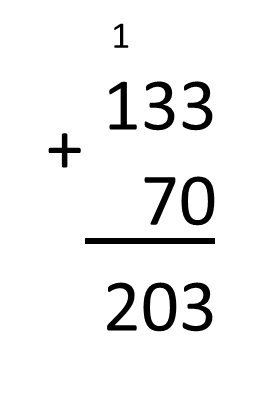
פעולות אריתמטיות לוגיות

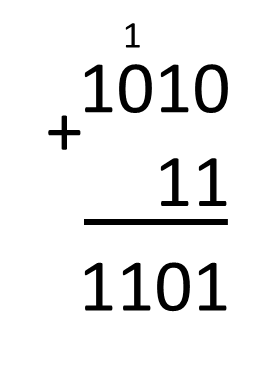
## פעולת חיבור וחיסור

* בסיסים הינם שיטות שונות לייצוג מספרים
* בחיי היום יום -הבסיס הנפוץ ביותר הוא בסיס עשרוני (בסיס 10)
* במחשבים הבסיס הנפוץ ביותר הוא בסיס בינארי (בסיס 2). ולנוחות המתכנתים בסיס 16.

נלמד כיצד לעשות פעולות חשבון בבסיסים.

# פעולות חיבור בסיס 2

בחיבור בבסיס 10, מעבירים נשא (Carry) כשהתוצאה גדולה מ-9

בחיבור בבסיס 2, מעבירים נשא כשהתוצאה גדולה מ-1

## תרגול חיבור בינארי

11002

+

10102

------------

101102

11002

+

10102

------------

101102

1011002

+

110102

--------------

10001102

110012

+

110002

--------------

1100012

1100002

+

1001012

------------

10101012

1001112

+

101002

------------

1110112

1110012

+

1010102

------------

11000112

1110012

+

100112

--------------

10011002

1101102

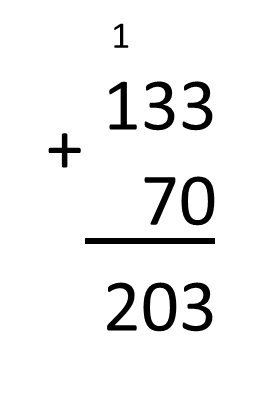
+

1010112

------------

11000012

# חיבור בבסיס 16

בחיבור בבסיס 10, מעבירים נשא (Carry) כשהתוצאה גדולה מ-9

בחיבור בבסיס 16, מעבירים נשא כשהתוצאה גדולה מ-15 (0Fh)

**ABCD**

**+**

**0123**

**-----------**

**ACE0**

(10) (11) (12) (13)

+

0 1 2 3

--------------------------

(10) (12) (14) (16)

A C E [10]

## תרגול חיבור הקסאדצימאלי

2916 [9]

+

B16 [11]

------------

3416 [20-16]

2516 [5]

+

F16 [15]

------------

3416 [20-16]

2616  [6]

+

A16  [10]

--------------

3016

2616 [6]

+

2616 [6]

--------------

4C16  [12⇨C]

1F16 [15]

+

2A16 [10]

------------

3916 [25-16]

3416

+

3416

------------

6816

3A16 [10]

+

1316  [3]

--------------

4D16 [13 ⇨ D]

2216

+

2F16

--------------

5116

1E16 [14]

+

1216  [2]

------------

3016 [16 ⇨ 0]

# פקודת ADD

הפקודה add מחברת את אופרנד המקור (source) עם ערך אופרנד היעד (destination) ושומרת את התוצאה באופרנד היעד. האופרנדים יכולים להיות מסוג רגיסטרים, משתנה או קבוע.

את החישובים מומלץ לבצע בעזרת הרגיסטר **ax**, המעבד מבצע אותם מהר יותר מאשר באמצעות רגיסטרים אחרים.

|  |  |  |
| --- | --- | --- |
| הפקודה | דוגמה | תוצאה |
| add register, register | add ax, bx | ax = ax + bx |
| add register, memory | add ax, [var1] | ax = ax + va1 |
| add register, constant | ad ax, 2 | ax = ax + 2 |
| add memory, register | add [var1], ax | var1 = var1 + ax |
| add memory, constant | add [var1], 2 | var1 = var1 + 2 |

