חומרי עזר
אסמבלי 8086
פרויקטים גראפיים

כתיבה ועריכה:
רחל פרלמן
דנה אבן חיים
עמליה אפל
חגית כהן
סוניה שמאי

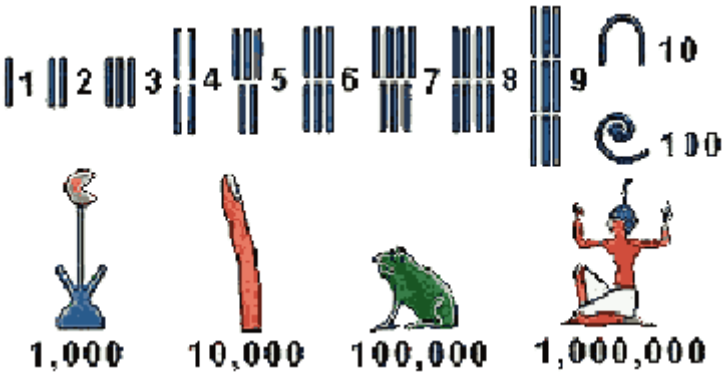
מרכז מורים ארצי במקצוע מדעי המחשב. הפרויקט מבוצע עפ"י מכרז 9/7/2013.
הפרויקט מבוצע עבור האגף לתכנון ופיתוח תכניות לימודים, המזכירות הפדגוגית, משרד החינוך

תוכן עניינים

מבוא לבסיסי ספירה

כתב החרטומים המצרי, הכתב ההירוגליפי, הנו כתב ציורי, שבו כל אות מייצגת עצם כלשהו במציאות. לכן גם המספרים בכתב המצרי מיוצגים ע"י ציורים של עצמים שונים.

מערכת המספרים המצרית מבוססת על השיטה העשרונית (כלומר בסיס הספירה הוא 10), ולכן הסימנים הבסיסיים שמוגדרים בה הם עבור מספרים שהם חזקות של 10. בסיס הספירה 10 נחשב לבסיס ה"טבעי" והוא היה נפוץ במערכות מספרים קדומות, עקב השימוש שנעשה בעשר האצבעות כדי לספור עצמים שונים. מערכת המספרים המצרית מוצגת באיור הבא:



ולכן ניתן להציג את הערכים הבאים כך:

לפי הטבלה לעיל, נסו לנחש מהם המספרים:

תרגמו למצרית את המספרים הבאים:

א. 36 ב. 372

משחקי אוגרים (רגיסטרים) - עמליה אפל

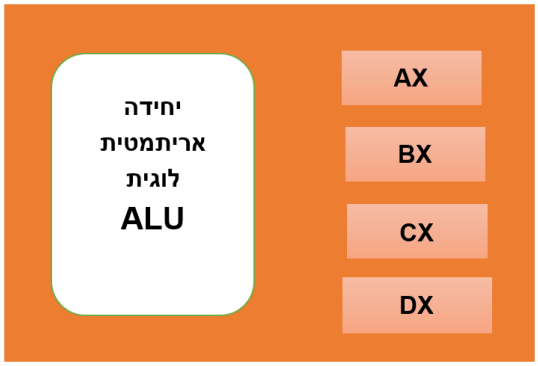
מבנה המעבד

יחידה אריתמטית לוגית - ALU

יחידה אריתמטית לוגית (ALU - Arithmetic Logic Unit), תפקידה לבצע את הפעולות המוגדרות בהוראה, כגון העתקת נתונים ממקום למקום, פעולות אריתמטיות ופעולות לוגיות.
המעבד אינו מסוגל לעשות פעולות רבות בו זמנית. הוא מבצע פעולה אחת בזמן נתון. חוזקו הוא במהירות החישוב שלו.

אוגרים

באוגרים מאוחסנים הנתונים בהם משתמשת היחידה האריתמטית לוגית.
גדלו של כל רגיסטר הוא 16 (2 * 8)
רגיסטר מורכב מ - 2 בתים – כלומר מילה (word)
הרגיסטרים יושבים במעבד ולכן ליחידה האריתמטית לוגית יש גישה מהירה אליהם.
(בניגוד לתאי הזיכרון של המחשב).



1. תכנית ראשונה ופקודות השמה

השמה של ערך לאוגר נעשית בפקודת mov.
פתחו את קובץ base.exe ב – Notepad++.
הקלידו את הפקודות הבאות ובדקו את מצב הרגיסטרים.
את הפקודה כתבו מתחת להערה של

```
; Your code here
mov ax, 5
mov bx, 10
```

שמרו את התכנית בשם אחר (t1.asm).

להרצת התכנית הפעילות את ה – **DosBox**.
זהו סימולטור – כלי המדמה סביבת עבודה ישנה.
ב *Dosbox* נמיר את התוכנית שלנו מאסמבלי לשפת מכונה – מפקודות באסמבלי לאפסים ואחדים. ההמרה מתבצעת בשני שלבים

הקלידו את הפקודות הבאות:

tasm t1

<שם הקובץ> *tasm* --> נוצר קובץ *obj*.<שם קובץ>

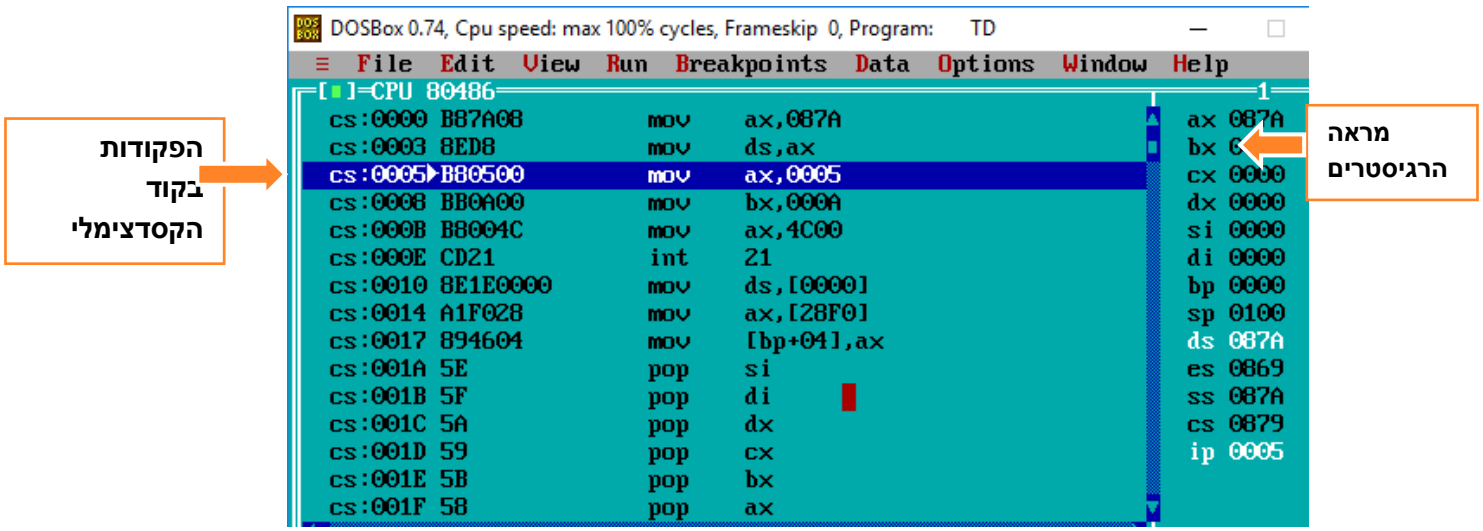
tlink t1

<שם הקובץ> *tlink*--> נוצר קובץ *exe*.<שם קובץ> - זהו קובץ ההרצה שלנו

td t1

Td- Turbo debugger
דיבאגר הוא כלי להרצת תוכניות ובדיקתן.
הוא ה"עיניים" שלנו למצב הרגיסטרים והזיכרון לאחר כל פקודה

הקשה על **F5** תגדיל את מסך ה - **td**.
להרצת הפקודות ב – **Turbo Debuger** הקישו **F7**.



מהו הערך הכתוב ברגיסטר AX לאחר ביצוע הפקודה?
מהו הערך הכתוב ברגיסטר BX לאחר ביצוע הפקודה?
האם תוכלו להסיר מדוע יש ברגיסטר bx את הערך A000?

כדי לצאת מה – td ולחזור לחלון הפקודות של ה – Dos Box יש להקיש על **Alt + X**

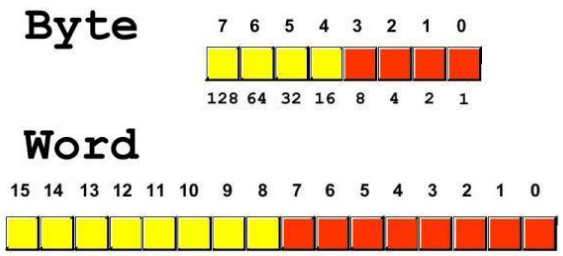
2. השמה של נתונים

ניתן להעביר נתונים לרגיסטרים בשלושה בסיסים: בינארי, עשרוני והקסדצימאלי.
על מנת שהאסמבלי ידע באיזה בסיס מודבר יש לסמן את הבסיס.
d - מסמל בסיס עשרוני (decimal) {לא חייבים לרשום זו ברירת המחדל – default}.
b – מסמל בסיס 2 (binary)
h - מסמל בסיס 16 (hexadecimal) {כאשר המספר מתחיל באות יש לרשום 0 (אפס) לפני האות}
לדוגמה: mov ax, 0ah (יעביר לרגיסטר ax את הערך 10 בבסיס עשרוני)

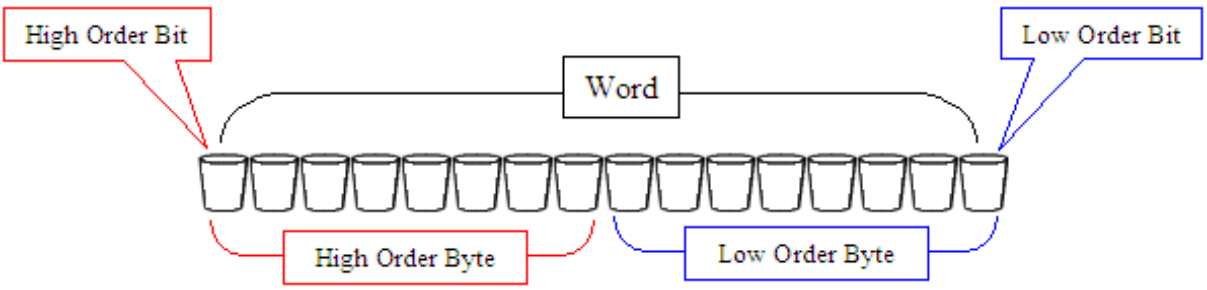
כתבו תכנית שבה תעבירו לרגיסטרים **ax, bx** ו - **cx** את הערך 13_{10} בבסיסים שונים.
בדקו את התוצאה ב – td האם בכל הרגיסטרים רשומים אותם ערכים?
אם לא תקנו את התוכנית, הריצו שוב ובדקו את התוצאה ברגיסטרים.

3. בית (**Byte**) ומילה (**Word**)

רגיסטר היא יחידת זיכרון היכולה לשמור עד 2^{16} bit = 65,536 ערכים.
רגיסטר הוא יחידת זיכרון של מילה – הכוללת שני בתים.
$2^8 * 2^8 = 256 * 256 = 65,536$



ניתן לחלק כל רגיסטר לשני בתים: גבוה ונמוך
בצד ימין של המספר נמצאים הערכים הנמוכים שלו ולכן הוא הוא נקרה בית נמוך – Low
בצד שמאל של המספר נמצאים הערכים הגבוהים ולכן הוא נקרא בית גבוה - High



כל רגיסטר מורכב מ 2 תאי זיכרון. על מנת לחסוך במשאבים ניתן גם להשתמש בחצי רגיסטר.

רגיסטר	חצי רגיסטר שמאלי "High	חצי רגיסטר "ימני" Low
Ax	Ah	Al
Bx	Bh	Bl
Cx	Ch	Cl
Dx	Dh	Dl

כאשר מעבירים מידע מרגיסטר לרגיסטר חייבים לבדוק שרגיסטר המקור ורגיסטר היעד באותו הגודל.
לדוגמה:

פקודות חוקיות מאחר וגדלי הרגיסטרים שווים.

```
mov al, 11b
mov al, ah
mov ah, bl
mov bl, ch
```

לא ניתן להעביר נתונים מרגיסטר בגודל מילה לרגיסטר בגודל בית
לדוגמה:

פקודות לא חוקיות:
גדלי רגיסטרים לא שווים
גודל הקבוע שרוצים להעביר לרגיסטר גדול מידי
לא ניתן להעביר ערך של רגיסטר לקבוע.

```
mov ch, dx
mov dx, al
mov al, 100000111b
mov 5, al
```

בדקו את הפקודות הבאות וכתבו האם חוקיות והאם לא?
הריצו ובדקו.

פקודה אופרנד מקור, אופרנד יעד, פקודה	אם הפקודה חוקית כתבו מה ערכו של אופרנד היעד?	אם הפקודה אינה חוקית כתבו מודע
mov bx, 100	bx = 100	
mov al, bx		al רגיסטר בגודל בית bx רגיסטר בגודל מילה (אופרנד מקור ואופרנד יעד לא שווים)
mov ds, 1234h		
mov al, 25		
mov ah, al		
mov bl, cx		
mov 7, ax		
mov al, 257		
mov ax, 257		
mov, dx, ax		

4. פעולת חיבור בין שני רגיסטרים.

הפקודה **add** מחברת את אופרנד המקור (source) עם ערך אופרנד היעד (destination) ושומרת את התוצאה באופרנד היעד.
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, constant	add ax, 2	ax = ax + 2

כתבו תכנית מבצעת את הפקודות הבאות:

- אפסו את רגיסטר ax עם הפקודה mov ax, 0
- השמו לרגיסטר bl את הערך 250
- השמו לרגיסטר bh את הערך 2
- חברו את שני הערכים והשמו את התוצאה ברגיסטר al

הריצו ובדקו.

- שנו את התכנית השמו לרגיסטר bh את הערך 10.
- חברו את שני המספרים ושמרו לרגיסטר al
- הריצו את התכנית ובדקו מה קרה לרגיסטר al?

האם קיבלתם תשובה נכונה?
כאשר נחבר שני מסרים שערכם גדול מ 255 יש להשתמש ברגיסטר בגודל מילה.
תקנו את התכנית וחברו 250 + 10 והשמו את התוצאה לרגיסטר ax.

5. חיבור **add** ופקודת **inc**

בצעו את הפקודות הבאות:

- השמו לרגיסטר ax את הערך fh1
- פקודת *inc* מגדילה את ערך הרגיסטר ב – 1
- הגדילו את הערך של ax ב -1
- הוסיפו ל - ax את הערך 10
- העבירו את התוצאה לרגיסטר bx
- הוסיפו לרגיסטר bx את הערך 10-
- העבירו לרגיסטר cx את הערך שברגיסטר ax
- הוסיפו לרגיסטר cx את הערך - 10h-
- הגדילו את cx ב – 1
- הוסיפו ל –cx 21
- הוסיפו ל – ax את הערך 1ah-

מה תוצאת החישוב בשלושת הרגיסטרים?

משחקי אוגרים (רגיסטרים) - תשובות

מבנה המעבד

יחידה אריתמטית לוגית - ALU

יחידה אריתמטית לוגית (ALU - Arithmetic Logic Unit), תפקידה לבצע את הפעולות המוגדרות בהוראה, כגון העתקת נתונים ממקום למקום, פעולות אריתמטיות ופעולות לוגיות.
המעבד אינו מסוגל לעשות פעולות רבות בו זמנית. הוא מבצע פעולה אחת בזמן נתון. חוזקו הוא במהירות החישוב שלו.

אוגרים.

באוגרים מאוחסנים הנתונים בהם משתמשת היחידה האריתמטית לוגית.
גדלו של כל רגיסטר הוא 16 bit.
רגיסטר מורכב מ - 2 בתים – שהם מילה (word)
הרגיסטרים יושבים במעבד ולכן ליחידה האריתמטית לוגית יש גישה מהירה אליהם.
(בניגוד לתאי הזיכרון של המחשב).

1. תכנית ראשונה ופקודות השמה

השמה של ערך לאוגר נעשית בפקודת mov
פתחו את קובץ base.exe ב – Notepad++.
הקלידו את הפקודות הבאות ובדקו את מצב הרגיסטרים.
את הפקודה כתבו מתחת להערה של

```
; Your code here
mov ax, 5
mov bx, 10
```

שמרו את התכנית בשם אחר (t1.asm).

להרצת התכנית הפעילות את ה – **DosBox**.
זהו סימולטור – כלי המדמה סביבת עבודה ישנה.
ב *Dosbox* נמיר את התוכנית שלנו מאסמבלי לשפת מכונה – מפקודות באסמבלי לאפסים ואחדים. ההמרה מתבצעת בשני שלבים

הקלידו את הפקודות הבאות:

tasm t1

<שם הקובץ> *tasm* --> נוצר קובץ *obj*.<שם קובץ>

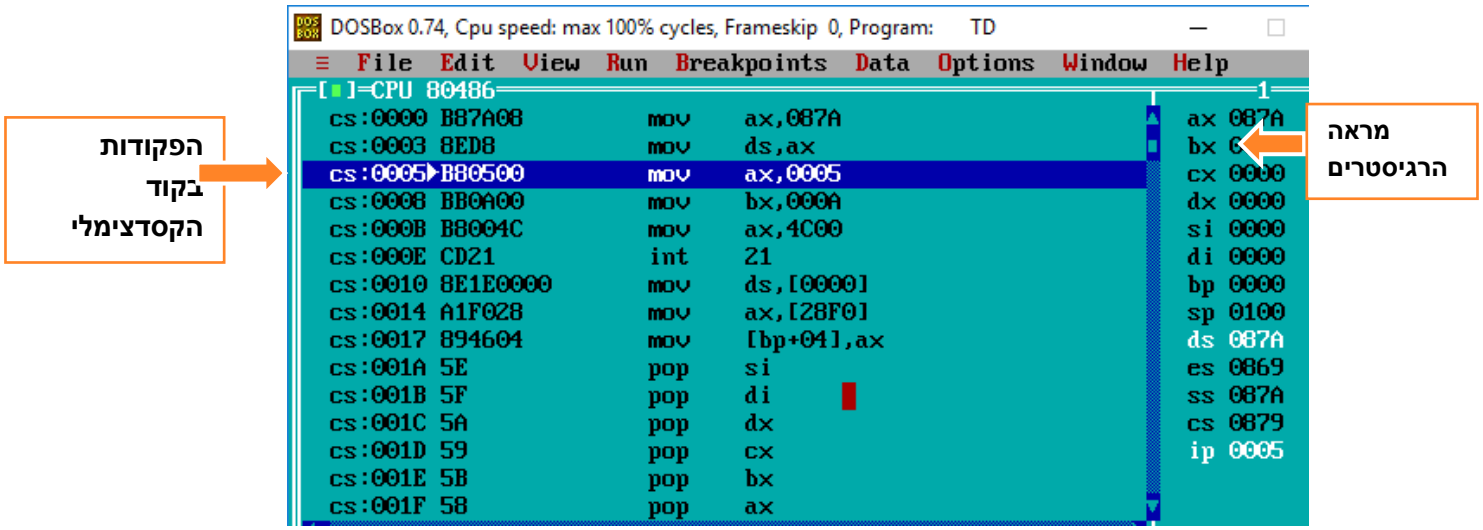
tlink t1

<שם הקובץ> *tlink*--> נוצר קובץ *exe*.<שם קובץ> - זהו קובץ ההרצה שלנו

td t1

Td- Turbo debugger- דיבאגר הוא כלי להרצת תוכניות ובדיקתן.
הוא ה"עיניים" שלנו למצב הרגיסטרים והזיכרון לאחר כל פקודה

הקשה על **F5** תגדיל את מסך ה - **td**.
להרצת הפקודות ב – **Turbo Debuger** הקישו **F7**.



מהו הערך הכתוב ברגיסטר AX לאחר ביצוע הפקודה? **0005**
מהו הערך הכתוב ברגיסטר BX לאחר ביצוע הפקודה? **000A**
האם תוכלו להסיר מדוע יש ברגיסטר bx את הערך 000 ?A **(10d = 0ah)**

כדי לצאת מה – td ולחזור לחלון הפקודות של ה – Dos Box יש להקיש על **Alt + X**

2. השמה של נתונים

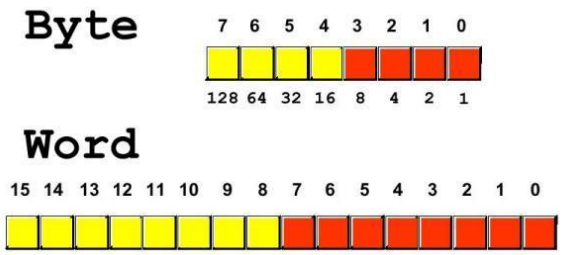
ניתן להעביר נתונים לרגיסטרים בשלושה בסיסים: בינארי, עשרוני והקסדצימאלי.
על מנת שהאסמבלי ידע באיזה בסיס מודבר יש להסמן את הבסיס.
d - מסמל בסיס עשרוני (decimal) {לא חייבים לרשום זו ברירת המחדל – default}.
b – מסמל בסיס 2 (binary)
h - מסמל בסיס 16 (hexadecimal) {כאשר המספר מתחיל באות יש לרשום 0 (אפס) לפני האות}
לדוגמה: mov ax, 0ah (יעביר לרגיסטר ax את הערך 10 בבסיס עשרוני)

כתבו תכנית שבה תעבירו לרגיסטרים **ax, bx** ו - **cx** את הערך 13_{10} בבסיסים שונים.
בדקו את התוצאה ב – td האם בכל הרגיסטרים רשומים אותם ערכים?
אם לא תקנו את התוכנית, הריצו שוב ובדקו את התוצאה ברגיסטרים.

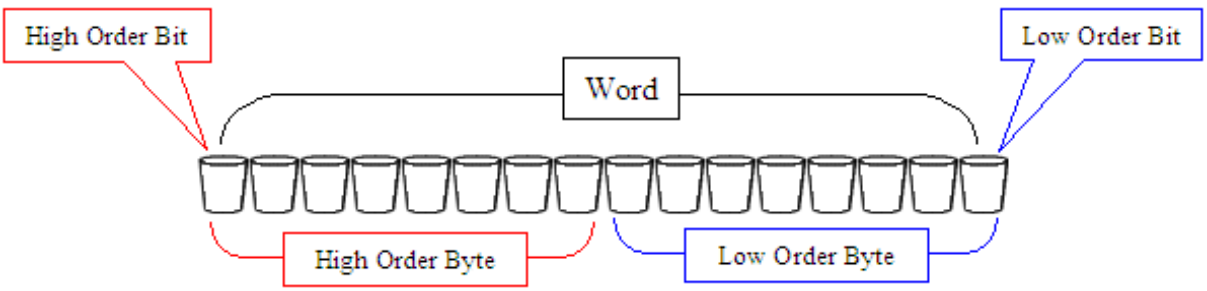
```
mov ax, 13d
mov bx, 0dh
mov cx, 1101b
```

3. בית (**Byte**) ומילה (**Word**)

רגיסטר היא יחידת זיכרון היכולה לשמור עד 2^{16} bit = 65,536 ערכים.
רגיסטר הוא יחידת זיכרון של מילה – הכוללת שני בתים.



ניתן לחל כל רגיסטר לשני בתים. גבוה ונמוך .
בצד ימין של המספר נמצאים הערכים הנמוכים שלו ולכן הוא הוא נקרה בית נמוך – Low
בצד שמאל של המספר נמצאים הערכים הגבוהים ולכן הוא נקרא בית גבוה - High



כל רגיסטר מורכב מ 2 תאי זיכרון. על מנת לחסוך במשאבים ניתן גם להשתמש בחצי רגיסטר.

