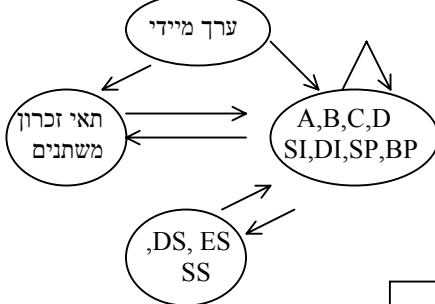


| | | | | | |
|---|--|-----------------|--|---------|---|
| CI [flag name] | איפוס דגל | Mov dest , src | העתקת הערך ב- src ל- dest | | |
| St [flag name] | הדלקת דגל | Add dest , src | dest = dest + src | | |
| Xchg reg, sec | החלפת ערכים בין אוגר לזיכרון חייב להיות לפחות אוגר אחד | Adc dest , src | dest = dest + src + CF | | |
| | | Inc src | Src = src + 1 | | |
| | | Sub dst , src | dest = dest - src | | |
| | | Sbb dest , src | dest = dest - src - CF | | |
| Lea dest,src | טעינת ה offset של src אל dest | Dec src | src = src - 1 | | |
| Ret (pop ip) | חזרה מפו' שמוגדרת near | Mul src | Src = word - dx ax = word * ax Src = byte - ax = al * src | | |
| Retf (pop ip , cs) | חזרה מפו' שמוגדרת far | Div src | Byte - ax / byte = al , ah (remainder) Word - (dxax)/word = ax , dx (remainder) | | |
| Iret (pop ip ,cs ,f) | חזרה מפסיקה | And dest , src | dest = dest and src | | |
| | | Or dest , src | dest = dest or src | | |
| | | Xor dest , src | dest = dest xor src | | |
| | | Not src | Src = not src | | |
| | | Neg src | פעולת המשלים ל- r | | |
| | | Lea dest , src | לוקח את הכתובת של src ומכניס אותה ל- dest | | |
| | | Cmp dest , src | פעולת חיסור בין dest ל- src ועדכון הדגלים בלבד ללא שינוי ערכי האופרנדים | | |
| | | Test dest , src | פעולת and בין dest ל- src ועדכון הדגלים בלבד ללא שינוי ערכי האופרנדים | | |
| הוראות LOOP | | | | | |
| ההוראה loop מורידה את cx באחד ובודקת אם הוא שונה מאפס | | | | | |
| Loop label | | | כל עוד $cx \neq 0$ תישאר בלולאה ברגע ש $cx = 0$ תצא מהלולאה | | |
| LoopZ/E label | | | כל עוד $cx \neq 0$ וגם $zf=1$ תישאר בלולאה ברגע ש- $cx=0$ או $zf=0$ תצא מהלולאה | | |
| LoopNZ/NE label | | | כל עוד $cx \neq 0$ וגם $zf=0$ תישאר בלולאה ברגע ש- $cx=0$ או $zf=1$ תצא מהלולאה | | |
| הוראות התניה עבור מספרים עם סימן | | | | | |
| JL | | | קפוץ אם קטן ממש | | |
| JG | | | קפוץ אם גדול ממש | | |
| JLE | | | קפוץ אם קטן שווה | | |
| JGE | | | קפוץ אם גדול שווה | | |
| JE | | | קפוץ אם שווה | | |
| JNE | | | קפוץ אם לא שווה | | |
| הוראות התניה עבור מספרים ללא סימן | | | | | |
| JB | | | קפוץ אם קטן ממש | | |
| JA | | | קפוץ אם גדול ממש | | |
| JBE | | | קפוץ אם קטן שווה | | |
| JAЕ | | | קפוץ אם גדול שווה | | |
| JE | | | קפוץ אם שווה | | |
| JNE | | | קפוץ אם לא שווה | | |
| הוראות התניה מושפעות מדגלים | | | | | |
| jnz | Zf = 0 | jz | Zf = 1 | Jnp/jpo | Pf = 0 |
| jnc | Cf = 0 | Jc | Cf = 1 | Jp/jpe | Pf = 1 |
| הוראות הזזה (בכל הוראות ההזזה counter== 1 or CL) | | | | | |
| SHL dest,counter | | | | | הזזת כל הביטים שמאלה אל ה- carry ומימין הוספת אפסים |
| SHR dest,counter | | | | | הזזת הביטים ימינה ומשמאל הכנסת אפסים |
| SAR dest,counter | | | | | הזזת הביטים ימינה ושכפול ביט הסימן |
| ROL dest,counter | | | | | הזזת הביטים שמאלה כשהביט השמאלי עובר גם לביט הימני וגם ל- carry |
| ROR dest,counter | | | | | הזזה ימינה, כשהביט הימני עובר גם לביט השמאלי וגם ל- carry |
| RCL dest,counter | | | | | הזזת הביטים שמאלה דרך ה- CF |
| RCR dest,counter | | | | | הזזת הביטים ימינה דרך ה- CF |

