Assembly programozás
Bevezetés

Varga László

Képfeldolgozás és Számítógépes Grafika Tanszék
Szegedi Tudományegyetem

(Tanács Attila nappali előadás fóliái alapján)

Elérhetőségek, anyagok

- **Személyesen**
  - Előadás időpontjában
  - Fogadóórán
    - Szerda 14-15 (Árpád tér, tetőtéri 218.)
  - Előre (e-mailben) egyeztetett időpontban
- **Elektronikus formában**
  - `vargalg@inf.u-szeged.hu`
- **Kurzus anyagok**
  - `http://www.inf.u-szeged.hu/~vargalg`

Követelményrendszer

- **Gyakorlat**
  - „Táblás” gyakorlatok + számítógép használat
    - Füzet, toll szükséges!
  - 1 alkalommal „nagy ZH”, 40 pont
    - Utolsó gyakorlaton
  - A ZH 1 alkalommal javítható/pótolható

Követelményrendszer

- **Gyakorlat, konzultáció**
  - Beadható feladat (otthoni munka)
  - Kötelező a konzultációs kurzust felvetteknek
    - 2008 előtt felvettek
  - Másoknak opcionális
  - A választható feladatok várhatóan 2013. március 18-ig kerülnek fel a CooSpace-re.
  - 1 feladatra csoportonként maximum 2 hallgató jelentkezhet, a jelentkezést a CooSpace kezeli
  - Feladatra jelentkezni 2013. április 28-ig lehet
  - Megoldások beküldhetők 2013. május 19-ig

Követelményrendszer

- **Gyakorlat, konzultáció**
  - Tetszőleges segédeszköz használható, de érteni kell a program működésének minden részét!
  - A megoldásokat meg kell védeni az első vizsga alkalmával, vagy a gyakorlatvezetővel egyeztetett időpontban
  - **0-10 pont** szerezhető így
  - A konzultáció teljesítése akkor sikeres, ha legalább 5 pontos a beadott feladat védése

Követelményrendszer

- **Gyakorlat (folytatás)**
  - A vizsgára bocsáthatósághoz **20 pont** (40%) szükséges!
  - Kerekítés nincs!

Követelményrendszer

- **Kollokviumi vizsga**
  - 10 kérdésből álló teszt
    - 2x10 = 20 pont
    - 20 perc munkaidő
    - Rövid, tömör, lényegretörő válaszok
  - Kiadott tételjegyzékből 2 kidolgozása
    - 2x15 = 30 pont
    - Esszé
  - Összesen 50 pont szerezhető
  - Semmilyen segédeszköz nem használható!

Követelményrendszer

- **Jegy megállapítása**
  - A gyakorlati és a kollokviumi összpontszám alapján

Összpontszám	Érdemjegy
Ha a vizsgán elért pontszám < 20	elégtelen (1)
Ha az elért pontszámok összege < 50	elégtelen (1)
Ha az elért pontszámok összege < 60, de >=50	elégséges (2)
Ha az elért pontszámok összege < 75, de >=60	közepes (3)
Ha az elért pontszámok összege < 90, de >=75	jó (4)
Ha az elért pontszámok összege >= 90	jeles (5)

Ajánlott irodalom

- Diós Gábor, Rodek Lajos féle **Assembly programozás** jegyzet.
- Pethő Ádám: **IBM PC/XT felhasználóknak és programozóknak 1. Assembly programozás** (SZÁMALK, 1992)
- Máté Eörs: **Assembly programozás** (NOVADAT, 1999, 2000)
- S. Tanenbaum: **Számítógép-architektúrák**, 2. átdolgozott, bővített kiadás (Panem 2006).
  - Csak a könyv 5. és 7. fejezete
- B. B. Brey: **Programming the 80286, 80386, 80468, and Pentium-baseb Personal Computer** (Prentice Hall, 1996)

Tematika

- Assembly alapfogalmak. Assembly nyelv előnyei, hátrányai, alkalmazási lehetőségei.
- A 80x86 memória modellje.
- A 80x86 regiszterkészlete.
- Adat- és kódterület címzése.
- Szegmens regiszterek, használatukra vonatkozó szabályok.
- Aritmetikai, adatmozgató, logikai utasítások.
- Vezérlésátadás, eljáráshívás, ciklusszervezés.
- Sztring műveletek, REP prefixumok.
- Paraméterátadási lehetőségek eljáráshíváskor: regiszterekben, vermen keresztül.
- Rekurzív és reentráns eljárások.

Eszközök

- **Gyakorlatokon**
  - Macro Assembler
  - Borland Turbo Assembler, Debugger

- **Otthon, 64-bites környezetben**
  - Dosbox: ingyenes, multiplatform DOS emuláció

Gépi, nyelvi szintek

5. Probléma orientált nyelv szintje
   fordítás (fordító program)
4. **Assembly nyelv szintje**
   fordítás (assembler)
3. Operációs rendszer szintje
   részben értelmezés (operációs rendszer)
2. **Gépi utasítás szintje**
   ha van mikroprogram, akkor értelmezés
   (Kompatibilitás!)
1. Mikroarhitektúra szintje
   hardver
0. Digitális logika szintje

Modern asztali számítógép

- **Neumann-elvű gép**
  - Központi feldolgozó egység (**CPU**)
  - **Operatív memória** az adatok és a programok futás közbeni tárolására
  - Egységek közötti **adatsín-rendszer** a kommunikációhoz
  - **Bementi/kimeneti rendszer** a felhasználóval való kapcsolattartáshoz
  - Működést biztosító járulékos egységek
    - Tápellátás, …

Operatív memória

- **Felépítése**
  - Alapja a **bit**: 0 vagy 1 érték
  - Bitek rendszerint csoportosítva kerülnek feldolgozásra
    - Cella: legkisebb címezhető egység
    - **8 bit = 1 bájt** (Oka: ASCII 7 bites karakterkód + 1 paritás)
    - Lehetnek más, pl. 4, 16 vagy más csoportosítások is!
  - A memória bájtok sorozata
  - Bájtok elérése a címükkel

Operatív memória

- **Bájtok értékeinek értelmezése**
  - Lehet **adat és program** is!
    - Adat: ASCII, UTF, BCD, kettes komplemens, lebegőpontos, …
    - Gépi kód: a számok utasításokat jelentenek
      - A CPU csak ezeket az utasításokat tudja végrehajtani!
      - Magasabb szintű programozási nyelvekről gépi kódra kell fordítani.
  - Akár **önmódosító program** is készíthető
    - Veszélyes!
    - Hibás működés, ha adatrészre kerül a vezérlés!
    - A mai modern operációs rendszerek védik a kódterületet
    - Ez elősegíti a virtuális memória hatékonyabb kezelését is

Memóriatérkép

- **Memória felosztása**
  - Nagy része szabadon használható terület
  - Bizonyos címterületek a hardverrel való kapcsolattartásra vannak fenntartva
    - Pl. kijelző, merevlemez, külső meghajtók
  - Bizonyos címek meghatározhatják az egyes címterületek tartalmát
    - Pl. RAM vagy ROM legyen ott elérhető
- **Mérete (PC-ken)**
  - Korábban: pár kilobájt, megabájt
    - Pl. A Commodore 64 gép 64 kilobájtot ért el, ez megfelel egy 256x256 méretű szürkeárnyalatos kép mátrixának!
  - Manapság: több gigabájt

Központi feldolgozó egység (CPU)

- **Feladata**
  - A programszámláló által mutatott memóriacímen lévő utasítást végrehajtja
  - A mutató továbblép a következő utasításra, vagy ugró utasítás esetén a megadott címre

- **Regiszterek**
  - **Nagyon gyors elérésű tárolóegységek**
  - Általában 1 gépi szó hosszúságúak (8, 16, 32 vagy 64-bit)
  - Áramköri vagy RAM megvalósítás
  - Általános vagy dedikált regiszter
    - Aritmetikai műveletre, memória címzésére
    - Címregiszter, állapotregiszterek, …
  - Minél több van, annál jobb
    - A memória elérése sokkal lassabb!

Számítógép (PC) működési vázlata

- Bekapcsolás
- Programszámláló a BIOS EPROM-ban található gépi kódú rendszerbetöltő programjának az elejére (rögzített cím)
- A bájt értékek által definiált utasítások végrehajtása
  - Rendszerteszt
  - Rendszerbetöltő megkeresi a rendszerindító egységet (pl. merevlemez) és annak betöltő programjára „ugrik” (rögzített helyen van)
  - Az betölti az operációs rendszert
  - Az operációs rendszer
    - Tartja a kapcsolatot felhasználóval,
    - Ütemezi a processzusokat,
    - Kezeli az erőforrásokat
    - …

Fogalmak

- **Gépi kód**
  - Numerikus gépi nyelv
  - Az utasítások és az operandusok számok
  - 1 utasítás 1 vagy több bájton kódolódik
  - Elemi műveletek végrehajtására
- **Assembly nyelv**
  - A numerikus gépi nyelv szimbolikus formája
  - **Mnemonik**
    - emlékeztető kód a numerikus utasítások helyett
  - Szimbolikus nevek és címek
  - Makrók
  - Feltételes fordítás
  - ...

```
                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A
0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA     RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG     SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON      GO TO START OF MONITOR
```

Cím | Gépi kód | Szimbólumok + mnemonikok | Megjegyzés

Kód forrása: *Thinking Soul*

Fogalmak

- **Assembler**
  - A fordító, amely assembly nyelvről gépi kódra fordít
- **Disassembler**
  - Gépi kódból mnemonik kód listázása
  - Vigyázni kell, hogy a listázás kezdőcíme valóban utasításhatáron legyen!
  - Ha szimbólumlista elérhető, akkor Assembly-szerű lista kapható

Assembly nyelv

- **Jellemzők**
  - Minden utasításnak egyetlen gépi utasítás felel meg
  - Architektúránként különböző Assembly nyelv!
    - Pl. Intel, UltraSPARC, RISC-alapú architektúrák
    - Nincs a magasszintű nyelveknél tapasztalható portabilitás!

Assembly

- **Hátrányok**
  - Nehézkes, időigényes
  - Sok hibalehetőség
  - Hosszadalmasabb hibakeresés, karbantartás

- **Előnyök**
  - Hatékonyság
  - A hardver teljes elérhetősége
    - Bizonyos regiszterek magasszintű nyelvekből nem használhatók
    - Hardver közvetlen elérése

