

```

; A 100 elemű array nevű tömbnek van-e
; 3-tól különböző eleme?
mov     cx, 100
mov     di, -1 ; előbb lehessen inc, mint cmp
mov     al, 3
NEXT:   inc     di
        cmp     array[di], al ; array di-edik eleme = 3?
        LOOPE  NEXT ; ugrás NEXT-re,
                    ; ha CX≠0 és a di-edik elem=3
        JNE    NEM3 ; CX = 0 vagy array[di] ≠ 3
        ...
        ...
        ...
NEM3:   ... ; di az első 3-tól különböző
        ... ; elem indexe
    
```

Máté: Assembly programozás 5. előadás 1

Ugyanennek a feladatnak a megoldása string kezelő utasítás segítségével:

```

; A 100 elemű array nevű tömbnek van-e
; 3-tól különböző eleme?
        mov     cx, 100
        mov     di, offset array
        mov     AL, 3
        REPE   SCAS array ; array 0., 1., ... eleme = 3?
        JNE    NEM3
        ...
        ...
        ...
NEM3:   DEC     DI ; DI az első ≠ 3 elemre mutat
        ...
    
```

Máté: Assembly programozás 5. előadás 2

A 100 elemű array nevű tömbnek van-e 3-tól különböző eleme?

```

mov     cx, 100
mov     di, -1
mov     al, 3
NEXT:   inc     di
        cmp     array[di], al
        LOOPE  NEXT
        JNE    NEM3
        ...
        ...
        ...
NEM3:   ...
        ...
    
```

```

mov     cx, 100
mov     di, offset array
        REPE   SCAS array
        JNE    NEM3
        ...
        ...
        ...
NEM3:   DEC     DI
        ...
    
```

Használja ES-t. Sokkal gyorsabb volt.

Nem minden eltérés lényeges!

Máté: Assembly programozás 5. előadás 3

Egyszerűsített lexikális elemző

Feladata, hogy azonosító, szám, speciális jelek és a program vége jel előfordulásakor rendre A, 0, , és . karaktert írjon a képernyőre. Az esetleges hibákat ? jelezze.

XLAT utasítás alkalmazásának tervezése:

Karakter típusok	karakterek	kód
Betű	A ... Z a ... z	2
Számjegy	0 ... 9	4
Speciális jel	, . tabulátor ; + - () cr lf	6
Vége jel	\$	8
Hibás karakter	a többi	0

Máté: Assembly programozás 5. előadás 4

```

data    segment    para public 'data'

; ugró táblák a szintaktikus helyzetnek megfelelően:

; kezdetben, speciális és hibás karakter után
;
; következő karakter
t_s     dw  hiba    ; hibás kar.: ? → spec. jel szint
        dw  lev_a   ; betű: A → azonosító szint
        dw  lev_n   ; számjegy: 0 → szám szint
        dw  lev_s   ; spec. jel: , → spec. jel szint
        dw  vege    ; $: . → program vége
    
```

Máté: Assembly programozás 5. előadás 5

```

; azonosító szint
t_a     dw  hiba    ; hibás kar.: ? → spec. jel szint
        dw  ok      ; betű: nincs teendő
        dw  ok      ; számjegy: nincs teendő
        dw  lev_s   ; speciális jel: , → spec. jel szint
        dw  vege    ; $: . → program vége

; szám szint
t_n     dw  hiba    ; hibás kar.: ? → spec. jel szint
        dw  hiba    ; betű: hiba: ? → spec. jel szint
        dw  ok      ; számjegy: nincs teendő
        dw  lev_s   ; speciális jel: , → spec. jel szint
        dw  vege    ; $: . → program vége
    
```

Máté: Assembly programozás 5. előadás 6

```

level    dw    ?           ; az aktuális ugrótábla címe
c_h     db    0           ; hibás karakter kódja
c_b     db    2           ; betű kódja
c_n     db    4           ; számjegy kódja
c_s     db    6           ; speciális jel kódja
c_v     db    8           ; végjel kódja
specjel db    ',. ;+()-', 13, 10 ; a speciális jelek
vegjel  db    '$'        ; vége jel, kihasználjuk,
                        ; hogy itt van!
table   db    256 dup (?) ; átkódoló tábla (256 byte)
text    db    'a,tz.fe&a 21 a12; 12a $' ; elemzendő szöveg
data    ends

