גירסה 1.00 - 16.4.2003

האסמבלי של 8086 – חלק ראשון

מסמך זה הורד מהאתר http://underwar.livedns.co.il.
אין להפיץ מסמך זה במדיה כלשהי, ללא אישור מפורש מאת המחבר.
מחבר המסמך איננו אחראי לכל נזק, ישיר או עקיף, שיגרם עקב השימוש במידע המופיע במסמך, וכן לנכונות התוכן של הנושאים המופיעים במסמך. עם זאת, המחבר עשה את מירב המאמצים כדי לספק את המידע המדויק והמלא ביותר.

כל הזכויות שמורות ל**ניר אדר**

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

מסמך זה הוא השני שאני כותב בנושא האסמבלי של 8086, אם כי הוא פותח סידרת מסמכים העומדת בפני עצמה.
במסמך זה ניסיתי לפנות אל המתכנת המתחיל, שאינו מכיר אף שפת אסמבלי, ורוצה לקבל הכרות ראשונית עם השפה ועם צורת החשיבה שלה.
נעשה ניסיון להתאים את המסמך אל תוכנית הלימודים של בתי הספר התיכוניים, אולם ייתכן שבמקרים מסויימים הרחבתי יותר מדי או פחות מדי, ביחס אל תוכנית הלימודים.

האסמבלי של 8086 – חלק ראשון

האוגרים

במעבד קיימים 14 אוגרים, אשר כל אחד מהם הוא בן מילה אחת - 16 סיביות.
ניתן לחלק את האוגרים ל-4 קבוצות.

אוגרים כלליים

	High	Low	
Accumulator	AH	AL	AX
Base	BH	BL	BX
Count	CH	CL	CX
Data	DH	DL	DX

ישנם 4 אוגרים כלליים: AX, BX, CX, DX.
לכל אחד מהם ניתן להתייחס כאל שני אוגרים קטנים - כל אחד בן 8 בתים. למשל, נוכל להתייחס אל AX כאל שני אוגרים - AL, AH.

1. AX - אוגר זה מכונה גם צובר. משתמשים בו לרוב לסיכום מספרים.
2. BX - אוגר הבסיס.
3. CX - אוגר המונה.
4. DX - אוגר הנתונים.

אוגרי המקטע

CS	Code Segment
DS	Data Segment
ES	Extra Segment
SS	Stack Segment

1. CS - אוגר מקטע הקוד - מכיל את כתובת ההתחלה של התוכנית.
2. DS - אוגר מקטע הנתונים - מכיל את כתובת ההתחלה של מקטע הנתונים.
3. ES - אוגר הנתונים הנוספים - מכיל את כתובת ההתחלה של הנתונים הנוספים אם ישנם.
4. SS - אוגר המחסנית - מכיל את כתובת תחילת המחסנית.

אוגרי מצביעים

SI	Source Index
DI	Destination Index
BP	Base Pointer
SP	Stack Pointer
IP	Instruction Pointer

1. SI - אוגר מצביע מקור - מצביע על כתובת תא זיכרון מבוקש.
2. DI - אוגר מצביע יעד - מצביע על כתובת תא זיכרון מבוקש.
3. BP - מצביע הבסיס - משמש כמצביע על כתובת תאי הזיכרון במחסנית.
4. SP - מצביע המחסנית - מצביע על כתובת תא הזיכרון בקצה המחסנית.
5. IP - מצביע הפקודה - כתובת הפקודה הבאה לביצוע.

אוגר הדגלים

F	Flags Register

1. F - אוגר הדגלים - מכיל 9 סיביות המתארים מאפיינים שונים המתארים את הפעולה האחרונה שבוצעה על ידי ה-ALU של המעבד.

תחביר בסיסי של שפת אסמבלר

תחביר בסיסי

מבנה פקודה

האסמבלר כוללת עשרות פקודות. לפקודות מספר תבניות.
התבנית הנפוצה ביותר היא מהצורה:

```
<Command> <Destination> , <Source>
```

דוגמא:
הפקודה MOV משמשת להעתקה של נתון ממקום אחד למקום אחר בתוכנית.
שימוש בפקודה יכול להראות כך:

```
MOV AX, BX
```

הערות

ניתן להוסיף הערות מימין לפקודה, או בתחילת שורה. לשם כך נשתמש בתו ; (נקודה פסיק).
כל טקסט שיופיע לאחר הנקודה פסיק ייחשב כהערה.
דוגמא:

```
MOV AX, BX ; This is a comment
```

תוויות

ניתן לתת שם לשורה כלשהי בתוכנית. שם זה נקרא תווית.
כדי ליצור תווית, נכתוב בתחילת השורה הרצויה את שם התווית, ולאחריו נקודתיים.

```
<Label Name>: <command>
```

לדוגמא:

```
MyLabel: MOV AX, BX
```

שם תווית:

- התו הראשון בשם התווית יכול להיות קו תחתון או אות אנגלית.
- כל שאר התווים יכולים להיות: קו תחתון, אות אנגלית (גדולה או קטנה) ומספרים בלבד.