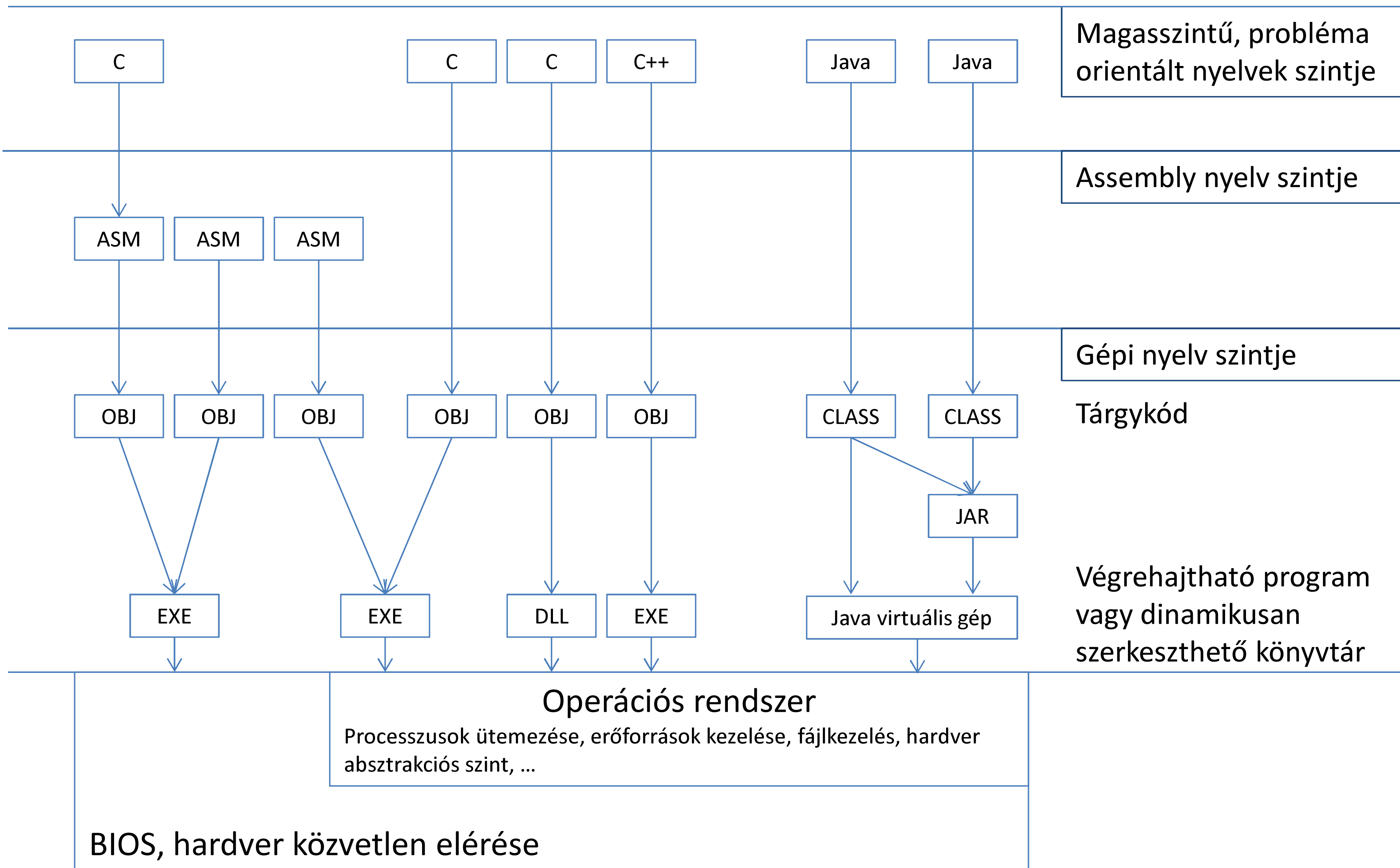
Fordítás, szerkesztés

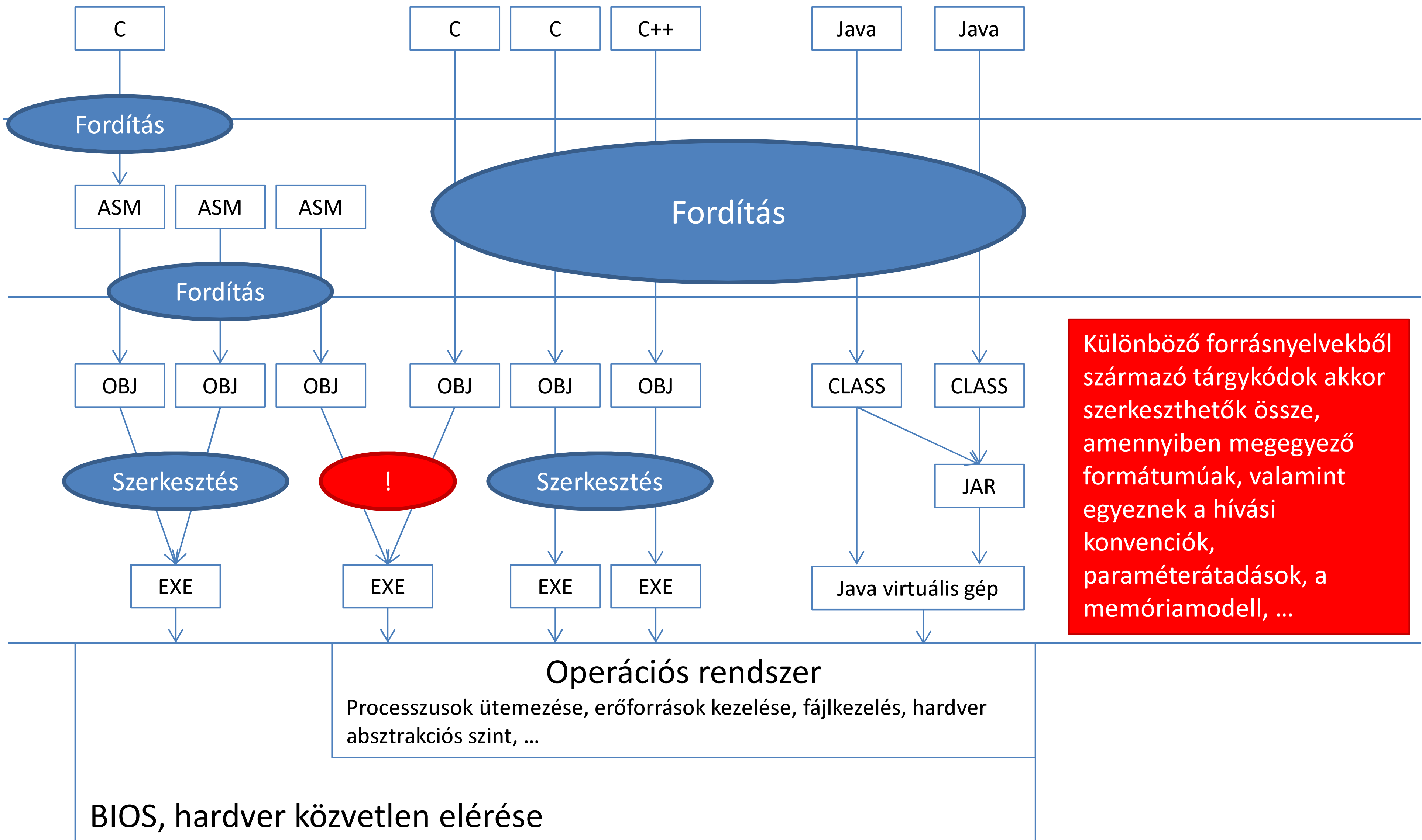
- **Fordító (Compiler)**
  - *Forrásnyelv*ből *célnyelv*re alakít, pl.:
    - C++, C -> tárgykód (.o/.obj)
    - Assembly -> tárgykód
    - Java -> class fájl
    - C -> Assembly (.asm)
    - FORTRAN -> C

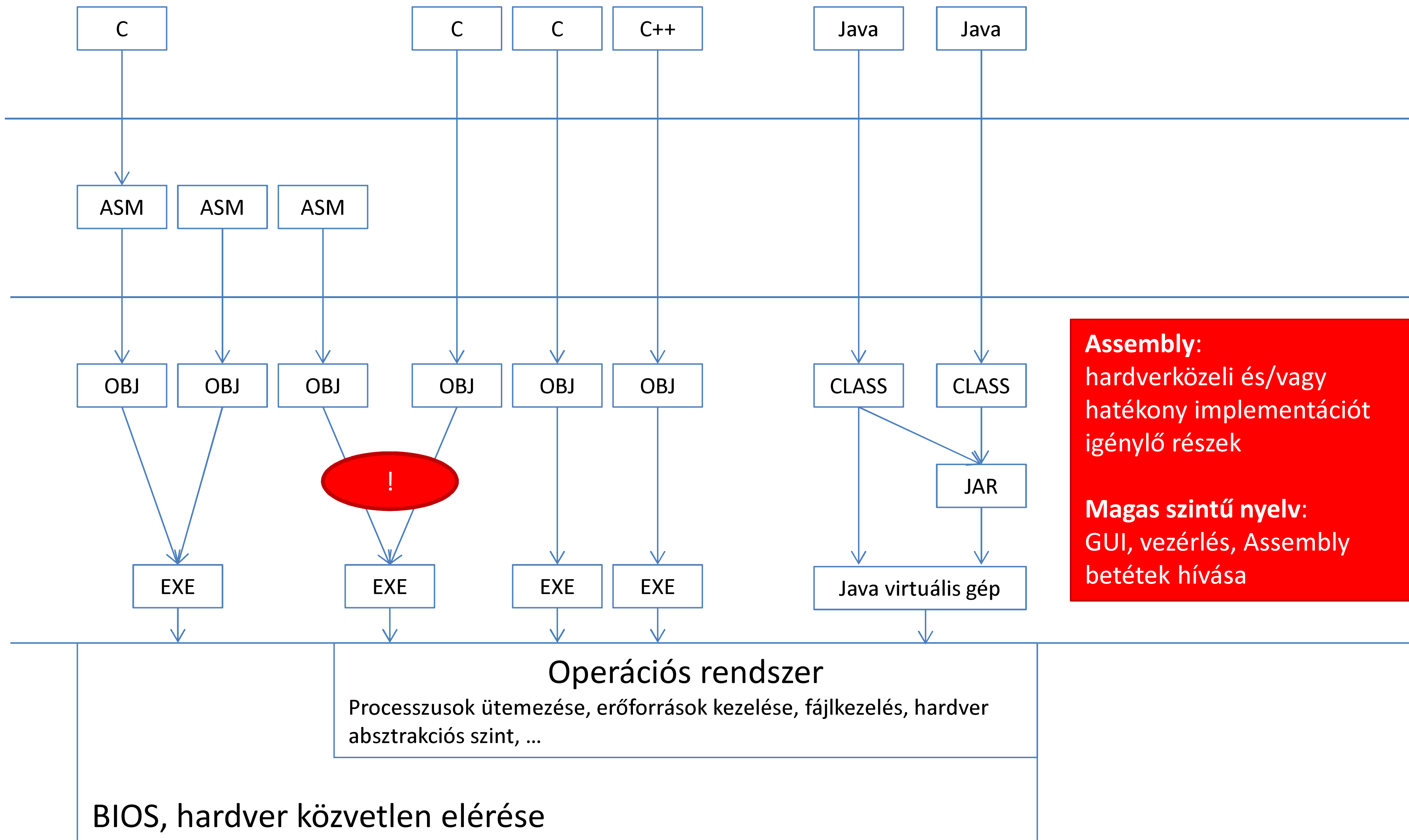
- **Szerkesztő (Linker)**
  - *Tárgykódok* összeszerkesztése *futtatható állománnyá*
    - Külső hivatkozások feloldása
    - Virtuális címek feloldása
    - Címterek összefésülése (relokáció)
    - …

Fordítás, szerkesztés

- **Fájlformátumok, kiterjesztések**
  - **Tárgykód**
    - .obj: DOS, Windows
    - .o: Unix, Linux, Cygwin, …
    - Fordító-specifikus formátum!
      - Azonos kiterjesztés még nem jelenti azt, hogy összeszerkeszthetők
  - **Végrehajtható (futtatható)**
    - .com: kis méretű DOS alkalmazások
    - .exe: DOS, Windows formátum (többféle szerkezet!)
    - .dll: dinamikusan szerkeszthető (Windows)
    - Unix-alapú rendszereknél nincs futtatható kiterjesztés, azt a fájl jogosultsága mondja meg
    - .so: dinamikusan szerkeszthető (Unix, Linux, …)
    - .a: statikusan szerkeszthető tárgykód gyűjtemény (Unix, Linux…)


Magasszintű, probléma orientált nyelvek szintje
C
C
C
C++
Java
Java
Assembly nyelv szintje
ASM
ASM
ASM
Gépi nyelv szintje
OBJ
OBJ
OBJ
OBJ
OBJ
OBJ
CLASS
CLASS
Tárgykód
JAR
EXE
EXE
DLL
EXE
Java virtuális gép
Végrehajtható program vagy dinamikusan szerkeszthető könyvtár
Operációs rendszer
Processzusok ütemezése, erőforrások kezelése, fájlkezelés, hardver absztrakciós szint, …
BIOS, hardver közvetlen elérése


C
C
C
C++
Java
Java
Fordítás
ASM
ASM
ASM
Fordítás
Fordítás
OBJ
OBJ
OBJ
OBJ
OBJ
OBJ
CLASS
CLASS
Szerkesztés
!
Szerkesztés
JAR
EXE
EXE
EXE
EXE
Java virtuális gép
Különböző forrásnyelvekből származó tárgykódok akkor szerkeszthetők össze, amennyiben megegyező formátumúak, valamint egyeznek a hívási konvenciók, paraméterátadások, a memóriamodell, …
Operációs rendszer
Processzusok ütemezése, erőforrások kezelése, fájlkezelés, hardver absztrakciós szint, …
BIOS, hardver közvetlen elérése


C
C
C
C++
Java
Java
ASM
ASM
ASM
OBJ
OBJ
OBJ
OBJ
OBJ
OBJ
CLASS
CLASS
!
JAR
EXE
EXE
EXE
EXE
Java virtuális gép
Assembly:
hardverközeli és/vagy hatékony implementációt igénylő részek
Magas szintű nyelv:
GUI, vezérlés, Assembly betétek hívása
Operációs rendszer
Processzusok ütemezése, erőforrások kezelése, fájlkezelés, hardver absztrakciós szint, …
BIOS, hardver közvetlen elérése

Egyszerű példa

C forráskód

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
  int a, b, sum;

  a = 5;
  b = 7;
  sum = a + b;

  printf( "Osszeg: %d\n", sum );

  return 0;
}
```

Generált Assembly kód (Intel Ubuntu Linux)

```
        .file   "sum.c"
        .section        .rodata
.LC0:
        .string "Osszeg: %d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        andl    $-16, %esp
        subl    $32, %esp
        movl    $5, 28(%esp)
        movl    $7, 24(%esp)
        movl    24(%esp), %eax
        movl    28(%esp), %edx
        leal    (%edx,%eax), %eax
        movl    %eax, 20(%esp)
        movl    $.LC0, %eax
        movl    20(%esp), %edx
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    printf
        movl    $0, %eax
        leave
        ret
        .size   main, .-main
        .ident  "GCC: (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2"
        .section        .note.GNU-stack,"",@progbits
```

Generált Assembly kód (SUN Sparc)

