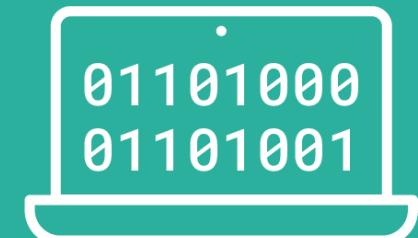




## ארQUITקטורה **אSEMBלי**



הדליקו מצלמות  
בבו מיקרופונים  
השתיקו טלפונים  
ארגנו מחברת וכלי בתיבה

# שיעור 3

## תנאים ולולאות

- מבוא למערכות הפעלה Windows
- כלי דיאגנוגטיקה ל-Windows
- Processes and Threads
- Memory
- Linux shell
- Shell המשר
- מערכות קבצים
- Bootstrapping
- פרויקט סיכום מערכות הפעלה



ארQUITקטורה  
**מערכות  
הפעלה**

- מבוא לאסמבלי
- אסמבלי – משתנים ופקודות
- תנאים ולולאות**
- מחסנית ופונקציות
- פרמטרים ופרוצדורות
- פסיקות
- סיכון אסמבלי
- תרגיל בסופות



ארQUITקטורה  
**אסמבלי**



## בשיעור הקודם

למדנו על פקודות בסיסיות ומשתנים

- למדנו על הפקודות הבסיסיות באסמבלי
- וגם כיצד עובד חישוב הכתבות
- תרגלנו את הפקודות השונות וחשוב בתובות
- היום נכיר דרכו בה אפשר לבצע את הפקודות השונות בסדר שונה

# חזרה

רענון

`mov ax, a5h`

מה לא בסדר?

גלה

`mov [200], 10`

מה הסכנה כאן?

גלה

`; cs=100h, ds=200h,  
; ss=300h, bx=10h`

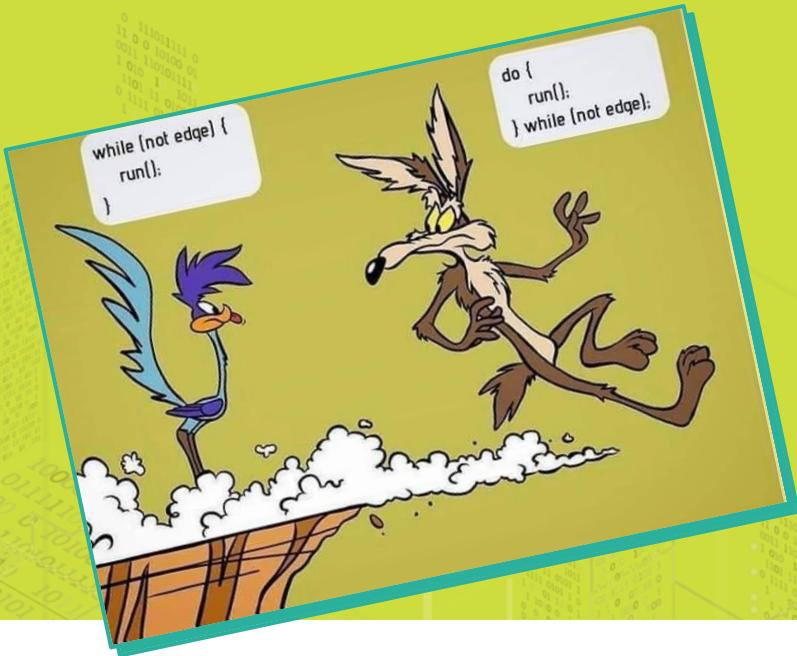
`mov al, [bx]`

מהי הכתובת ששמנה נעתיק  
ערך לאוגר? ah

גלה

# שיעור 3

## תנאים ולולאות



סיכום

לולאות

קפיצה מותנת

קפיצה

עבודה עם  
זיכרון

# אופרנד (Operand)

רענון

חלק מהפקודה המציין את הנתונים עליהם מבצעים את הפעולה

לדוגמא:

`mov AL, 97`

# משתנים ומיון

רענון

הגדרת משתנה מאופיינת על ידי גודל שם וערך התחלתי  
כל משתנה שומר בזיכרון בכתבוב (היסט)

```
num db 7
```

```
mov bx,offset num
mov ax,[bx]
; option B
mov ax, num
```

נשמר את הערך של המשתנה 7 באוגר AX

# שיטות מיעון

היכן מאוחסנים האופרנדים?  
בהוראה עצמה? באוגר? בזיכרון?

