Lab 4: *Data Related Operators, Directives, Pointers & Loop*	**Session:** Fall 2019
Instructor(s): Sumaiyah Zahid & Fahim Ahmed	

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
```

Activity:

Initialize a double word array consisting of elements 61,43,11,52, 25. Sort the given array in ascending order in another array directly with the help of registers (you do not need to use a loop here). Use direct-offset addressing to access the array elements.

TYPE Operator

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

Syntax:

MOV reg16, TYPE mem

Example:

```
.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
```

Indexed Operator

Example in 32-bit mode:

Syntax:

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

Example:

```
.data
arrayB  db      20, 40, 60, 80
arrayW dw       100, 150, 250, 300
.code
mov esi, 1                      ;       SI = 0001
mov al, arrayB[esi]             ;       AL = 40
mov al, [arrayB + 3]            ;       AL = 80
mov esi, 2                      ;       SI = 2
mov cx, arrayW[esi]             ;       CX = 150
mov cx, [arrayW + 4]            ;       CX = 250
```

Example using scale factors:

Syntax:
constant [reg32 * TYPE constant]

Example:

```
.data
arrayW WORD 1000, 2000, 3000, 4000
.code
main PROC
mov ax, arrayW
mov esi, 1
mov ax, arrayW[esi * TYPE arrayW]
mov esi, 2
mov ax, arrayW[esi * TYPE arrayW]
mov esi, 3
mov ax, arrayW[esi * TYPE arrayW]
call DumpRegs
```

Activity:
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.

OFFSET Operator

The OFFSET operator returns the offset of a data label.
Syntax:
MOV reg32, OFFSET mem ; our 32-bit register now points to mem

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

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
```

Activity:

Use following array declarations:

```
arrayB      BYTE        5, 6, 2
arrayW      WORD        15, 5, 10
arrayD      DWORD       60, 12, 18
```

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: You can use PTR or any other directives/operators, if required.

Pointers

Syntax:
constant1 TYPE OFFSET constant2

Example:
```
.data
arrayW WORD  1000, 2000, 3000, 4000
ptrW DWORD OFFSET arrayW
.code
main PROC
mov eax, ptrW
```

Activity:
Initialize an array:
```
arr     DWORD       1000, 2000, 4000, 6000
```

Initialize four different pointer variables addressing each of the elements of this array.

ALIGN Directive

The ALIGN directive aligns a variable on a byte, word, double-word, 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
```

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 = 3
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
```

JMP instruction

Syntax:
Jmp destination

Example:
```
.code
top:
        ; any statements
jmp top
```

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

Example:

```
.data
        var1 DWORD 250
        var2 DWORD 125
        larger DWORD ?
.code
        mov     eax, var1
        mov     larger, eax
        mov     ebx, var2
        cmp     eax, ebx
        jae     L1
        mov     larger, ebx
        L1:
        exit
```

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

LOOP instruction

Syntax:
Loop destination

Example 1:

```
.data
intArray WORD 100, 200, 300, 400, 500
.code
main PROC
mov esi, 0
mov ax, 0
mov ecx, LENGTHOF intArray
L1:
        mov ax, intArray [esi]
        add esi, TYPE intArray
loop L1
```

Example 2:

```
.code
mov eax, 0
mov ebx, 0
mov ecx, 5

L1:
        inc eax
        mov edx, ecx
        mov ecx, 10
        L2:
                inc ebx
        loop L2
        mov ecx, edx
loop L1
```

Activity:
Use a loop with direct or indirect addressing to reverse the elements of an integer array in place. Do not copy elements to any other array. Use SIZEOF, TYPE and LENGTHOF operators to make program flexible.

Write a program that uses a loop to calculate the first ten numbers of Fibonacci sequence.