```
        .file   "sum.c"
gcc2_compiled.:
.section        ".rodata"
        .align 8
.LLC0:
        .asciz  "Osszeg: %d\n"
.section        ".text"
        .align 4
        .global main
        .type    main,#function
        .proc   04
main:
        !#PROLOGUE# 0
        save    %sp, -128, %sp
        !#PROLOGUE# 1
        st      %i0, [%fp+68]
        st      %i1, [%fp+72]
        mov     5, %o0
        st      %o0, [%fp-20]
        mov     7, %o0
        st      %o0, [%fp-24]
        ld      [%fp-20], %o0
        ld      [%fp-24], %o1
        add     %o0, %o1, %o0
        st      %o0, [%fp-28]
        sethi   %hi(.LLC0), %o1
        or      %o1, %lo(.LLC0), %o0
        ld      [%fp-28], %o1
        call    printf, 0
         nop
        mov     0, %i0
        b       .LL2
         nop
.LL2:
        ret
        restore
.LLfe1:
        .size    main,.LLfe1-main
        .ident  "GCC: (GNU) 2.95.3 20010315 (release)"
```

Tárgykód (Intel Borland C fordító)

```
Lister - [D:\Study\Assembly\src\sum.obj]
File  Edit  Options  Encoding  Help                                        100 %
00000000: 80 07 00 05 73 75 6D 2E|63 00 88 15 00 00 00 11 |    |sum.c .
00000010: 42 6F 72 6C 61 6E 64 20|43 2B 2B 20 35 2E 35 2E | Borland C++ 5.5.
00000020: 31 00 88 0E 00 00 E8 00|05 73 75 6D 2E 63 2E 9A | 1 .  č |sum.c.š
00000030: 25 40 00 88 29 00 00 E9|20 28 DB 28 21 63 3A 5C | %@ .)  é (Ű(!c:\
00000040: 42 6F 72 6C 61 6E 64 5C|42 63 63 35 35 5C 69 6E | Borland\Bcc55\in
00000050: 63 6C 75 64 65 5C 5F 6E|66 69 6C 65 2E 68 00 88 | clude\_nfile.h .
00000060: 28 00 00 E9 20 28 DB 28|20 63 3A 5C 42 6F 72 6C | (  é (Ű( c:\Borl
00000070: 61 6E 64 5C 42 63 63 35|35 5C 69 6E 63 6C 75 64 | and\Bcc55\includ
00000080: 65 5C 5F 6E 75 6C 6C 2E|68 00 88 28 00 00 E9 20 | e\_null.h .(  é
00000090: 28 DB 28 20 63 3A 5C 42|6F 72 6C 61 6E 64 5C 42 | (Ű( c:\Borland\B
000000A0: 63 63 35 35 5C 69 6E 63|6C 75 64 65 5C 5F 64 65 | cc55\include\_de
000000B0: 66 73 2E 68 00 88 2A 00|00 E9 20 28 DB 28 22 63 | fs.h .*  é (Ű("c
000000C0: 3A 5C 42 6F 72 6C 61 6E|64 5C 42 63 63 35 35 5C | :\Borland\Bcc55\
000000D0: 69 6E 63 6C 75 64 65 5C|5F 73 74 64 64 65 66 2E | include\_stddef.
000000E0: 68 00 88 28 00 00 E9 20|28 DB 28 20 63 3A 5C 42 | h .(  é (Ű( c:\B
000000F0: 6F 72 6C 61 6E 64 5C 42|63 63 35 35 5C 69 6E 63 | orland\Bcc55\inc
00000100: 6C 75 64 65 5C 73 74 64|69 6F 2E 68 00 88 0D 00 | lude\stdio.h ..
00000110: 00 E9 2E 9A 25 40 05 73|75 6D 2E 63 00 88 03 00 |  é.š%@|sum.c .
00000120: 00 E9 00 96 28 00 05 5F|54 45 58 54 04 43 4F 44 |  é -( |_TEXT COD
00000130: 45 00 05 5F 44 41 54 41|04 44 41 54 41 06 44 47 | E |_DATA DATA DG
00000140: 52 4F 55 50 04 5F 42 53|53 03 42 53 53 00 98 07 | ROUP _BSS BSS .
00000150: 00 A9 23 00 01 02 00 00|98 07 00 A9 0C 00 04 05 |  ©#  .    ©
00000160: 00 00 98 07 00 A9 00 00|07 08 00 00 9A 06 00 06 |   .  ©      š
00000170: FF 03 FF 02 00 8C 17 00|0B 5F 5F 73 65 74 61 72 |        Ś  __setar
00000180: 67 76 5F 5F 00 07 5F 70|72 69 6E 74 66 00 00 90 | gv__  _printf  .
00000190: 0C 00 00 01 05 5F 6D 61|69 6E 00 00 00 00 A0 27 |     .|_main
000001A0: 00 01 00 00 55 8B EC B8|05 00 00 00 BA 07 00 00 |  .  U‹ě¸|    ş
000001B0: 00 03 D0 8B C2 50 68 00|00 00 00 E8 00 00 00 00 |   Đ‹ÂPh     č
000001C0: 83 C4 08 33 C0 5D C3 00|9D 09 00 E4 13 54 02 A4 | .Ä 3Ŕ]Ă ť. ä T
000001D0: 18 56 02 00 A0 10 00 02|00 00 4F 73 73 7A 65 67 |  V         Osszeg
000001E0: 3A 20 25 64 0A 00 00 8B|02 00 00 00             | : %d.  ‹
```

Fájl pozíció

Fájl tartalma: kód, adat és járulékos (pl. szerkesztőnek szóló) információk.

Középen hexa számokként, jobb oszlopban karakterkódokként.

Assembly programozás
Adatformátumok
8086 címzési módjai

Számrendszerek

- **Fontos számrendszerek Assembly nyelv esetén**
  - **Bináris**: 2-es számrendszer
    - 0 és 1 számjegyek
  - **Decimális**: 10-es számrendszer
    - 0-9 számjegyek
  - **Hexadecimális**: 16-os számrendszer
    - 0-9 számjegyek, A-F betűk
- **Átváltás számrendszerek között, aritmetikai műveletek**
  - Gyakorlaton, illetve korábban a Számítógép architektúrák előadáson

Adatformátumok

- **ASCII**
  - A bájtok értékei karaktereket jelentenek
  - Eredetileg az alsó 7 bit használatos
    - 0-31: vezérlő karakterek (pl. új sor, ESC, szöveg vége)
    - 32: szóköz
    - 33-47: írásjelek
    - 48-57: számjegyek
    - 58-64: írásjelek
    - 65-90: nagybetűk
    - 91-96: írásjelek
    - 97-122: kisbetűk
    - 123-126: írásjelek
    - 127: DEL

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Ábra forrása: Wikipedia

- **Kiterjesztett ASCII**
  - 128-255 közötti értékek is definiáltak (pl. nemzeti karakterek, szimbólumok)

Adatformátumok

- **Egész számok**
  - **Előjel nélküli bájt**
    - Bitek kettes számrendszerbeli értékének megfelelően
  - **Előjeles bájt**
    - A legmagasabb helyérték -128 értékű, a többi pozitív
    - **Kettes komplemens** ábrázolás:
      - Pozitív értékből negatív: minden bitet ellentettre (egyes komplemens), majd 1 érték hozzáadása
        - » Decimális 8 kettes számrendszerben: 00001000
        - » Ellentett képzés: 11110111
        - » 1 hozzáadása: 11111000

Adatformátumok

- **Egész számok**
  - **Szó**
    - Regiszter szélességének megfelelő méret
      - 8086: 16 bites (2 bájtos) egész
    - Előjel nélküli vagy előjeles (kettes komplemens)
  - **Dupla szó**
    - Két szó
      - 8086: 32 bites (4 bájtos) egész
- **Valós számok**
  - Közelítő érték tárolása
  - Előjel bit, mantissza és exponens
  - IEEE-754 szabvány (ld. Architektúrák előadás)
    - Egyszeres pontosság: 4 bájt
    - Dupla pontosság: 8 bájt

Processzor családok

- **RISC, CISC** és vegyes architektúrák
- **Architektúránként rendszerint különböző Assembly nyelv**
- **Néhány ismertebb család**
  - Intel 80x86
  - Sun UltraSparc I-IV
  - MIPS R3000-R10000
  - Motorola 68000
  - MOS 650x (Commodore)
  - Zilog Z80

Intel 80x86 processzor család

Lapka	Dátum	MHz	Tranz.	Mem.	Megjegyzés
I-4004	1971/4	0.108	2300	640	Első egylapkás mikroprocesszor
I-8008	1972/4	0.108	3500	16 KB	Első 8 bites mikroprocesszor
I-8080	1974/4	2	6000	64 KB	Első általános célú mikroprocesszor
I-8086	**1978/6**	**5-10**	**29000**	**1 MB**	**Első 16 bites mikroprocesszor**
I-8088	**1979/6**	**5-8**	**29000**	**1 MB**	**Az IBM PC processzora**
I-80286	**1982/6**	**8-12**	**134000**	**16 MB**	**Védett üzemmód**
I-80386	**1985/10**	**16-33**	**275000**	**4 GB**	**Első 32 bites mikroprocesszor, virtuális 8086 üzemmód**
I-80486	1989/4	25-100	1.2M	4 GB	8 KB beépített gyorsítótár
Pentium	1993/5	60-233	3.1M	4 GB	Két csővezeték, MMX
P. Pro	1995/3	150-200	5.5M	4 GB	Két szintű beépített gyorsítótár
P. II	1997/5	233-400	7.5M	4 GB	Pentium Pro + MMX
P. III	1999/2	650-1400	9.5M	4 GB	SSE utasítások 3D grafikához
P. 4	2000/11	1300-3800	42M	4 GB	Hyperthreading + több SSE

80x86 memória modell

- **Operatív memória felépítése**
  - Bit: Alapegység , 0 vagy 1 érték
  - Cella: Legkisebb címezhető egység
    - 8 bit = 1 bájt
  - Szó: Regiszterek bit-szélességének megfelelő
    - 16, 32 vagy 64 bit
  - Paragrafus: 16 bájt, 16-tal osztható címen kezdődik
  - Lap: 256 (512, 1024, akár több) bájt
  - Bájt sorrend: Little-endian
    - legkisebb helyértékű rész szerepel elől
  - Negatív számok kettes komplemens alakban

80x86 memória modell

- **Operatív memória elérése**
  - Lineáris cím
    - Bájtok sorszámozása 0-tól
    - Bájt címe az indexe
  - **Szegmentált cím**
    - BÁZISCÍM : ELTOLÁS (Szegmens : Offszet) alakban
    - Szegmensen belüli relatív címek használhatók
    - Könnyebb relokáció
      - A szegmens áthelyezhető a memória más részére a címzések és vezérlésátadások módosítása nélkül
    - **8086 ezt alkalmazza**

8086 szegmentált címképzése

- **Tulajdonságok**
  - **20 bites címbusz**, de csak **16 bites regiszterek**
  - **1 regiszter nem elegendő** a címzéshez!
  - **Használjunk kettőt**
    - SZEGMENS : OFFSZET alakban írjuk
    - Fizikai cím előállítása: a SZEGMENS értéke 16-tal (10H) szorzódik / 4 bittel balra tolódik / hexa 0 íródik utána
    - Hozzáadódik az OFFSZET értéke
  - **Vagyis**
    - Egy szegmens mérete maximálisan 64 KB lehet
    - Egy fizikai címnek több szegmentált címe létezik!
    - Túlcsordulás kezelés nincs
      - FFFF:0010 az első (0.) memóriapozíciót jelenti!
  - Könnyebb **relokáció**
  - Az egyes szegmensek részben vagy teljesen **átfedőek lehetnek**!

