

פרק 6

הגדרת משתנים ופקודות

העתקה

- נלמד את הפקודות הבסיסיות של אסמבלי בחלקים:
 - הגדרת משתנים ופקודת העתקה (מצגת 6)
 - פקודות אריתמטיות, לוגיות, הזזה (מצגת 7)
 - פקודות השוואה, קפיצה ולולאות (מצגת 8)
- לאחר מכן נוכל לכתוב תוכניות שכוללות אלגוריתמים
 - מציאת האיבר הגדול ברשימה
 - מיון איברים ברשימה
 - חישוב ערכי איברים בסדרה
 - וכו'

הגדרת משתנים

- למה צריך להגדיר משתנים?
- הריצו את הפקודה הבאה:

```
mov al, [ds:1h]
```

- עקבו אחרי הזיכרון ב-DATASEG

- תזכורת: CTRL+G ואז לרשום ds:0

- עקבו אחרי al

- הבעיה: אם התוכנית מלאה הצבות זיכרון כאלה, יהיה קשה לדבג אותה

- האסמבלר מאפשר לנו לתת למיקומים בזיכרון שמות משמעותיים

◦ משתנים - Variables

- לדוגמה אם הכתובת ds:1h תיקרא age, ניתן יהיה לכתוב את הקוד שלנו

```
mov al,[age]
```

- סוגריים מרובעים - הערך שנמצא במיקום בזיכרון

◦ במקרה זה age=ds:1h

◦ [age] שווה לערך שנמצא בזיכרון במיקום ds:1h

- נעשית בתוך סגמנט ה-DATA
- Ds מצביע על סגמנט ה-DATA (בתנאי שדאגנו לזה...)

DATASEG

age db ?

day db ?

month db ?

שם המשתנה

גודל

ערך התחלתי

הגדרת משתנים בגדלים שונים

- משתנה בגודל בית – DB– Define Byte
- משתנה בגודל מילה – DW– Define Word
- משתנה בגודל מילה כפולה – DD– Define Double

DATASEG:

ByteVar **db** ? ; allocate byte

WordVar **dw** ? ; allocate word

DoubleWordVar **dd** ? ; allocate double word

CODESEG

start:

```
mov ax, @data
```

```
mov ds, ax
```

```
mov [var], 5
```

quit:

```
mov ax, 4c00h
```

```
int 21h
```

```
END start
```

- הגדירו משתנה בשם `var`, בגודל `byte`
- הקוד הבא מעתיק לתוך `var` את הערך `5`. העתיקו והריצו.
 - בידקו ש-`var` מקבל `5`.
 - הכניסו את השורה המודגשת להערה והריצו שוב. מה קרה?

ערכים signed ו-unsigned

DATA SEG:

Var1 db ?

Var2 db ?

CODE SEG

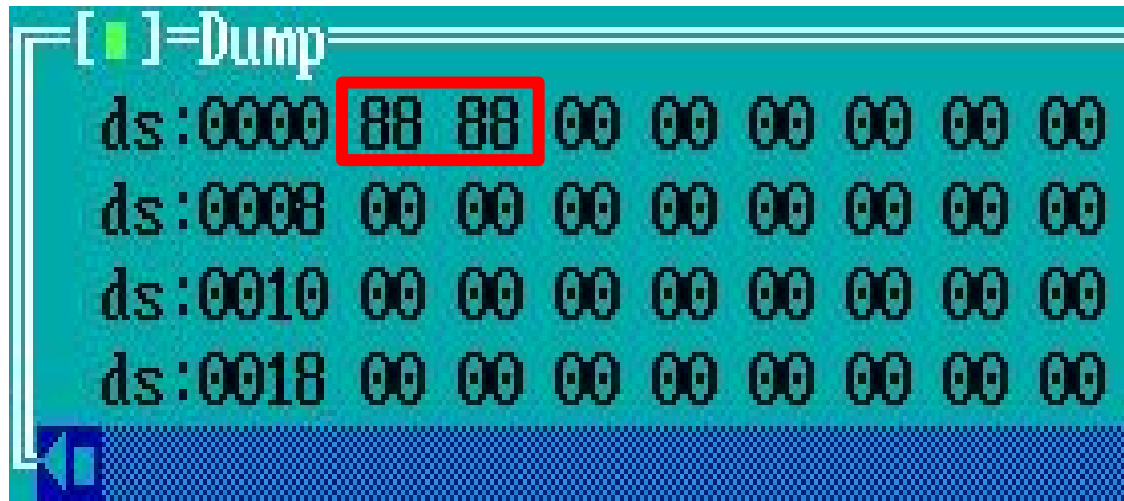
mov al, -120

mov [Var1], al

mov al, 136

mov [Var2], al

- באסמבלי אין משתנים unsigned ו- signed
- סוג המשתנה נקבע לפי הפרשנות שאנחנו מעניקים לערך שלו.
- לדוגמה:



הסבירו- מדוע
התוצאה נכונה

ערכים signed ו-unsigned

- האם העובדה שלשני ערכים שונים יש את אותו הייצוג בזיכרון, אינה גורמת לשגיאות חישוב?
- הוסיפו לערכים בדוגמה האחרונה את הערך +120 ובידקו את התוצאות.

$$\begin{array}{r}
 136 \\
 + \\
 \hline
 120 \\
 \hline
 ???
 \end{array}
 \qquad
 \begin{array}{r}
 -120 \\
 + \\
 \hline
 120 \\
 \hline
 ???
 \end{array}$$

קביעת ערך ראשוני למשתנים

DATASEG:

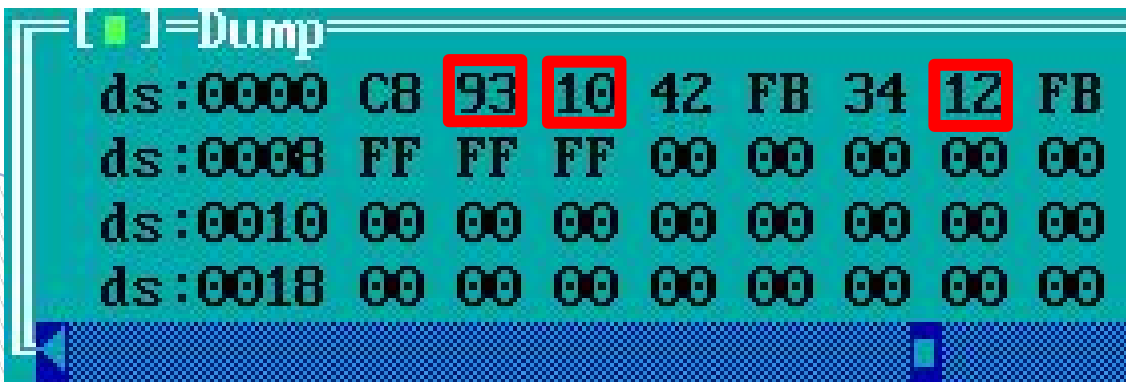
ByteVarName1	db 200
ByteVarName2	db 10010011b
ByteVarName3	db 10h
ByteVarName4	db 'B'
ByteVarName5	db -5
WordVarName	dw 1234h
DoubleWordVarName	dd -5

העתיקו את המשתנים ל-DATASEG ובידקו את תמונת הזיכרון.
מהם הערכים בכתובות ds:1? ds:2? ds:6?

קביעת ערך ראשוני למשתנים

DATASEG:

```
ByteVarName1      db  200      ; store the value 200 (C8h)
ByteVarName2      db  10010011b ; store the bits 10010011
                   ; (93h)
ByteVarName3      db  10h      ; store the value 16 (10h)
ByteVarName4      db  'B'      ; store the ASCII code of
the letter B (42h)
ByteVarName5      db  -5       ; store the value -5 (0FBh)
WordVarName       dw  1234h     ; 34h in low address, 12h in
                   ; high address
DoubleWordVarName dd  -5       ; store -5 as 32 bit format
                   ; (0FFFFFFFBh)
```



Little Endian, Big Endian

- כאשר הגדרנו את המשתנה:

WordVarName **dw1234h**

ראינו שהוא נשמר בזיכרון בצורה "הפוכה". מדוע? ומה הקשר לביצים?

```

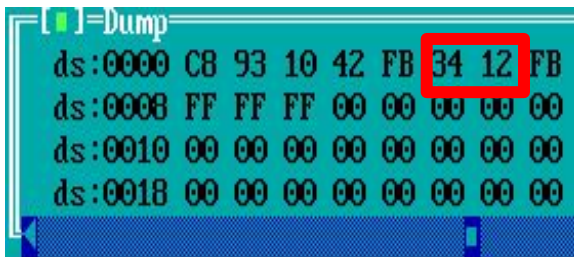
[ ]=Dump
ds:0000 C8 93 10 42 FB 34 12 FB
ds:0008 FF FF FF 00 00 00 00 00
ds:0010 00 00 00 00 00 00 00 00
ds:0018 00 00 00 00 00 00 00 00
  
```



Little Endian, Big Endian

ביטים 8 ביטים 8
עליונים תחתונים

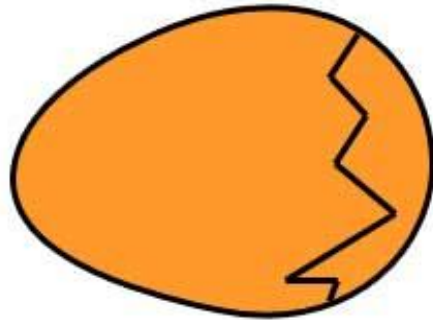
1234 ה



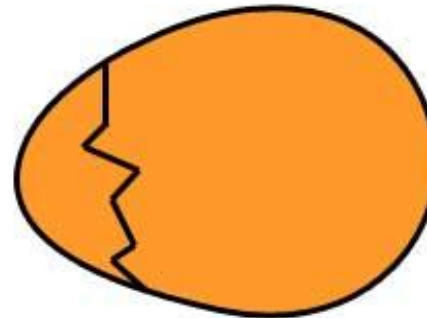
- שמירת מידע בזיכרון יכולה להיות באחת משתי שיטות:
 - Big Endian – הבית ה-High Order נשמר בכתובת הנמוכה יותר בזיכרון
 - Little Endian – הבית ה-High Order נשמר בכתובת הגבוהה יותר בזיכרון
- הכלל תקף לגבי כל משתנה או רגיסטר שגודלו יותר מבית אחד
- מעבד ה-8086 פועל בשיטת Little Endian

Little Endian, Big Endian

- המקור לביטוי הוא מהספר "מסעות גוליבר"



BIG ENDIAN - The way people always broke their eggs in the Lilliput land



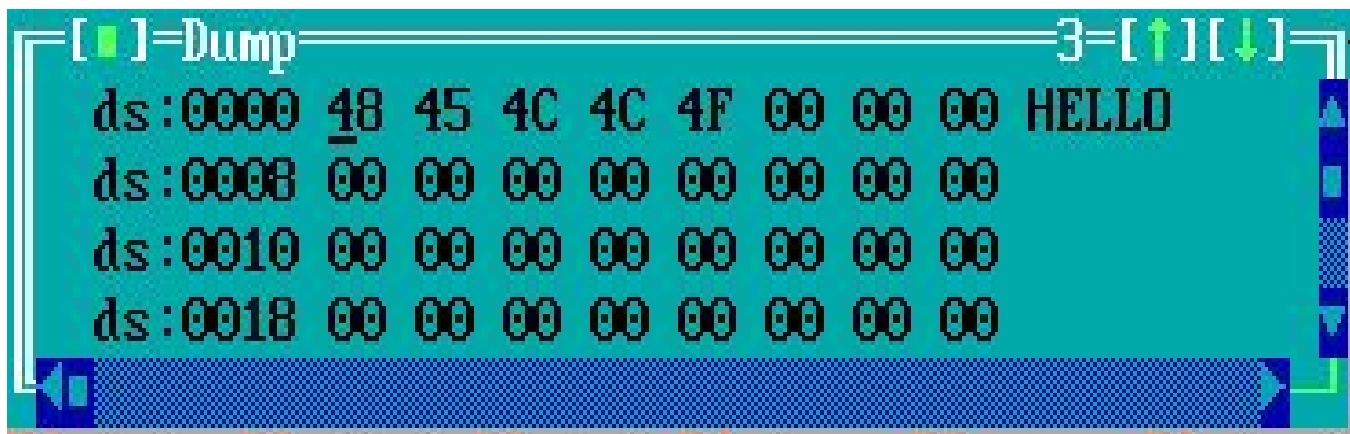
LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Little Endian – תרגיל

