Az összeadástól és kivonástól eltérően a szorzás és osztás esetében különbséget kell tennünk, hogy előjeles vagy előjel nélküli számábrázolást alkalmazunk-e. További lényeges eltérés, hogy két 8 bites vagy 16 bites mennyiség szorzata ritkán fér el 8 illetve 16 biten, ezért a szorzás műveletét úgy alakították ki, hogy 8 bites tényezők szorzata 16, 16 biteseké pedig 32 biten keletkezzék:

Szorzásnál **op** nem lehet közvetlen operandus!

MUL op ; előjel nélküli szorzás (MULtiplicate),
IMUL op ; előjeles szorzás (Integer MULtiplicate).

Ha op 8 bites AX ⇐ AL * op.

Ha op 16 bites (DX:AX) ⇐ AX * op.

Osztásnál **op** nem lehet közvetlen operandus!

```
DIV   op   ; (DIVide) előjel nélküli osztás,
IDIV  op   ; (Integer DIVide) előjeles osztás,
           ; A nem 0 maradék előjele megegyezik
           ; az osztandóéval.
```

Ha op 8 bites: AL ⇐ AX/op hányadosa,
AH ⇐ AX/op maradéka.
Ha op 16 bites: AX ⇐ (DX:AX)/op hányadosa,
DX ⇐ (DX:AX)/op maradéka.

Osztásnál **túlcsordulás** → azonnal elhal (abortál) a programunk!

Ha bájtot bájttal vagy szót szóval akarunk osztani, akkor:

- Előjel nélküli osztás előkészítése **AH** illetve **DX** nullázásával történik.
- Előjeles osztás előkészítésére szolgál az alábbi két előjel kiterjesztő utasítás:

```
CBW   ; (Convert Byte to Word)
      ; AX ⇐ AL előjel helyesen
CWD   ; (Convert Word to Double word)
      ; (DX:AX) ⇐ AX előjel helyesen
```

Pozitív számok esetén (az előjel **0**) az előjel kiterjesztés az **AH** illetve a **DX** regiszter nullázását, negatív számok esetén (az előjel **1**) csupa **1**-es bittel való feltöltését jelenti.

Az előjel kiterjesztés máskor is alkalmazható.


```
; Két vektor skalár szorzata. 1. változat
code    segment  para public 'code'
        assume   cs:code, ds:data, ss:stack, es:nothing

skalar  proc     far
        push     ds        ; visszatérési cím a verembe
        xor      ax,ax     ; ax ⇐ 0
        push     ax        ; visszatérés offset címe
        mov      ax,data   ; ds a data szegmensre mutasson
        mov      ds,ax     ; sajnos „mov ds,data"
                           ; nem megengedett
```



```
; A      skalár szorzat számítása
        mov     cl,n      ; cl ⇐ n, 0 ≤ n ≤ 255
        xor     ch,ch     ; cx = n szavasan
        xor     dx,dx     ; az eredmény ideiglenes helye
        JCXZ    kesz      ; ugrás a kesz címkére,
                          ; ha CX (=n) = 0
        xor     bx,bx     ; bx ⇐ 0,
                          ; bx-et használjuk indexezéshez
```



```
ism:  mov     al,a[bx]  ; al ⇐ a[0], később a[1], ...
      imul    b[bx]     ; ax ⇐ a[0]*b[0], a[1]*b[1], ...
      add     dx,ax     ; dx ⇐ részösszeg
      inc     bx        ; bx ⇐ bx+1, az index növelése

; B      ciklus vége
      dec     cx        ; cx ⇐ cx-1, (vissza)számlálás
      JCXZ    kesz      ; ugrás a kész címkére, ha cx=0
      jmp     ism       ; ugrás az ism címkére
kesz: mov     ax,dx     ; a skalár szorzat értéke ax-ben
```

```
; C         eredmények kiírása
      call      hexa      ; az eredmény kiírása
                          ; hexadecimálisan
      mov       si,offset kvse ; kocsi vissza soremelés
      call      kiiro     ; kiírása
      ret                 ; vissza az Op. rendszerhez
skalar  endp              ; a skalár eljárás vége
; D
```



```
hexa  proc                ; ax kiírása hexadecimálisan
      xchg      ah,al     ; ah és al felcserélése
      call      hexa_b    ; al (az eredeti ah) kiírása
      xchg      ah,al     ; ah és al visszacserélése
      call      hexa_b    ; al kiírása
      ret                 ; visszatérés
hexa  endp                ; a hexa eljárás vége
; ---------------------------------------------------------
hexa_b    proc            ; al kiírása hexadecimálisan
      push      cx        ; mentés a verembe
      mov       cl,4      ; 4 bit-es rotálás előkészítése
      ROR       al,CL     ; az első jegy az alsó 4 biten
      call      h_jegy    ; az első jegy kiírása
      ROR       al,CL     ; a második jegy az alsó 4 biten
      call      h_jegy    ; a második jegy kiírása
      pop       cx        ; visszamentés a veremből
      ret                 ; visszatérés
hexa_b    endp            ; a hexa_b eljárás vége
```



```
h_jegy    proc            ; hexadecimális jegy kiírása
      push      ax        ; mentés a verembe
      AND       al,0FH    ; a felső 4 bit 0 lesz,
                          ; a többi változatlan
      add       al,'0'    ; + 0 kódja
      cmp       al,'9'    ; ≤ 9 ?
      JLE       h_jegy1   ; ugrás h_jegy1 -hez, ha igen
      add       al,'A'-'0'-0AH    ; A-F hexadecimális jegyek
                          ; kialakítása
h_jegy1:  mov ah,14       ; BIOS szolgáltatás előkészítése
      int       10H       ; BIOS hívás: karakter kiírás
      pop       ax        ; visszamentés a veremből
      ret                 ; visszatérés
h_jegy    endp            ; a hexa_b eljárás vége
```



```
kiiro proc                ; szöveg kiírás (DS:SI)-től
      push      ax
      cld
ki1:  lodsb               ; al⇐a következő karakter
      cmp       al, 0     ; al =? 0
      je        ki2       ; ugrás a ki2 címkéhez, ha al=0
      mov       ah,14     ; BIOS rutin paraméterezése
      int       10H       ; az AL-ben lévő karaktert
                          ; kiírja a képernyőre
      jmp       ki1       ; a kiírás folytatása
ki2:  pop       ax
      ret                 ; visszatérés a hívó programhoz
kiiro endp                ; a kiíró eljárás vége
; ---------------------------------------------------------
code  ends                ; a code szegmens vége
```



```
data      segment   para public 'data'
n         db        3
a         db        1, 2, 3
b         db        3, 2, 1
kvse      db        13, 10, 0   ; kocsi vissza, soremelés
data      ends                  ; a data szegmens vége
; =========================================
stack     segment   para stack 'stack'
          dw   100 dup (?)      ; 100 word legyen a verem
stack     ends                  ; a stack szegmens vége
; =========================================
      end   skalar              ; modul vége,
                                ; a program kezdő címe: skalar
```