```
SZEGMENS: 1234H
OFFSZET:  0012H

 12340H    * 10H
+ 0012H    + OF.
-------
 12352H
```

8086 szegmensek

- **Kialakítása**
  - Törekedjünk az **adat és programszegmensek számának csökkentésére**
    - Kevesebbszer kelljen szegmenst váltani
  - Programszegmensek között váltásra **lehetőleg eljáráshívást (CALL) használjunk** a feltétel nélküli ugrások (JMP) helyett
    - Eljárásból visszatéréskor automatikusan visszaállítódik a CS
    - Programból szabályosan kilépni abból a kódszegmensből kell, ahol a belépési pont volt

8086 regiszterkészlete

- **Szegmens regiszterek**
  - CS, DS, ES, SS
- **Speciális célú regiszterek**
  - IP, SP, FLAGS
- **Index regiszterek**
  - BP, SI, DI
- **Általános célú regiszterek**
  - AX, BX, CX, DX

- 16 bites regiszterek

8086 regiszterkészlete

- **Szegmens regiszterek**
  - CS (Code Segment): kód szegmens
    - Csak ugró utasítással módosítható
  - DS (Data Segment): adat szegmens
  - ES (Extra Segment): (másodlagos) adat szegmens
  - SS (Stack Segment): verem szegmens

8086 regiszterkészlete

- **Speciális célú regiszterek**
  - IP: Utasítás számláló (Csak vezérlésátadással írható felül)
  - SP: Veremmutató (Verem műveletek állítják)
  - FLAGS: CPU állapotát jelző bitek összessége
    - C (Carry): Átvitel előjel nélküli műveleteknél
    - P (Parity): Az eredmény alsó 8 bitjének paritása
    - A (Aux Carry): Átvitel a 3. és 4. bitek között (BCD számoknál)
    - Z (Zero): 1 (igaz), ha az eredmény 0, különben 0 (hamis)
    - S (Sign): Az eredmény legmagasabb helyiértékű bitje (előjel)
    - T (Trap): 1: debug mód, 0: automatikus
    - I (Interrupt): 1: maszkolható megszakítás engedélyezve, 0: tiltva
    - D (Direction): Sztring műveletek iránya 1: csökkenő, 0: növekvő
    - O (Overflow): Előjeles túlcsordulás

8086 regiszterkészlete

- **Index regiszterek**
  - BP: Bázis mutató
    Egyedi szerep verem indexelt címzésére
  - SI: Forrás index
    Egyedi szerep sztring műveleteknél
  - DI: Cél index
    Egyedi szerep sztring műveleteknél

8086 regiszterkészlete

- **Általános célú regiszterek**
  - AX: Akkumulátor, alsó és felső bájtja AL és AH
    Egyedi szerep szorzásnál és osztásnál
  - BX: Bázis index, alsó és felső bájtja BL és BH
  - CX: Számláló, alsó és felső bájtja CL és CH
    Egyedi szerep bitléptető, sztring és ismétléses műveleteknél
  - DX: Adat, alsó és felső bájtja DL és DH
    Egyedi szerep szorzásnál és osztásnál

8086 utasításszerkezete

- **Fordítás**
  - 1 Assembly utasítás -> 1 gépi kódú utasítás
- **Általános szerkezet**

Prefixum	Operációs kód	Címzési mód	Operandus(ok)
Ismételtetett végrehajtás, sín zárolása, szegmens regiszter átdefiniálása	Mit kell csinálni	Operandus értelmezése, ha az nem egyértelmű	Min kell a műveletet elvégezni
Opcionális 1-2 bájt	Kötelező 1-2 bájt	Utasítás függő 0-1 bájt	Utasítás függő 0-4 bájt

További részletek (pl. címzési mód bájt értelmezése) nem kellenek!

Memóriacímzési módok

- **Adatterület címzése**
  - MOV utasításon keresztül
    - `MOV op1,op2`
    - Adat mozgatása op2-ből op1-be
    - op1 és op2 regiszter vagy memóriacím
    - Legalább az egyik operandus regiszter kell legyen!
- **Kódterület címzése**
  - Vezérlésátadási lehetőségek
  - Rövid, közeli és távoli ugrások

Adatterület címzése

- **Kódba épített adat (közvetlen címzés)**
  - Az ***adat*** (operandus) az ***utasítás része***

```
MOV AX,07FFH ; AX tartalma 07FFH lesz
MOV CL,34H   ; CL 34H lesz,
             ; CH nem változik meg!
             ; Ha CX 1256H volt, akkor
             ; ezután 1234H lesz.
MOV DS,02H   ; szegmens regiszterbe nem!
MOV 34H,CL   ; érték nem lehet cél!
```

Adatterület címzése

- **Direkt memóriacímzés**
  - Az ***operandus címe*** az utasítás része
  - Assembly kódban **címkét** használunk

```
ADAT DW 34F2H ; 1 szóhossznyi adat
;...
MOV AX,ADAT ; AX tartalma 34F2H lesz
```

  - Disassemblerben így láthatjuk

```
MOV AX,[07FE] ; amennyiben ADAT
              ; címe 07FEH volt
```

Adatterület címzése

- **Direkt memóriacímzés**
  - Az ***adatok típusának*** egyezni kell!

```
ADAT1 DB 12H,34H
ADAT2 DW 1234H   ; Little-endian
                 ; tárolás: 34H 12H
;...
MOV AX,ADAT1 ; hibás, mert ADAT1 bájt!
MOV AL,BYTE PTR ADAT2; OK, típus
                     ; átalakítás: AL=34H
```

Adatterület címzése

- **Indexelt címzés**
  - Az utasításban elhelyezett számot (eltolás) hozzáadja a kiválasztott [index regiszterhez] (SI vagy DI)

```
MOV AX,07FH[ DI ]
; AX tartalma DS * 10H + DI + 7FH
; memóriacím szavas tartalma lesz

MOV 07FH[ DI ],AX
; AX értéke a fenti memóriacímre kerül
```

Adatterület címzése

- **Indexelt címzés**
  - **Túlcsordulást** nem kezel:

```
MOV AX,07FFH[ DI ]
```

    - Ha DI = FF00H és az eltolás 07FFH, akkor a virtuális cím 06FFH lesz!
  - **Bájtos negatív eltolás** esetén előjel kiterjesztésre vigyázzunk!

```
MOV AX,0F8H[ DI ]
```

    - Ha DI = 3265H és az eltolás 0F8H, akkor a cím 3265H + F8H = 3265H + FFF8H = 325DH lesz, vagyis kivontunk 8-at!

Adatterület címzése

- **Regiszter indirekt címzés**
  - Az előző, indexelt címzés speciális esete, amikor az eltolást nem használjuk (0 az értéke)

```
MOV AX,[ DI ]
; AX tartalma DS * 10H + DI
; memóriacím szavas tartalma lesz

MOV [ DI ],AX
; A DS * 10H + DI memóriacímen
; elhelyezésre kerül AX tartalma
```

Adatterület címzése

- **Bázisrelatív címzés**
  - Mint az indexelt címzés, csak itt BX-et használjuk az index regiszterek helyett

```
MOV AX,057FH[ BX ]
```

  - Kombinálható index regiszterrel is

```
MOV AX,057FH[ SI ][ BX ]
; AX értéke DS * 10H + SI + BX + 057FH
; címen található szó értéke lesz
MOV AX,[ BX + SI + 057FH ] ; ugyanaz
```

Adatterület címzése

- **Verem címzés**
  - Verem
    - LIFO-elvű (utoljára be – először ki) adatterület
    - SP regiszter automatikusan állítódik eljáráshíváskor és veremműveletek esetén (csökken berakáskor, növekszik kivételkor)
    - SS:SP a verem tetejét (legfelső elemét) mutatja
  - Használható például
    - Regiszterek értékének átmeneti tárolására
    - Eljárás híváskor a visszatérési cím tárolására
    - Eljárásnak paraméterek átadására

Adatterület címzése

- **Verem címzés**
  - Eljárás, amely vermen keresztül kap két paramétert.
  - Itt tárolódik az eljárás visszatérési címe, valamint példánkban egy belső változó is.
  - BP-ben „megjegyezzük” SS értékét a belépéskor, mivel SS automatikusan változik

```
PUSH BP ; BP mentése verembe
MOV BP,SP ;veremmutató BP-be
MOV AX,3465H ; belső változó
PUSH AX       ; verembe

MOV AX,4[ BP ] ; első param.
MOV BX,6[ BP ] ; 2. param.
MOV CX,-2[ BP ] ; belső vált

POP BP ; belső változó ki
POP BP ; BP visszaállítása

RET ; visszatérés
```

Adatterület címzése

Átadott értékek legyenek
1234H és 5678H.

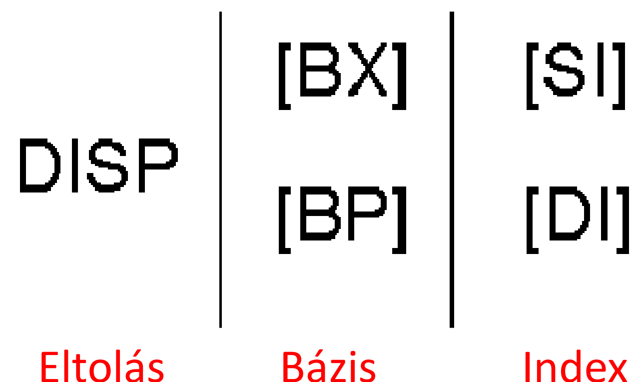
```
SS: ....   ; SS autom.
    ....   ; változik
    3465H  ; belső vált.
BP: xxxxH  ; BP mentése
    xxxxH  ; visszatér.c.
    5678H  ; második par.
    1234H  ; első param.
```

Verem tartalma
Alacsonyabb címek felé bővül

```
PUSH BP ; BP mentése verembe
MOV BP,SP ;veremmutató BP-be
MOV AX,3465H ; belső változó
PUSH AX      ; verembe

MOV AX,6[ BP ] ; első param.
MOV BX,4[ BP ] ; 2. param.
MOV CX,-2[ BP ] ; belső vált

POP BP ; belső változó ki
POP BP ; BP visszaállítása

RET ; visszatérés

; hívó kód takarítja a
; veremből a paramétereket
```

Memóriacímzések összefoglalása

- **Érvényes hivatkozások összefoglaló szabálya**



  - A mindhárom csoportból válasszunk 0 vagy 1 elemet
  - Legalább 1 kiválasztásra kerüljön a háromból
  - Összesen 17 féle lehetőség

Adatterület címzése

- **Regisztercímzés**
  - Az operandusok regiszterek

```
MOV AX,BX
MOV BP,SP
```

  - **Nem megengedett esetek**re példa

```
MOV BX,AL ; hibás, eltérő típusok!
MOV CS,AX ; CS nem írható!
MOV ES,DS ; hibás, szegmens regiszterek
          ; között közvetlenül nem megy!
MOV DS,DAT_SEG ; hibás, DS,ES,SS adattal
               ; közvetlenül nem írható!
```

Kódterület címzése

	Követlen (direkt)	IP-relatív	Regiszter indirekt	Címzett indirekt	Megjegyzés
Rövid ugrás	?	-128, +127 közötti távolság, IP-hez adódik	-	-	Feltételes ugrások csak ilyenek!
Közeli ugrás	?	-32768, +32767 közötti távolság, IP-hez adódik	JMP AX Regiszter -> IP	JMP [DI+2] Memóriatartalom -> IP	
Távoli (szegmens közi) ugrás	Műveleti kód után IP és CS új értéke	-		JMP **FAR PTR** [AX] AX címen található két szó kerül IP, CS-be	

Megjegyzések:

A szegmensek címzése „körkörös", nincs túlcsordulás kezelés!

'?': Egyes könyvek állítják, hogy ilyen létezik, de nem írják le, hogyan érhető el

Kódterület címzése

- **Rossz hír**
  - Ellentmondó szakirodalom
- **Jó hír**
  - Ezzel nem kell foglalkoznunk
  - **Címkéket használva a fordító (assembler) feladata a megfelelő címzés kiválasztása és használata**
    1. **Rövid IP-relatív ugrás** (2 bájt: opkód + távolság)
       - ha szegmensen belüli és a távolság -128 és +127 közötti,
       - vagy feltételes ugrás
    2. **Közeli IP-relatív ugrás** (3 bájt: opkód + távolság)
       - ha nagyobb, de szegmensen belüli (feltétel nélküli)
    3. **Direkt ugrás** (5 bájt: opkód + IP + CS új értéke)
       - szegmensen kívüli vagy FAR címkére történik (feltétel nélküli)

Alapértelmezett szegmens regiszterek

- **Aktuális művelet címe**
  - CS:IP, nem definiálható felül!
- **Verem**
  - SS az alapértelmezett szegmens regiszter
  - Csak BP és SP használható címzésre, minden más módhoz felüldefiniálás kell (pl. `MOV AX,SS:[SI]`).
- **Adat szegmens**
  - BX, DI, SI, 16-bites szám esetén DS az alapértelmezett
  - Sztring műveleteknél DI alapértelmezett szegmens regisztere ES (DS:SI a forrás, ES:DI a cél)

Alapértelmezés felülbírálása

- **Alapértelmezett szegmens regiszter felülbírálható**
  - Prefix alkalmazásával

```
MOV AX,DS:[BP] ; verem helyett adat szegmens
MOV AX,ES:[BP] ; verem helyett extra szegmens
MOV AX,SS:[DI] ; adat szegmens helyett verem sz.
MOV AX,CS:[SI] ; adat szegmens helyett kód sz.
MOV AX,ES:LIST ; adat szegmens helyett extra sz.
LODS ES:DATA   ; adat szegmens helyett extra sz.
MOV AX,SS:[SI]
```

További szabályok

- **Kétoperandusú műveleteknél**
  - legalább egyik operandusa regiszter kell legyen,
  - a két operandus mérete egyező kell legyen,
  - a két operandus nem lehet két szegmens regiszter.
- **DS, SS, ES beépített adattal nem írható**
  - Először valamelyik általános célú regiszterbe töltsük az értéket, majd innen a szegmens regiszterbe
- **SS írásával vigyázzunk**, mert elvész a verem!
- **Szegmens regiszter** nem lehet aritmetikai művelet operandusa
- **CS és IP regiszter** nem írható, nem tehető verembe, értéke csak ugrással állítható

