

```

A paraméterek regiszterekben történő átadása
; Két vektor skalár szorzata. 3. változat
...
; A
; ELJÁRÁS HÍVÁS A PARAMÉTEREK
; REGISZTEREKBE TÖRTÉNŐ ÁTADÁSÁVAL
MOV     CL, n      ; PARAMÉTER BEÁLLÍTÁSOK
XOR     CH, CH     ; CX = n, ÉRTÉK
MOV     SI, OFFSET a ; SI ← a OFFSET CÍME
MOV     DI, OFFSET b ; DI ← b OFFSET CÍME
call    skal       ; eljárás hívás
                     ; eredmény az ax regiszterben
call    hexa       ; az eredmény kiírása
mov     si, offset kvse ; kocsí vissza, soremelés
call    kiíro      ; kiírása
...
ret     ; visszatérés az Op. rendszerhez
skal    endp       ; a skalár eljárás vége

```

Máté: Assembly programozás 4. előadás 1

```

skal    proc       ; Közeli (NEAR) eljárás kezdete
push    bx         ; mentések
push    cx
push    dx
xor     dx, dx     ; az eredmény ideiglenes helye
jcxz    kész       ; ugrás a kész címkeére, ha n=0
xor     bx, bx     ; bx ← 0,
                     ; bx-et használjuk indexezéshez

```

Máté: Assembly programozás 4. előadás 2

```

ism:    mov     al, [SI+BX] ; FÜGGETLEN a-tól
        imul    BYTE PTR [DI+BX]; FÜGGETLEN b-től
; csak „BYTE PTR”-ből derül ki, hogy 8 bites a szorzás
        add     dx, ax     ; dx ← részösszeg
        inc     bx        ; bx ← bx+1, az index növelése
        loop    ism       ; ugrás az ism címkeére,
                     ; ha kell ismételni
kész:   mov     ax, dx     ; a skalár szorzat értéke ax-ben
        pop     dx        ; visszamentések
        pop     cx
        pop     bx
        ret     ; visszatérés a hívó programhoz
skal    endp       ; a skalár eljárás vége
; D
...

```

Így csak kevés paraméter adható át!

Máté: Assembly programozás 4. előadás 3

```

; Két vektor skalár szorzata. 4. változat
...
; A
; ELJÁRÁS HÍVÁS A PARAMÉTEREK
; VEREMBEN TÖRTÉNŐ ÁTADÁSÁVAL
MOV     AL, n      ; AL-t nem kell menteni, mert
XOR     AH, AH     ; AX-ben kapjuk az eredményt
PUSH    AX         ; AX=n a verembe
MOV     AX, OFFSET a ; AX ← a OFFSET címe
PUSH    AX         ; a verembe
MOV     AX, OFFSET b ; AX ← b OFFSET címe
PUSH    AX         ; a verembe

```

A verembe került:
n értéke, a címe, b címe *paraméterek*

Máté: Assembly programozás 4. előadás 4

```

call    skal       ; eljárás hívás
                     ; eredmény az ax regiszterben
ADD     SP, 6      ; paraméterek ürítése a veremből
...
ret     ; visszatérés az Op. rendszerhez
skal    endp       ; a skalár eljárás vége

```

call skal

Hatására a verembe került a visszatérési cím

Máté: Assembly programozás 4. előadás 5

```

skal    proc       ; Közeli (near) eljárás kezdete
PUSH    BP         ; BP értékét mentenünk kell
MOV     BP, SP     ; BP ← SP,
                     ; a stack relatív címzéshez
PUSH    SI         ; mentések
PUSH    DI
push    bx
push    cx
push    dx

```

A verem tartalma:
n értéke, a címe, b címe *paraméterek*
visszatérési cím,
bp, si, di, bx, cx, dx *mentett regiszterek*

Máté: Assembly programozás 4. előadás 6

A verem tartalma:			
n értéke, a címe, b címe		paraméterek	
visszatérési cím,			
bp, si, di, bx, cx, dx		mentett regiszterek	
(SS:SP)	dx	PUSH BP ; BP értékét mentenünk kell	- 10
+ 2	cx	MOV BP,SP ; BP \Leftarrow SP,	- 8
+ 4	bx		- 6
+ 6	di		- 4
+ 8	si		- 2
+10	bp	----- (SS:BP)	
+12		visszatérési cím	+ 2
+14		b címe	+ 4
+16		a címe	+ 6
+18		n értéke	+ 8
...		korábbi mentések	...

