אופרטור LENGTH

האופרטור LENGTH מחזיר את מספר יחידות הזיכרון שהוקצו עבור משתנה באמצעות **DUP**

דוגמא:

```
DATA SEGMENT
dates_array dw 100h DUP (1998)
DATA ENDS
CODE SEGMENT
...
mov ax, LENGTH dates_array
```

ההוראה תתורגם ע"י האסמבלי :

```
mov ax, 100h
```

❖לגבי משתנים שהוגדרו ללא שימוש בDUP, יחזיר LENGTH את הערך 1

אופרטור SIZE

האופרטור SIZE מחזיר את תוצאת הכפל בין LENGTH (מספר יחידות האחסון) לTYPE (מספר הבתים בכל יחידה)

דוגמא:

```
DATA SEGMENT
dates_array dw 100h DUP (1998)
DATA ENDS
CODE SEGMENT
mov bx, SIZE dates_array
```

הוראה תתורגם ע"י אסמבלי :

```
mov bx, 200h
LENGTH(100h)*TYPE(2)
```

➢השימוש באופרטרים LENGTH וSIZE מקל על ביצוע של שינויים בתוכניות

```
DATA SEGMENT
input_buffer db 128 DUP (0)
DATA ENDS
CODE SEGMENT
...
mov cx, SIZE input_buffer
rep movsb
```

אופרטורים אריתמטיים

Operator	תיאור פעולה	דוגמא	תרגום אסמבלר
+	פעולת חיבור	Z db 'ASSEMBLY' mov Z+4, 'W'	mov [0004], 'W'
-	פעולת חיסור	copy_str DB 100h DUP(?) end_of_range DB '$CC' mov di, OFFSET end_of_range-1	mov di, 0ffh
*	פעולת כפל	Dim1 equ 20 Dim2 equ 40 Buff db Dim1*Dim2 DUP (?)	Buff db 800 DUP(?)
/	פעולת חילוק	cmp dx, Dim2/Dim1	Cmp dx, 2
Mod	שארית מחילוק	mov ax, 100h mod 17	mov ax, 1
SHL	הזזה שמאלה של האופרנד	mov bx, 0EFh shl 2	mov bx, 3BCh
SHR	הזזה ימינה של האופרנד	cmp ah, 14h shr 4	cmp ah, 01h
HIGH/ LOW	מחזיר את הערך הגבוה או הנמוך של האופרנד	mov al, HIGH (0ABC1h) mov al, LOW (0ABC1h)	mov al, 0ABh mov al, 0C1h

הרחבה

אופרטור THIS

האופרטור THIS משמש ליצירת סמל מכל טיפוס רצוי (dword ,word ,byte)

הסמל החדש יוגדר במקום בזיכרון שאליו הגיע האסמבלר בעת תרגום ההוראה המכילה את THIS

BYTE_VAR equ this byte/word

שימושי כדי לתת הגדרה חלופית לגודל משתנה

דוגמא:

```
Byte_var equ this byte
word_var dw 0
```

```
inc byte_var
```

עכשיו ניתן לגשת אל הבית הראשון של word_var באמצעות הסמל byte_var כפי שמדגימה ההוראה הבאה

אופרטור THIS דוגמא 2

הר

```
AB_word equ THIS word
first_byte        db AAh
second_byte       db BBh
```

```
mov ax, AB_word
```

עכשיו ניתן לגשת אל צמד הבתים כאילו הוגדרו כמילה כפי שמדגימה ההוראה הבאה

הוראה זו מעתיקה לax את 0BBAAh

קדימות אופרטורים

	האופרטור
גבוהה ביותר	ביטוי בסוגריים [ ] או ( )
	length, size
	פעולת אילוץ סגמנט
	ptr,offset,seg,type,this
	high,low
	*,/,mod,shl,shr
	+,-
	EQ,NE,LT,LE,GT,GE
	NOT
	AND
	OR,XOR
נמוכה ביותר	SHORT

הוראות מדומות להגדרת נתונים

כל הוראות הגדרת הנתונים שקימות באסמבלי הם למעשה הוראות מדומות:

DB = Define Byte

DW = Define Word

DD = Define Doubleword

DQ = Define Quadword

DT = Define Tenbytes

סוגי המשתנים הקיימים באסמבלי (Type):

