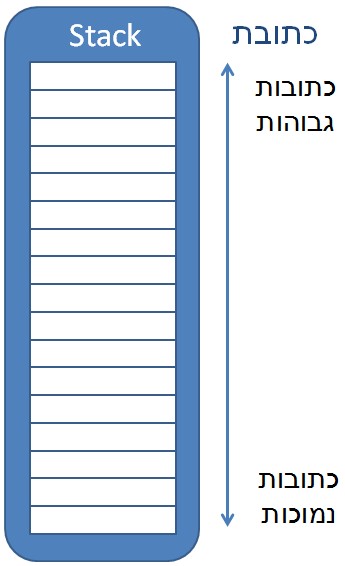
# פעולות ומחסנית

## מחסנית

push & pop



המחסנית (stack), היא סגמנט בזיכרון, המשמשת לאחסון של משתנים לזמן קצר.

המחסנית מנוהלת בדרך הדומה מאוד לדרך שבה מנוהלת מחסנית תחמושת של רובה אוטומטי. הכדורים נטענים במחסנית תחמושת בזה אחר זה, ודוחפים זה את זה דרך הפתח הנמצא בראש המחסנית. פריקת הכדורים מן המחסנית נעשית גם היא דרך הפתח, אך סדר הוצאת הכדורים הפוך מסדר הכנסת הכדורים: זה שנכנס-אחרון-יוצא-ראשון

LIFO – Last in First Out.

כל דחיפה של כדור למחסנית התחמושת מזיזה את כל הכדורים שכבר נמצאים בה, ומפנה

מקום לכדור החדש. פתח מחסנית התחמושת (ראש המחסנית) נשאר, כמובן, קבוע במקומו.

בדומה למחסנית התחמושת, גם המחסנית שבזיכרון המחשב מנוהלת בדרך המאפשרת

להכניס ולהוציא נתונים בשיטת נכנס-אחרון-יוצא-ראשון.

הכנסת נתון למחסנית שבזיכרון אינה גורמת להזזת הנתונים הקודמים. בִמקום זה, ראש המחסנית זז, כלומר המקום שאליו נכנס הנתון הבא משתנה, כדי שהנתון החדש שייכנס לא ייכתב על הנתון הקודם ויהרוס אותו.

הכנסת נתון למחסנית נקראת דחיפה - Pushing,

הוצאת נתון מן המחסנית נקראת שליפה - Popping

מעקב אחר מיקומו של ראש המחסנית, המשתנה עם כל דחיפה ושליפה, נעשה על-ידי האוגר SP שנקרא מצביע המחסנית, ותפקידו להצביע בכל רגע נתון על הכתובת שאליה נדחף הנתון האחרון. (SP מכיל כתובת יחסית לאוגר .SS)

שימו לב 🎔: כאשר משתמשים במחסנית, יש להימנע משליפת נתונים מכתובות שלא הוכנסו

אליהן נתונים. כדי למנוע מצב של שליפת נתונים אקראיים, כיוון שלכתובת הזו במחסנית לא הוכנס כל נתון במהלך העבודה עם המחסנית.

המצב הזה מכונה גלישת מחסנית ( Stack overflow ), שכן ראש המחסנית גולש בשליפה הזו מעבר למיקומו הראשוני, לאזור בזיכרון שאינו שייך למחסנית.

בהגדרת המחסנית עלינו להקפיד להקצות מספיק מקום לסגמנט המחסנית כדי שיהיה אפשר לנהל את כל הנתונים הדרושים.

לסיכום, המעקב אחר מיקומו של ראש המחסנית נעשה באוגר sp, כל פעולת דחיפה מקטינה באופן אוטומטי את ערכו של האוגר SP , וכל פעולת שליפה מגדילה את הערך הזה.