לדוגמא:
תוויות חוקיות:

```
abc
Ab3
__s
```

תוויות לא חוקיות:

```
3ABC
BC*
```

<u>מספרים</u>

האסמבלר תומך במספר שיטות ספירה. נוכל לעבוד בבסיס דצימלי, בבסיס בינארי ובבסיס הקסדצימלי.
מספר בבסיס בינארי יסומן על ידי האות B בסוף המספר. למשל, המספר 1101B הינו מספר בייצוג בינארי.
מספר בייצוג דצימלי מופיע כשלאחריו האות D, או ללא כל סימון, למשל 34D או 374.
מספר בייצוג הקסדצימלי מופיע עם האות H בסיומו, למשל, A4H או 34H.

כללים בסיסיים

התאמה בגודל האופרנדים

עבור כל הפקודות באסמבלי, יש לשמור על כלל ההתאמה בגודל האופרנדים, האומר כי גודלו של אופרנד היעד חייב להיות זהה לגודל אופרנד המקור.

ניתן לראות טענה זו במספר מקרים:

1. אסור להכניס לחצי אוגר (8 סיביות) מספר בעל יותר מ-2 ספרות הקסדצימליות (כי הרי כל סיפרה ב-HEX מתרגמת ל-4 ביטים. לדוגמא:

```
MOV AL, 436H
```

2. אסור להכניס לאוגר מספר הגדול מ-4 ספרות הקסדצימליות:

```
MOV BX, 4374H
```

3. חייבת להיות התאמה בגודל האוגרים: לא ניתן להעתיק נתון בן 16 סיביות לחצי אוגר בן 8 סיביות או להפך:

```
MOV AL, CX
MOV CX, AL
```

העתקת נתון מאוגר לתא זיכרון

על מנת לפנות לזיכרון, אנו משתמשים באופרטור [] (סוגרים מרובעים). למשל:

```
MOV [BX], AL
```

עבור כל פניה לזיכרון, יש לציית לכללים הבאים:

1. כדי לציין כתובת של תא זיכרון בתוך הסוגריים המרובעים, אסור לכתוב מספר, אלא רק את אחד מהאוגרים BX, SI, DI.
   קיימות מספר אפשרויות לשימוש באוגרים אלו:
   - שימוש באחד מהאוגרים בלבד, למשל: [BX], [SI], [DI].
   - שימוש באוגר וערך, למשל: [BX+34H], [SI+1].
   - שימוש באוגר הבסיס BX ואוגר אינדקס SI או DI, כך: [BX+SI], [BX+DI].
   - כאשר רוצים לפעול על תא זיכרון אחד, יש להשתמש באוגר בגודל בית.
2. אין לרשום פקודה הכוללת תא זיכרון ומספר. כדי לבצע פקודה בין תא זיכרון ומספר יש לבצע את הפעולות הבאות:
   - להציג את הנתון באוגר בגודל מתאים.
   - לבצע פעולה בין תא הזיכרון לאוגר.
   למשל, הפקודה: MOV [BX], 5 שגויה.

3. אסור לבצע פעולות ישירות בין שני תאי זיכרון, כלומר אסור לבצע פעולה שבה שני האופרנדים הם תאי זיכרון.
   למשל, הפקודה MOV [BX], [DI] שגויה.
4. כאשר רוצים לפעול על תא זיכרון אחד יש להשתמש באוגר בגודל בית.
   כאשר רוצים לפעול על שני תאי זיכרון יש להשתמש באוגר בגודל מילה.

דוגמא

נניח שנרצה להכניס לתא בכתובת 1000 בזיכרון את המספר 12, נבצע את הפעולות הבאות:

```
MOV BX, 1000
MOV AL, 12
MOV [BX], AL
```

דוגמא

התוכנית הבאה מעבירה את תוכן התא 600H אל 605H:

```
MOV BX, 600H
MOV AL, [BX]
MOV BX, 605H
MOV [BX], AL
```

פקודות בסיסיות

פקודה	הסבר	דוגמא
INC	הפקודה מורה למעבד להגדיל ערך של אוגר או משתנה ב-1.	חוקי: INC DL לא חוקי: INC [SI]
DEC	הפקודה מורה למעבד להקטין ערך של אוגר או משתנה ב-1.	חוקי: DEC AX לא חוקי: DEC [BX}
ADD	הפקודה מורה למעבד לבצע פעולת חיבור.	חוקי: ADD CH, 5 חוקי: ADD AL, [BX] לא חוקי: ADD [SI], [DI]
SUB	הפקודה מורה למעבד לבצע פעולת חיסור:	חוקי: SUB CH, 5 חוקי: SUB AL, [BX] לא חוקי: SUB [SI], [DI]
NOP	פקודה שאיננה מבצעת דבר. הפקודה איננה מקבלת פרמטרים כלשהם.	חוקי: NOP
CMP	ביצוע השוואה בין שני ערכים. השוואה נעשית על ידי חיסור של אופרנד המקור מאופרנד היעד, מבלי לשנות את אופרנד היעד. תוצאות ההשוואה יכולות להיות: שווה, קטן או גדול. התוצאה תשמר בדגלים, כך שלאחר הביצוע ניתן לבדוק ולהשתמש בה.	חוקי: CMP AL, 2 חוקי: CMP AX, BX לא חוקי: CMP [SI], [DI]
JMP	פקודת קפיצה לא מותנית. אחרי הפקודה מופיע אופרנד אחד, המציין את הכתובת אליה נקפוץ. בדרך כלל אופרנד זה יהיה תווית (Label).	JMP LABEL1 ... LABEL1: ...

