CHAPTER 4

Data Transfers, Addressing, and Arithmetic

Part 2

Data-Related Operators and Directives

Q (True/False): The OFFSET operator always returns a 16-bit value.
A False

Q (True/False): The PTR operator returns the 32-bit address of a variable.
A False

Q (True/False): The TYPE operator returns a value of 4 for doubleword operands.
A True

Q (True/False): The LENGTHOF operator returns the number of bytes in an operand.
A False

Q (True/False): The SIZEOF operator returns the number of bytes in an operand.
A True
**LENGTHOF & SIZEOF**
- LENGTHOF Counts the number of elements in a single data declaration.
- SIZEOF Counts the number of bytes in a data declaration.

```assembly
.DATA
array2    WORD    5 DUP(3 DUP(?))
.CODE
mov ecx, LENGTHOF array2    ; ecx = 15
mov ecx, SIZEOF array2      ; ecx = 30
```

**Q** If you declare an array that spans multiple program lines, LENGTHOF and SIZEOF only regard the data from the first line as part of the array. Given the following data:

```assembly
.DATA
myArray    WORD 10,20,30,40,50
            WORD 60,70,80,90,100
.CODE
mov ecx, LENGTHOF myArray    ; ecx = 5
mov ecx, SIZEOF myArray      ; ecx = 10
```

**Q** Alternatively, you can end the first line with a comma and continue the list of initializers onto the next line. Given the following data:

```assembly
.DATA
myArray    WORD 10,20,30,40,50,
            60,70,80,90,100
.CODE
mov ecx, LENGTHOF myArray    ; ecx = 10
mov ecx, SIZEOF myArray      ; ecx = 20
```

**ALIGN Directive**
- ALIGN directive aligns a variable in memory
- Syntax: ALIGN bound
- Where bound can be 2, 4, or 16

```assembly
.DATA ; Assume that
b1 BYTE   ?    ; Address of b1 = 00404000h
ALIGN 2    ; Skip one byte
w1 WORD    ?    ; Address of w1 = 00404002h
w2 WORD    ?    ; Address of w2 = 00404004h
ALIGN 4    ; Skip two bytes
d1 DWORD   ?    ; Address of d1 = 00404008h
d2 DWORD   ?    ; Address of d2 = 0040400Ch
```

**LABEL Directive**
- Assigns an alternate name and type to a memory location
- LABEL does not allocate any storage of its own
- Format: Name LABEL Type
Use the following data definitions for the next exercises:

```
data
myBytes BYTE 10h,20h,30h,40h
myWords WORD 3 DUP(?),2000h
myString BYTE "ABCDE"
```

Q Insert a directive in the given data that aligns `myBytes` to an even-numbered address.

A `.data
ALIGN 2
myBytes BYTE 10h,20h,30h,40h`

Q What will be the value of EAX after each of the following instructions execute?

```
mov eax,TYPE myBytes ; a.
mov eax,LENGTHOF myBytes ; b.
mov eax,SIZEOF myBytes ; c.
mov eax,TYPE myWords ; d.
mov eax,LENGTHOF myWords ; e.
mov eax,SIZEOF myWords ; f.
mov eax,SIZEOF myString ; g.
```

A (a) 1; (b) 4; (c) 4; (d) 2; (e) 4; (f) 8; (g) 5

Q Write a single instruction that moves the first two bytes in `myBytes` to the DX register. The resulting value will be 2010h.

A `mov dx, WORD PTR myBytes`

Q Write an instruction that moves the second byte in `myWords` to the AL register.

A `mov al, BYTE PTR myWords+1`
Q Insert a LABEL directive in the given data that permits myWords to be moved directly to a 32-bit register.

A myWordsD LABEL DWORD
  myWords WORD 3 DUP(?),2000h
.data
  mov eax,myWordsD

➢ Indirect Addressing
Q (True/False): Any 16-bit general-purpose register can be used as an indirect operand.

A False

Q (True/False): The BX register is usually reserved for addressing the stack.

A False

Q (True/False): The following instruction is invalid: inc [esi]

A True - (the PTR operator is required)

Q (True/False): The following is an indexed operand: array[esi]

A True

Points
- You can declare a pointer variable that contains the offset of another variable.

```assembly
.data
  arrayW WORD 1000h,2000h,3000h
  ptrW DWORD arrayW
.code
  mov esi,ptrW
  mov ax,[esi] ; AX = 1000h
```

Use the following data definitions for the remaining questions:

```assembly
  myBytes BYTE 10h,20h,30h,40h
  myWords WORD 8Ah,3Bh,72h,44h,66h
  myDoubles DWORD 1,2,3,4,5
  myPointer DWORD myDoubles
```

Q Fill in the requested register values on the right side of the following instruction sequence:

```assembly
  mov esi,OFFSET myBytes
  mov al,[esi] ; a. AL =
```
mov al, [esi+3] ; b. AL =
mov esi, OFFSET myWords+2
mov ax, [esi] ; c. AX =
mov edi, 8
mov edx, [myDoubles + edi] ; d. EDX =
mov edx, myDoubles[edi] ; e. EDX =
mov ebx, myPointer
mov eax, [ebx+4] ; f. EAX =

A (a) 10h; (b) 40h; (c) 003Bh; (d) 3; (e) 3; (f) 2

➤ JMP and LOOP Instructions
- JMP is unconditional, so a loop like this will continue endlessly unless another way is found to exit the loop.

```
top:
  ...
  JMP top ; repeat the endless loop
```
- The LOOP instruction creates a counting loop
- Syntax: LOOP destination
- Logic: ECX ← ECX – 1
- if ECX != 0, jump to destination label
- ECX register is used as a counter to count the iterations.

Copying a String Example:
Q (True/False): A JMP instruction can only jump to a label inside the current procedure.

A True

Q (True/False): The LOOP instruction first checks to see whether ECX is not equal to zero; then LOOP decrements ECX and jumps to the destination label.

A False

Q In real-address mode, which register is used as the counter by the LOOP instruction?

A CX

Q What will be the final value of EAX in this example?

```assembly
mov eax, 0
mov ecx, 10 ; outer loop counter
L1:
    mov eax, 3
    mov ecx, 5 ; inner loop counter
L2:
    add eax, 5
    loop L2 ; repeat inner loop
loop L1 ; repeat outer loop
```

A The program does not stop, because the first LOOP instruction decrements ECX to zero. The second LOOP instruction decrements ECX to FFFFFFFFh, causing the outer loop to repeat.