תקציר מספר 5

המחסנית

מחסנית באופן כללי היא מבנה נתונים המאחסן ומנפק נתונים בסדר LIFO - Last In First Out נאי"ר - נכנס אחרון יוצא ראשון. המוסכמה היא שהפעולות על מחסנית נקראים PUSH ו-POP. לדוגמא הסידרה הבאה של פעולות מחסנית יוצרת (פולטת) את הסידרת התוים 'CEDBFA':

```
PUSH('A')
PUSH('B')
PUSH('C')
POP
PUSH('D')
PUSH('E')
POP
POP
POP
PUSH('F')
POP
POP
```

למחסנית כמבנה נתונים יש שימושים שונים - למשל ביטויים אריתמטיים מממושים ע"י מחסנית. אבל אנחנו נדון כאן במחסנית המערכת - מחסנית שיש לה תמיכה בחומרה (בתוך ה-CPU). המחסנית הזאת נועדה למימוש גורמים שונים במערכת ההפעלה - כמו העברת פרמטרים, משתנים לוקליים אוטומטיים, רקורסיה ותהליכים. השימוש במחסנית לא יהיה רק מחסנית כפשוטו, כלומר לא רק ע"י הפקודות PUSH ו-POP בלבד.

התמיכה למחסנית המערכת בחומרה

ב-8086, המחסנית ממומשת בזכרון הרגיל של המחשב. זאת בניגוד למחשבים מסוימים אחרים, שלמחסניות שלהם זכרון מיוחד משלהן.

עיקר התמיכה בחומרה למחסנית היא כלהלן:

אוגר סגמנט SS.

אוגר מצביע SP.

אוגר מצביע BP.

וכן פקודות מכונה PUSH, POP, PUSHF, POPF.

ניהול המחסנית

המחסנית ב-8086 מנוהלת באופן הבא:

נניח שבראש התוכנית מופיע

```
.STACK 100h
```

אזי עם התחלת הריצה, המערכת דואגת להיווצרות המצב הבא:

כלומר ל-SP יש את ה-OFFSET של הבית הראשון שלאחר המחסנית.

פקודות ניהול המחסנית

אופרנד PUSH - קודם באופן אוטומטי SP = SP - 2.
ואחר כך דוחף מילה (2 בתים) למחסנית.

לדוגמא:

```
PUSH AX

PUSH Mem16

PUSH 1053
```

האופרנד היחיד יכול להיות אוגר 16 ביט או זכרון 16 ביט או קבוע 16 ביט.

אופרנד POP - קודם שולף מילה (2 בתים) מהמחסנית
ואחר כך באופן אוטומטי SP = SP + 2.

לדוגמא:

```
POP AX

POP Mem16
```

האופרנד היחיד יכול להיות אוגר 16 ביט או זכרון 16 ביט.

PUSHF - ללא אופרנדים - קודם באופן אוטומטי SP = SP - 2.
ואחר כך דחיפת אוגר הדגלים (Flag Register)
לתוך המחסנית.

POPF - ללא אופרנדים - קודם שליפת אוגר הדגלים (Flag Register)
מתוך המחסנית.
אחר כך באופן אוטומטי SP = SP + 2.

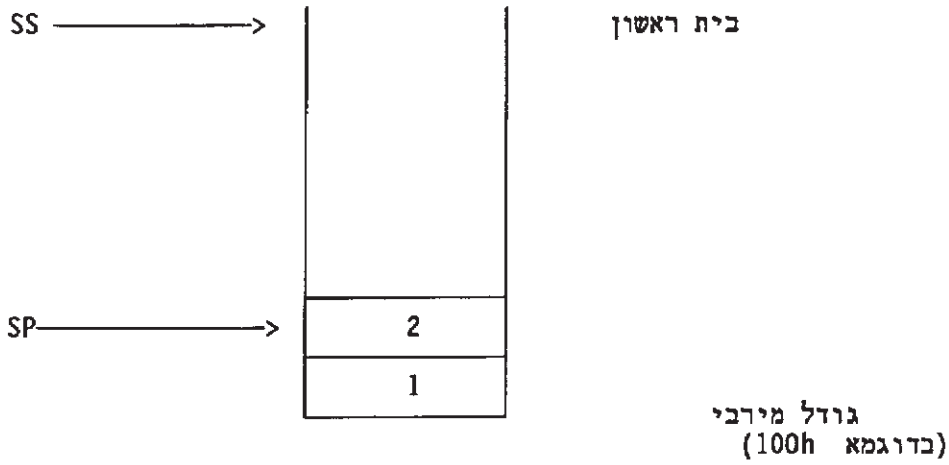
כפי שמשתמע לעיל ב-286, 8086 PUSH ו-POP תמיד 2 בתים. אי אפשר לעשות PUSH ל-byte אחד.