Byte
Word
Dword
Qword
Tbyte

הוראות נפוצות נוספות להגדרת נתונים הם:
label ו equ

ההוראה המדומה LABEL

name LABEL type

name או שם משתנה כלשהוא

Type הוא טיפוס משתנה byte\word\dword\qword\tbyte

ההוראה LABEL מאפשרת לתת כמה שמות לנקודה כלשהיא בתוכנית או לשטח זיכרון

השימוש בlabel יתן תוצאות זהות לשימוש ב**equ THIS**

- ניתן להשתמש בשם AB_word לצורך גישה לתוכן צמד הכתובות first_byte ו second_byte כמילה אחת
- ניתן להשתמש בLABEL גם לצורך גישה לרצף של בתים במערך

```
AB_word LABEL word
first_byte      db AAh
second_byte     db BBh
```

```
arrayB LABEL byte
arrayA dw 100 dup(?)
```


הוראות מדומות - ORG

ORG num

כאשר התוכנית מקצה זיכרון להוראות או נתונים, מוקצים תאי הזיכרון לפי סדר הרישום בתוכנית.
כך קשה לדעת מה יהיה הכתובת המדויקת של נתון או הוראה.

הקצאת הזיכרון הבאה תתחיל מכתובת num בתוך הסגמנט הנוכחי (היסט = num)

num יכול להיות גם ביטוי אבל הוא חייב להתרגם למספר אבסולוטי (כמו כל האופרטורים שמתורגמים בזמן האסמבלי)

הערך הבא שיוקצה בזיכרון יוקצה בהיסט 200h בסגמנט הנוכחי (בד"כ משתמשים ב data segment)

דוגמא:

ORG 200h

אופרטור $ מונה המקום

האופרטור $ מצביע לכתובת הפנויה הבאה שבו יאחסן האסמבלר הוראה או נתון מהתוכנית המתורגמת.בקיצור הוא מונה את המקום הבא בזיכרון.

שימוש בשילוב עם ORG :

ORG $+2

הקצאת הזיכרון הבאה תתחיל 2 בתים קדימה במקום מהמקום הנוכחי

הוראה מדומה EVEN

גורמת לקידום מונה המקום לכתובת הזוגית הקרובה ביותר. לא מקבלת פרמטרים

(אם אנחנו כבר בכתובת זוגית?)

EVEN

★ יכול להיות שימושי בגישה למחסנית

אופרטור EQU והאופרטור =

equ משמש להגדרת שם או ערך

דוגמא:

```
enter equ mov
address equ DS:[BP]
initilaizer equ ax, ds
```

דוגמא:

```
enter initilaizer
enter ax, address
```

יתורגם ל:

```
mov ax,ds
mov ax, DS:[BP]
```

שם שהוגדר בהוראת equ **אינו ניתן להגדרת מחדש**

טבלת סיכום הוראות מדומות להגדרת נתונים

הוראות מדומות להגדרת נתונים	שימוש
name **LABEL** type	הגדרת שם
DB,DW,DD,DQ,DT	הקצאה ואתחול של משתנים
name **EQU** expression	הגדרת קבוע סימלי
name = expression	הגדרת קבוע שניתן להגדירו מחדש

הגדרת מצביעים

איך מגדירים מצביעים באסמבלר?

בDATA SGMENENT : שם המשתנה הוא מצביע לכתובת בזיכרון

```
DATA SEGMENT
        var1    db 'A'
        var2    db 'B'
        var3    db 'C'
        var4    db '$'
        P       dw var1, var2, var3
DATA ENDS
```

P הוא מערך של מצביעים

מדוע גודלו DW?

מה יהיו הערכים בP?

```
ds:0000 41 42 43 24 00 00 01 00 ABC$
ds:0008 02 00 00 00 00 00 00 00
ds:0010 FB 52 03 04 37 00 00 00 √R♥♦7
ds:0018 09 00 00 00 1E 00 00 00 ○  ▲
ds:0020 00 00 00 00 07 00 00 00    •
```

איך משנים את P במהלך התוכנית?

ע"י שימוש בפקודות offset ו LEA נראה בהמשך...

כתובת פיזית physical address

כתובת פיזית (מוחלטת) תיוצג ע"י שני גדלים של 16 סיביות: מספר מקטע וההיסט. SEG:OFFSET

- כדי ליעל ולהגמיש את נהול הזכרון, כל 16 בתים יכול להתחיל מקטע חדש.
- פסקה **paragraph** - כתובת ההתחלה של מקטע תמיד מתחלקת ב- 16.