Vezérlés átadó utasítások

Eljárásokkal kapcsolatos utasítások

Eljárás hívás:

CALL op ; eljárás hívás

- közeli: *push* IP, IP ⇐ op,

- távoli: *push* CS, *push* IP, (CS:IP) ⇐ op.

Visszatérés az eljárásból:

RET ; visszatérés a hívó programhoz (RETurn)

- közeli: *pop* IP,

- távoli: *pop* IP, *pop* CS.

RET op ; . . . , SP ⇐ SP+op

; op csak közvetlen adat lehet!

Feltétlen vezérlés átadás (ugrás)

```
JMP     op   ; ha op közeli:    IP ⇐ op,
             ; ha távoli:   (CS:IP) ⇐ op.
```


Feltételes ugrások, aritmetikai csoport

Előjeles	Reláció	Előjel nélküli
JZ ≡ JE	=	JZ ≡ JE
JNZ ≡ JNE	≠	JNZ ≡ JNE
JG ≡ JNLE	>	JA ≡ JNBE
JGE ≡ JNL	≥	JAE ≡ JNB ≡ JNC
JL ≡ JNGE	<	JB ≡ JNAE ≡ JC
JLE ≡ JNG	≤	JBE ≡ JNA

A feltételek: Zero, **E**qual, **N**o (**N**ot),
Greater, **L**ess, **A**bove, **B**elow, **C**arry

Feltételes ugrások, logikai csoport

a flag igaz (1)	flag	a flag hamis (0)
JZ ≡ JE	Zero	JNZ ≡ JNE
JC	Carry	JNC
JS	Sign	JNS
JO	Overflow	JNO
JP ≡ JPE	Parity	JNP ≡ JPO
JCXZ	CX = 0	

A feltételek: Zero, **E**qual, **N**o (**N**ot), **C**arry, **S**ign,
Overflow, **P**arity **E**ven, **P**arity **O**dd.

Minden feltételes vezérlés átadás **IP** relatív címzéssel (**SHORT**) valósul meg!

