גירסה 1.00 - 28.4.2003

האסמבלי של 8086 – חלק שני

מסמך זה הוא המשך ישיר למסמך הראשון בסדרה. מומלץ לקרוא את המסמך הראשון לפני קריאת מסמך זה.

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

האסמבלי של 8086 – חלק שני

גישה לזיכרון

גישה נוחה יותר לזיכרון

במסמך הקודם הצגנו גישה לזיכרון בצורה הבאה:
נניח למשל שאנו רוצים להציב בתא 1000H את הערך 7, אזי עלינו לכתוב:

```
MOV DL, 7
MOV BX, 1000H
MOV [BX], DL
```

בעזרת המילים BYTE PTR ניתן לגשת ישירות לזיכרון ללא צורך באוגר מתווך.
הקוד הבא, לדוגמא, מבצע בדיוק את אותה פעולה שהרגע הדגמנו:

```
MOV BX, 1000H
MOV BYTE PTR [BX], 7
```

ניתן אף לקצר עוד יותר, ולכתוב את הפקודה הבאה:

```
MOV BYTE PTR DS:[1000H], 7
```

הקידומת :DS מציינת כי הבסיס של הכתובת הינו הרגיסטר DS. במקרה זה מותר לנו לכתוב בתוך הסוגריים המרובעים כתובת, ללא שימוש ברגיסטר נוסף שיכיל את הכתובת.

כדי להציב, לדוגמא, את המספר 1234H בשני תאי הזיכרון העוקבים, 1000H, 1001H, היינו צריכים לכתוב:

```
MOV BX, 1000H
MOV AX, 1234H
MOV [BX], AX
```

גם את רצף זה נוכל לקצר בעזרת המילים WORD PTR, בדרך הבאה:

```
MOV WORD PTR DS:[1000H], 1234H
```

פעולות על סיביות

הפעולה AND

הפקודה משמשת לביצוע כפל לוגי בין שני מספרים, ביט אחר ביט.
שימוש לדוגמא ב-AND הוא מיסוך - איפוס כל הסיביות מלבד סיבית אחת, ובדיקת ערכה.
לדוגמא: התוכנית הבאה תדפיס על המסך את ההודעה OK אם הסיבית השלישית של AL איננה אפס.

```
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 AL, 6  ; Put some value in AX

      ; Check the third bit
      AND AL, 0100B
      JE FINAL

      ; Print the string
      MOV DX, OFFSET OKSTR
      MOV AH, 9H
      INT 21H
FINAL:
      ; End the program
      MOV AX, 4C00H
      INT 21H

      OKSTR DB 'OK$'

CODE ENDS
END START
```

AND מקבלת שני אופרנדים - יעד ומקור, ומבצעת ביניהם את פעולת הכפל הלוגי. התוצאה נשמרת באופרנד היעד.

הפקודה TEST

הפקודה TEST מבצעת פעולה זהה לזו של AND, מלבד העובדה שהיא איננה משנה את ערכו של אופרנד היעד.
לכן, נעדיף להשתמש בפקודה זו כשנרצה לבדוק סיבית מסויימת, למשל, אך לא נרצה לפגוע בערך המקורי.
התוכנית הבאה מבצעת בדיוק את אותה פעולה שביצעה התוכנית בדוגמא הקודמת, אך היא איננה משנה את ערכו של AL.

```
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 AL, 6   ; Put some value in AX

      ; Check the third bit
      TEST AL, 0100B
      JE FINAL

      ; Print the string
      MOV DX, OFFSET OKSTR
      MOV AH, 9H
      INT 21H
FINAL:
      ; End the program
      MOV AX, 4C00H
      INT 21H

      OKSTR DB 'OK$'

CODE ENDS
END START
```

הפקודה OR

פקודה זו מקבלת שני אופרנדים, מבצעת ביניהם חיבור לוגי (OR) סיבית אחרי סיבית, ושומרת את התוצאה באופרנד היעד.
שימוש מקובל בפקודה הוא להעלות ביט אחד בבית ל-1. (מעין הרמת דגל).

דוגמא לשימוש בפקודה OR:
הקוד הבא מאפס את AX, ואז מדליק את הביט הראשון, השלישי והחמישי שבו.

```
MOV AX, 0
OR AX, 10101B
```

הפקודה NOT

הפקודה NOT מקבלת אופרנד בודד. הפקודה הופכת כל ביט, כלומר, כל 0 יהפוך ל-1, וכל 1 יהפוך ל-0.