מחסנית

* המחסנית (stack), היא סגמנט בזיכרון, המשמשת לאחסון של משתנים לזמן קצר.
* המחסנית מנוהלת בדרך הדומה מאוד לדרך שבה מנוהלת מחסנית תחמושת של רובה אוטומטי.
* הכדורים נטענים במחסנית תחמושת בזה אחר זה, ודוחפים זה את זה דרך הפתח הנמצא בראש המחסנית.
* פריקת הכדורים מן המחסנית נעשית גם היא דרך הפתח, אך סדר הוצאת הכדורים הפוך מסדר הכנסת הכדורים: זה שנכנס-אחרון-יוצא-ראשון
* LIFO – Last in First Out.

Stack

בדומה למחסנית התחמושת, גם המחסנית שבזיכרון המחשב מנוהלת בדרך המאפשרת להכניס ולהוציא נתונים בשיטת   
LIFO – Last in First Out נכנס-אחרון-יוצא-ראשון.

הכנסת נתון למחסנית שבזיכרון אינה גורמת להזזת הנתונים הקודמים. בִמקום זה, ראש המחסנית זז, כלומר המקום שאליו נכנס הנתון הבא משתנה, כדי שהנתון החדש שייכנס לא ייכתב על הנתון הקודם ויהרוס אותו.

הכנסת נתון למחסנית נקראת דחיפה - Pushing,

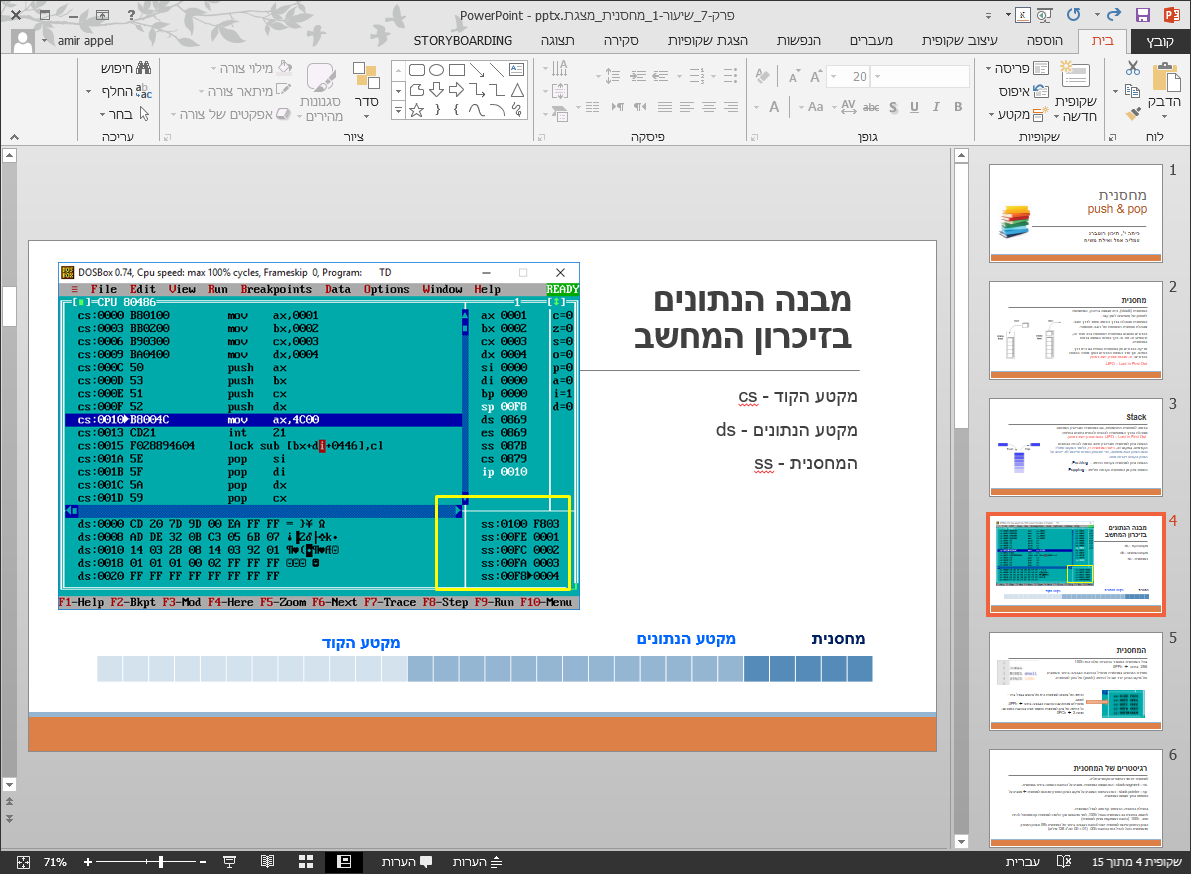
הוצאת נתון מן המחסנית נקראת שליפה – Popping