- מוגדרים בזיכרון הנתונים DATASEG ארבעה משתנים:
 - משתנה בגודל בית, שערכו ההתחלתי 1
 - משתנה בגודל מילה, שערכו ההתחלתי 1
 - משתנה בגודל בית, שערכו ההתחלתי 2-
 - משתנה בגודל מילה, שערכו ההתחלתי 2-
- כיצד ייראה הזיכרון? כיתבו במחברת / מחשב את ההשערה שלכם
- כיתבו תוכנית מתאימה ובידקו האם צדקתם ☺

הגדרת מחרוזת – String

ByteVarName db 'HELLO'



הגדרת מערך – Array

- מערך: אוסף של איברים בגודל זהה
 - מחרוזת: מקרה פרטי של מערך (האיברים הם תווים)
- הגדרת מערך בזיכרון:

ArrayName SizeOfElement N dup (?)

ערך התחלתי כמות האיברים גודל כל איבר שם המערך

איברים במערך

- לכל איבר במערך יש אינדקס
 - האיבר הראשון במערך הוא באינדקס 0
 - האיבר השני במערך הוא באינדקס 1 וכו'

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

הגדרת מערך – Array

- למערך יש כתובת התחלה
- כתובת תחילת המערך היא הכתובת של האיבר הראשון
- כתובת האיבר באינדקס N :

$$\text{ElementAddress} = \text{ArrayBaseAddress} + N * \text{ElementSize}$$

- נתון מערך של מילים words. המערך מתחיל בכתובת ds:000Eh. מה הכתובת של האיבר באינדקס 2?

ArrayOfTenFives db 10 dup (5)



ArrayOf1234 db 8 dup (1,2,3,4)



תרגילים – הגדרת מערכים

- הגדירו ב-DATASEG מערכים בגדלים שונים: 3, 5 ו-7 בתים
- הגדירו מערך של 10 בתים שמאותחלים לערך '5'. הגדירו מערך של 10 מילים שמאותחלות לערך '5'. השוו את תמונת הזיכרון בשני המקרים!
- הגדירו מערך ששומר 20 פעמים את הרצף 4,5,6, כל משתנה בגודל בית.
- הגדירו מערך בגודל word, ובו 10 איברים, שמאותחלים לערך 5.
 - מהו הערך בבית השלישי של המערך?

שיעורי בית

- צרו תוכנית, שמגדירה בתוך DATASEG מערך של בתים, שגודלו מספיק על מנת לשמור את השם שלכם. הגדרת המערך לא תכלול ערכים.
- מיד לאחר הגדרת המערך, הגדירו מחרוזת של בתים. הגדרת המחרוזת תכלול את מספר הטלפון שלכם.
- בתוך CODESEG כיתבו קוד שמעתיק לתוך המערך את השם שלכם, אות אחרי אות.
 - הדרכה לפעולת הכתיבה למערך תמצאו בספר הלימוד בעמודים 106-107

```
ds:0000  ???  BARAK GO
ds:0008  ???  NEN052-7
ds:0010  ???  909839
ds:0018  00 00 00 00 00 00 00 00
ds:0020  00 00 00 00 00 00 00 00
```

כעת, נלמד איך להעתיק...



פקודת העתקה mov

- mov – קיצור של move
- הפקודה מקבלת שני אופרנדים (operands) – מקור ויעד

mov destination, source

- יוצרת העתק של source בתוך destination
- איננה משנה את source

movax, 22 ; decimal
movax, 16h ; hexadecimal
movax, 00010110b ; binary

- אפשר להעתיק לרגיסטר ערך בבסיסי ספירה שונים
 - מספר דצימלי כשנוח לנו (לדוגמה כמות פעמים לחזרה על לולאה)
 - מספר הקסדצימלי כשמעבירים כתובת בזיכרון
 - מספר בינארי כשמבצעים פעולות על ביטים

mov – דוגמאות ב'

- mov מקבל כאופרנד את כל הרגיסטרים הכלליים

```
mov bx, 199  
mov cx, 2321  
mov dx, 10
```

- mov אינו מקבל באותו אופן רגיסטרים אחרים

```
mov cs, 100h  
mov ip, 10h
```

Illegal use of segment register

Undefined symbol: IP

mov – דוגמאות ג'

- העתקה מרגיסטר לרגיסטר

mov ax, cx

mov ax, dx

mov ax, ax

; legal, still does not change anything

mov ds, ax

; the correct method of changing
; segment registers

דרכים חוקיות להעתקה

mov register, register
mov register, constant
mov register, memory
mov memory, register
mov memory, constant

- דרכי רישום חוקיות:

mov memory, memory

- דרך רישום לא חוקית:
(חישובו - מדוע?)