| אופרטורים | |
|---|---|
| PTR | Casting לפויינטר דוג: word PTR 5 |
| EQU | הגדרת קבועים Max EQU 100 |
| = | מתן ערך מספרי לסימבול, את הערך ניתן לשנות במהלך התוכנית הסימבול לא תופס מקום בזכרון $N = 3 \dots N = 5$ |
| Local var | נותן כתובת ל-label בכל קריאה ל-macro |
| Seg X | מחזיר את תחילת כתובת המקטע שבו הוגדר X |
| Type X | מחזיר את טיפוס המשתנה Db = 1 Dw = 2 |
| Length X | מחזיר את מספר השכפולים דוג: X db 100 dup (3,5,7) mov cx, length x cx = 100 |
| Size X | Size x = length x * type x |
| \$ | \$ הוא הכתובת של השורה הנוכחית הכתובת של len פחות הכתובת של X : Len dw \$ -x |
| & | שרשור בין מחרוזות בתוך מאקרו |
| %exp | מעביר את הערך שיש ב-exp ולא את exp עצמו |
| Label Exmp' : Dor label word X db 3,5,1 | בדוגמא dor הוא מצביע מסוג word על x שלא תופס מקום בזכרון ועד סוף המקטע הוא מצביע על word. לכן יתקיים: mov ax,d[1] →→ ax←0105 |
| exp1 EQ exp2 | -1 exp1 = exp2 0 אחרת |
| exp1 NE exp2 | -1 exp1 ≠ exp2 0 אחרת |
| exp1 LT exp2 | -1 exp1 < exp2 0 אחרת |
| exp1 LE exp2 | -1 exp1 ≤ exp2 0 אחרת |
| exp1 GT exp2 | -1 exp1 > exp2 0 אחרת |
| exp1 GE exp2 | -1 exp1 ≥ exp2 0 אחרת |
| MACRO מותנה | |
| If exp endif | אם exp ≠ 0 נכנס ל-if |
| If exp Else endif | אם exp ≠ 0 נכנס ל-if אם exp = 0 נכנס ל-else |
| Ife exp Endif | אם exp = 0 נכנס ל-if |
| Ife exp Else endif | אם exp = 0 נכנס ל-if אם exp ≠ 0 נכנס ל-else |
| Ifb <arg> Else endif | אם arg = blank נכנס ל-if אם arg ≠ blank נכנס ל-else |
| Ifnb <arg> Else endif | אם arg ≠ blank נכנס ל-if אם arg = blank נכנס ל-else |