מבנה הנתונים בזיכרון המחשב

מקטע הקוד - cs

מקטע הנתונים - ds

המחסנית - ss



כמו כל סגמנט בזיכרון, המחסנית היא אזור בזיכרון שמתחיל בכתובת כלשהי ותופס גודל מוגדר של זיכרון.

גודל האזור בזיכרון שמוקצה למחסנית נקבע על-ידי המתכנת בתחילת התוכנית. הקצאת המקום נעשית בדרך הבאה:

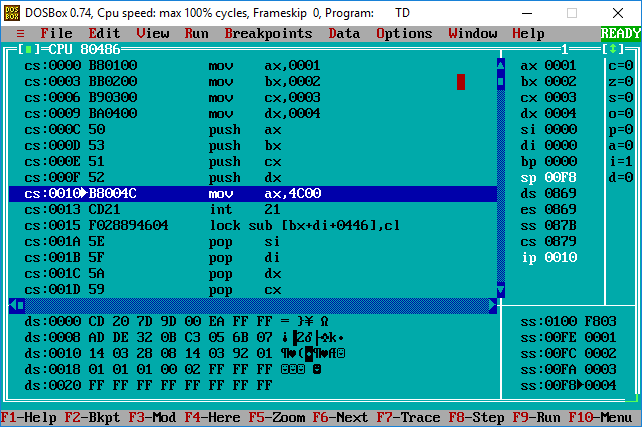
STACK number of bytes

לדוגמה, כדי להקצות מחסנית בגודל 256 בתים, נגדיר - כפי שמוגדר גם בקובץ base.asm:

STACK 100h

המחסנית

* גודל המחסנית המוגדר בתכניות שלנו הוא 100h ⇦ 256 בתים ⇦ 0FFh
* שמירת הנתונים במחסנית מתחיל בכתובת הגבוהה ביותר והמצביע של מיקם הנתון יורד עם כל דחיפה (push) של נתון למחסנית



דחיפה של נתונים למחסנית היא של נתונים בגודל מילה – word.

מתחילים מהתא עם הכתובת הגבוהה ביותר ⇦ 0FFh.

כל דחיפה של נתון למחסנית תשמור אותו בכתובת האחרונה פחות 2 ⇦ 0FCh

רגיסטרים של המחסנית

למחסנית יש שני רגיסטרים שקשורים אליה.

stack segment – ss - הוא סגמנט המחסנית. מצביע על הכתובת הנמוכה ביותר במחסנית.

stack pointer – sp - הוא הרגיסטר המצביע על מיקום הנתון האחרון שהוכנס למחסנית ⇦ מצביע על האופסט בתוך סגמנט המחסנית.

בתחילת התוכנית, הרגיסטר sp שווה לגודל המחסנית.