חצי רגיסטר "ימני" Low	חצי רגיסטר שמאלי" High	רגיסטר
Al	Ah	Ax
Bl	Bh	Bx
Cl	Ch	Cx
Dl	Dh	Dx

כאשר מעבירים מידע מרגיסטר לרגיסטר חייבים לבדוק שרגיסטר המקור ורגיסטר היעד באותו הגודל.

לדוגמה:

פקודות חוקיות מאחר וגדלי הרגיסטרים שווים.

```
mov al, 11b
mov al, ah
mov ah, bl
mov bl, ch
```

לא ניתן להעביר נתונים מרגיסטר בגודל מילה לרגיסטר בדול בית

לדוגמה:

פקודות לא חוקיות :

גדלי רגיסטרים לא שווים
גודל הקבוע שרוצים להעביר לרגיסטר גדול מידי
לא ניתן להעביר ערך של רגיסטר לקבוע.

```
mov ch, dx
mov dx, al
mov al, 100000111b
mov 5, al
```

בדקו את הפקודות הבאות וכתבו האם חוקיות והאם לא?

הריצו ובדקו.

אם הפקודה אינה חוקית כתבו מודע	אם הפקודה חוקית כתבו מה ערכו של אופרנד היעד?	פקודה אופרנד מקור, אופרנד יעד, פקודה
	bx = 100 (64h)	mov bx, 100
al רגיסטר בגודל בית bx רגיסטר בגודל מילה (אופרנד מקור ואופרנד יעד לא שווים)		mov al, bx
ערך גדול מ 255 ולכן גדול מידי לרגיסטר dl (רגיסטר בגודל בית)		mov ds, 1234h
	ax = 25 (19h)	mov al, 25
	ah = al	mov ah, al
לא ניתן להעביר מרגיסטר בגודל מילה (16 ביט) לרגיסטר בגודל בית (8 ביט).		mov bl, cx
לא ניתן להעביר לקבוע		mov 7, ax

mov al, 257		ערך גדול מ 255 ולכן גדול מידי לרגיסטר al (רגיסטר בגודל בית)
mov ax, 257	al = 257 (101h)	
mov, dx, ax	dx = ax	

4. פעולת חיבור בין שני רגיסטרים.

הפקודה **add** מחברת את אופרנד המקור (source) עם ערך אופרנד היעד (destination) ושומרת את התוצאה באופרנד היעד.
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, constant	add ax, 2	ax = ax + 2

כתבו תכנית מבצעת את הפקודות הבאות:

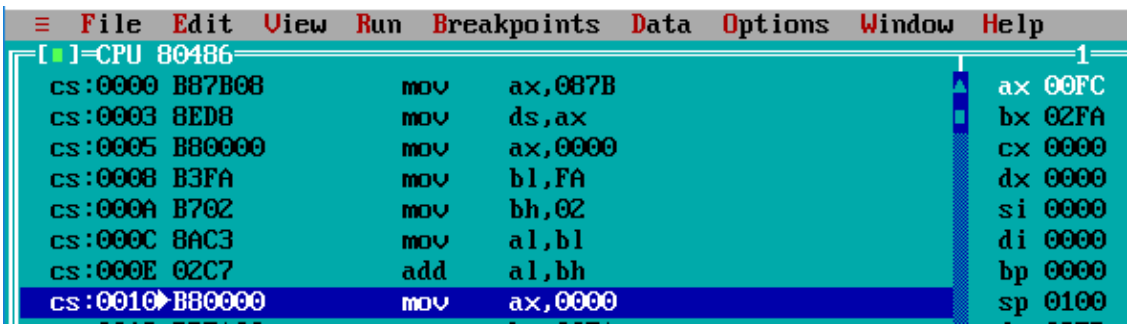
- אפסו את רגיסטר ax
- השמו לרגיסטר bl את הערך 250
- השמו לרגיסטר bh את הערך 2
- חברו את שני הערכים והשמו את התוצאה ברגיסטר al

הריצו ובדקו.

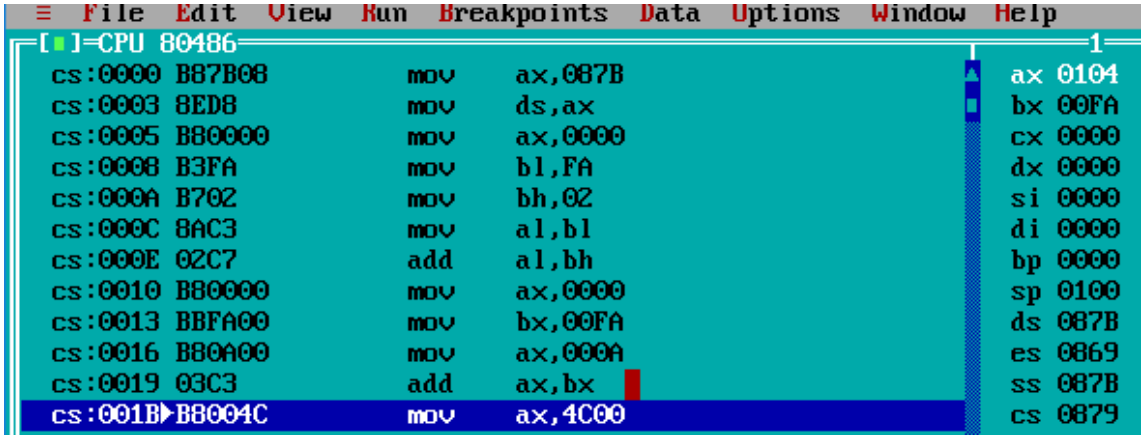
- שנו את התכנית השמו לרגיסטר bh את הערך 10.
- חברו את שני המספרים ושמרו לרגיסטר al
- הריצו את התכנית ובדקו מה קרה לרגיסטר al?

האם קיבלתם תשובה נכונה?
כאשר נחבר שני מסרים שערכם גדול מ 255 יש להשתמש ברגיסטר בגודל מילה.
תקנו את התכנית וחברו 250 + 10 והשמו את התוצאה לרגיסטר ax.

```
mov ax, 0
mov bl, 250
mov bh, 2
mov al, bl
add al, bh
```



```
mov ax, 0
mov bx, 250
mov ax, 10
add ax, bx
```



5. חיבור **add** ופקודת **inc**

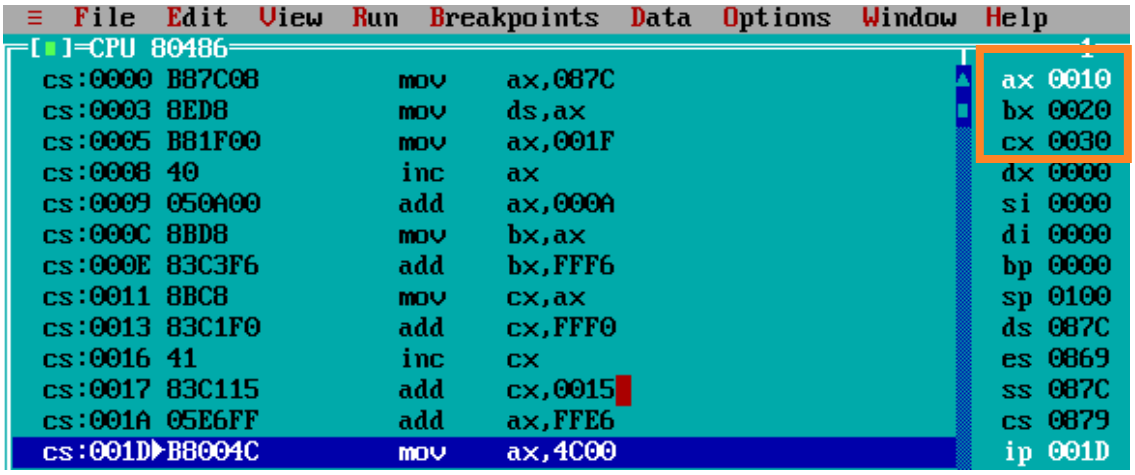
בצעו את הפקודות הבאות:

- השמו לרגיסטר ax את הערך fh1
- פקודת *inc* מגדילה את ערך הרגיסטר ב – 1
- הגדילו את הערך של ax ב -1
- הוסיפו ל - ax את הערך 10
- העבירו את התוצאה לרגיסטר bx
- הוסיפו לרגיסטר bx את הערך 10-
- העבירו לרגיסטר cx את הערך שברגיסטר ax
- הוסיפו לרגיסטר cx את הערך - 10h-
- הגדילו את cx ב – 1
- הוסיפו ל –cx 21
- הוסיפו ל – ax את הערך 1ah-

מה תוצאת החישוב בשלושת הרגיסטרים?

```
mov ax, 1fh
inc ax
add ax, 10
mov bx, ax
add bx, -10
mov cx, ax
add cx, -10h
inc cx
add cx, 21
add ax, -1Ah
```

פעולות



פעולות אריתמטיות – מהם אוגרים? – סוניה שמאי

קבוצת אוגרי הנתונים

	8 סיביות	8 סיביות
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

16 סיביות

קבוצת אוגרי הנתונים כוללת 4 אוגרים בני 16 סיביות כל אחד, אך ניתן להתייחס לכל אוגר כאל 2 אוגרים בני 8 סיביות כל אחד.

שמות החלקים מאופיינים באות H או באות L. H - High, L - Low. כלומר AH, BH וכו' הוא האוגר התופס את החלק העליון של האוגר AX, BX וכו' ואילו AL תופס את החלק התחתון של האוגר AX. האוגרים הללו הם לשימוש כללי ומתפקדים במרבית הפעולות המתבצעות בתוכנית.

- אוגר **AX** - מכונה צובר ומשתמשים בו לצורך סיכום מספרים.
- אוגר **BX** - מכונה אוגר בסיס ומשתמשים בו לצורך ייצוג כתובת.
- אוגר **CX** - מכונה אוגר מונה ומשתמשים בו לצורך מניה.
- אוגר **DX** - מכונה אוגר נתונים ומשתמשים בו לצורך שמירה זמנית של נתונים.

קבוצת אוגרי ההצבעה ואוגרי האינדקס

כל האוגרים מהקבוצה הזו הם בני 16 סיביות ואלו שמותם.

IP - מכיל את כתובת ההוראה הבאה לביצוע.

SP - מכיל את כתובת הנתון שנמצא בראש המחסנית.

BP
SI
DI

נועדו לשימוש זיכרון כתובות

הפקודה MOV

הפקודה גורמת להעתקה (ולא להעברה) של נתון ממקום למקום.

1. העתקת ערך ישירות לאוגר – זוהי הצבת נתון באוגר.
2. העתקת נתון מאוגר לאוגר.
3. העתקת נתון מאוגר לתא זכרון.
4. העתקת נתון מתא זכרון לאוגר.

<u>העתקת ערך ישירות לאוגר – הצבת נתון באוגר</u>

- גודל הנתון חייב להיות קטן או שווה לגודל האוגר אליו מכניסים נתון.
- אם הנתון קטן מגודל האוגר מוצמדים אפסים משמאל למספר (כך לא משתנה ערכו)
- כל ספרה הקסאדצימלית תופסת מקום של 4 סיביות.
- אם הנתון הוא מס' הקסאדצימלי המתחיל באות, יש להקדימו באפס. הוספה זו אינה משנה את המספר, אך מסמנת ליע"מ שמדובר במספר ולא באותיות טקסט.

<u>דוגמאות:</u>

MOV DX , 3456H

	DH	DL
DX	34H	56H

MOV AL , 0A2H

	AH	AL
AX	????	A2H

MOV AX , 376H

	AH	AL
AX	03H	76H

<u>העתקת נתון מאוגר לאוגר</u>

- גודל האוגרים חייב להיות זהה.
- האופרנד השמאלי הוא אופרנד היעד.

<u>דוגמאות:</u>

```
MOV  AL , AH
MOV  DX , CX
MOV  BH , DL
```

<u>תרגילים</u>

1. כתוב תוכנית המציבה את הנתון 35h באוגר DL ומעתיקה אותו ל- BL ו- CH.

<u>תשובה:</u>

```
MOV   DL , 35H
MOV   BL , DL
MOV   CH , DL
```

2. ערוך טבלת מעקב וכתוב מה ערכי האוגרים הבאים: AX , BX , CX, פרט איזה נתון מצוי בחלק העליון של כל אוגר ואיזה נתון בחלק התחתון של כל אוגר.

```
MOV   AX , 1234H
MOV   DL , AH
MOV   AL , DL
MOV   DX , 52H
MOV   CX , 123H
```

<u>תשובה:</u>

AX		**CX**		**DX**	
AH	AL	CH	CL	DH	DL
12H	34H	01H	23H	00H	12H
	12H				52H

<u>העתקת נתון מאוגר לתא זכרון והעתקת נתון מתא זכרון לאוגר</u>

בכל פעם שרוצים לסמן תא זכרון, משתמשים בסימן [ ].

למשל, כדי שהנתון המאוחסן באוגר AL יועתק לתא זיכרון 1000h היינו צריכים לכתוב את הפקודה הבאה: **MOV [1000H] , AL**

אבל למעבד 8086/88 ישנם מס' מגבלות:

1. כדי לציין כתובת של תא זכרון, אסור לכתוב בתוך הסוגריים המרובעים מס' אלא רק את אחד האוגרים **DI, SI** (אוגרי ההצבעה והאינדקס) או **BX** (שהוא גם אחד מאוגרי הנתונים). במקרים מיוחדים ניתן להשתמש גם באוגר BP.

<u>דוגמא:</u> אם האוגר BX מכיל את הערך H700 ואנו כותבים [BX] אנו פונים לתא זכרון שכתובתו H700.

2. לא ניתן להעביר נתון ישירות לתא זכרון. לצורך העברת נתון לתא זכרון יש לבצע שני תתי שלבים:
   - הצבת הנתון באוגר בגודל מתאים.
   - ביצוע פעולה בין תא זכרון לבין אוגר.
3. לא ניתן לבצע פעולות ישירות בין שני תאי זכרון כדוגמא הפקודה הלא חוקית הבאה:

mov [si], [di]

<u>דוגמאות</u>: העתקת נתון מאוגר לתא זכרון מתבצעת בכמה דרכים המודגמות בפקודות הבאות ע"פ הנתונים הבאים:

- SI = 100H
- DI=200H
- BX=1500H
- התווית NUMBER מצביעה על כתובת H2345.

אופרנד היעד הוא האופרנד השמאלי – תמיד !!!!

1. MOV AL , [SI] — הפקודה מעתיקה לתוך האוגר AL את הנתון היושב בזכרון בכתובת עליה מצביעה SI, כלומר הנתון היושב בכתובת H100 בזכרון.
2. MOV BX , [DI] — הפקודה מעתיקה לתוך האוגר BX את שני הבתים המופיעים בזכרון בכתובות H200 ו- H201, בהתאם לערכו של DI.
3. MOV CH , [SI+5] — הפקודה מעתיקה לתוך האוגר CH את תוכן הנתון היושב בכתובת H105 בזכרון.
4. MOV DX , [BX+SI] — הפקודה מעתיקה לתוך האוגר DX את שני הבתים המופיעים בזכרון בכתובות H1600 ו- H1601, בהתאם לערכם של BX ו- SI (וסכומם).

5. MOV AL , NUMBER הפקודה מעתיקה לתוך האוגר AL את תוכן הנתון היושב בזכרון הכתובת המסומנת ע"י התווית NUMBER, כלומר הנתון היושב בתא זכרון בכתובת 2345h.

6. MOV [SI] , AL כמו 1 אבל הפוך.

7. MOV [DI] , BX כמו 2 אבל הפוך.

8. MOV [SI+5] , CH כמו 3 אבל הפוך.

9. MOV [BX+SI] , DX כמו 4 אבל הפוך.

10. MOV NUMBER , AL כמו 5 אבל הפוך.

<u>תרגיל</u>

סמן אלו פקודות חוקיות ואילו אינן חוקיות.

1. MOV DL , [SI]
2. MOV [DI] , [SI]
3. MOV AL , [CX]
4. MOV [DI] , DL
5. MOV [BX] , BL
6. MOV [BX+5] , DL

<u>תשובה:</u>

פקודות לא חוקיות: 2,3

פקודות בעיתיות: 5

הפקודה ADD

ADD OP1 , OP2

הפקודה ADD מחברת את שני האופרנדים ומכניסה את התוצאה לתוך אופרנד היעד (האופרנד השמאלי). הדגלים שיושפעו הם: דגל הנשא (CF), דגל האפס (ZF) דגל הסימן (SF) ועוד דגלי מצב.

<u>דוגמאות:</u>

1. ADD AL , AH — AL <= AL+AH
2. ADD CL , [DI] — CL <= CL+[DI]

הנתון היושב בזכרון בכתובת עליה מצביע DI יחובר עם תוכן האוגר CL

3. ADD BX , CX — BX <= BX+CX

גודל האוגרים הוא של 16 סיביות

4. ADD DX , [SI] — DX <= DX+[SI]

הנתון היושב בזכרון בכתובת עליה מצביע SI ותא הזכרון שאחריו, יחוברו עם תוכן האוגר DX, התוצאה תשב באוגר DX.

5. ADD [SI] , BH — [SI] <= [SI]+BH

הנתון היושב בזכרון בכתובת עליה מצביע SI יחובר עם תוכן האוגר BH , התוצאה תוצב בזכרון בתא עליו מצביע האוגר SI.

הפקודה SUB

SUB OP1 , OP2

הפקודה SUB מבצעת את החיסור הבא: **OP1 <= OP1 - OP2**

הדגלים שיושפעו הם: דגל הנשא (CF), דגל האפס (ZF) דגל הסימן (SF) ועוד דגלי מצב.

<u>דוגמאות:</u>

1. SUB AL , AH — AL <= AL – AH
2. SUB CL , [DI] — CL <= CL - [DI]

הנתון היושב בזכרון בכתובת עליה מצביע DI יחוסר מתוכן האוגר CL

3. SUB BX , CX — BX <= BX – CX

גודל האוגרים הוא של 16 סיביות

4. SUB DX , [SI] — DX <= DX - [SI]

הנתון היושב בזכרון בכתובת עליה מצביע SI ותא הזכרון שאחריו, יחוסרו מתוכן האוגר DX, התוצאה תשב באוגר DX.

5. SUB [SI] , BH [SI] <= [SI] - BH

הנתון היושב באוגר BH יחוסר מתוכן תא הזכרון עליו מצביע האוגר SI התוצאה תוצב בזכרון בתא עליו מצביע האוגר SI.

<u>תרגיל</u>

1. כתוב תוכנית המוסיפה 4 לתוכן תא זכרון 990h.
2. כתוב תוכנית הכופלת פי 2 את הערך שנמצא בתא 558h ואח"כ מציבה את התוצאה בתא 560h בזכרון.
3. כתוב תוכנית המחברת את תוכן תא זכרון 1200h עם תוכן תא 1201h ומציבה את התוצאה בתא 1203h.
4. כתוב תוכנית המחליפה בין תוכן תא 700h לבין תוכן תא 800h.
5. כתוב תוכנית שתמצא מהו ההפרש בין תוכן תא 3000h לבין תוכן תא 200h, ותציב את התוצאה בתאים 100h ו- 102h.

<u>פתרונות:</u>

1.
```
MOV  AH , 4H
MOV  SI , 990H
ADD  [SI] , AH
```

2.
```
MOV  SI , 558H
MOV  AH , [SI]
ADD  AH , AH
MOV  [SI+2] , AH
```

3.
```
MOV  SI , 1200H
MOV  BL , [SI]
ADD  BL , [SI+1]
MOV  [SI+3] , BL
```

4.
```
MOV  SI , 700H
MOV  AH , [SI]
MOV  [SI+100H] , AH
```

5.
```
MOV  SI , 3000H
MOV  DI , 200H
MOV  BX , 100H
MOV  AL , [SI]
SUB  AL , [DI]
MOV  [BX] , AL
MOV  [BX+2] , AL
```

הפקודה INC

INC OP1

הפקודה INC מוסיפה 1 לתוכן האופרנד המופיע בפקודה. הדגלים שיושפעו הם: דגל האפס (ZF) דגל הסימן (SF) ועוד דגלים נוספים.

דוגמאות:

1. INC AL AL <= AL+1
2. INC SI SI <= SI+1

הפקודה DEC

DEC OP1

הפקודה DEC מחסירה 1 מערך האופרנד המופיע בפקודה. הדגלים שיושפעו הם: דגל האפס (ZF) דגל הסימן (SF) ועוד דגלים נוספים.

דוגמאות:

1. DEC BH BH <= BH-1
2. DEC DI DI <= DI-1

תרגיל

1. כתוב תוכנית המקדמת ב- 1 את ערכו של תא זכרון 754h.
2. כתוב תוכנית המחברת את ערכי התאים 305h ו- 306h ומחסירה מהסכום 1. את התוצאה יש להציב בתא זכרון 100h.

פתרונות:

1.
```
MOV  SI , 754H
MOV  AL , [SI]
INC  AL
MOV  [SI] , AL
```

2.
```
MOV  SI , 305H
MOV  AL , [SI]
ADD  AL , [SI+1]
DEC  AL
MOV  DI , 100H
MOV  [DI] , AL
```

הערה: השימוש בפקודה INC בשאלה 1 אינו יעיל במקרה זה, כיוון שללא הפקודה INC ניתן לכתוב את התוכנית ב- 3 פקודות.

מערכים – עמליה אפל

הצהרה על משתנים נעשית בסגמנט הנתונים – Data Segment.

```
name   type   value
num db 5
```

הצהרה על מערך

מערך הוא אוסף של איברים בעלי גודל זהה. לכל איבר במערך יש אינדקס המצביע על מיקומו במערך.

הצהרה על מערך

```
name  type value1, value2, value3
array1 db 2, 4, 6
array2 db 5 dup (0)
```

יוצר מערך של 5 איברים בגודל byte המאותחלים בערך 0

```
array3  dw 4 dup (?)
```

יוצר מערך של 4 איברים בגודל word שאין להם ערכים התחלתיים

```
array4 db 6 dup ('X')
```

יוצר מערך של 6 איברים בגודל byte המכילים את האות X.

לכל איבר במערך יש אינדקס המצביע על מיקומו במערך והכתובת שלו:

```
array1 db 2, 4, 6
```

כתובת בזיכרון	array1	array1 + 1	array1 + 2
אינדקס במערך	0	1	2
ערך	2	4	6

מצביע bx

bx הוא רגיסטר מיוחד. הוא משמש בעיקר כמצביע לכתובת בזיכרון.

פקודת offset

פקודת ה – offset מעתיקה לרגיסטר bx את הכתובת של תחילת המערך, האיבר הראשון במערך הוא באינדקס 0.

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

```
mov bx, offset array
```

לרגיסטר bx תועתק כתובת תחילת המערך,
כתובת האיבר הראשון במערך היא כתובת האיבר שנמצא באינדקס 0.

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

העתקת האיבר באינדקס 2 של המערך לתוך al:

תרגילים

א. הצהירו על מערך בן 4 איברים.
השמו בו ערכים בסדר עולה מ 1 עד 4.
הגדילו כל תא ב – h10.

כאשר רוצים להוסיף למשתנה במערך, שפנינו לכתובת שלו בעזרת רגיסטר [bx], מספר קבוע.

```
add [bx], 10h
```

בהרצת התוכנית נקבל הודעת שגיאה.