Máté: Assembly programozás

4. előadás

7

+10	bp	----- (SS:BP)	
+12		visszatérési cím	+ 2
+14		b címe	+ 4
+16		a címe	+ 6
+18		n értéke	+ 8
...		korábbi mentések	...
MOV	SI,6[BP]	; SI \Leftarrow az egyik vektor címe	
MOV	DI,4[BP]	; DI \Leftarrow a másik vektor címe	
MOV	CX,8[BP]	; CX \Leftarrow a dimenzió értéke	
xor	dx,dx	; az eredmény ideiglenes helye	
jcxz	kesz	; ugrás a kesz címkére, ha n=0	
xor	bx,bx	; bx \Leftarrow 0, indexezéshez	

Máté: Assembly programozás

4. előadás

8

ism: mov	al,[si+bx]	; független a-tól
imul	byte ptr [di+bx]	; független b-től
; csak „byte ptr”-ből derül ki, hogy 8 bites a szorzás		
add	dx,ax	; dx \Leftarrow részösszeg
inc	bx	; bx \Leftarrow bx+1, az index növelése
loop	ism	; ugrás az ism címkére,
		; ha kell ismételni
kesz: mov	ax,dx	; a skalár szorzat értéke ax-ben

Máté: Assembly programozás

4. előadás

9

pop	dx	; visszamentések
pop	cx	
pop	bx	
POP	DI	
POP	SI	
POP	BP	
ret		; visszatérés a hívó programhoz
skal	endp	; a skal eljárás vége
; D		
...		
ADD	SP,6	; paraméterek ürítése a veremből

helyett más megoldás:

RET	6	; visszatérés a hívó programhoz
		; verem ürítéssel: ... SP = SP + 6

Máté: Assembly programozás

4. előadás

10

C konvenció	
<p>Hogy egy eljárás különböző számú paraméterrel legyen hívható, azt úgy lehet elérni, hogy a paramétereket fordított sorrendben tesszük a verembe, mert ilyenkor a visszatérési cím alatt lesz az első, alatta a második, stb. paraméter, és általában a korábbi paraméterek döntik el, hogy hogyan folytatódik a paramétersor.</p>	
f(x,y);	push y
	push x
	call f

Máté: Assembly programozás

4. előadás

11

Lokális adat terület, rekurzív és re-entrant eljárások

Ha egy eljárás működéséhez lokális adat területre, munkaterületre van szükség, és a működés befejeztével a munkaterület tartalma fölösleges, akkor a munkaterületet célszerűen a veremben alakíthatjuk ki. A munkaterület lefoglalásának ajánlott módja:

...	proc	...	
PUSH		BP	; BP értékének mentése
MOV		BP,SP	; BP \Leftarrow SP,
			; a stack relatív címzéshez
SUB		SP,n	; n byte-os munkaterület lefoglalása
...			; további regiszter mentések

Máté: Assembly programozás

4. előadás

12

Máté: Assembly programozás

4. előadás

12

Lokális adat terület (NEAR eljárás esetén)

```

(SS:SP)  lokális adat terület      ...
+ 2      ...
...      ...
...      - 2
bp ----- (SS:BP)
visszatérési cím      + 2
paraméterek      ...
korábbi mentések      ...

```

A munkaterület negatív displacement érték mellett stack relatív címzéssel érhető el. (A veremben átadott paraméterek ugyancsak stack relatív címzéssel, de pozitív displacement érték mellett érhetők el.)

Máté: Assembly programozás

4. előadás

13

A munkaterület felszabadítása visszatéréskor a

```

...      ; visszamentések
MOV      SP,BP      ; a munkaterület felszabadítása
POP      BP      ; BP értékének visszamentése
ret      ...      ; visszatérés

```

utasításokkal történhet.

Máté: Assembly programozás

4. előadás

14

Rekurzív és re-entrant eljárások

Egy eljárás **rekurzív**, ha önmagát hívja közvetlenül, vagy más eljárásokon keresztül.

Egy eljárás **re-entrant**, ha többszöri belépést tesz lehetővé, ami azt jelenti, hogy az eljárás még nem fejeződött be, amikor újra felhívható. A rekurzív eljárással szemben a különbség az, hogy a rekurzív eljárásban „programozott”, hogy mikor történik az eljárás újra hívása, re-entrant eljárás esetén az esetleges újra hívás ideje a véletlentől függ. Ez utóbbi esetben azt, hogy a munkaterületek ne keveredjenek össze, az biztosítja, hogy újabb belépés csak másik processzusból képzelhető el, és minden processzus saját vermet használ.

Máté: Assembly programozás

4. előadás

15

Rekurzív és re-entrant eljárások

Ha egy eljárásunk készítésekor betartjuk, hogy az eljárás a paramétereit a vermen keresztül kapja, kilépéskor visszaállítja a belépéskori regiszter tartalmakat – az esetleg eredményt tartalmazó regiszterek kivételével –, továbbá a fenti módon kialakított munkaterületet használ, akkor az eljárásunk rekurzív is lehet, és a többszöri belépést is lehetővé teszi (re-entrant).