דוגמא: הפקודה הבאה הופכת את הסיביות של בית 1000H בזיכרון:

```
NOT BYTE PTR DS:[1000H]
```

הפקודה XOR

הפקודה מקבלת שני אופרנדים, מבצעת ביניהם XOR ביט אחר ביט, ושומרת את התוצאה באופרנד היעד.

הפקודה NEG

הפקודה מקבלת אופרנד אחד, מספר אחד - והופכת את הסימן שלו. מספר חיובי יהפוך למספר שלילי, ומספר שלילי לחיובי. הפקודה מתייחסת למספר כאל מספר בשיטת "המשלים ל-2".

הפקודה SHR

הפקודה SHR מזיזה ימינה את כל הסיביות שבאוגר מקום אחד או יותר, ומוסיפה אפסים מצד שמאל.
הפקודה מקבלת שני פרמטרים. הראשון, היעד, הוא האוגר שאת הסיביות שלו אנו רוצים להזיז, והפרמטר השני הוא בכמה להזיז אותן.
דוגמא:

```
MOV AX, 100
SHR AX, 1
```

עבור מספרים חיוביים בלבד, פעולה זו מבצעת חלוקה ב-2.
לגבי מספרים שליליים זה לא נכון, מכיוון ששיטה זו לא מקיימת את התנאים הדרושים כדי שהמספר יישאר מספר שלילי בשיטת המשלים ל-2.

הפקודה SHL

הפקודה SHL מזיזה שמאלה את כל הסיביות שבאוגר מקום אחד או יותר, ומוסיפה אפסים מצד ימין.
הפקודה מקבלת שני פרמטרים. הראשון, היעד, הוא האוגר שאת הסיביות שלו אנו רוצים להזיז,
והפרמטר השני הוא בכמה להזיז אותן.

הפקודה ROR

הפקודה ROR מסובבת את כל הסיביות ימינה. הסיבית הימנית שיצאה חוזרת להיות הסיבית השמאלית ביותר.
מבחינה תחבירית, הפקודה זהה ל-SHR.

הפקודה ROL

הפקודה ROL מסובבת את כל הסיביות שמאלה. הסיבית השמאלית שיצאה חוזרת להיות הסיבית הימנית ביותר.
מבחינה תחבירית, הפקודה זהה ל-SHL.

משתנים

משתנים בשפת אסמבלי

כמו בשפות תכנות רבות, גם שפת אסמבלי תומכת במשתנים.
כדי להשתמש במשתנה בשפת אסמבלי, עלינו להגדירו. עלינו לציין את שם המשתנה, את גודלו ואת הערך ההתחלתי שלו.
דוגמא:

```
MY_VAR DB 7
```

הגדרנו משתנה בשם MY_VAR, שגודלו בית אחד (Define Byte = DB). הערך ההתחלתי של המשתנה הינו 7.

שם המשתנה מקיים את אותם החוקים שעל תוויות לקיים:
שם משתנה:

- 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.
- אין לתת שם משתנה הזהה למילה שמורה של שפת אסמבלר.

את גודל המשתנה, מספר הבתים שיתפוס, קובעים בהתאם לשימוש בו.
גודל המשתנה יכול להיות אחד מהאפשרויות הבאות:

- Define Byte - DB - בית (8 ביט).
- Define Word - DW - מילה (16 ביט).
- Define Double - DD - מילה כפולה (32 ביט).

ניתן לקבוע למשתנה ערך התחלתי, כפי שראינו בדוגמא. ניתן גם לקבוע משתנה שאינו מאותחל, על ידי הסימן ?, למשל:

```
MY_VAR2 DW ?
```

הגדרת משתנים

ניתן להגדיר משתנים בתוך מקטע התוכנית (Code Segment) או במקטע נפרד, מקטע נתונים (Data Segment).
כאשר אנו שומרים את המשתנים במקטע התוכנית, לרוב אנו שומרים אותם בסוף המקטע, לאחר התוכנית.
לדוגמא:

```
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 AH, NUM1

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

      ; Variables definitions
      NUM1 DB 7

CODE ENDS
END START
```

חשוב לשמור על הפרדה בין פקודות התוכנית לבין הגדרת המשתנים. אם לא נשמור על הפרדה כזו, המעבד יתייחס לנתונים כאל פקודות לביצוע.