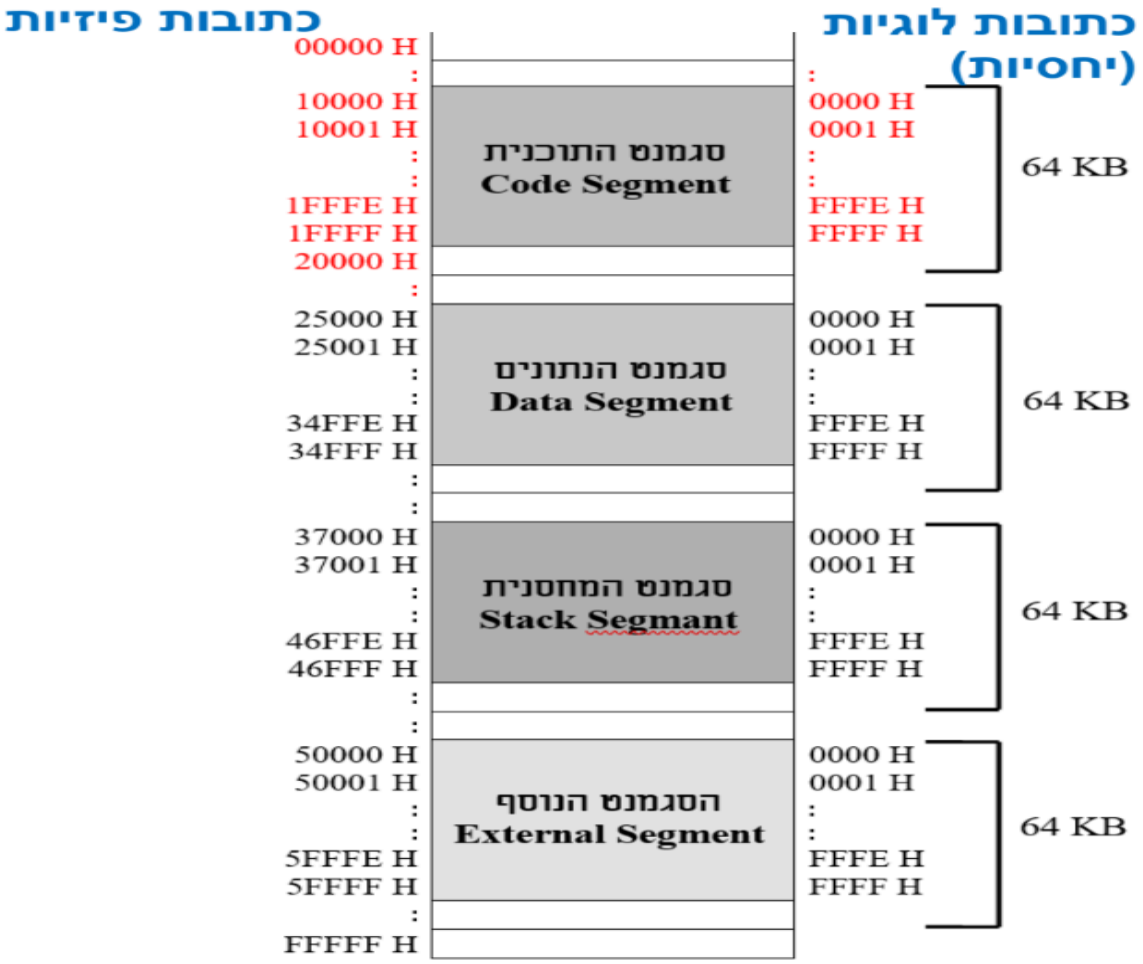
אם נתונה כתובת כלשהי במבנה **segment : offset** , הרי הכתובת המוחלטת תחושב ע"י הנוסחה:

$$\text{absolute address} = \text{segment} * 10_{16} + \text{offset}$$

חישוב זה מובנה באלקטרוניקה של המעבד.
צורת כתיבה זו מפשטת את החישובים של הכתובות: מוסיפים 0 מימין למספר המקטע, ומחברים את ההסט.

דוגמה: הכתובת **B000:8000** מחושבת בדרך הבאה:

כתובת המקטע	$B0000_{16}$
ההיסט	$+ \quad 8000_{16}$
הכתובת הפיזית	$B8000_{16}$


כתובות פיזיות
כתובות לוגיות
(יחסיות)
00000 H
10000 H
10001 H
1FFFE H
1FFFF H
20000 H
25000 H
25001 H
34FFE H
34FFF H
37000 H
37001 H
46FFE H
46FFF H
50000 H
50001 H
5FFFE H
5FFFF H
FFFFF H
סגמנט התוכנית
Code Segment
סגמנט הנתונים
Data Segment
סגמנט המחסנית
Stack Segmant
הסגמנט הנוסף
External Segment
0000 H
0001 H
FFFE H
FFFF H
64 KB