ARGUMENT NEEDS TYPE OVERRIDE

```
Assembling file:   t1.ASM
*Warning* t1.ASM(18) Argument needs type override
Error messages:    None
Warning messages:  1
Passes:            1
Remaining memory:  468k
```

הודעה זו היא מאחר והאסמבלי לא יודע האם התא שהפנינו אליו את הוא בגודל מילה, בית או מילה כפולה. לכן יש להודיע לו לאיזה גודל משתנה אנו רוצים לפנות וזאת נעשה ע"י פקודת:
BYTE PTR / WORD PTR

```
add [byte ptr bx], 10h
```

6. הצהירו על מערך של 5 אותיות.
השמו בו את האותיות: 'a','b','c','d'.
הפרש בייצוג של אותיות "גדולות" ו"קטנות" הוא 32. הפחיתו מכל אות 32 ובדקו את הייצוג של מערך ב – Data Segment האם מראה הייצוג של הנתונים השתנה?

כדי לראות את התוצאה בסגמנט הנתונים (*DATA SEGMENT*) הקישו *TAB* עד שתגיעו לייצוג של הזיכרון ב – *TD* אח"כ הקישו *CTRL + G* ואז בשורת התפריט הקישו *ENTER* ⇦ *DS:0*. חזרו לסגמנט הפקודות שוב ע"י הקשה על *TAB*.

בתרגילים עם מערכים כדי לקצר את התנית ניתן לבצע פעולות שחוזרות על עצמם משתמשים בלולאות. לולאה היא פקודה החזרת על עצמה מספר פעמים.

רגיסטר *CX* משמש כמונה הלולאה. (סופר כמה פעמים נבצע את הפעולה).
הוא מתחיל מהמספר שהצבנו לו ויורד בכל ביצוע ב – 1 עד שערכו 0.
לתחילת הלולאה נרשום תווית – שם לפעולה ונקודתיים - *change*:
ובסופה נרשום פקודה אשר תסמן לאסמבלי שמיקום זה הוא סוף הלולאה (ביצוע חוזר)
loop change

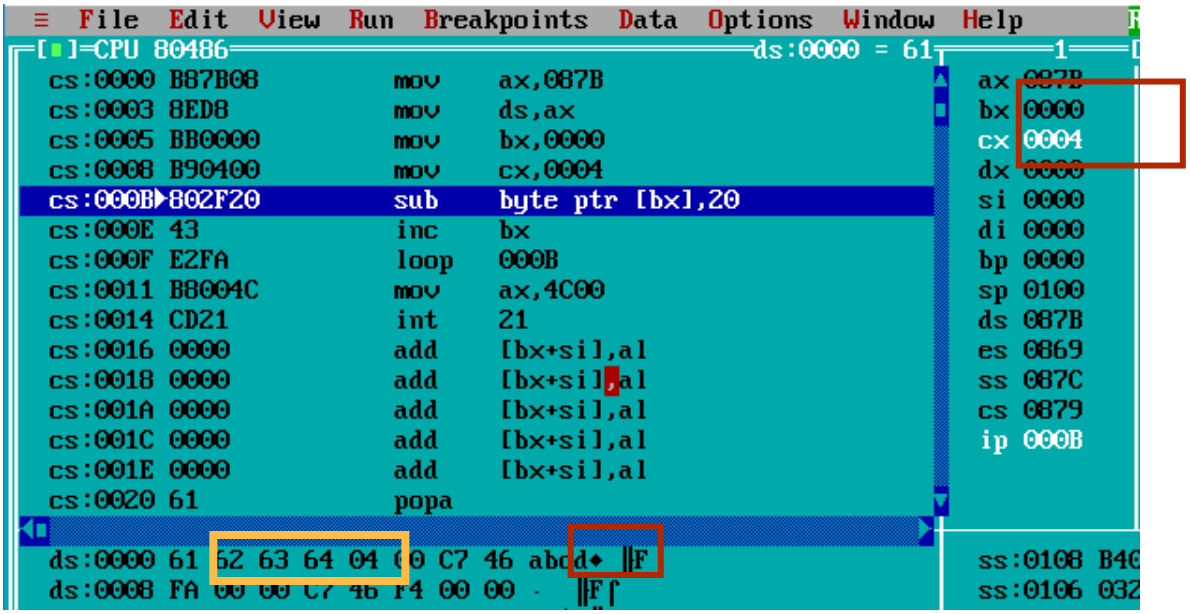
נריץ את התכנית הקודמת שוב אבל בסיוע לולאה.

```
DATASEG
; -------------------------
; Your variables here
        array1 db 'a','b','c','d'

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov bx, offset array1
        mov cx, 4
change:
        sub [byte ptr bx], 32
        inc bx
        loop change
; -------------------------
```

בתחילת הלולאה רגיסטר cx = 4

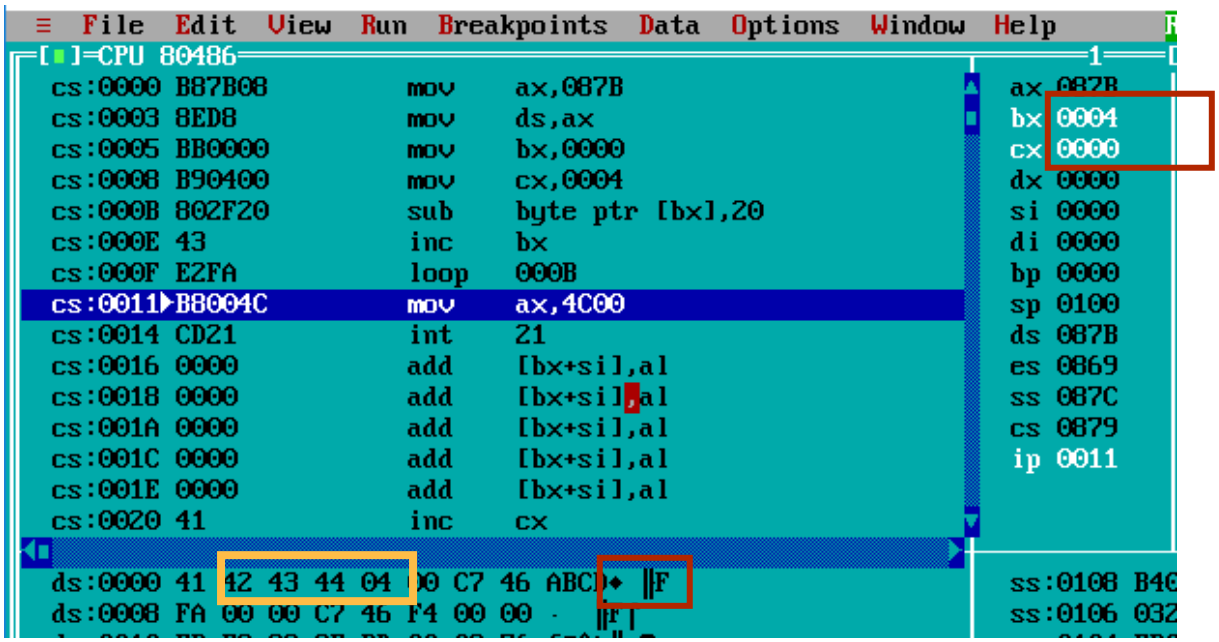
רגיסטר bx מצביע על תא בכתובת [00] (תחילת המערך)

הערכים בזיכרון הם של אותיות "קטנות" = abcd.



בסוף הלולאה רגיסטר cx = 0

רגיסטר bx מצביע על תא בכתובת [04] תא אחד אחרי סוף המערך, עברנו על כל תאי המערך)
הערכים בזיכרון הם של אותיות "גדולות" = ABCD.



7. כתבו תכנית המחשבת את סכום האיברים במערך.
המערך יהי מערך של בתים בעל 5 איברים. השמו במערך כל ערך.

8. כתבו תכנית בה מערך של מספרים, בעל 8 תאים. החליפו את המיקום של כל זוג מספרים.
(אינדקס 1 יוחלף עם אינדקס 2 וכך הלאה).

9. כתבו תכנית בה שני מערכים בעלי 8 בתים, ובכל אחד מהם ערכים שונים.
צרו מערך שלישי והשמו לכל תא את סכום שני המספרים שבאותו האינדקס במערכים שיצרתם.
לדוגמה: אם בתא 0 במערך הראשון ערך 2, ובתא 0 במערך שני ערך 3, תא 0 במערך 3 יקבל את הערך 5 (2+3).
ניתן לגשת לתא במערך בעזרת שם המערך + קבוע או שם המערך + רגיסטרים *si* או *di*.
לדוגמה: *[array+si]*

מערכים - תשובות

תרגילים

1. הצהירו על מערך בן 4 איברים.
השמו בו ערכים בסדר עולה מ 1 עד 4.
הגדילו כל תא ב – h10.
כאשר רוצים להוסיף למשתנה במערך, שפנינו לכתובת שלו בעזרת רגיסטר [bx], מספר קבוע.

```
add [bx], 10h
```

בהרצת התוכנית נקבל הודעת שגיאה.

ARGUMENT NEEDS TYPE OVERRIDE

```
Assembling file:   t1.ASM
*Warning* t1.ASM(18) Argument needs type override
Error messages:    None
Warning messages:  1
Passes:            1
Remaining memory:  468k
```

הודעה זו היא מאחר והאסמבלי לא יודע האם התא שהפנינו אליו את הוא בגודל מילה, בית או מילה כפולה. לכן יש להודיע לו לאיזה גודל משתנה אנו רוצים לפנות וזאת נעשה ע"י פקודת:

BYTE PTR / WORD PTR

```
add [byte ptr bx], 10h
```

```
DATASEG
; --------------------------
        array1 db 1,2,3,4
CODESEG
start:
        mov ax, @data
        mov ds, ax
; ---------------------------
        mov bx, offset array1
        add [byte ptr bx], 10h
        inc bx
        add [byte ptr bx], 10h
        inc bx
        add [byte ptr bx], 10h
        inc bx
        add [byte ptr bx], 10h
        inc bx
; --------------------------
```

2. הצהירו על מערך של 5 אותיות.
השמו בו את האותיות: 'a','b','c','d'.
הפרש בייצוג של אותיות "גדולות" ו"קטנות" הוא 32. הפחיתו מכל אות 32 ובדקו את הייצוג של מערך ב – Data Segment האם מראה הייצוג של הנתונים השתנה?

כדי לראות את התוצאה בסגמנט הנתונים (*DATA SEGMENT*) הקישו *TAB* עד שתגיעו לייצוג של הזיכרון ב – *TD* אח"כ הקישו *CTRL + G* ואז בשורת התפריט הקישו *ENTER* ⇦ *DS:0*. חזרו לסגמנט הפקודות שוב ע"י הקשה על *TAB*.

בתרגילים עם מערכים כדי לקצר את התנית ניתן לבצע פעולות שחוזרות על עצמם משתמשים בלולאות. לולאה היא פקודה החזרת על עצמה מספר פעמים.

רגיסטר CX משמש כמונה הלולאה. (סופר כמה פעמים נבצע את הפעולה).
הוא מתחיל מהמספר שהצבנו לו ויורד בכל ביצוע ב – 1 עד שערכו 0.
לתחילת הלולאה נרשום תווית – שם לפעולה ונקודתיים - change:
ובסופה נרשום פקודה אשר תסמן לאסמבלי שמיקום זה הוא סוף הלולאה (ביצוע חוזר) loop change

נריץ את התכנית הקודמת שוב אבל בסיוע לולאה.

```
DATASEG
; -------------------------
; Your variables here
        array1 db 'a','b','c','d'

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov bx, offset array1
        mov cx, 4
change:
        sub [byte ptr bx], 32
        inc bx
        loop change
; -------------------------
```

בתחילת הלולאה רגיסטר cx = 4

רגיסטר bx מצביע על תא בכתובת [00] (תחילת המערך)
הערכים בזיכרון הם של אותיות "קטנות" = abcd.

```
≡ File Edit View Run Breakpoints Data Options Window Help
[■]=CPU 80486                                   ds:0000 = 61
cs:0000 B87B08      mov   ax,087B                 ax 087B
cs:0003 8ED8        mov   ds,ax                   bx 0000
cs:0005 BB0000      mov   bx,0000                 cx 0004
cs:0008 B90400      mov   cx,0004                 dx 0000
cs:000B 802F20      sub   byte ptr [bx],20        si 0000
cs:000E 43          inc   bx                      di 0000
cs:000F E2FA        loop  000B                    bp 0000
cs:0011 B8004C      mov   ax,4C00                 sp 0100
cs:0014 CD21        int   21                      ds 087B
cs:0016 0000        add   [bx+si],al              es 0869
cs:0018 0000        add   [bx+si],al              ss 087C
cs:001A 0000        add   [bx+si],al              cs 0879
cs:001C 0000        add   [bx+si],al              ip 000B
cs:001E 0000        add   [bx+si],al
cs:0020 61          popa
ds:0000 61 62 63 64 04 00 C7 46 abcd♦ ‖F
ds:0008 FA 00 00 C7 46 F4 00 00 ·  ‖F⌠
```

בסוף הלולאה רגיסטר cx = 0
רגיסטר bx מצביע על תא בכתובת [04] תא אחד אחרי סוף המערך, עברנו על כל תאי המערך)
הערכים בזיכרון הם של אותיות "גדולות" = ABCD.

```
≡ File Edit View Run Breakpoints Data Options Window Help
[■]=CPU 80486
cs:0000 B87B08      mov   ax,087B                 ax 087B
cs:0003 8ED8        mov   ds,ax                   bx 0004
cs:0005 BB0000      mov   bx,0000                 cx 0000
cs:0008 B90400      mov   cx,0004                 dx 0000
cs:000B 802F20      sub   byte ptr [bx],20        si 0000
cs:000E 43          inc   bx                      di 0000
cs:000F E2FA        loop  000B                    bp 0000
cs:0011 B8004C      mov   ax,4C00                 sp 0100
cs:0014 CD21        int   21                      ds 087B
cs:0016 0000        add   [bx+si],al              es 0869
cs:0018 0000        add   [bx+si],al              ss 087C
cs:001A 0000        add   [bx+si],al              cs 0879
cs:001C 0000        add   [bx+si],al              ip 0011
cs:001E 0000        add   [bx+si],al
cs:0020 41          inc   cx
ds:0000 41 42 43 44 04 00 C7 46 ABCD♦ ‖F
ds:0008 FA 00 00 C7 46 F4 00 00 ·  ‖F⌠
```

3. כתבו תכנית המחשבת את סכום האיברים במערך.
המערך יהי מערך של בתים בעל 5 איברים. השמו במערך כל ערך.

```
DATASEG
; -------------------------
; Your variables here
        array db 5 dup (5)

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov bx, offset array
        mov cx, 5
        mov al, 0
sum:
        mov ah, [bx]
        add al, ah
        inc bx
        loop sum
; -------------------------
```

4. כתבו תכנית בה מערך של מספרים, בעל 8 תאים. החליפו את המיקום של כל זוג מספרים.
אינדקס 1 יוחלף עם אינדקס 2 וכך הלאה).

```
DATASEG
; -------------------------
; Your variables here
    array db 2,4,6,8,10,12,14,16

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov bx, offset array
        mov cx, 4
```

```
change:
        mov ah, [bx]
        mov al, [bx+1]
        mov [bx], al
        mov [bx+1], ah
        add bx,2
        loop change
; -------------------------
```

5. כתבו תכנית בה שני מערכים בעלי 8 בתים, ובכל אחד מהם ערכים שונים.
צרו מערך שלישי והשמו לכל תא את סכום שני המספרים שבאותו האינדקס במערכים שיצרתם.
לדוגמה: אם בתא 0 במערך הראשון ערך 2, ובתא 0 במערך שני ערך 3, תא 0 במערך 3 יקבל את הערך 5 (2+3).
ניתן לגשת לתא במערך בעזרת שם המערך + קבוע או שם המערך + רגיסטרים *si* או *di*.
לדוגמה: [*array+si*]

```
DATASEG
; -------------------------
; Your variables here
        array1 db 8 dup (3)
        array2 db 8 dup (2)
        array3 db 8 dup (?)

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov cx, 8
        mov si, 0
sum:
        mov al, [array1+si]
        add al, [array2+si]
        mov [array3+si], al
        inc si
        loop sum
; -------------------------
```

תנאים ולולאות - עמליה אפל

תוויות

באסמבלי ניתן לקפוץ לפקודה אחרת הסגמנט הפקודות (code segment). כדי להקל על הקפיצה אנו נותנים למקום שאליו נרצה לעבור תווית עם שם.

label:

פקודת המעבר לתווית היא פקודת קפיצה jmp.

jmp lable

פקודה זו תקפוץ לתווית ותבצע את הפעולות הבאות בתכנית.
פקודת הקפיצה מסייעת לנו בפעולות של השוואה, (תנאי) cmp.
התווית יכולה לסייע לנו ביצירת לולאות.

פעולת השוואה, תנאי

פעולת cmp מאפשרת לנו להשוות בין ערכים.
פעולה זו מתבצעת ע"י פעולת חיסור בין האופרנדים, אם נדלק דגל האפס ⇦ האופרנדים שווים.
הדלקה של דגלים אחרים (גלישה, נשא, סימן) מאפשרת למעבד לבדוק איזה אופרנד גדול יותר.

cmp אופרנד מקור, אופרנד יעד

ישנם כמה קפיצות על פי תוצאת ההשוואה. אם התנאי מתקיים ה – ip יקפוץ למיקום שהוגדר מראש. אחרת נמשיך לפקודה הבאה שלאחר פקודת הקפיצה.

```
cmp ax, 5
je Equal
```

אם הערך של ax = 5 נקפוץ לתווית Equal

לולאה

כאשר נרצה לבצע מספר פקודות כמה פעמים נבצע לולה.
הלולאה יכולה להתקיים כל עוד תנאי מתבצע מספר קבוע של פעמים.
לולאה המתקיימת כל עוד תנאי מתבצע:

```
        mov al, 10
check:
        cmp al, [num]
        je sof          ; Jump Equal
        dec al
        jmp check
sof:
```

לולאה המתבצעת מספר פעמים: מונה הלולאה יהיה רגיסטר **cx**.
cx - יקבל את מספר הפעמים לביצוע הלולאה,
בכל סבב של פקודות הוא יפחת באחד עד שיגיע ל - 0.

```
        mov cx, 5
check:
        dec al
        loop check
```

ריכוז פקודות *JMP*

כל פקודת **jmp** תחיל ב – **J** **N - not / E - equal** משמעות הפקודה	כל פקודת **jmp** תחיל ב – **J** **G - greater / L - less** מספרים **Signed**	כל פקודת jmp תחיל ב – J **A** – above / **B** - below מספרים **Unsigmed**
מאופרנד קפוץ אם היעד גדול המקור	JG – Jump if Greater	JA – Jump if above
קפוץ אם אופרנד היעד קטן מאופרנד המקור	JL – Jump if Less	JB – Jump if Below
קפוץ אם אופרנד היעד והמקור שווים	JE – Jump Equal	
קפוץ אם אופרנד היעד והמקור שונים	JNE – Jump Not Equal	
קפוץ אם אופרנד היעד גדול או לאופרנד המקור שווה	JGE – Jump if Greater or Equal	JAE – Jump if Above or Equal
קפוץ אם האופרנד היעד קטן לאופרנד המקור או שווה	JLE – Jump if Less or Equal	JBE – Jump if Below or Equal

שאלה 1 - קפיצה

לפניכם קטע תכנית.
בסופה ערך רגיסטר ax = 16d.
הוסיפו לתכנית פקודת jmp כך שערך רגיסטר ax בסוף ההרצה יהיה d10.

```
mov ax, 5
add ax, 5
add ax, 6
```

שאלה 2 – יצירת תנאי

שנו את התכנית שכתבתם כך שאם ערך רגיסטר ax גדול או שווה ל – 10 רגיסטר bx יקבל את הערך 10 אחרת רגיסטר bx יקבל את הערך 15.

שמו ערכים שונים לרגיסטר ax ובדקו את תוצאת הריצה.

הרחיבו את התכנית: כך שאם רגיסטר ax מקבל ערך קטן או שווה ל – 10 רגיסטר bx יקבל את הערך 10, אם רגיסטר ax יקבל ערך שבין 11 ל – 20 (כולל) רגיסטר bx יקבל את הערך 20.
אחרת יקבל את הערך 30.

שאלה 3 – לולאות

הפסיקה (הוראות) להדפסת תו למסך היא:
התו שיודפס למסך הוא התו שבמרכאות "#"

```
mov dl,'#'
mov  ah, 2h
int 21h
```

כתבו תכנית הדפיסה למסך 5 תווים.
שימו ל! בפסיקה יש שימוש ברגיסטר ah, עליכם להשתמש ברגיסטר אחר או משתנה לצורך ההשוואה כמות התווים שהודפסו

בהרצת התכנית ב – **Alt + F5** TD מציג את המסך, **Esc** מחזיר חזרה למסך ה – TD.

שנו את התכנית כך שנדפיס את התווים למסך בעזרת לולאת חזור

שאלה 4 – לולאות מקוננות

כתבו תכנית שתציג למסך את ריבוע של 4 תווים על 4 תווים.
בחרו איזה תו שתרצו.

לדוגמה: תו מספר 3 מצייר למסך ♥

לצורך ציור ריבוע תווים למסך יש צורך בשתי לולאות .חשבו כיצד לצייר את הריבוע:

♥♥♥♥
♥♥♥♥
♥♥♥♥
♥♥♥♥

הפסיקה למעבר שורה היא:

```
mov dl, 0ah    ; new line
mov ah, 2h
int 21h
```

טבלת אסקי

Low Ascii

000:	013:♪	026:→	039:'	052:4	065:A	078:N	091:[	104:h	117:u
001:☺	014:♫	027:←	040:(	053:5	066:B	079:O	092:\	105:i	118:v
002:☻	015:☼	028:∟	041:)	054:6	067:C	080:P	093:]	106:j	119:w
003:♥	016:►	029:↔	042:*	055:7	068:D	081:Q	094:^	107:k	120:x
004:♦	017:◄	030:▲	043:+	056:8	069:E	082:R	095:_	108:l	121:y
005:♣	018:↕	031:▼	044:,	057:9	070:F	083:S	096:`	109:m	122:z
006:♠	019:‼	032:	045:-	058::	071:G	084:T	097:a	110:n	123:{
007:•	020:¶	033:!	046:.	059:;	072:H	085:U	098:b	111:o	124:¦
008:◘	021:§	034:"	047:/	060:<	073:I	086:V	099:c	112:p	125:}
009:○	022:▬	035:#	048:0	061:=	074:J	087:W	100:d	113:q	126:~
010:◙	023:↨	036:$	049:1	062:>	075:K	088:X	101:e	114:r	127:⌂
011:♂	024:↑	037:%	050:2	063:?	076:L	089:Y	102:f	115:s	
012:♀	025:↓	038:&	051:3	064:@	077:M	090:Z	103:g	116:t	

שנו אתה התכנית כך שתצייר משולש של סימנים:

♥♥♥♥
♥♥♥
♥♥
♥

שאלה 5 לולאות ומערכים

צרו מערך של 6 מספרים,
הגדילו כל ערך במערך ב – 2.
בדקו את התוצאה.
מצאו את המספר הגדול במערך ושמרו אותו למשתנה max. הריצו ובדקו.
מצאו את המספר הקטן במערך ושמרו אותו למשתנה min.

שאלה 6 – מערך של ☺

לפניכם תכנית המציירת על המסך מערך של 4 ☺.
תצוגת המסך של ה – dos box היא 80 תווים לרוחב המסך על 25 תווים בגובה המסך.
ניתן למקם את סמן על פני המסך בציר x ובציר y.
יצרנו מערך של 4 מיקומים של הסמן על ציר x (עמודות שונות למיקום הסמן)
ומשתנה לציר y.
הפעלנו פיסקה (סט פקודות) המציירות את סימן ה - ☺ 2 באסקי, במיקום הסמן.
כתבו את תכנית והריצו.
אין צורך להיכנס ל – *td*. לאחר הפעלת *file_name* ***tasm*** נוצר קובץ *file_name.obj* . לאחר הפעלת *file_name* ***tlink***, נוצר קובץ *file_name.exe*. קובץ בסיומת *exe* הוא קובץ להפעלת התכנית במחשב. הריצו את הקובץ (רק את שמו) ובדקו את התוצאה.