```
MOV AX,[SI] ; OK
MOV DS,AX   ; OK
MOV [SI],[DI] ; hiba!
```

```
ADAT SEGMENT
SZAM DB 00,01,02
ADAT ENDS
; ...
MOV DS,ADAT ; hiba!
MOV AX,ADAT ; cím be
MOV DS,AX   ; OK
```

```
MOV CS,AX ; lefagy!
JMP cimke      ; OK
CALL eljaras ; OK
JNE ciklus     ; OK
```

Assembly programozás
8086 aritmetikai utasításai

Aritmetikai utasítások

- **Összeadás, kivonás**
  - Előjeles (kettes komplemens): **`ADD`**, **`ADC`**, **`SUB`**, **`SBB`**
  - BCD: `DAA`, `DAS`
  - ASCII: `AAA`, `AAS`
- **Szorzás, osztás**
  - Előjeles, előjel nélküli: **`MUL`**, **`IMUL`**, **`DIV`**, **`IDIV`**
  - BCD: `AAD`, `AAM`
- **Növelés, csökkentés, negálás**
  - `INC`, `DEC`, `NEG`
- **Előjel kiterjesztés**
  - `CBW`, `CWD`
- **Összehasonlítás**
  - `CMP`

Aritmetikai utasítások

- **Hatásuk a FLAGS regiszter bitjeire**
  - **I, D, T**: nem változnak
  - **C, P, A, Z, S, O**: változhatnak
    - Átvitel (C): 8 vagy 16 biten nem fér el az eredmény
    - Paritás (P): 1-es értékű bitek (csak az alsó bájton!)
    - Másodlagos átvitel (A): 3. és 4. bitek közötti átvitel
    - Zéró (Z): eredmény nulla-e?
    - Előjel (S): eredmény legmagasabb bitje
    - Túlcsordulás (O): 1, ha az eredmény nem fért el az adott típus értéktartományán
- **Az eredmény elkészülte mellett**
  - Feltételes ugrásokat végezhetünk a flag bitek alapján
  - C és O vizsgálatával az eredmény előjelét megállapíthatjuk

Összeadás (ADD)

- **Előjeles (kettes komplemens), átvitel nélkül**
  - Z, C, A, S, P, O flageket állítja

```
ADD AX,BX    ; AX = AX + BX
ADD AL,BL    ; AL = AL + BL
ADD CL,44H   ; CL = CL + 44H
ADD [BX],AL  ; AL értéke hozzáadódik BX által
             ; mutatott memóriacím értékéhez,
             ; tárolás a memóriacímen
```

~~ADD DS,4~~ ; szegmens regiszterhez nem!

~~ADD [BX],ADAT~~ ; legalább 1 regiszter kell!

Összeadás (ADC)

- **Előjeles (kettes komplemens), átvitellel**
  - Z, C, A, S, P, O flageket állítja
  - C flag bit értékét is hozzáadja az összeghez

```
ADC AL,AH    ; AL = AL + AH + C
ADC CL,44H   ; CL = CL + 44H + C

; 32 bites összeadás BX-AX + DX-CX
ADD AX,CX ; alsó 16 bites rész, átvitel C-be
ADC BX,DX ; felső 16 bit: BX + DX + C
```

Növelés, csökkentés, negálás

- `INC op`
  - Operandus értékét 1-gyel növeli
  - 8 vagy 16 bites
  - **C flaget NEM állítja**

- `DEC op`
  - Operandus értékét 1-gyel csökkenti
  - 8 vagy 16 bites
  - **C flaget NEM állítja**

- `NEG op`
  - Operandus ellentettjét képzi
  - Kettes komplemens alakban!

Kivonás (SUB)

- **Előjeles (kettes komplemens), átvitel nélkül**
  - Z, C, A, S, P, O flageket állítja

```
SUB AX,BX    ; AX = AX – BX
SUB CL,44H   ; CL = CL - 44H
SUB [BX],AL  ; AL értéke kivonódik a BX által
             ; mutatott memóriacím értékéből,
             ; tárolás a memóriacímen
MOV CH,22H ; decimális 34-ből
SUB CH,44H ; decimális 68 kivonása
           ; az eredmény DEH lesz, ami
           ; helyes (-34)
           ; nincs túlcsordulás sem!
```

Kivonás (SBB)

- **Előjeles (kettes komplemens), átvitellel**
  - Z, C, A, S, P, O flageket állítja
  - C flag bit értékét is kivonja

```
SBB AL,AH    ; AL = AL – AH - C
SBB CL,44H   ; CL = CL - 44H - C
```

Összehasonlítás (CMP)

- **Hatása**

```
CMP op1,op2
```

  - Az operandusok nem változnak!
  - A flag-ek az `op1-op2` kivonásnak megfelelően állnak be
  - Feltételes vezérlésátadáshoz jól felhasználható

Szorzás (IMUL, MUL)

- **Általános szabály 8-bites szorzásra**
  - A szorzandó mindig az AL regiszter!
  - Eredmény AX-ben képződik (16 bites eredmény)!

- **8-bites előjeles szorzás: `IMUL`**

```
IMUL DH ; AX = AL * DH
```

- **8-bites előjel nélküli szorzás: `MUL`**

```
MOV BL,5    ; BL előkészítése
MOV CL,100  ; CL előkészítése
MOV AL,CL   ; CL AL-be töltése
MUL BL      ; AX = AL * BL
MOV DX,AX   ; DX = AX = AL * BL = CL * BL
```

Szorzás

- **Általános szabály 16-bites szorzásra**
  - A szorzandó mindig az AX regiszter!
  - Eredmény DX-AX regiszterekben képződik (32 bites eredmény)!
  - IMUL, MUL utasítások, mint 8-bites esetben

```
MUL CX ; DX-AX = AX * CX
MUL WORD PTR [ SI ] ; DX-AX = AX *
  ; SI helyen a memóriában található
  ; szó értéke
  ; kell a WORD PTR, mert a CPU nem
  ; tudja 8 vagy 16-bites a művelet
```

Osztás (IDIV, DIV)

- **Általános szabály**
  - **16 bites osztandó (AX), 8 bites osztó (operandus)**
    - AL a hányadost, AH az egész maradékot tartalmazza osztás után
    - Előjeles osztásnál a maradék előjele az osztandó előjele
  - **32 bites osztandó (DX-AX), 16 bites osztó**
    - AX a hányadost, DX az egész maradékot tartalmazza osztás után
    - Előjeles osztásnál a maradék előjele az osztandó előjele
  - **Az osztó szélességét ki kell terjeszteni!**
    - Előjel nélküli esetben 0-val (AH, illetve DX tartalma)
    - Előjeles esetben az előjel értékével
      - `CBW` (Convert Byte to Word): bájtból előjelhelyes szó
      - `CWD` (Convert Word to Double Word): szóból előjelhelyes dupla szó

Osztás (IDIV, DIV)

- **Vizsgálatok osztás előtt**
  - 0-val osztás hibát okoz!
  - Ha az osztás eredménye nem fér el AH-ban, illetve AX-ben, akkor a programunk kilép!
    - Pl. 8 bites osztásnál: ha AH >= op, akkor túl nagy az osztandó!

Osztás

- **Példa**

```
; 8 bites előjel nélküli osztás
MOV AL,NUMB ; NUMB adat betöltése
MOV AH,0    ; AH nullázása előjel nélküli
            ; osztás előtt
DIV NUMB1   ; előjel nélküli osztás

; 16 bites előjeles osztás
MOV AX,-100
MOV CX,9
CWD        ; DX-AX: előjelhelyes -100 érték
IDIV CX
```

Osztás

- **Példa: maradék vizsgálata kerekítéshez**

```
    DIV BL ;
    ADD AH,AH ; maradék duplázása
    CMP AH,BL ; nagyobb-e az osztónál?
    JB  NEXT  ; ha nem, akkor jó az eredmény
    INC AL    ; ha igen, akkor felfelé kerekít
NEXT: ...
```

Flag-ek viselkedése

- **1. példa**

  53 + 18 = 71
    00110101
  \+ 00010010
  \-----------
  0 01000111 Rendben, előjelhelyes is.

- **2. példa**

  53 + 83 = 136
    00110101
  \+ 01010011
  \----------- Ez -120! Előjel rossz,
  0 **1**0001000 túlcsordulás is van!
  16 bitesre kiterjesztve jó.

Flag-ek viselkedése

- **3. példa**

```
-53 + -18 = -71
  11001011
+ 11101110
----------
1 10111001    Rendben, de C=1!
```

- **4. példa**

```
-53 + -83 = -136
  11001011
+ 10101101
----------     Ez +120! Előjel is rossz,
1 01111000     túlcsordulás és átvitel
               is van!
```

Flag-ek viselkedése

- **5. példa**

```
-53 + 18 = -35
  11001011
+ 00010010
----------
0 11011101    Rendben (végre valami...)!
```

- **6. példa**

```
53 + -18 = 35
  00110101
+ 11101110
----------
1 00100011     Jó, de C=1!
```

Flag-ek viselkedése

- **7. példa**

```
-18 + 18 = 0
  11101110
+ 00010010
----------
1 00000000    Jó, de C=1!
```

Flag-ek viselkedése a példákban

- **Z (Zero) flag**
  - Csak a 7. példában 1, mert csak akkor 0 az eredmény. Más példákban 0.
- **S (Sign) flag**
  - Eredmény legmagasabb bitjének értéke
  - 2. és 4. példában nem a valódi előjelet adja! Belső átvitel elrontotta…
- **C (Carry) flag**
  - Legfelső biten keletkezett túlcsordulás
  - Logikus értelme csak azonos előjelű számok összeadásakor van a példákban: ekkor az eredmény helyes előjelét adja (S flag helyett…)
- **O (Overflow) flag**
  - Ha az eredmény értéke kívül esik az ábrázolható tartományon (itt: [-128,+127])
  - Összeadás után O alapján dönthető el, hogy az eredmény előjelhelyes-e?
    - Ha O flag értéke 1, akkor C a helyes előjel. (2. és 4. példa)
    - Ha O flag értéke 0, akkor S a helyes előjel. (A többi példában.)
- **P (Parity) flag**
  - 1-es értékű bitek paritása (számának párossága) az eredményben
  - Értéke 1 az 1., 2., 4., 5. és 7. példákban

Flag-ek viselkedése

- **Konklúzió**
  - O, S és C flag-ek alapján dönthetünk a helyes értékről
  - Magaszintű nyelveket is érinti ez a probléma, de ott nincs lehetőségünk a flag értékek vizsgálatára!
- **Fontos**
  - Olyan regiszter(eke)t célszerű választani, amelyben az eredmény is „elfér”
  - Ha pl. előfordulhat, hogy az összeg kivezet a [-128,+127] tartományból, akkor használjunk 16 bites regisztereket

Probléma magasszintű nyelvben

C forráskód

```
#include <stdio.h>
int main( int argc, char *argv[] )
{
  char a, b, c;
  a = 53;
  b = 83;
  c = a + b;
  printf( "Osszeg: %d\n", (int)c );

  return 0;
}
```

Eredmény

- -120 értéket kapunk!
- Az Assembly-vel ellentétben itt nem „látjuk” a flag-ek értékét a művelet elvégzése után!

Assembly programozás
Adatmozgató utasítások

Utasítások

- **Adatmozgatás operandusok között**
  - **MOV**
- **Operandusok felcserélése**
  - **XCHG**
- **Kódkonverzió táblázat alapján**
  - **XLAT**
- **Tényleges cím betöltése**
  - **LDS, LES, LEA**
- **Verem műveletek**
  - **PUSH**, **POP**, **PUSHF**, **POPF**
- Intel 8080 kompatibilitás miatt
  - LAHF, SAHF (nem kell!)

Mozgatás, csere

- **MOV op1, op2**
  - op2 tartalma op1-be
  - Legalább az egyik operandus regiszter kell legyen
  - Részletesen ismertetésre került a címzési módoknál

- **XCHG op1, op2**
  - op1 és op2 tartalmának felcserélése
  - Segédváltozó használata nélkül
  - Legalább az egyik operandus regiszter kell legyen

Kódkonverzió táblázat alapján

- **XLAT**
  - Táblázat bájtos adatokkal
  - Legfeljebb 256 érték szerepelhet benne
  - AL <- [AL + BX]
    - BX: táblázat kezdőcíme
    - AL: elem indexe
    - Eredmény: AL-be bekerül a táblázat adott indexű eleme

```
HEXTB DB    '0123456789ABCDEF'; hexa szjegyek
…
MOV AL,12D ; AL 12D
MOV BX,OFFSET HEXTB
XLAT ; AL értéke 'C' karakterkódja lesz
```

Tényleges cím betöltése

- Használatuk
  - Ki lehet váltani velük több műveletet
- **LDS reg,mem** (Load Data Segment)
  - Az adott címen lévő első két bájt *reg*-be kerül
  - A harmadik-negyedik bájt DS regiszterbe
- **LES reg,mem** (Load Extra Segment)
  - Ua. mint LDS, csak itt ES regiszter szerepel
- **LEA reg,mem** (Load Effective Address)
  - A memóriacímzés offszetje a kért regiszterbe kerül
  - Tetszőleges memóriacímzés eredménye 1 utasítással meghatározható
  - **Futási időben történik a számítás!**