| | |
|--|---|
| Ifidn <arg1> , <arg2> Else endif | אם $arg1 = arg2$ נכנס ל- if אם $arg1 \neq arg2$ נכנס ל- else $arg2 - arg1 = arg2$ זהים בשם ז"א התאמת מחרוזות. |
| Ifdif <arg1> , <arg2> Else endif | אם $arg1 \neq arg2$ נכנס ל- if אם $arg1 = arg2$ נכנס ל- else $arg2 - arg1 = arg2$ זהים בשם. ז"א התאמת מחרוזות |
| עבור התנאים: לא צריך " " עבור מחרוזות ואין לרשום אותן בתוך <> אבל אם משתמשים במשתנה מאקרו אז צריך לרשום אותו בתוך <> | |
| הוראות משכפלות | |
| Rept exp endm | שכפל את הבלוק כמספר הפעמים שמופיע ב- exp |
| Irp va, < arg1 , arg2 , ...,argN > endm | שכפל את הבלוק כמספר הארגומנטים כאשר בכל שכפול va מקבל ארגומנט אחר. |
| Irpc va , string Endm | שכפל את הבלוק כמספר התווים שיש ב- string כאשר בכל פעם va מקבל תו אחר |
| Exitm | מוציאה אותנו מהוראה משכפלת כמו : Rept , Irp , Irpc |
| הוראות מחרוזות | |
| Cld | $Df = 0$ - נבצע + ל- SI ו- DI |
| Std | $Df = 1$ - נבצע - ל- SI ו- DI |
| Movsb Movsw | הבית או המילה שבכתובת DS:SI מועתקת לכתובת ES:DI הרגיסטרים SI ו- DI מתעדכנים בהתאם |
| Rep | חזור על הוראת המחרוזות שאחרי הוראה זו כל עוד cx שונה מאפס והפחת 1 מ- cx בכל מחזור לפני הבדיקה אם $cx=0$ אז לא תבצע פקודת המחרוזות |
| Stosb stosw | הבית או המילה שברגיסטר A מועברת לכתובת ES:DI ו- DI מתעדכן בהתאם |
| Lodsb Lodsw | הבית או המילה שבכתובת DS:SI מועברת לרגיסטר A, (אם מדובר ב- lodsb מועבר ל- AL אם מדובר ב- lodsw מועבר ל- AX) SI מתעדכן בהתאם |
| Cmpsb Cmpsw | הבית או המילה שבכתובת ES:DI מחוסרת מזו של DS:SI ורק הדגלים נקבעים בהתאם. SI ו- DI מתעדכנים בהתאם DS:SI-ES:DI |
| Scasb Scasw | הבית או המילה שבכתובת ES:DI מחוסר מ- AL או AX והדגלים נקבעים בהתאם. DI מכוון כתוצאה מהפעולה |
| RepE/Z | כל עוד $cx \neq 0$ וגם $zf=1$ תמשיך בשכפול ברגע ש- $cx = 0$ או $zf=0$ צא מהשכפול |
| RepNE/NZ | כל עוד $cx \neq 0$ וגם $zf=0$ תמשיך בשכפול ברגע ש- $cx = 0$ או $zf=1$ צא מהשכפול |