תכנית:

mov ax, 0

mov al, [v1]

add al, [v2]

mov [v1], al

add al, 2

DATASEG

v1 db 16

v2 db 10

מעקב:

|  |  |  |  |
| --- | --- | --- | --- |
| פקודה | רגיסטר al | משתנה v1 | משתנה v2 |
| v1 db 16d |  | 10h |  |
| V2 db 10d |  |  | Ah |
| mov al, [v1] | 10h |  |  |
| add al, [v2] | 1A (10h+Ah) |  |  |
| mov [v1], al |  | 1A (26d) |  |
| add al, 2 | 1Ch (28d) |  |  |

## מה קורה שתוצאת החיבור גדולה יותר מבית?

mov al, 0ffh

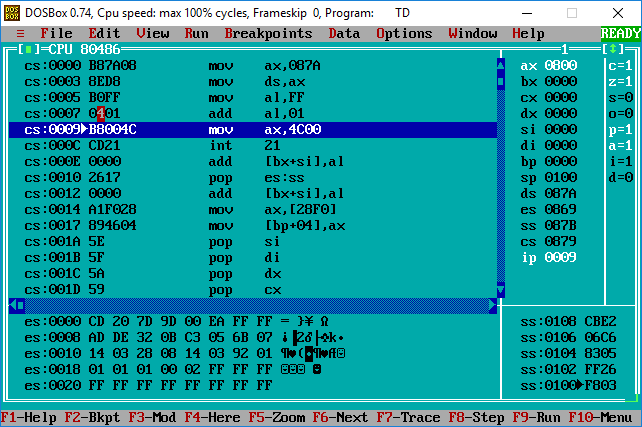
add al, 1

ffh = 255d ⇨ 255+1 = 256d = 100h

מעבר של ספרה מינה – מעבירים נשא

שימו לב , יש לכך סימון. תוצאת החיבור ברגסיטר al היא 00

נדלק דגל הנשא **carry flag** = 1 המסמן שיש לנו ספרה שגלשה מגודל הבית.



לכן יש לתכנן מראש את הגול הרגיסטר או המשתנה הרצוי לנו לקבלת תוצאת החישוב כדי למנוע טעויות.

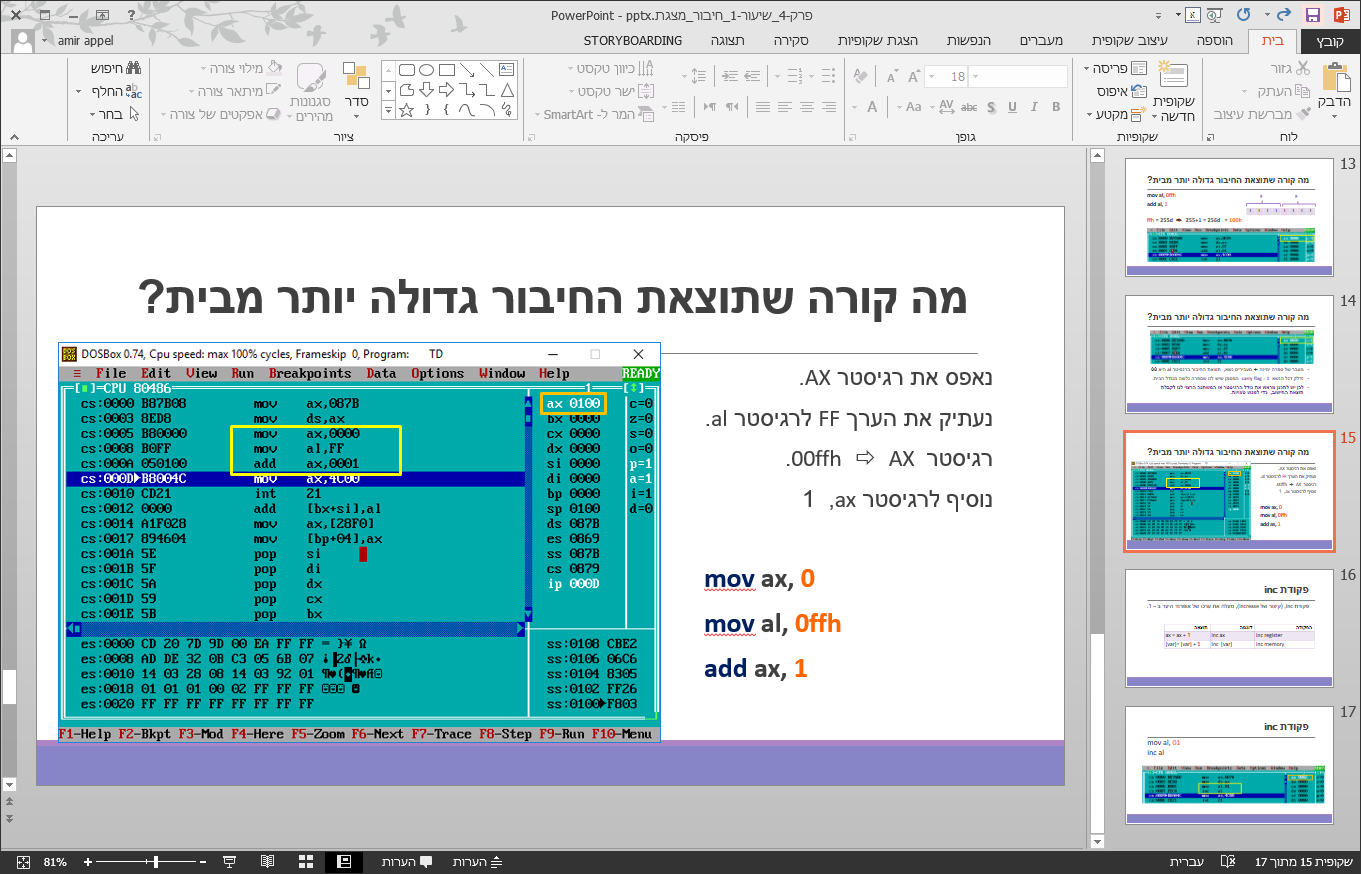
נאפס את רגיסטר AX.