לדוגמה בתוכנית בה המחסנית בגודל 100h, לפני שהכנסנו ערך כלשהו למחסנית sp מאותחל להיות שווה 100h (כתובת הממוקמת מחוץ למחסנית)

* הנתון הראשון שיכנס למחסנית יכנס לכתובת הגבוהה ביותר של המחסנית 0ffh והנתון האחרון שהמחסנית תוכל להכיל הוא הכתובת 00h. (01 ו- 00 סה"כ 128 מילים)

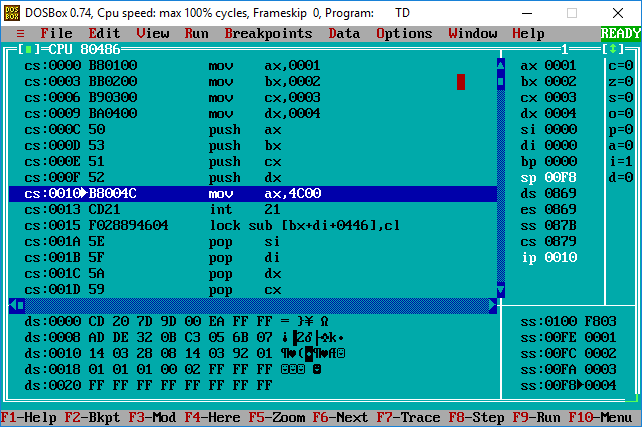
mov ax, 01

mov bx, 02

* mov cx, 03
* mov dx, 04
* push ax
* push bx
* push cx
* push dx

עם כל דחיפה של נתון למחסנית הביטים הגבוהים יכנסו לכתובת הגבוהה והביטים הנמוכים לכתובת הנמוכה.

ה – SP יצביע על הכתובת הנמוכה של המילה במחסנית

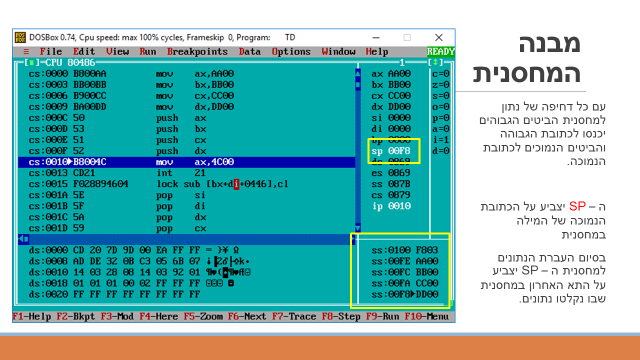


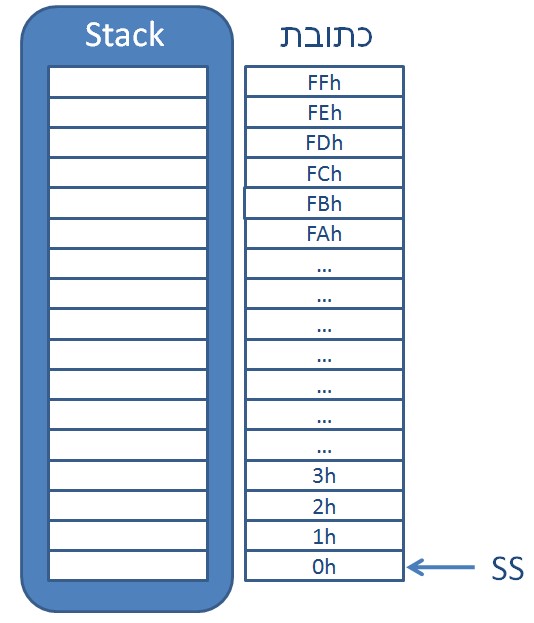
|  |  |
| --- | --- |
| **נתון** | **כתובת במחסנית** |
| 00  01 | FF  FE |
| 00  02 | FD  FC |
| 00  03 | FB  FA |
| 00  04 | F9  F8 |

עם כל דחיפה של נתון למחסנית הביטים הגבוהים יכנסו לכתובת הגבוהה והביטים הנמוכים לכתובת הנמוכה.