הדרך השניה לשמור משתנים היא במקטע נפרד - מקטע הנתונים.
דרך זו היא הדרך המקובלת לעבודה עם משתנים. בשיטה זו נגדיר מקטע בו יישמרו הנתונים. למשל:

נגדיר מקטע בשם DATA בו שני משתנים לא מאותחלים:

```
DATA SEGMENT
      INCOME DB ?
      SUM DB ?
DATA ENDS
```

מקטע זה יבוא לרוב לפני מקטע ה-CODE.

כמו כן, נשנה מעט את ההצהרות בתחילת מקטע הקוד: נשנה את ASSUME ונרשום בו את שם מקטע הנתונים. במקום ההצהרה הבאה:

```
      ASSUME CS:CODE, DS:CODE
```

נכתוב:

```
      ASSUME CS:CODE, DS:DATA
```

בהוראה ASSUME אנו מודיעים לאסמבלר את הכתובת ההתחלתית של התוכנית ושל הנתונים.
הכוונה נתונים היא לתאי הזיכרון, המשתנים, בהם התוכניתת משתמשת. כאשר הגדרנו מקטע נתונים נפרד, אנו מודיעים לאסמבלר ש-DS יכיל את כתובת ההתחלה של מקטע הנתונים.

כמו כן נשנה את ההוראות בתחילת התוכנית. עד כה כתבנו:

```
      MOV AX, CODE
      MOV DS, AX
```

נחליף זאת כעת בקוד הבא:

```
      MOV AX, DATA
      MOV DS, AX
```

פקודות אלו מציבות באוגר DS את כתובת ההתחלה של הנתונים.
נשים לב: ASSUME רק מודיעה לאסמבלר כיצד עומדים לנהוג, איך איננה מציבה בפועל באוגרים ערכים באוגרים.

תוכנית כללית תראה כעת כך:

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

DATA SEGMENT
      ; ...
DATA ENDS

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

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

CODE ENDS
END START
```

כללי שימוש במשתנים

לצורך פירוט הכללים, נניח שקיימת תוכנית במבנה הבא, ואנו כותבים הוראות בתור איזור ה-CODE שלה:

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

DATA SEGMENT
      VAR1 DB ?
      VAR2 DB ?
DATA ENDS

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

      ; Code goes here...

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

CODE ENDS
END START
```

הכללים

1. ניתן להציב ערך במשתנה ללא שימוש באוגר, למשל:

```
      MOV VAR1, 4
```

2. לא ניתן להעתיק משתנה אחד אל משתנה אחר.
הפקודה הבאה איננה חוקית:

```
MOV VAR1, VAR2
```

פתרון:

```
      MOV DL, VAR1
      MOV VAR2, DL
```

3. יש לשים לב שכמות הנתונים שאנו רוצים להכניס למשתנה מתאימה לגודל המשתנה שהגדרנו.

4. אין לבצע העברת נתונים באופן ישיר בין משתנה לתא זיכרון. אם ברצוננו להעביר מידע בין משתנה לתא זיכרון צריך אוגר מתאים שיתווך באמצע.

5. אפשר לבצע פעולות לוגיות ומתמטיות על משתנים, כגון SHR, CMP, INC, SUB ועוד.

6. ניתן לרשום את שם המשתנה בסוגריים מרובעים. המשמעות זהה לאותו שם משתנה ללא הסוגריים המרובעים.
לדוגמא:
הקוד הבא ידפיס את המספר 3 (קוד 33H ASCII) פעמיים:

```
MOV VAR1, 33H
MOV [VAR2], 33H

MOV DL, VAR1
MOV AH, 2
INT 21H

MOV DL, [VAR2]
MOV AH, 2
INT 21H
```

<u>מחרוזות</u>

מחרוזת היא למעשה משתנה המכיל רצף תווים. ראינו כבר מספר פעמים את השימוש במחרוזות במסמך זה ובמסמך הקודם.
דוגמא להגדרת מחרוזת:

```
HELLO DB 'Hello, World$'
```

המשמעות של DB במקרה זה, היא שכל תו במחרוזת יתפוס בית.
המחרוזת מושמת ברצף בזיכרון, תו אחר תו. כל תו יימלא בית אחד.

