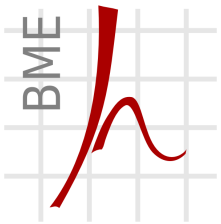
Programok működése dióhéjban

Intel x68



Híradástechnikai Tanszék

Izsó Tamás

2015. október 30.

Section 1

Fordítás

authenticate.c

```
1  // cl /GS- authenticate.c
2  #include <stdio.h>
3
4  int verify_password(char* username, char* password) {
5     /* Csak a próba kedvéért! */
6     if( strcmp(username,"tomi") == 0 &&
7         strcmp(password,"csillag") == 0 ) return 1;
8     return 0;
9  }
10
11 void log(const char* fmt, const char* msg) {
12    fprintf(stderr,fmt, msg );
13 }
```

authenticate.c

```
15 int authenticate(char *username, char *password) {
16    int authenticated;
17    char buffer[1024];
18    authenticated = verify_password(username, password);
19    if(authenticated == 0)   {
20       sprintf(buffer,
21          "password is incorrect for user %s\n",
22          username);
23       log("%s", buffer);
24    }
25    return authenticated;
26 }
27
28 main(int argc, char* argv[] ) {
29    int res;
30    if( argc < 3 ) return 100;
31    res = authenticate( argv[1], argv[2] );
32    printf("Password : %s\n", res ? "OK" :"No!" );
33    return 0;
34 }
```

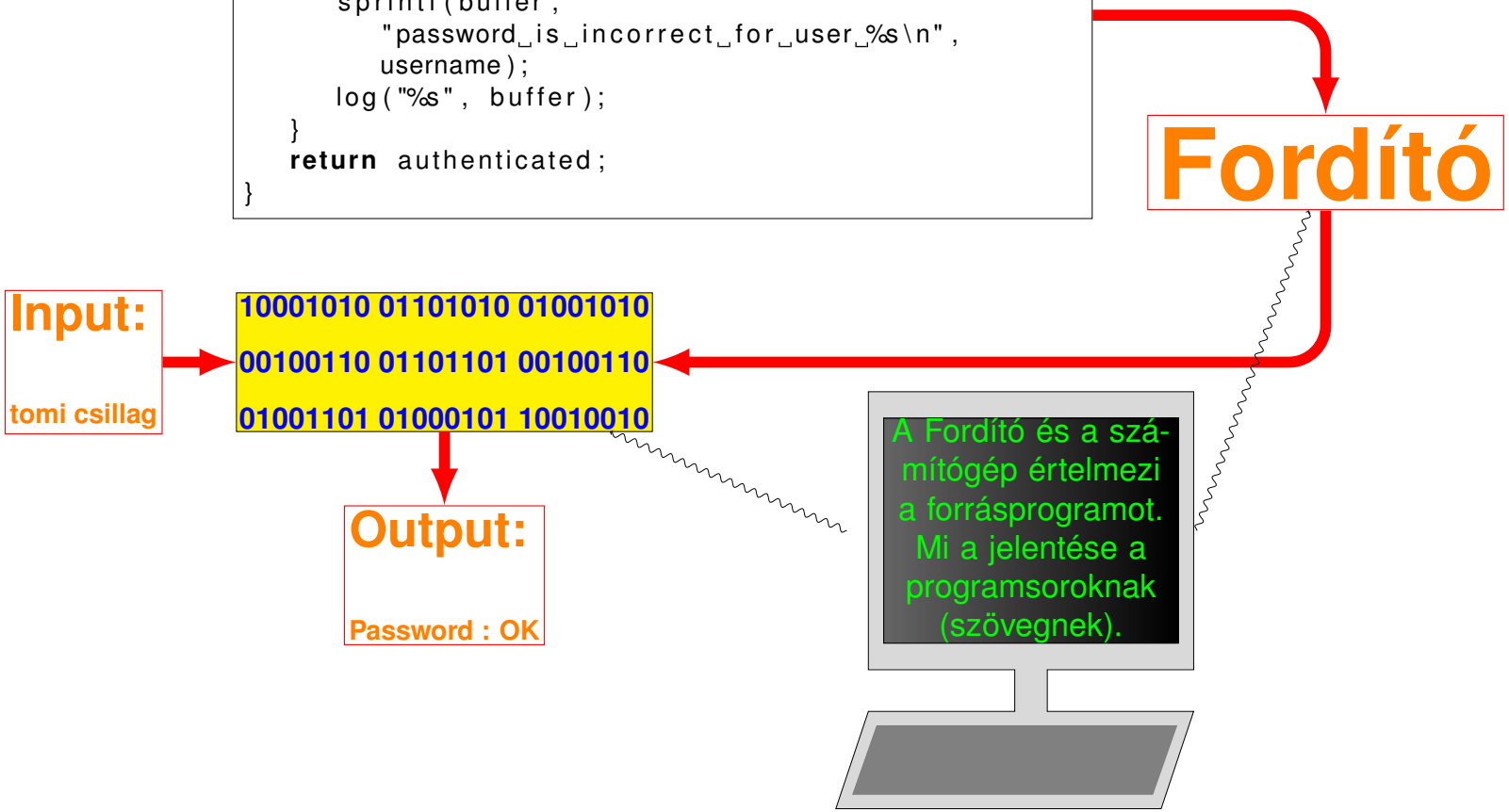
Kérdés

Az előző program biztonságos ?

Attól függ, hogy miként fut le a program.

Program értelmezése

```
int authenticate(char *username, char *password) {
    int authenticated;
    char buffer[1024];
    authenticated = verify_password(username, password);
    if(authenticated == 0)    {
        sprintf(buffer,
            "password is incorrect for user %s\n",
            username);
        log("%s", buffer);
    }
    return authenticated;
}
```



Bináris futtatható program

Támadás megtervezése

A támadó célja a számítógép felett átvenni a felügyeletet.

Megvalósítás:

1. A támadást végrehajtó kód bejuttatása a számítógépre. Kivitelezés:
   - A támadó input ablakba beírja a gépi kódú programot.
   - A támadó preparált film, kép, mp3 fájl segítségével juttatja be a kártékony programot.
   - stb.
2. A támadó eltéríti a program futását, és ráadja a vezérlést a bejuttatott kódra.
3. A kártékony program célja elérése érdekében használja a feltört számítógép erőforrásához.

Kulcskérdések

- Rendszeresen frissített operációs rendszer esetén a bejuttatott kód csak az operációs rendszerhíváson keresztül érheti el az erőforrásokat, például a fájlokat, hálózatot, stb.[1] Ehhez ismerni kell a DLL (.so) fájl formátumát.
- A támadónak ismerni kell a processzor működését és a gépi utasításokat. Ismerni kell, hogy hol vannak azok a pontok, ahol az utasítás számláló regiszter kapcsolatba lép a memóriával, vagy indirekten módon befolyásolható az értéke.

[1] Különben ismernie kellene még hivatalosan nem publikált úgynevezett *Zero-Day Vulnerability*-t.

Mit kell tudni a támadás kivitelezéséhez

- a processzor felépítését és működését
  - bájt-sorrend;
  - regiszterek;
  - utasítások;
- operációs rendszert
  - végrehajtható [2] fájl formátumát[3];
  - dinamikusan linkelt függvények hívási mechanizmusa;
  - rendszer szintű kivételkezelés (SEH);
  - fontosabb operációs rendszerhívások;
- fordító által generált kód működését
  - stack működése;
  - heap szerkezete;
  - függvényhívás megvalósítása;
  - virtuális függvények megvalósítása;
  - programozási nyelvben megvalósított kivételkezelés (exception handler);

[2] Windows esetén PE (Portable Executable), Linuxon ELF formátum

[3] A DLL/shared object is ilyen formátumú fájl.

Trojan.Cryptowall

„Nearly all men can stand adversity, but if you want to test a man's character, give him power."

— Abraham Lincoln

Terjedése:

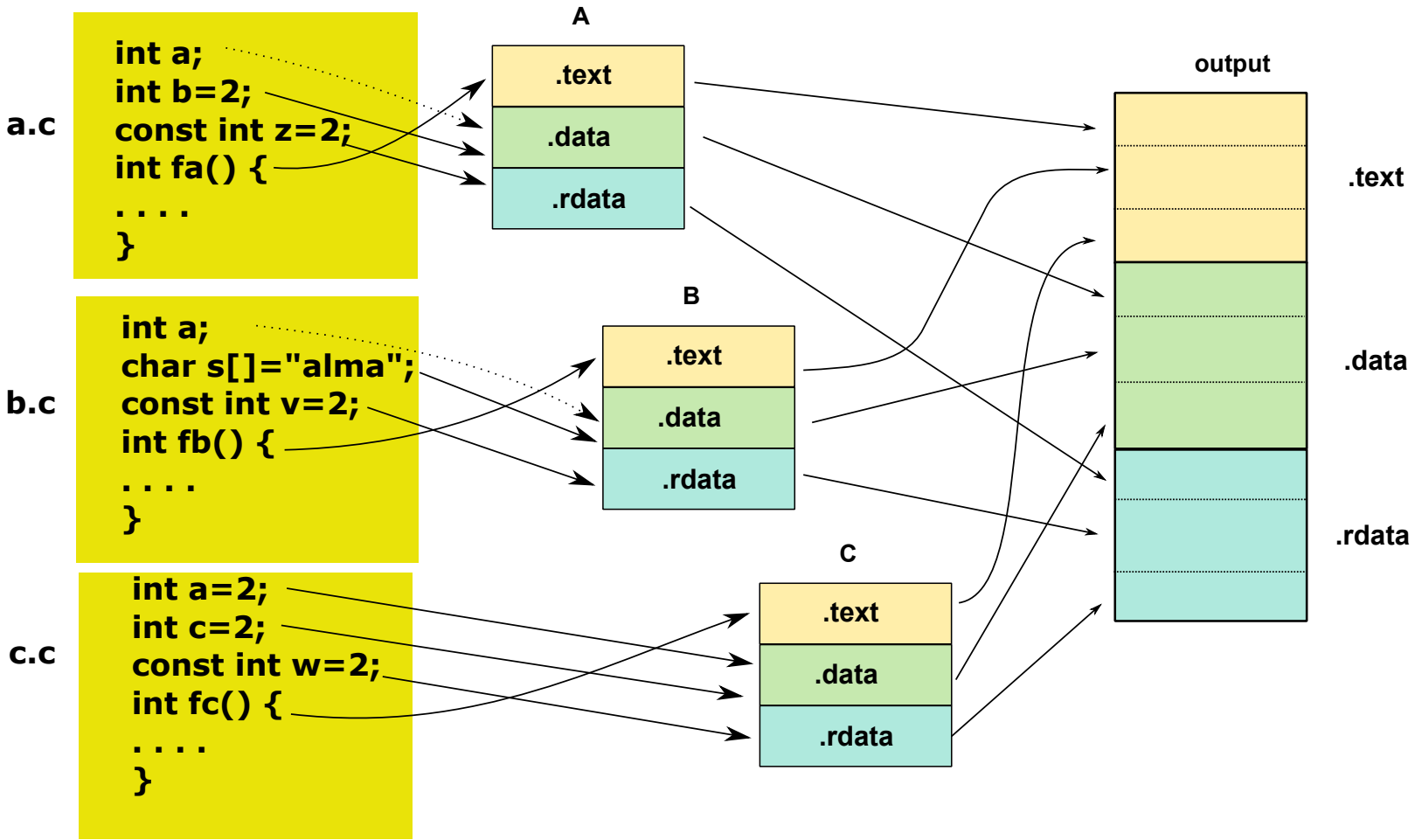
- levélben;
- Exploit Kits
  - Oracle Java SE Remote Java Runtime Environment Code Execution Vulnerability (CVE-2012-0507)
  - Adobe Flash Player Buffer Overflow Vulnerability (CVE-2014-0515)
  - Adobe Flash Player and AIR Unspecified Heap Based Buffer Overflow Vulnerability (CVE-2014-0556)
- Malware utján.

Hatás: Titkosítja a fájlokat 2048-bit-es RSA algoritmussal.

