

LAB 05

Working with Data related Operator and Directives, Addressing



STUDENT NAME

ROLL NO

SEC

LAB ENGINEER'S SIGNATURE & DATE

MARKS AWARDED: /

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
(NUCES), KARACHI**

Prepared by: Muhammad Nadeem

Lab Session 05: Working with Data Related Operators and Directives, Addressing

OBJECTIVES:

- Observing effect of Arithmetic Instructions on Flag
- Register Direct-offset operands
- OFFSET operator
- PTR operator
- TYPE operator
- LENGTHOF operator
- SIZEOF operator
- Indirect operands
- Indexed operands

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
arrayW WORD 100h,200h,300h

.code
mov al,arrayB           ; AL = 10h
mov al,[arrayB+1]       ; AL = 20h
mov ax,arrayW           ; AX = 100h
mov ax,[arrayW+2]       ; AX = 200h
```

Similarly, the second element in a doubleword array is 4 bytes beyond the first one.

DATA-RELATED OPERATORS AND DIRECTIVES**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
```

PTR Operator:

We can use the PTR operator to override the declared size of an operand. Note PTR must be used in combination with one of the standard assembler data types.

For example, that we would like to move the lower 16 bits of a doubleword variable named myDouble into AX. The assembler will not permit the following move because the operand sizes do not match:

```
.data
myDouble DWORD 12345678h
.code
mov ax, myDouble ; error
```

But the WORD PTR operator makes it possible to move the low-order word (5678h) to AX:

```
mov ax, word ptr myDouble      ; AX = 5678H
```

and higher word (1234h) to AX:

```
mov dx, word ptr myDouble+2    ; DX = 1234H
```

Moving Smaller Values into Larger Destinations

We might want to move two smaller values from memory to a larger destination operand. In the next example, the first word is copied to the lower half of EAX and the second word is copied to the upper half.

The DWORD PTR operator makes this possible:



```
.data
wordList WORD 5678h, 1234h
.code
mov eax, DWORD PTR wordList           ; EAX = 12345678h
```

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 var =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 10 DUP(?),0
array2 WORD 5 DUP(3 DUP(0))
msg BYTE 'File not found', 0

.code
mov ax, LENGTHOF val1 ; AX = 1
mov ax, LENGTHOF val2 ; AX = 3
mov ax, LENGTHOF array ; AX = 11
mov ax, LENGTHOF array2 ; AX = 15
mov ax, LENGTHOF msg; AX=15
```

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:

```
.data
intArray WORD 32 DUP(0)
.code
mov eax,SIZEOF intArray ; EAX =64
```

Indirect Operands

In protected mode, an indirect operand can be any 32-bit general-purpose register (EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP) surrounded by brackets. The register is assumed to contain the address of some data.

Example:

```
.data
byteVal BYTE 10h
.code
mov esi,OFFSET byteVal
mov al,[esi] ; AL = 10h
```



If the destination operand uses indirect addressing, a new value is placed in memory at the location pointed to by the register.

```
mov [esi],bl
```

Using PTR with Indirect Operands

```
inc [esi] ; error: operand must have size
```

The assembler does not know whether ESI points to a byte, word, doubleword, or some other size. The PTR operator confirms the operand size:

```
inc BYTE PTR [esi]
```

Arrays

Indirect operands are ideal tools for stepping through arrays.

Example:

```
.data
```

```
arrayB BYTE 10h,20h,30h
```

```
.code
```

```
mov esi,OFFSET arrayB
```

```
mov al,[esi] ; AL = 10h
```

```
inc esi
```

```
mov al,[esi] ; AL = 20h
```

If we use an array of 16-bit integers, we add 2 to ESI to address each subsequent array element.

```
.data
```

```
arrayW WORD 1000h,2000h,3000h
```

```
.code
```

```
mov esi,OFFSET arrayW
```

```
mov ax,[esi] ; AX = 1000h
```

```
add esi,2
```

```
mov ax,[esi] ; AX = 2000h
```



If we use an array of 32-bit integers, we add 4 to ESI to address each subsequent array element.

Indexed Operands

An indexed operand adds a constant to a register to generate an effective address. Any of the 32-bit general-purpose registers may be used as index registers.

SYNTAX:

constant [reg32] ; reg32 can be any of the 32-bit general registers

[constant + reg32]

EXAMPLE:

.data

arrayB BYTE 20, 40, 60, 80

.code

mov esi, 1

mov al, arrayB[esi]

inc esi

mov al, arrayB[esi]

mov esi, 3

mov al, [arrayB + esi]

Adding Displacements: The second type of indexed addressing combines a register with a constant offset. The index register holds the base address of an array.

INCLUDE Irvine32.inc

.data


arrayW WORD 1000h,2000h,3000h

.code

main PROC

mov eax,0

mov ebx,0

 mov ecx,0

```
mov esi,OFFSET arrayW  
mov ax,[esi] ; AX = 1000h  
mov bx,[esi+2] ; AX = 2000h  
mov cx,[esi+4] ; AX = 3000h
```

Scale Factors in Indexed Operands

Indexed operands must take into account the size of each array element when calculating offsets.

SYNTAX:

constant [reg32 * TYPE constant]

EXAMPLE:

```
INCLUDE Irvine32.inc
```

```
.data
```

```
arrayW WORD 1000h, 2000h, 3000h, 4000h
```

```
.code
```

```
main PROC
```

```
mov eax,0
```

```
mov ebx,0
```

```
mov ecx,0
```

```
mov esi, 1
```

```
mov ax, arrayW[esi * TYPE arrayW]
```

```
mov esi, 2
```

```
mov bx, arrayW[esi * TYPE arrayW]
```

```
mov esi, 3
```

```
mov cx, arrayW[esi * TYPE arrayW]
```

```
call DumpRegs
```



Exercises:

1. Declare a 32-bit signed integer `val1` and initialize it with the eight thousand. If `val1` is incremented by 1 using the `ADD` instruction, what will be the values of the Carry and Sign flags?
2. Write down the values of the Carry, Sign, Zero, and Overflow flags after each instruction has executed:

```
mov ax,7FF0h
add al,10h      ; a. CF = SF = ZF = OF =
add ah,1        ; b. CF = SF = ZF = OF =
add ax,2        ; c. CF = SF = ZF = OF =
```

3. Initialize a Byte array consisting of elements 61,43,11,52, 25. Sort the given array in ascending order directly with the help of registers (you do not need to use a loop here). Use direct-offset addressing to access the array elements. (Hint: Use new array for sorted elements). What would be changes in code if declared array was of `WORD` and `DWORD` size?
4. Use following array declarations:
arrayB BYTE 10, 20, 30 arrayW
WORD 150, 250, 350 arrayD
DWORD 600, 1200, 1800

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]`

5. Initialize two arrays: `array1`
BYTE 10, 20, 30, 40
`array2` BYTE 4 DUP (?)

Copy elements of `array1` into `array2` in reverse order using either indirect addressing or direct-offset addressing. Use `ESI` and `EDI` Registers. (Hint: `INC` and `DEC` of `OFFSET`).

6. Subtract an array of 5 doublewords using indirect operands. Save the final result in a variable.
7. Use following array declarations:



arrayB BYTE 60, 70, 80

arrayW WORD 150, 250, 350

arrayD DWORD 600, 1200, 1800

For each array, add its 1st and last element using scale factors and display the result in a separate register. (Hint: Use ESI and TYPE Operator).