העתקה מרגיסטר לרגיסטר

mov register, register

mov ax, bx ; 16 bit registers

mov cl, dh ; 8 bit registers

mov si, bp ; The mov instruction works ;
with ALL general purpose ; registers

mov ax, bl ; what will happen?

```
Assembling file: base.asm  
**Error** base.asm(10) Operand types do not match
```

תרגילים – העתקה מרגיסטר לרגיסטר

- העתיקו את ax לתוך bx .
- העתיקו את bx לתוך ax .
- העתיקו את ah לתוך ch .
- העתיקו את al לתוך dl .

העתקה של קבוע לרגיסטר

`mov` **register**, **constant**

`movcl, 10h`

`movah, 10`

; Note the difference from last command!
; 10 decimal, not 10h (=16)

`movax, 555`

• האם הפקודה הבאה חוקית?

`mov ah, 300`

העתקה של קבוע לרגיסטר – תרגילים

- העתיקו לתוך al את הערך 100 (דצימלי), בשלוש שיטות ספירה: בייצוג העשרוני שלו, בייצוג הקסדצימלי ובייצוג הבינארי.
 - הערה: ייצוג בינארי מסתיים באות b, לדוגמה 00001111b.
 - הריצו את התוכנית ב-TD ובידקו ש-al מקבל את הערכים הרצויים.

העתקה של רגיסטר לזיכרון

mov **memory**, **register**

mov[1], ax ; Direct addressing

mov[Var], ax ; Another form of direct addressing,
; using a variable

mov[bx], ax ; Indirect addressing

mov[bx+1], ax ; Indexed addressing

הסבר על Indirect Addressing:
הפקודה:

mov[1], ax

שקולה לפקודות:

movbx, 1

mov[bx], ax

תרגיל – העתקה של קבוע לזיכרון

- מה עושה התוכנית הבאה?
 - העתיקו, הריצו ועיקבו
אחרי DATASEG

```
IDEAL
MODEL small
Stack 100h
DATASEG
    Var db 0
CODESEG
start:
    mov ax,@data
    mov ds, ax
    mov al, 10h
    mov bx, 2
    mov [Var], al
    mov [1], al
    mov [bx], al
    mov [bx+1], al

quit:
    mov ax, 4C00h
    int 21h
End start
```

העתקה של תא בזיכרון אל רגיסטר

movregister, memory

movax, [1] ; Direct addressing

movax, [Var] ; Another form of direct addressing,
; using a variable

movax, [bx] ; Indirect addressing

movax, [bx+1] ; Indexed addressing

תרגיל: העתקה מהזיכרון לרגיסטר

```
mov [Var], al
mov [1], al
mov [bx], al
mov [bx+1], al
;-----1-----
mov al, 0
mov al, [var]
;-----2-----
mov al, 0
mov al, [1]
;-----3-----
mov al, 0
mov al, [bx]
;-----4-----
mov al, 0
mov al, [bx+1]
```

- כהמשך לתרגיל הקודם, הוסיפו את שורות הקוד הבאות, הריצו ובידקו מהו הערך ש-al מקבל בסיום כל קטע 1,2,3,4

העתקה של קבוע לזיכרון

mov **memory**, **constant**

mov[1], 5

mov[bx], 5

למעשה צורת הכתיבה הנכונה היא

mov[byte ptr 1], 5

mov[byte ptr bx], 5

או:

mov[word ptr 1], 5

mov[word ptr bx], 5

הסבר - בהמשך.