Pl.:

```
        JZ   MESSZE     ; Hibás, ha
                        ; MESSZE messze van
```

Megoldás:

```
        JNZ     IDE     ; Negált feltételű ugrás
        JMP     MESSZE
IDE:    . . .
```


Ciklus szervező utasítások

IP relatív címzéssel (**SHORT**) valósulnak meg.

```
LOOP    ipr  ; CX ⇐ CX – 1, ugrás ipr -re,
             ; ha CX ≠ 0
LOOPZ   ipr  ; CX ⇐ CX – 1, ugrás ipr -re,
             ; ha (CX ≠ 0 és Z=1)
LOOPE   ipr  ; ugyanaz mint LOOPZ
LOOPNZ  ipr  ; CX ⇐ CX – 1, ugrás ipr -re,
             ; ha (CX ≠ 0 és Z=0)
LOOPNE  ipr  ; ugyanaz mint LOOPNZ
```



```
; B = A n-dik hatványa,
; A és n előjel nélküli byte, B word
; Feltétel: A n-1 –dik hatványa elfér AL –ben.
        mov     cl, n     ; a ciklus előkészítése
        xor     ch, ch
        mov     al, 1     ; lehetne:    mov ax, 1
        xor     ah, ah    ; akkor ez nem kell
        JCXZ    kesz      ; ha n=0, akkor 0-szor
                          ; fut a ciklus mag
c_mag:  mul     A         ; ciklus mag
        LOOP    c_mag     ; ismétlés, ha kell
kesz:   mov     B, ax
```

Egyszerűsítési lehetőség a skalár szorzatot kiszámító programban:

```
; B
      dec       cx        ; cx ⇐ cx-1, (vissza)számlálás
      jcxz      kesz      ; ugrás a kész címkére, ha cx=0
      jmp       ism       ; ugrás az ism címkére
kesz: mov       ax,dx     ; a skalár szorzat értéke ax-ben
```

helyett:

```
; B
      LOOP      ism       ; ugrás az ism címkére,
                          ; ha kell ismételni
kesz: mov       ax,dx     ; a skalár szorzat értéke ax-ben
```


Annak érdekében, hogy a skalárszorzatot kiszámító program ne rontson el regisztereket, kívánatos ezek mentése:

```
; A
      PUSH      BX  ; mentés
      PUSH      CX
      PUSH      DX
```

és visszamentése:

```
      POP       DX  ; visszamentés
      POP       CX
      POP       BX
; C
```


A paraméterek szabványos helyen történő átadása

```
; Két vektor skalár szorzata. 2. változat
...
; A          skalár szorzat számítása
; ELJÁRÁS HÍVÁS A PARAMÉTEREK
; SZABVÁNYOS HELYEN TÖRTÉNŐ ÁTADÁSÁVAL
      CALL      SKAL          ; ELJÁRÁS HÍVÁS
                              ; eredmény az AX regiszterben
; C          eredmények kiírása
      call      hexa          ; az eredmény kiírása
      mov       si,offset kvse ; kocsi vissza, soremelés
      call      kiiro         ; kiírása
...
      ret                     ; vissza az Op. rendszerhez
skalar  endp                  ; a skalár eljárás vége
; D
```



```
SKAL      PROC          ; KÖZELI (NEAR) ELJÁRÁS
                        ; KEZDETE
; Az A-tól C-ig tartó program rész:
      PUSH      BX      ; MENTÉSEK
      PUSH      CX
      PUSH      DX
      mov       cl,n    ; cl ⇐ n, 0 ≤ n ≤ 255
      xor       ch,ch   ; cx = n szavasan
      xor       dx,dx   ; az eredmény ideiglenes helye
      jcxz      kesz    ; ugrás a kesz címkére, ha n=0
      xor       bx,bx   ; bx ⇐ 0,
                        ; bx-et használjuk indexezéshez
```



```
ism:  mov       al,a[bx]  ; al ⇐ a[0], később a[1], ...
      imul      b[bx]     ; ax ⇐ a[0]*b[0], a[1]*b[1],...
      add       dx,ax     ; dx ⇐ részösszeg
      inc       bx        ; bx ⇐ bx+1, az index növelése
; B          ciklus vége
      LOOP      ism       ; ugrás az ism címkére,
                          ; ha kell ismételni
kesz: mov       ax,dx     ; a skalár szorzat értéke ax-ben
      POP       DX        ; VISSZAMENTÉSEK
      POP       CX
      POP       BX
      RET                 ; VISSZATÉRÉS A HÍVÓ
                          ; PROGRAMHOZ
;          visszatérés a C ponthoz
SKAL      ENDP            ; A SKAL ELJÁRÁS VÉGE
; D
…
```

Csak az **a** és **b** vektor skalár szorzatát tudja kiszámolni!