אם התכנית לא עובדת כמצופה היכנסו ל – Turbo debugger ובדקו אותה.
שנו את הערכים במערך ובדקו את התוצאה על המסך.
נסו לשנות את משתנה y למערך ומקמו את התווים על פני המסך במיקומים שונים.
הוסיפו לולאה שתצייר עוד תווים במיקומים שונים על פני המסך.

```
DATASEG
; -------------------------
; Your variables here
        arrayX db 10, 12, 14, 17
        y db 10

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov al, 03h   ; set screen mode 80 * 25
        mov ah, 0
        int 10h
        mov cx, 4
        mov si, 0
print:
        mov dh, [y]             ; place cursor on  raw
        mov dl, [arrayX + si]   ; place cursor on  column
        mov bh, 0               ; page number
        mov ah, 2               ; place the cursor on the screen
        int 10h
        mov dl, 2               ; print sign
        mov ah, 2h
        int 21h
        inc si
        loop print
; -------------------------
exit:
        mov ax, 4c00h
        int 21h
END start
```

תנאים ולולאות - תשובות

שאלה 1 - קפיצה

לפניכם קטע תכנית.

בסופה ערך רגיסטר ax = 16d.

הוסיפו לתכנית פקודת jmp כך שערך רגיסטר ax בסוף ההרצה יהיה d10.

```
mov ax, 5
add ax, 5
add ax, 6
```

לתווית קפיצה

```
mov ax, 5
add ax, 5
jmp not_add
add ax, 6
```

הקוד במקטע ip - לקפיצה

```
mov ax, 5
add ax, 5
jmp cs:11h
add ax, 6
```

שאלה 2 – יצירת תנאי

שנו את התכנית שכתבתם כך שאם ערך רגיסטר ax גדול או שווה ל – 10 רגיסטר bx יקבל את הערך 10 אחרת רגיסטר bx יקבל את הערך 15.

שמו ערכים שונים לרגיסטר ax ובדקו את תוצאת הריצה.

```
        mov ax, 5
        add ax, 5
        cmp ax, 10
        jae bigger
        mov bx, 15
        jmp the_end
bigger:
        mov bx, 10
the_end:
```

הרחיבו את התכנית: כך שאם רגיסטר ax מקבל ערך קטן או שווה ל – 10 רגיסטר bx יקבל את הערך 10, אם רגיסטר ax יקבל ערך שבין 11 ל – 20 (כולל) רגיסטר bx יקבל את הערך 20.
אחרת יקבל את הערך 30.

```
        mov ax, 22
        cmp ax, 10
        ja bigger1
        mov bx, 10
        jmp sof
bigger1:
        cmp ax, 20
        ja bigger2
        mov bx, 20
        jmp sof
bigger2:
        mov bx, 30
sof:
```

שאלה 3 – לולאות

הפסיקה (הוראות) להדפסת תו למסך היא:
התו שיודפס למסך הוא התו שבמרכאות "#"

```
mov dl,'#'
mov  ah, 2h
int 21h
```

כתבו תכנית הדפיסה למסך 5 תווים.
שימו לב! בפסיקה יש שימוש ברגיסטר ah, עליכם להשתמש ברגיסטר אחר או משתנה לצורך ההשוואה כמות התווים שהודפסו

בהרצת התכנית ב **Alt + F5** TD מציג את המסך, **Esc** מחזיר חזרה למסך ה – TD.

שנו את התכנית כך שנדפיס את התווים למסך בעזרת לולאת חזור

חזור לולאת .

```
        mov cx, 5
print:
        mov dl, "%"
        mov ah, 2h
        int 21h
        loop print
```

לולאה מותנית

```
        mov cx, 1
print:
        mov dl, "%"  ; print sign
        mov ah, 2h
        int 21h
        cmp cx, 6
        jae bigger
        inc cx
        jmp print
bigger:
```

שאלה 4 – לולאות מקוננות

כתבו תכנית שתציג למסך את ריבוע של 4 תווים על 4 תווים.
בחרו איזה תו שתרצו.

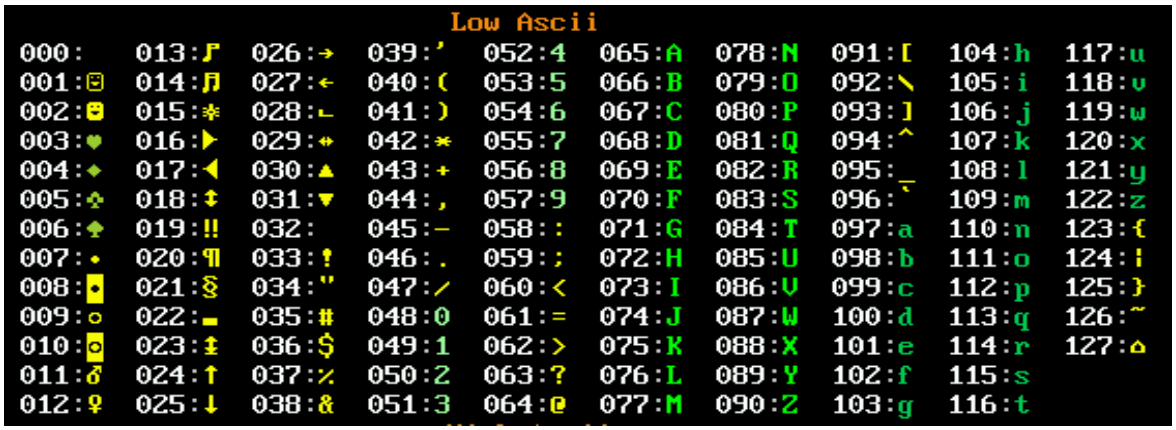
לדוגמה: תו מספר 3 מצייר למסך ♥

♥♥♥♥
♥♥♥♥
♥♥♥♥
♥♥♥♥

לצורך ציור ריבוע תווים למסך יש צורך בשתי לולאות. חשבו כיצד לצייר את הריבוע:

```
mov dl, 0ah    ; new line
mov ah, 2h
int 21h
```

טבלת אסקי



Low Ascii

000:	013:♪	026:→	039:'	052:4	065:A	078:N	091:[	104:h	117:u
001:☺	014:♫	027:←	040:(	053:5	066:B	079:O	092:\	105:i	118:v
002:☻	015:☼	028:∟	041:)	054:6	067:C	080:P	093:]	106:j	119:w
003:♥	016:►	029:↔	042:*	055:7	068:D	081:Q	094:^	107:k	120:x
004:♦	017:◄	030:▲	043:+	056:8	069:E	082:R	095:_	108:l	121:y
005:♣	018:↕	031:▼	044:,	057:9	070:F	083:S	096:`	109:m	122:z
006:♠	019:‼	032:	045:-	058::	071:G	084:T	097:a	110:n	123:{
007:•	020:¶	033:!	046:.	059:;	072:H	085:U	098:b	111:o	124:|
008:◘	021:§	034:"	047:/	060:<	073:I	086:V	099:c	112:p	125:}
009:○	022:▬	035:#	048:0	061:=	074:J	087:W	100:d	113:q	126:~
010:◙	023:↨	036:$	049:1	062:>	075:K	088:X	101:e	114:r	127:⌂
011:♂	024:↑	037:%	050:2	063:?	076:L	089:Y	102:f	115:s	
012:♀	025:↓	038:&	051:3	064:@	077:M	090:Z	103:g	116:t	

משולש של תווים:
שימוש במשתנה

```
        in_line db 3
חזור לולאת חיצונית לולאה
ברגיסטר שימוש ,מותנית לולאה פנימית לולאה
בשורה לציור התווים הפחתת לצורך bl
        mov cx, 4
new_row:
        mov bl, [in_line]
        mov dl, 0ah
        mov ah, 2h
        int 21h
print_line:
        mov dl, 3
        mov ah, 2h
        int 21h
        cmp bl, 0
        je continu
        dec bl
        jmp print_line
continu:
        dec [in_line]
        loop new_row
```

ריבוע של תווים:
שימוש במשתנה

```
row db 3
לולאה חיצונית לולאה מותנית,
לולאה פנימית לולאת חזור.
new_row:
        mov dl, 0ah  ; new line
        mov ah, 2h
        int 21h
        mov cx, 4
print_line:
        mov dl, 3  ; print sign
        mov ah, 2h
        int 21h
        loop print_line
        cmp [row], 0
        je end_print
        dec [row]
        jmp new_row
end_print:
```

שנו את התכנית כך שתצייר משולש של סימנים

שאלה 5 לולאות ומערכים

צרו מערך של 6 מספרים,
הגדילו כל ערך במערך ב – 2.
בדקו את התוצאה.
מצאו את המספר הגדול במערך ושמרו אותו למשתנה max\ הריצו ובדקו.
מצאו את המספר הקטן במערך ושמרו אותו למשתנה min.
בעמוד הבא

```
DATASEG
        array db 10, 12, 14, 16, 18, 20
        min db ?
        max db ?

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
        mov bx, offset array
        mov cx, 6
add2:
        add [byte ptr bx], 2
        inc bx
        loop add2

        mov cx, 5
        mov si, 0
        mov al, [array + si]
        inc si
find_min:
        cmp [array + si], al
        jae bigger
        mov al, [array + si]
bigger:
        inc si
        loop find_min
        mov [min], al

        mov cx, 5
        mov si, 0
        mov al, [array + si]
        inc si
find_nax:
        cmp [array+si], al
        jbe smaller
        mov al, [array + si]
smaller:
                inc si
        loop find_nax
        mov [max], al
```

שאלה 6 – מערך של ☺

לפניכם תכנית המציירת על המסך מערך של 4 ☺.
תצוגת המסך של ה – dos box היא 80 תווים לרוחב המסך על 25 תווים בגובה המסך.
ניתן למקם את סמן על פני המסך בציר x ובציר y.
יצרנו מערך של 4 מיקומים של הסמן על ציר x (עמודות שונות למיקום הסמן)
ומשתנה לציר y.

```
DATASEG
; -------------------------
; Your variables here
        arrayX db 10, 12, 14, 17
        y db 10

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Your code here

        mov al, 03h   ; set screen mode 80 * 25
        mov ah, 0
        int 10h
        mov cx, 4
        mov si, 0
print:
        mov dh, [y]             ; place cursor on  raw
        mov dl, [arrayX + si]   ; place cursor on  column
        mov bh, 0               ; page number
        mov ah, 2               ; place the cursor on the screen
        int 10h
        mov dl, 2               ; print sign
        mov ah, 2h
        int 21h
        inc si
        loop print
; -------------------------
exit:
        mov ax, 4c00h
        int 21h
END start
```

הפעלנו פיסקה (סט פקודות) המציירות את סימן ה - ☺ (2 באסקי) במיקום הסמן.
כתבו את תכנית והריצו.
אין צורך להיכנס ל – *td*!
לאחר הפעלת ***tasm*** *file_name* נוצר קובץ *file_name.obj*, לאחר הפעלת ***tlink*** *file_name*, נוצר קובץ *file_name.exe*. קובץ בסיומת *exe* הוא קובץ להפעלת התכנית במחשב. הריצו את הקובץ (רק את שמו) ובדקו את התוצאה.

אם התכנית לא עובדת כמצופה היכנסו ל – Turbo Debugger ובדקו אותה.

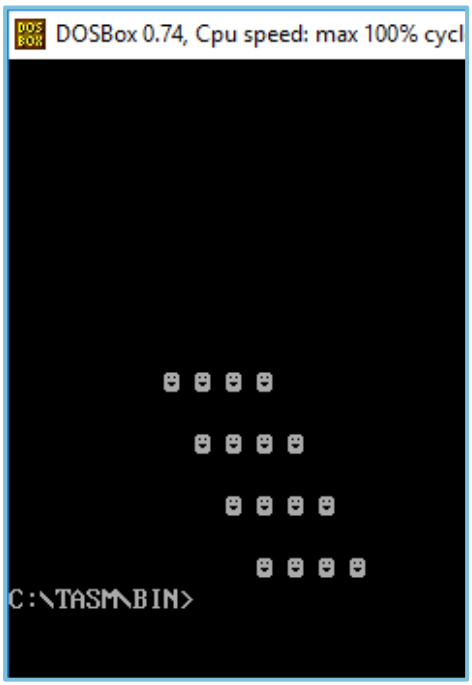
שנו את הערכים במערך ובדקו את התוצאה על המסך.

נסו לשנות את משתנה y למערך ומקמו את התווים על פני המסך במיקומים שונים.

הוסיפו לולאה שתצייר עוד תווים במיקומים שונים על פני המסך.

ציור של סמיילי על המסך לולאות מקוננות.

DATASEG




```
; Your variables here
        arrayX db 10, 12, 14, 16
        y db 10, 12, 14, 16
        num db 4

CODESEG
start:
        mov ax, @data
        mov ds, ax
; Your code here

        mov al, 03h  ; set screen mode 80 * 25
        mov ah, 0
        int 10h
add_print:
        mov cx, 4
        mov si, 0
print:
        mov dh, [y+si]          ; raw
        mov dl, [arrayX + si]   ; column
        mov bh, 0       ; page number
        mov ah, 2               ; place the cursor on the screen
        int 10h
        mov dl, 2  ; print sign
        mov ah, 2h
        int 21h
        inc si
        loop print
        mov cx, 4
        mov si, 0
add2:
        add [arrayX + si], 2
        inc si
        loop add2
        dec [num]
        cmp [num], 0
        je end_loop
        jmp add_print
end_loop:
```

הוראת תנאי – רחל פרלמן

תרגיל 1

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

```
int max, no1=5,
no2=6;
if (no1>no2){
   max=no1 ;
}
else{
   max=no2;
}
```

תרגיל 2

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

C#	JAVA

int a = int.Parse(Console.ReadLine());
int b = int.Parse(Console.ReadLine());
if ((a>5) && (b<4)) {
a=a-3;
b=b+3;
}
	int a = reader.nextInt();
	int b = reader.nextInt();
	if ((a>5) && (b<4)) {
	a=a-3;
	b=b+3;
	}

תרגיל 3

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

```
int
number=4,su
m=0;
for (int i=1;i<=4;i++){
    sum=sum+number;
    number=number+4;
}
```

תרגיל 4

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

```
int number=4,sum=0;
for (int
i=1;i<=4;i++){
        sum = sum +
    number;
        number = number +
    4;
}
```

תרגיל 5

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

```
int num = 15:
while (num>10){
        num=num-1;
}
```

תרגיל 6

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

C#	JAVA

int num1 =
int.Parse(Console.ReadLine());
int num2 =
int.Parse(Console.ReadLine());
while (num1>5 && num2 >5){
Console.WriteLine('X');
num1=num1-1;
num2=num2+1
}
	int num1 = reader.nextInt();
	int num2 = reader.nextInt();
	while (num1>5 && num2 >5){
	system.out.println('X');
	
	num1=num1-1;
	num2=num2+1;
	}

תרגיל 7

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

```
int a = 30;
int x = 60;
while (a>20){
```

```
if (x>50){
x=x-10;
 }
else x=x+1;
a = a-2;
}
```

תרגיל 8

כתבו תוכנית באסמבלי שמבצעת את ההוראות הבאות:

C#	JAVA

int i=12;
int j=8;
int k=3
int c=i+j-k;
for (int m=1;m<=c; m++){
Console.WriteLine('#');
}
	int i=12;
	int j=8;
	int k=3
	int c=i+j-k;
	for (int m=1;m<=c; m++){
	system.out.println('#');
	}

תרגיל 8

כתוב תוכנית באסמבלי שתציג את הפלט הבא :

```
# לשרטוט הפסיקה
היא המסך על:
mov dl,'#'
mov ah,2h
int 21h
```

```
שורה לירידת הפסיקה
mov dl, 0ah
mov ah, 2h
int 21h
```

X XX XXX XXXX XXXXX	XXXXX XXXX XXX XX X	XXXXX XXXXX XXXXX XXXXX

הוראות תנאי – סוניה שמאי

בד"כ סדר ביצוע ההוראות בתוכנית הוא כסדר מיקומן בזכרון. אולם לעיתים צריכים לשבור את הסדר הזה לבצע קפיצה - כלומר להמשיך את התוכנית בהוראה שאינה נמצאת בזכרון מיד לאחר ההוראה הנוכחית.

קפיצה מבוצעת ע"י טעינת הכתובת ממנה יש להמשיך את התוכנית לתוך האוגר IP.

2 סוגי הוראות קפיצה:

1. קפיצה שאינה מותנית JMP.
2. קפיצה מותנית $J_{תנאי}$ או באנגלית J_{cond}.

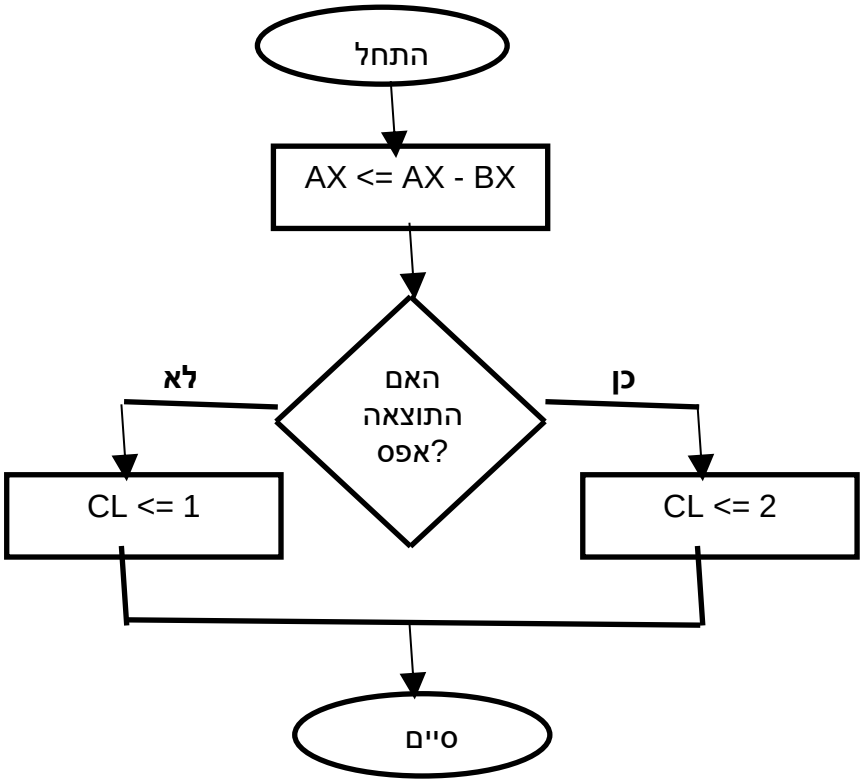
קפיצה שאינה מותנית תתבצע תמיד ואילו קפיצה מותנית תתבצע רק אם התנאי שהיא מציבה יתקיים.

תתבצע קפיצה למקום המסומן בעזרת התווית ABC	ABC	JMP
תתבצע קפיצה למקום המסומן בעזרת התווית SOF רק אם תוצאת החישוב האחרונה היא אפס	SOF	JZ
תתבצע קפיצה למקום המסומן בעזרת התווית CONT רק אם בחישוב האחרון היה נשא או לווה.	CONT	JC
תתבצע קפיצה למקום המסומן בעזרת התווית SHLILI רק אם תוצאת החישוב האחרונה היא שלילית.	SHLILI	JS

שימוש בפקודות קפיצה מותנית ולא מותנית

נתון קטע התוכנית הבא:

התוכנית בודקת אם הערכים של האוגרים AX ו- BX שווים. אם שווים מכניס לאוגר CL את הערך 2.
אם לא שווים מכניס ל- CL את הערך 1.



הסבר התוכנית	התוכנית		
הפקודה מבצעת חיסור בין AX ל- BX		SUB	AX,BX
אם התוצאה 0, מתבצעת קפיצה לתווית SHAVIM		JZ	SHAVIM
אם הגענו לפקודה זו סימן שהתוצאה אינה אפס ולכן אינם שווים. עכשיו עלינו להכניס לאוגר CL את הערך 1.		MOV	CL,1
סיימנו את המטלה ולכן יש לקפוץ לסוף התוכנית.		JMP	END
אם הגענו לפקודה זו סימן שהתוצאה אפס ולכן שווים. עכשיו עלינו להכניס לאוגר CL את הערך 2.	SHAVIM:	MOV	CL,2
סיום התוכנית. הפקודה NOP לא עושה כלום.	END:	NOP	NOP

בעצם כאשר מתבצעת קפיצה למשל: JZ SHAVIM היע"מ בודק איזה כתובת מסתתרת מאחורי התווית SHAVIM ומחליף אותה. למשל אם התווית SHAVIM מייצגת את הכתובת הלוגית A1CH0 אז הפקודה תתורגם כך: JZ 0A1CH.

אח"כ IP מקבל את ערך הכתובת החדשה ומתבצעת קפיצה.

<u>תרגילים</u>

<u>הערה:</u> רצוי לצרף לכל תוכנית תרשים זרימה.

1. כתוב תוכנית הבודקת אם הערך שבתא 234h הוא 3. אם כן – התוכנית מציבה FFH בתא H800, אם לא – היא מסתיימת.
2. כתוב תוכנית שבודקת אם תא זכרון 159h מכיל ערך גדול מזה של תא 357h. אם כן – הצב 1 בתא זכרון 300h, אם לא – הצב 0 בתא זכרון 300h.
3. כתוב תוכנית הבודקת אם הסכום של ערכי התאים 220h ו- 221h, גדול מערך התא 500h. אם כן – התוכנית תציב 0 בתאים 600h ו- 601h. אם לא – התוכנית תציב 99 בתאים אלו.
4. כתוב תוכנית הבודקת אם תאי זכרון בכתובות 666h ו- 777h מכילים ערכים זהים. אם כן – יש להציב בכתובת 1000h את הערך הזהה, אחרת הצב את ההפרש ביניהם בכתובת זו.
5. כתוב תוכנית הבודקת את הנתון היושב בתא 100h והנתון היושב בכתובת 200h ופועל ע"פ הטבלה הבאה:

[100h] > [200h]	[300h] ← [100h] - [200h]
[100h] < [200h]	[300h] ← [200h] - [100h]
[100h] = [200h]	[300h] ← [100h] + [200h]

6. כתוב תוכנית הבודקת אם ערך התא המסומן ע"י התוית NUMBER, הוא בתחום שבין 50 ל- 70. אם כן יש להציב ערך זה בתא המסומן באמצעות התווית IN_RANGE.
7. כתוב תוכנית המציבה בתא 900h את הערך הגבוה מבין התאים 400h, 401h, 402h.