רגיסטרים חוקיים לגישה לזיכרון

AX •

BX •

CX •

SI •

DX •

DI •

רגיסטרים ייעודיים •

BP (למחסנית) •

רגיסטרי סגמנט •

SP (למחסנית) •

mov [cx], ax

mov [bx], ax

```
Assembling file:  base.asm  
**Error** base.asm(11) Illegal indexing mode
```

כללים לגישה לכתובת בזיכרון

- נשתמש ב-**bx** לשמירת הכתובת

```
mov    ax, [bx]
mov    [bx], ax
```

- אם נרצה היסט קבוע, ניתן להוסיף קבוע ל-**bx**

```
mov    ax, [bx+2]
mov    [bx+2], ax
```

- אם נרצה היסט משתנה, נוסיף ל-**bx** את **si** או **di**
 - אך לא את שניהם יחד - לא חוקי

```
mov    ax, [bx+si]
mov    ax, [bx+di]
mov    [bx+si], ax
mov    [bx+di], ax
```

סיכום – טעויות של מתחילים

```
mov al, bx
```

```
mov ds, 1234h
```

```
mov [var1], [var2]
```

```
mov [ax], cx
```

```
mov [bx], 5
```

```
mov 5, ax
```

- גדלי המקור והיעד אינם זהים
- העתקת קבוע לתוך רגיסטר סגמנט
- העתקה מזיכרון לזיכרון
- גישה לזיכרון בעזרת רגיסטר לא מתאים
- העתקת מידע בגודל נתון לפרשנות
- נסיון להעתיק מידע לתוך קבוע

חוקי או לא חוקי?



שאלות על syntax12(חוקי כתיבה)

חוקי או לא חוקי?



1

mov ax, 0Bh

חוקי או לא חוקי?



2

mov 0Bh, ax

חוקי או לא חוקי?



3

`mov[ax], 10h`

חוקי או לא חוקי?



4

mov ax, [bx]

חוקי או לא חוקי?



5 mov [bx], [bx]

חוקי או לא חוקי?



6

mov cx, si

חוקי או לא חוקי?



7

mov cx, [si]

חוקי או לא חוקי?



8 mov [cx], si

חוקי או לא חוקי?



9

`mov[bx+si], cx`

חוקי או לא חוקי?



10 `mov ax, [bx+dx]`

חוקי או לא חוקי?



11 `mov [bx+di+2], dx`

חוקי או לא חוקי?



12

`mov [bx+si+di], ax`

שיעורי בית

1. `mov ax, 0Bh`
2. `mov 0Bh, ax`
3. `mov [ax], 10h`
4. `mov ax, [bx]`
5. `mov [bx], [bx]`
6. `mov cx, si`
7. `mov cx, [si]`
8. `mov [cx], si`
9. `mov [bx+si], cx`
10. `mov ax, [bx+dx]`
11. `mov [bx+di+2], dx`
12. `mov [bx+si+di], ax`

- הריצו את הפקודות במחשב. צרו טבלה עם העמודות הבאות:
 - הפקודה
 - הודעת השגיאה (אם יש)
 - הפקודה לאחר שינוי - בצעו שינוי כלשהו בפקודה, שיהפוך אותה לבעלת syntax חוקי

תרגום אופרנד לכתובת בזיכרון

- שאלה: איך האסמבלר ממיר את האופרנדים הבאים (בגודל 16 ביט) לכתובות בזיכרון (בגודל 20 ביט)?

```
mov    [1], ax  
mov    [Var], ax  
mov    [bx], ax
```

- תשובה:

- האופרנדים מייצגים אופסט בלבד
- האסמבלר מניח שאנחנו מתייחסים לסגמנט ds
- כלומר [1] הוא למעשה [ds:1]

תרגום אופרנד לכתובת בזיכרון – דוגמה

```
mov     ax, 0AABBh
mov     [1], ax
```

ds:0000	00	BB	AA	00	00	00	00	00
ds:0008	53	54	41	52	54	10	0A	FF
ds:0010	26	81	78	88	02	00	00	00
ds:0018	00	00	00	00	00	00	00	00

תרגום אופרנד לזיכרון - דוגמה ב'

mov [es:1], ax

es	0000	CD	BB	AA	9D	00	EA	FF	FF
es:0008	AD	DE	32	0B	C3	05	6B	07	
es:0010	14	03	28	08	14	03	92	01	
es:0018	01	01	01	00	02	04	FF	FF	

העתקה ממערכים ואל מערכים

DATASEG:

```
Array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh, 0FFh
```

- העתקת האיבר באינדקס 2 לתוך al:

```
mov al, [Array+2]
```

- האסמבלר יתרגם את Array+2 מכתובת יחסית לכתובת קבועה

```
#base#10: mov al, [Array+2]  
cs:0005 A00200 mov al, [0002]
```