פקודות קפיצה מותנית

פקודות קפיצה מותנות הן פקודות קפיצה, אשר גורמות לקפיצה למקום אחר בתוכנית רק אם מתקיים תנאי כלשהו. התנאי נקבע לפי מצב הדגלים.

פקודות הקפיצה המותנית:

משמעות	פקודה
קפוץ אם הערכים שווים.	JE
קפוץ אם הערכים אינם שווים.	JNE
קפוץ אם הערך שרשום משמאל גדול מהערך שרשום מימין.	JG
קפוץ אם הערך משמאל קטן מהערך מימין.	JL
קפוץ אם הערך השמאלי גדול או שווה מהערך הימני.	JGE
קפוץ אם התוצאה האחרונה שווה 0.	JZ
קפוץ אם התוצאה האחרונה שונה מ-0.	JNZ

כתיבת תוכנית שלמה

כתיבת תוכנית

כדי לכתוב תוכנית שלמה בשפת אסמבלי יש להוסיף לתוכנית שכתבנו פתיחה וסיום.

הפתיחה של כל תוכנית תהיה:

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
START:
      MOV AX, CODE
      MOV DS, AX
```

הסיום של כל תוכנית יהיה:

```
CODE ENDS
END START
```

חלק מהפתיחה והסיום אינם פקודות בשפת אסמבלי, אלא הן הנחיות לאסמבלר.
נשים לב להבדל בין פקודות להנחיות:
פקודות מתורגמות לשפת מכונה ומבוצעות בשלב הרצת התוכנית.
הנחיות מיועדות להנחות את תוכנית האסמבלר כיצד לתרגם את התוכנית לשפת מכונה.

- המילה SEGMENT מציינת שזהו מקטע של תוכנית אסמבלי, שגודלו המקסימלי 64KB.
- המילה CODE נבחרה כדי לייצג את שם מקטע התוכנית. במקומה ניתן לבחור כל שם אחר. בדרך כלל נהוג לקרוא למקטע התוכנית בשם CODE או CSEG.
- המילה ASSUME מרחיבה את ההסבר אודות מקטע תוכנית האסמבלי: אוגר CS, אשר מכיל את כתובת ההתחלה של התוכנית, יתחיל במקטע בשם OCDE. כך גם לגבי האוגר DS: כתובת ההתחלה של הנתונים, במידה וישנם, יתחיל במקטע בשם CODE.
- ההוראה CODE ENDS מציינת שזהו סוף המקטע CODE.
- ההוראה END START מציינת שני דברים:
  1. זהו סוף התוכנית.
  2. תחילת התוכנית נמצאת במקום בו מופיעה התווית START.

דוגמא

כתוב תוכנית שתחליף בין תוכן האוגרים CX ו-BX.

פתרון א':

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
START:
      MOV AX, CODE
      MOV DS, AX

      MOV AX, CX
      MOV DX, BX
      MOV CX, DX
      MOV BX, AX

CODE ENDS
END START
```

פתרון ב':

נרצה לחסוך: קודם כל בכמות הרגיסטרים שאנחנו משתמשים בהם, וכן גם בכמות השורות שאנו כותבים.

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
START:
      MOV AX, CODE
      MOV DS, AX

      MOV AX, CX
      MOV CX, BX
      MOV BX, AX

CODE ENDS
END START
```

עבודה עם debug

debug הוא כלי המאפשר לנו להציץ בקוד המכונה של קבצי EXE, להריצם שורה שורה, וכן לשנות את ערכי הרגיסטרים או הזיכרון תוך כדי הריצה.

להתחלת העבודה עם debug נכתוב:

```
debug <שם תוכנית>
```

למשל:

```
debug myprog.exe
```

להצגת התוכנית שלנו, החל מכתובת 0 בסגמנט הקוד, נכתוב: u0
להרצת התוכנית שורה שורה, נשתמש בפקודה t.
כדי לקבל רשימה של הפקודות הזמינות, נקיש ?.
כדי להדפיס את ערכי הרגיסטרים ברגע כלשהו, נכתוב r.
להרצת התוכנית מכתובת 0 עד כתובת A, למשל, נכתוב: g=0 A

לולאות

לולאות בעזרת פקודות קפיצה

כדי לבצע לולאות באסמבלי, אין צורך בפקודות נוספות מעבר למה שלמדנו עד כה.
לולאה הינה חזרה על קטע קוד מספר פעמים. קטע הקוד יכול לחזור מספר פעמים קבוע מראש או יכול לחזור עד שיתקיים תנאי מסוים.

נביא כעת דוגמא לתוכנית עם לולאה. בסוף התוכנית, ברגיסטר AX נשמר סכום המספרים מ-1 עד 10.

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
START:
      MOV AX, CODE
      MOV DS, AX

      MOV CX, 10
      MOV AX, 0
MYLOOP:
      ADD AX, CX
      DEC CX
      JNZ MYLOOP

CODE ENDS
END START
```

הפקודה LOOP

פקודה זו מיועדת לסייע למתכנת בכתיבת לולאות. הפקודה מחליפה את שתי הפקודות:

```
      DEC CX
      JNZ MYLOOP
```

במקומן נכתוב את הפקודה:

```
      LOOP MYLOOP
```

פעולתה זהה לפעולת שתי פקודות אלו. נשים לב כי הפקודה LOOP פועל על אוגר CX בלבד.