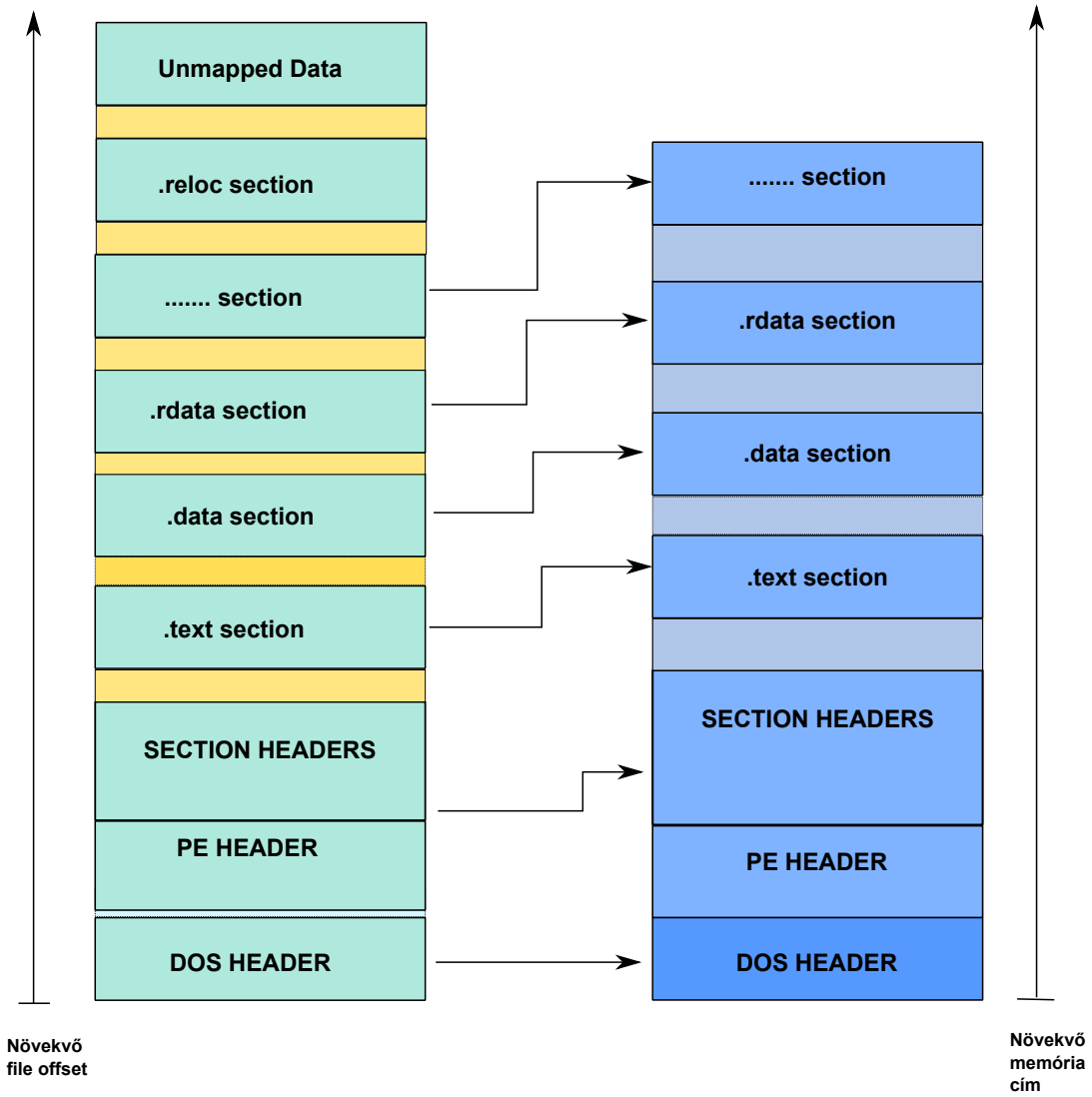
Section 2

PE/COFF

Program szegmensek linkelése



Exe file leképzése a memóriába



Portable Executable fájl

- DOS operációs rendszerrel kompatibilis, figyelmeztető üzenetet ad, ha nem futtatható DOS alatt a program. A PE fájl eleje megegyezik a MZ végrehajtható fájlformátummal.
- DLL hívást támogatja. Az exe és a DLL azonos szerkezetű. A DLL fájlokat sokszor más kiterjesztéssel használják, pl. OCX , CPL
- Alpha, MIPS és .NET MSIL végrehajtható fájlformátuma is egyben.
- 64 bites programok hasonló formában vannak tárolva, egyes helyeken a 32 bitet 64 bites adatok váltották fel. PE32+ elnevezést kapta.

Relative Virtual Address (RVA)

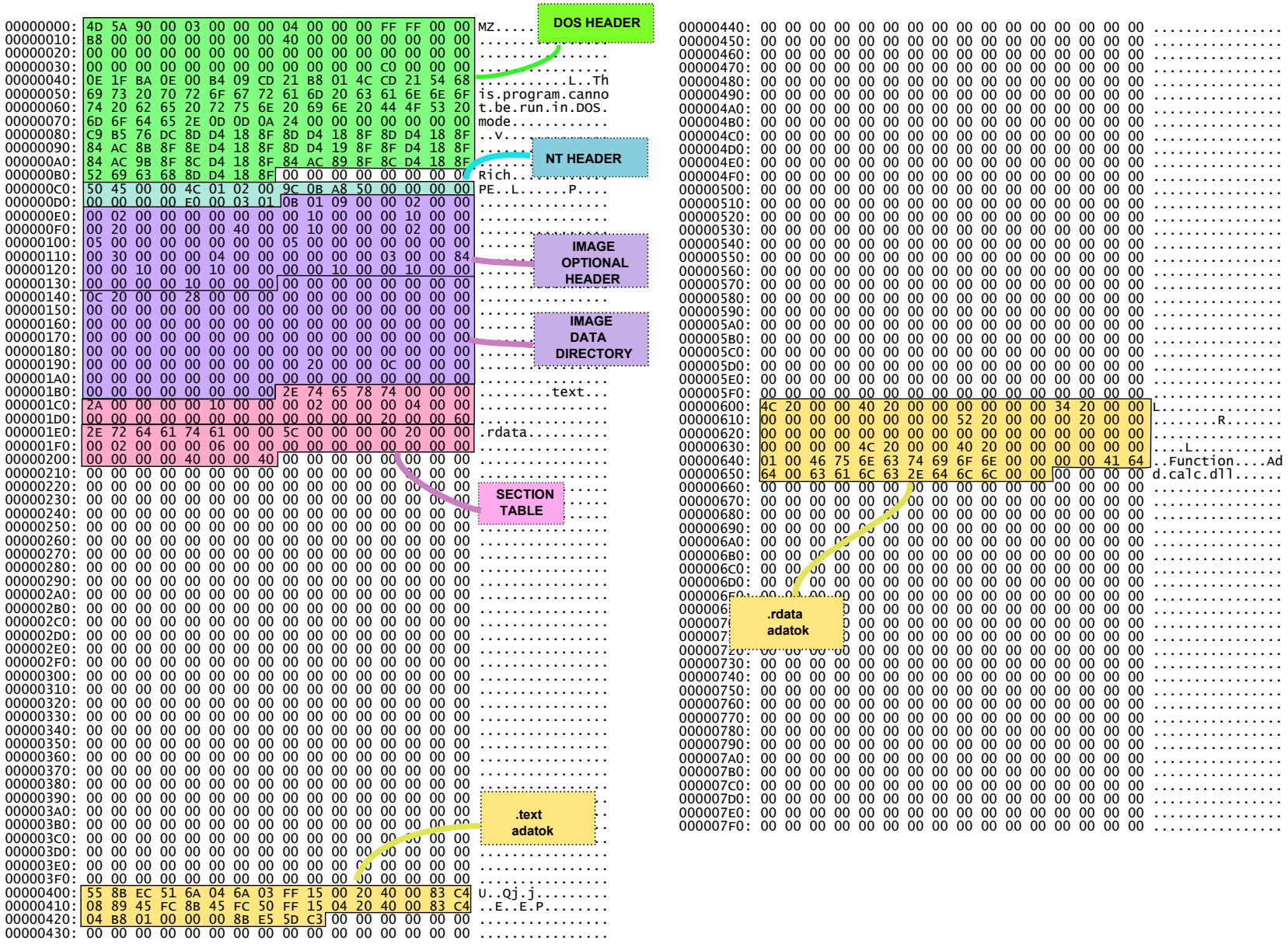
A futtatható fájlt a loader nem másolja be a memóriába, hanen az egyes szekciókat memory mapped file-ként kezeli. A fájlban tárolt szekciók 512-vel (0x200) osztható byte határon kezdődnek, míg a memóriában új lapra kerülnek, ahol egy lap 4Kbyte (vagy 8 Kbyte). Ezért a fájl adott tartalma a fájl elejétől más távolságra van, mint amikor ugyanezt az adatot a memóriába a betöltési ponthoz viszonyítjuk.

RVA kiszámítása

A memóráiban lévő címek a betőltés helyétől függetlenül vannak megadva. Ha egy cím a 0x4010DA címen szerepel és a program képe a 0x400000 címtől kezdve lett betöltve, akkor a realitív vírtuális cím (RVA):

$$\text{RVA} = 0x4010DA - 0x400000 = 0x10DA$$

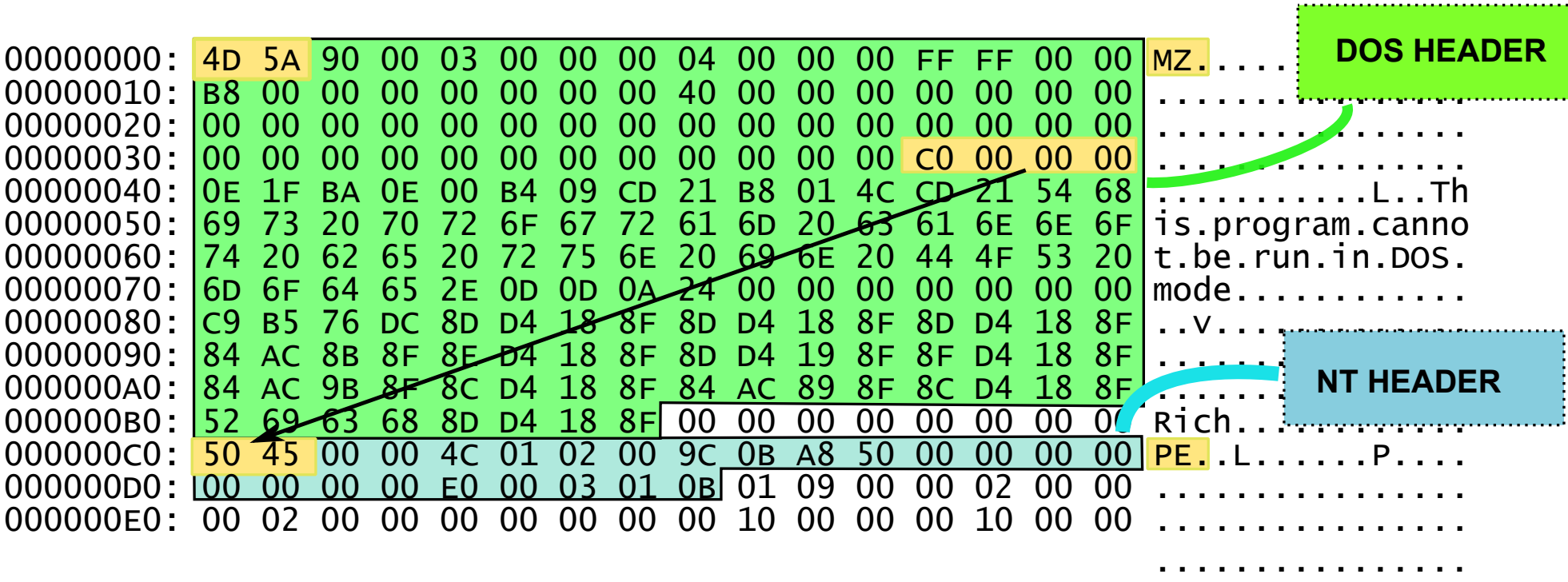
Exe file dumpja



DOS HEADER tartalma

- Két érdemi részt tartalmaz:
  1. e_magic értéke "MZ" (Mark Zbikowski)
  2. e_lfanew file offset, a PE header-re mutat.
- DOS_HEADER után kis DOS program következik.
- A PE rész közvetlenűl ez után, 8-cal osztható címre igazítva található.