```

Máté: Assembly programozás 5. előadás 7

```

code     segment para public 'code'
         assume  cs:code, ds:data, es:data, ss:stack
lex      proc    far
         push   ds
         xor    ax,ax
         push  ax ; visszatérési cím a veremben
         mov   ax,data
         mov   ds,ax
         mov   es,ax ; assume miatt
         call  prepare ; átkódoló tábla elkészítése
         mov   si, offset text ; az elemzendő szöveg
                        ; kezdőcíme
         call  parsing ; elemzés
         ret   ; vissza az Op. rendszerhez
lex      endp

```

Máté: Assembly programozás 5. előadás 8

```

prepare  proc    ; az átkódoló tábla elkészítése
; az eljárás rontja ax, bx, cx, di, si tartalmát
         cld    ; a string műveletek iránya pozitív
         mov   bx, offset table
         mov   di, bx
         mov   al, c_h ; hibás karakter kódja
         mov   cx, 256 ; a tábla hossza
         rep  stos table; table ← minden karakter hibás

```

Máté: Assembly programozás 5. előadás 9

```

         mov   al, c_b ; betű kódja
         mov   di, 'A' ; A ASCII kódja
         add  di, bx ; A helyének offset címe
         mov  cx, 'Z'-'A'+1 ; a nagybetűk száma
                        ; a betűk ASCII kódja folyamatos!
         rep  stosb
         mov  di, 'a' ; a ASCII kódja
         add  di, bx ; a helyének offset címe
         mov  cx, 'z'-'a'+1 ; a kisbetűk száma
         rep  stosb

```

Máté: Assembly programozás 5. előadás 10

```

         mov  al, c_n ; számjegy kódja
         mov  di, '0' ; 0 ASCII kódja
         add  di, bx ; 0 helyének offset címe
         mov  cx, '9'-'0'+1 ; a számjegyek száma
                        ; a számjegyek ASCII kódja folyamatos!
         rep  stosb

```

Máté: Assembly programozás 5. előadás 11

```

         mov  si, offset specjel; speciális jelek
                        ; feldolgozása
pr1:     xor  ah, ah ; hogy ax=al legyen
         lods specjel ; speciális jel ASCII kódja
         mov  di, ax ; ah=0 miatt ax = a jel kódja
         cmp  al, vegjel ; vegjel közvetlenül a
                        ; speciális jelek után van!
         je   pr2 ; ez már a vegjel
         mov  al, c_s ; speciális karakter kódja
         mov  [bx+di], al ; elhelyezés a táblában
         jmp  pr1 ; ciklus vége

```

Máté: Assembly programozás 5. előadás 12

```

pr2:   mov     al,c_v   ; a végjel kódja
       mov     [bx+di],al ; elhelyezés a táblában
       ret                               ; vissza a hívó eljáráshoz
prepare endp

```

Máté: Assembly programozás 5. előadás 13

```

parsing proc          ; elemzés
; az eljárás rontja ax, bx, cx, di, si tartalmát
       cld          ; a string műveletek iránya pozitív
       mov         bx,offset table
       mov         di,offset t_s ; spec. jel szint
lv1:   mov         level,di ; szint beállítás
       xor         ah,ah   ; hogy ax=al legyen
ok:    lods        text    ; a következő karakter
       xlat        ; al ← 0, 2, 4, 6 vagy 8
       mov         di, level ; di ← az akt. ugrót. cím
       add         di, ax   ; az ugrótáblán belüli cím
       jmp         [di]    ; kapcsoló utasítás

```

Máté: Assembly programozás 5. előadás 14

```

hiba:   mov     di,offset t_s ; hibás karakter,
       ; spec. jel szint következik
       mov     al,'?'
lv2:   mov     ah,14 ; BIOS hívás előkészítése
       int     10h ; BIOS hívás:
       ; karakter írás a képernyőre
       jmp     lv1
lev_a:  mov     di,offset t_a ; azonosító kezdődik
       mov     al,'A'
       jmp     lv2