דבר נוסף הוא שהמחסנית מתמלאת מהכתובות הגבוהות לכיוון הכתובות הנמוכות. להמחשה, כאשר המחסנית מלאה נתונים עד הסוף, SP = 0. כאשר המחסנית ריקה מנתונים (כל הזכרון של המחסנית פנוי לשימוש) - למשל בתחילת הריצה של התוכנית - הגודל המירבי של המחסנית = SP.

לדוגמא, אם נבצע את הפעולות הבאות:

```
MOV DX,0102h
PUSH DX
```

אז המחסנית תראה כך:



זאת משום שב-x86 הבית המשמעותי פחות מקדים תמיד את המשמעותי יותר (במובן מסוים ההיפך מאיך שלנו מקובל לחשוב).

כלומר, SP תמיד מצביע על המילה הראשונה לשליפה (תא לא פנוי ראשון).

המחסנית ב-386 ואילך

כל המתואר קודם ממשיך כמובן להתקיים. האוגרים SS, SP ו-BP ממשיכים להתקיים, וכמו כן הפקודות PUSH, ו-POP על אופרנדים 16 ביט.

ב-386 נוספו הדברים הבאים:

1. האוגרים SP ו-BP הורחבו ל-32 ביט ESP ו-EBP. לפיכך ההיסטים של המחסנית יכולים להיות (אם מעונינים בכך) בגודל 32 ביט (במגבלות שציינתי קודם, כלומר ב-DOS בדרך כלל רק עד 65536).
2. פעולות הדחיפה ושליפה מהמחסנית יכולים להיות על, בנוסף ל-16 ביט, גם 32 ביט (4 בתים) של אינפורמציה.

הפקודות PUSH, POP תומכת גם פעולות דחיפה של מילים כפולות (DWORD):

לדוגמא:

```
PUSH EAX

PUSH Mem32

PUSH 100000

POP EAX

POP Mem32
```

כאן בצורה דומה,

PUSH מפחית ESP = ESP - 4 לפני כתיבה,

POP קודם קורא ואח"כ מקדם ESP = ESP + 4.

קיימות גם פקודות

PUSHFD, POPFD ששומרים את אוגר הדגלים המורחב 32bit.

יש גם פקודות מחסנית נוספות (PUSHAD, POAD, ENTER, LEAVE) שלא ניכנס אליהן כאן.

<u>קריאת המחסנית ללא שליפה</u>

תהליך <u>חיוני</u> <u>בשפה עלית כמו C</u> שבו <u>הפרמטרים והמשתנים הלוקליים</u> ממומשים

במחסנית ויש צורך לקרוא ולכתוב את תוכן המחסנית בדרך שאינה שליפה ודחיפה.

ב-8086 הדבר נעשה בעיקר דרך האוגר BP.

התהליך הוא

```
MOV BP,SP
```

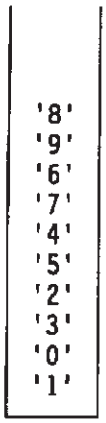
קדם או הפחת את BP לפי הצורך והתיחס
דרכו לזכרון.

[BP + k], [BP - k] התיחסות יחסית ל-SS במחסנית עם ל-BP כ-offset.

תוכנית דוגמא stack3.asm

התוכנית stack3.asm עושה משהו דומה ל-memory4.asm רק שחלק מהתוכן המודפס מועתק מהמחסנית. המטרה של התוכנית הזו היא להמחיש את האפשרויות של קריאת מידע מתוך המחסנית ללא שליפה (שימוש ב-POP) באמצעות האוגר BP המשמש פוינטר טבעי למחסנית.