דוגמא

התוכנית הבאה מזיזה תא אחד ימינה את הערכים של שישה תאים, החל מהכתובת 200H. התוכנית תאפס את התא השמאלי ביותר.

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
START:
      MOV AX, CODE
      MOV DS, AX

      MOV SI, 205H
      MOV CX, 6
CONT:
      MOV AL, [SI]
      MOV [SI+1], AL
      DEC SI
      LOOP CONT

      MOV SI, 200H
      MOV AL, 0
      MOV [SI], AL

CODE ENDS
END START
```

פסיקות תוכנה

כאשר אנו קוראים לפסיקת תוכנה, אנו מפסיקים למעשה את ביצוע התוכנית שלנו, קופצים אל תוכנית אחרת הכתובה בזיכרון המחשב, מבצעים אותה, ולאחר מכן חוזרים להמשך התוכנית שלנו, מהמקום בו היא הופסקה.
מערכת ההפעלה ויצרני המחשב מספקים עבור המתכנת רשימה ארוכה של פסיקות מוכנות בהן אנו יכולים להיעזר. הפסיקות משמשות למגוון פעולות - קליטת נתונים מהמקלדת, הצגת נתונים על המסך, סיום התוכנית ועוד.

על מנת להשתמש בפסיקות, אנו צריכים להגדיר גם מקטע מחסנית לתוכנית שלנו.
נעשה זאת על ידי כתיבת הקטע הבא:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS
```

כמו כן, צריך לציין לאסמבלר כי זהו מקטע המחסנית. נעשה זאת על ידי שינוי הפקודה ASSUME כלהלן:

```
ASSUME CS:CODE, DS:CODE, SS:STA
```

כעת המבנה של תוכנית בסיסית ייראה כך:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX


CODE ENDS
END START
```

על מנת להשתמש בפסיקות אנו משתמשים בפקודה INT. מיד נראה דוגמאות לשימוש בפקודה זו.

פסיקות נבחרות

סיום התוכנית

פסיקת DOS מספר 21H, קוד הפסיקה 4C00H נשמר באוגר AX.
הפסיקה גורמת לסיום התוכנית ולהחזרת השליטה אל מערכת ההפעלה.

דוגמא

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      MOV AX, 4C00H
      INT 21H

CODE ENDS
END START
```

הצגת מחרוזת על המסך

פסיקת DOS מספר 21H, קוד הפסיקה 9H נשמר באוגר AH.
הפסיקה גורמת להדפסת מחרוזת על המסך.
שלבי הפעולה:

1. עלינו לכתוב את המחרוזת שברצוננו להציג. המחרוזת נרשמת בסוף התוכנית בין גרשיים, ובסיומה נרשם הסימן $, המציין את סיומה. הצהרת המחרוזת תתחיל בשם כלשהו שייצג אותה, לאחריו ההוראה DB, ולאחריו המחרוזת. נסביר צורת כתיבה זו בפירוט בהמשך.
2. נציב ב-DX את כתובת ההתחלה של המחרוזת. הדבר נעשה באמצעות הפקודה OFFSET המחשבת את ההיסט של תחילת ההודעה מתחילת הסגמנט.
3. קריאה לפסיקה.

הערה: פסיקה זו גורמת לשינוי התוכן של רגיסטר AL.

דוגמא

התוכנית הבאה מדפיסה את המחרוזת "Hello, World" על המסך:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      ; Print the string
      MOV DX, OFFSET STR1
      MOV AH, 9H
      INT 21H

      ; End the program
      MOV AX, 4C00H
      INT 21H

      STR1 DB 'Hello, World$'

CODE ENDS
END START
```

דוגמא

נרצה לכתוב תוכנית שתדפיס יותר משורה אחת על המסך. לשם כך נעזר בעובדה הבאה:
רצף התווים שערך ה-ASCII שלהם הוא 10, 13, גורם לסמן לעבור שורה.
בתוכנית הבאה אנו כותבים את המילה Hello בשורה אחת, ו-World בשורה השניה.
נשים לב כי למרות שהמחרוזת מורכבת ממספר שורות, רק בסיומה אנו שמים $.

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX
```

```
      ; Print the string
      MOV DX, OFFSET STR1
      MOV AH, 9H
      INT 21H

      ; End the program
      MOV AX, 4C00H
      INT 21H

      STR1 DB 'Hello'
           DB 10, 13
           DB 'World$'

CODE ENDS
END START
```

קליטת מקש מהמקלדת

פסיקת DOS מספר 21H, קוד הפסיקה 1H נשמר באוגר AH.
אם קוד הפסיקה 7H נשמר באוגר AH, נקלוט תו אך לא נציג אותו על המסך.