נעתיק את הערך FF לרגיסטר al.

רגיסטר AX ⇨ 00ffh.

נוסיף לרגיסטר ax, 1

**mov ax, 0**

**mov al, 0ffh**

**add ax, 1**

# פקודת inc

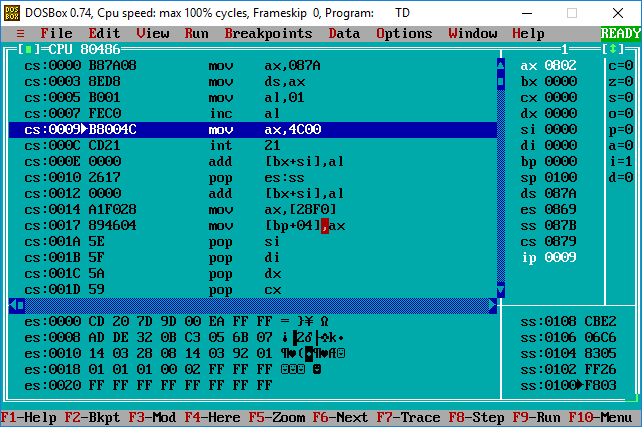
פקודת inc, (קיצור של increase), מעלה את ערכו ש אופרנד היעד ב – 1.

אפשר כמובן לבצע הוראה זו באמצעות פקודת add של 1, אך כוון שפקודה זו נפוצה מאוד הוקדשה לה פקודה נפרדת.

|  |  |  |
| --- | --- | --- |
| **הפקודה** | **דוגמה** | **תוצאה** |
| inc register | inc ax | ax = ax + 1 |
| inc memory | Inc [var] | [var]= [var] + 1 |

mov al, 01

inc al



סיכום

* פעולת add מחברת בין אופרנד היעד ואופרנד המקור ושומרת את התוצאה באופרנד היעד.
* פעולת חיבור ניתן לבצע בין רגיסטרים, משתנה לרגיסטר, משתנה וקבוע, רגיסטר וקבוע, ורגיסטר ומשתנה.
* יש לחשוב על גודל הרגיסטר או המשתנה של אופרנד היעד.
* הפעולה inc מוסיפה לרגיסטר או משתנה 1.

# פעולות אריתמטיות לוגיות

פעולת חיסור

* בסיסים הינם שיטות שונות לייצוג מספרים
* בחיי היום יום -הבסיס הנפוץ ביותר הוא בסיס עשרוני (בסיס 10)
* במחשבים הבסיס הנפוץ ביותר הוא בסיס בינארי (בסיס 2). ולנוחות המתכנתים בסיס 16.

# פעולת חיסור

* אם הספרה העליונה גדולה מהתחתונה- חיסור פשוט
* אם הספרה העליונה שווה לתחתונה- התוצאה אפס
* אם הספרה העליונה קטנה מהתחתונה- "נשא שלילי"

## תרגול חיסור בינארי

אם הספרה העליונה קטנה מהתחתונה **מעבירים מהספרה שבצד ימין 2 (102)**

ובספרה שהעברנו ממנה נשאר 0.