NEAR /FAR pointer

תכניות קטנות, אשר קטנות מ- 64KB, אינן זקוקות לכתובת המלאה בכל פניה לזכרון -

מאחר שהמקטע קבוע – די לציין את הכתובת היחסית (ההסט) בתוך המקטע

מצביע המכיל כתובת יחסית בלבד (16 סיביות) נקרא "מצביע קרוב" **Near Pointer**

מצביע המכיל כתובת מלאה (seg : off) נקרא "מצביע רחוק" **Far Pointer**

ביחס לפרוצדורות זה אותו דבר:

הפרוצדורה הנמצאת באותו מקטע נקראת (Near Call) Near Procedure

הפרוצדורה הנמצאת במקטע אחר נקראת (Far Call) Far Procedure

```
<procedure name>  PROC {FAR | NEAR}
            גוף השגרה
            RET
<procedure name>  ENDP
```


הוראות מדומות לניהול הזיכרון SEGMENT ו ENDS

הצירוף

```
name SEGMENT
.
.
name ENDS
```

משמש להגדרת יחידה לוגית הנקראית סגמנט.

שתי הוראות אלה מציינות את ראשיתו וסיומו של סגמנט אחד

כל ההוראות הרשומות בין SEGMENT לENDS שייכות לאותו סגמנט.

לכל קטע כזה יש לתת **שם**.

בנוסף עלינו לציין לMASM כאשר משתמשים ב SEGMENT וENDS איזה אוגר סגמנט משויך לכל סגמנט בתוכנית בשביל זה קיימת ההוראה המדומה **ASSUME**

הוראות מדומות לניהול הזיכרון ASSUME

בתחילת כל סגמנט קוד (או לפניו) עלינו להודיע לאסמבלר איזה ערך הוא יכול **להניח** שיהיה באוגרי הסגמנט בזמן תרגום ההוראות באותו סגמנט.

ההוראה ASSUME אינה גורמת להשמת ערכים לאוגרי הסגמנט זוהי רק הצהרה אלו ערכים ניתן להניח (to assume) שימצאו באוגרי הסגמנט.

```
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:SSEG
start:
```

אסמבלר – ניהול זיכרון

השמת ערכים באוגר הסגמנט

- האוגרים CS ו SS מאותחלים **תמיד** ע"י מערכת ההפעלה (dosbox במקרה שלנו)
- על המתכנת <u>בעצמו</u> לדאוג להשמת ערכים לאוגרים DS ו ES

```
mov ax, DATA
mov ds, ax
mov ax, EXTRA
mov es, ax
```

שיטות הצהרת סגמנטים שונות בMASM

```
MODEL small
STACK 100h
DATASEG
; --------------------
; Your variables here
; --------------------
CODESEG
start:
        mov ax, @data
        mov ds, ax
```

```
.MODEL small
.STACK 100h
.DATA
; --------------------
; Your variables here
; --------------------
.CODE
start:
        mov ax, @data
        mov ds, ax
```

אלו קיצורים שנכנסו מאוחר יותר לאסמבלר
לסגמנטים שמות ברירת מחדל:
והם _TEXT,_DATA,STACK
ניתן לראות זאת ב <file name>.map שנוצר מהקישור (שלב הtlink)

הוראות מדומות לניהול הזיכרון SEGMENT

ניתן לרשום בהוראה המדומה SEGMENT עד 3 פרמטרים אופציונלים:

- align-type
- combine-type
- class-name

```
name SEGMENT [align] [combine] ['class']
.
.
name ENDS
```

פרמטר align-type עבור ההוראה SEGMENT

align-type מתאר את היישור (alignment) של כתובת

הכתובת בה יטען הסגמנט	השפעה	align-type
XXXX0h	ברירת המחדל. הסגמנט חיב להתחיל בכתובת המתחלקת ב 16d (גבול פסקה)	**PARA** (paragraph boundary)
XXX00h	סגמנט PAGE חייב להתחיל בכתובת המתחלקת ב256d (גבול דף)	**PAGE** (page boundary)
XXXXYh (Y זוגי)	סגמנט WORD חייב להתחיל בכתובת זוגית (גבול מילה, **<u>הסיבית</u>** הימנית ביותר היא 0)	**WORD**
XXXXXh	סגמנט BYTE יכול להתחיל בכל מקום בזיכרון	**BYTE**

פרמטר class-name עבור ההוראה SEGMENT

שם הclass הניתן לסגמנט צריך להופיע בין גרשיים בודדים

```
CSEG SEGMENT PARA 'CODE'
```

```
SSEG SEGMENT PARA STACK 'OUR_STACK'
```