| | | |
|---|--|---|
| <pre> נתונים שני מערכים אחד בשם text והשני בשם check עליך לבדוק כמה פעמים מערך check מופיע בתוך מירך text ולהחליף ב- text את check ב- check1 .model small .stack 100h .data N1 equ 200 N2 equ 3 text db N1 dup (?) check db N2 dup (?) check1 db N2 dup (?) counter db 0 .code start: mov ax , @data mov ds , ax mov es , ax cld lea di , text mov cx , N1 again : cmp cx , N2 jl exit lea si , check lodsb repne scasb cmp cx , N2 - 1 jl exit push cx push di mov cx , N2 - 1 repe cmps jnz cont lea si , check1 pop di dec di mov cx , N2 rep movsb pop cx sub cx , N2 inc counter jmp again cont : pop di pop cx jmp again exit : mov ax , 4c00h int 21h end start </pre> | <p>הדפסת מחרוזת : (לא לשכוח לשים בסוף \$) '.....\$' Infor db</p> <pre> Mov dx , offset infor Mov ah , 9 Int 21h </pre> <p>קריאה מטבלת הפסיקות את פסיקה k:</p> <pre> Mov ax , 0 Mov es , ax Mov bx , es :[4k] Mov es , es:[4k+2] Cli :טביחה לטבלת הפסיקות Mov ax , 0 Mov es , ax Mov dx , offset new_int Mov es:[4k] , dx Push cs Pop ax Mov es:[4k+2] , ax Sti </pre> <p>טביחה ע"י שימוש ב dos:</p> <pre> Push cs Pop ds Mov dx,offset new_int Mov ah,25h Mov al,k ; הוא הוקטור שאליו נכתוב Int 21h </pre> | <p>אוגר הדגלים:</p> <p><u>zero flag - Zf</u> - ערכו שווה ל-1 אם התוצאה עקב פעולה אריתמטית לוגית שווה לאפס, אחרת ערכו שווה לאפס.</p> <p><u>sign flag - SF</u> - האם התוצאה שקיבלנו היא חיובית או שלילית. Sf=0 חיובי sf=1 שלילי</p> <p><u>parity flag - PF</u> - דגל הזוגיות – בודק את מספר האחדות במספר. Pf=1 מס' האחדות זוגי Pf=0 מס' האחדות אי זוגי</p> <p><u>carry flag - CF</u> - אם חיברנו שני אופרנדים שכל אחד בגודל x וקיבלנו תוצאה בעלת x+1 ביטים אז הביט ב- x+1 הוא ה-CF</p> <p><u>OVF</u> - דגל הגלישה – מציין האם פעולות אריתמטיות לוגיות לא הגיוניות. לדוג, חיבור שני מספרים חיוביים וקבלת מספר שלילי. מימוש: ניקח את ה-CF ונחבר אותו בשער XOR עם ה-CF הקודם.</p> |
| <pre> יצירת 8 משתנים tow1,...,tow8 הערכים 2,4,8,...,256 (ex=0, tp=1) Mac1 macro Rept 8 ex = ex + 1 tp = tp * 2 Mac2 %ex , %tp Endm Endm Mac2 macro lc , val tow & lc dd val Endm </pre> | <p>מספרה של רוטינת פסיקת שרון הוא 8h והיא נקראת 18.2 פעמים בשניה (כל 55 אלפיות השניה). ברוטינה זו מופיעה קריאה לרוטינת שירות פסיקה 1ch שבה יש פקודה בודדת IRET, מטרתה שהמשתמש יוכל להשתלט על רוטינת שירות פסיקת השרון. בנוסף ישנו מונה בגודל dw בכתובת 40:6c והוא סופר את מספר הפעמים שקראנו לרוטינת שירות הפסיקה מחצות.</p> <p>מאקרו בשם push שדוחף למחסנית כמה רגיסטרים שרוצים</p> <pre> Pushh macro r1,r2,r3,r4 ,.... Irp reg , <r1,r2,...> Ifb <reg> Exitm Else Push reg Endif Endm endm </pre> <p>בדיקה האם שני מערכים הפוכים זה לזה</p> <pre> start : mov ax , @data mov ds ,ax mov es , ax mov cx , N mov si , offset x mov di , offset y + n-1 again : cld lodsb std scasb jnz exit loop again mov flag , 1 exit : </pre> | <p>5 ספרות → Address 8086 – 20 bits 4 ספרות → Seg , offset – 16 bits</p> <p>מכפילים ב-10h את כתובת הסיגמנט כדי להוסיף 4 אפסים מימין (כדי להגיע ל-20 ביטים של ה-address) גודל כל מקטע : 2 בחזקת מס' הביטים של ה-offset</p> <p>דוג:</p> <pre> Seg = 6d04 Offset = 0004 PA = 6d04*10h + 0004h = 6d044h </pre> <p>סיפור ההוראה - mov var1 , var2</p> <pre> IS_A_REGISTER macro reg Isreg = 0 Irp va , < ax ,bx,cx,...,al,ah ... > Ifidn <va> , <reg> Isreg = 1 Exitm Endif Endm Endm Mov macro to , from IS_A_REGISTER to If isreg Mov to from Else IS_A_REGISTER from If isreg Mov to , from Else Push ax Ife type to -1 Mov al , from Mov to , al Else Mov ax , from Mov to , ax Endif Pop ax Endif Endif endm </pre> |