11102

-

00112

----------

10112

11002

-

10102

------------

00102

1101002

-

102

-------------

1100102

1010102

-

101002

--------------

101102

101002

-

11012

------------

1112

110012

-

101102

------------

000112

1010112

+

110102

------------

100012

1110002

-

11112

--------------

1010012

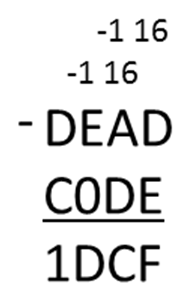
1001012

-

100112

------------

100102



# חיסור בבסיס 16

* אם הספרה העליונה גדולה מהתחתונה- חיסור פשוט
* אם הספרה העליונה שווה לתחתונה- התוצאה אפס
* אם הספרה העליונה קטנה מהתחתונה- "נשא שלילי"

## תרגול חיסור בינארי

אם הספרה העליונה קטנה מהתחתונה **מעבירים מהספרה שבצד ימין 16 (1016)**

ובספרה שהעברנו ממנה מחסירים 1.

9316 [16+3]

-

3D16 [13]

------------

5616 [6]

F34A16 [16+10]

-

098F16 [15]

---------------

E9BB16  [11]

A23516

-

003416

---------------

A20116

13016 [16]

-

9316 [3]

--------------

9D16  [13⇨D]

E716 [E ⇨ 14]

-

6016 [6]

------------

8716

C316 [16+3]

-

1916  [9]

------------

AA16 [10 ⇨A]

5416 [16+4]

-

D16  [13]

--------------

4716 [13 ⇨ D]

4716

-

1216

--------------

3516

CF16 [12 , 15]

-

5616

------------

7916

פקודת SUB

הפקודה sub (קיצור של subtract) מחסרת את אופרנד המקור (source) מערך אופרנד היעד (destination) ושומרת את התוצאה באופרנד היעד. האופרנדים יכולים להיות מסוג רגיסטרים, משתנה או קבוע.

צורת הכתיבה זהה לצורת הכתיבה של add.

|  |  |  |
| --- | --- | --- |
| הפקודה | דוגמה | תוצאה |
| sub register, register | sub ax, bx | ax = ax - bx |
| sub register, memory | sub ax, [var1] | ax = ax - va1 |
| sub register, constant | sub ax, 2 | ax = ax - 2 |
| sub memory, register | sub [var1], ax | var1 = var1 - ax |
| sub memory, constant | sub [var1], 2 | var1 = var1 - 2 |

תכנית:

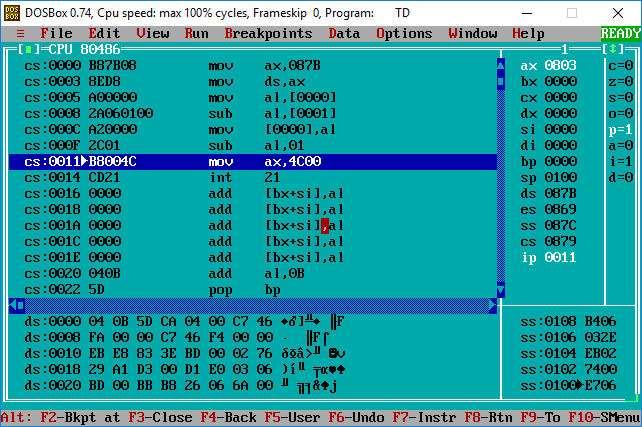
התכנית מחסירה ממשתנה v1, את v2 ושומרת את התוצאה למשתנה v1.

אחר-כך מחסירה מרגיסטר al את הערך 1.