מערכים

מערך הוא רצף של תאים בזיכרון, שניתן להתייחס לכל אחד מהם כאל משנה, ששמו הוא שם המערך בצירוף אינדקס.
למשל, נניח כי נגדיר את המערך ARR, אזי התא הראשון במערך הינו ARR[0], התא השני הינו ARR[1] וכו'.

יצירת מערך מאותחל

כדי ליצור מערך ולאתחל את התאים שבו, נגדיר משתנה, ולאחריו נרשום מספר ערכים המופרדים ביניהם על ידי פסיקים.
לדוגמא:

```
ARR1 DB 7, 8, 9
```

על ידי פקודה זו יצרנו מערך בן שלושה תאים, אשר כל אחד מהם בן בית אחד.
בתא הראשון שמור המספר 7, בתא השני שמור המספר 8 ובתא השלישי שמור המספר 9.
התא הראשון במערך הוא תא מספר 0, התא השני הוא תא מספר 1 והתא השלישי הוא תא מספר 2.
הקוד הבא מדגים שימוש באיברי המערך. אנו שמים את התא הראשון של המערך ב-AL ואת התא האחרון ב-AH:

```
MOV AL, ARR1[0]
MOV AH, ARR1[2]
```

יצירת מערך לא מאותחל

ניצור מערך לא מאותחל על ידי שימוש במילה DUP, לדוגמא:

```
ARR2 DB 9 DUP (?)
```

הפקודה יוצרת מערך לא מאותחל בעל 9 תאים.

דוגמא

הדוגמא הבאה מעתיקה נתונים ממערך אחד לשני, ומדפיסה לאחר מכן את תוכן המערך השני על המסך.

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

DATA SEGMENT
      ARR1 DB 31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H
      ARR2 DB 9 DUP (?)
DATA ENDS

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

      MOV SI, 0
CPYLOOP:
      MOV AL, ARR1[SI]
      MOV ARR2[SI], AL
      INC SI
      CMP SI, 9
      JNE CPYLOOP

      MOV SI, 0
PRNLOOP:
      MOV DL, ARR2[SI]
      MOV AH, 2H
      INT 21H

      INC SI
      CMP SI, 9
      JNE PRNLOOP

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

CODE ENDS
END START
```

נשים לב כי היינו חייבים להשתמש באוגר ביניים כדי להעביר מידע בין המערכים.

המחסנית

רקע

המחסנית היא מבנה נתונים בו אנו מסוגרים לשמור נתונים, ולקבל אותם חזרה מאוחר יותר.
הנתונים נשמרים במחסנית לפי עקרון LIFO (Last In First Out), כלומר, הנתון האחרון שנכנס אל המחסנית, הוא הנתון הראשון שייצא ממנה.

שימושי המחסנית

כאשר אנו משתמשים בפסיקות במהלך התוכנית שלנו, הביצוע קופץ למקום אחר, וכאשר הוא מסתיים ביצוע התוכנית שלנו ממשיך. הכתובת שממנה התוכנית שלנו תמשיך נשמרת על המחסנית.
בהמשך נראה שימוש בשגרות (פונקציות) בשפת אסמבלי. כאשר אנו קופצים אל שגרה, כתובת החזרה (הכתובת אליה נחזור כאשר התוכנית תסתיים) נשמרת על המחסנית, ונשלפת כשאנו חוזרים אל התוכנית הקוראת.
כמו כן, המתכנת מסוגל לשמור נתונים על המחסנית לפי רצונו.

הגדרת המחסנית

הגדרת המחסנית איננה חובה בכל תוכנית. נגדיר מחסנית במקרים הבאים:
- התוכנית שלנו כוללת פסיקות (המעבד שומר במחסנית את כתובת החזרה לתוכנית וכן את אוגר הדגלים).
- התוכנית כוללת שגרות (המעבד שומר את כתובת החזרה של השגרות על המחסנית).
- המתכנת משתמש בפקודות הנוגעות במחסנית.

הגדרת המחסנית תעשה על ידי הפעולות הבאות:
- הגדרת מקטע חדש - תוך שימוש במילה השמורה STACK.
- הגדרת גודל המחסנית באמצעות הקצאת מספר רצוי של תאי זיכרון.
- הכרזה על האוגר SS בהצהרה ASSUME.

דוגמא להגדרת מחסנית:

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

הגדרנו מקטע מחסנית בשם STA בן 100 בתים, אשר אף אחד מהם אינו מאותחל.

הפקודה PUSH

הפקודה PUSH דוחפת, שמה בראש המחסנית נתון. הנתון חייב להיות בגודל מילה שלמה בלבד.
הפקודה מקבלת אופרנד יחיד, המועתק אל המחסנית. האופרנד עצמו נשאר בלא שינוי.

דוגמאות לשימוש חוקי בפקודה:

```
PUSH SI
PUSH [BX]
```

בעת ביצוע השורה הראשונה יידחף ערכו של SI אל המחסנית ובעת ביצוע השורה השניה יידחפו למחסנית שני תאים צמודים: התא בכתובת BX והתא בכתובת BX+1.

דוגמא לשימוש לא חוקי בפקודה:

```
PUSH AL
```

השימוש לא חוקי, מכיוון שגודלו של AL הוא חצי מילה בלבד.

הפקודה POP

הפקודה POP שולפת נתון מראש המחסנית. הנתון הנשלף הוא בגודל מילה שלמה.
הפקודה מקבלת אופרנד יחיד, אליו מועתק המידע מראש המחסנית.

דוגמאות לשימוש חוקי בפקודה:

```
POP SI
POP [BX]
```

בעת ביצוע השורה הראשונה יישלף ערך מהמחסנית ויוכנס אל SI.
בעת ביצוע השורה השניה יישלפו מהמחסנית שני ערכים, וייכנסו אל שני תאים צמודים: התא בכתובת BX והתא בכתובת BX+1.

דוגמא לשימוש לא חוקי בפקודה:

```
POP AL
```

השימוש לא חוקי, מכיוון שגודלו של AL הוא חצי מילה בלבד.

דוגמא

התוכנית הבאה מחליפה בין תוכנם של AX ו-BX:

```
PUSH AX
PUSH BX
POP AX
POP BX
```

מבנה המחסנית

נענה כעת על השאלות הבאות: כיצד נשמרים בפועל הנתונים על המחסנית? מה קורה בפועל כאשר אנו משתמשים בפקודות PUSH ו-POP?
כאשר אנו מגדירים מקטע מחסנית בתוכנית שלנו, אנו למעשה מגדירים מקטע בלוקים בו יישמרו נתוני המחסנית.
כאשר אנו מגדירים את מקטע המחסנית, אוגר SS מכיל את כתובת הבסיס של מקטע המחסנית, ואילו אוגר SP מכיל את קצה המחסנית (היסט מקסימלי), למשל, עבור מחסנית שתוגדר בצורה הבאה, ערכו של SS יהיה כתובת הבסיס, וערכו של SP יהיה 8:

```
STA SEGMENT STACK
      DB 8 DUP (0)
STA ENDS
```

כאשר נבצע פקודת PUSH בה נדחוף מילה, ערכו של SP יקטן ב-2 ויהפך ל-6 בדוגמא לעיל.
בתאים 6, 7 יישמר הערך אותו דחפנו.

באופן כללי: בכל רגע SP מכיל את הכתובת שמתחתיה יוכנסו הנתונים.
כאשר אנו מבצעים פעולת PUSH אנו שומרים את הנתון שנדחף בתאי הזיכרון שמתחת SP, ומשנים את ערכו של SP כך שיצביע אל התא התחתון שהוכנס.

נשים לב כי המחסנית של מחשב ה-PC גדלה כלפי מטה, כלומר, כאשר אנחנו מכניסים עוד ועוד נתונים, הם נשמרים בכתובות זיכרון נמוכות יותר מהכתובות בהן נשמרו קודמיהם.

כאשר מתבצעת פקודת POP, נשלפים הנתונים מהכתובת SP ומ-SP+1, ולאחר מכן ערכו של SP גדל ב-2.

עד כה ראינו כיצד אנו מסוגלים להביט בנתונים שבראש המחסנית בלבד. נוכל גם להביט בנתונים הנמצאים בעומק המחסנית. לשם כך אנו נעזרים באוגר BP.
נביט בדוגמא הבאה:

```
STA SEGMENT STACK
      DB 20 DUP (0)
STA ENDS

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

      MOV AX, 31H
      MOV BX, 32H
      MOV CX, 33H

      MOV BP, SP

      PUSH AX
      PUSH BX
      PUSH CX

      MOV DX, [BP-4]

