^ = AND

| Overflow: OV | | Direction: UP | | Interrupt: EI | | Sign: PL | | Zero: ZR | | Auxiliary: AC | | Parity: PE | | Carry: CY |

(*MSDN reference: https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/kwydd1t7(v=vs.85))*

The status flags (bits 0, 2, 4, 6, 7, and 11) of the EFLAGS register indicate the results of arithmetic instructions (NOT for MOV), such as the ADD, SUB, MUL, and DIV instructions. The functions of the status flags are as follows:

CF (bit 0) Carry flag. Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.

ZF (bit 6) Zero flag. Set if the result is zero; cleared otherwise.

SF (bit 7) Sign flag. Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)

OF (bit 11) Overflow flag. Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.

(source: http://www.cs.princeton.edu/courses/archive/fall13/cos375/IA32StatusFlags.pdf)

Overflow Flag: ADD FFFFFFFF, 1

0xFFFFFFFF +

0x00000001 =

0x00000000 (Carry Flag = 1, Auxiliary flag = 1 although you don't need to worry about the Auxiliary flag).

When the carry flag is set, you have the answer in 32-bit DWORD which is 1. This is just in case you interpreted these values as signed values so that you say 0xFFFFFFFF is -1 and 0x00000001 is 1 but for the CPU 0xFFFFFFFF is equal to (2^32)-1 = 4294967295 = 0xFFFFFFFF. So when you ask the CPU to add 4294967295 (0xFFFFFFFF) and 1 (0x00000001), it will definitely add them as unsigned numbers and it will set the flags accordingly. Now if you interpret the number as a signed value, the carry should mean nothing to you because you were not looking for carry but otherwise it (the carry) tells you that in the process of adding two (unsigned) numbers, the carry flag is set and thus this means that the addition could not be completed in only 32-bits.

Source

http://www.asmcommunity.net/forums/topic/?id=25921

https://stackoverflow.com/questions/23990071/sign-carry-and-overflow-flag-assembly/24002847

1. After the program completes the array "newValue" should be the reverse of "oldValue". E.g. if the values in "oldValue" are 1,2,3,4,5 then after the program completes the values in "newValue" should be 5,4,3,2,1. This program should be flexible enough handle arrays with 2 to 100 elements.

```
.DATA
        oldValue WORD 1,2,3,4,5
        newValue WORD LENGTHOF oldValue DUP(?)
.CODE
        main PROC
                _____________________
                M:
                _____________________
                LOOP M
                _____________________
                N:
                _____________________
                LOOP N
        INVOKE ExitProcess, 0
    main ENDP
END main
```

2. What is the value of EAX after each of the lines of code executes?

```
.DATA
        val SWORD 0FFAAh
.CODE
        main PROC
                MOVSX EAX, val          ;EAX =
                MOVZX EAX, val          ;EAX =
                MOV AL, BYTE PTR [val]     ;EAX =
                MOV AL, BYTE PTR [val + 1]  ;EAX =
        INVOKE ExitProcess, 0
    main ENDP
END main
```

3. Use the following code to list values of flags. Assume all the flags start with a value of 0.

```
.DATA
        var1 WORD 7Fh
.CODE
        main PROC
                MOV EAX, 0   ;  SF=          ZF=          CF=
                MOV AX, var1 ; SF=           ZF=          CF=
                ADD EAX, 1   ;  SF=          ZF=          CF=
                SUB EAX, 1   ;   SF=         ZF=          CF=
                INVOKE ExitProcess, 0
        main ENDP
END main
```