* mov al, [v1]
* sub al, [v2]
* mov [v1], al
* sub al, 1

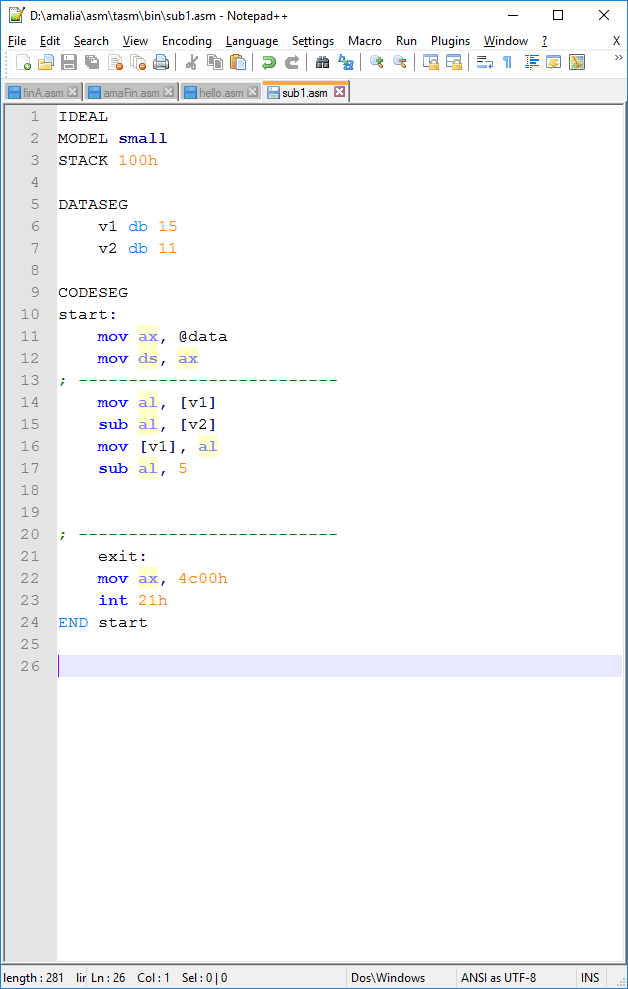
מעקב:

|  |  |  |  |
| --- | --- | --- | --- |
| פקודה | רגיסטר al | משתנה v1 | משתנה v2 |
| v1 db 15d |  | 0fh |  |
| v2 db 11d |  |  | 0bh |
| mov al, [v1] | 0fh (15d) |  |  |
| sub al, [v2] | 4 |  |  |
| mov [v1], al |  | 4 |  |
| sub al, 1 | 3 |  |  |



בפעולת חיסור יכולה להתקבל תוצאה שלילית. טרם למדנו על הצגת של מספרים שלילים במחשב על כן נדון על כך בהמשך.

כרגע נציין כי שי דגל הנדלק ומציין שהמספר הוא שלילי. דגל הסימן – Signed Flag



תוצאת החישוב היא: 1-

al ⇦ 15d

al ⇦ 15d – 11d = 4d

[vl] ⇦ 4

al ⇦ 4d – 5d = -1

**ייצוג המספר 1- במחשב בשיטת**

**המשלים ל - 2.**

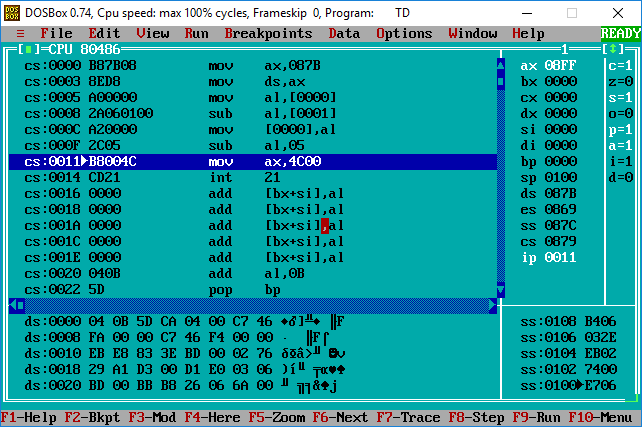
1 = 0001 ⇨ ייצוג בינארי

1110 ⇨ הופכי

1111 ⇨ הוספה של 1

1111b ⇨ 0FFh

מספר שלילי - Signed



# פקודת DEC

פקודת dec (קיצור של decrease) מחסירה מערכו של אופרנד היעד 1.

|  |  |  |
| --- | --- | --- |
| **הפקודה** | **דוגמה** | **תוצאה** |
| dec register | dec ax | ax = ax - 1 |
| dec memory | dec [var] | [var]= [var] - 1 |

# סיכום

* פעולת sub מחסרת מאופרנד המקור את הערך שאופרנד היעד ושומרת את התוצאה באופרנד היעד.
* פעולת חיסור ניתן לבצע בין רגיסטרים, משתנה לרגיסטר, משתנה וקבוע, רגיסטר וקבוע, ורגיסטר ומשתנה.
* תוצאת החיסור יכולה להיות מפסר שלילי, נדלק דגל המסמל שזהו מספר שלילי Signed Flag.
* הפעולה dec מחסירה מרגיסטר או משתנה את הערך 1.

תרגול

## bg4

הזזת השחקן על פני לוח המשחק

שמרו את התכנית שכתבתם בשבוע שעבר בשם חדש והמשיכו לעבוד על הקובץ החדש.