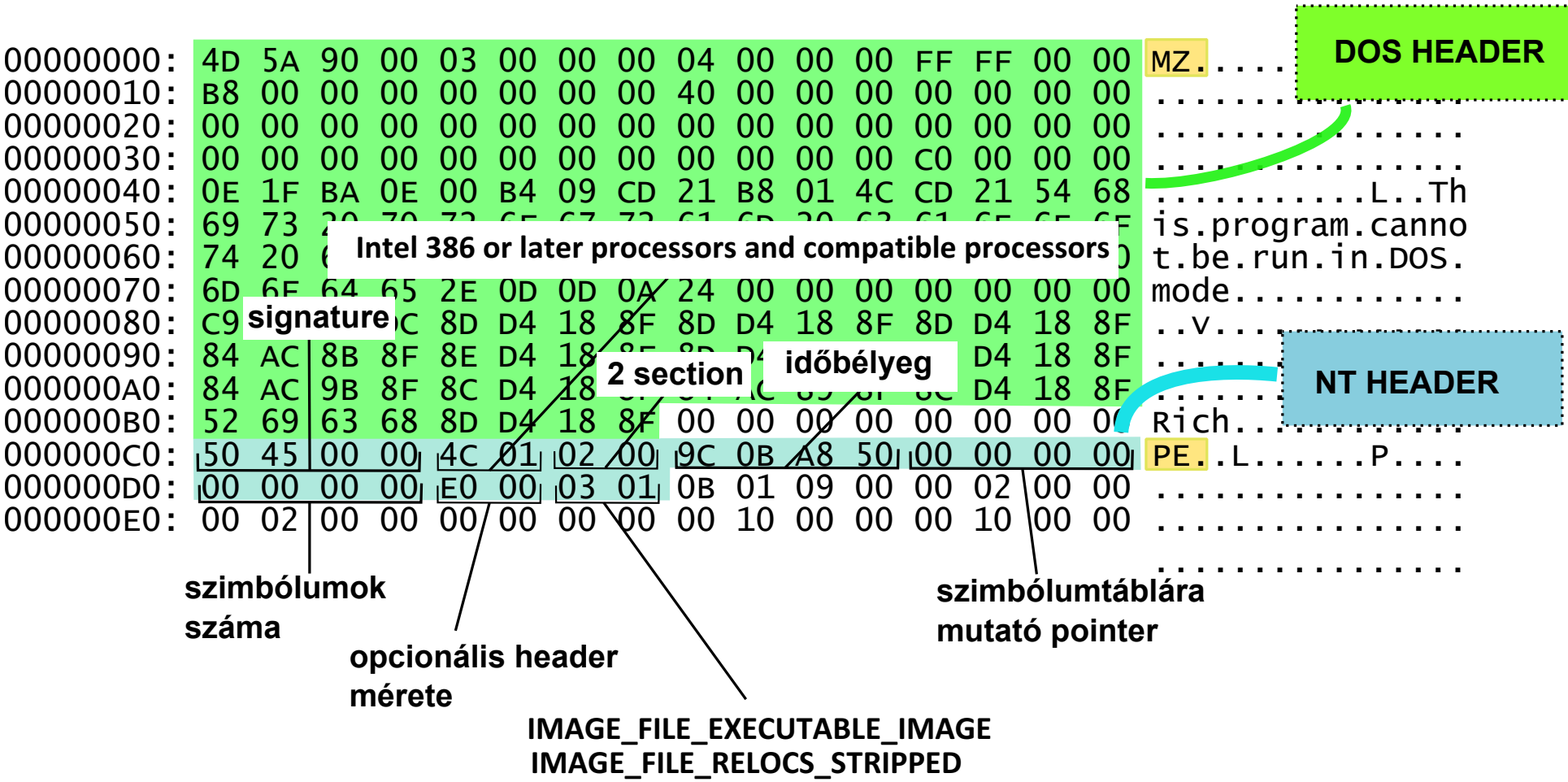
DOS header dumpja



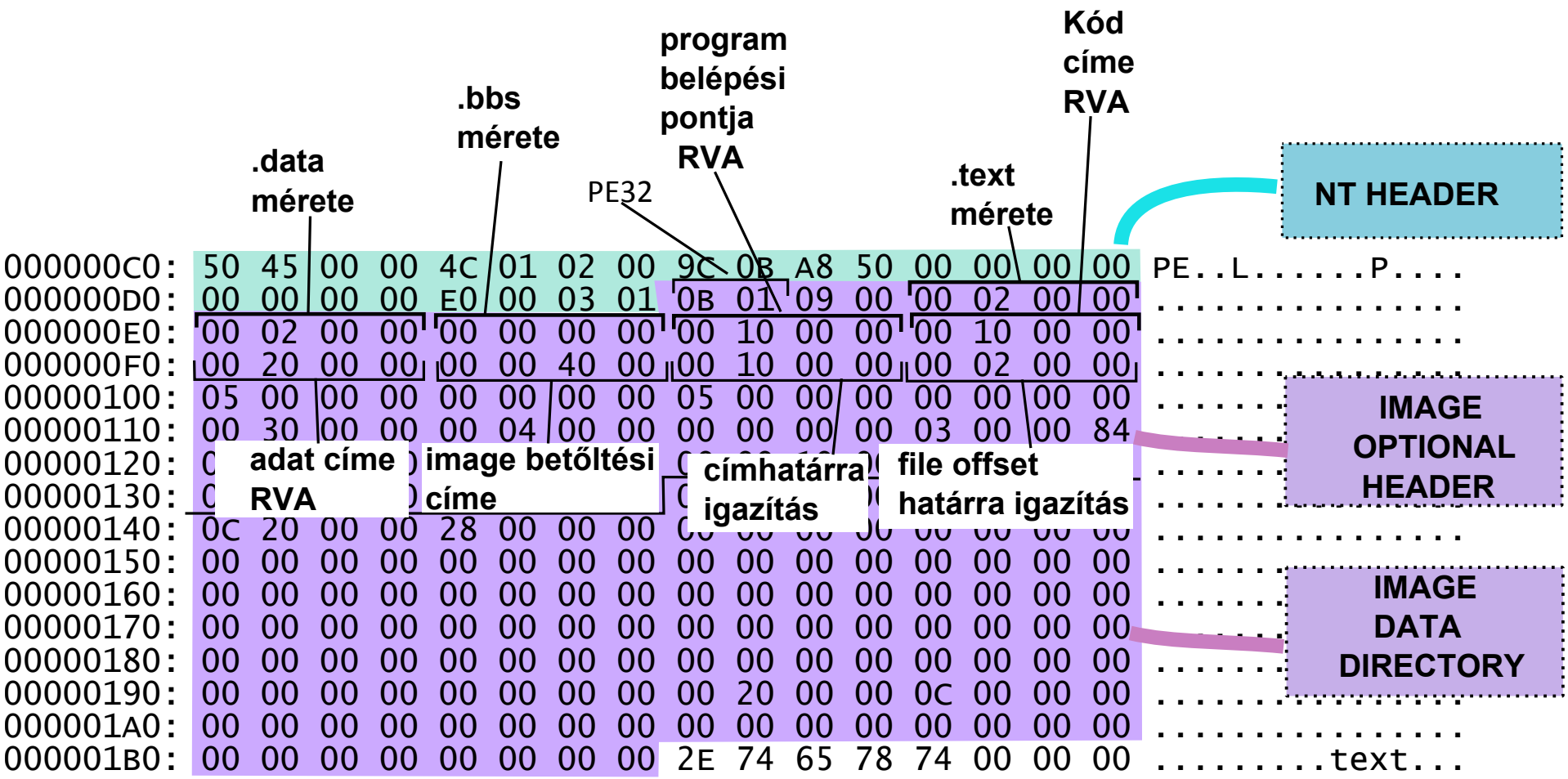
NT HEADER

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;    // "PE"
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

NT header dumpja



Optional header dumpja



SECTION TABLE

A section táblák a fordítás során keletkezett kód, adat, konstans adat stb. területek információit tartalmazza. Egy section adatai:

- neve;
- helye a file-ban (blokkhatáron kezdődnek);
- section helye a memóriában (laphatáron kezdődnek);
- memóriavédelmi beállítások (írható, olvasható, futtatható, stb.);
- méret;
- relokációs adatok száma és azok helye a file-ban;
- debug-os fordítás esetén kapcsolat a forrásszöveggel (sorok száma);

SECTION TABLE dumpja

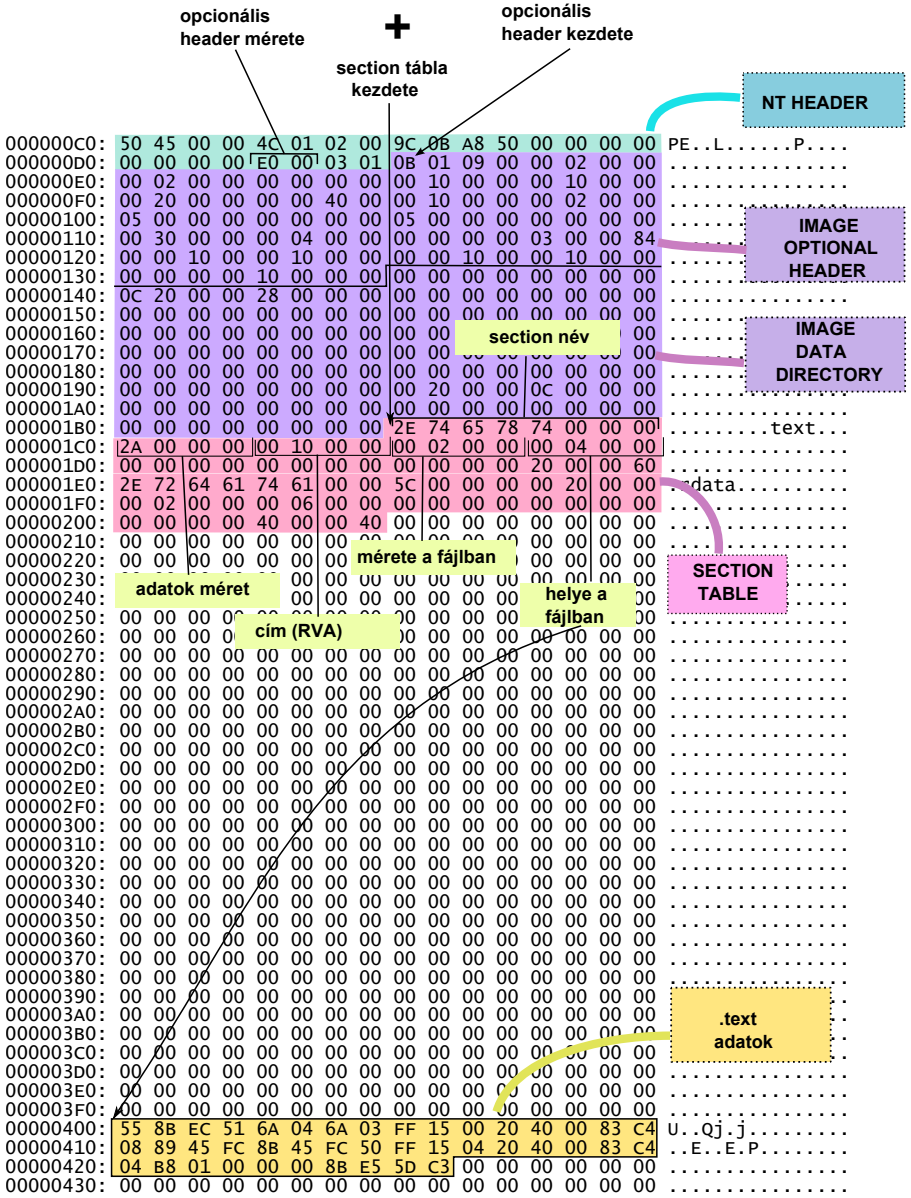


IMAGE DATA DIRECTORY

IMAGE DATA directory megadja a program futásánál speciális szerepet betöltő adatok. Ezeknek a helye valamelyik section-ban van.

