

Aritmetikai utasítások

ADD op1, op2 ; $op1 \leftarrow op1 + op2$ (ADD)

Pl.: előjeles/előjel nélküli számok összeadása

MOV AX, -1 ; $AX = -1$ (=0FFFFH)

ADD AX, 2 ; $AX = 1$, C=1, O=0

ADC op1, op2 ; $op1 \leftarrow op1 + op2 + C$

; (ADD with Carry)

Pl.: két szavas összeadás (előjeles/előjel nélküli)

ADD AX, BX

ADC DX, CX ; $(DX:AX) = (DX:AX) + (CX:BX)$

INC op ; $op \leftarrow op + 1$, C változatlan! (INCRement)

Máté: Assembly programozás

3. előadás

1

SUB op1, op2 ; $op1 \leftarrow op1 - op2$ (SUBtraction)

CMP op1, op2 ; flag-ek op1 - op2 szerint (CoMPare)

**SBB op1, op2 ; $op1 \leftarrow op1 - op2 - C$;
; a több szavas kivonást segíti.**

**DEC op ; $op \leftarrow op - 1$, C változatlan
; (DECrement)**

NEG op ; $op \leftarrow -op$ (NEGate)

Máté: Assembly programozás

3. előadás

2

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 $AX \leftarrow AL * op$.

Ha op 16 bites $(DX:AX) \leftarrow AX * op$.

Máté: Assembly programozás

3. előadás

3

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: $AL \leftarrow AX/op$ hányadosa,
 $AH \leftarrow AX/op$ maradéka.**

**Ha op 16 bites: $AX \leftarrow (DX:AX)/op$ hányadosa,
 $DX \leftarrow (DX:AX)/op$ maradéka.**

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

Máté: Assembly programozás

3. előadás

4

Ha bájtot bájtval 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)

; $AX \leftarrow AL$ előjel helyesen

CWD ; (Convert Word to Double word)

; $(DX:AX) \leftarrow AX$ 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.

Máté: Assembly programozás

3. előadás

5

; 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 ; $ax \leftarrow 0$

push ax ; visszatérés offset címe

mov ax, data ; ds a data szegmensre mutasson

mov ds, ax ; sajnos „mov ds, data”

; nem megengedett

Máté: Assembly programozás

3. előadás

6

; 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 kez címkére,
		; ha CX (=n) = 0
xor	bx,bx	; bx ← 0,
		; bx-et használjuk indexezéshez
Máté: Assembly programozás 3. előadás 7		

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
Máté: Assembly programozás 3. előadás 8		

; C eredmények kiírása		
call	hexa	; az eredmény kiírása
		; hexadecimálisan
mov	si,offset kvse	; koci vissza soremelés
call	kiiro	; kiírása
ret		; vissza az Op. rendszerhez
skalar endp		; a skalár eljárás vége
; D		
Máté: Assembly programozás 3. előadás 9		

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
Máté: Assembly programozás 3. előadás 10		

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
Máté: Assembly programozás 3. előadás 11		

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 címkehez, 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
Máté: Assembly programozás 3. előadás 12		

```

data      segment para public 'data'
n         db      3
a         db      1, 2, 3
b         db      3, 2, 1
kvse      db      13, 10, 0 ; kocszi 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

```

Máté: Assembly programozás 3. előadás 13

Vezérlés átadó utasítások

Eljárásokkal kapcsolatos utasítások

Eljárás hívás:

CALL op ; eljárás hívás

- közeli: **push IP, $IP \leftarrow op$,**
- távoli: **push CS, push IP, $(CS:IP) \leftarrow op$.**

Visszatérés az eljárásból:

RET ; visszatérés a hívó programhoz (RETurn)

- közeli: **pop IP,**
- távoli: **pop IP, pop CS.**

RET op ; ... , $SP \leftarrow SP+op$
; op csak közvetlen adat lehet!

Máté: Assembly programozás 3. előadás 14

Feltétlen vezérlés átadás (ugrás)

JMP op ; ha op közeli: $IP \leftarrow op$,
; ha távoli: $(CS:IP) \leftarrow op$.

Máté: Assembly programozás 3. előadás 15

Feltételes ugrások, aritmetikai csoport

Előjeles	Reláció	Előjel nélküli
JZ $\equiv$ JE	=	JZ $\equiv$ JE
JNZ $\equiv$ JNE	$\neq$	JNZ $\equiv$ JNE
JG $\equiv$ JNLE	>	JA $\equiv$ JNBE
JGE $\equiv$ JNL	$\geq$	JAe $\equiv$ JNB $\equiv$ JNC
JL $\equiv$ JNGE	<	JB $\equiv$ JNAE $\equiv$ JC
JLE $\equiv$ JNG	$\leq$	JBE $\equiv$ JNA

A feltételek: Zero, Equal, No (Not),
Greater, Less, Above, Below, Carry

Máté: Assembly programozás 3. előadás 16

Feltételes ugrások, logikai csoport

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

A feltételek: Zero, Equal, No (Not), Carry, Sign,
Overflow, Parity Even, Parity Odd.

Máté: Assembly programozás 3. előadás 17

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: ...

Máté: Assembly programozás 3. előadás 18

Ciklus szervező utasítások

IP relatív címzéssel (SHORT) valósulnak meg.

LOOP ipr ; $CX \leftarrow CX - 1$, ugrás ipr -re,
; ha $CX \neq 0$

LOOPZ ipr ; $CX \leftarrow CX - 1$, ugrás ipr -re,
; ha ($CX \neq 0$ és $Z=1$)

LOOPE ipr ; ugyanaz mint LOOPZ

LOOPNZ ipr ; $CX \leftarrow CX - 1$, ugrás ipr -re,
; ha ($CX \neq 0$ és $Z=0$)

LOOPNE ipr ; ugyanaz mint LOOPNZ

Máté: Assembly programozás

3. előadás

19

; $B = A$ 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 kész ; ha n=0, akkor 0-szor

; fut a ciklus mag

c_mag: mul A ; ciklus mag

LOOP c_mag ; ismétlés, ha kell

kész: mov B, ax

Máté: Assembly programozás

3. előadás

20