      MOV AH, 2
      INT 21H

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

CODE ENDS
END START
```

לפני שאנו דוחפים נתונים למחסנית, אנו שומרים את ערכו של SP בתוך האוגר BP.
לאחר שדחפנו נתונים לתוך המחסנית, ייתקיים כי [BP-2] ,[BP-1] מכילים את האיבר הראשון שדחפנו למחסנית. [BP-4] ,[BP-3] מכילים את האיבר השני שדחפנו למחסנית, וכו'. בצורה כזו אנו מסוגלים לגשת לאיברים שדחפנו, לפי סדר הכנסתם למחסנית.

שגרות

שיגרה היא קטע קוד - תת תוכנית, שאנו מסוגלים לקרוא לו מהתוכנית הראשית, ולאחר שהוא מתבצע התוכנית הראשית ממשיכה מהנקודה בה היא הופסקה.
למרות שזה לא מדויק, ניתן לאמר כי ראינו כבר מנגנון דומה - מנגנון פסיקות התוכנה. כאשר קראנו לפסיקה, ביצוע התוכנית שלנו נעצר, הפסיקה ביצעה את פעולתה ולאחר מכן התוכנית שלנו המשיכה לרוץ מהנקודה בה הפסיקה.
רעיון השגרות הוא רעיון דומה. על ההבדלים בין פסיקות תוכנה לשגרות לא נעמוד במסמך זה.
למושג שגרה תרגומים רבים לעברית. על מנת להקל על הקורא המעיין במקורות נוספים, נזכיר חלק מהם. מלבד הבדלי משמעויות מינוריים, כאשר אנו משתמשים במושגים: שיגרה, פרוצדורה, פונקציה, שיטה - אנו למעשה מתכוונים אל אותו מושג.

מדוע נרצה להשתמש בשגרות?

1. כדי לשפר את בהירות התוכנית - חלוקת משימה גדולה למספר משימות משנה מקלה על הבנת וכתיבת התוכנית.
2. על מנת להמנע משכפול קוד - נניח שאנו משתמשים ברצף פעולות מסויים במספר מקומות בתוכנית שלנו, אזי נהפוך רצף פקודות אלו לתת תוכנית, שגרה, ובמקום לכתוב את רצף הפעולות בכל מקום בתוכנית שלנו, נקרא לשגרה כשמתעורר הצורך.

הגדרת שגרה:
הגדרת השגרה נעשית בתוך סגמנט התוכנית.
שיגרה מתחילה בתווית המציינת את שם השיגרה, ומסתיימת בפקודה RET.
הקריאה לשגרה נעשה על ידי הפקודה CALL. הפקודה CALL מקבלת אופרנד יחיד, והוא שם השגרה.
הפקודה CALL גורמת לקפיצה אל קוד השיגרה. קוד השגרה מתבצע עד שמגיעים אל הפקודה RET, ואז המשך ריצת התוכנית ממשיך מהתוכנית/השגרה הקוראת.

נציג לדוגמא שיגרה המקצרת את תהליך הדפסת תו על המסך.
נקבע כי לפני הקריאה לשגרה נשים ב-DL את התו להדפסה. כאשר נקרא לשגרה, היא תדפיס תו זה.

```
STA SEGMENT STACK
      DB 20 DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, SS:STA
START:

      MOV DL, 'A'
      CALL PUTCHAR
      MOV DL, 'B'
      CALL PUTCHAR
      MOV DL, 'C'
      CALL PUTCHAR

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

```
PUTCHAR:
      MOV AH, 2
      INT 21H
      RET

CODE ENDS
END START
```

בין קוד התוכנית הראשית לבין השגרות צריכה להיות הפרדה מוחלטת, על מנת שקוד השגרות יתבצע רק כאשר מתרחשת קריאה אליהן בעזרת CALL.

על מנת שמכונה כלשהי תתמוך בשגרות, היא צריכה לתמוך בשני מנגנונים:

1. מנגנון להעברת בקרה:
   - קריאה לשגרות.
   - חזרה משגרות.
2. מנגנון להעברת פרמטרים.

<u>העברת בקרה</u>

העברת הבקרה נעשית על ידי הפקודות CALL ו-RET.
המעבד מבצע את הקוד פקודה אחר פקודה. כאשר המעבד פוגש בפקודת CALL, מתבצעות הפעולות הבאות:

1. כתובת הפקודה הבאה לביצוע, זו שאחרי הפקודה CALL, נדחף אל המחסנית.
2. ערכו של IP משתנה. ב-IP מושמת כתובת תחילת השגרה.