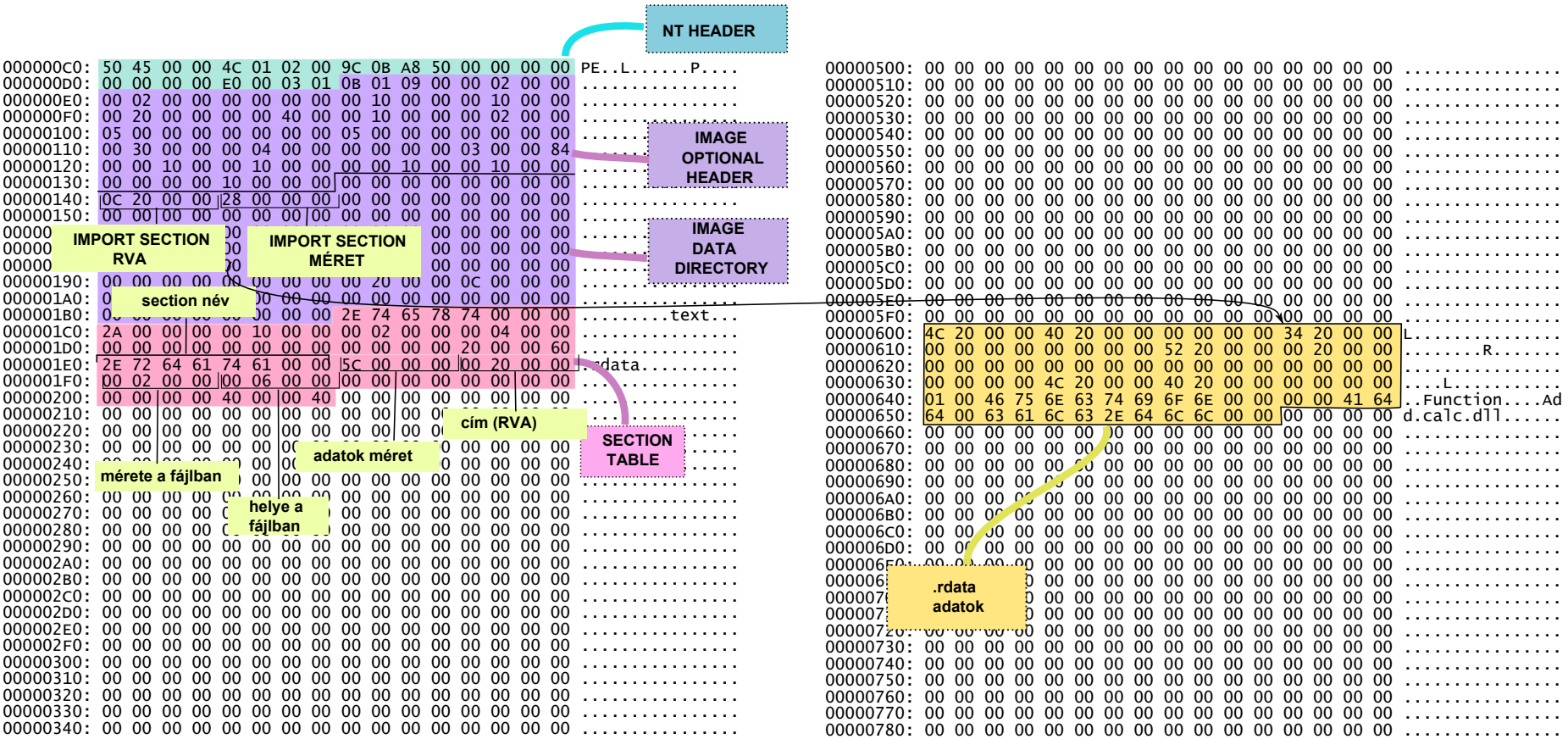
- 16 darab bejegyzés van, amiből 15-öt használnak.
- Egy bejegyzés az adat helyét és méretét tartalmazza.

IMAGE DATA 16 bejegyzése

Illusztráció!

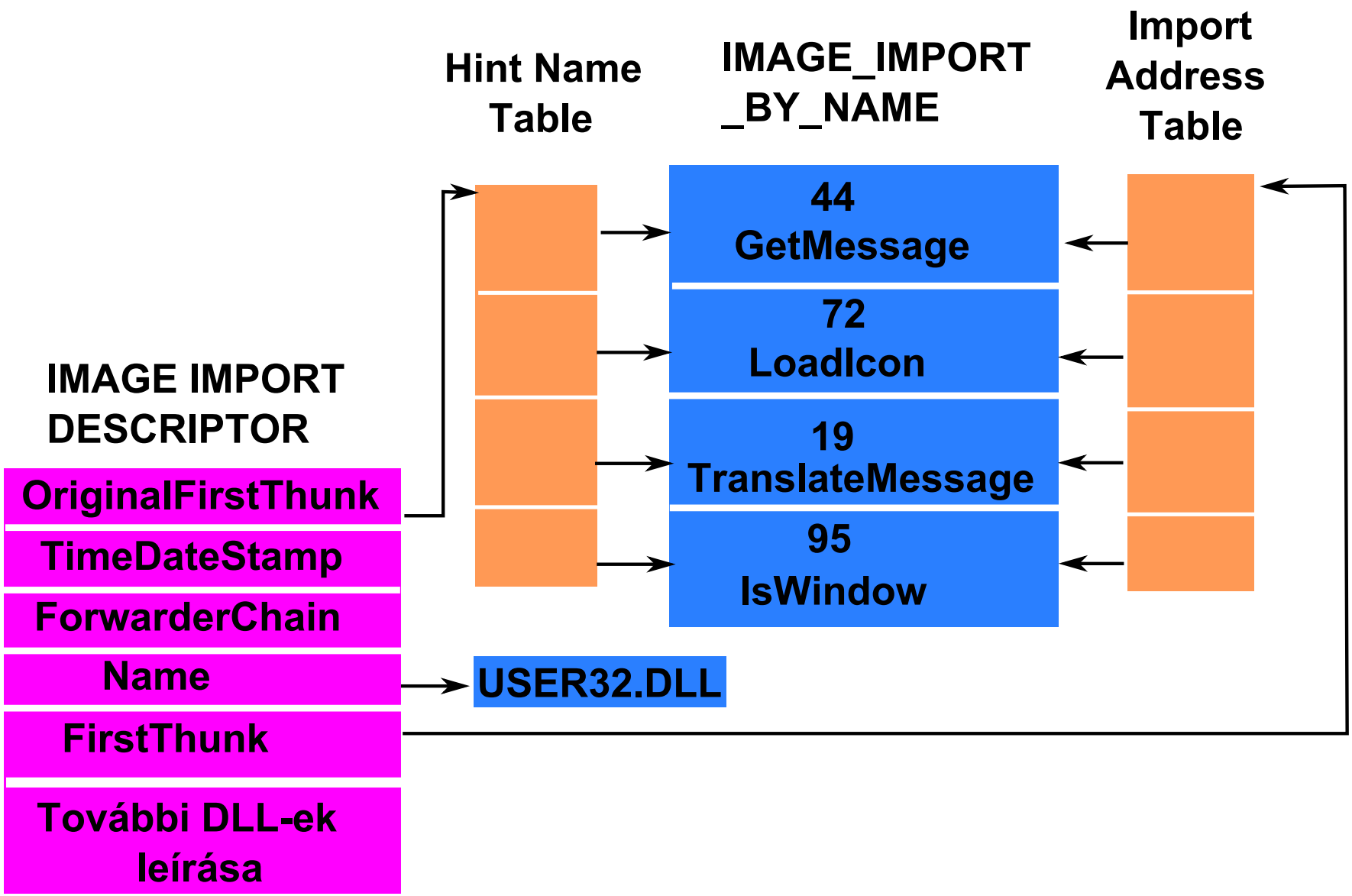
0	Export Table	dll .edata Section
1	Import Table	dll .idata
2	Resource Table	.rsrc Section
3	Exception Table	.pdata Section
4	Certificate Table	
5	Base Relocation Table	.reloc Section
6	Debug	.debug Section
7	Architecture Specific Data	reserved 0
8	Global Ptr	
9	TLS Directory	
10	Load Configuration Directory	
11	Bound Import	
12	IAT	import address table
13	Delay Load Import Descriptors	
14	CLR Runtime descriptor	
15	reserved	0

IMPORT DIRECTORY DATA dumpja

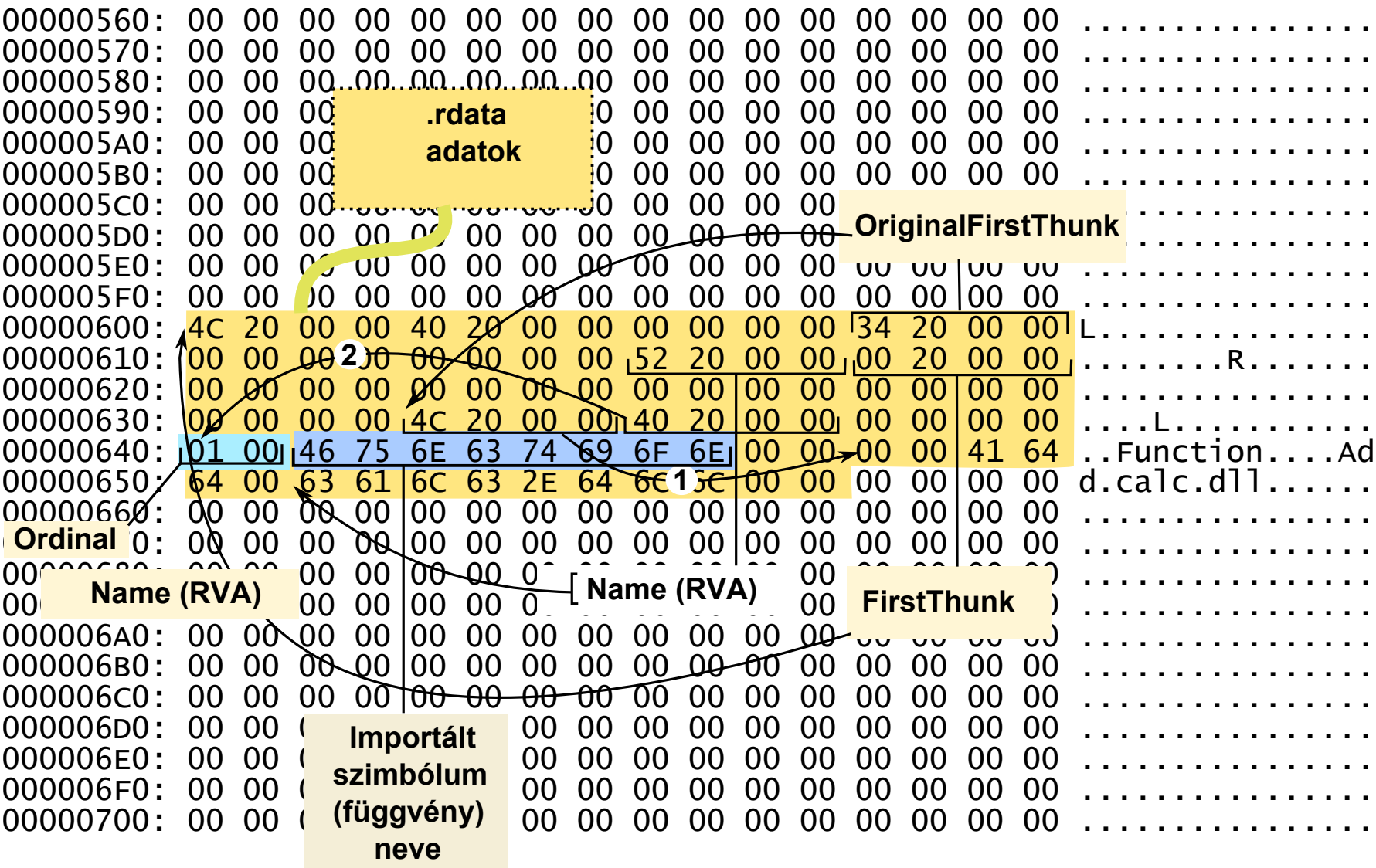


Látható, hogy a konstans adatszegmensben kapott helyet.