קיימות 3 שיטות מיעון מרכזיות:

1. **מיעון מיידי** – אופרנד המקור הוא נתון
2. **מיעון אוגר** – אופרנדים הם אוגרים
3. **מיעון זיכרון** – אחד האופרנדים הוא משתנה בזיכרון



# שיטות מייעון

## מייעון זיכרון

אחד האופרנדים הוא  
משתנה בזיכרון  
למשל:

mov ax, [bx]

## מייעון אוגר

אופרנדים הם אוגרים  
למשל:

mov ax, bx

## מייעון מיידי

אופרנד המקור הוא נתון  
למשל

mov ax, 5

## שיטת מיעון - מיעון זיכרון

את מיעון זיכרון (מהשך הקודם) ניתן לחלק לשתי קבוצות

**מיעון ישיר** – בו מצוינת הכתובת בזיכרון בצורה  
מפורשת

**מיעון עקיף** – בו מצוינת הכתובת בזיכרון בצורה  
עקביה

# שיטות מיעון – מיעון זיברון

## מיעון זיכרון

אחד האופרנדים הוא  
משתנה בזיכרון.  
למשל:  
למשל:

`mov ax, [bx]`

## חישוב אוניבר

### מיעון ישיר

למumble.

### מיעון עקיף

`mov ax, bx`

## מיעון מיידי

אופרנד המקור הוא נתון  
למשל

`mov ax, 5`

# שיטות מיון - מיון זיכרון

מיון ישיר  
בו מצוינת הכתובת בזיכרון  
בצורה מפורשת  
למשל:

mov al,**ds:[0x100]**

# שיטות מיון - מיון זיכרון

עקיף בעזרת אוגר

מיון אינדקס ישיר

מיון בסיס

מיון אינדקס בסיס

מיון עקיף

- בו מציינת הכתובת  
בזיכרון בצורה עקיפה

ישנם 4 שיטות של מיון  
עקיף

מיון ישיר

בו מציינת הכתובת בזיכרון  
בצורה מפורשת  
למשל:

mov al,[ds:0x100]

# שיטות מיעון - מיעון זיברון

## מיעון בסיס

דומה למיעון אינדקס ישיר  
השונה באוגרים  
האוגרים שיכולים לשמש  
במצבייעים: *ax, bp*  
למשל:

`mov ax,[bx+4]`

## מיעון אינדקס ישיר

הערך בין הסוגרים [] אוגר  
אינדקס בתוספת קבוע  
למשל:  
האוגרים שיכולים לשמש  
במצבייעים: *di, si*

`mov ax,[si+8]`

## עקיף בעזרת אוגר

הערך בין הסוגרים [] אוגר  
למשל:  
האוגרים שיכולים לשמש  
במצבייעים: *di, si, bx*

`mov ax,[bx]`

# שיטות מיעון - מיעון זיברון

מיעון בסיס  
דוגמא למיעון אינדקס ישיר  
השונה באוגרים  
האוגרים שיכולים לשמש  
במצביים: `ax`, `bx`,  
למשל:

```
mov ax,[bx+4]
```