הפסיקה גורמת שהמחשב יעצור וימתין ללחיצה על מקש. כאשר יילחץ מקש, הקוד שלו ייכנס לאוגר AL, וכך התוכנית תוכל לדעת איזה מקש נלחץ.
לכל מקש קוד מזהה שלו - שנקבע לפי סטנדרט ASCII.
למשל: ערכי מקשי הספרות 0 עד 9 הם 30H עד 39H. הערך של האות הקטנה a הוא 61H.

דוגמא

התוכנית הבאה תקבל קלט מהמקלדת, עד אשר תוצג האות a ואז היא תדפיס הודעה ותסתיים.

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      ; Get input from the user
INPUT_LOOP:
      MOV AH, 1
      INT 21H
      CMP AL, 61H
      JNE INPUT_LOOP
```

```
      ; Print the string
      MOV DX, OFFSET QUIT_MSG
      MOV AH, 9H
      INT 21H

      ; End the program
      MOV AX, 4C00H
      INT 21H

      QUIT_MSG DB 10, 13
             DB 'Exiting...$'

CODE ENDS
END START
```

קליטת תווים מיוחדים:

חלק מהמקשים מאופיינים על ידי שני קודי ASCII, לעומת קוד אחד, המאפיין את רוב התווים.
עבור מקשים אלו, הקוד הראשון הינו 0, והקוד השני משתנה בהתאם למקש.
בקבוצת מקשים אלו נמצאים F1 עד F9, מקשי החצים, HOME, DELETE ועוד.
כדי לזהות בתוכנית אם נלחץ מקש מיוחד יש לבדוק האם קוד ה-ASCII שהתקבל הוא 0.
אם כן, יש לבצע פסיקה פעם נוספת, כדי לקבל את הקוד השני השייך למקש.

הצגת תו בודד על המסך

פסיקת DOS מספר 21H, קוד הפסיקה 2H נשמר באוגר AH.
התו שאותו אנו רוצים להדפיס יוצב לפני כן ברגיסטר DL.

הפסיקה גורמת להדפסת התו שנמצא ברגיסטר DL על המסך.

דוגמא

התוכנית הבאה מדפיסה את התו a על המסך:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX


      MOV DL, 'a'

      MOV AH, 2
      INT 21H

      ; End the program
      MOV AX, 4C00H
      INT 21H

CODE ENDS
END START
```

נשים לב לצורת הכתיבה בה השתמשנו:

```
      MOV DL, 'a'
```

האסמבלר מחליף את 'a' לערך ה-ASCII שלו - 61H, כאשר הוא מתרגם את התוכנית לשפת מכונה.
מצד שני, לנו נוח יותר להשתמש בסימון 'a' מאשר לזכור את ערך ה-ASCII שלו.

דוגמא

התוכנית הבאה מציגה את הספרות 0 עד 9 על המסך, אחת אחרי השנייה.

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      ; Initialize
      MOV CX, 10
      MOV DL, '0'

      ; Printing Loop
PRN_LOOP:
      MOV AH, 2
      INT 21H
      INC DL
      LOOP PRN_LOOP

      ; End the program
      MOV AX, 4C00H
      INT 21H

CODE ENDS
END START
```

פסיקה הבודקת האם הוקש מקש

פסיקת DOS מספר 21H, קוד הפסיקה 0BH נשמר באוגר AH.

כדי לבדוק האם הוקש תו כלשהו במקלדת נשתמש בפסיקה זו. אם נקיש על מקש כלשהו, יוצב ברגיסטר AL הערך FFH, אחרת יוצב בו הערך 0.

קביעת מיקום הסמן במסך

פסיקת BIOS מספר 10H, קוד הפסיקה 2H נשמר באוגר AH.

פסיקה זו משמשת לשינוי מיקומו של הסמן על המסך.
על מנת לשנות את מיקומו של הסמן יש לבצע את הפעולות הבאות:

1. לשים ברגיסטר BH את מספר המסך המבוקש (מסך ראשון הינו מסך מס' 0).
2. להציב באוגר DH את מספר השורה (0-23) ובאוגר DL את מספר הטור (0-79).
3. לקרוא לפסיקה.

דוגמא

התוכנית הבאה מדפיסה את המחרוזת 'Hello, World' במרכז המסך:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      ; Set cursor location to (11, 33)
      MOV BH, 0
      MOV DH, 11
      MOV DL, 33
      MOV AH, 2H
      INT 10H

      ; Print the string
      MOV DX, OFFSET HELLO
      MOV AH, 9H
      INT 21H

      ; End the program
      MOV AX, 4C00H
      INT 21H

      HELLO DB 'Hello, World$'

CODE ENDS
END START
```

קליטת מחרוזת מהמקלדת

פסיקת DOS מספר 21H, קוד הפסיקה 0AH נשמר באוגר AH.

פסיקה זו מאפשרת לנו לקלוט מספר תווים מהמקלדת על ידי פסיקה אחת, במקום המצב שהיה עד כה, שיכולנו בעזרת פסיקה לקרוא רק תו בודד בכל פעם.
בעזרת פסיקה זו נוכל לקרוא רצף של תווים – עד שהמשתמש לוחץ על מקש ה-Enter.