IMPORT TABLE



IMPORT TABLE



IMPORT TABLE – program indítása után a memóriában

```
00402000  00 10 00 10|30 10 00 10|00 00 00 00|34 20 00 00|  ....0.......4  ..
00402010  00 00 00 00|00 00 00 00|52 20 00 00|00 20 00 00|  ........R  ...  ..
00402020  00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|  ................
00402030  00 00 00 00|4C 20 00 00|40 20 00 00|00 00 00 00|  ....L  ...  ......
00402040  01 00 46 75|6E 63 74 69|6F 6E 00 00|00 00 41 64|  ..Function....Ad
00402050  64 00 63 61|6C 63 2E 64|6C 6C 00 00|00 00 00 00|  d.calc.dll......
```

DLL-ben lévő függvény hívása

Illusztráció!

```
00401000    55              PUSH EBP
00401001    8BEC            MOV EBP,ESP
00401003    51              PUSH ECX
00401004    6A 04           PUSH 4
00401006    6A 03           PUSH 3
00401008    FF15 00204000   CALL DWORD PTR DS:[<&calc.Add>]
0040100E    83C4 08         ADD ESP,8
00401011    8945 FC         MOV DWORD PTR SS:[LOCAL.1],EAX
00401014    8B45 FC         MOV EAX,DWORD PTR SS:[LOCAL.1]
00401017    50              PUSH EAX
00401018    FF15 04204000   CALL DWORD PTR DS:[<&calc.Function>]
0040101E    83C4 04         ADD ESP,4
00401021    B8 01000000     MOV EAX,1
00401026    8BE5            MOV ESP,EBP
00401028    5D              POP EBP
00401029    C3              RETN
```

DLL tartalma

Illusztráció!

```
10001000    55              PUSH EBP
10001001    8BEC            MOV EBP,ESP
10001003    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001006    0345 0C         ADD EAX,DWORD PTR SS:[EBP+0C]
10001009    5D              POP EBP
1000100A    C3              RETN

10001010    55              PUSH EBP
10001011    8BEC            MOV EBP,ESP
10001013    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001016    2B45 0C         SUB EAX,DWORD PTR SS:[EBP+0C]
10001019    5D              POP EBP
1000101A    C3              RETN

10001020    55              PUSH EBP
10001021    8BEC            MOV EBP,ESP
10001023    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001026    0FAF45 0C       IMUL EAX,DWORD PTR SS:[EBP+0C]
1000102A    5D              POP EBP
1000102B    C3              RETN

10001030    55              PUSH EBP
10001031    8BEC            MOV EBP,ESP
10001033    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001036    50              PUSH EAX
10001037    68 00C00010     PUSH OFFSET 1000C000        ; ASCII "Printf from DLL %d "
1000103C    E8 7F000000     CALL 100010C0
10001041    83C4 08         ADD ESP,8
10001044    5D              POP EBP
10001045    C3              RETN
```

Importált függvények címének feloldása

- A program a dll-ben lévő **Add**() függvényt a 00401008 címen a **CALL** [00402000] utasítással hívja meg. Az indirekten hívott függvény címét az Import Address Table (IAT) tárolja.
- A fordító a __declspec(dllimport) **int Add**(**int**,**int**) *;* függvény *tárolási osztály módosító* hatására generál optimálisabb kódot.
- A loadernek a betöltésnél csak az IAT-t kell kell megváltoztatni.
- A függvényeket nemcsak név, hanem sorszám (ordinal number) alapján is meg lehet hívni.

DII Export függvények feltérképezése

- Program kezdetének a megállapítása.
- DOS header-ben lévő érték alapján a NT header kezdetének az elérése.
- Opcional header-ből a export tábla kezdő és végcímének a kiszámítása (RVA).
- Függvénynevek, függvényekhez tartozó sorszámok és címek kiolvasása.

Ha a memóriába bejuttatott program nem tudja a rendszerhívások memória címét, akkor így tudja azokat kikeresni.

Irodalom

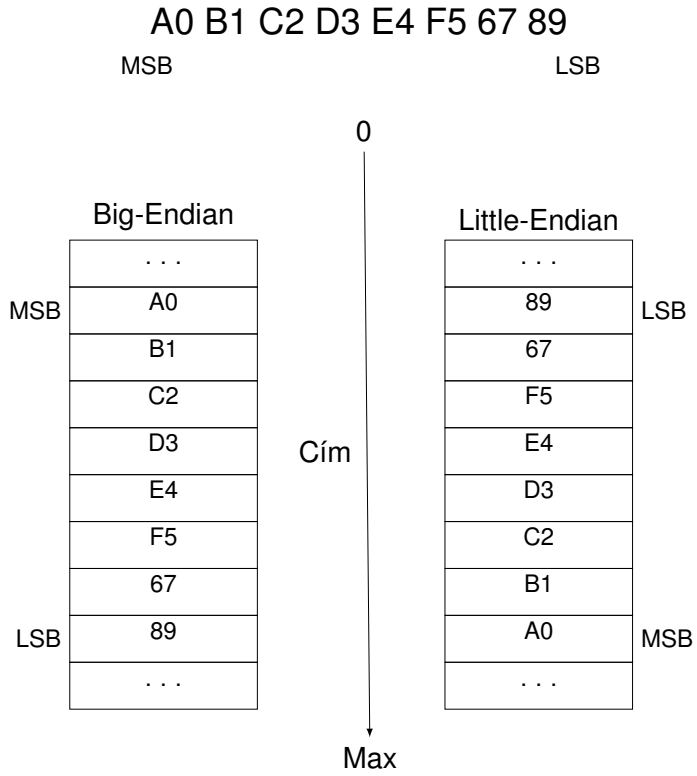
1. Matt Pietrek An In-Depth Look into the Win32 Portable Executable File Format `http://msdn.microsoft.com/en-us/magazine/bb985992.aspx`
2. Matt Pietrek Peering Inside the PE: A Tour of the Win32 Portable Executable File Format `http://msdn.microsoft.com/en-us/magazine/ms809762.aspx`
3. Microsoft PE and COFF Specification. `http://msdn.microsoft.com/en-us/library/windows/hardware/gg463119.aspx`

Section 3

X86 utasítás

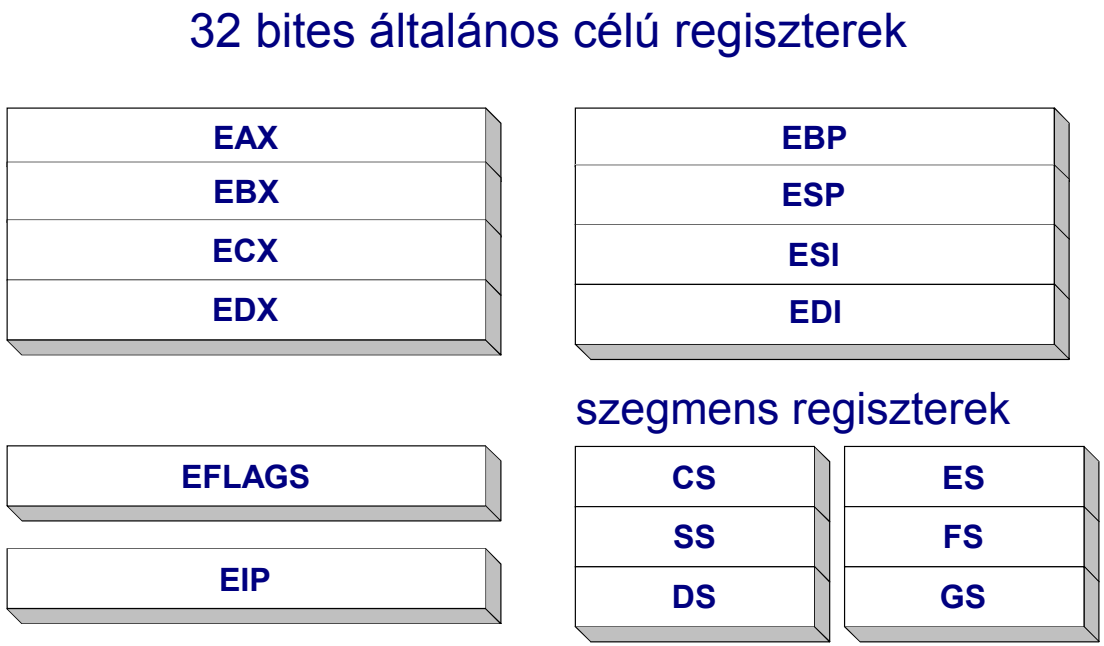
Byte-sorrend

A byte-sorrend megadja hogy a 2, 4, 8 byte-os adatok milyen sorrendben helyezkednek el a memóriában. A byte-on belül a bitek sorrendje változatlan.



X86 regiszterek

- 8 darab 32 bites regiszter;
- 6 darab szegmens regiszter;
- statusz regiszter EFLAGS
- utasításszámláló EIP



Regiszterek speciális szerepe egyes utasításokban

Speciális de nem kizárólagos felhasználása a regiszteréknek.

EAX akkumulátor regiszter, szorzáshoz osztáshoz;

ECX counter regiszter (ciklusszámlálásra);

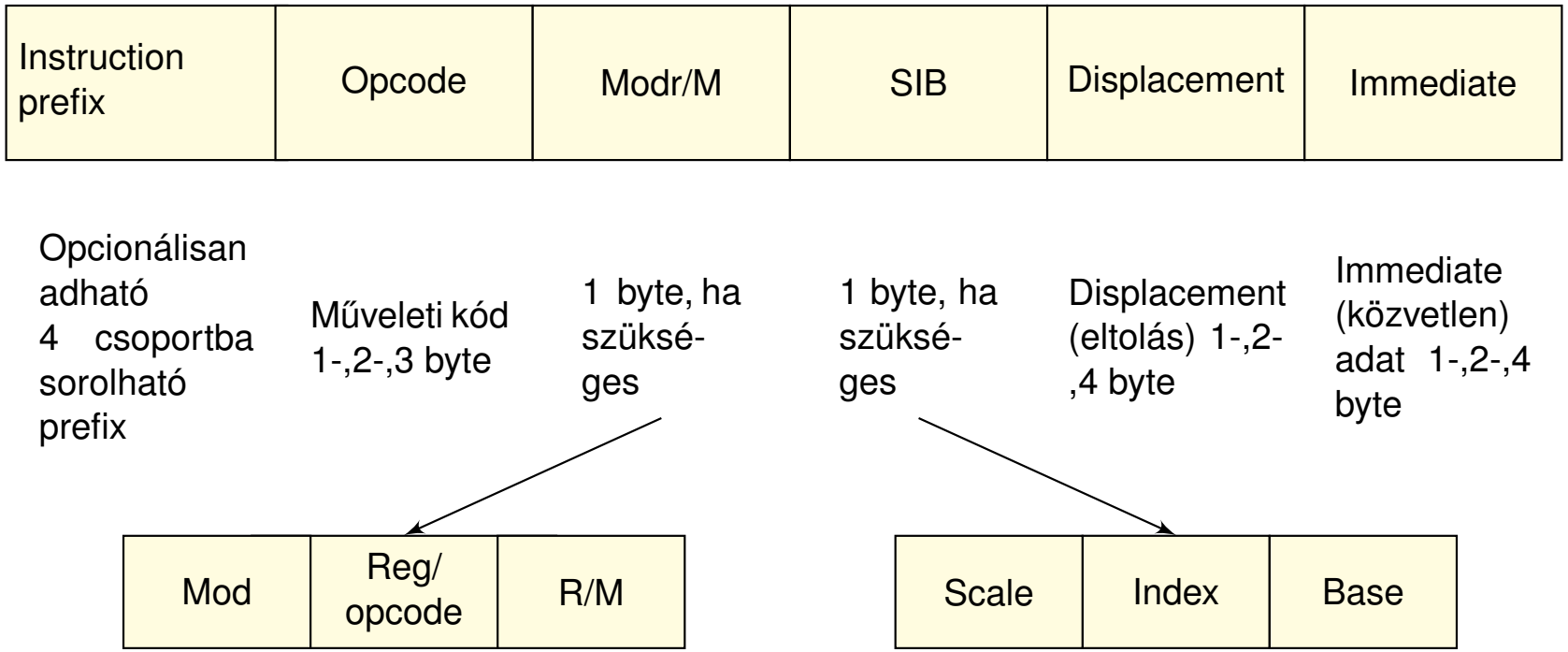
ESP stack pointer;

ESI string műveletek esetén a forrás memóriaterületet indexeli;

EDI string műveletek esetén a cél memóriaterületet indexeli;

EBP bázis pointer a stack kezeléshez.