ה – SP יצביע על הכתובת הנמוכה של המילה במחסנית

בסיום העברת הנתונים למחסנית ה – SP יצביע על התא האחרון במחסנית שבו נקלטו נתונים.



למחסנית יש שני רגיסטרים שקשורים אליה.

stack segment – ss - הוא סגמנט המחסנית. מצביע על הכתובת הנמוכה ביותר במחסנית.

stack pointer – sp - הוא מצביע המחסנית. מצביע על האופסט בתוך סגמנט המחסנית.

בתחילת התוכנית, הרגיסטר sp שווה לגודל המחסנית.

בדוגמה שנשתמש בה, מחסנית בגודל 100h, לפני שהכנסנו ערך כלשהו למחסנית sp מאותחל להיות שווה h100.

במחסנית בגודל 100h, כלומר 256 בתים, מרחב הכתובות אינו מגיע עד 100h אלא הוא בין 0 ל - offh. (הערך ההתחלתי של מצביע על כתובת שהיא בדיוק בית אחד מעל קצה המחסנית).

המחסנית מנוהלת בשיטת LIFO – Last In First Out,

כלומר הערך שנכנס אחרון הוא הראשון לצאת מהמחסנית.

לפני הכנסה של נתונים למחסנית, ערכו של sp יורד ולאחר הוצאה של נתונים מהמחסנית ערכו של sp עולה.

פקודת PUSH

פקודת push גורמת להכנסה של ערך למחסנית.

הפקודה נכתבת כך:

push operand

מה יבוצע?

- ערכו של sp יירד בשתיים: 2-sp=sp.

- ערכו של האופרנד יועתק למחסנית, לכתובת ss:sp.

ערכו של sp תמיד יורד בשתיים עם פקודת push, כלומר הוא מצביע על כתובת שרחוקה שני בתים מהכתובת האחרונה עליה הצביע.

המשמעות היא שאפשר לדחוף למחסנית רק משתנים בגודל של שני בתים – word. כל ניסיון לבצע push לכמות אחרת של בתים – יוביל לשגיאה.

push ax

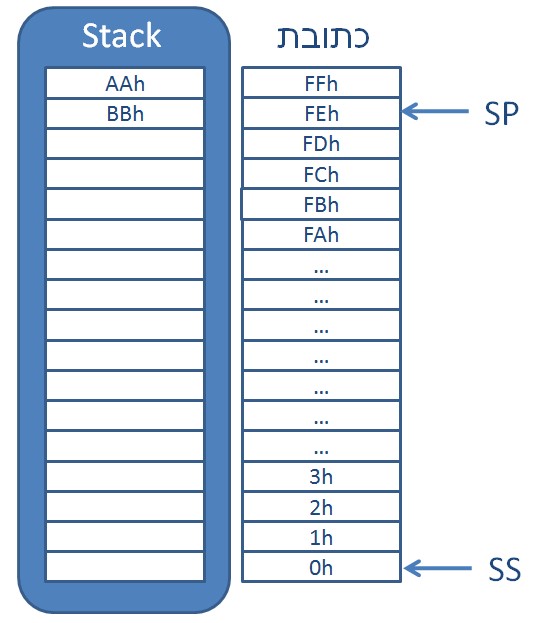
push 10

push [var]

הפקודה הראשונה תדחוף למחסנית את ax.

הפקודה השניה תדחוף למחסנית את הערך 10 (בצורתו כ-word, לא כ־byte).

הפקודה השלישית תדחוף למחסנית את תוכן המשתנה [var] – בתנאי שהוא מגודל word.