Máté: Assembly programozás

4. előadás

16

String kezelő utasítások

Az **s** forrás területet (DS:SI),
a **d** cél területet pedig (ES:DI) címzi.

A mnemonik végződése (**B** / **W**) vagy az operandus jelzi, hogy bájtos vagy szavas a művelet.

A címzésben résztvevő indexregiszterek értéke

- 1-gyel módosul bájtos,
- 2-vel szavas művelet esetén.

Ha a **D (Direction)** flag értéke

- 0, akkor az indexregiszterek értéke növekszik,
- 1, akkor csökken.

CLD ; D ← 0

STD ; D ← 1

Máté: Assembly programozás

4. előadás

17

Az alábbi utasítások

– mint általában az adat mozgó utasítások –
érintetlenül hagyják a flag-eket

Átvitel az (ES:DI) által mutatott címre
a (DS:SI) által mutatott címről:

MOVSB ; MOVE String Byte

MOVSW ; MOVE String Word

MOVS d,s ; MOVE String (byte vagy word)

d és s csak azt mondja meg,
hogy bájtos vagy szavas az átvitel!

Máté: Assembly programozás

4. előadás

18

Betöltés **AL**-be illetve **AX**-be a **(DS:SI)** által mutatott címről (csak **SI** módosul):

LODSB ; LOaD String Byte
LODSW ; LOaD String Word
LODS s ; LOaD String (byte vagy word)

Tárolás az **(ES:DI)** által mutatott címre **AL**-ből illetve **AX**-ből (csak **DI** módosul):

STOSB ; STOrE String Byte
STOSW ; STOrE String Word
STOS d ; STOrE String (byte vagy word)

Máté: Assembly programozás

4. előadás

19

Az alábbi utasítások beállítják a flag-eket

Az **(ES:DI)** és a **(DS:SI)** által mutatott címen lévő byte illetve szó összehasonlítása, a flag-ek **s – d** (!!!) értékének megfelelően állnak be.

CMPSB ; CoMPare String Byte
CMPSW ; CoMPare String Word
CMPS d,s ; CoMPare String (byte vagy word)

Az **(ES:DI)** által mutatott címen lévő byte (word) összehasonlítása **AL**-l (AX-szel), a flag-ek **AL – d** illetve **AX – d** (!!!) értékének megfelelően állnak be.

SCASB ; SCAn String Byte
SCASW ; SCAn String Word
SCAS d ; SCAn String (byte vagy word)

Máté: Assembly programozás

4. előadás

20

Ismétlő prefixumok

REP \equiv **REPZ** \equiv **REPE** és **REPZ** \equiv **REPNE**

A **Z**, **E**, **NZ** és **NE** végződésnek hasonló szerepe van, mint a **LOOP** utasítás esetén.

Ismétlő prefixum használata esetén a string kezelő utasítás **CX**-szer kerül(het) végrehajtásra:

- ha **CX = 0**, akkor egyszer sem (!!!),
- különben minden végrehajtást követően 1-gyel csökken a **CX** regiszter tartalma. Amennyiben **CX** csökkentett értéke **0**, akkor nem történik további ismétlés.

A flag beállító string kezelő utasítás ismétlésének további feltétele, hogy a flag állapota megegyezzen a prefixum végződésében előírttal.

Máté: Assembly programozás

4. előadás

21

Ismétlő prefixumok

REP \equiv **REPZ** \equiv **REPE** és **REPZ** \equiv **REPNE**

A program jobb olvashatósága érdekében

- flag-et nem állító utasítások előtt mindig **REP**-et használjunk,
- flag-et beállító utasítás előtt pedig sohase **REP**-et, hanem helyette a vele egyenértékű **REPE**-t vagy **REPZ**-t!

Máté: Assembly programozás

4. előadás

22

```
; 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
        ...
        ...
        ; array ≡ 3
NEM3:   ...
        ...
        ; di az első 3-tól különböző
        ; elem indexe
```

Máté: Assembly programozás

5. előadás

23

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
...
; array ≡ 3,
...
NEM3:   DEC     DI          ; DI az első ≠ 3 elemre mutat
...
```

Máté: Assembly programozás

5. előadás

24

A 100 elemű array nevű tömbnek van-e 3-tól különböző eleme?			
mov	cx, 100	mov	cx, 100
mov	di, -1	mov	di, offset array
mov	al, 3	mov	AL, 3
NEXT: inc	di	REPE	SCAS array
cmp	array[di], al		
LOOPE	NEXT		
		JNE	NEM3
		...	
		...	
NEM3: ...		NEM3: DEC	DI
		...	
			Használja ES-t. Sokkal gyorsabb volt.
Nem minden eltérés lényeges!			
Máté: Assembly programozás		5. előadás	
		25	