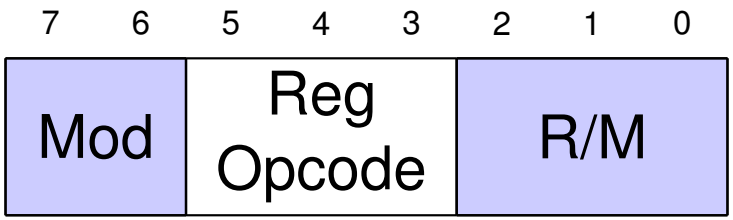
x86 utasítás formátum



Példa X86 utasításokra 1.

```
add eax,ecx          eax = eax + ecx;

sub esi,10           esi = esi - 10;
```

ModR/M

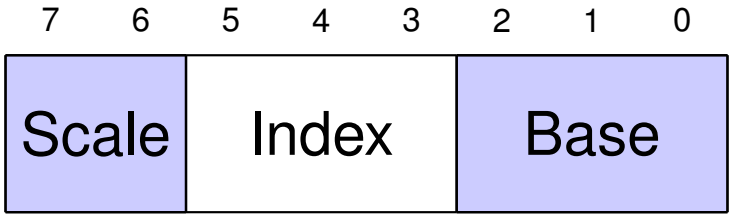


A *ModR/M* mező a műveletben résztvevő regisztereket vagy memória címzés típusát azonosítja.

- A *mod* és az *r/m* mező összevonva 5 bites, azaz 32 lehetséges értéket vehet fel. Ebből 8 érték regisztert címez, 24 pedig a címzési módot adja meg.
- A *reg/opcode* vagy regiszter cím, vagy 3 bit még hozzáadódik az utasításhoz, ami plusz információt hordoz.
- Az *r/m* mező a regiszter típusú operandust adja meg, vagy a *mod* mezővel kombinálva a címzési módot írja le.

Némely *ModR/M* értékhez még egy byte címzési információt kell elhelyezni, ez a *SIB* byte.

SIB mező



- skálafaktor
- index
- bázis

Néhány címzési módnál a *ModR/M* vagy ha van *SIB* byte után találjuk az eltolás (displacement) értéket. Ez az érték 1, 2, vagy 4 byte-os lehet.

Címszámítás a SIB alapján

$$\underset{base}{\begin{bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{bmatrix}} + \left[\underset{index}{\begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{pmatrix}} * \underset{scale}{\begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \end{pmatrix}} \right] + \underset{displacement}{\begin{bmatrix} nincs \\ 8-bit \\ 16-bit \\ 32-bit \end{bmatrix}}$$

$$\text{offset} = \text{base} + (\text{index} * \text{scale}) + \text{displacement}$$

ModR/M számítása

Illusztráció!

000	AL	AX	EAX	MM0	XMM0
001	CL	CX	ECX	MM1	XMM1
010	DL	DX	EDX	MM2	XMM2
011	BL	BX	EBX	MM3	XMM3
100	AH	SP	ESP	MM4	XMM4
101	CH	BP	EBP	MM5	XMM5
110	DH	SI	ESI	MM6	XMM6
111	BH	DI	EDI	MM7	XMM7

regiszterek számozása

MOD	11			
R/M			000	
REG		001		
ModR/M	11	001	000	= C8H
		ECX	EAX	

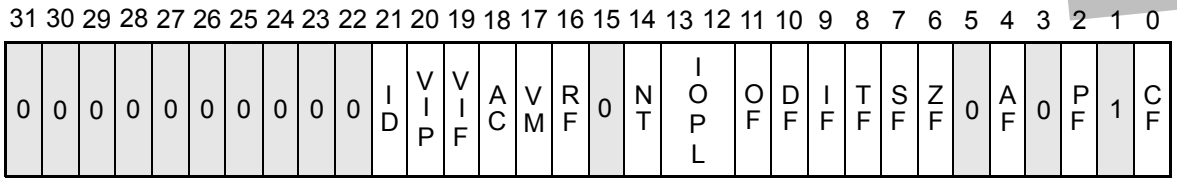
Példa X86 utasításokra 2.

Memória címzési módok:

```
mov   esi, [10128CF4]          esi = *(unsigned int *) 0x10128CF4;
mov   edx, [ecx+0A8h]          esi = *(unsigned int *) (ecx+0x0A8);
mov   edx, [ecx+eax*4]         esi = *(unsigned int *) (ecx+eax*4)
mov   edx,[ebp+esi*4-74h]      edx = *(unsigned int*)
                                   (ebp+esi*4 - 0x74);
```

Állapotregiszter

Illusztráció!



X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates Status Flag
C Indicates a Control Flag
X Indicates a System Flag

Foglalt bitek

Vezérlésátadó utasítások

Illusztráció!

Utasítás	Feltétel	Leírás
jmp	1	feltétel nélkül
je	ZF = 1	$=$ vagy 0
jne	ZF = 0	$\neq$
js	SF = 1	< 0
jns	SF= 0	≥ 0
jg	((SF XOR OF) OR ZF) = 0	$>$
jge	(SF XOR OF) = 0	$\geq$
jl	(SF XOR OF) = 1	$<$
jle	((SF XOR OF) OR ZF) = 1	$\leq$
ja	(CF OR ZF) = 0	>
jb	CF = 1	<

Vezérlésátadó utasítások

Illusztráció!

Utasítás	Kód (bináris)	Feltétel 3-2-1 bitek	Negálás 0 bit
JZ	01110100	010	0
JNZ	01110101	010	1
JB	01110010	001	0
JNB	01110011	001	1
JG	01111111	111	1
JNG	01111110	111	0

Intel X86 utasításkészlet:

`http://ref.x86asm.net/index.html`

Section 4

stack

Stack feladata

Eljárás paraméterei és lokális változóinak az élettertama az eljárás futási idejére korlátozódik. A fordító ezeknek a változóknak csak a függvény hívásától a visszatéréséig foglal helyet, fordítóírók terminológiájával élve az *aktivációs rekord* létrehozásával. X86 architekturán ez a stacket jelenti.

Stack működésének a részei

- aktív regiszter értékének ideiglenes tárolása;
- paraméterek átadása;
- visszatérési cím elmentése;
- lokális adatok helyének a biztosítása;

A függvények hívásához különböző hívási konvekciókat használhatunk.

Stackkezelő utasítások

```
push eax
```

```
sub esp, 4
mov dword ptr [esp], eax
```

```
pop eax
```

```
mov eax, dword ptr [esp]
add esp, 4
```

Call és return

Eljárás hívás **call** (kódszegmens nem változik)

1. **push** <return address>
2. **jmp** func

Visszatérés a hívóhoz **ret** n:

1. **add esp**, n
2. **jmp dword ptr** [**esp**−n][4]

[4] n a letett paraméterek stacken elfoglalt mérete byte-ban.

Stack pointer munkában

Stack felszabadítása nagyobb egységekben

```
pop eax          ≡      add esp, 12
pop eax
pop eax
```

Stack lefoglalás nagyobb egységekben

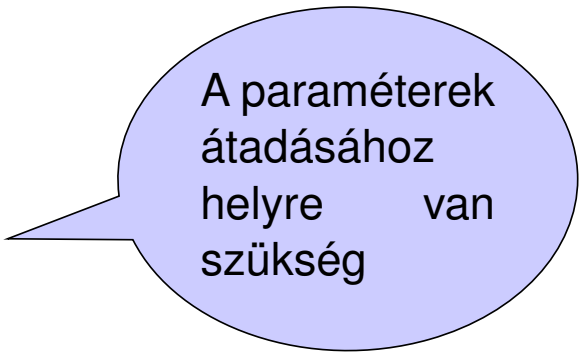
```
sub esp,400
```

Stack megtekintés

```
mov eax, [esp+4]
```

C példaprogram

```
int func(int i, int j, int k )
{
    double r;
    int l=0;
    l = i;
    l +=j;
    l +=k;
    return l;
}

int main()
{
    int a=1;
    int b=2;
    int c=3;
    a=func(a,b,c);
    return 0;
}
```



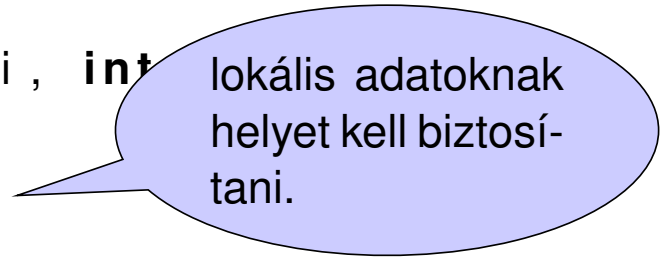
C példaprogram

```
int func(int i, int j, int k )
{
    double r;
    int l=0;
    l = i;
    l +=j;
    l +=k;
    return l;
}

int main()
{
    int a=1;
    int b=2;
    int c=3;
    a=func(a,b,c);
    return 0;
}
```

Át kell venni a paramétereket.

C példaprogram

```
int func(int i, int
{
    double r;
    int l=0;
    l = i;
    l +=j;
    l +=k;
    return l;
}

int main()
{
    int a=1;
    int b=2;
    int c=3;
    a=func(a,b,c);
    return 0;
}
```



C példaprogram

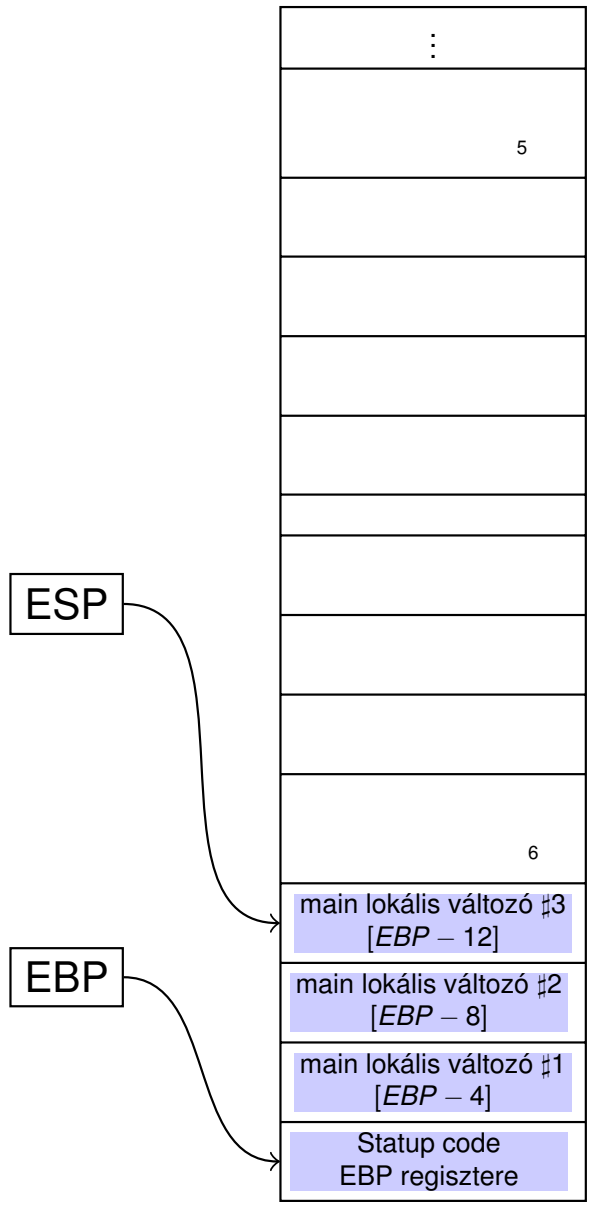
```
int func(int i, int j, int k )
{
    double r;
    int l=0;
    l = i;
    l +=j;
    l +=k;
    return l;
}

int main()
{
    int a=1;
    int b=2;
    int c=3;
    a=func(a,b,c);
    return 0;
}
```

Vissza kell adni az értéket.