| | | |
|---|--|--|
| <p><u>רקורסיה :</u></p> <p>Name proc near בדיקה לתנאי עצירה Jmp not Re_call ערכים התחלתיים ללולאה השניה Jmp done Re_call : push Call Name Pop Done : ret</p> |  | <p><u>טיפוסי משתנים</u></p> <p>DB - משתנה בגודל 1 בייט DW - משתנה בגודל 2 מילים DD - משתנה בגודל 4 בתים DQ - משתנה בגודל 8 בתים DT - משתנה בגודל 10 בתים</p> <p>אם משתנה או רגיסטר מופיעים בסוגריים ריבועיים הם מאבדים את הטיפוס שלהם.</p> |
|---|--|--|

```

נתונה רשימה מקושרת הבנויה מזוגות
מילים. המילה הראשונה מכילה ערך
בזוג והמילה השניה מצביעה לזוג הבא.
סוף הרשימה -1 בכתובת. מצאו באופן
רקורסיבי את המספר המינימלי
והמקסימלי ואת מספר האפסים.
.model small
.stack 100h
.....
start : mov bx list
        call res
        mov ax , 4c00h
        int 21h
        end start

```

תוכנית המבצעת n! ברקורסיה
המספר יועבר דרך bx

```

N_prod proc near
    Cmp bx , 1
    Jg Re_call
    Mov ax , 1
    Jmp done
Re_call : push bx
          Dec bx
          Call N_prod
          pop bx
          Mul bx
Done :   ret

```

תוכנית שמעתיקה מערך X למערך Y
(מחרוזות)

```

.model small
.stack 100h
.data
x db 100 dup (?)
y db 100 dup (?)
.code
start : mov ax , @data
        mov ds , ax
        mov es , ax
        cld
        lea si , x
        lea di , y
        mov cx , 100
        rep movsb
        mov ax , 4c00h
        int 21h
        end start

```

```

Res proc near
Cmp bx , -1
Jne re_call
Mov bp , sp
Mov dx , [bp+2]
Mov min , dx
Mov max , dx
Mov counter , 0
Jmp done
re_call : mov dx , [bx]
          push dx
          mov bx , [bx+2]
          call Res
          pop dx  cmp max , dx
          jge cont
          mov max , dx
cont :    cmp min , dx
          jle cont1
          mov min , dx
cont1 :  cmp dx , 0
          jne done
          inc counter
done :   ret
Res endp

```

נתונים 3 משתנים x,y,z עליך למיין
אותם כשהמיון יתבצע דרך מקרו
המקרו : (תוצאה: x<=y<=z)

```

Sort macro var1,var2,reg
    Local cont
    Mov reg , var2
    Cmp var1 , reg
    Jle cont
    Xchg var1 , reg
    Mov var2 , reg
Cont : nop
Endm

```

התוכנית:

```

.data
x dw ?
y dw ?
z dw ?
.code
start : mov ax , @data
        mov ds , ax
        sort x , y , ax
        sort y , z , ax
        sort x , y , ax
        mov ax , 4c00h
        int 21h
        end start

```

כתיבה לטבלת הפסיקות :

```

Code segment
Assume cs:code , ds:code
Original_int16h dd ?
New_resident : שם ה tsr שיישאר בזכרון
                Cli
                Pushf
                Push ax
                Push es
                Mov ax , 0
                Mov es , ax
                Test byte ptr es:[417h] , 40h
                Pop es
                Pop ax
                Jnz go_usual
                Sti
                Popf
                Iret
Go_usual :
                sti
                Popf
                Jmp dword ptr cs : original_int16h
Init_resident : (מקביל ל start)
                mov ax , code
                mov ds , ax
                mov ax , 3516h
                int 21h ;get int 16h vector at ES:BX
                שמירת כתובת פסיקת המקלדת במשתנה עזר;
                mov word ptr original_int16h , bx
                mov word ptr original_int16h+2 , es
                שינוי וקטור הפסיקה 16h;
                mov dx , offset new_resident
                mov ax , 2516h
                int 21h
                Mov dx , offset init_resident ;tsr קיבוע ה-

```

הדפסת תו :

```

Mov bx , 0
Mov ah , 14
Mov al , the char
Int 10h

```

קליטת תו :

```

Mov ah , 0
Int 16h

```

בסיום התו יופיע ב al

המשך :

```

Mov cl , 4
Shr dx , cl
Add dx , 20
Mov ax , 3100h
Int 21h
Code ends
End init_resident

```