```
LEA DI,322[ BX ][ DI ] ; BX + DI + 322
```

Verem műveletek

- **PUSH op**
  - Operandus értéke a verembe kerül (regiszter vagy memóriacím értéke)
  - Csak 16 bites érték mozgatható!
  - SS:[SP-2] = op; SP = SP - 2
  - A verem az alacsonyabb memóriacímek felé bővül!
- **POP op**
  - A verem tetejének értéke op-ba kerül
  - Csak 16 bites érték mozgatható!
  - Op = SS:[SP]; SP = SP + 2
- **PUSHF, POPF**
  - FLAGS regiszter tartalma kerül a verembe, illetve onnan visszaolvasásra

Verem inicializálása

- **Automatikus**
  - Ha a forráskódban megfelelően definiáltuk a verem szegmenst, akkor
    - SS felveszi a szegmenscímet,
    - SP pedig a lefoglalt verem terület legvégére mutat

    a program indulásakor.

Assembly programozás
Vezérlésátadó utasítások

Vezérlésátadás

- **Feltétel nélküli**
  - `JMP, CALL`
- **Feltételes**
  - FLAG regiszter bit-értékeinek megfelelően
    - Számos utasítás az S,Z,C,P,O FLAG bitek vizsgálatával
    - FLAG biteket aritmetikai, logikai, bitforgató, flag beállító műveletek módosítanak, ezek után lehet feltételes ugrás
  - Csak *rövid* feltételes ugrás lehetséges!
    - -128,+127 tartományon belül!
  - Ciklusszervezés
    - `LOOP` utasítás, CX számláló regiszter felhasználásával

Kódterület címzése (ismétlés)

	Követlen (direkt)	IP-relatív	Regiszter indirekt	Címzett indirekt	Megjegyzés
Rövid ugrás	?	-128, +127 közötti távolság, IP-hez adódik	-	-	Feltételes ugrások csak ilyenek!
Közeli ugrás	?	-32768, +32767 közötti távolság, IP-hez adódik	JMP AX Regiszter -> IP	JMP [DI+2] Memóriatartalom -> IP	
Távoli (szegmens közi) ugrás	Műveleti kód után IP és CS új értéke	-		JMP **FAR PTR** [AX] AX címen található két szó kerül IP, CS-be	

Megjegyzések:

A szegmensek címzése „körkörös”, nincs túlcsordulás kezelés!

‘?’: Egyes könyvek állítják, hogy ilyen létezik, de nem írják le, hogyan érhető el

Kódterület címzése (ismétlés)

- **Rossz hír**
  - Ellentmondó szakirodalom
- **Jó hír**
  - Ezzel nem kell foglalkoznunk
  - Címkéket használva a fordító (assembler) feladata a megfelelő címzés kiválasztása és használata
    1. Rövid IP-relatív ugrás (2 bájt: opkód + távolság)
       - ha szegmensen belüli és a távolság -128 és +127 közötti,
       - vagy feltételes ugrás
    2. Közeli IP-relatív ugrás (3 bájt: opkód + távolság)
       - ha nagyobb, de szegmensen belüli (feltétel nélküli)
    3. Direkt ugrás (5 bájt: opkód + IP + CS új értéke)
       - szegmensen kívüli vagy FAR címkére történik (feltétel nélküli)

Feltétel nélküli vezérlésátadás

- **Ugrás**
  - Rövid, közeli, vagy távoli címre (NEAR, FAR)
  - `JMP címke`
- **Eljárás**
  - Rövid, közeli, vagy távoli címre (NEAR, FAR)
  - `CALL címke`
    - Vezérlésátadás a címre, visszatérési cím a verembe
  - `RET`
    - Visszatérés a verem tetején található címre
  - `RET n`
    - Visszatérés, és n darab érték kivétele a veremből
    - Vermen keresztül átadott paraméterek takarítására
  - Célszerűbb eljárásokat használni ugrások helyett!

Feltételes vezérlésátadás

- **FLAG bitek vagy CX regiszter vizsgálata**
  - Ugrás, ha feltétel teljesül
  - Következő utasítás végrehajtása, ha nem

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

Jump, Not, Equals, Zero, Less than, Greater than, Above, Below

Flag/reg.	0 értékű	1 értékű
O	JNO	JO
P	JNP, JPO	JP, JPE
S	JNS	JS
C	JNC	JC
Z	JNZ, JNE	JZ, JE
CX	JCXZ	

Jump, Overflow, Parity, Sign, Carry, Zero, CX, Not, Equals, Parity Even, Parity Odd, Zero

Ciklusok szervezése

- **Utasítások**
  - **`LOOP címke`**
    - CX számláló regiszter csökkentése 1 értékkel, ugrás ha még nem nulla az értéke
    - Egyébként következő utasítás jön
  - **`LOOPNZ, LOOPNE`**
    - Ha Z FLAG értéke 0, akkor CX csökkentése, ugrás ha az még nem nulla
    - Egyébként következő utasítás jön
  - **`LOOPZ, LOOPE`**
    - Ha Z FLAG értéke 1, akkor CX csökkentése, ugrás ha az még nem nulla
    - Egyébként következő utasítás jön

Rövid ugrás kikerülése

- **Probléma**
  - Feltételes ugrások csak rövid távolságra használhatók (-128,+127 bájt között)
- **Megoldás**

```
Cimke:
    ; ... hosszú kód rész, > 128 bájt
    JNZ Címke ; nem jó, túl nagy távolság!
    JZ CimkeZ ; OK, mivel 3 bájtot kell ugrani
    JMP Cimke ; OK, mivel JMP mehet közeli címre
CimkeZ:
    ; Folytatás
```

Példaprogram #1

- **Adatszegmensben található tömbök értékeinek összeadása**
  - Konstans méretű és megadott kezdőértékű tömbök
  - Az eredmény ezekkel egyező méretű tömb, előre lefoglalt területen
  - (Rodek Lajos és Diós Gábor megoldása)

```
ADAT SEGMENT PARA PUBLIC 'DATA'
   ElemSzam     EQU     5                   ; 5 elemű tömbök
   Vekt1        DB      6,-1,17,100,-8      ; tömb értékek
   Vekt2        DW      1000,1999,-32768,4,32767 ; eltérő típusok!
   Eredm        DW      ElemSzam DUP (?)    ; helyfoglalás eredménynek
ADAT ENDS

VEREM SEGMENT PARA STACK 'STACK'
   DB 1024  DUP (?)    ; helyfoglalás veremnek
VEREM ENDS
```

```
KOD SEGMENT PARA PUBLIC 'CODE'
   ASSUME CS:KOD, DS:ADAT, SS:VEREM

PRGSTART:
   mov ax,adat    ;ds betoltese
   mov ds,ax      ;ds beallitasa

   ;; ide jon maga a program

   MOV CX,ElemSzam ; Elemszám számláló reg.-be
   XOR BX,BX       ; BX nullázása
   MOV SI,BX       ; SI nullázása

Ciklus:
   MOV AL,[Vekt1+BX] ; AL-be Vekt1 értéke
   CBW               ; Előjelhelyes kiterjesztés
   ADD AX,[Vekt2+SI] ; Vekt2 elemének hozzáadása
   MOV [Eredm+SI],AX ; Összeg tárolása
   INC BX            ; BX növelése   (DB adat!)
   ADD SI,2          ; SI = SI + 2 ! (DW adat!)
   LOOP Ciklus       ; CX csökkentése, ugrás ha > 0

   ;; kilepes

   mov ah,4ch    ;kilepes
   mov al,00h    ;visszateresi kod
   int 21h       ;dos megszakitás
KOD ENDS
END PRGSTART
```

```
KOD SEGMENT PARA PUBLIC 'CODE'
   ASSUME CS:KOD, DS:ADAT, SS:VEREM

PRGSTART:
   mov ax,adat    ;ds betoltese
   mov ds,ax      ;ds beallitasa

   ;; ide jon maga a program

   MOV CX,ElemSzam
   XOR BX,BX
   MOV SI,BX

Ciklus:
   MOV AL,[Vekt1+BX]
   CBW
   ADD AX,[Vekt2+SI]
   MOV [Eredm+SI],AX
   INC BX
   LEA SI,[SI+2]      ; SI = SI + 2 ! (DW adat!)
   LOOP Ciklus

   ;; kilepes

   mov ah,4ch    ;kilepes
   mov al,00h    ;visszateresi kod
   int 21h       ;dos megszakitás
KOD ENDS
END PRGSTART
```

Alternatívák SI növelésére:

ADD SI,2

- OK, de FLAG regiszter értékek is állítódnak!

INC SI
INC SI

- Két utasítás 1 helyett

LEA SI,[SI+2]

- op2 offszet címének betöltése
- 1 utasítás, nincs FLAG állítás
- Összetettebb címzésnél is használható!

Példaprogram #2

- **Adatszegmensben található karakter tömb megfordítása verem segítségével**
  - A sztring végjelét mi definiáljuk
  - Végjelig olvasunk, az értékek verembe kerülnek
  - Ha elértük a végjelet, akkor kivesszük sorban az elemeket a veremből és a kezdeti memóriaterületre tesszük

```
ADAT SEGMENT PARA PUBLIC 'DATA'
   VEGJEL       DB      '$'
   SZO          DB      'Ez a szoveg$'
ADAT ENDS

VEREM SEGMENT PARA STACK 'STACK'
   DB 1024  DUP (?)   ; helyfoglalás veremnek
VEREM ENDS
```

```
KOD SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:KOD, DS:ADAT, SS:VEREM

PRGSTART:
    mov ax,adat     ;ds betoltese
    mov ds,ax       ;ds beallitasa
    ; megoldás
    MOV SI,OFFSET SZO   ; szó offszet címe SI regiszterbe
    MOV BX,0            ; BX nullázása
    XOR AH,AH           ; AH nullázása (szöveg bájtos, verembe AX tehető)
PAKOL:
    MOV AL,[SI+BX]      ; aktuális karakter AL-be töltése
    CMP AL,VEGJEL       ; összehasonlítás végjel karakterrel
    JE PAKOLVEGE        ; ha elértük, akkor lépjünk ki a ciklusból
    PUSH AX             ; karakter verembe kerül
    INC BX              ; következő karakterre pozícionálunk
    JMP PAKOL           ; ciklusmag újbóli feltétel nélküli végrehajtása
PAKOLVEGE:              ; ha elértük a végjelet
    CMP BX,0            ; ha csak végjel volt
    JE KILEP            ; kilépés
    MOV CX,BX           ; a sztring megszámolt hossza CX-be
    XOR BX,BX           ; BX nullázása: sztring elejére állás, SI változatlan
FORDIT:                  ; ciklusmag a fordításhoz
    POP AX              ; utolsó karakter kivétele a veremből
    MOV [SI+BX],AL      ; berakása az elején következő helyre
    INC BX              ; következő karakterpozícióra állás
    LOOP FORDIT         ; CX csökkentése, ha még nem nulla, akkor ciklusmag végrehajt.
KILEP:
    mov ah,4ch        ;kilepes
    mov al,00h        ;visszateresi kod
    int 21h           ;dos megszakitás
KOD ENDS
END PRGSTART
```

Példaprogram #2

- **Veszélyforrások!**
  - **Ha a sztring végéről lefelejtjük a végjelet**
    - A verembe pakolás folytatódik a verem- és kódszegmens területekről is
    - A verem túlcsordulhat
    - Kódterület felülírásra kerülhet
  - **Sztring hossza nagyobb mint a verem mérete**
    - Betelik a verem, a szegmens végétől felülírja az ottlévő tartalmat
- **Eredmény**
  - Lefagy(hat) a program, vagy akár a rendszer is!
  - Ennek oka, hogy a 8086 nem biztosítja a kódterületek írásvédelmét.

Assembly programozás
Logikai, léptető/forgató és processzorvezérlő utasítások

Utasítások

- **Logikai utasítások**
  - A FLAG regiszter értékei állítódnak!
  - `AND, OR, XOR`
  - `TEST`
  - `NOT`
- **Bitléptető, bitforgató utasítások**
  - `SAL, SHL, SAR, SHR`
  - `RCL, RCR, ROL, ROR`
- **Processzorvezérlő utasítások**
  - `CLC, STC, CMC`
  - `CLD, STD`
  - `CLI, STI`
  - `NOP`
  - `WAIT, LOCK, ESC, HLT`

Logikai utasítások

- **`AND, OR, XOR`**
  - Logikai ÉS, VAGY, KIZÁRÓ VAGY művelet az operandusok bitjei között
  - A „szokásos” igazságtáblák szerint
  - Eredmény **`op1`**-be

    ```
    AND op1,op2
    ```
- **`NOT`**
  - Operandus bitjeit negálja (egyes komplemens)

    ```
    NOT op
    ```
  - (Kettes komplemens: `NEG` utasítás (aritmetikai művelet))
- **`TEST`**
  - Logikai ÉS művelet az operandusok között, de csak a flag-ek állítódnak, az operandusok változatlanok maradnak!
  - Általában 1 bit tesztelésére használatos

    ```
    TEST op1,op2
    ```

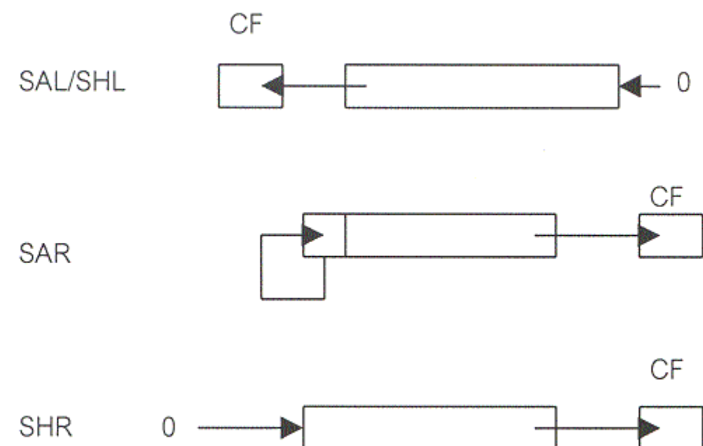
Bitléptető utasítások

- **Jelentőségük**
  - Bitek egyel „balra” léptetése 2-vel való szorzásnak
  - Bitek egyel „jobbra” léptetése 2-vel való osztásnak felel meg

- **Működésük (SHift, Shift Aritmetical)**