__cdecl hívási konvenció, hívó feladata

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

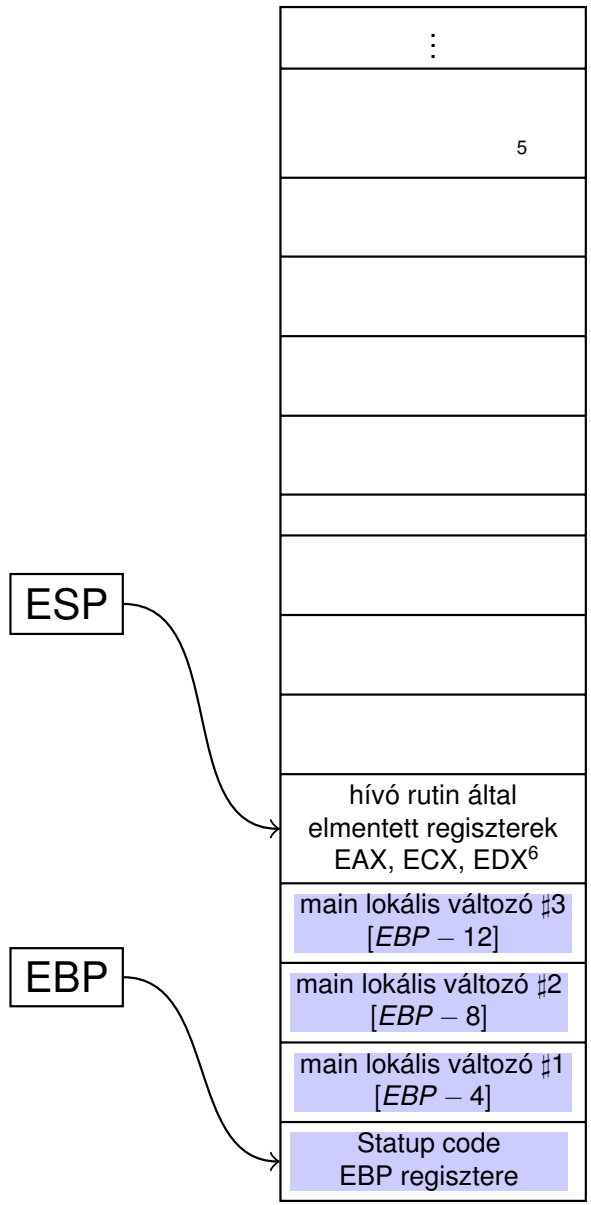
[5]csak ha szükséges



__cdecl hívási konvenció, hívó feladata

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

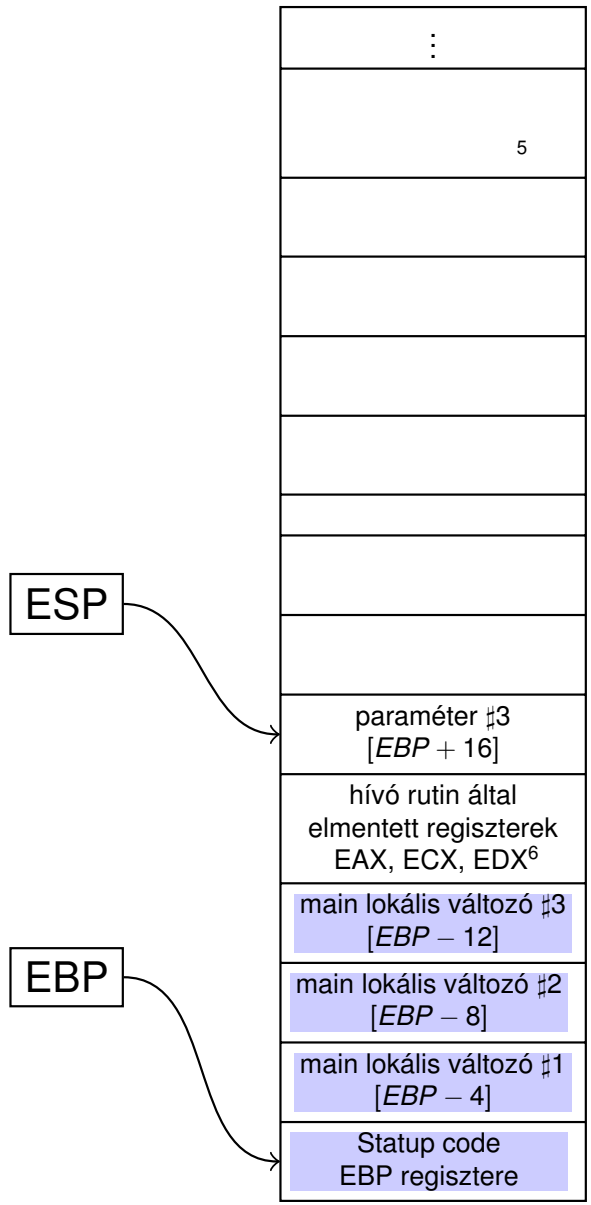
[5]csak ha szükséges



__cdecl hívási konvenció, hívó feladata

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

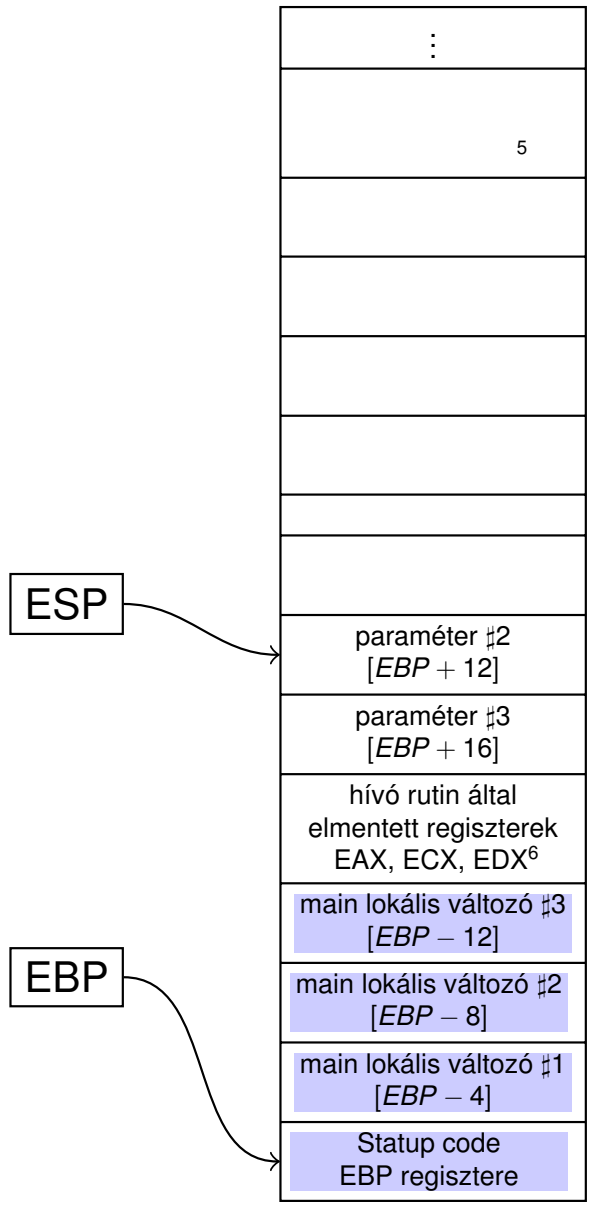
[5]csak ha szükséges



__cdecl hívási konvenció, hívó feladata

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

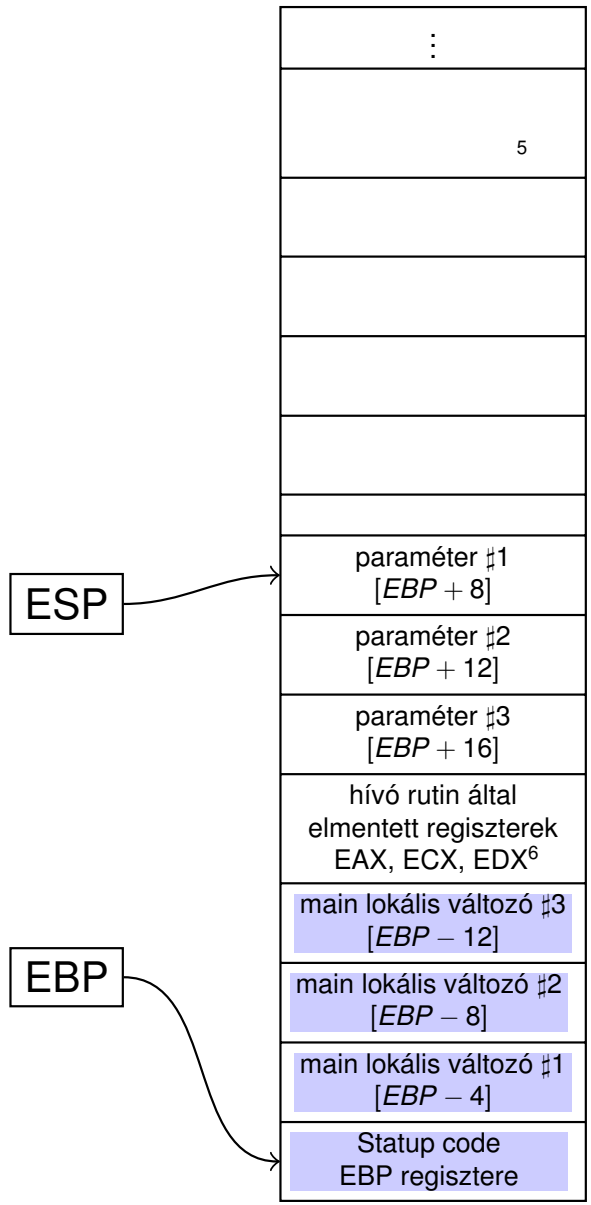
[5]csak ha szükséges



__cdecl hívási konvenció, hívó feladata

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

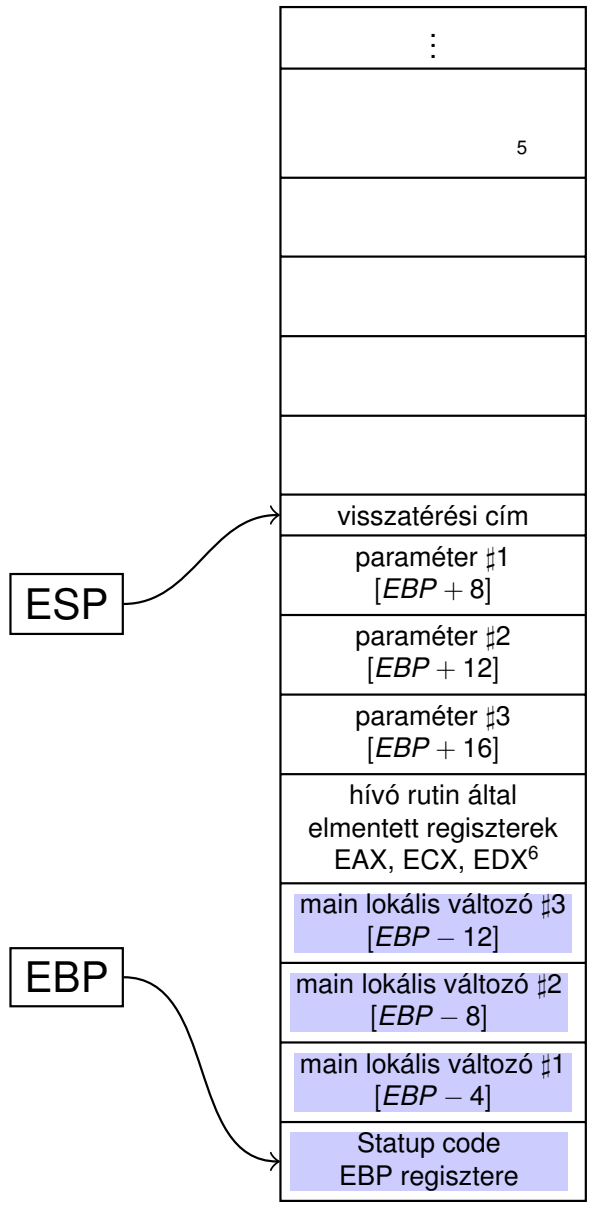
[5]csak ha szükséges