8. בדוק מה מבצעת כל אחת מהתוכניות הבאות:

תוכנית א'

```
        MOV  BX , 666H
        MOV  CH , [BX]
        INC  BX
        MOV  CL , [BX]
        CMP  CL , CH
        JNE  FIN
        MOV  CL , 0
        MOV  CH , 0
FIN:    NOP
```

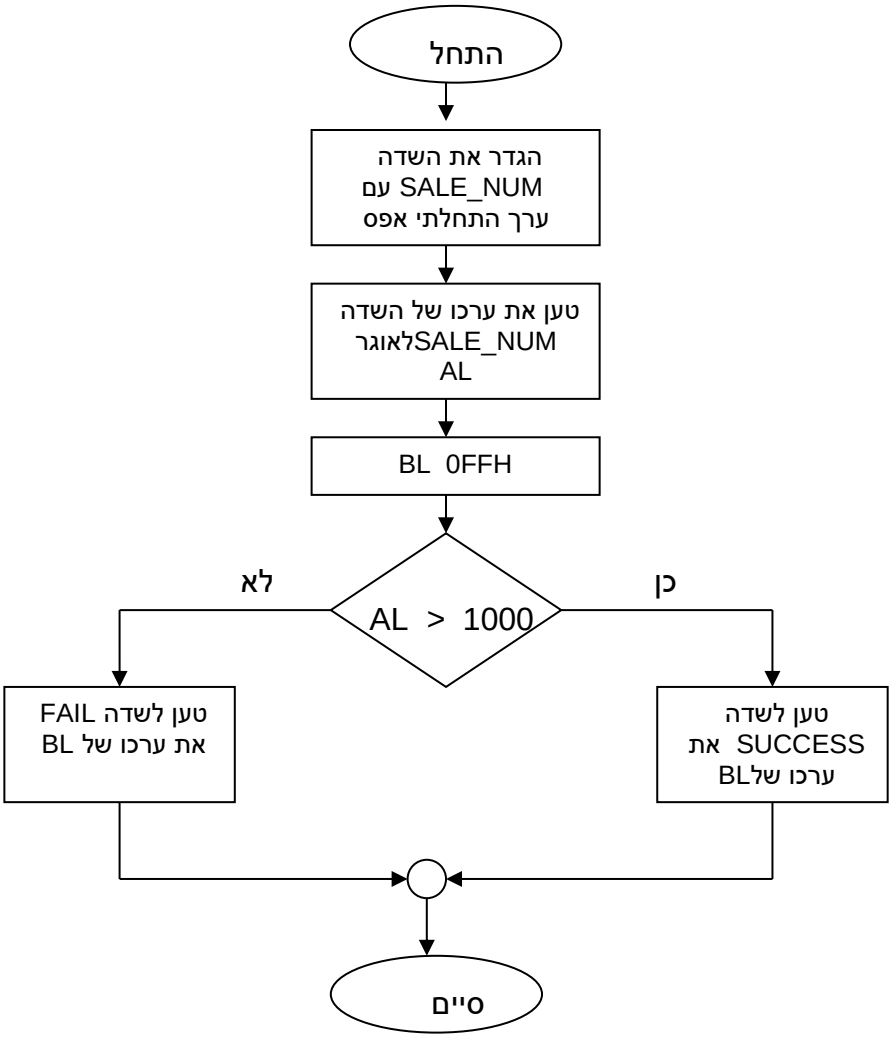
תוכנית ב'

```
        MOV  AL , 8
        MOV  DI , 1000H
        CMP  [DI] , AL
        JE   YES
        JMP  SOF
YES:    MOV  AL , 0
        MOV  [DI] , AL
SOF:    NOP
```

תרגיל מס' 1

כתוב תוכנית מלאה מלווה בתרשים זרימה, המגדירה שדה של מס' המוצרים הנמכרים במפעל. אם מס' המוצרים גדול מ- 1000 "הדלק" את הבית שבשדה FAIL ו"כבה" את הבית שבשדה SUCCESS. אחרת, "הדלק" את SUCCESS ו"כבה" את FAIL.

תרשים זרימה



תוכנית מס' 1

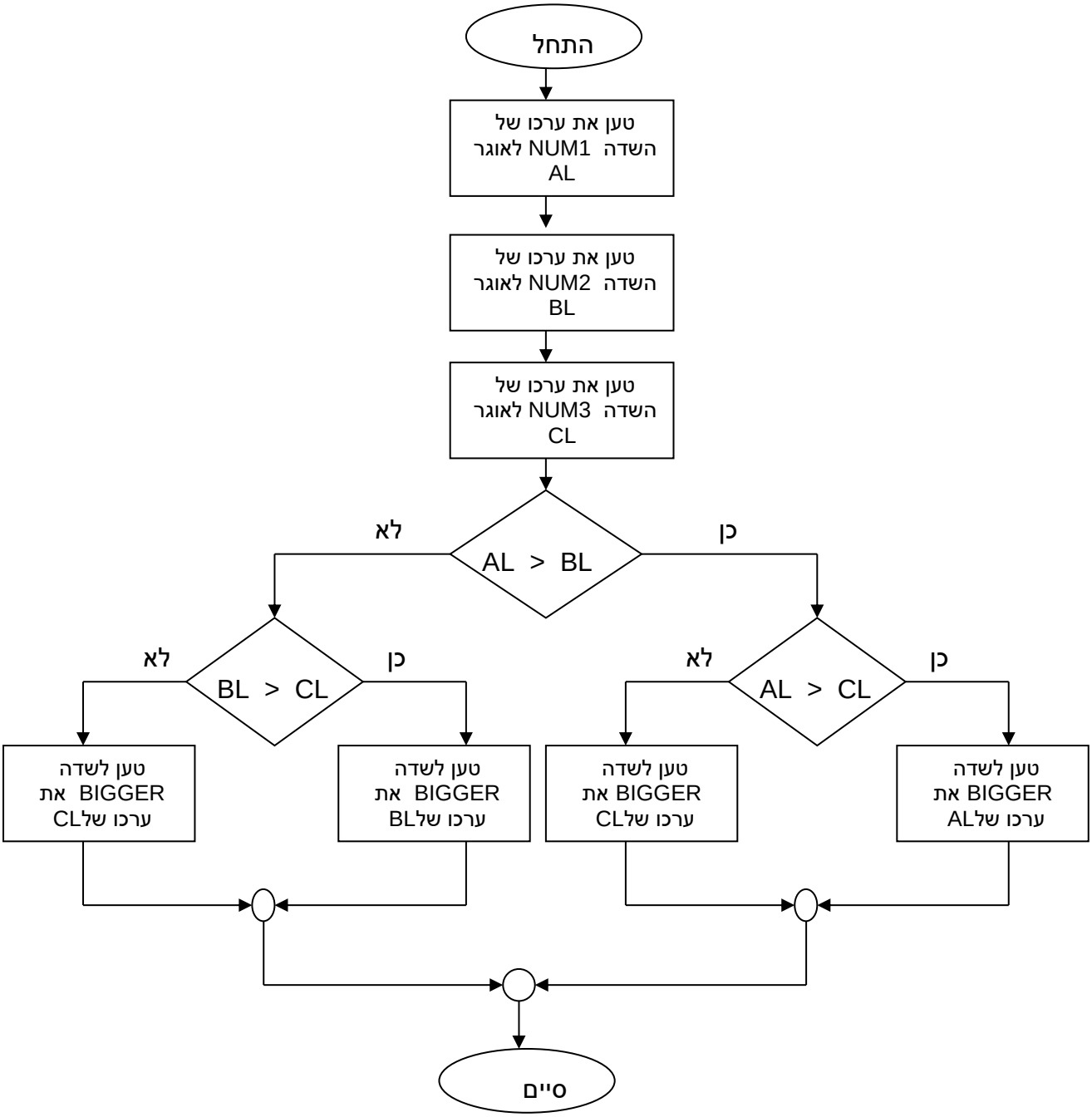
זוהי תוכנית הבודקת האם מספר המכירות גבוה מ- 1000, אם כן מדליקה את SUCESS ואם לא מדליקה את FAIL.

```
SSEG        SEGMENT
            DB    100H  DUP(?)
SSEG        ENDS
DSEG        SEGMENT
SALE_NUM    DW    1700   ; מספר המכירות
SUCCESS     DB    00000000B
FAIL        DB    00000000B
DSEG        ENDS
CSEG        SEGMENT
            ASSUME     CS:CSEG , DS:DSEG , SS:SSEG
START:      MOV   AX , DSEG
            MOV   DS , AX
            MOV   AL , SALE_NUM
            MOV   BL , 0FFH
            CMP   AL , 1000
            JH    GOOD
            MOV   FAIL , BL
            JMP   FIN
GOOD:       MOV   SUCCESS , BL
FIN:        MOV   AH , 4CH
            INT   21H
CSEG        ENDS
            END   START
```

תרגיל מס' 2

כתוב תוכנית מלאה מלווה בתרשים זרימה, המגדירה 3 שדות של מספר וטוענת את המספר הגבוה לשדה BIGGER.

תרשים זרימה



תוכנית מס' 2

התוכנית טוענת שלושה מספרים הנמצאים בזיכרון וטוענת את הגדול מביניהם ל**bigger**.

```
SSEG        SEGMENT
            DB    100H  DUP(?)
SSEG        ENDS
DSEG        SEGMENT
NUM1        DB    1AH
NUM2        DB    B5H
NUM3        DB    13H
BIGGER      DB    ?
DSEG        ENDS
CSEG        SEGMENT
            ASSUME    CS:CSEG , DS:DSEG , SS:SSEG
BEGIN:      MOV   AX , DSEG
            MOV   DS , AX
            MOV   AL , NUM1
            MOV   BL , NUM2
            MOV   CL , NUM3
            CMP   AL , BL
            JH    AL_H_BL
            CMP   BL , CL
            JH    BL_HIGH
CL_HIGH:    MOV   BIGGER , CL
            JMP   SOF
BL_HIGH:    MOV   BIGGER , BL
            JMP   SOF
AL_H_BL:    CMP   AL , CL
            JH    AL_HIGH
            MOV   BIGGER , CL
            JMP   SOF
AL_HIGH:    MOV   BIGGER , AL
SOF:        MOV   AH , 4CH
            INT   21H
```

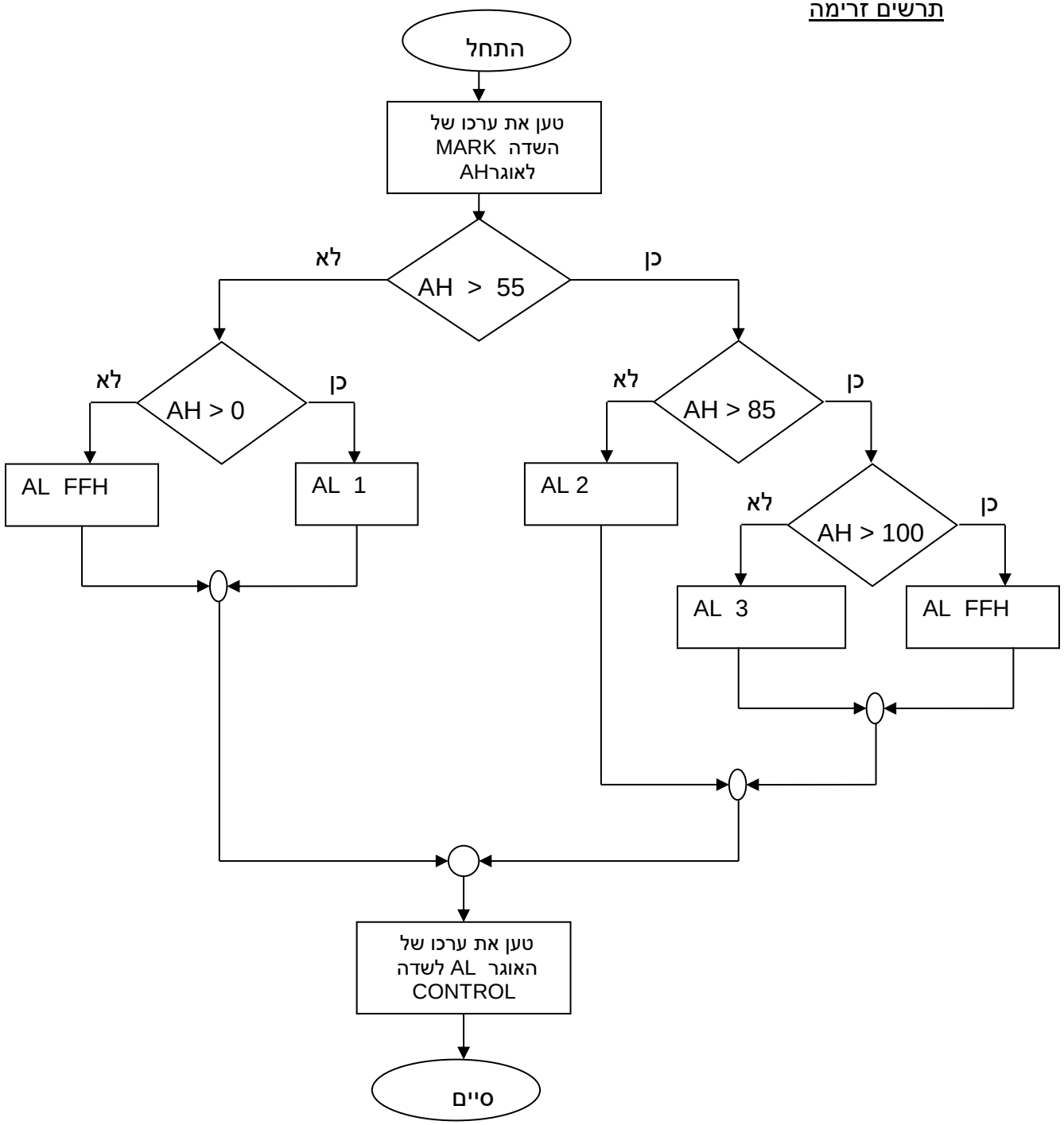
```
CSEG        ENDS
            END   BEGIN
```

<u>תרגיל בונוס</u>

כתוב תוכנית מלאה מלווה בתרשים זרימה, המגדירה שדה ציון ובודקת האם הציון הוא שלילי, בינוני, מצטיין או שגוי ומדווחת בהתאם בשדה CONTROL.

- אם שלילי (0-54) יש לטעון 1 לשדה CONTROL.
- אם בינוני (55-85) יש לטעון 2 לשדה CONTROL.
- אם מצטיין (86-100) יש לטעון 3 לשדה CONTROL.
- אם שגוי (לא בתחום שבין 0 ל- 100) יש לטעון FFH לשדה CONTROL.

<u>תרשים זרימה</u>



תוכנית בונוס

```
SSEG        SEGMENT
            DB    100H  DUP(?)
SSEG        ENDS
DSEG        SEGMENT
MARK        DB    89
CONTROL     DB    ?
DSEG        ENDS
CSEG        SEGMENT
            ASSUME    CS:CSEG , DS:DSEG , SS:SSEG
PROG:             MOV   AX , DSEG
            MOV   DS , AX
BEGIN:      MOV   AH , MARK
            CMP   AH , 55
            JH    H_THEN_55
L_THEN_55:  CMP   AH , 0
            JH    BEN_0_55
WRONG:      MOV   AL , 0FFH
            JMP   HAZAVA
BEN_0_55:   MOV   AL , 1
            JMP   HAZAVA
H_THEN_55:  CMP   AH , 85
            JH    H_THEN_85
            MOV   AL , 2
            JMP   HAZAVA
H_THEN_85:  CMP   AH , 100
            JH    WRONG
MITZTAYEN:  MOV   AL , 3
HAZAVA:     MOV   CONTROL , AL
            MOV   AH , 4CH
            INT   21H
CSEG        ENDS
            END   PROG
```

תרגיל פעולות לוגיות ופקודות בקרה - דנה אבן חיים

א. סוכנים וסוכנות יקרים,
נבחרתם להישלח למשימה בה אתם צריכים להעביר חבילת כספים למקום מסוים בשעה מדויקת.

קיבלתם מה - FBI מזוודה המכילה שני פריטים:

1. מסר טקסטואלי המכיל את השעה בה תצטרכו לבצע את המשימה, אך מטעמי אבטחה הוא מוצפן. כנסו לקישור הזה כדי להיכנס לתוכנית:

https://drive.google.com/open?id=1EFVcRK7-2VOVyTSJ6nZ5bMID9-pCGQ-q

תוכלו לראות את המסר הסודי ב - DATASEG כמשתנה הנקרא "clock"

2. מכתב עם התוכן הבא:

"מפתח ההצפנה מוגדר בתוכנית כמספר בינארי בן שמונה ביטים.

זהו מספר שכל התקדמות בחזקותיו מתבטאת בכמות אפסים מימינו הגדלה בכפולות של שלוש"

3. על מנת לבצע את המשימה בהצלחה, תצטרכו לפתוח את ההצפנה באמצעות המפתח. לאחר שתפענחו, המסר יהפוך משרבוט לא קריא לשעה המדויקת! בשביל לראות את המסר לאחר ההצפנה, תוכלו להדפיס אותו ע"י הוספת הפקודות הבאות (בסוף התוכנית) כדי לוודא שאכן הצלחתם. – בשלב זה לא חשוב להבין מה פקודות אלו מבצעות.

```
mov dx, offset CLOCK
mov ah, 9h
int 21h
```

מספר דגשים:
א. אין להשתמש בלולאות.
ב. כדי לדעת שהגעתם לסוף מערך המחרוזות ב – DS, תוכלו להשתמש בתו ה – '$' הייחודי שנמצא בסיומו.
ג. כדי להקל על העבודה, תוכלו להשתמש בפקודות בקרה כדי לבצע דה – הצפנה על כל המערך.

ב. הכניסו את השעה שמצאתם לרגיסטר AL.
כעת, הציגו את המספר בצורתו השלילית (סימן מינוס) באמצעות שיטת המשלים ל – 2, ללא שימוש בפקודת **Neg**, או רגיסטרים נוספים.

פרוצדורות ומערכים – סוניה שמאי

תרגיל 1:

כתוב תכנית שתזמן את הפרוצדורה:

SWAPBYREG(a, b)

המחליפה בין ערכי שני פרמטרים וממשת העברת פרמטרים לפי השיטה של By Address . לפני הזימון של הפרוצדורה עלינו לדחוף למחסנית את הכתובות של המשתנים a ו- b . להלן הגדרות הנתונים:

```
.MODEL SMALL
.STACK 100h
.DATA
        a       DW      12h
        b       DW      A9h
.CODE
```

תרגיל 2:

א. נניח שהמשתנים a ו- b- שכתבנו בדוגמה 1 הם מטיפוס בית, האם צריך להכניס שינויים בתכנית? אם לא – הסבירו מדוע, ואם כן – בצעו את השינויים הדרושים.

ב. השתמשו בפרוצדורה **SWAPBYREF** כדי לכתוב תכנית שתמיין שלושה משתנים a, b, c מטיפוס מילה, בסדר עולה.

תרגיל 3:

כתוב תכנית שתסכם את n האיברים הראשונים בסדרת המספרים … 1,3,5,7 . התכנית תשתמש בפרוצדורה **SUMNUM** המסכמת n איברים בסדרה. הפרוצדורה תקבל את הפרמטרים האלה: n כפרמטר לפי ערך ו - sum כפרמטר לפי כתובת.

בתכנית מוגדרים שני משתנים:

- משתנה n מטיפוס מילה המכיל את מספר האיברים בסדרה
- משתנה sum מטיפוס מילה שבו יישמר סכום הסדרה.

תרגיל 4:

כתוב תכנית שתזמן פרוצדורה המסכמת איברים במערך, החל מהאיבר ה- iעד האיבר האחרון (n) .
הפרוצדורה מחזירה את התוצאה במשתנה sumarr.

בתכנית מוגדרים המשתנים האלה:

- מערך a מטיפוס מילה
- משתנה n מטיפוס מילה, המכיל את מספר האיברים במערך.
- משתנה i מטיפוס מילה, שהוא אינדקס של איבר במערך.
- משתנה sumarr מטיפוס מילה, שיכיל את סכום האיברים.

התכנית תשתמש בפרוצדורה sum המסכמת ומחזירה את סכום האיברים במערך a , החל מהאיבר ה -i ועד סוף המערך. הפרוצדורה תקבל כפרמטרים את הנתונים האלה:
n ו- i כפרמטר לפי ערך.

- כתובת המערך a וכתובת משתנה sumarr כפרמטר לפי כתובת.
- הפרוצדורה תחזיר את סכום הסדרה שחושבה.

תרגיל בנושא מערך "מסוג מחרוזת"

שאלה:

כתוב תוכנית המוצאת את אינדקס התו * במחרוזת 1STR (מחרוזת המוגדרת באמצעות DB) בגודל LEN (LEN זהו משתנה המוגדר EQU). את אינדקס התו במחרוזת יש להציב במשתנה INDEX המוגדר בגודל בית. הערה: אינדקס המחרוזת הוא מקומה של הכוכבית מתחילת המחרוזת ולא כתובתה.

קליק עכבר ופרויקט **Paint** - דנה אבן חיים ורחל פרלמן

<u>שאלה 1</u>

לפניכם תוכנית שצובעת פיקסל במקום ספציפי שנלחץ בעכבר בצבע אדום. (עמוד 300 בספר)

שנו את התוכנית כך שתדפיס ריבוע בגודל 5*5 בכל מקום שתהיה בו לחיצת עכבר. (הכניסו ללולאה)

כנסו לקישור הבא בשביל להוריד את התוכנית:

https://drive.google.com/open?id=1pa6Dsno0Xh7Ov0qRmwYVthXSL2cgwO4O

<u>שאלה 2</u>

כל לחיצה על העכבר תשנה את צבעו מאדום לכחול לחילופין.

<u>שאלה 3</u>

כל לחיצה על העכבר תשנה את מיקומו של הריבוע ימינה 2+x

<u>שאלה 4</u>

הכינו צייר עם 5 אפשרויות צבע

אלגוריתם:

לולאה המתן_ללחיצה:

המתן ללחיצת עכבר

אם נלחץ עכבר:

האם הלחיצה הייתה באזור הפלטה?

אם כן,שמור את הצבע וחזור ללולאת "המתן ללחיצה"

אם לא, תצייר פיקסל.

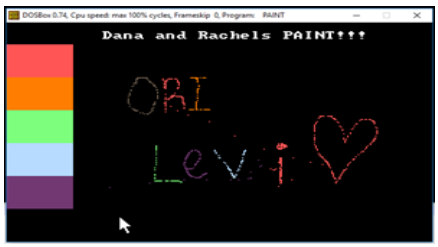
קריאת צבע ערך פיקסל מהמסך (הצבע יכנס לרגיסטר **al**):

```
Mov bh, 0h
Mov cx, [x]
Mov dx, [y]
Mov ah, 0Dh
Int 10h
```

נתקלתם בבעיה ? קראו כאן:

https://stackoverflow.com/questions/38505867/wrong-data-when-reading-pixels-in-mode-13-386-assembly-dos

דוגמה:



מקשי מקלדת – דנה אבן חיים ורחל פרלמן

על מנת לבדוק לחיצה במקשי המקלדת, נשתמש בשתי פסיקות:

א. פסיקה שבודקת אם יש בבאפר תו חדש (רצה בלולאה).

```
in al, 64h
cmp al, 10b
```

ב. אם יש תו חדש – פסיקה נוספת שתגלה לנו מהו אותו תו.

לתוך al יכנס ערך התו, ה-scan code

```
in al, 60h
```

לאחר שיש לנו את התו, נוכל לבדוק מהו ערכו ע"י השוואה לערך ה-scan code שלו. כלומר, קיימת טבלה המראה לנו את ערכו של כל תו (גם בלחיצה, וגם בשחרור).