**מיעון אינדקס בסיס**  
משלב את שני המיעונים  
אינדקס ובסיס  
למשל:

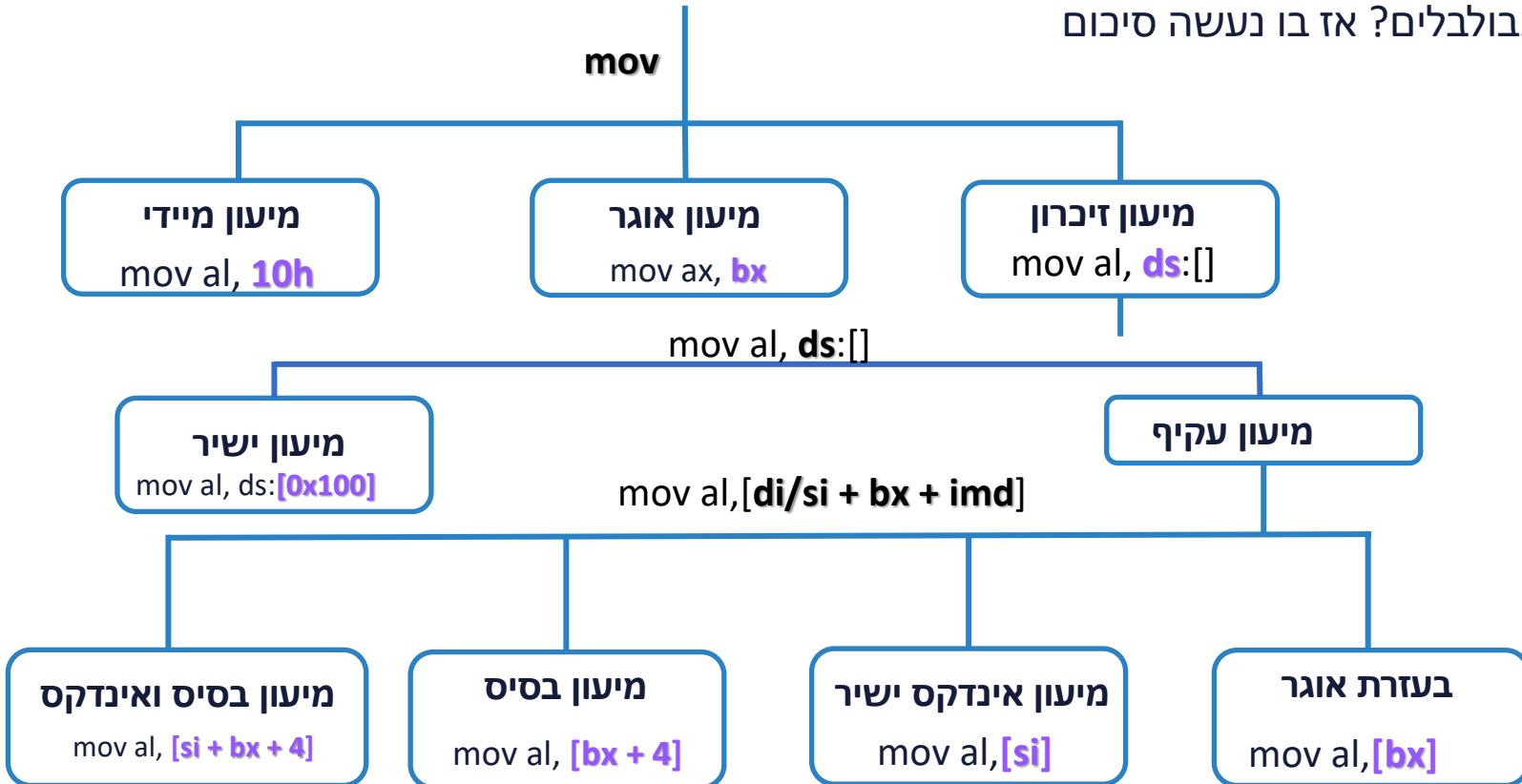
```
mov ax,[bx+si+8]
```

מיעון אינדקס ישיר  
הערך בין הסוגרים [ ] אוגר  
אינדקס בתוספת קבוע  
למשל:  
האוגרים שיכולים לשמש  
במצביים: `di`, `si`

```
mov ax,[si+8]
```