```

Máté: Assembly programozás 5. előadás 15

```

lev_n:  mov     di,offset t_n ; szám kezdődik
       mov     al,'0'
       jmp     lv2
lev_s:  mov     di,offset t_s ; speciális jel
       mov     al','
       jmp     lv2
vege:   mov     al,'.' ; szöveg vége
       mov     ah,14 ; BIOS hívás előkészítése
       int     10h ; BIOS hívás:
       ; karakter írás a képernyőre
       ret     ; elemzés vége, vissza a hívóhoz
parsing endp
code    ends

```

Máté: Assembly programozás 5. előadás 16

```

stack  segment para stack 'stack'
       dw 100 dup (?) ; 100 word legyen a verem
stack  ends
       end lex ; modul vége, start cím: lex

```

Máté: Assembly programozás 5. előadás 17

Logikai utasítások

Bitenkénti logikai műveleteket végeznek.
1 az igaz, **0** a hamis logikai érték.

AND op1,op2 ; op1 ← op1 & op2, bitenkénti és
TEST op1,op2 ; flag-ek op1 & op2 szerint
OR op1,op2 ; op1 ← op1 | op2, bitenkénti vagy
XOR op1,op2 ; op1 ← op1 ^ op2 (eXclusive OR),
; bitenkénti kizáró vagy
NOT op ; op ← ~op, bitenkénti negáció,
; nem módosítja STATUS tartalmát!

Máté: Assembly programozás 5. előadás 18

Bit forgató (Rotate) és léptető (Shift) utasítások

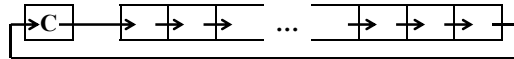
Forgatják (Rotate) illetve léptetik (Shift) **op** tartalmát. A forgatás/léptetés történhet

- 1 bittel,
- vagy byte illetve word esetén a **CL** regiszter alsó 3 illetve 4 bit-jén megadott bittel jobbra (Right) vagy balra (Left).

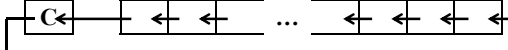
Az utoljára kilépő bit lesz a **Carry** új tartalma.

A rotálás történhet a **Carry**-n keresztül, ilyenkor a belépő bit a **Carry**-ből kapja az értékét:

RCR op,1/CL ; Rotate through Carry Right

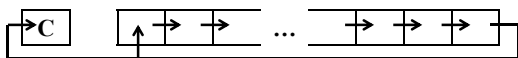


RCL op,1/CL ; Rotate through Carry Left

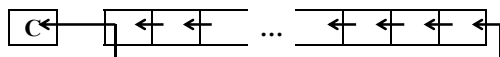


A rotálás történhet úgy, hogy **Carry** csak a kilépő bitet fogadja, a belépő bit értékét a kilépő bit szolgáltatja:

ROR op,1/CL ; Rotate Right

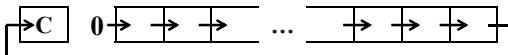


ROL op,1/CL ; Rotate Left



Logikai léptetés jobbra: A belépő bit **0**:

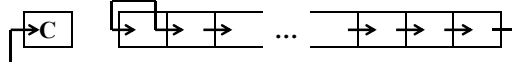
SHR op,1/CL ; Shift Right



Előjel nélküli egész számok **2 hatványával** történő osztására alkalmas.

Aritmetikai léptetés jobbra: A belépő bit **op** előjele:

SAR op,1/CL ; Shift Arithmetical Right



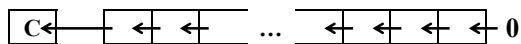
Előjeles egész számok **2 hatványával** történő osztására alkalmas. Negatív számok esetén csali!

Balra léptetéskor a belépő bit mindig **0**:

SHL op,1/CL ; Shift Left

SAL op,1/CL ; Shift Arithmetical Left

SAL ≡ SHL



Előjel nélküli vagy előjeles egész számok **2 hatványával** történő szorzására alkalmas.