בטבלה הבאה נוכל לראות:

Key	Down	Up	Key	Down	Up	Key	Down	Up	Key	Down	Up
ESC	1	81	[{	1A	9A	, <	33	B3	center	4C	CC
1 !	2	82	] }	1B	9B	. >	34	B4	right	4D	CD
2 @	3	83	Enter	1C	9C	/ ?	35	B5	+	4E	CE
3 #	4	84	Ctrl	1D	9D	R shift	36	B6	end	4F	CF
4 $	5	85	A	1E	9E	PrtSc	37	B7	down	50	D0
5 %	6	86	S	1F	9F	alt	38	B8	pgdn	51	D1
6 ^	7	87	D	20	A0	space	39	B9	ins	52	D2
7 &	8	88	F	21	A1	CAPS	3A	BA	del	53	D3
8 *	9	89	G	22	A2	F1	3B	BB	/	E0 35	B5
9 (	0A	8A	H	23	A3	F2	3C	BC	enter	E0 1C	9C
0)	0B	8B	J	24	A4	F3	3D	BD	F11	57	D7
- _	0C	8C	K	25	A5	F4	3E	BE	F12	58	D8
= +	0D	8D	L	26	A6	F5	3F	BF	ins	E0 52	D2
Bksp	0E	8E	; :	27	A7	F6	40	C0	del	E0 53	D3
Tab	0F	8F	"	28	A8	F7	41	C1	home	E0 47	C7
Q	10	90	` ~	29	A9	F8	42	C2	end	E0 4F	CF
W	11	91	L shift	2A	AA	F9	43	C3	pgup	E0 49	C9
E	12	92	\ |	2B	AB	F10	44	C4	pgdn	E0 51	D1
R	13	93	Z	2C	AC	Num	45	C5	left	E0 4B	CB
T	14	94	X	2D	AD	SCRL	46	C6	right	E0 4D	CD
Y	15	95	C	2E	AE	home	47	C7	up	E0 48	C8
U	16	96	V	2F	AF	up	48	C8	down	E0 50	D0
I	17	97	B	30	B0	pgup	49	C9	R alt	E0 38	B8
O	18	98	N	31	B1	-	4A	CA	R ctrl	E0 1D	9D
P	19	99	M	21	B2	left	4B	CB	pause	E1 1D	-

לדוגמה, אם נרצה לראות אם נלחץ מקש ESC, נשווה את ערך התו שהתקבל מהפסיקה ל – 1. ואם נרצה לראות שהמקש שוחרר, נשווה את ערך התו שהתקבל מהפסיקה ל81. שימו לב – הערכים בטבלה נמצאים בפורמט של הקסה דצימלי.

בדף הבא תוכלו לראות דוגמה לתכנית שמדפיסה בלולאה; אם נלחץ ESC – תדפיס "You pressed ESC", וכשהמקש ישוחרר יודפס "You released ESC".

התכנית: לחצו על האייקון!

המשימות שלכם:

משימה 1:

עליכם לכתוב תוכנית, שאם לוחצים על מקש: A יודפס "Left", אם יילחץ מקש "D" יודפס "right", אם יילחץ מקש "w" יודפס "up" ואם יילחץ מקש "s" יודפס "down".

העזרו בשלד התוכנית הבא:

https://drive.google.com/open?id=1WIvAfNieZBKxamP7urlndZVC7DptO450

משימה 2:

א. השתמשו בידע שרכשתם בשיעור שעבר לגבי ציור פיקסל על מסך. ציירו ריבוע בגודל 100*100 בצבע אדום.

ב. כאשר ילחץ מקש a הריבוע יעלם מהמסך.

משימה 3:

עכשיו לאחר הידע שרכשתם ציירו אורגן עם 8 קלידים, לחיצה על מקש 1-8 תגרום לקליד ספציפי לשנות צבע בהתאמה, יש לחזור לצבע המקורי כשהמקש משתחרר.

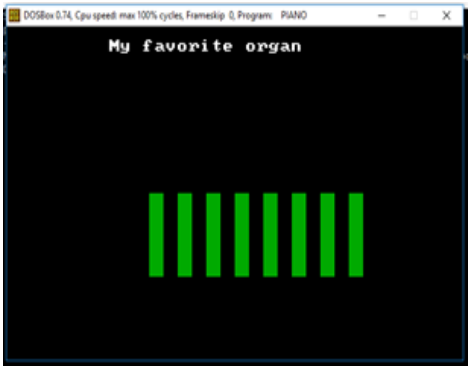
לעיונכם, מצרפים בדף הבא דוגמת הרצה, וכן רשימה של צבעים שונים והקוד שלהם.

List of BIOS color attributes

Dec	Hex	Binary	Color
0	0	0000	Black
1	1	0001	Blue
2	2	0010	Green
3	3	0011	Cyan
4	4	0100	Red
5	5	0101	Magenta
6	6	0110	Brown
7	7	0111	Light Gray
8	8	1000	Dark Gray
9	9	1001	Light Blue
10	A	1010	Light Green
11	B	1011	Light Cyan
12	C	1100	Light Red
13	D	1101	Light Magenta
14	E	1110	Yellow
15	F	1111	White

דוגמת הרצה

תמונה של הפסנתר לפני ואחרי הלחיצה על מקש "1":



בהצלחה!

פרויקט **draw** – מדריך לתלמיד - עמליה אפל

בדף עבודה זה נתנסה בעבודה במצב גראפי. (בשונה למצב טקסט)
נצייר ביחד ריבוע, קווים אנכיים וקווים אופקיים.

מעבר למצב גרפי

המעבד יכול לעבוד בשני מצבים הראשון מצב טקסט (בו עבדנו עד היום) והשני מצב גרפי בו משתמשים בגרפיקה (ציור ושרטוט על המסך) כדי לעבור למוד גראפי נשתמש בפסיקת ה bios 10h

```
mov ax, 13h
int 10h
```

צרו תכנית העוברת למוד גראפי. בדקו בעזרת ה Alt+ f5 שב – turbo debugger, מה קורה למסך ה – DosBox במוד זה?

חזרה ממצב גרפי:

```
mov ax, 2
int 10h
```

הוסיפו בסוף התוכנית שלכם את ה interrupt החוזר למצב text.
הריצו את התכנית ב – Turbo Debugger, כדי לצפות במסך המשתמש הקישו **Alt + F5**
חזרה למסך ה – TD הקישו **Esc**

ציור במצב גרפי – ציור פיקסל (נקודה על המסך)

המסך מורכב מנקודות. כל נקודה נקראת פיקסל. במצב גרפי המסך מורכב מ 200 שורות ו 320 עמודות של פיקסלים. לכל פיקסל מיקום וצבע.
המיקום מורכב משורה ועמודה -> row, column

0, 0							319,0
0,199							319, 199

הצבע:
בזיכרון המחשב יש טבלה בגודל 256 תאים ובכל תא שלה מצוי מספר המייצג צבע.
כדי לצייר פיקסל ניתן מיקום וצבע.
לדוגמה: שורה 0
עמודה 0
צבע 4

יצבע בצבע אדום את הפיקסל בפינה השמאלית העליונה

0, 0							319,0
0,199							319, 199

האינטרפט (פסיקה) המצייר פיקסל למסך, הוא פסיקת **int 10h** - bois
הפרמטרים עבור אינטרפט זה:

ah = 0ch – קוד הפסיקה
bh = 0 – איפוס רגיסטר bh
cx – עמודה
dx – שורה
ax - צבע

כתבו תכנית המציירת פיקסל אדום (תא 4 בפלטת הצבעים) בשורה 20 עמודה 50.

```
xor bh, bh
mov cx, [x_coordinate]
mov dx, [y_coordinate]
mov ax, [color]
mov ah, 0ch
int 10h
```

בהרצת התכנית ב – TD ניתן לראות את הציור על מסך (לאחר הפעלת הפסיקה) ע"י הקשה על המקשים **Alt+F5**

או בחירה בתפריט Window ו - User screen.

פרוצדורה

שנו את קטע הקוד לפרוצדורה העושה שימוש במשתנים לקואורדינטות x ו y של פיקסל ואת צבעו ומציירת אותו על המסך.

כדי לשמר את מצב הרגיסטרים בכל פעולה של פרוצדורה נבצע פעולות push ו – pop לכל הרגיסטרים.
ניתן לבצע push וpop לכל הרגיסטרים יחד.
נוסיף בתחילת התכנית את הפקודה 186p (לפני ה – DATASEGMENT).

הוספת מקש

נוסיף לתכנית פסיקה אשר בהקשה על מקש היא יוצאת מהמצב הגרפי וחוזרת למצב טקסט.

הפסיקה הבודקת אם הוקש מקש במקלדת היא:

```
; Wait for key press
        mov ah,00h
        int 16h
```

הוסיפו פסיקה זו לאחר הקריאה של הפרוצדורה לציור פיקסל. ולפני הפקודה של חזרה למצב של טקסט.

```
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Enter Graphic mod
        mov ax, 13h
        int 10h

        call drawPixel

; Wait for key press
        mov ah,00h
        int 16h

;Retern text mod
        mov ax, 2
        int 10h
; -------------------------
exit:
        mov ax, 4c00h
        int 21h
END start
```

במצב זה ניתן להריץ את התכנית ללא ה – Turbo Debugger.

הריצו את התכנית שוב ישירות

הריצו את הקובץ ב – tasm.
הריצו את הקובץ ב – tlink.
הריצו את הקובץ - רשמו את שמו בלבד ו – [Enter]

ליציאה מהתכנית הקישו על מקש (כל מקש).
הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור קו אופקי

קו הוא אוסף של נקודות ..... על ציר ה – x

כתבו פרוצדורה המציירת קו על המסך (לרוחב). הפרוצדורה עושה שימוש בקואורדינטת תחילת השורה אורך הקו, ובצבע הפיקסל ומציירת את השורה.

- לצורך ביצוע השרטוט הקו עלינו ליצור משתנה חדש שבו נשמור את ארוך הקו, [len]

ניצור פרוצדורה חדשה המציירת קו למסך - **drawHorizontalLine**

- בתחילת הפרוצדורה נבצע פקודת push לכל הרגיסטרים כדי לשמור את ערכם למחסנית.
- בתחילת התכנית (לפני ה – DATASEGMENT) נוסיף את ההוראה 186p
  בפרוצדורה זו יש שימוש בלולאה ולכן חשוב מאוד לשמר את הרגיסטרים בכלל ואת רגיסטר cx בפרט.
- בתחילת הפרוצדורה נכתב את הפקודה **pusha** - פקודה העושה push לכל הרגיסטרים.
- ניצור לולאה שערכה הוא כאורך הקו בפיקסלים (נעביר לרגיסטר cx את הערך של משתנה len).
  - נזמן את הפעולה של ציור הפיקסל מתוך פרוצדורה זו.
  - נגדיל את ערכו של [x_coordinate]
- בסיום הפרוצדורה נבצע פעולת **popa** לערכים של כל הרגיסטרים.

```
IDEAL
MODEL small
STACK 100h
p186
DATASEG
        x_coordinate dw 50          ; place in line
        y_coordinate dw 20          ; place in column
        len  dw 20
        color dw 5
CODESEG
start:
        mov ax, @data
        mov ds, ax

; Enter Graphic mod
        mov ax, 13h
        int 10h

        call drawHorizontalLine
```

```
proc drawHorizontalLine
; draw line of pixels
        pusha
        mov cx, [len]
drawLine:
        call drawPixel
        inc [x_ coordinate]
        loop drawLine
        popa
        ret
        endp drawHorizontalLine
```

הריצו ובדקו.

שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור קו אנכי

כתבו פעולה המציירת קו אנכי.
קו אנכי הוא אוסף נקודות על ציר y
כתבו פרוצדורה חדשה drawVerticalLine

אל תשכחו לפני הקריאה לפרוצדורה להשים ערכים למשתנים של תחילת ציור הקו על ציר ה – x ועל ציר ה – y.

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

List of BIOS color attributes

Dec	Hex	Binary	Color
0	0	0000	Black
1	1	0001	Blue
2	2	0010	Green
3	3	0011	Cyan
4	4	0100	Red
5	5	0101	Magenta
6	6	0110	Brown
7	7	0111	Light Gray
8	8	1000	Dark Gray
9	9	1001	Light Blue
10	A	1010	Light Green
11	B	1011	Light Cyan
12	C	1100	Light Red
13	D	1101	Light Magenta
14	E	1110	Yellow
15	F	1111	White

ציור מסגרת למסך

גודל המסך הוא רוחב 320 פיקסלים וגובה 200 פיקסלים.
ציירו מסגרת למסך כל צבע שתבחרו.
השאירו שוליים צרים מכל צד של מסגרת ה – DoxBox.
חשבו כך שגודל המסגרת יהי זוגי ויתחיל בנקודה זוגית בנקודות ציר
ה – x וציר ה – y.

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור ריבוע

ריבוע הוא אוסף של קווים

בציור הריבוע נשתמש בטכניקה שונה. זה יהיה השחקן שלנו שיזוז בתוך המסגרת. לכן חשוב לנו לשמור את נקודת ה – x וה – y של מיקום תחילת ציור הריבוע.
תזוזה של הריבוע על פני המסך היא ציור ריבוע שוחר על הריבוע הקיים – מחיקת הריבוע וציור ריבוע חדש בנקודה אחרת על המסך.

ניצור פרוצדורה חדשה לציור הריבוע. לצורך ציור הריבוע נשתמש בשתי לולאות מותנות.
נקבע 4 משתנים חשים:
מיקום תחילת הריבוע על ציר x,
מיקום תחילת הריבוע על ציר y,
משתנה שיהיה מונה הלולאה של ציור הקו האופקי
משתנה שיהיה מונה הלולאה על הציר האנכי (ציור הקווים)

```
x_begin   dw 100        ; Starting point on line
y_begin   dw 30         ; Starting point on column
x_count   db 10         ; loop count line draw
Y_count   db 10         ; loop count column draw
```

פרוצדורה לציור ריבוע בשתי לולאות מותנות:

```
Proc   DrawRectangle
; Draw square at x_begin, y_begin position, size 10*10
        mov  [x_count], 10
        mov  [y_count], 10
        mov  ax, [x_begin]
        mov  [x_coordinate], ax
        mov  ax,[y_begin]
        mov  [y_coordinate], ax
line_loop:
        ; Print one dot on screen
        call  drawPixel

        inc  [x_coordinate]
        dec  [x_count]
        cmp  [x_count],  0
        jnz  line_loop
```

קביעת גודל הריבוע והעברת הערכים של מיקם תחילת הריבוע ל משתנים x_coordinate ו – y_coordinate

לאחר ציור נקודה אחת על המסך, מגדילים את הערך של מיקום הנקודה על ציר האיקס (מיקום על השורה) ומקטינים את מונה הלולאה של ציור הקו האופקי ב 1 – כאשר מונה הלולאה מגיע ל 0 – יוצאים ממנה.

```
        mov ax, [x_begin]          ;reset line counters
        mov  [x_coordinate],  ax
        mov  [x_count],10          ;reset column counters
        inc  [y_coordinate]
        dec  [y_count]
        cmp  [y_count],  0
        jnz  line_loop

        ret
endp DrawRectangle
```

לאחר ציור קו אחד למסך, מאפסים את תחילת ציור הקו על ציר איקס.
מאפסים את מונה הלולאה המותנית של ציור קו אופקי.
מגדלים את מיקום ציור הנקודה על ציר y – ציר אופקי, עמודה.
מקטינים את מונה הלולאה של ציור הקווים – ציר אופקי.
כאשר מונה הלולאה של ציור הקווים מגיע לאפס, יוצאים משתי הלולאות.

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

הזזת השחקן על פני המסך בעזרת מקשים

לצורך הזזת הריבוע על המסך עלינו תחילה לצייר ריבוע שחור על הריבוע הצבעוני ולאחר מכן לצייר את הריבוע החדש במיקום הבא.

Scan codes

כל לחיצה או שחרור של מקש במקלדת מייצרים קוד מיוחד – **scan code**.
המספר שנוצר נשמר בזיכרון מיוחד של המעבד שנקרא זיכרון (Input/Output) **I/O**

תהליך ההקלדה

- המשתמש לוחץ או משחרר מקש
- בזיכרון ה **I/O** נכתב ה **scan code** המתאים
- ה־PIC מקבל אינטרפט מהמקלדת.
- ה־ PIC שולח למעבד אינטרפט (9 int), שאומר למעבד שיש מידע בזיכרון ה I/O של המקלדת
- כתגובה לפסיקה, המעבד מריץ ISR שמטפל בפסיקה 9. ה־ISR "מעתיק" ה־scan code אל מקום מוגדר בזיכרון שנקרא Type Ahead Buffer

Type Ahead Buffer

מקום בזיכרון בו המעבד שומר את המידע מהמקלדת.
עבור כל תו המעבד שומר את ה Scan Code ואת התרגום שלו ל ASCII
כלומר לכל תו נשמרים 2 בתים
גודל הבאפר הוא 32 בתים. כלומר יש מקום ל 16 תווים.

לכן עלינו לדאוג לנקות את הבאפר.

יש לנו בתכנית כבר פסיקה הממתינה למקש, וכאשר מקישים על מקש (כל מקש) אנחנו יוצאים מתצוגה גרפית וחוזרים לתצוגת טקסט רגילה.

נוסף לפסיקה הוראה שתבדוק האם הקישו על מקש Esc אם הקישו על מקש Esc נצא למצב טקסט.

נוסיף לפני הפסיקה של ההמתנה למקש תווית:

WaitForKeyPress:

אם המשתמש הקיש על מקש הבאפר (זיכרון) המקלדת מיכל את הערך של המקש.

הפסיקה:

```
mov ah, 1h
int 16h
```

מדליקה את ה zero flag– אם יש תו מוכן לקריאה (zf = 1) , ומכבה אם אין תו מוכן (zf = 0).
אם יש תו מוכן, al ו-ah יקבלו את ערכי ה־ASCII וה־code scan של התו.
(הפסיקה אינה "מנקה" את התו מהבאפר של המקלדת).

האם מקש **Esc** נלחץ?

ניצור רצף פקודות שאם הקישו על מקש Esc נצא מהתכנית.

כדי לבדוק האם הקישו תו במקלדת נבדקו את דגל האפס -> zf = 1.
אם דגל האפס = 0 לא הוקש מקש במקלדת, נמשיך למתקין למקש.
אחרת:
נעביר לרגיסטר ah את ערך המקש.
נבדק אם הערך של **ah=1h** (נלחץ על **ESC** ליציאה מהלולאה)

```
; Wait for key press
WaitForKeyPress:
        mov  ah,1
        int  16h
        jz   WaitForKeyPress
        mov  ah,0
        int  16h
        cmp  ah,1h
        je   TheEnd
        jmp  WaitForKeyPress
```

```
TheEnd:
;Retern to text mod

        mov ax, 2
        int 10h
```

הריצו ובדקו.

שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

הזזת השחקן למעלה

ניצור פעולה של הזזת הריבוע למעלה: UpKeyPress

תחילה נמחק את הריבוע הקיים.

נצייר את הריבוע בצבע שחור.

נזיז את תחיל ציור הריבוע במינו שני פיקסלים על ציר ה – y[y_begin]

```
proc UpKeyPress
        mov [color], 0
        call DrawRectangle
        sub [y_begin], 2
        mov [color], 15
        call DrawRectangle
        ret
endp UpKeyPress
```

נחזור לתכנית הראשית ונוסיף בדיקה האם הקישו על מקש חץ למעלה -> **048h**

```
        ; Wait for key press
        WaitForKeyPress:
                mov  ah,1
                int  16h
                jz  WaitForKeyPress
                mov  ah,0
                int  16h
                cmp  ah,1h
                je   TheEnd
                cmp  ah, 048h
                je up
                jmp  WaitForKeyPress

        up:
                call UpKeyPress

                jmp  WaitForKeyPress

TheEnd:
```

לב התכנית:

ממתינים להקשת תו

אם הוקש Esc

יציאה מהתכנית

אם הוקש - חץ מעלה

קפיצה לתווית חץ מעלה

חזרה להמתנה למקש.

```
;Retern to text mod
        mov ax, 2
        int 10h
```

הרחיבו את התכנית, היספו תנאים ופרוצדורות למקשים של שאר החיצים.

050h – חץ מטה

04bh – חץ שמאלה

04dh - חץ ימינה

בדקו אחר כל הוספת תנאי ופרוצדורה והריצו את התכנית.
שמרו את התכנית בשם חדש והמשיכו לשלב הבא.

האם נוגע בצבע

במשק שיצרנו עם השחקן עולה על המסגרת הוא צבוע אותה בשחור ויכול לצאת ממסגרת המסך ולחזור בצדו השני. אנחנו רוצים למנוע מצב זה.
לשם כך לפני תזוזת השחקן נבדוק מהו הצבע של הפיקסלים במיקום הבא שאליו השקן צריך להגיע אליו. אם הצבע כמו צבע הרקע השחקן ינוע למיקום הבא אם הצבע הוא צבע כצבע המסגרת לא נאפשר את תנועת השחקן.

פסיקת **bios ah = 0dh** מאפשרת לנו לקרא ערך צבע של פיקסל מהמסך. הפסיקה מחזירה את ערך הצבע בנקודה שמבצעים עליה לרגיסטר al.

הפסיקה:

```
        mov bh, 0h
        mov cx, [x]
        mov dx, [y]
        mov ah, 0dh
        int 10h
```

ערך הצבע נשמר לרגיסטר al.
כתבו פעולה הבודקת את ערך הצבע בנקודה מסוימת על פני המסך ושומרת את הצבע למשתנה [check].
נוסיף משתנים [x] ו – [y] שיצביעו על מיקום הבדיקה של הצבע.

```
proc CheckColor
; get color of a single pixel --> return it to al
        mov bh, 0h
        mov cx, [x]
        mov dx, [y]
        mov ah, 0dh
        int 10h
        mov [check], al  ; save color of pixel
        ret
endp checkColor
```

לפני כל תזוזה של השחקן (הריבוע)
עלינו להעביר ל - [x] ול -[y] את המיקום הפיקסל שנרצה לבדוק את צבעו.
מיקום הפיקסל שנרצה לבדוק הוא המיקום הבא שאליו יזוז השחקן.
נוסיף לכל פרוצדורה את הפקודות שיעזרו לנו לבדוק את הצבע בצעד הבא של השחקן.

```
proc UpKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        sub [y], 2      ;  sub 2 for 2 pixel move
        call CheckColor
        cmp [check], 5
        je endUp
        mov [color], 0
        call DrawRectangle
        sub [y_begin], 2
        mov [color], 15
        call DrawRectangle
endUp:
        ret
endp UpKeyPress
```

נעביר את הערך של תחילת ציור הריבוע למשתנים החדשים.[y] [x]

אנו רוצים להזיז את השחקן מעלה על כן נפחית 2 פיקסלים מהמיקום בעמודה (משתנה y)

נזמן את הפרוצדורה של בדיקת הצבע

נשווה את הצבע שהתקבל לצבע המסגרת. אם אותו צבע נצא מהפרוצדורה .
אחרת נתקדם.

תקנו את הפרוצדורות של שאר הכיוונים. שימו לב באיזה מיקום לבדוק את הפיקסל.

הרחבות

שפרו את התכנית.
הוסיפו קווים לציור מבוך.
אפשר לשנות את השחקן (ציור דמות)
אפשר להוסיף ניקד.

ועד כיד הדמיון הטובה עליכם.

פרויקט Maze – עמליה אפל

בדף עבודה זה נתנסה בעבודה במצב גראפי. (בשונה למצב טקסט)
נצייר ביחד ריבוע, קווים אנכיים וקווים אופקיים.

מעבר למצב גרפי

המעבד יכול לעבוד בשני מצבים הראשון מצב טקסט (בו עבדנו עד היום) והשני מצב גרפי בו משתמשים בגרפיקה (ציור ושרטוט על המסך) כדי לעבור למוד גראפי נשתמש בפסיקת ה bios 10h

```
mov ax, 13h
int 10h
```

צרו תכנית העוברת למוד גראפי. בדקו בעזרת ה – ALT+F5 ב – turbo debugger. מה קורה למסך ה – DosBox במוד זה?

חזרה ממצב גרפי:

```
mov ax, 2
int 10h
```

הוסיפו בסוף התוכנית שלכם את ה interrupt החוזר למצב text.
הריצו את התכנית ב – Turbo Debugger, כדי לצפות במסך המשתמש הקישו **Alt + F5**.
חזרה למסך ה – TD הקישו **Esc**.