```
Shift.op.kód op1,szám ; léptetés szám darabszor
Shift.op.kód op1,CL   ; léptetés CL érték darabszor
```

  - **Balra (Left) léptetésnél** jobbról 0 lép be, a balról kilépő bit Carry-be kerül (SH**L**, SA**L**)
  - **Jobbra (Right) léptetésnél** a jobbról kilépő bit Carry-be kerül és
    - 0 lép be balról előjel nélküli esetben (SH**R**),
    - a legmagasabb bit pozíció (előjel!) értéke ismétlődik (SA**R**) az előjel megőrzésére
  - CL értéke nem változik!

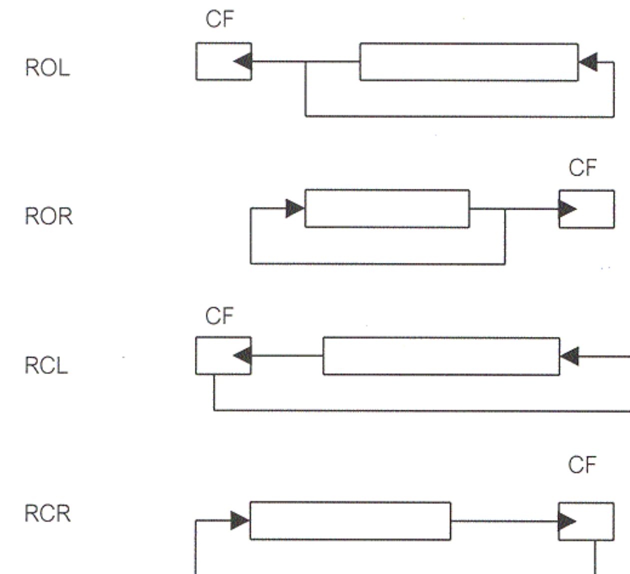


Bitforgató utasítások

- **Működésük (Rotate through Carry, ROtate)**

```
Rot.op.kód op1,szám ; forgatás szám darabszor
Rot.op.kód op1,CL   ; forgatás CL érték darabszor
```

- **Balra (Left) forgatásnál**
  - **RCL:** A balról kilépő bit Carry-be kerül, Carry eredeti tartalma lép be jobbról
  - **ROL:** A balról kilépő bit Carry-be kerül és jobbról is belép
- **Jobbra (Right) forgatásnál**
  - **RCR:** A jobbról kilépő bit Carry-be kerül, Carry eredeti tartalma lép be balról
  - **ROR:** A jobbról kilépő bit Carry-be kerül és balról is belép



Hatásuk a FLAG-ekre

mnemonic	assembler format	OF	DF	IF	TF	SF	ZF	AF	PF	CF
		flags								
	LOGICAL									
and	and destination,source	0	-	-	-	*	*	?	*	0
or	or destination,source	0	-	-	-	*	*	?	*	0
xor	xor destination,source	0	-	-	-	*	*	?	*	0
not	not destination	-	-	-	-	-	-	-	-	-
test	test destination,source	0	-	-	-	*	*	?	*	0
	Shift									
sal/shl	sal destination,count	*	-	-	-	*	*	?	*	*
sar	sar destination,count	*	-	-	-	*	*	?	*	*
shr	shr destination,count	*	-	-	-	0	*	?	*	*
	Rotate									
rol	rol destination,count	*	-	-	-	-	-	-	-	*
ror	ror destination,count	*	-	-	-	-	-	-	-	*
rcl	rcl destination,count	*	-	-	-	-	-	-	-	*
rcr	rcr destination,count	*	-	-	-	-	-	-	-	*

Note! * means changed, – means unchanged, and ? means undefined

Processzorvezérlő utasítások

- **Átvitel (Carry) flag állítása**
  - `STC, CLC, CMC`
    - Carry értékének 1-re, 0-ra, ellentettre állítása
    - Carry aritmetikai műveletekben használatos
    - Jelezhet hibát is egy eljárásból visszatérve (pl. BIOS, DOS eljárások rendszerint C=1 értékkel jelzik a hibát)
- **Megszakítás (Interrupt) flag állítása**
  - `STI, CLI`
    - Megszakítás flag 1-re, 0-ra állítása, vagyis engedélyezi/letiltja a hardver megszakítások fogadását

Processzorvezérlő utasítások

- **Egyéb utasítások**
  - **NOP**
    - Üres utasítás, nem csinál semmit (No OPeration)
  - **WAIT**
    - Várakozás pl. a társprocesszor eredményére
    - Amíg a processzor TEST lába 1 értékű, addig áll a futás
  - **LOCK** (utasítások előtti prefixum!)
    - Zárolja a rendszersínt az utasítás végrehajtása alatt
    - Többprocesszoros környezetben használatos a művelet megszakításának tiltására
  - **ESC op.kód,mem**
    - Más processzoroknak (pl. matematikai társprocesszor) szóló utasítás adatait a sín adat- és címvonalaira továbbítja
  - **HLT**
    - Futás felfüggesztése pl. megszakítás várására
    - Folytatás: megszakítás, hardver reset vagy DMA művelet hatására

Assembly programozás
Sztring műveletek

Sztring műveletek

- **Hasznuk**
  - Memóriablokkokkal (sztringekkel) végzett műveletek egyszerűsítésére
- **Utasítások**
  - MOVSB, MOVSW, MOVS (adatmozgatás)
  - CMPSB, CMPSW, CMPS (összehasonlítás)
  - LODSB, LODSW, LODS (betöltés)
  - STOSB, STOSW, STOS (tárolás)
  - SCASB, SCASW, SCAS (keresés)
  - REP, REPE, REPZ, REPNE, REPNZ prefixumok (ismétlés)

Sztring műveletek

- **Működésük**
  - Bájtos és szavas adatokra
  - **Forrás** adat: **`DS:SI`**
    - DS helyett más szegmens regiszter is használható
  - **Cél** adat: **`ES:DI`**
    - **`ES`** rögzített!
  - **Irány**
    - **`D`** flag tartalma alapján a memóriacím változik
      - Ha D = 0, akkor növelés (**`CLD`** utasítás)
      - Ha D = 1, akkor csökkenés (**`STD`** utasítás)
    - Maximum 64 KB adat másolható!
  - **Egyszerűsített ciklusszervezés**
    - **`REP`** prefixekkel
    - Művelet ismétlődik, **`CX`** és **`Z`** flag értéke alapján

Adatmásolás

- **MOVSB, MOVSW, MOVS**
  - Move String Byte/Word
  - A **DS:SI** által címzett bájt/szó átkerül az **ES:DI** címre
  - **SI** és **DI** értéke a **D** flag szerint változik
  - **MOVS** esetén
    - Két operandus is kell!
    - Az operandusok típusa dönti el, melyik utasítást kell használni
    - Így definiálható felül a forrás szegmens regiszter
    - Használata nem célszerű!

Adatmásolás

```
SOURCE DB   30   DUP (’  ’)
DEST   DB   40   DUP (’  ’)

       MOV DI,OFFSET DEST
       MOV SI,OFFSET SOURCE
       MOV CX,30
COPY_LOOP:
       MOVSB   ; MOVS  DEST,SOURCE is jó
       DEC CX
       JNZ COPY_LOOP
```

Adatmásolás

```
SOURCE DB   30   DUP (’  ’)
DEST   DB   40   DUP (’  ’)

       MOV DI,OFFSET DEST
       MOV SI,OFFSET SOURCE
       MOV CX,30
  REP  MOVSB
       ; ciklus egyszerűsítése
       ; REP prefixszel
```

Adatösszehasonlítás

- **CMPSB, CMPSW, CMPS**
  - Compare String Byte/Word
  - **DS:SI** által címzett bájt/szó összehasonlítása **ES:DI** cím tartalmával
  - FLAG regiszter bitjei a **CMP** utasításnak megfelelően állnak be
  - **SI** és **DI** értéke a **D** flag szerint változik
  - **CMPS**
    - Működése analóg a **MOVS** utasításéval

Adat betöltése és tárolása

- **LODSB, LODSW, LODS**
  - Load String Byte/Word
  - **DS:SI** cím tartalma **AL**-be (**AX**-be) kerül
  - Csak **SI** regiszter változik, **DI** nem
- **STOSB, STOSW, STOS**
  - Store String Byte/Word
  - **AL** (**AX**) tartalma **ES:DI** címre kerül
  - Csak **DI** regiszter változik, **SI** nem

Adat keresése

- **SCASB, SCASW, SCAS**
  - SCAn String Byte/Word
  - **AL** (**AX**) tartalmát összehasonlítja **ES:DI** cím tartalmával
  - **DI** regiszter változik, **SI** nem
  - FLAG bitjei a **CMP** utasításnak megfelelően állítódnak be

Ciklikus művelet végrehajtás

- **`REP`**
  - REPeat
  - Nem önálló utasítás!
  - Sztring művelet előtt helyezhető el
- **Működése**
  - Ismétlés számát CX-be töltjük
  - REP sztringművelet
    - Művelet végrehajtása, SI és/vagy DI léptetése
    - CX csökkentése
    - Ha CX nem 0, akkor ismétlés
  - **`CMPS`** és **`SCAS`** műveletek előtt alkalmazva
    - **`REPZ`**-vel (és **`REPE`**-vel) egyenértékű a hatása!
  - **`LODS`**-vel használni nincs értelme

Ciklikus művelet végrehajtás

- **`REPZ, REPE`**
  - REPeat while Zero/Equal
  - **`CX`** regiszter mellett **`Z`** flag értéke is számít
    - Ismétlés megszakad, ha **`CX`** 0 vagy **`Z`** értéke 0
    - **`Z`** értékét **`CMPS`** és **`SCAS`** utasítások állítják, **`MOVS`**, **`STOS`** utasítások nem
- **`REPNZ, REPNE`**
  - Mint fent, csak itt **`Z`** = 1 érték esetén szakad meg az ismétlés

Ciklikus művelet végrehajtás

- A **`REP`** műveletek lehetővé teszik memóriablokkok
  - Új helyre másolását (**`MOVS`**)
  - Adott értékkel feltöltését (**`STOS`**)
  - Bájt vagy szó érték előfordulásának első/utolsó helyét (**`SCAS`**)
  - Tartalmának összehasonlítását (**`CMPS`**)
- Egyetlen utasítás végrehajtásával

Példa

- **Feladat**
  - Memóriablokkok összehasonlítása
  - Az első eltérő forráselem **`AL`**-be, a célelem **`AH`**-ba kerüljön
  - **`AL`** és **`AH`** legyen 0, ha egyeznek

```
            ; CX, SI és DI előkészítése!

    REPZ    CMPSB
            JZ EGYEZO         ; megegyeznek
            MOV AL,[ SI ]     ; ha nem, AL és AH
            MOV AH,[ DI ]     ; beállítása
            JMP KILEPES
EGYEZO:
            XOR AX,AX ; AL és AH is 0
KILEPES:
            ; itt folytatódik a program
```

Assembly programozás

Paraméterátadás eljáráshíváskor
Skalár szorzat példa
Rekurzív, re-entráns eljárások

Paraméterátadás eljárásnak

- **Technika**
  - **Magasszintű nyelvek**
    - Formális paraméterlista
  - **Assembly**
    - Átadás módja nincs szabályozva, a programozónak kell gondoskodnia róla
    - Magasszintű nyelvekkel kombinált Assembly program esetén a másik nyelv hívási mechanizmusát kell használni
- **Történhet**
  - **Regiszter(ek)en keresztül**
    - Változó értékek
    - Memóriaterületek címei
  - **Vermen keresztül**
    - Változók értékei, címek a verembe kerülnek
    - Bázis-relatív címzéssel elérhetők az eljárásban

Paraméterátadás eljárásnak

- **Példa: Skalár szorzat**
  - Egyforma elemszámú tömbök, az eredmény a tömbelemek szorzatösszege
  - Eredmény kiíratása hexa alakban

A	x1	x2	x3	x4	x5
B	y1	y2	y3	y4	y5
Szorzat =	x1*y1 +	x2*y2 +	x3*y3 +	x4*y4 +	x5*y5

Paraméterátadás eljárásnak

- **Skalár szorzat**
  - **Megoldások**
    - Eljárás nélkül, *A* és *B* vektorok szorzása
    - Eljárással, *A* és *B* vektorok szorzása
    - Eljárással, vektorok címe `SI`, `DI` regiszterekben
    - Eljárással, paraméterek a vermen keresztül

```
; Két vektor skalár szorzata. 1. változat
; Eljárás nélkül, A és B szorzása

data        segment     para public 'data'
N           db          3               ; vektorok hossza
A           db          1, 2, 3         ; első vektor
B           db          3, 2, 1         ; második vektor
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
; ==========================================
```

```
code        segment     para public 'code'
     assume      cs:code, ds:data, ss:stack, es:nothing

skalar      proc        far
     ; előkészítés
     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 rész              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 rész               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 rész              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 rész            segédeljárások

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
end   skalar                     ; modul vége,
                                 ; a program kezdő címe:
   skalar
```

Egyszerűsítési lehetőség