שים לב שבשל האופי המיוחד של פעולת ה-PUSH והעובדה שהמחסנית הזו מתמלאת מהכתובות הגבוהות לנמוכות תוכן המחסנית עם סיום לולאת ה-PUSH-ים (StackStore) נראה ככה:



שנים מהלולאות (StackLoop1 ו-StackLoop2) של התוכנית הם סריקות (בשני הכיוונים) של המחסנית באמצעות BP וזה מסביר את תוצאות הפלטים שלהם ("8967452301" ו-"1032547698" בהתאמה).

```
;
;   stack3.asm - demonstrate hardware stack usage.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
Separator        EQU '#'
DisplayString    DB  64 DUP('$')
Numbs            DB '0123456789'
Stack_Size       DW (?)
      .CODE
ProgStart:
      MOV AX,@DATA                      ; Set DS to point ...
      MOV DS,AX                         ; ... to data segment
                                        ;
      MOV DI,OFFSET DisplayString       ; Have DI point to start ...
                                        ; ... of DisplayString
      MOV Stack_Size,SP                  ; Save stack size
                                        ;
      XOR BX,BX                         ;  BX := 0;
      MOV CX,5                          ;  Size(Numbs) = 2*5
                                        ;
StackStore:
        MOV DX,WORD PTR Numbs[BX]       ; Read two bytes in Numbs
        PUSH DX                         ; Store in Stack
        MOV [DI],DX                     ; Store in DisplayString
        ADD BX,2                        ; BX point to next word in Numbs
        ADD DI,2                        ; DI point to next available position
       LOOP StackStore                  ; Repeat 5 times
                                        ;
      MOV DL,Separator                  ;
      MOV [DI],DL                       ; Store '#' ...
      INC DI                            ;   ... to separate
                                        ;
      MOV CX,Stack_Size                 ;
      SUB CX,SP                         ; Set CX to equal ...
                                        ;  ... number of relevant ...
                                        ;  ... bytes in stack
     MOV BP,SP                          ;
StackLoop1:                             ; Print Stack downward
        MOV BL,[BP]                     ;
        INC BP                          ;
        MOV [DI],BL                     ; Store in DisplayString
        INC DI                          ; point to next available position
     LOOP StackLoop1                    ;
                                        ;
      MOV DL,Separator                  ;
      MOV [DI],DL                       ; Store '#' ...
      INC DI                            ;   ... to separate
                                        ;
      MOV BX,[BP-3]                     ;  Retrieve eighth and ninth bytes ...
      MOV [DI],BX                       ;   ... in stack
      ADD DI,2                          ;
                                        ;
      MOV DL,Separator                  ;
      MOV [DI],DL                       ; Store '#' ...
      INC DI                            ;   ... to seperate
                                        ;
```

```
      MOV BP,SP                          ;  Retrieve third and forth bytes ...
      MOV BX,[BP+2]                      ;    ... in stack
      MOV [DI],BX                        ;
      ADD DI,2                           ;
                                         ;
      MOV DL,Separator                   ;
      MOV [DI],DL                        ; Store '#' ...
      INC DI                             ;   ... to seperate
                                         ;
      MOV CX,Stack_Size                  ;
      SUB CX,SP                          ; Set CX to equal ...
                                         ;  ... number of relevant ...
                                         ;  ... bytes in stack
      MOV BP,Stack_Size                  ;
      DEC BP                             ; Set BP to point to last ...
                                         ; ... byte in stack
StackLoop2:                              ; Print stack, upward
         MOV BL,[BP]                     ;
         DEC BP                          ;
         MOV [DI],BL                     ; Store in DisplayString
         INC DI                          ; point to next available position
      LOOP StackLoop2                    ;
                                         ;
      MOV AH,9                           ; Set print option for int 21h
      MOV DX,OFFSET DisplayString        ;  Set  DS:DX to point to DisplayString
      INT 21h                            ;  Print DisplayString
                                         ;
      MOV AH,4Ch                         ; Set terminate option for int 21h
      INT 21h                            ; Return to DOS (terminate program)
      END  ProgStart
```

```
E:\>stack3
0123456789#8967452301#30#67#1032547698
E:\>
```