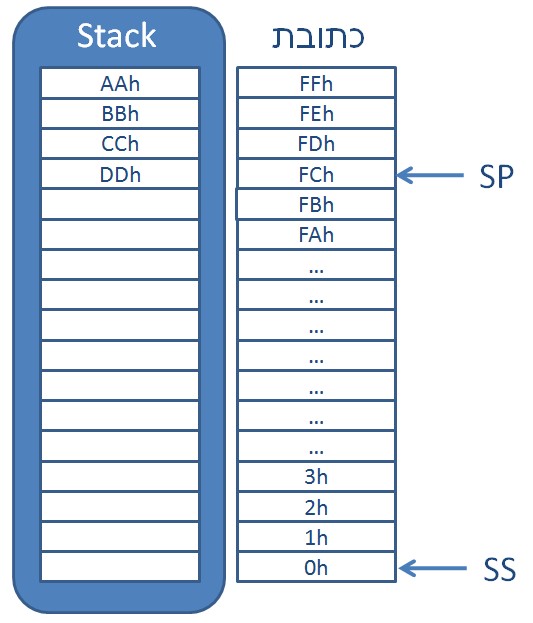
מה יהיה מצב המחסנית לאחר ביצוע הפקודות הבאות?

mov ax, 0AABBh

push ax

הביטים הגבוהים נכנסים לכתובת הגבוהה של המחסנית.

ורגיסטר sp מצביע על הכתובת הנמוכה של המילה בה נשמר הנתון האחרון.



push 0CCDDh

הדחיפה הבאה הפחיתה את הכתובת שרגיסטר sp מצביע עליה ב -2

FE – 2 = FC

פקודת pop

פקודת pop היא הפקודה ההפוכה ל־push.

פקודה זו גורמת להוצאה של מילה (שני בתים) מהמחסנית והעתקה שלה לאופרנד היעד:

pop operand

פקודת pop מושכת מילה מראש המחסנית ומעתיקה ללאופרנד.

ערכו של sp יעלה ב־2.

דוגמאות לשימוש בפקודת pop:

pop ax

pop [var]

הפקודה הראשונה תעתיק לתוך ax את המילה שבראש המחסנית.

הפקודה השנייה תעתיק לתוך המשתנה את [var], את המילה שבראש המחסנית.

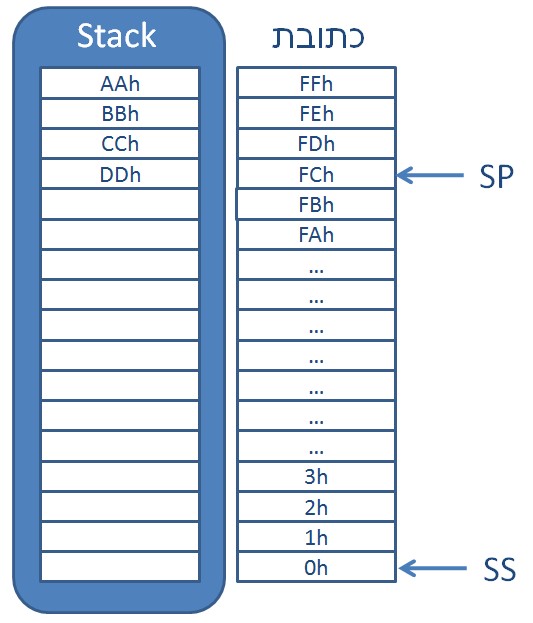
(כדי למנוע שגיאת קומפילציה ,var צריך להיות מטיפוס מילה).

* האופרנד חייב להיות בגודל מילה

pop al ;illegal

* אסור לאופרנד להיות קבוע

pop 5h ;illegal



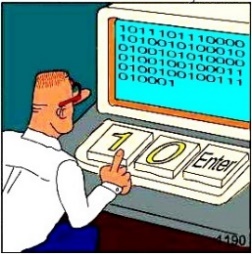
פקודת POP – המשך

נתונה מחסנית. מה יקרה לאחר הפקודה הבאה?

pop bx

bx יקבל את הערך - 0CCDDh,

ערכו של sp יעלה בשתיים ויהיה שווה 0FEh

תרגול:

## הפסיקות בתכנית שלנו

|  |  |
| --- | --- |
| פקודה להדפסת מחרוזת  (שם משתנה המחרוזת msg) | mov dx, offset msg  mov ah, 9h  int 21h |
| פקודה לקליטת תו מהמקלדתושמירתו לרגיסטר al | mov ah, 1h  int 21h |
| הדפסת תו למסך הדפסת תו למסך בדוגמה התו המודפס הוא x | mov dl, 'x' ; print x  mov ah, 2  int 21h |

## תכנית המעבירה ערך ממשתנה לרגיסטר ללא שימוש בפקודת mov

IDEAL

MODEL small

STACK 100h

DATASEG

; --------------------------

num dw 123h

; --------------------------

CODESEG

start:

mov ax, @data

mov ds, ax

; --------------------------

push [num]

pop ax

; --------------------------

exit:

mov ax, 4c00h

int 21h

END start

INT 16h / AH = 00h - get keystroke from keyboard (no echo).

return:

AH = BIOS scan code.

AL = ASCII character.

(if a keystroke is present, it is removed from the keyboard buffer).

1. חשבו האם הפקודות הבאות חוקיות מדוע?

|  |  |
| --- | --- |
| **האם תקין ומדוע** | **פקודה** |
| לא תקין, ניתן להעביר רגיסטר של בית | push al |
| תקין | push ax |
| תקין | push 10h |
| תקין, כל עוד המשתנה בגודל מילה | push [var] |
| לא, תקין רגיסטר בגודל של בית | pop al |
| תקין | pop ax |
| לא תקין, לא ניתן להעביר נתון לקבוע | pop 10h |
| תקין | pop [var] |

1. קבעו ערך לרגיסטר ax, העתיקו את ax לתוך bx בעזרת המחסנית ללא שימוש בפקודת mov.

mov ax, 1234h

push ax

pop bx

1. צרו שני משתנים והעבירו אותם למחסנית . שמרו לתא הבא של המחסנית את סכום המשתנים.

ATASEG

n1 dw 5

n2 dw 10

CODESEG

start:

mov ax, @data

mov ds, ax

push [n1]

push [n2]

mov ax, [n1]

add ax, [n2]

push ax

**נכתוב תחילה את התוכנית עבור תו אחד.**

* ראשית נדפיס הודעה למסך לקליטת תו:

תזכורת: הפסיקה להדפסת מחרוזת

mov dx, offset msg

mov ah, 9h

int 21h

* לא נשכח להגדיר את msg בסגמנט הנתונים
* הריצו את התכנית ובדקו שיש הדפסת הודעה למסך המבקשת מהמשתמש להקליד תו.

הפסיקה למעבר שורה

mov dl, 0ah

mov ah, 2h

int 21h

* כעת נקרא את התו שהמשתמש ידפיס.

תזכורת: הפסיקה לקליטת תו מהמקלדת ושמירתו לרגיסטר al:

mov ah, 1h

int 21h

* שמרו (push) את התו למחסנית.

*מאחר והמחסנית שומרת ערכים של מילים.   
נעביר את כל רגיסטר ax למחסנית ולפני כן נאפס את רגיסטר ah.*

* שלפו (pop) את התו מהמחסנית והדפיסו אותו.  
  תזכורת: הפסיקה המדפיסה תו השמור ברגיסטר dl

mov dl, 'x' ; print x

mov ah, 2

int 21h

*מאחר והמחסנית שומרת ערכים של מילים.   
נכניס ל dl תו על ידי שליפת התו השמור במחסנית* ***לרגיסטר dx****.*

הריצו את התכנית האם קיבלתם הדפסה למסך של התו אותו הקלדתם?

הריצו את התכנית ב – td ובדקו את השמירה והמשיכה של התו מהמחסנית.