ציור במצב גרפי – ציור פיקסל (נקודה על המסך)

המסך מורכב מנקודות. כל נקודה נקראת פיקסל. במצב גרפי המסך מורכב מ 200 שורות ו 320 עמודות של פיקסלים. לכל פיקסל מיקום וצבע.
המיקום מורכב משורה ועמודה -> row, column

0, 0							319,0
0,199							319, 199

הצבע:
בזיכרון המחשב יש טבלה בגודל 256 תאים ובכל תא שלה מצוי מספר המייצג צבע.
כדי לצייר פיקסל ניתן מיקום וצבע.

לדוגמה: שורה 0
עמודה 0
צבע 4

יצבע בצבע אדום את הפיקסל בפינה השמאלית העליונה

0, 0							319,0
0,199							319, 199

האינטרפט (פסיקה) המצייר פיקסל למסך, הוא פסיקת **int 10h** - bois

הפרמטרים עבור אינטרפט זה:

ah = 0ch – קוד הפסיקה

bh = 0 – איפוס רגיסטר bh

cx – עמודה

dx – שורה

ax - צבע

כתבו תכנית המציירת פיקסל אדום (תא 4 בפלטת הצבעים) בשורה 20 עמודה 50.

```
xor bh, bh
mov cx, [x_coordinate]
mov dx, [y_coordinate]
mov ax, [color]
mov ah, 0ch
int 10h
```

בהרצת התכנית ב – TD ניתן לראות את הציור על מסך (לאחר הפעלת הפסיקה) ע"י הקשה על המקשים **Alt+F5**

או בחירה בתפריט Window ו - User screen

פרוצדורה

שנו את קטע הקוד לפרוצדורה העושה שימוש במשתנים לקואורדינטות x ו y של פיקסל ואת צבעו ומציירת אותו על המסך.

```
DATASEG
; -------------------------
; Your variables here
        x_coordinate dw 50          ;  in column
        y_coordinate dw 20          ; in line
        color dw 4

CODESEG
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Enter Graphic mod
        mov ax, 13h
        int 10h

; draw pixel
        xor bh, bh
        mov cx, [x_coordinate]  ;in  column
        mov dx, [y_coordinate]  ; in line
        mov ax, [color]
        mov ah, 0ch
        int 10h
; Enter text mod
        mov ax, 2
        int 10h
; -------------------------
```

כדי לשמר את מצב הרגיסטרים בכל פעולה של פרוצדורה נבצע פעולות push ו – pop לכל הרגיסטרים.
ניתן לבצע push וpop לכל הרגיסטרים יחד.
נוסיף בתחילת התכנית את הפקודה 186p (לפני ה – DATASEGMENT).

```
proc drawPixel
; draw pixel
        xor bh, bh
        mov cx, [x_coordinate]  ; place in column
        mov dx, [y_coordinate]  ; place in line
        mov ax, [color]
        mov ah, 0ch
        int 10h
        ret
        endp drawPixel
```

הוספת מקש

נוסיף לתכנית פסיקה אשר בהקשה על מקש היא יוצאת מהמצב הגרפי וחוזרת למצב טקסט.

הפסיקה הבודקת אם הוקש מקש במקלדת היא:

```
; Wait for key press
        mov ah,00h
        int 16h
```

הוסיפו פסיקה זו לאחר הקריאה של הפרוצדורה לציור פיקסל. ולפני הפקודה של חזרה למצב של טקסט.

במצב זה ניתן להריץ את התכנית ללא ה – Turbo Debugger.

הריצו את התכנית שוב ישירות

הריצו את הקובץ ב – tasm.
הריצו את הקובץ ב – tlink.
הריצו את הקובץ - רשמו את שמו בלבד ו – [Enter]

ליציאה מהתכנית הקישו על מקש (כל מקש).

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

```
start:
        mov ax, @data
        mov ds, ax
; -------------------------
; Enter Graphic mod
        mov ax, 13h
        int 10h

        call drawPixel

; Wait for key press
        mov ah,00h
        int 16h

;Retern text mod
        mov ax, 2
        int 10h
; -------------------------
exit:
        mov ax, 4c00h
        int 21h
END start
```

ציור קו אופקי

קו הוא אוסף של נקודות ..... על ציר ה – x

כתבו פרוצדורה המציירת קו על המסך (לרוחב). הפרוצדורה עושה שימוש בקואורדינטת תחילת השורה אורך הקו, ובצבע הפיקסל ומציירת את השורה.

- לצורך ביצוע השרטוט הקו עלינו ליצור משתנה חדש שבו נשמור את ארוך הקו, [len]

ניצור פרוצדורה חדשה המציירת קו למסך - **drawHorizontalLine**

- בתחילת הפרוצדורה נבצע פקודת push לכל הרגיסטרים כדי לשמור את ערכם למחסנית.
- בתחילת התכנית (לפני ה – DATASEGMENT) נוסיף את ההוראה 186p
  בפרוצדורה זו יש שימוש בלולאה ולכן חשוב מאוד לשמר את הרגיסטרים בכלל ואת רגיסטר cx בפרט.
- בתחילת הפרוצדורה נכתב את הפקודה **pusha** - פקודה העושה push לכל הרגיסטרים.
- ניצור לולאה שערכה הוא כאורך הקו בפיקסלים (נעביר לרגיסטר cx את הערך של משתנה len).
  - נזמן את הפעולה של ציור הפיקסל מתוך פרוצדורה זו.
  - נגדיל את ערכו של [x_coordinate]
- בסיום הפרוצדורה נבצע פעולת **popa** לערכים של כל הרגיסטרים.

```
IDEAL
MODEL small
STACK 100h
p186
DATASEG
        x_coordinate dw 50          ; place in line
        y_coordinate dw 20          ; place in column
        len  dw 20
        color dw 5
CODESEG
start:
        mov ax, @data
        mov ds, ax

; Enter Graphic mod
        mov ax, 13h
        int 10h

        call drawHorizontalLine
```

```
proc drawHorizontalLine
; draw line of pixels
        pusha
        mov cx, [len]
drawLine:
        call drawPixel
        inc [x_ coordinate]
        loop drawLine
        popa
        ret
        endp drawHorizontalLine
```

הריצו ובדקו.

שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור קו אנכי

כתבו פעולה המציירת קו אנכי.
קו אנכי הוא אוסף נקודות על ציר y
כתבו פרוצדורה חדשה drawVerticalLine

אל תשכחו לפני הקריאה לפרוצדורה להשים ערכים למשתנים של תחילת ציור הקו על ציר ה – x ועל ציר ה – y.

```
mov [x_coordinate], 30
mov [y_coordinate], 30
call drawVerticalLine
```

```
proc drawVerticalLine
; draw column of pixels
        pusha
        mov cx, [len]
drawColum:
        call drawPixel
        inc [y_ coordinate]
        loop drawColum
        popa
        ret
        endp drawVerticalLine
```

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור מסגרת למסך

גודל המסך הוא רוחב 320 פיקסלים וגובה 200 פיקסלים.
ציירו מסגרת למסך כל צבע שתבחרו.
השאירו שוליים צרים מכל צד של מסגרת ה – DoxBox.
חשבו כך שגודל המסגרת יהי זוגי ויתחיל בנקודה זוגית בנקודות ציר
ה – x וציר ה – y.

```
mov [x_coordinate], 6
mov [y_coordinate], 6
mov [len], 308
call drawHorizontalLine
mov [x_coordinate], 6
mov [y_coordinate], 6
mov [len], 188
call drawVerticalLine
mov [x_coordinate], 6
mov [y_coordinate], 194
mov [len], 308
call drawHorizontalLine
mov [x_coordinate], 314
mov [y_coordinate], 6
mov [len], 189
call drawVerticalLine
```

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

ציור ריבוע

ריבוע הוא אוסף של קווים

בציור הריבוע נשתמש בטכניקה שונה. זה יהיה השחקן שלנו שיזוז בתוך המסגרת. לכן חשוב לנו לשמור את נקודת ה – x וה – y של מיקום תחילת ציור הריבוע.
תזוזה של הריבוע על פני המסך היא ציור ריבוע שוחר על הריבוע הקיים – מחיקת הריבוא וצייר ריבוע חדש בנקודה אחרת על המסך.

ניצור פרוצדורה חדשה לציור הריבוע. לצורך ציור הריבוע נשתמש בשתי לולאות מותנות.
נקבע 4 משתנים חשים:
מיקום תחילת הריבוע על ציר x,
מיקום תחילת הריבוע על ציר y,
משתנה שיהיה מונה הלולאה של ציור הקו האופקי
משתנה שיהיה מונה הלולאה על הציר האנכי (ציור הקווים)

```
x_begin   dw 100          ; Starting point on line
y_begin   dw 30           ; Starting point on column
x_count   db 10           ; loop count line draw
Y_count   db 10           ; loop count column draw
```

פרוצדורה לציור ריבוע בשתי לולאות מותנות:

```
Proc   DrawRectangle
; Draw square at x_begin, y_begin position, size 10*10
        mov  [x_count], 10
        mov  [y_count], 10
        mov  ax, [x_begin]
        mov  [x_coordinate], ax
        mov  ax,[y_begin]
        mov  [y_coordinate], ax
line_loop:
        ; Print one dot on screen
        call  drawPixel

        inc  [x_coordinate]
        dec  [x_count]
        cmp  [x_count],  0
        jnz  line_loop

        mov ax, [x_begin]          ;reset lin
        mov  [x_coordinate],  ax
        mov  [x_count],10          ;reset co
        inc  [y_coordinate]
        dec  [y_count]
        cmp  [y_count],  0
        jnz  line_loop
        ret
endp DrawRectangle
```

קביעת גודל הריבוע והעברת הערכים של מיקם תחילת הריבוע ל משתנים: x_coordinate, y_coordinate

לאחר ציור נקודה אחת על המסך, מגדילים את הערך של מיקום הנקודה על ציר ה – X (מיקום על השורה) ומקטינים את מונה הלולאה של ציור הקו האופקי ב 1 – כאשר מונה הלולאה מגיע ל 0 – יוצאים ממנה.

לאחר ציור קו אחד למסך, מאפסים את תחילת ציור הקו על ציר איקס.
מאפסים את מונה הלולאה המותנית של ציור קו אופקי.
מגדלים את מיקום ציור הנקודה על ציר הy – ציר אופקי (עמודה)
מקטינים את מונה הלולאה של ציור הקווים (ציר אופקי)
כאשר מונה הלולאה של ציור הקווים מגיע לאפס, יוצאים משתי הלולאות.

הריצו ובדקו.
שמרו את הקובץ, לשם חדש והמשיכו בקובץ החדש.

הזזת השחקן על פני המסך בעזרת מקשים

לצורך הזזת הריבוע על המסך עלינו תחילה לצייר ריבוע שחור על הריבוע הצבעוני ולאחר מכן לצייר את הריבוע החדש במיקום הבא.

Scan codes

כל לחיצה או שחרור של מקש במקלדת מייצרים קוד מיוחד – **scan code**.
המספר שנוצר נשמר בזיכרון מיוחד של המעבד שנקרא זיכרון (Input/Output) **I/O**

תהליך ההקלדה

- המשתמש לוחץ או משחרר מקש
- בזיכרון ה **I/O** נכתב ה **scan code** המתאים
- ה־PIC מקבל אינטרפט מהמקלדת.
- ה־ PIC שולח למעבד אינטרפט (9 int), שאומר למעבד שיש מידע בזיכרון ה I/O של המקלדת
- כתגובה לפסיקה, המעבד מריץ ISR שמטפל בפסיקה 9. ה־ISR "מעתיק" ה־scan code אל מקום מוגדר בזיכרון שנקא Type Ahead Buffer

Type Ahead Buffer

מקום בזיכרון בו המעבד שומר את המידע מהמקלדת.
עבור כל תו המעבד שומר את ה Scan Code ואת התרגום שלו ל ASCII
כלומר לכל תו נשמרים 2 בתים
גודל הבאפר הוא 32 בתים. כלומר יש מקום ל 16 תווים.
לכן עלינו לדאוג לנקות את הבאפר

יש לנו בתכנית כבר פסיקה הממתינה למקש, וכאשר מקישים על מקש (כל מקש) אנחנו יוצאים מתצוגה גרפית וחוזרים לתצוגת טקסט רגילה.

נוסף לפסיקה הוראה שתבדוק האם הקישו על מקש Esc אם הקישו על מקש Esc נצא למצב טקסט.

נוסיף לפני הפסיקה של ההמתנה למקש תווית:

WaitForKeyPress:

אם המשתמש הקיש על מקש הבאפר (זיכרון) המקלדת מיכל את הערך של המקש.

הפסיקה:

```
mov ah, 1h
int 16h
```

מדליקה את ה zero flag– אם יש תו מוכן לקריאה (zf = 1) , ומכבה אם אין תו מוכן (zf = 0).
אם יש תו מוכן, al ו-ah יקבלו את ערכי ה־ASCII וה־code scan של התו.
(הפסיקה אינה "מנקה" את התו מהבאפר של המקלדת).

האם מקש **Esc** נלחץ?

ניצור רצף פקודות שאם הקישו על מקש Esc נצא מהתכנית.
כדי לבדוק האם הקישו תו במקלדת נבדקו את דגל האפס -> zf = 1.
אם דגל האפס = 0 לא הוקש מקש במקלדת, נמשיך למתקין למקש.
אחרת:
נעביר לרגיסטר ah את ערך המקש.
נבדק אם הערך של **ah = 1h** (נלחץ **ESC** כדי לצאת מלולאה)

```
; Wait for key press
WaitForKeyPress:
        mov  ah,1
        int  16h
        jz   WaitForKeyPress
        mov  ah,0
        int  16h
        cmp  ah,1h
        je   TheEnd
        jmp  WaitForKeyPress

TheEnd:
;Retern to text mod

        mov ax, 2
        int 10h
```

בדקו והריצו:
שמרו את הקובץ בשם חדש והמשיכו בקובץ החדש.

הזזת השחקן למעלה

ניצור פעולה של הזזת הריבוע למעלה: UpKeyPress
תחילה נמחק את הריבוע הקיים.
נצייר את הריבוע בצבע שחור.
נזיז את תחיל ציור הריבוע במינו שני פיקסלים על ציר ה – y[y_begin]

```
proc UpKeyPress
        mov [color], 0
        call DrawRectangle
```

```
        sub [y_begin], 2
        mov [color], 15
        call DrawRectangle
        ret
endp UpKeyPress
```

נחזור לתכנית הראשית ונוסיף בדיקה האם הקישו על מקש חץ למעלה -> **048h**

```
        ; Wait for key press
        WaitForKeyPress:
                mov  ah,1
                int  16h
                jz  WaitForKeyPress
                mov  ah,0
                int  16h
                cmp  ah,1h
                je   TheEnd
                cmp  ah, 048h
                je up
                jmp  WaitForKeyPress

        up:
                call UpKeyPress

                jmp  WaitForKeyPress

TheEnd:

;Retern to text mod
        mov ax, 2
        int 10h
```

לב התכנית:
ממתינים להקשת תו
אם הוקש Esc
יציאה מהתכנית
אם הוקש - חץ מעלה
קפיצה לתווית חץ מעלה
חזרה להמתנה למקש.

הרחיבו את התכנית, היספו תנאים ופרוצדורות למקשים של שאר החיצים.

050h – חץ מטה
04bh – חץ שמאלה
04dh - חץ ימינה

בדקו אחר כל הוספת תנאי ופרוצדורה והריצו את התכנית.
שמרו את התכנית בשם חדש והמשיכו לשלב הבא.

האם נוגע בצבע

במשק שיצרנו עם השחקן עולה על המסגרת הוא צבוע אותה בשחור ויכול לצאת ממסגרת המסך ולחזור בצדו השני. אנחנו רוצים למנוע מצב זה.
לשם כך לפני תזוזת השחקן נבדוק מהו הצבע של הפיקסלים במיקום הבא שאליו השקן צריך להגיע אליו. אם הצבע כמו צבע הרקע השחקן ינוע למיקום הבא אם הצבע הוא צבע כצבע המסגרת לא נאפשר את תנועת השחקן.

פסיקת **bios ah = 0dh** מאפשרת לנו לקרא ערך צבע של פיקסל מהמסך. הפסיקה מחזירה את ערך הצבע בנקודה שמבצעים עליה לרגיסטר al.

הפסיקה:

```
mov bh, 0h
mov cx, [x]
mov dx, [y]
mov ah, 0dh
int 10h
```

ערך הצבע נשמר לרגיסטר al.
כתבו פעולה הבודקת את ערך הצבע בנקודה מסוימת על פני המסך ושומרת את הצבע למשתנה [check].
נוסיף משתנים [x] ו – [y] שיצביעו על מיקום הבדיקה של הצבע.

```
proc CheckColor
; get color of a single pixel --> return it to al
        mov bh, 0h
        mov cx, [x]
        mov dx, [y]
        mov ah, 0dh
        int 10h
        mov [check], al  ; save color of pixel
        ret
endp checkColor
```

לפני כל תזוזה של השחקן (הריבוע)
עלינו להעביר ל - [x] ול -[y] את המיקום הפיקסל שנרצה לבדוק את צבעו.
מיקום הפיקסל שנרצה לבדוק הוא המיקום הבא שאליו יזוז השחקן.
נוסיף לכל פרוצדורה את הפקודות שיעזרו לנו לבדוק את הצבע בצעד הבא של השחקן.

```
proc UpKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        sub [y], 2     ;  sub 2 for 2 pixel move
        call CheckColor
        cmp [check], 5
        je endUp
        mov [color], 0
        call DrawRectangle
        sub [y_begin], 2
        mov [color], 15
        call DrawRectangle
endUp:
        ret
endp UpKeyPress
```

נעביר את הערך של תחילת ציור הריבוע למשתנים החדשים [x] [y]

אנו רוצים להזיז את השחקן מעלה על כן נפחית 2 פיקסלים מהמיקום בעמודה (משתנה y)

נזמן את הפרוצדורה של בדיקת הצבע

נשווה את הצבע שהתקבל לצבע המסגרת.
אם אותו הצבע נצא מהפרוצדורה .
אחרת נתקדם.

תקנו את הפרוצדורות של שאר הכיוונים. שימו לב באיזה מיקום לבדוק את הפיקסל.

```
proc DownKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        add [y], 12     ;  add 2 for 2 pixel move & 10 for rec size
        call CheckColor
        cmp [check], 5
        je endDown
        mov [color], 0
        call DrawRectangle
        add [y_begin], 2
        mov [color], 15
        call DrawRectangle
endDown:
        ret
endp DownKeyPress
```

```
proc LeftKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        sub [x], 2  ;  sub 2 for 2 pixel move
        call CheckColor
        cmp [check], 5
        je endLeft
        mov [color], 0
        call DrawRectangle
        sub [x_begin], 2
        mov [color], 15
        call DrawRectangle
endLeft:
        ret
endp LeftKeyPress
```

```
proc RightKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        add [x], 12  ;  add 2 for 2 pixel
move & 10 for rec size
        call CheckColor
        cmp [check], 5
        je endRight
        mov [color], 0
        call DrawRectangle
        add [x_begin], 2
        mov [color], 15
        call DrawRectangle
endRight:
        ret
endp RightKeyPress
```

קוד התכנית:

פתיח + משתנים

```
IDEAL
MODEL small
STACK 100h
p186
; תרגיל מודרך ציור מבוך
; ציור מסגרת למסך, ציור שחקן - ריבוע
; הזזת הריבוע
; נוגע בצבע

DATASEG
; --------------------------
; Your variables here
        x_coordinate dw ?   ; place in line
        y_coordinate dw ?   ; place in column
        len dw ?
        color dw 5
        x_begin     dw 100          ; Starting point on line
        y_begin     dw 30       ; Starting point on column
        x_count     db 10       ; loop count line draw
        Y_count     db 10       ;loop count column draw
        x  dw ?     ; x coordinate for check if tach color on next step
        y  dw ?             ; y coordinate for check if tach color on next step
        check db ?  ; save cheac coolor, color
```

פעולות

```
CODESEG

proc drawPixel
        pusha
        xor bh, bh
        mov cx, [x_coordinate]  ; place in line
        mov dx, [y_coordinate]  ; place in column
        mov ax, [color]
        mov ah, 0ch
        int 10h
        popa
        ret
```

```
        endp drawPixel
proc drawHorizontalLine
; draw line of pixels
        pusha
        mov cx, [len]
drawLine:
        call drawPixel
        inc [x_coordinate]
        loop drawLine
        popa
        ret
        endp drawHorizontalLine

proc drawVerticalLine
; draw column of pixels
        pusha
        mov cx, [len]
drawColum:
        call drawPixel
        inc [y_coordinate]
        loop drawColum
        popa
        ret
        endp drawVerticalLine
```

```
Proc  DrawRectangle
; Draw square at x_begin, y_begin position,
size 10*10
        mov    [x_count],10
        mov [Y_count],10
        mov ax,[x_begin]
        mov [x_coordinate], ax
        mov ax,[y_begin]
        mov [y_coordinate], ax

line_loop:
        ; Print one dot on screen
        call drawPixel

        inc [x_coordinate]
        dec [x_count]
        cmp [x_count], 0
        jnz line_loop

        mov ax,[x_begin]            ;reset line
counters
        mov    [x_coordinate], ax
        mov [x_count],10            ;reset
column counters
        inc [y_coordinate]
        dec [y_count]
        cmp [y_count], 0
        jnz line_loop

        ret
endp DrawRectangle
```

```
proc UpKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        sub [y], 2        ;  sub 2 for 2 pixel move
        call CheckColor
        cmp [check], 5
        je endUp
        mov [color], 0
        call DrawRectangle
        sub [y_begin], 2
        mov [color], 15
        call DrawRectangle
endUp:
        ret
endp UpKeyPress

proc DownKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        add [y], 12      ;  add 2 for 2 pixel move & 10 for rec size
        call CheckColor
        cmp [check], 5
        je endDown
        mov [color], 0
        call DrawRectangle
        add [y_begin], 2
        mov [color], 15
        call DrawRectangle
endDown:
        ret
endp DownKeyPress
```

```
proc LeftKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        sub [x], 2        ;  sub 2 for 2 pixel move
        call CheckColor
        cmp [check], 5
        je endLeft
        mov [color], 0
        call DrawRectangle
        sub [x_begin], 2
        mov [color], 15
        call DrawRectangle
endLeft:
        ret
endp LeftKeyPress

proc RightKeyPress
        mov ax, [y_begin]
        mov [y], ax
        mov ax, [x_begin]
        mov [x], ax
        add [x], 12      ;  add 2 for 2 pixel move & 10 for rec size
        call CheckColor
        cmp [check], 5
        je endRight
        mov [color], 0
        call DrawRectangle
        add [x_begin], 2
        mov [color], 15
        call DrawRectangle
endRight:
        ret
endp RightKeyPress
```

```
proc CheckColor
; get color of a single pixel --> return it to al
        mov bh, 0h
        mov cx, [x]
        mov dx, [y]
        mov ah, 0dh
        int 10h
        mov [check], al  ; save color of pixel
        ret
endp checkColor
```

תכנית ראשית

```
start:
        mov ax, @data
        mov ds, ax
; --------------------------
; Enter Graphic mod 320 * 200 pixels, 256 colors
        mov ax, 13h
        int 10h
```

ציור מסגרת ושחקן

```
        mov [x_coordinate], 6
        mov [y_coordinate], 6
        mov [len], 308
        call drawHorizontalLine
        mov [x_coordinate], 6
        mov [y_coordinate], 6
        mov [len], 188
        call drawVerticalLine
        mov [x_coordinate], 6
        mov [y_coordinate], 194
        mov [len], 308
        call drawHorizontalLine
        mov [x_coordinate], 314
        mov [y_coordinate], 6
        mov [len], 189
        call drawVerticalLine

        mov [color], 15
        call DrawRectangle
```

לב המשחק, הקשת מקשים

```
; Wait for key press
WaitForKeyPress:
        mov  ah,1      ; check for keystroke in the keyboard buffer
        int  16h
        jz        WaitForKeyPress   ; if keystroke is not available
        mov  ah, 0  ; get keystroke from keyboard  -->  ah = BIOS scan code
        int  16h
        cmp  ah, 1h    ; Esc key
        je        TheEnd
        cmp     ah, 048h  ; up key
        je up
        cmp     ah, 050h  ; down key
        je down
        cmp     ah, 04bh  ; left key
        je left
        cmp     ah, 04dh  ; right key
        je right
        jmp  WaitForKeyPress

up:
        call UpKeyPress
        jmp WaitForKeyPress

down:
        call DownKeyPress
        jmp  WaitForKeyPress

left:
        call LeftKeyPress
        jmp  WaitForKeyPress

right:
        call RightKeyPress
        jmp  WaitForKeyPress

TheEnd:

;Retern to text mod
        mov ax, 2
        int 10h
; --------------------------
```

```
exit:
        mov ax, 4c00h
        int 21h
END start
```

פרויקט Pacman – סוניה שמאי

הערות:

- עקוב אחרי ההנחיות במדויק. ההנחיות כתובות מהקל אל הכבד ע"פ נושאי הלימוד בכיתה.
- במקומות בהם כתוב "שנה את התוכנית" וודא שאתה שומר עותק במידה וקלקלת את התוכנית הקודמת.

