Each of the programming snippets is assumed to have the following snippet preceding the code.

```
.386

.model flat,stdcall

.stack 4096

ExitProcess proto,dwExitCode:dword
```

1. Fill in the blanks with the proper instructions to solve the equation (A - B) + (C - D) and store the result in the EAX register. The code will always have the proper value in the EAX register.

```
.DATA

        varA BYTE 5

        varB BYTE 2

        varC BYTE 10

        varD BYTE 5

.CODE

        main PROC

                mov EAX, 0

                mov AL, varA

                sub AL, varB

                mov BL, varC

                sub BL, varD

                add AL, BL

                INVOKE ExitProcess, 0

        main ENDP

END main
```

2. After the program completes the sum of the numbers from 1 to n should be in the EAX register. E.g. if n is 5 then 15 (1+2+3+4+5) should be in EAX.

```
.DATA

        n DWORD 5

.CODE

        main PROC

                mov EAX, 0

                mov ecx, n
                S:

                ADD EAX, ecX

                Loop S
                INVOKE ExitProcess, 0

        main ENDP

END main
```

3. What is the value of AL after each of the lines of code executes?

```
.DATA

        val WORD 1,2

.CODE

        main PROC

                MOV AL, TYPE val    ;AL = 02 H

                MOV AL, SIZEOF val  ;AL = 04 H

                NEG AL              ;AL = FC

                INC AL              ;AL = F0

                DEC AL              ;AL = FC

        main ENDP

END main
```

FF - 03 = FC

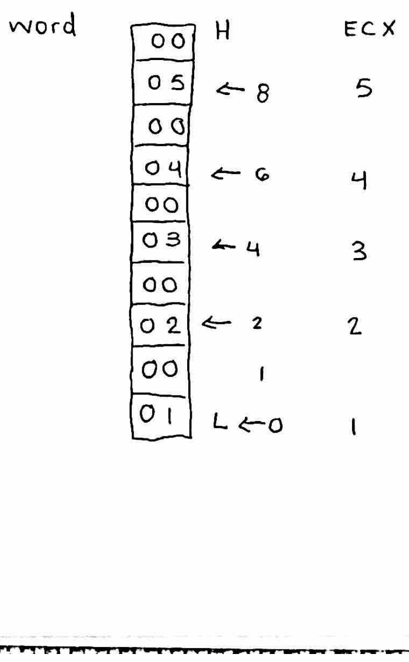
4. What is the value of EAX after each of the lines of code executes?

```
.DATA
        val SWORD 0FFAAh
.CODE
        main PROC
                MOVSX EAX, val          ;EAX = FFFF FFAA h
                MOVZX EAX, val          ;EAX = 0000 FFAA h
                MOV AL, BYTE PTR [val]  ;EAX = AAh
                MOV AL, BYTE PTR [val + 1] ;EAX = FFh
        INVOKE ExitProcess, 0
    main ENDP
END main
```

(must be flexible)

After the program completes "new Value" should be the reverse of "oldValue"

```
. DATA
                                          —> Type : 2 , SIZEOF : 10
    oldValue   WORD   1 , 2 , 3 , 4 , 5
    newValue   WORD   LENGTHOF oldValue DUP(?)
. CODE
    main PROC
        mov  ECX , LENGTHOF oldValue
        M :
        push  oldValue [(ECX - 1) * 2]
        Loop  M
        mov  ECX, LENGTHOF old Value
        N :
        pop  newValue [(ECX - 1) * 2]
        Loop  N
    INVOKE  ExitProcess, 0
    main  ENDP
END  main
```


word
00
05
00
04
00
03
00
02
00
01
H
← 8
← 6
← 4
← 2
1
L ← 0
ECX
5
4
3
2
1