דרך הפעולה:
1. נגדיר משתנה כלשהו שיכיל:
   - בתא הראשון שלו יירשם מספר המציין את גודל המחרוזת אותו אנו רוצים לקלוט.
   - נגדיר שטח של מספר תאים ריקים, אשר לתוכם תוכנס המחרוזת המוקלדת.
   מספר הבתים הריקים שעלינו להשאיר הוא גודל המחרוזת אותה אנו רוצים לקלט + 2 תאים נוספים.
2. נציב באוגר DX את כתובת ההתחלה של המשתנה.
3. נקרא לפסיקה.

תוצאה:
1. בתא השני של המשתנה יירשם מספר התווים שנקלטו בפועל, לא כולל המקש Enter.
2. התווים שהוקשו יוכנסו החל מהתא השלישי של המשתנה.

דוגמא

התוכנית הבאה מבקשת מהמשתמש להקליד את שמו, ולאחר מכן מציגה אותו על המסך.

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      ; Ask the user to insert his name
      MOV DX, OFFSET QUESTION_STRING
      MOV AH, 9H
      INT 21H

      ; Get input from the user
      MOV DX, OFFSET USER_NAME
      MOV AH, 0AH
      INT 21H
```

```
      ; Save string length on CL
      MOV SI, DX
      MOV CL, [SI+1]

      ADD SI, 2

      ; Step to the next line
      MOV DL, 10
      MOV AH, 2
      INT 21H

      ; Print welcome message
      MOV DX, OFFSET WELCOME
      MOV AH, 9H
      INT 21H

      ; Print user name
PRNLOOP:
      MOV DL, [SI]
      MOV AH, 2
      INT 21H
      INC SI
      DEC CL
      JNZ PRNLOOP

      ; End the program
      MOV AX, 4C00H
      INT 21H

      QUESTION_STRING DB 'Please enter your name: $'
      WELCOME DB 'Welcome, $'

      USER_NAME DB 10
              DB 12 DUP (?)

CODE ENDS
END START
```

בתוכנית הדגמנו שימוש בפקודה חדשה:

```
        DB 12 DUP (?)
```

בפקודה זו אנו מגדירים 12 תאים ריקים. הפקודה DUP (מלשון Duplicate – לשכפל) אומרת לאסמבלר ליצור 12 תאים. ה-? אומר לאסמבלר שתוכנם של התאים אינו מוגדר. כלומר – אנו אומרים לאסמבלר ליצור 12 תאים שתוכנם אינו מוגדר. בתאים אלו תישמר בהמשך המחרוזת שנקלוט מן המשתמש.

אתחול המחשב

פסיקת DOS מספר 19H, ללא קוד פסיקה.

פסיקה זו מאפשרת לנו לאתחל את המחשב.

דוגמא

התוכנית הבאה מאתחלת את המחשב מיד כשאנו מריצים אותה:

```
STA SEGMENT STACK
      DB 100H DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:STA
START:
      MOV AX, CODE
      MOV DS, AX

      INT 19H

CODE ENDS
END START
```

<u>סיכום הפסיקות</u>

הפעולה	הפסיקה	כיצד להשתמש בפסיקה
סיום התוכנית	פסיקת DOS מספר 21H 4C00H נשמר באוגר AX.	1. קריאה לפסיקה.
הצגת מחרוזת על המסך	פסיקת DOS מספר 21H. 9H נשמר באוגר AH.	1. עלינו לכתוב את המחרוזת שברצוננו להציג. המחרוזת נרשמת בסוף התוכנית בין גרשיים, ובסיומה נרשם הסימן $, המציין את סיומה. הצהרת המחרוזת תתחיל בשם כלשהו שייצג אותה, לאחריו ההוראה DB, ולאחריו המחרוזת. 2. נציב ב-DX את כתובת ההתחלה של המחרוזת. הדבר נעשה באמצעות הפקודה OFFSET המחשבת את ההיסט של תחילת ההודעה מתחילת הסגמנט. 3. קריאה לפסיקה. פסיקה זו גורמת לשינוי התוכן של רגיסטר AL.
קליטת מקש מהמקלדת	פסיקת DOS מספר 21H. 1H נשמר באוגר AH. אם 7H נשמר באוגר AH, נקלוט תו אך לא נציג אותו.	הפסיקה גורמת שהמחשב יעצור וימתין ללחיצה על מקש. כאשר יילחץ מקש, הקוד שלו ייכנס לאוגר AL, וכך התוכנית תוכל לדעת איזה מקש נלחץ.
הצגת תו בודד על המסך	פסיקת DOS מספר 21H. 2H נשמר באוגר AH.	1. נציג את התו המבוקש ברגיסטר DL. 2. נקרא לפסיקה.
פסיקה הבודקת האם הוקש מקש	פסיקת DOS מספר 21H. 0BH נשמר באוגר AH.	1. קריאה לפסיקה. 2. אם מקש כלשהו לחוץ, יוצב ברגיסטר AL הערך FFH, אחרת יוצב בו הערך 0.
קביעת מיקום הסמן במסך	פסיקת BIOS מספר 10H. 2H נשמר באוגר AH.	1. נשים ברגיסטר BH את מספר המסך המבוקש (מסך ראשון הינו מסך מס' 0). 2. נשים באוגר DH את מספר השורה (0-23) ובאוגר DL את מספר הטור (0-79). 3. נקרא לפסיקה.
קליטת מחרוזת מהמקלדת	פסיקת DOS מספר 21H. 0AH נשמר באוגר AH.	1. נגדיר משתנה כלשהו שיכיל: • בתא הראשון שלו יירשם מספר המציין את גודל המחרוזת אותו אנו רוצים לקלוט. • נגדיר שטח של מספר תאים ריקים, אשר לתוכם תוכנס המחרוזת המוקלדת. מספר הבתים הריקים שעלינו להשאיר הוא גודל המחרוזת אותה אנו רוצים לקלט + 2 תאים נוספים. 2. נציב באוגר DX את כתובת ההתחלה של המשתנה. 3. נקרא לפסיקה. תוצאה: 1. בתא השני של המשתנה יירשם מספר התווים שנקלטו בפועל, לא כולל המקש Enter. 2. התווים שהוקשו יוכנסו החל מהתא השלישי של המשתנה.
אתחול המחשב	פסיקת DOS מספר 19H.	1. קריאה לפסיקה.

זיכרון המחשב

התוכניות שאנו כותבים, וכן הנתונים עליהם הן פועלות נמצאים בזיכרון המחשב.
הזיכרון מכיל אוסף תאים, כשכל תא הוא בגודל בית – 8 סיביות.
כל תא מזוהה באופן יחיד על ידי מספר מזהה המכונה כתובת התא.
כיצד ניגש לתאי זיכרון? ראינו גישה למשל בצורה הבאה:

```
MOV BX, 100H
MOV AX, [BX]
```

במקרה זה הבאנו את תוכן התא שכתובתו 100H לתוך AX.
כזכור, BX הוא רגיסטר בן 16 סיביות, כלומר המספר הכי גדול שנוכל לשים בתוכו הוא 65535.
עם זאת, בזיכרון המחשב ישנם יותר מ-65535 תאים. כיצד נוכל לגשת אל התאים הנוספים?
הפתרון שהוצא על ידי מתכנני ה-PC: כל כתובת בזיכרון תורכב משני חלקים: , <Offset>.
Segment זוהי כתובת הבסיס, ו-Offset זהו ההיסט.

הבסיס הוא ערך שיסמן את כתובת ההתחלה. נשתמש בכתובת ההתחלה במספר תפקידים, שלכל אחד מהם הוקדש אוגר נפרד:

1. האוגר CS – Code Segment – מכיל את כתובת ההתחלה של הקוד – התוכנית שאנו מריצים.
2. האוגר DS – Data Segment – מכיל את כתובת ההתחלה של הנתונים של התוכנית.
3. האוגר ES – Extra Segment – מכיל את כתובת ההתחלה של הנתונים הנוספים של התוכנית, אם ישנם.
4. האוגר SS – Stack Segment – מכיל את כתובת ההתחלה של מחסנית התוכנית.

ההיסט הוא המרחק בבתים שבין כתובת הבסיס עד לתא הרצוי.
כתובת הזיכרון שאליה פונה המעבד מחושבת לפי הנוסחה:

$$Address = <Segment> \cdot 16 + <Offset>$$

Address היא הכתובת הפיסית בה נשמרים הנתונים.

נבדיל כעת בין שני מושגים: כתובת פיסית וכתובת לוגית.
כתובת פיסית הינה הכתובת שאליה יפנה המעבד בפועל. כתובת לוגית הינה ההיסט מהבסיס.
בכל התוכניות שכתבנו עד כה, כל הכתובות בהן השתמשנו היו כתובות לוגיות.

נשים לב כי כתובת פיסית היא יחידה, אולם ייתכנו כתובות לוגיות רבות שמתאימות לה. כלומר, תיתכן יותר מכתובת לוגית אחת עבור אותה כתובת פיסית.

פנייה לכתובות פיסיות באמצעות תוכנית

בכל פעם שאנו משתמשים באוגרים BX, SI, DI כמצביעים על כתובות תאי זיכרון, הערכים של אוגרים אלו הינם היסט, ואילו הבסיס הינו תוכן אוגר DS כברירת מחדל.
לכן אם נרצה, למשל, לשים את הערך 7 בכתובת 920H, עלינו לבחור צירוף של בסיס (האוגר DS) והיסט (אחד מהאוגרים BX, SI, DI) כך שנקבל את הכתובת הפיסית 920H.

התוכנית הבאה שמה בכתובת 920H את הערך 7:

```
CODE SEGMENT
     ASSUME CS:CODE
START:
     MOV AX, 90H
     MOV DS, AX
     MOV DL, 7
     MOV BX, 20H
     MOV [BX], DL
CODE ENDS
END START
```

בתוכנית זו שמנו ב-DS את הערך 90H במקום הערך CODE שאנו שמים בד"כ, וזאת מכיוון שאיננו רוצים שהוא יצביע על התחלת סגמנט הקוד, אלא על סגמנט אחר.
לא היינו צריכים לכתוב גם ASSUME DS:CODE מאותה סיבה.

ניתן להשתמש באוגרים נוספים מלבד DI, SI, BX בתור היסט בזיכרון.
ההבדל הוא שעבור שלושת אוגרים אלו, הבסיס נקבע לפי DS, ולגבי האוגרים הנוספים אוגר אחר משמש כהיסט. נציג זאת בטבלה:

אוגרים מצביעים	הבסיס
DI, SI, BX	DS
BP, SP	SS
IP	CS

הבסיס שנקבע הינו ברירת מחדל. ניתן לשנות את הבסיס.
למשל, בדוגמא הבאה נעביר ל-AH את התא שנמצא בהיסט 15H החל מהבסיס SS:

```
     MOV BX, 15H
     MOV AH. SS:[BX]
```

אוגר הדגלים

סקירה כללית

אוגר הדגלים הוא מילה בת 16 סיביות. 9 סיביות מתוכו משמשות כדגלים.
הדגלים מושפעים מפעולות לוגיות ואריתמטיות בלבד. לא כל פקודה משפיעה על הדגלים.
את ערכם של חלק מהדגלים ניתן לקבוע על ידי פקודות מיוחדות.
הדגלים השונים:

1. zero flag – דגל זה יהיה במצב 1 כאשר ערכה של התוצאה הלוגית או אריתמטית האחרונה הוא 0.
2. sign – ערכו כערכה של סיבית ה-MSB של התוצאה האחרונה. אם אנו מתייחסים את התוצאה האחרונה כאל מספר בעל סימן, דגל זה אומר לנו האם התוצאה היא חיובית או שלילית.
3. overflow – בפעולות חיבור/חיסור, דגל זה הינו 1 אם תוצאת הפעולה היא מעבר לתחום האופרנד בו היא תישמר.
4. carry – דגל זה הינו 1 אם ישנו נשא מהפעולה האחרונה.
5. auxiliary carry – דגל זה הינו 1 אם יש נשא מה-nibble התחתון ל-nibble העליון. nibble זוהי קבוצה בת 4 סיביות – חצי אוגר.
6. interrupt – דגל זה קובע אם פסיקות חומרה מתאפשרות או לא. אם ערכו 0, לא מתאפשרות פסיקות חומרה.
7. parity – בפעולות חיבור/חיסור, דגל זה הוא 1 אם יש מספר זוגי של סיביות 1 בבית התחתון (LSB) של התוצאה.
8. direction – דגל זה קובע את הכיוון עבור פקודות מחרוזת.
9. trap – כאשר דגל זה במצב 1, מתבצעת הרצת צעד יחיד.

מיקום הדגלים באוגר:

	overflow	direction	interrupt	trap	sign	zero flag		auxiliary carry		parity		carry
	11	10	9	8	7	6		4		2		0

הדגלים השונים

נרחיב כעת על מספר דגלים ועל אופן השימוש בהם.

דגל האפס – zero flag

כאשר מתבצע חישוב והתוצאה שלו היא 0, דגל זה עובר למצב 1.
לדוגמא, לאחר הקטע הבא, דגל האפס יקבל 1:

```
MOV AX, 4
SUB AX, 4
```

דגל הסימן – sign flag

מספרים שליליים נרשמים במחשב בשיטת "המשלים ל-2". כאשר הערך של הפעולה החשבונית האחרונה הוא שלילי (אם מתייחסים אל התוצאה כאל מספר בעל סימן), ערכו של דגל זה הוא 1.

דגל הנשא – carry

דגל הנשא עולה ל-1, כאשר בפעולה האחרונה שהתבצעה היה carry או borrow.

דגל הזוגיות – parity

בניגוד לשמו של הדגל, שעשוי להטעות, דגל זה אינו אומר לנו האם תוצאת החישוב האחרונה היא זוגית או לא. בפעולות חיבור/חיסור, דגל זה הוא 1 אם יש מספר זוגי של סיביות 1 בבית התחתון (LSB) של התוצאה. לדוגמא, עבור התוכנית הבאה, דגל הזוגיות יכיל 1:

```
MOV AX, 0
ADD AX, 3
```

דגל הגלישה – overflow

דגל זה עולה ל-1, כאשר אופרנד היעד אינו גדול מספיק כדי להכיל את התוצאה.
לדוגמא:

```
MOV AH, 7FH
ADD AH, 7
```

היינו אמורים לקבל מספר חיובי, אולם AH אינו גדול מספיק, ולכן אנו מקבלים תוצאה שלילית.דגל הגלישה עולה ל-1.

פקודות קפיצה נוספות

מלבד פקודות הקפיצה שראינו עד כה, ישנן פקודות נוספות הקשורות לדגלים השונים. נציג את פקודות הקפיצה השונות בטבלה הבאה:

Instruction	Description	Condition	Aliases	Opposite
JC	Jump if carry	Carry = 1	JB, JNAE	JNC
JNC	Jump if no carry	Carry = 0	JNB, JAE	JC
JZ	Jump if zero	Zero = 1	JE	JNZ
JNZ	Jump if not zero	Zero = 0	JNE	JZ
JS	Jump if sign	Sign = 1	-	JNS
JNS	Jump if no sign	Sign = 0	-	JS
JO	Jump if overflow	Ovrflw=1	-	JNO
JNO	Jump if no Ovrflw	Ovrflw=0	-	JO
JP	Jump if parity	Parity = 1	JPE	JNP
JPE	Jump if parity even	Parity = 1	JP	JPO
JNP	Jump if no parity	Parity = 0	JPO	JP
JPO	Jump if parity odd	Parity = 0	JNP	JPE

EOF