(2) = 1, Fib(n) = Fib(n-1)+Fib(n-2). Place each value in the EAX register and display it with a **call DumpRegs** statement (see Section 3.2) inside the loop.

1. **Arithmetic Expression**

Write a program that implements the following arithmetic expression:

EAX = −val2 + 7 −val3 + val1

Use the following data definitions:  
val1 SDWORD 8  
val2 SDWORD 15  
val3 SDWORD 20

In comments next to each instruction, write the hexadecimal value of EAX. Insert a **call** **DumpRegs** statement at the end of the program.

Note that you are not allowed to change the values of val1, val2 and val3

1. **Copy a String Backwards**

Write a program using the LOOP instruction with indirect addressing that copies a string from **source** to **target**, reversing the character order in the process. Use the following variables:

source BYTE "This is the source string",0  
target BYTE SIZEOF source DUP('*#*')

Insert the following statements immediately after the loop to display the hexadecimal contents  
of the target string:  
mov esi,OFFSET target ; offset of variable  
mov ebx,1 ; byte format  
mov ecx,SIZEOF target ; counter  
call DumpMem

If your program works correctly, it will display the following sequence of hexadecimal bytes:  
67 6E 69 72 74 73 20 65 63 72 75 6F 73 20 65 68  
74 20 73 69 20 73 69 68 54 00  
(The DumpMem procedure is explained in Section 5.3.2.) *(A VideoNote for this exercise is posted*  
*on the Web site.)*