אילוץ סגמנט על ידי ASSUME

```
DATA SEGMENT
        data_b db 41h
DATA ENDS
EXTRA SEGMENT
        screen_display db 2000h dup (?)
EXTRA ENDS
CODE SEGMENT
        ASSUME CS:CODE,DS:DATA, ES:EXTRA
                mov ax, DATA
                mov ds, ax
                mov ax, EXTRA
                mov es, ax
                mov al, data_b
                mov screen_display, al
                .
```

עד עכשיו אמרנו שברירת המחדל היא DS ואם רוצים לפנות לES צריך לאלץ סגמנט

```
mov es:screen_display, al
```

Assume על ES נותן לנו פתרון טוב יותר:
בשלב התרגום הassembler מסתכל באיזה segment יושב המשתנה (screen_display)
ומחפש האם יש רגיסטר (שניתן להניח) שמשויך אליו

```
mov al, data_b
mov screen_display, al
```

מתורגם →

```
mov al, ds:data_b
mov es:screen_display, al
```

שינוי סגמנטים תוך כדי ריצת התוכנית

ניתן לשנות את שיוך הסגמנטים על ידי assume תוך כדי ריצת התוכנית.

בתוכנית הזו יש משחקים שונים:

בפעם הראשונה שנכנסים ללופ על מה מצביע ds? DATA

בפעם השניה? EXTRA

```
DATA SEGMENT
        data_b db 41h
DATA ENDS
EXTRA SEGMENT
        screen_display db 2000h dup (?)
EXTRA ENDS
CODE SEGMENT
        ASSUME CS:CODE,DS:DATA, ES:EXTRA
                mov ax, DATA
                mov ds, ax
                mov ax, EXTRA
                mov es, ax
                mov al, data_b
                mov screen_display, al
        ASSUME DS:DATA, ES:DATA
                .
                .
```

```
next_letter:    inc data_b
                .
                .
                mov ax, extra
                mov ds, ax
                jmp next_letter
```

ההוראה המדומה RADIX.

.RADIX expression (2/3/4/...16)

- קובע מהו בסיס המספרים שהוא ברירת המחדל.
- בסיס ברירת המחדל ללא הוראת RADIX. הוא 10 ולכן מספר שנכתב בלי סיומת (d, b, h) הוא דצימלי

```
.RADIX 16
DATASEG
    ORG 10
    LEGALS db LEN-1, '+=&*-/'
    LEN=$-LEGALS
    P DW LEGALS
; --------------------------
; Your variables here
; --------------------------
SSEG SEGMENT STACK 'STACK'
    DB 100 DUP (?)
SSEG ENDS
```

פקודות LDS/LES

LDS *register, 32 bit pointer*

LES *register, 32 bit pointer*

- העברת מצביע 32 ביט (4 בתים בתצורת seg:offset):
  - 2 הבתים – בכתובת pointer+2 ו pointer+3 יעותקו לDS או ES
  - 2 הבתים – בכתובת pointer ו pointer+1 יעותקו לרגיסטר

```
ADR      DD MSG1
```

```
LDS DX, ADR
MOV AH, 9
INT 21H
```

המשתנה בזיכרון חייב להיות בגודל 32 ביט. 4 הבתים במצביע/במשתנה יעותקו.

פקודות העתקה/שינוי אוגר הדגלים LAHF/STHF

LAHF (Load Register AH from Flags)

הפקודה מעתיקה את ביטים 7, 6, 4, 2 ו0 אל הבית **התחתון** של אוגר הדגלים. מחליפה את הערכים הקודמים של דגל הסימן, האפס, נשא העזר, הזוגיות והנשא. דגלי הבקרה ודגל הגלישה לא מושפעים.

SAHF(Store Register AH into Flags)

הפקודה מעתיקה את הבית **התחתון** של אוגר הדגלים לאוגר ah. הפקודה לא מקבלת פרמטרים. (דגל הסימן, האפס, נשא העזר, הזוגיות והנשא מעותקים לביטים 7, 6, 4, 2 ו0 בהתאמה) הדגלים עצמם לא מושפעים.

שימו לב שהפקודה מאפשרת לגשת רק לבית התחתון מבין 2 הבתים של אוגר הדגלים

מיקום הדגלים באוגר:

	overflow	direction	interrupt	trap	sign	zero flag		auxiliary carry		parity		carry
	11	10	9	8	7	6		4		2		0

פקודות לשמירה ושיחזור של רגיסטר הדגלים PUSHF/POPF

PUSHF

דוחף את 16 הביט של אוגר הדגלים למחסנית (ss:sp)

POPF

הפקודה מעתיקה המילה שבראש המחסנית (ss:sp) לאוגר הדגלים