```
; 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
```

Regiszterek mentése

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

```
; A
        PUSH   BX           ; mentés
        PUSH   CX
        PUSH   DX
és visszaállítása:
        POP    DX           ; visszaállítás
        POP    CX
        POP    BX
; C
```

```
; Két vektor skalár szorzata. 2. változat
; Eljáráshívással, A és B szorzása

; A rész              skalár szorzat számítása
        CALL   SKAL           ; ELJÁRÁS HÍVÁS
                              ; eredmény az AX regiszterben

; C rész              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
```

```
SKAL   PROC          ; Közeli (near) eljárás
; Az A és B program részek:
       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,
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              ; Visszaállítások
        POP     CX
        POP     BX

        RET                     ; Visszatérés a hívó programhoz

SKAL            ENDP            ; Eljárás vége

; D rész              segédeljárások
        ; változatlanok
```

Paraméterek regiszterekben

- **Probléma**
  - Csak *A* és *B* szorzatát tudja kiszámítani!

- **Megoldás**
  - Az eljárás paraméterként kapja meg a két vektor memóriacímét, így tetszőleges vektorok szorzatát számítani tudja
  - A paramétereket két regiszterben (**`SI`**,**`DI`**) adjuk át

```
; Két vektor skalár szorzata. 3. változat
; Eljáráshívással, paraméterként kapott vektor címekkel

; A rész            skalár szorzat számítása
      MOV   CL,N          ; Paraméter beállítások
      XOR   CH,CH         ; CX = N, érték
      MOV   SI,OFFSET A   ; SI ⇐ A offszet címe
      MOV   DI,OFFSET B   ; DI ⇐ B offszet címe
      call  skal          ; eljárás hívás
                          ; eredmény az ax regiszterben

; C rész            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
```

```
SKAL   PROC          ; Közeli (near) eljárás
; Az A és B program részek:
       push  bx            ; Mentések
       push  cx
       push  dx
       ;mov  cl,n          ; nem kellenek, mert
       ;xor  ch,ch         ; cx paraméterként jön!
       xor   dx,dx         ; az eredmény ideiglenes helye
       jcxz  kesz          ; ugrás a kesz címkére, ha n=0
       xor   bx,bx         ; bx ⇐ 0,
ism:   mov   al,[si+bx]          ; A-tól független!
       imul  byte ptr [di+bx]    ; byte ptr kell!
       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
       ; További részek változatlanok
```

Paraméterek a veremben

- **Probléma**
  - Regisztereken keresztül csak limitált számú paraméter adható át
- **Megoldás**
  - A paramétereket egy memóriaterületen helyezzük el és annak a címét adjuk át
  - A **paramétereket helyezzük el a veremben**, ahol a meghívott eljárás eléri

```
; Két vektor skalár szorzata. 4. változat
; Eljáráshívással, vermen keresztül átadott paraméterekkel

; A rész             skalár szorzat számítása
        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
        ; Verembe került eddig:
        ; N értéke, A címe, B címe (2*3 = 6 bájt)
        ; Verembe kerül még a visszatérési cím is (2 bájt)!
        call   skal          ; eljárás hívás
                             ; eredmény az ax regiszterben
        ADD    SP,6          ; paraméterek ürítése a veremből
                             ; 2*3 = 6 bájt
                             ; a címet a RET előzőleg kivette
; C rész             eredmények kiírása
        ; változatlan
```

```
SKAL  PROC          ; Közeli (near) eljárás
; Az A és B program részek:
      PUSH  BP            ; BP értékét mentenünk kell
      MOV   BP,SP         ; BP ⇐ SP,
                          ; a verem relatív címzéshez
      PUSH  SI            ; mentések
      PUSH  DI
      push  bx
      push  cx
      push  dx

; A verem tartalma jelenleg:
; N értéke, A címe, B címe          paraméterek
; visszatérési cím,
; BP, SI, DI, BX, CX, DX            mentett regiszterek
```

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 ⇐ 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			. . .

```
+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 ⇐ az egyik vektor címe
      MOV   DI,4[BP]      ; DI ⇐ a másik vektor címe
      MOV   CX,8[BP]      ; CX ⇐ a dimenzió értéke
      xor   dx,dx ; az eredmény ideiglenes helye
      jcxz  kesz          ; ugrás a kesz címkére, ha n=0
      xor   bx,bx ; bx ⇐ 0, indexezéshez
ism:  mov   al,[si+bx]; független A-tól
      imul  byte ptr [di+bx]
      add   dx,ax ; dx ⇐ részösszeg
      inc   bx            ; bx ⇐ bx+1, az index növelése
      loop  ism
kesz:       mov   ax,dx ; a skalár szorzat értéke ax-ben
```

```
        pop          dx              ; visszaállítások
        pop          cx
        pop          bx
        POP          DI
        POP          SI
        POP          BP
        ret                          ; visszatérés a hívó
   programhoz
skal    endp                         ; a skal eljárás vége

; D rész             segédeljárások
        ; Változatlanok
```

```
        ADD          SP,6   ; paraméterek ürítése a veremből
```

helyett más megoldás:

```
        RET          6      ; visszatérés a hívó programhoz
                            ; verem ürítéssel: . . . SP = SP +
   6
                            ; ha ismerjük az átadott
   paraméterek
                            ; számát
```

C konvenció

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.

f(x,y);

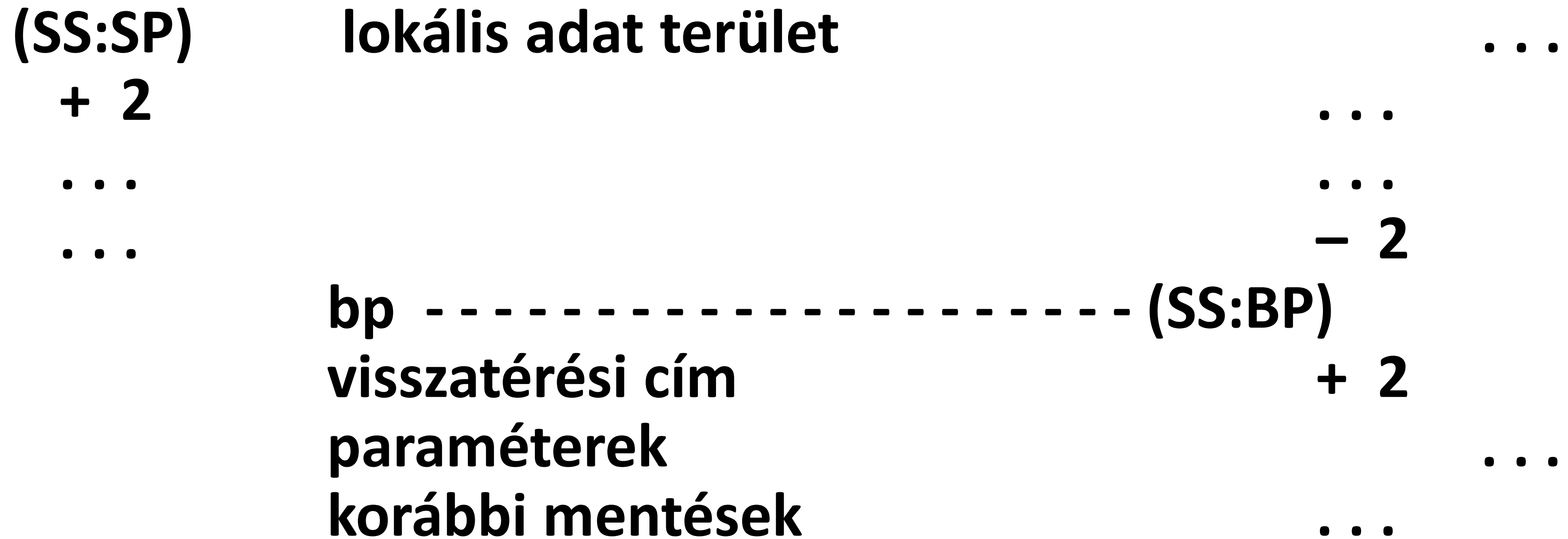
```
push  y
push  x
call  f
```

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 felesleges, akkor a munkaterületet célszerűen a **veremben** alakíthatjuk ki. A munkaterület lefoglalásának ajánlott módja:

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

Lokális adat terület (**NEAR** eljárás esetén)



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

A munkaterület felszabadítása visszatéréskor a

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

utasításokkal történhet.

Rekurzív és re-entráns eljárások

- **Rekurzív**
  - Ha önmagát hívja közvetlenül, vagy más eljárásokon keresztül
- **Re-entráns**
  - Ha többszöri belépést tesz lehetővé, ami azt jelenti, hogy az eljárás még nem fejeződött be, amikor újra hívható
  - A rekurzív is egyfajta re-entráns

Rekurzív és re-entráns eljárások

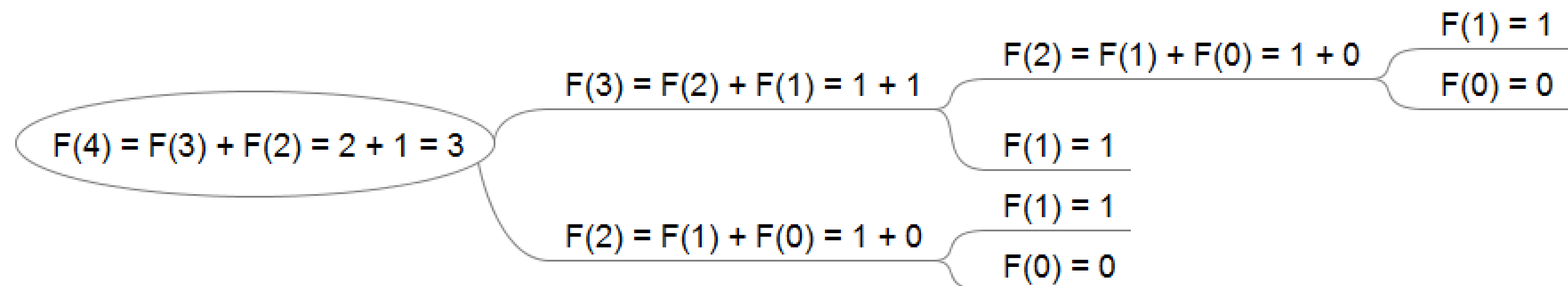
- **Re-entráns**
  - A rekurzív eljárással szemben a különbség az, hogy a rekurzív eljárásban „programozott”, hogy mikor történik az eljárás újra hívása, re-entráns eljárás esetén az újra hívás ideje „kiszámíthatatlan”
    - Pl.: Egy eljárást meghívhat a programunk és egy általunk kezelt megszakítási rutin is.
    - A programból meghívott eljárás futása bármikor megszakítódhat és a megszakítási rutin meghívhatja újra.
    - Emiatt az eljárás minden változóját minden híváskor lokálisan célszerű létrehozni, a munkaterületek keveredésének elkerülése érdekében!

Rekurzív és re-entráns eljárások

- **Re-entráns**
  - Munkaterületek összekeveredése ellen
    - Ne legyen önmódosító a program
    - Minden regiszter értéket menteni és visszaállítani kell
    - A paramétereket a vermen keresztül célszerű fogadni és visszaadni
    - A lokális változókat a vermen célszerű létrehozni
    - Processzusok között megosztott kód esetén a processzusok egymástól független, önálló verem szegmenssel rendelkeznek

Példa rekurzióra

- **Fibonacci sorozat**
  - **Rekurzív definíció**
    - `F(0) = 0`
    - `F(1) = 1`
    - `F(n) = F(n-1) + F(n-2)`
  - **Első néhány eleme**
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368



```
VEREM SEGMENT PARA STACK 'STACK'
      DB 1024  DUP (?)
VEREM ENDS

KOD SEGMENT PARA PUBLIC 'CODE'
      ASSUME CS:KOD, DS:ADAT, SS:VEREM

PRGSTART:
      mov ax,adat   ;ds betoltese
      mov ds,ax     ;ds beallitasa

      ; Bemenet AX-ben, eredmény DX-ben
      MOV AX,4
      CALL FIBO

V0:   mov ah,4ch        ;kilepes
      mov al,00h        ;visszateresi kod
      int 21h           ;dos megszakitás
```

```
FIBO PROC NEAR
        CMP AX,0
        JNE TOVABB1
        MOV DX,0            ; F(0) = 0
        JMP KILEP
TOVABB1:
        CMP AX,1
        JNE TOVABB2
        MOV DX,1            ; F(1) = 1
        JMP KILEP
TOVABB2:
        PUSH AX             ; n értékének mentése
        SUB AX,1            ; n-1
        CALL FIBO           ; F(n-1) számítása
V1:     POP AX              ; n elővétele veremből
        PUSH DX             ; F(n-1) részeredmény verembe
        SUB AX,2            ; n-2
        CALL FIBO           ; F(n-2) számítása
V2:     POP AX              ; F(n-1) részeredmény elővétele
        ADD DX,AX           ; F(n) = F(n-1) + F(n-2)
KILEP:
        RET                 ; Eredmény DX regiszterben
FIBO ENDP
KOD ENDS
END PRGSTART
```

Fibonacci sorozat

- **Feladat**
  - Verem tartalmának követése különböző `F(n)` értékek számításakor