פקודת offset

- דרך נפוצה לטיפול במערכים היא להעתיק ל- bx את כתובת תחילת המערך:

```
mov     bx, offset Array
```

- העתקת האיבר באינדקס 2 לתוך al :

```
mov     al, [bx+2]
```

פקודת LEA

- LEA: Load Effective Address

- כמו פקודת offset
- התרגום הוא לאותו קוד מכונה

```
#base#10:  mov bx, offset Array
           cs:0005 BB0000          mov     bx,0000
#base#11:  lea bx, [Array]
           cs:0008 BB0000          mov     bx,0000
```

ההנחיה byte ptr / word ptr

- מה הבעיה בקוד הבא?

```
DATASEG
```

```
Array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh
```

```
CODESEG
```

```
...
```

```
mov ax, [Array+2]
```

```
Assembling file: base.asm  
**Error** base.asm(10) Operand types do not match
```

byte ptr / word ptr – המשך

DATASEG

Array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh

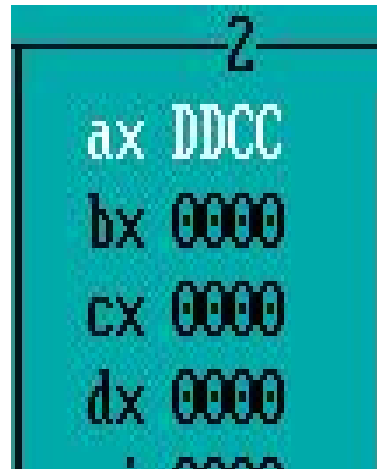
CODESEG

...

mov ax, [word ptr Array+2]

- צריך להודיע לאסמבלר לבצע העתקה של שני בתים

- תרגול – נסו זאת!



byte ptr / word ptr – המשך

```
mov [bx], 5
```

- מה הבעיה
בפקודה הבאה?

```
*Warning* basebg.asm(28) Argument needs type override
```

- ניתן להשתמש בזיכרון בגדלים שונים כדי לשמור את הערך 5
- באיזה גודל זיכרון להשתמש?
- צריך לתת הנחיה בעלת פרשנות אחת

```
mov [byte ptr bx], 5  
mov [word ptr bx], 5
```

- תרגול – נסו את שתי השיטות
והשוו את התוצאות

אתגר – שינוי קוד תוך כדי ריצה

- ניצור תוכנית שהקוד שלה משתנה תוך כדי ריצה!
- נניח שאנחנו פונים למקום בזיכרון [1], לדוגמה:

```
mov [1], al
```

- איך האסמבלר יודע לתרגם את [1] לכתובת מלאה בגודל 20 ביטים?

- למעשה הפקודה האחרונה זהה לפקודה הבאה:

```
mov [ds:1], al
```


אתגר – שינוי קוד תוך כדי ריצה

- מה יקרה אם נכתוב:

```
mov [cs:1], al
```

- מה יקרה אם נכניס לסגמנט הקוד ערכים שמייצגים opcodes אמיתיים בשפת מכונה?

תרגיל

- נתון קטע קוד:
- גירמו לכך שבסיום ריצת הקוד $ax=3$ ולא 4.
- גירמו לכך שבסיום ריצת הקוד $ax=bx=3$.
- מגבלות:
 - אין להוסיף שורות בתוך קטע הקוד, רק לפניו ואחריו.
 - מותר להשתמש רק בפקודות `mov`. אסורות קפיצות תוויות וכו'.
 - בקידוד אסור לפנות ישירות ל-`ax`, `bx` או `ah`, `al`, `bh`, `bl`.

...בהצלחה!

סיכום הפרק

- למדנו להגדיר בזיכרון ה-DATA משתנים בגדלים שונים
 - למדנו לאתחל אותם
 - למדנו ליצור מחרוזות
 - למדנו להגדיר מערכים
- למדנו כיצד לבצע העתקה מ/אל רגיסטרים ומ/אל הזיכרון
 - התמקדנו בצורות חוקיות ולא חוקיות של גישה לזיכרון
- למדנו כיצד ניגשים לזיכרון שהוא חלק ממערך
- למדנו אודות big endian / little endian