בפקודת RET מתרחשים השלבים הבאים:

1. כתובת החזרה נשלפת מהמחסנית.
2. IP מקבל את הכתובת שנשלפה מן המחסנית.

<u>העברת פרמטרים לשגרות</u>

נוכל להעביר פרמטרים בשתי גישות:

- לפי ערך by value
- לפי כתובת by address

בשיטה הראשונה הפרמטר שמועבר לשיגרה הוא הפרמטר עצמו.
העברת פרמטר לפי כתובת, כלומר, מעבירים את הפרמטר בעזרת הכתובת שלו, וכאשר הפונקציה רוצה לפנות לפרמטר היא פונה לכתובת.

מיקום הפרמטר בזכרון

שגרות מסוגלות לקבל פרמטרים - מידע מהתוכנית הקוראת, שהן יוכלו לעבד ולהשתמש בו.
ישנן מספר דרכים להעביר פרמטרים לשגרות.

רגיסטרים
הפרמטרים יוצבו ברגיסטרים.
השיגרה מקבלת פרמטר ברגיסטר מסוים ומחזירה ברגיסטר כלשהו.

יתרונות השיטה:
- פשטות
- יעילות

חסרונות השיטה:
- לא ניתן להעביר הרבה פרמטרים בעזרת שיטה זו.
- השיטה גורמת ל"בזבוז" של אוגרים.

הפונקציה שהדגמנו קודם, PUTCHAR, משתמשת ברגיסטר DL כפרמטר.

העברת פרמטרים על ידי מחסנית

ניתן להעביר לשגרה נתונים באמצעות המחסנית. דרך זו היא הדרך המקובלת ביותר על ידי שפות עיליות להעברת פרמטרים לפונקציות.
לפני שקוראים לשגרה, התוכנית דוחפת הפרמטרים שלה על המחסנית, ורק לאחר מכן נעשית הקריאה לשגרה.

יתרונות השיטה:
- מודולריות.
- תמיכה ברקורסיה.

חסרונות השיטה:
- איטית ביחס לעבודה עם רגיסטרים.

דוגמא 1:
נשכתב את הדוגמא שהצגנו קודם, ונגרום שהפעם הפרמטר עבור PUTCHAR יעבור דרך המחסנית ולא כרגיסטר.

```
STA SEGMENT STACK
      DB 20 DUP (0)
STA ENDS

CODE SEGMENT
      ASSUME CS:CODE, SS:STA
START:
      MOV AX, 'A'
      PUSH AX
      CALL PUTCHAR
```

```
      MOV AX, 'B'
      PUSH AX
      CALL PUTCHAR
      MOV AX, 'C'
      PUSH AX
      CALL PUTCHAR

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

PUTCHAR:
      POP BX
      POP DX
      MOV AH, 2
      INT 21H
      PUSH BX
      RET

CODE ENDS
END START
```

דוגמא 2:

התוכנית הבאה כוללת פונקציה בשם PUTS אשר מקבלת מצביע אל מחרוזת דרך המחסנית, ומדפיסה אותו על המסך.

```
DATA SEGMENT
      STR1 DB 'Hello, World$'
DATA ENDS

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

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

      MOV CX, OFFSET STR1
      PUSH CX
      CALL PUTS

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

PUTS:
      POP BX
```

```
      POP DX
      MOV AH, 9
      INT 21H
      PUSH BX
      RET

CODE ENDS
END START
```

העברת משתנים בעזרת משתנים

בשיטה זו נעביר את הפרמטרים לשגרה על ידי משתנים. לדוגמא, השגרה תצפה תמיד כי פרמטר השיגרה יהיה תמיד בכתובת x וכן כי התוצאה שלה תשמר ב-y. בשיטה זו אין למעשה העברת פרמטרים.

יתרונות השיטה:

- הכי פחות פעולות - הכתובת איתן אנו עובדים הן קבועות.

חסרונות השיטה:

- כל החסרונות של משתנים גלובליים המוכרים בשפות עיליות.

<u>דרך עבודה מומלצת</u>

פרמטרים: קלט במחסנית, פלט ברגיסטר.

EOF