להלן ההנחיות:

1. כתוב תוכנית prog1.asm. התוכנית מציגה נקודה צהובה על המסך.
2. כתוב תוכנית prog2.asm המבוסס על שלב 1. התוכנית מציגה ריבוע צהוב על המסך. (שים לב, בתוכנית זו, הקוד להצגת נקודה צריך להיות כתוב כפונקציה, איזה עוד פונקציות תכתוב?).
3. כתוב תוכנית prog3.asm המבוסס על שלב 2. התוכנית מציגה ריבוע צהוב על המסך לחיצה על מקש a מעלימה אותו, לחיצה על מקש b מציגה אותו שוב. (שים לב, בתוכנית זו, הקוד להצגת ריבוע צריך להיות כתוב כפונקציה).
4. שנה את התוכנית prog3.asm כך שתציג ריבוע צהוב, ניתן להזיז את הריבוע ימינה, שמאלה, למעלה או למטה בהתאם להקשה.
   - הזזה פרושה הצגת ריבוע שחור במקום הריבוע הצהוב ואח"כ הצגת ריבוע צהוב במקום החדש.
   - המלצה, למען נוחות העבודה, אפשר גם הקשה על Esc ליציאה מהתוכנית.
5. כתוב תוכנית prog4.asm המגדירה את המטריצה הבאה:

```
ylclPac     dw    0001111111111000B
            dw    0011111111111100B
            dw    0111111111111110B
            dw    1111001111111111B
            dw    1111001111111111B
            dw    1111111111111111B
            dw    1111111111111111B
            dw    1111111111111111B
            dw    1111111111111111B
            dw    1111111111111111B
            dw    0000001111111111B
            dw    1111111111111111B
            dw    1111111111111111B
            dw    0111111111111110B
            dw    0011111111111100B
            dw    0001111111111000B
```

התוכנית תציג את התמונה על המסך, כל סיבית מייצגת נקודה. סיבית 1 מייצגת נקודה צהובה, סיבית 0 מייצגת נקודה שחורה.

- העזר בפעולות למיסוך סיביות לפי בחירתך: AND/TEST/ROR/ROL/SHR
- כתוב פונקציה בשם Pac המקבלת כתובת של מטריצה באוגר SI ומדפיסה את הדמות.
- משמעות השם היא קיצור של השם: yellowClosePacman כלומר פאקמן צהוב עם פה סגור.

6. מזג את התוכנית prog4.asm עם התוכנית prog3.asm וכתוב תוכנית חדשה prog5.asm. התוכנית תציג את דמות הפאקמן. הקשה על מקש a תעלים אותו, הקשה על מקש b תציג את הדמות שוב.
   - לצורך כך הגדר מטריצה של 16X16 אפסים (כאמור, 0 מייצג שחור) באופן הבא:
   blPac dw 16 dup(0)
   - משמעות השם היא קיצור של השם: blackPacman
7. מעכשיו נעבוד על אותו הקובץ prog5.asm ונוסיף לו עוד ועוד אופציות. לכן דאג לגבות כל פעם את השלב הקודם. שמור את prog5.asm בשם packman.asm ובסיום כל שלב, לאחר שווידאת שאכן הכל עובד, שמור אותו בשם backup.asm והמשך לעבוד על packman.asm.
8. שנה את התוכנית packman.asm כך שדמות הפאקמן תזוז ימינה/שמאלה/למעלה/למטה בהתאם להקשה על מקשי החצים בדומה למה שעית בסעיף 4.

פרויקט כדורים נופלים – חגית כהן

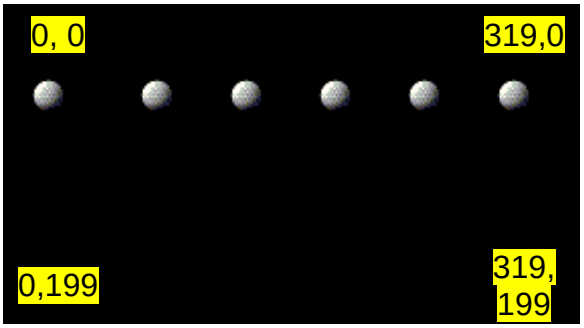
שלב 1 – הסבר המשחק

שני כדורים נופלים מהחלק העליון של המסך. מטרת המשחק הנה להכניס את הכדורים אל הסל. את הסל אנו יכולים להזיז ימינה ושמאלה. המשחק נגמר כאשר פספסנו כדור.

שלב 2 – הכנת תמונות לכדור, סל, הודעת משחק נגמר וריבוע שחור.

ניצור תמונה של עיגול לבן ברקע שחור (כרקע המסך). נשאיר מרווח בחלק העליון של התמונה כך שכאשר הכדור נופל, לא נצטרך לצבוע את המקום הקודם שלו (מעליו) – הכדור יורד למטה בלבד.

<u>מאפייני המסך:</u>



שלב 3 – טעינת תמונות:

בשלב זה, נטען שתי תמונות כדורים לחלק העליון של המסך ותמונה של סל בחלק התחתון של המסך.

לשם כך, נשתמש בקוד טעינת תמונה אותו כבר כתבנו בתרגיל קודם. קוד זה מצורף בסוף המסמך.

להלן הגדרת משתנים ושמות התמונות:

```
Pic1 db 'ball.bmp',0
Pic2 db 'ball.bmp',0
Pic3 db 'ballb.bmp',0
Busket db 'busket.bmp',0
GameOver db 'Gameover.bmp',0
clear db 'clear.bmp',0
```

```
proc pic3P ←פעולה הטוענת תמונה
        mov dx,offset Pic3
        call OpenShowBmp
        ret
endp pic3P
```

שלב 4 – מאפיינים למיקום התמונות על המסך:

עלינו לייצר 2 משתנים לכל תמונה – מיקום פינה שמאלית ציר X וציר Y.

להלן הגדרת משתנים למיקום התמונות:

```
x dw ?                  ; Ball #1 X value
y dw ?          ; Ball #1 Y value
x1 dw ?         ; Ball #2 X value
y1 dw ?         ; Ball #2 y value
x2 dw ?         ; Ball #3 X value
y2 dw ?         ; Ball #3 y value
xb dw ?         ; Busket x value
yb dw 190       ; Busket Y value
BmpColSize dw 30  ; Balls size
BmpRowSize dw 30          ; Balls size
BmpLeft dw ?              ; Balls location
BmpTop dw ?               ; Balls location
Stop db 0
```

שלב 5 –יצירת תנועה של כדורים – מהחלק העליון לחלק התחתון:

לשם יצירת תנועה של תמונה על המסך, עלינו להשתמש בפסיקה של שעון. נגדיר פעולה בה נשנה את ציר Y של הכדורים ונציג את התמונה מחדש כל שניה. שינוי ציר ה- Y של התמונה תייצר תנועה מעלה / מטה ואיללו שינוי ציר X של התמונה תייצר תנועה ימינה / שמאלה.

נשתמש בפסיקת שעון – השתמשנו בשיעור קודם.

```
start:
        mov ax, @data
        mov ds, ax
        call SetGraphic ← מעבר למצב גרפי
lp1:
        mov [x],10←איתחול מיקומי כדור
        mov [y],50
        mov [x1],60
        mov [y1],80
        call showBus← הצגת תמונת סל

lp:     call countsec← קריאה לפעולת המתנה שנייה
        mov [BmpColSize], 30
        mov [BmpRowSize], 24
        mov ax,[x]
        mov [BmpLeft], ax
        mov ax,[y]
        mov [BmpTop] ,ax
        call pic1P ← הצגת תמונת כדור 1 לאחר עדכון מיקומו
        mov ax,[x1]
        mov [BmpLeft], ax
        mov ax,[y1]
        mov [BmpTop] ,ax
        call pic2P← הצגת תמונת כדור 2 לאחר עדכון מיקומו
        loop lp ← ביצוע חוזר של הפעולה
```

בקוד הבא ניתן לראות כיצד מעדכנים את ערכי הגובה של הכדורים.
לאחר עדכון ערך זה, קוראים שוב לפעולה המציגה את התמונות.

שלב 6 – הזזת תמונת הסל ימינה / שמאלה בלחיצת כפתור:

עלינו להאזין ללחיצת כפתור ימינה / שמאלה תוך כדי הזזת התמונות. כלומר, אנו רוצים ליצור מצב בו נזיז את התמונות של שניה וגם נאזין ללחיצת כפתור בזמן זה.

כאמור, עדכון מיקום התמונה מתבצע לאחר שנייה. כלומר, כל שנייה הפעולה countsec נראת מחדש (ראה קוד שלב 5). אנו נאזין ללחיצת כפתור בפעולה countsec עצמה. כלומר, במהלך שנייה נבדוק אם נלחץ כפתור. אם כן, נזיז את מיקום הסל ימינה או שמאלה בהתאם.

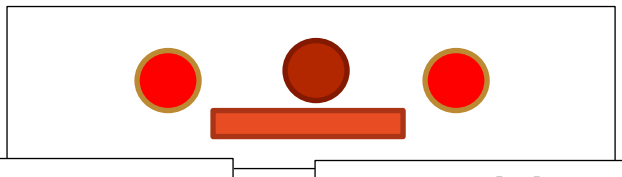
נסתכל על הפעולה countsec:

```
proc countsec
        pusha
        push es
        mov dx, 40h
        mov es, dx
        mov dx, [Clock]
FirstTick:
        cmp dx, [Clock]
        je FirstTick
        ; count 0.5 sec
        mov cx, 10 ; 182x0.055sec =
~10sec
DelayLoop:
        mov dx, [Clock]
Tick:
        in al, 64h ; check Keyboard
        cmp al, 10b
        je chTic
        in al, 60h
        mov ah,0
```

```
        cmp al,4Bh ; check if left arrow
        je subX
        cmp al,4dh ; check if rigth arrow
        jne chTic
        add [xb],4
        call showBus← הצג תמונה של סל
        jmp chTic
subX:
        sub [xb],4
        call showBus←הצג תמונה של סל
chTic:
        cmp dx, [Clock]
        je Tick

        ;loop DelayLoop
        pop es
        popa
ret
endp countsec
```

שלב 7 – בדיקת משחק נגמר:

בשלב זה, שני כדורים נעים מלמעלה למטה והסל זז ימינה או שמאלה לפי הכפתור שנלחץ.
השלב האחרון שנותר הוא לבדוק האם המשחק נגמר. את הבדיקה נבצע כל פעם שנעדכן את מיקום הכדור.
משחק נגמר כאשר הכדור נמצא בגובה של הסל אך הסל במקום X אחר. כלומר:
X כדור > X סל ופינה שמאלית או X כדור < X סל פינה ימנית.
בתמונה: הכדורים האדומים מציגים מצב של פסילה. הכדור הירוק מראה מצב תקין.



```
proc checkG
        mov ax,[yb]
        ;add ax,30       ; ball lower y
        cmp [y],ax
        jg ch1
        cmp [y1],ax
        jl okG← כדור לא בגובה הסל
        ; ball 2 in busket Y
        mov ax,[xb]
        ;add ax,30       ; ball lower y
        cmp [x1],ax
        jl noG
        add ax,60
        cmp [x1],ax
        jg noG← לא בתחום הסל
        sub ax,60
        mov [BmpLeft],ax
        mov ax, [y1]
        mov [BmpTop],ax
        call pic3P
        mov [y1],0
        sub [x1],20
        jmp okG
ch1:
        ; ball 1 in busket Y
        mov ax,[xb]
        cmp [x],ax
        jl noG
        add ax,60
        cmp [x],ax
        jg noG
```

```
        mov [y1],0
        sub [x1],20
        jmp okG
ch1:
        ; ball 1 in busket Y
        mov ax,[xb]
        cmp [x],ax
        jl noG
        add ax,60
        cmp [x],ax
        jg noG
        sub ax,20
        mov [BmpLeft],ax
        mov ax, [y]
        mov [BmpTop],ax
        call pic3P
        mov [y],0
        add [x],20
        jmp okG
noG:
        mov ah,1
        jmp ex
okG:
        mov ah,0
ex:     ret
endp checkG
```

```
okG: ←מצב תקין
        mov ah,0
ex:     ret
endp checkG
```

שלב 8 – תוספות:

בשלב זה המשחק פעיל. כעת ניתן לשדרג את המשחק כראות עיניכם. דוגמא:

1. הוסף ניקוד – כל תפיסת כדור.
2. הוספת חיים – הוסף כדור לב המציין חיים נוספים.
3. הוספת פצצה – תפיסת פצצה מסיימת משחק מידית.
4. הוסף צליל תפיס / צליל פסילה.

ניתן לשדרג את המשחק על סמך כל הנלמד במהלך השנה.

הקוד המלא:

```
jumps
P186
IDEAL
MODEL small
STACK 0f500h
MAX_BMP_WIDTH = 320
MAX_BMP_HEIGHT = 200

SMALL_BMP_HEIGHT = 40
SMALL_BMP_WIDTH = 40

DATASEG
        Clock equ es:6Ch
        Pic1 db 'ball.bmp',0
        Pic2 db 'ball.bmp',0
        Pic3 db 'ballb.bmp',0
        Busket db 'busket.bmp',0
        GameOver db 'Gameover.bmp',0
        clear db 'clear.bmp',0
        OneBmpLine          db MAX_BMP_WIDTH dup (0)  ; One Color line read
buffer
   ScreenLineMax   db MAX_BMP_WIDTH dup (0)  ; One Color line read buffer
        ;BMP File data
        saveKey db 0
        FileHandle    dw ?
        Pic      db 0
        Header            db 54 dup(0)
        Palette          db 400h dup (0)

        BmpFileErrorMsg    db 'Error At Opening Bmp File .', 0dh, 0ah,'$'
        ErrorFile          db 0
   BB db "BB..","$"

        x dw ?          ; Ball #1 X value
        y dw ?          ; Ball #1 Y value
        x1 dw ?                 ; Ball #2 X value
        y1 dw ?         ; Ball #2 y value
        x2 dw ?                 ; Ball #3 X value
        y2 dw ?         ; Ball #3 y value
        xb dw ?                 ; Busket x value
        yb dw 190       ; Busket Y value
        BmpColSize dw 30  ; Balls size
```

```
        BmpRowSize dw 30        ; Balls size
        BmpLeft dw ?            ; Balls location
        BmpTop dw ?                     ; Balls location
        Stop db 0
CODESEG


start:
        mov ax, @data
        mov ds, ax

        call SetGraphic
lp1:
        mov [x],10
        mov [y],50
        mov [x1],60
        mov [y1],80
        call showBus

lp:     call countsec
        mov [BmpColSize], 30
        mov [BmpRowSize], 24
        mov ax,[x]
        mov [BmpLeft], ax
        mov ax,[y]
        mov [BmpTop] ,ax
        call pic1P
        mov ax,[x1]
        mov [BmpLeft], ax
        mov ax,[y1]
        mov [BmpTop] ,ax
        call pic2P
        add [y],5
        add [y1],5
        call showBus
        call checkG
        cmp ah,1
        je game

        loop lp
game:
        call showGame
```

```
        jmp lp1
exit:
        mov dx, offset BB
        mov ah,9
        ;int 21h
        mov ah,0
        int 16h
        mov ax,2
        int 10h
        mov ax, 4c00h
        int 21h
;========================
;==========================
;===== Procedures  Area ===
;==========================
proc checkG
        mov ax,[yb]
        ;add ax,30               ; ball lower y
        cmp [y],ax
        jg ch1
        cmp [y1],ax
        jl okG
        ; ball 2 in busket Y
        mov ax,[xb]
        ;add ax,30               ; ball lower y
        cmp [x1],ax
        jl noG
        add ax,60
        cmp [x1],ax
        jg noG
        sub ax,60
        mov [BmpLeft],ax
        mov ax, [y1]
        mov [BmpTop],ax
        call pic3P
        mov [y1],0
        sub [x1],20
        jmp okG
ch1:
        ; ball 1 in busket Y
        mov ax,[xb]
        cmp [x],ax
        jl noG
```

```
        add ax,60
        cmp [x],ax
        jg noG
        sub ax,20
        mov [BmpLeft],ax
        mov ax, [y]
        mov [BmpTop],ax
        call pic3P
        mov [y],0
        add [x],20
        jmp okG
noG:
        mov ah,1
        jmp ex
okG:
        mov ah,0
ex:     ret
endp checkG

proc OpenShowBmp near
        push cx
        push bx
        call OpenBmpFile
        cmp [ErrorFile],1
        je @@ExitProc
        call ReadBmpHeader
        ; from  here assume bx is global param with file handle.
        call ReadBmpPalette
        call CopyBmpPalette
        call ShowBMP
        call CloseBmpFile
@@ExitProc:
        pop bx
        pop cx
        ret
endp OpenShowBmp
; input dx filename to open
proc OpenBmpFile  near
        mov ah, 3Dh
        xor al, al
        int 21h
        jc @@ErrorAtOpen
        mov [FileHandle], ax
```

```
        jmp @@ExitProc
@@ErrorAtOpen:
        mov [ErrorFile],1
@@ExitProc:
        ret
endp OpenBmpFile

proc CloseBmpFile near
        mov ah,3Eh
        mov bx, [FileHandle]
        int 21h
        ret
endp CloseBmpFile


; Read 54 bytes the Header
proc ReadBmpHeader      near
        push cx
        push dx

        mov ah,3fh
        mov bx, [FileHandle]
        mov cx,54
        mov dx,offset Header
        int 21h

        pop dx
        pop cx
        ret
endp ReadBmpHeader



proc ReadBmpPalette near ; Read BMP file color palette, 256 colors * 4 bytes (400h)
                         ; 4 bytes for each color BGR + null)

        push cx
        push dx

        mov ah,3fh
        mov cx,400h
        mov dx,offset Palette
        int 21h
```

```
        pop dx
        pop cx

        ret
endp ReadBmpPalette


; Will move out to screen memory the colors
; video ports are 3C8h for number of first color
; and 3C9h for all rest
proc CopyBmpPalette                near

        push cx
        push dx

        mov si,offset Palette
        mov cx,256
        mov dx,3C8h
        mov al,0  ; black first
        out dx,al ;3C8h
        inc dx   ;3C9h
CopyNextColor:
        mov al,[si+2]           ; Red
        shr al,2                        ; divide by 4 Max (cos max is 63 and we have here
max 255 ) (loosing color resolution).
        out dx,al
        mov al,[si+1]           ; Green.
        shr al,2
        out dx,al
        mov al,[si]             ; Blue.
        shr al,2
        out dx,al
        add si,4                        ; Point to next color.  (4 bytes for each color BGR +
null)

        loop CopyNextColor

        pop dx
        pop cx

        ret
endp CopyBmpPalette
```

```
proc ShowBMP
; BMP graphics are saved upside-down.
; Read the graphic line by line (BmpRowSize lines in VGA format),
; displaying the lines from bottom to top.
        push cx

        mov ax, 0A000h
        mov es, ax

        mov cx,[BmpRowSize]

        mov ax,[BmpColSize] ; row size must dived by 4 so if it less we must calculate
the extra padding bytes
        xor dx,dx
        mov si,4
        div si
        mov bp,dx

        mov dx,[BmpLeft]

@@NextLine:
        push cx
        push dx

        mov di,cx  ; Current Row at the small bmp (each time -1)
        add di,[BmpTop] ; add the Y on entire screen


        ; next 5 lines  di will be  = cx*320 + dx , point to the correct screen line
        mov cx,di
        shl cx,6
        shl di,8
        add di,cx
        add di,dx

        ; small Read one line
        mov ah,3fh
        mov cx,[BmpColSize]
        add cx,bp  ; extra  bytes to each row must be divided by 4
        mov dx,offset ScreenLineMax
        int 21h
```

```
        ; Copy one line into video memory
        cld ; Clear direction flag, for movsb
        mov cx,[BmpColSize]
        mov si,offset ScreenLineMax
        rep movsb ; Copy line to the screen

        pop dx
        pop cx

        loop @@NextLine

        pop cx
        ret
endp ShowBMP

proc  SetGraphic
        mov ax,13h   ; 320 X 200
        int 10h
        ret
endp SetGraphic
proc countsec
        pusha
        push es
        mov dx, 40h
        mov es, dx
        mov dx, [Clock]
FirstTick:
        cmp dx, [Clock]
        je FirstTick
        ; count 0.5 sec
        mov cx, 10 ; 182x0.055sec = ~10sec
DelayLoop:
        mov dx, [Clock]
Tick:
        in al, 64h ; check Keyboard
        cmp al, 10b
        je chTic
        in al, 60h
        mov ah,0

        cmp al,4Bh ; check if left arrow
        je subX
        cmp al,4dh ; check if rigth arrow
```

```
        jne chTic
        add [xb],4
        call showBus
        jmp chTic
subX:
        sub [xb],4
        call showBus
chTic:
        cmp dx, [Clock]
        je Tick

        ;loop DelayLoop
        pop es
        popa
ret
endp countsec

proc pic1P
        mov dx,offset Pic1
        call OpenShowBmp

        ret
endp pic1P

proc pic2P
        mov dx,offset Pic2
        call OpenShowBmp
        ret
endp pic2P
proc pic3P
        mov dx,offset Pic3
        call OpenShowBmp
        ret
endp pic3P
proc showBus
        mov dx,offset Busket
        mov ax,[xb]
        mov [BmpLeft], ax
        mov ax,[yb]
        mov [BmpTop], ax
        mov [BmpColSize], 60
        mov [BmpRowSize], 12
        call OpenShowBmp
```

```
        ret
endp showBus

proc showGame
        mov [Stop],1
        mov dx,offset GameOver
        mov ax,50
        mov [BmpLeft], ax
        ;mov ax,[yb]
        mov [BmpTop], ax
        mov [BmpColSize], 180
        mov [BmpRowSize], 112
        call OpenShowBmp
        call waitSpace
        ret
endp showGame

proc waitSpace
;check if there is a a new key in buffer
no:     in al, 64h
        cmp al, 10b
        je no

        in al, 60h
        cmp al, [saveKey]  ;check if the key is same as already pressed
        je no

        mov [saveKey], al  ;new key - store it
        mov ah,0

        cmp al,39h ;space
        jne no
        mov [Stop],0
        mov dx,offset clear
        mov ax,0
        mov [BmpLeft], ax
        mov [BmpTop], ax
        mov [BmpColSize], 320
        mov [BmpRowSize], 200
        call OpenShowBmp
        ret
endp waitSpace
END start
```