**נרחיב את התכנית לקליטת והדפסת 5 תווים**

* קליטת 5 תווים והכנסתם למחסנית:

בצעו את קליטת הנתונים בעזרת לולאה:   
מונה הלולאה הוא cx, קבעו לו ערך

בתחילת הלולאה נקבע תווית וקבעו לה שם - loopName:  
בסיום הלולאה נקבע את הפקודה loop ושם התווית - loop loopName

* הדפסת 5 התווים

בלולאה נוספת משכו את התווים מהמחסנית והדפיסו אותם למסך.

שימו לב 🎔, אם משכתם את הערך מהמחסנית לרגיסטר dx, הערך הרצוי להדפסה נשמר לכם ברגיסטר dl.

**מהו סדר הדפסת התווים ולמה?**

DATASEG

msg db "Enter 5 numbers", 10, 13, '$'

CODESEG

start:

mov ax, @data

mov ds, ax

mov dx, offset msg

mov ah, 9h

int 21h

mov cx, 5

getKey:

; get charecter from keyboard

mov ah, 1h

int 21h

mov ah, 0

push ax

loop getKey

mov cx, 5

printKey:

pop dx

; print charecter is dl

mov ah, 2

int 21h

loop printKey

exit:

1. תרגיל חקר:  
   הכניסו את הערך 1234h לתוך ax.   
   בצעו push של axלתוך המחסנית.

**הריצו את התכנית ב td , בתצוגת cpu.**

בדקו איך השתנה רגיסטר sp לאחר פקודת push?

הסתכלו על הזיכרון שבמחסנית ומצאו את הערך שדחפתם.

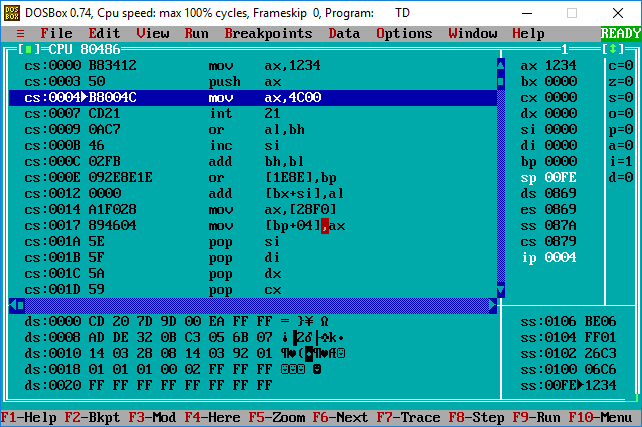
בצעו pop למחסנית לתוך ax.

איך השתנה רגיסטר sp?

הסתכלו על הזיכרון שבמחסנית – האם הערך 1234h נמחק?

העבירו לרגיסטר bx את הערך 5678h,

בצעו push של bx למחסנית מהי הכתובת של הנתון במחסנית?

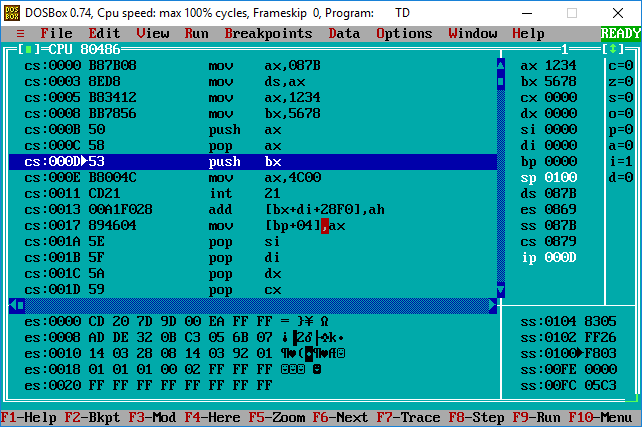
 mov ax, 1234h

push ax

pop ax

mov bx, 5678h

push bx



בכתבות של 0FEh נקלט הערך 5678h