Microprocessors (0630371)
Fall 2010/2011 – Lecture Notes # 12

Data-Related Operators and Directives

Outline of the Lecture

- **Operators.**
- **Directives.**
- **Programming Example.**
- **Programming Exercises.**

Operators

Use these operators to get information about address and size characteristics of data.

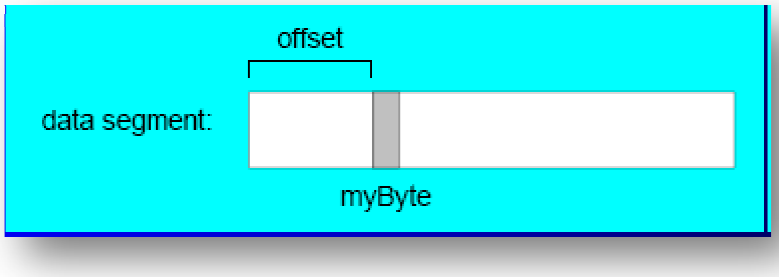
Operators:

- **OFFSET** - return the distance of variable from the start of segment.
- **PTR** - allows for override of variable's default size.
- **TYPE** - returns the size (in bytes) of an operand or each piece of an array.
- **LENGTHOF** - returns number of elements in an array.
- **SIZEOF** - returns number of bytes used by an array initializer (same as `LENGTHOF * TYPE`)

OFFSET Operator:

- The **OFFSET** operator returns the number of bytes between the label and the beginning of its segment.

  (`OFFSET = address` of a variable within its segment).



- In **FLAT** memory model, one address space is used for code and data.

  (`OFFSET = linear address` of a variable (32-bit number)).

Example 1: The OFFSET operator 8086

```
mov   bx, offset count   ; Let BX point to count
```

Example 2: The OFFSET operator with Flat Model

```
.data
bVal  BYTE  ? ;           <--- assume that bVal address
              ;          <-- is equal to 00404000
wVal  WORD  ?
dVal1 DWORD ?
dVal2 DWORD ?
mov esi,OFFSET bVal ;      <-----ESI= 00404000
```

```
mov esi,OFFSET wVal ;      <-----ESI= 00404001
mov esi,OFFSET dVal1;      <----ESI= 00404003
mov esi,OFFSET dVal2 ;     <-----ESI= 00404007
```

Example 3: The OFFSET operator with Direct-Offset Operands

```
.data
      bList  db  10h, 20h, 30h, 40h
      wList  dw  1000h, 2000h, 3000h
.code
      mov   di, offset bList   ; DI = 0000
      mov   bx, offset bList+1 ; BX = 0001
      mov   si, offset wList+2 ; SI = 0006
```

Example 4: The pointer Declaration

```
.data
      bigArray     DWORD 500 DUP(?)
      pArray       DWORD bigArray; pArray DWORD pointer to bigArray
.code
      mov   esi, pArray
```

PTR Operator:

- Assembly instructions require operands to be the same size. However, it may be required at some point to operate on data in a size other than that originally declared. This can be done with the **PTR** operator (override a variable's default size).
- For example, the PTR operator can be used to access the high-order word of a DWORD-size variable. The syntax for the PTR operator is:

```
type PTR expression
```

- Must be used in combo with a data type: BYTE, SBYTE, WORD, SWORD, DWORD, SDWORD, FWORD, QWORD, TBYTE.

```
.data
myDouble DWORD 12345678h
```

- Suppose want to move 5678 to AX?
  - Can NOT do

```
mov ax,myDouble ; since mismatched sizes
```

- Remember **little-endian** storage:

Doubleword	Word	Byte	Offset	
12345678	5678	78	0000	myDouble
		56	0001	myDouble + 1
	1234	34	0002	myDouble + 2
		12	0003	myDouble + 3

Example 1: The PTR Operator

```
.data
myDouble DWORD 12345678h
.code
mov ax,WORD PTR myDouble       ; 5678
mov ax,WORD PTR [myDouble+2]   ; 1234
mov bl,BYTE PTR myDouble       ; 78
```

- Can use it to move **smaller to larger too…**

Example 2: Small values into Larger Destination

```
.data
wordList WORD 5678h, 1234h
.code
mov eax, DWORD PTR wordList
```

TYPE Operator:

- Returns the size, in bytes, of a single element of a variable.
- **Syntax**:

```
TYPE var_name
```

  - BYTE: 1
  - WORD: 2
  - Etc.

```
.data
v1 BYTE  ?
v2 WORD  ?
v3 DWORD ?
v4 QWORD ?
TYPE v1   returns  1
TYPE v2   returns  2
TYPE v3   returns  4
TYPE v4   returns  8
```

Example 1: The Type Operator

```
.data
      var1   db    20h
      var2   dw    1000h
      var3   dd    ?
      var4   db    10, 20, 30, 40, 50
      msg    db    "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
```

Example 2: TYPE Operator with constant symbolic

```
.data
      ;declaration of arrays
      array_1  WORD  40 DUP (5)
      num      DWORD 4, 5, 6, 7, 8, 9, 10, 11
      warray   WORD  40 DUP (40 DUP (5))
      tarray   EQU   TYPE array_1     ; 2 bytes per element
      tnum     EQU   TYPE num         ; 4 bytes per element
      typ      EQU   TYPE warray      ; 2 bytes per element
```

LENGTHOF Operator:

- Returns number of elements in an array, base on values appearing on same line as its label

```
.data
byte1    BYTE  10,20,30
array1   WORD  30 DUP(?),0,0
array2   WORD  5 DUP(3 DUP(?))
array3   DWORD 1,2,3,4
digitStr BYTE  "12345678",0
LENGTHOF byte1    returns  3
LENGTHOF array1   returns  30+2
LENGTHOF array2   returns  5*3
LENGTHOF array3   returns  4
LENGTHOF digitStr  returns  9
```

Example: LENGTHOF Operator

```
.data
      ;declaration of arrays
      array_1  WORD      40 DUP (5)
      num      DWORD     4, 5, 6, 7, 8, 9, 10, 11
      warray   WORD      40 DUP (40 DUP (5))
      larray EQU   LENGTHOF array     ; 40 elements
      lnum   EQU   LENGTHOF num       ; 8 elements
      len    EQU   LENGTHOF warray    ; 1600 elements
```

- Be careful:

```
myArray BYTE 10,20,30,40,50
        BYTE 60,70,80,90,100
LENGTHOF myArray    is  5
```

- as opposed to

```
myArray BYTE 10,20,30,40,50,
             60,70,80,90,100
LENGTHOF myArray    is  10
```

SIZEOF Operator:

- The number of bytes taken up by a structure, Basically:

```
LENGTHOF * TYPE
```

Example 1: SIZEOF Operator

```
.data
intArray WORD 32 DUP(0)
.code
mov   eax,SIZEOF intArray ;  returns 64 = 32 * 2
```

Example 2: SIZEOF Operator

```
.data
      array_1      WORD   40 DUP (5)
      num          DWORD  4, 5, 6, 7, 8, 9, 10, 11
      warray       WORD   40 DUP (40 DUP (5))
      sarray   EQU SIZEOF array       ; 80 bytes
      snum     EQU SIZEOF num         ; 32 bytes
      siz      EQU SIZEOF warray      ; 3200 bytes
```

Directives

Directives:

- **ALIGN** - aligns a variable on a byte, word, double word, or paragraph boundary.
- **LABEL** - provides a way to redefine the same variable with different size attributes.

ALIGN Directives:

Syntax

```
ALIGN [bound]
```

- **Bound = 1, 2, 4, or 16** for byte, word, double word or paragraph boundary.
- Control how data is stored in memory.
- The assembler can insert one or more empty bytes before the variable to fix the alignment.
- CPU processes data stored at even addresses quicker than odd addresses.

Example 1: The ALIGN operator

```
.data
bVal1  BYTE  ? ;            <--- assume that bVal address
               ;           <-- is equal to 00404000
ALIGN 2
wVal  WORD  ?  ;            00404002
bVal2  BYTE  ? ;            00404004
ALIGN 4
dVal1 DWORD ?  ;            00404008
dVal2 DWORD ?  ;            0040400c
```

LABEL Directive

- Give the size of a variable without allocating any storage.
- Provide an alternate size for a variable without using PTR…
- All standard size attributes can be used with label , such as BYTE, SBTE, WORD, DWORD, etc

Example 1: The LABEL operator
(Contruct a smaller integers from a larger)

```
.data
val16 LABEL WORD
val32 DWORD 12345678h
.code
mov ax,val16          ax is 5678h
mov dx,[val16+2]      dx is 1234h
mov eax, val32
```

Example 2: The LABEL operator
(Contruct a larger integers from a smaller)

```
.data
LongValue LABEL DWORD
val1 WORD 5678h
val2 WORD 1234h
mov eax,LongValue      eax is 12345678h
```

Programming Exercises

Example 4: TYPE, LENGTHOF, SIZEOF, OFFSET, and PTR operators

```
TITLE Operators                (File: Operators.asm)
; Demonstration of TYPE, LENGTHOF, SIZEOF, OFFSET, and PTR operators
.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc
.data
byte1    BYTE  10,20,30,40
array1   WORD  30 DUP(?),0,0
array2   WORD  5 DUP(3 DUP(?))
array3   DWORD 01234567h,2,3,4
digitStr BYTE  '12345678',0
myArray  BYTE  10h,20h,30h,40h,50h,60h,70h,80h,90h
.code
main PROC
    ; Demonstrating TYPE operator
    mov al, TYPE byte1
    mov bl, TYPE array1
    mov cl, TYPE array3
    mov dl, TYPE digitStr
    ; Demonstrating LENGTHOF operator
    mov eax, LENGTHOF array1
    mov ebx, LENGTHOF array2
    mov ecx, LENGTHOF array3
    mov edx, LENGTHOF digitStr
    ; Demonstrating SIZEOF operator
    mov eax, SIZEOF array1
    mov ebx, SIZEOF array2
    mov ecx, SIZEOF array3
     mov edx, SIZEOF digitStr
     ; Demonstrating OFFSET operator
     mov eax, OFFSET byte1
     mov ebx, OFFSET array1
     mov ecx, OFFSET array2
     mov edx, OFFSET array3
     mov esi, OFFSET digitStr
     mov edi, OFFSET myArray
     ; Demonstrating PTR operator
     mov al,  BYTE  PTR array3
     mov bx,  WORD  PTR array3
     mov cx,  WORD  PTR myArray
     mov edx, DWORD PTR myArray
     exit
main ENDP
END main
```

Programming Exercises

Write down the value of each destination operand:

```
.data
varB BYTE 65h,31h,02h,05h
varW WORD 6543h,1202h
varD DWORD 12345678h
.code
mov ax,WORD PTR [varB+2] ;
mov bl,BYTE PTR varD ;
mov bl,BYTE PTR [varW+2] ;
mov ax,WORD PTR [varD+2] ;
mov eax,DWORD PTR varW ;
```