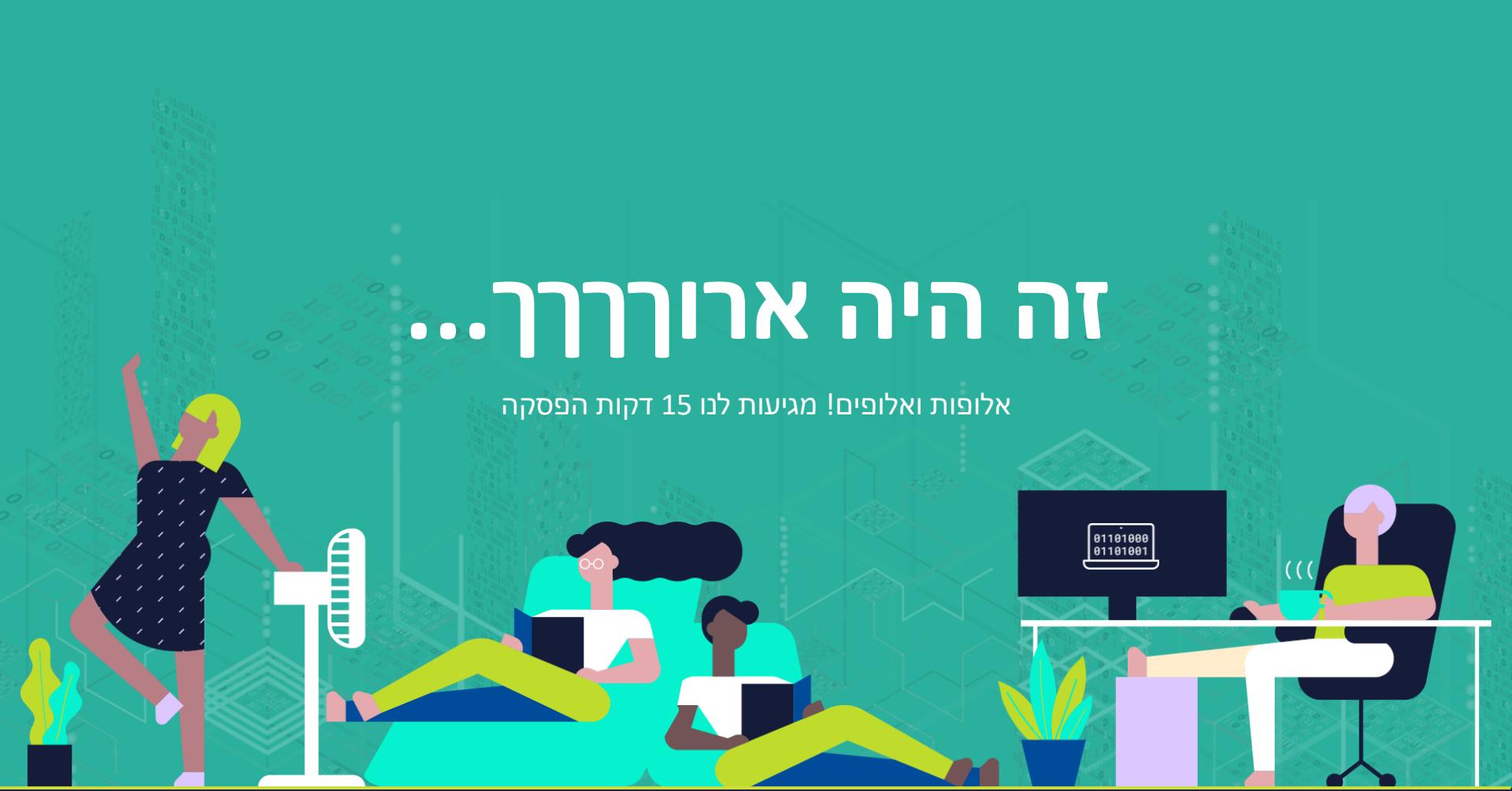
# סיכום ביניים

מבולבלים? אז בו נעשה סיכום



# זה היה אורור...

אלופות ואלופים! מגיעות לנו 15 דקות הפסקה





# הפקודה LEA (Load Effective Address)

- בדומה להוראת offset , הפקודה lea מאפשרת לטען כתובות לאוגר
- בשונה מ lea, offset מתבצעת בזמן ריצה

```
jmp start
num1 dw ?
start:
    mov ax, offset num1
    lea ax, num1
```

## האם הפקודה תקינה או לא?

ב

Mov ax,offset [bx+si+ 2]

לא תקין

א

lea ax,[bx+si+2]

תקין

# הפקודה - LEA - דוגמאות

הפעולות הבאות שקולות:

1

```
lea ax,[0]
;mov ax,[0]
```

2

```
lea ax,[bx]
;mov ax,bx
```

3

```
lea ax,[bx + si + 2]
;mov ax,bx
;add ax,si
;add ax,2
```



# הפקודה LEA (Load Effective Address)

למה זה יכול להיות שימושי?

- לבצע חישוב כתובת בפקודה אחת
  - מוביל לשנות את ערכי הדגלים
- לחשב כתובת של איבר במערך

למשל במערך של משתנים מסוג short, רצה למשוך לולאה שתעבור על איברי המערך. מספיק שנקדם את גודל המשתנה (2 בתים) כל פעם ונשתמש בשורה הבאה:

lea ax, [bx + si]

# שיעור 3

## תנאים ולולאות

**! false**

It's funny because  
it's true

סיכום

לולאות

קפיצה מותנת

**דגלים  
וקפיצות**

עבודה עם  
זיכרון



# JMP קפיצה שאינה מותנית

- הוראות בקרה, הדורך שלנו לשלוט בקוד Flow Control

JMP 0xBA34	קפוץ לכתובת שמצוינה ;
JMP label	קפוץ לכתובת של התווית ;
JMP bx אוגר	קפוץ לכתובת הנמצאת בתוך ;

mov IP, address

הפקודה JMP מבצעת את הפעולה הבאה למעשה:

```
start:  
    mov ax, 3  
    jmp start
```

- דוגמא:

# תזכורת – מספרים מכוונים ובלתי מכוונים

**מספרים מכוונים (signed)** – מספרים שליליים וחוביים יחד. מספרים שליליים יוצגו בשיטת המשלים ל-2.

**מספרים בלתי מכוונים (unsigned)** - מספרים חיוביים (כולל 0) בלבד.

לדוגמא: רוחבו של האוגר AX במעבד 8086 הוא 16 סיביות.