בשיעור זה נתרגל את התזוזה של השחקן שלנו.

לצורך תזוזת הדמות יצרנו משתנים אשר ישמרו את הערכים של מיקום הסמן על ציר x, מיקום הסמן על ציר y וצבע הדמות.

במקטע הנתונים – DATASEG

; character coordinates

x\_cord db 7 ; column

y\_cord db 6 ; row

color dw 0Eh ; color

למדנו שכדי שנזיז את הדמות (ליצור אנימציה) "מוחקים" את הדמות ע"י ציור הדמות בצבע שחור במיקום בו היא נמצאת.

שינוי מיקום הסמן על ציר ה – x ועל צירה ה – y

ציור השחקן בצבע במיקום החדש.

כדי להזיז את הדמות ימינה יש להגדיל את המיקום על ציר ה – x ⇦

כדי להזיז את הדמות שמאלה יש להקטין את המיקום על ציר ה – x ⇨

כדי להזיז את הדמות למטה יש להגדיל את המיקום על ציר ה – y ⇩

כדי להזיז את הדמות למעלה יש להקטין את המיקום על ציר ה – y ⇧

הפסיקות הממקמות את הסמן ומציירות את השחקן.

;============ On key press, move character ===========

mov [color], 0 ; black color

"מחיקת הדמות" ציור הדמות בשחור במיקום הסמן .

; draw smiley - ascii 2 at cursor position

mov ah, 9

mov al, 2 ; aL = character to display

mov bx, [color] ; bh = Background bl = Foreground

mov cx, 1 ; cx = number of times to write character

int 10h

mov [y\_cord], 19

השמת ערכים חדשים לציור הדמות החדשה במיקום ובצבע חדש.

mov [x\_cord] , 5

mov [color], 4 ; red color

; print chararcter

; set cursore location

mov dh, [y\_cord] ; row

הזזת הסמן למיקום החדש וציור הדמות במיקום החדש

mov dl, [x\_cord] ; column

mov bh, 0 ; page number

mov ah, 2

int 10h

; draw smiley - ascii 2 at cursor position

mov ah, 9

mov al, 2 ; aL = character to display

mov bx, [color] ; bh = Background bl = Foreground

mov cx, 1 ; cx = number of times to write character

int 10h

; waits for character

המתנה למקש

mov ah, 0h

int 16h

;===============================================

## עבודה עם ה –Turbo Debugger

הריצו את התכנית ב TD

לפני תחילת ההרצה נבדוק את ערכי המשתנים:

Alt + v

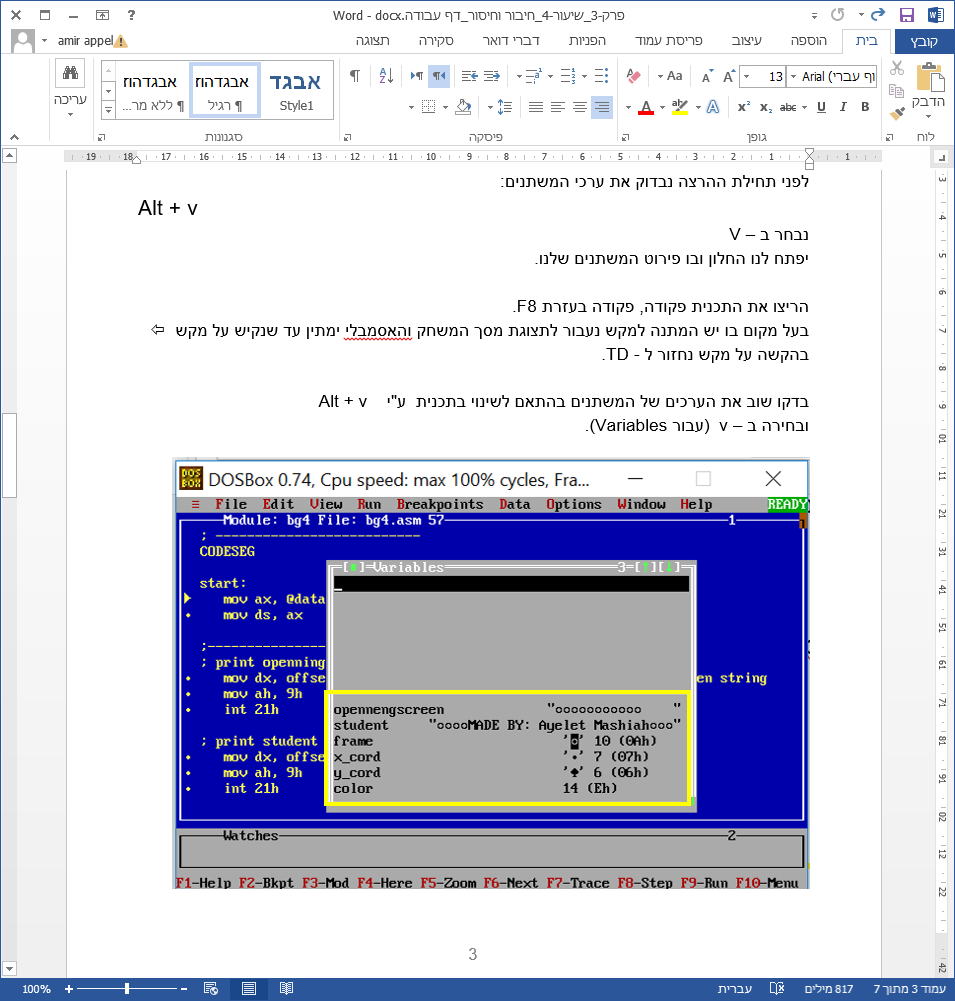
נבחר ב – V

הריצו את התכנית פקודה, פקודה בעזרת F8.

בעל מקום בו יש המתנה למקש נעבור לתצוגת מסך המשחק והאסמבלי ימתין עד שנקיש על מקש ⇦ בהקשה על מקש נחזור ל - TD.

בדקו שוב את הערכים של המשתנים בהתאם לשינוי בתכנית ע"י Alt + v

ובחירה ב – v (עבור Variables).

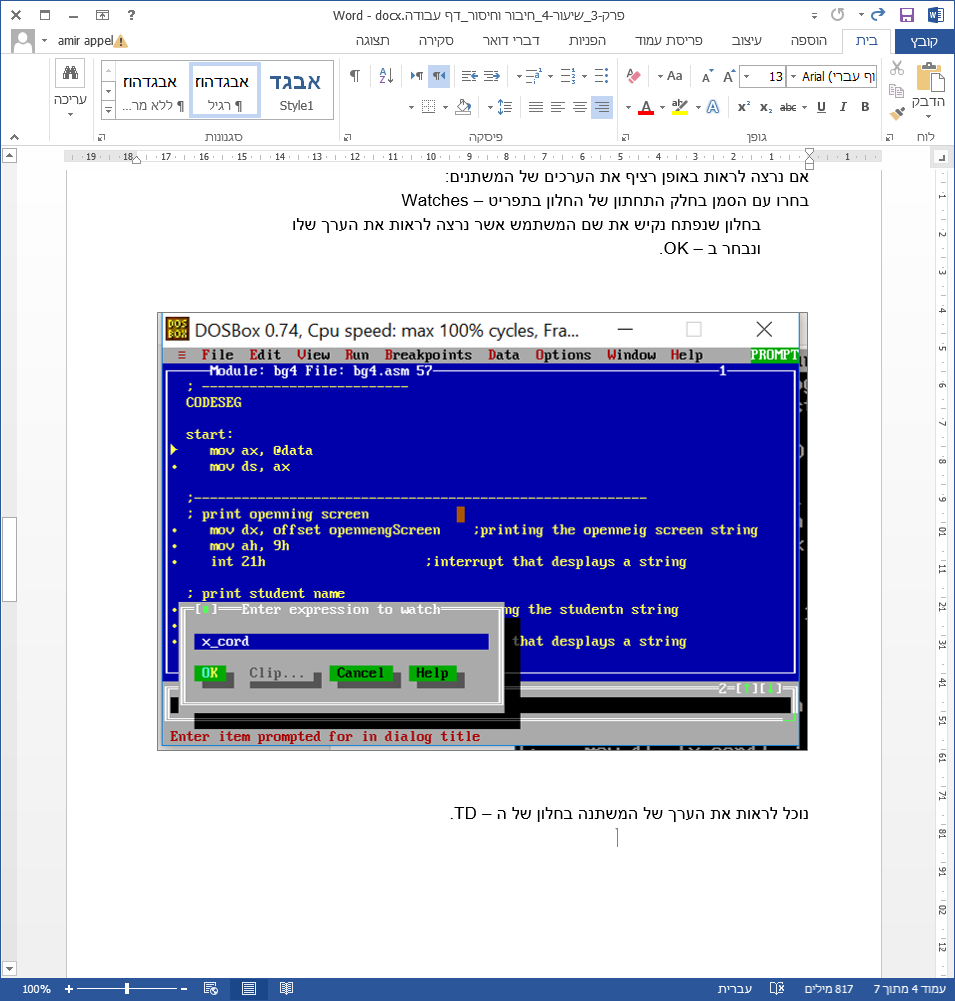


אם נרצה לראות באופן רציף את הערכים של המשתנים:

בחרו עם הסמן בחלק התחתון של החלון בתפריט – Watches

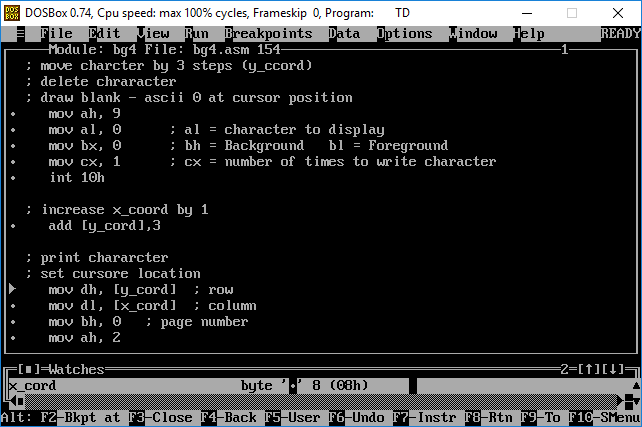
בחלון שנפתח נקיש את שם המשתמש אשר נרצה לראות את הערך שלו

ונבחר ב – OK.



נוכל לראות את הערך של המשתנה בחלון של ה – TD.

שם המשתנה גודלו ערכו בבסיס 10 (וערכו בבסיס הקסדצימאלי)



|  |
| --- |
| בשיעור הקודם הזזנו את השחקן מספר פעמים כל ידי פעולות השמה.  הפעם נזיז את הדמות שלנו על ידי **חישוב** הקואורדינטות של המיקום הבא ולא על ידי קביעתן.  נשנה את פקודות ההשמה לפקודות אריתמטיות לוגיות.  מחקו את פעולות ההשמה למשתנים [x\_cord] ן – [y\_cord] ובמקומם נרשום פקודות חיבור וחיסור.  אם נרצה לזוז ימינה נגדיל את x\_cord על ידי פעולת חיבור ואם נרצה לזוז שמאלה נקטין אות על ידי פעולת חיסור.  אם נרצה לזוז למטה נגדיל את y\_cord על ידי פעולת חיבור  ואם נרצה לזוז למעלה נקטין אות על ידי פעולת חיסור. |

## פעולת חיבור

1. הזיזו את השחקן שלכם ימינה בצעד אחד

inc [x\_cord]

1. הזיזו את השחקן שלכם ימינה ב - 3 צעדים

add [y\_cord], 3

1. הזיזו את השחקן שלכם למטה בצעד אחד

inc [y\_cord]

1. הזיזו את השחקן שלכם למטה ב - 5 צעדים

add [y\_cord], 5

## פעולת חיסור

1. הזיזו את השחקן שלכם שמאלה בצעד אחד

dec [x\_cord]

1. הזיזו את השחקן שלכם שמאלה ב - 3 צעדים

sub [y\_cord], 3

1. הזיזו את השחקן שלכם למעלה בצעד אחד

dec [y\_cord]

1. הזיזו את השחקן שלכם למעלה ב - 5 צעדים

sub [y\_cord], 5

1. האם תוכלו להזיז את השחקן שלכם באלכסון?

; move charcter by -2 steps (y\_ccord) +2 (x\_coord)

; delete chraracter

; draw blank - ascii 0 at cursor position

mov [color], 0

mov ah, 9

mov al, 0 ; al = character to display

mov bx, [color] ; bh = Background bl = Foreground

mov cx, 1 ; cx = number of times to write character

int 10h

; increase x\_coord by 1

; decrease y\_coord by 2

mov [color], 2

sub [y\_cord], 2

add [x\_cord], 2

; print chararcter

; set cursore location

mov dh, [y\_cord] ; row

mov dl, [x\_cord] ; column

mov bh, 0 ; page number

mov ah, 2

int 10h

; draw smiley - ascii 2 at cursor position

mov ah, 9

mov al, 2 ; aL = character to display

mov bx, [color] ; bh = Background bl = Foreground

mov cx, 1 ; cx = number of times to write character

int 10h

; waits for character

mov ah, 0h

int 16h