**EL-213: Comp. Organization & Assembly Language Lab**

**Lab#04:** *Data Related Operators & Directives*

# OBJECTIVES:

* Direct-offset operands
* OFFSET operator
* ALIGN directive
* PTR operator
* TYPE operator
* LENGTHOF operator
* SIZEOF operator
* PTR operator

**Direct-offset Operands:**You can add a displacement to the name of a variable, creating a direct-offset operand.

**Example:**

.data

arrayB BYTE 10h,20h,30h,40h,50h

.code

mov al,arrayB ; AL = 10h

mov al,[arrayB+1] ; AL = 20h

mov al,[arrayB+2] ; AL = 30h

**OFFSET Operator:**The OFFSET operator returns the offset of a data label.

**Syntax:**MOV reg32, OFFSET mem ; reg32 points to count

**Example:**

.data

bVal BYTE ?

wVal WORD ?

dVal DWORD ?

dVal2 DWORD ?

• If bVal is located at offset 00404000h, we would get:

mov esi, OFFSET bval ; ESI = 00404000

mov esi, OFFSET wVal ; ESI = 00404001

mov esi, OFFSET dVal ; ESI = 00404003

mov esi, OFFSET dVal2 ; ESI = 00404007

**ALIGN Directive:**The ALIGN directive aligns a variable on a byte, word, doubleword, or paragraph boundary.

**Syntax:**ALIGN bound *(where bound is either 1, 2 or 4)*

**Example:**

bVal BYTE ? ; 00404000

ALIGN 2

wVal WORD ? ; 00404002

bVal BYTE ? ; 00404004

ALIGN 4

dVal DWORD ? ; 00404008

dVal2 DWORD ? ; 0040400C

**PTR Operator:**You can use the PTR operator to override the declared size of an operand.

**Example:**

.data

val32 DWORD 12345678h

.code

mov ax, word PTR val32 ;AX=5678h

mov dx, word PTR val32+2 ;DX=1234h

**TYPE Operator:**The TYPE operator returns the size, in bytes, of a single element of a variable.

**Syntax:**MOV reg16, TYPE mem

**Example 1:**

.data

var1 BYTE ? ; TYPE var1 = 1

var2 WORD ? ; TYPE var2 = 2

var3 DWORD ? ;TYPE var3 = 4

var4 QWORD ? ;TYPE var4 = 8

**Example 2:**

.data

var1 BYTE 20h

var2 WORD 1000h

var3 DWORD ?

var4 BYTE 10, 20, 30, 40, 50

msg BYTE ‘File not found’, 0

.code

mov ax, TYPE var1 ; AX = 0001

mov ax, TYPE var2 ; AX = 0002

mov ax, TYPE var3 ; AX = 0004

mov ax, TYPE var4 ; AX = 0001

mov ax, TYPE msg ; AX = 0001

**LENGTHOF Operator:**

The LENGTHOF operator counts the number of individual elements in a variable that has been defined using DUP.

**Syntax:**MOV reg16 , LENGTHOF mem

**Example:**

.data

val1 WORD 1000h

val2 SWORD 10, 20, 30

array WORD 32 DUP(0)

array2 WORD 5 DUP(3 DUP(0))

message BYTE ‘File not found’, 0

.code

mov ax, LENGTHOF val1 ; AX = 1

mov ax, LENGTHOF val2 ; AX = 1

mov ax, LENGTHOF array ; AX = 32

mov ax, LENGTHOF array2 ; AX = 5

mov ax, LENGTHOF message ; AX = 1

**SIZEOF Operator:**The SIZEOF operator returns the number of bytes an array takes up. It is similar in effect to multiplying LENGTHOF with TYPE.

**Syntax:**MOV reg16/32 , SIZEOF mem

**Example:**

intArray WORD 32 DUP(0) ; SIZEOF = 64

**EXERCISES:**

**1.** Initialize a double word array consisting of elements 7,4,1,5, 2. Sort the given array in ascending order directly with the help of registers. Use direct-offset addressing to access the elements.

**2.** Use following array declarations:  
arrayB BYTE 10, 20, 30  
arrayW WORD 150, 250, 350  
arrayD DWORD 600, 1200, 1800AS

Now initialize three double word variables SUM1, SUM2, SUM3 and perform following operations (expressed in pseudo-code here):  
  
SUM1 = arrayB[0] + arrayW[0] + arrayD[0]  
SUM2 = arrayB[1] + arrayW[1] + arrayD[1]  
SUM3 = arrayB[2] + arrayW[2] + arrayD[2]

*Note: Use PTR or any other required directives/operator*.

**3.** Write instructions to evaluate the expressions. Variables x , y , w and v must store consecutive uninitialized storage in memory between range -127 to +128.

z = x + y + w – v +u

**4.** Initialize two arrays:  
  
array1 BYTE 10, 20, 30, 40  
array2 BYTE DUP(?)  
  
Copy elements of array1 into array2 in reverse order.