נניח שערך כרגע הוא 1000 1000 1000 1000

אם אנחנו מתיחסים למספר כ-**signed**, אזי ערכו -30,584

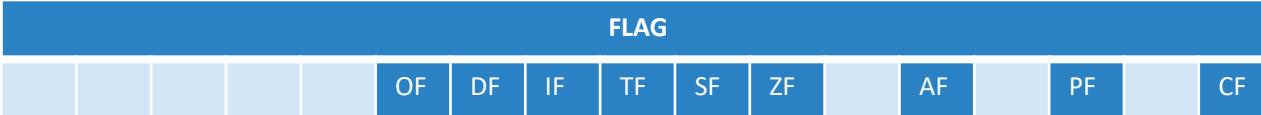
אם אנחנו מתיחסים למספר כ-**unsigned**, אזי ערכו 34,952



# אגיר הדגלים (Flag register)

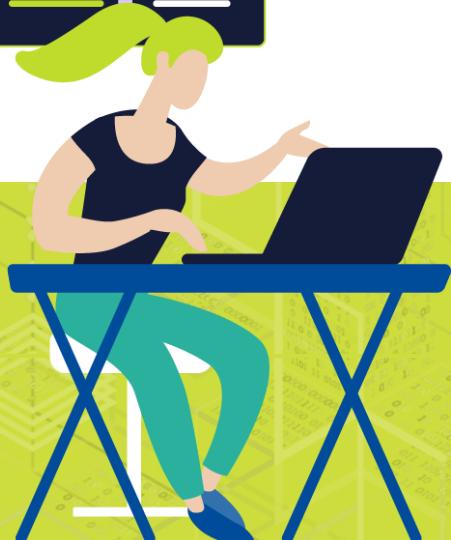
FLAG

למדנו על אוגרים שונים ותפקידים ועבשו הגיעו תרו של אוגר בגודל 16 סיביות כאשר 9 מהם בשימוש כאשר כל אחד בנפרד מהוות דגל נפרד עם ערכי 0 או 1



# Live Coding

פיתוח בעורף קוד ↶

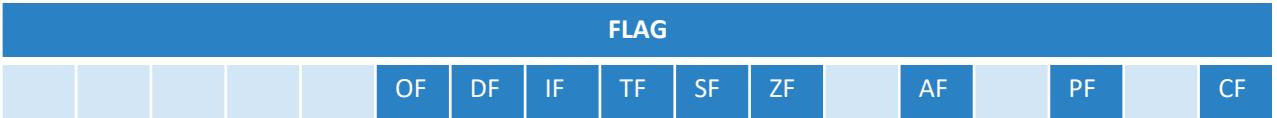


נצח ביצד ניתן לעקוב אחרי הדגמים  
עקוב אחרי השינוי שלהם אחורי פעולות





# הסבר קצר על כל דגל



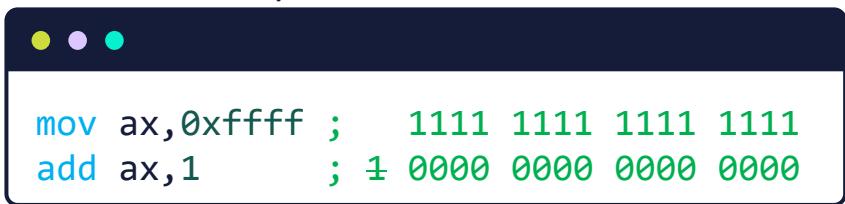
- ZF - דולק אם בפקודה האחרונה התקבל 0
- SF - דולק אם בפקודה האחרונה התקבל ערך שלילי
- CF - דולק אם בפקודה האחרונה התקבל נשא\*
- OF - דולק אם בפקודה האחרונה התקבל סימן תוצאה שגוי (באשר מתייחסים למספרים במכונין)
- PF - דולק אם בפקודה האחרונה מספר הביטים הדולקים ב- LSB הוא זוגי

# Carry flag

בד"כ נבדוק את הדגל CF כאשר נעבד עם מספרים לא מכוונים.

ה- CF יידלק במספר מצבים:

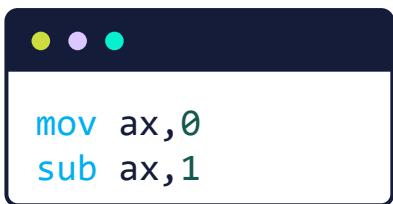
- אם לאחר פעולה חיבור הערך שהתקבל היה בעל יותר סיביות ממה שהאוגר יכול לאחסן. לדוגמה:



The screenshot shows a debugger interface with three colored dots at the top left. The assembly code and its binary representation are displayed in two columns:

Assembly	Binary
mov ax, 0xffff ;	1111 1111 1111 1111
add ax, 1 ;	1 0000 0000 0000 0000

- אם בשבייל לבצע פעולה חיסור היינו צריכים "ללוות 1" מחוץ לאוגר. לדוגמה:



The screenshot shows a debugger interface with three colored dots at the top left. The assembly code is shown in two lines:

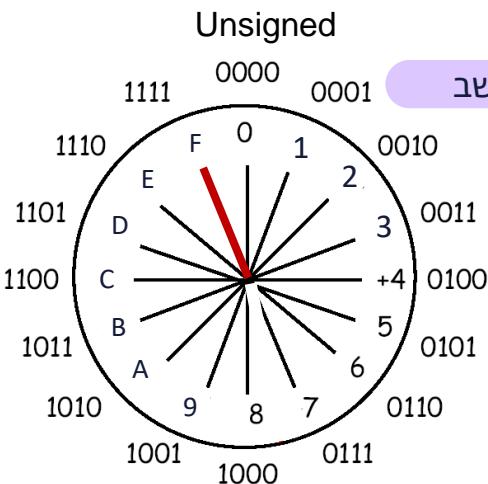
```
mov ax, 0  
sub ax, 1
```



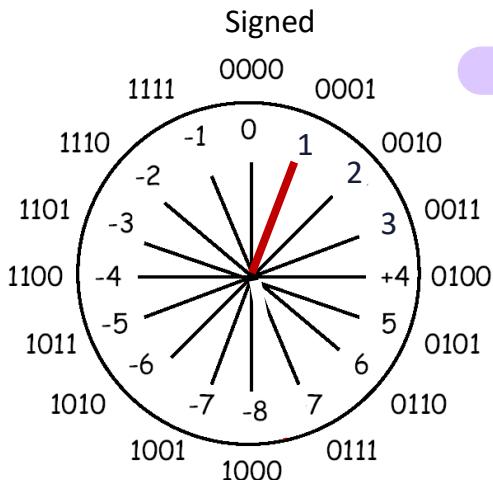
# באיילו מקרים CF נדלק עבור הפקודות ?SHL/SHR

```
mov ax, 0xffff ; 1111 1111 1111 1111  
shl ax, 1
```

# Overflow vs Carry



חשב +



חשב +

# Parity Flag

ידלק כאשר מספר הביטים הדולקים ב - LSB (הבית הפחות משמעותי) בתוצאה הפעולה האריתמטית الأخيرة הוא זוגי. יכבה כאשר היא אי-זוגית.



```
mov ax,2 ; 0000 0000 0000 0010  
add ax,4 ; 0000 0000 0000 0100
```



# מדוע ידליך דגל ה-PF?

נבעה פעולה של חיבור של 4 + 1 ונקבל 5 מספר אי-זוגי

```
mov ax,1 ; 0000 0000 0000 0001  
add ax,4 ; 0000 0000 0000 0100
```

# הערה חשובה

חשובי!

שימוש לב שלא כל האופקודים מושנים את הדגלים!

בא לבדוק תמיד בקורס 8086 Intel לדוגמא:

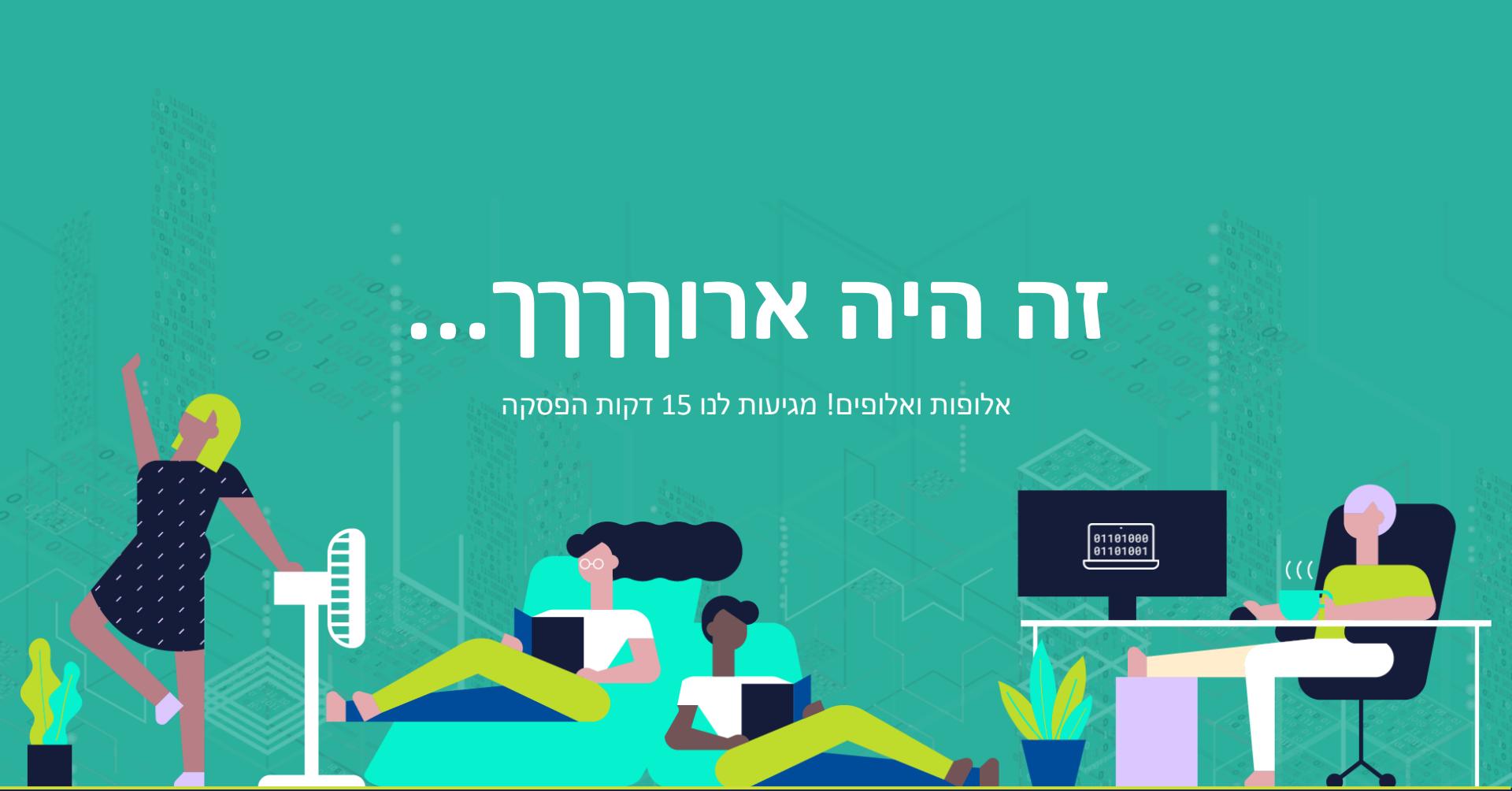
SUB - Subtract

Usage: SUB dest,src

Modifies flags: AF CF OF PF SF ZF

# זה היה אורור...

אלופות ואלופים! מגיעות לנו 15 דקות הפסקה



# שיעור 3

## תנאים ולולאות

**! false**

It's funny because  
it's true

סיכום

לולאות

קפיצה מותנת

דרכים  
וקפיצות

עבודה עם  
זיכרון

# קפיצה מותנית

למה אנחנו צריכים את כל הדגלים האלה?

- השימוש המרבי בדגלים הוא של **קפיצות מותניות**
- קיימים אופקודים של קפיצה **המוחתניים בערכיהם של דגלים מסוימים**
  - לדוגמא: תבצע קפיצה אם התוצאה יצאא אף (ז"א, תבדוק אם הדגל ZF דлок)
  - קיימת הבחנה בין אופקודים הפעילים על מספרים מכוונים (signed) לבין אופקודים הפעילים על מספרים בלתי מכוונים (unsigned)

# מספרים מכוונים ובלתי מכוונים

תיאור	הוראות עבר מספרים מכוונים	התנאי	הוראות עבר מספרים בלתי מכוונים	התנאי
קפוץ אם גדול ממש	<b>JG (JNLE) – Jump if greater</b>	ZF=0 and SF=OF	<b>JA (JNBE) – Jump if above</b>	CF=0 and ZF=0
קפוץ אם קטן ממש	<b>JL (JNGE) – Jump if less</b>	SF != OF	<b>JB (JNAE) – Jump if below</b>	CF=1
קפוץ אם גדול שווה	<b>JGE (JNL) – Jump if greater or equal</b>	SF=OF	<b>JAE (JNB) – Jump if above or equal</b>	CF=0
קפוץ אם קטן שווה	<b>JLE (JNG) – Jump if lower or equal</b>	ZF=1 or SF != OF	<b>JBE – Jump if below or equal</b>	CF=1 or ZF=1
קפוץ אם שווה	<b>JE – Jump if equal</b>	ZF=1	<b>JE – Jump if equal</b>	ZF=1
קפוץ אם שונה	<b>JNE – Jump if not equal</b>	ZF=0	<b>JNE – Jump if not equal</b>	ZF=0



## מה הטעיה בהשוואה ?

כאשר נרצה להשוות בין שני מספרים ניתן לעשות זאת  
באמצעות חישור

compare(a,b):is **a-b < 0?**

השימוש באופקוד **sub** משנה את ערכם של האוגרים

# השוואה CMP

ראינו כי שימוש ב sub עושה שינוי באוגרי עצם איז מה הפתרון ?  
שימוש באופקוד CMP

CMP משנה את הערך של אוגר הדגלים Caino בוצעה הפעולה sub  
**a,b** ללא שינוי של הרגיסטרים עצם

דוגמאות

● ● ● unsigned

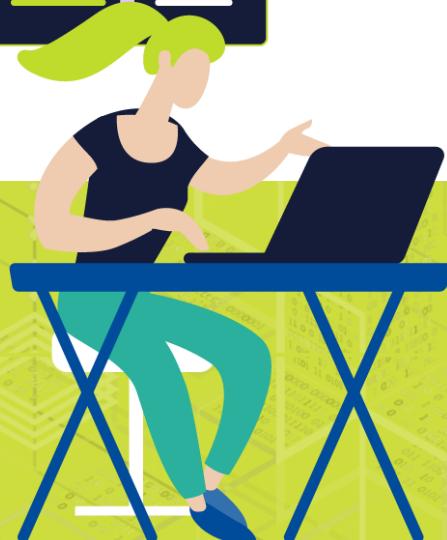
```
cmp ax,bx  
ja mylabel
```

● ● ● signed

```
cmp ax,bx  
jg mylabel
```

# Live Coding

פיתוח בעורך קוד ↶



נדגמים כתיבת השוואה בין שני מספרים



## בדיקה הבנה



```
mov ax,0xffff  
xor ax,0  
dec ax  
jz labelA
```

לא תקוף

תקוף



נכון



לא נכון

```
mov ax,0xffff  
or ax,0  
inc ax  
jz labelB
```

לא תקוף

תקוף



לא נכון



נכון

```
mov ax,0xf  
shr ax,1  
jc labelC
```

לא תקוף

תקוף



לא נכון



נכון

## בדיקה הבנה



```
mov ax,0xffffa  
xor ax,0xB  
jpe labelD
```

לא תקוף

תקוף



נכון



לא נכון

```
mov ax,0xffff0  
mov bx,15  
cmp bx,ax  
jae labelE
```

לא תקוף

תקוף



נכון



לא נכון

```
mov ax,0xffff0  
mov bx,15  
cmp bx,ax  
jge labelF
```

לא תקוף

תקוף



נכון



לא נכון

# שיעור 3

# תנאים ולולאות



סיכום

לולאות

קפיצה מותנת

דגלים  
וקפיצות

עבודה עם  
זיכרון

# LOOP

- שימוש לב שזו פקודת מעבד ולא פונקציה
- פקודת שנועדה לבצע לולאות For
- **loop label**
- המעבד משתמש באוגר CX וرك ב- CX כמנונה
- הוראת loop שקופה לפעולות הבאות:

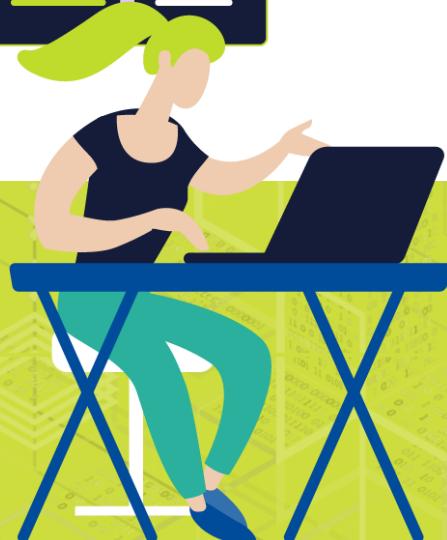
DEC cx

CMP cx, 0

JNE label

# Live Coding

פיתוח בעורך קוד ↶



נדגמים כתיבת השוואה בין שני מספרים



# הדפסה למסך

על מנת להדפס למסך בtabbenו לכמ ספירה פשוטה.

בukoב אחרי השלבים

1. בתיקייה של הקוד שלנו נוסף את הקובץ

magshimim.inc

מה NEO

2. בתחתית הקוד שלנו נרשום

include magshimim.inc

print number

```
mov ax, 17  
call print_num
```

print char

```
mov al, 'C'  
call print_al_chr
```

print string

```
str dw 'hello'  
mov ax, offset str  
call print_ax_str
```

print string with \n

```
PRINTN "BLA BLA"
```

הולכים לתוכנית!

לפני שנתחיל את תרגיל הבית  
פתחו את תרגיל הכתיבה  
מהמצגת ע"י לחיצה על הקישור

## תרגיל כתבה

WHAT DO YOU CALL AN  
ALLIGATOR IN A VEST?

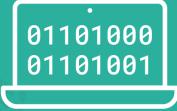


AN INVESTIGATOR.



יאלה  
לעבודה

ארQUITקטורה  
**אסmbלי**



ביצוע Hands-On

\*נמצא בתיקית החניכים



סיקום שיעור



תרגיל הבית



מעבר נוסף  
על המציגת



איך כדי לחזור  
על החומר?

# שאלות? תהיות?

תכתבו עבשו שאלות בציג



ארקיטקטורה  
**אסמבלי**



**מגשיים**  
תכנית הסייבר הלאומית

המרכז לחינוך סייבר  
CYBER EDUCATION CENTER



סיכון

פונקציות  
ופרוצדורות

שמירת מצב  
תכנית במחסנית

פעולות  
המחסנית

מבנה  
המחסנית

# שיעור 3

# סיימנו!!!