__cdecl hívási konvenció, hívó feladata

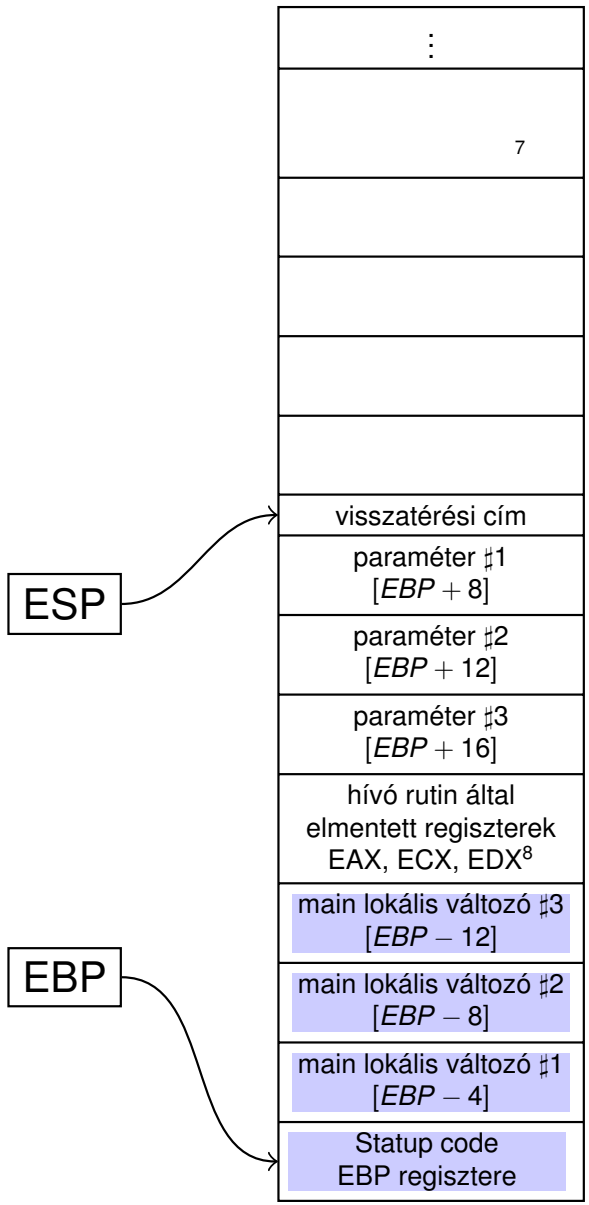
```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

[5]csak ha szükséges



__cdecl hívási konvenció, hívott feladata

```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

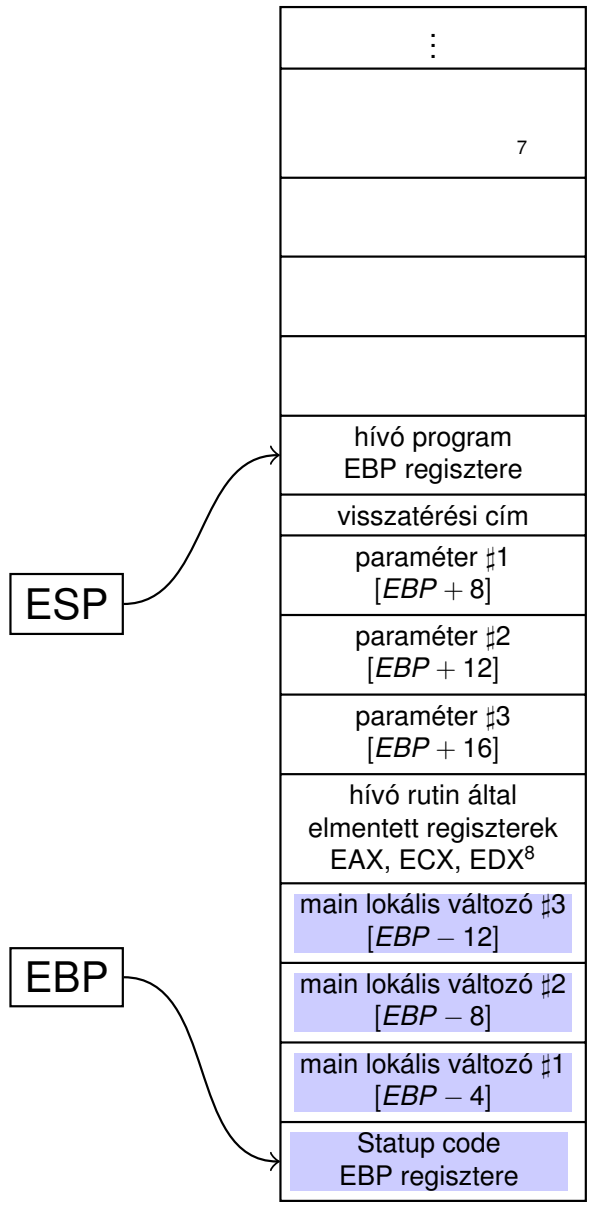


[7] csak ha szükséges

__cdecl hívási konvenció, hívott feladata

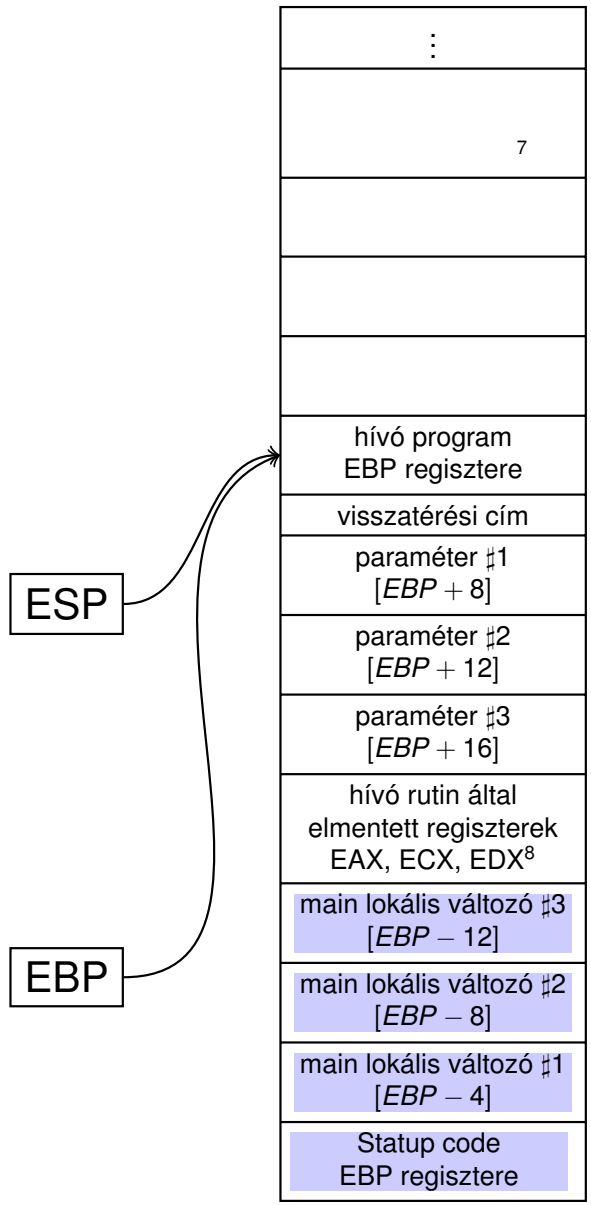
```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

[7] csak ha szükséges



__cdecl hívási konvenció, hívott feladata

```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

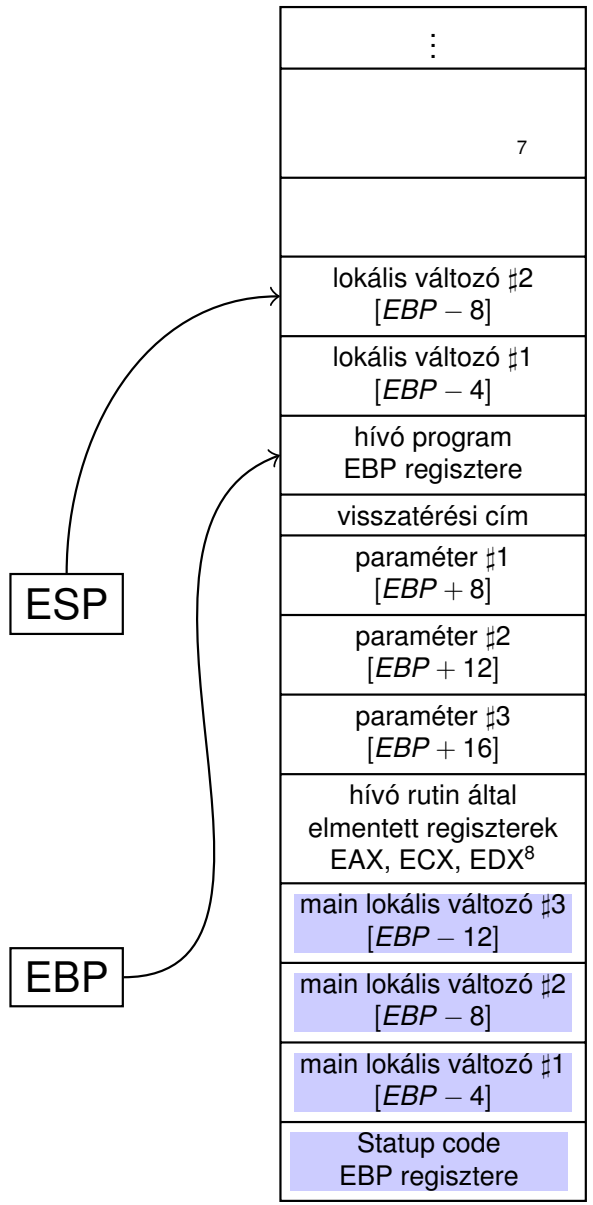


[7] csak ha szükséges

__cdecl hívási konvenció, hívott feladata

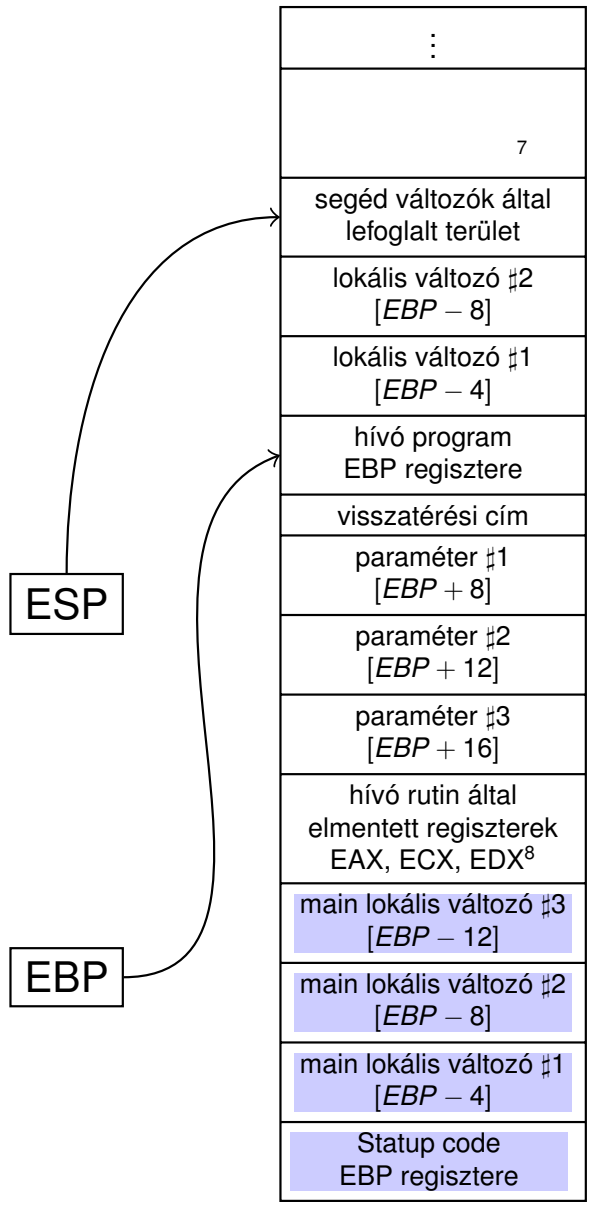
```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

[7] csak ha szükséges



__cdecl hívási konvenció, hívott feladata

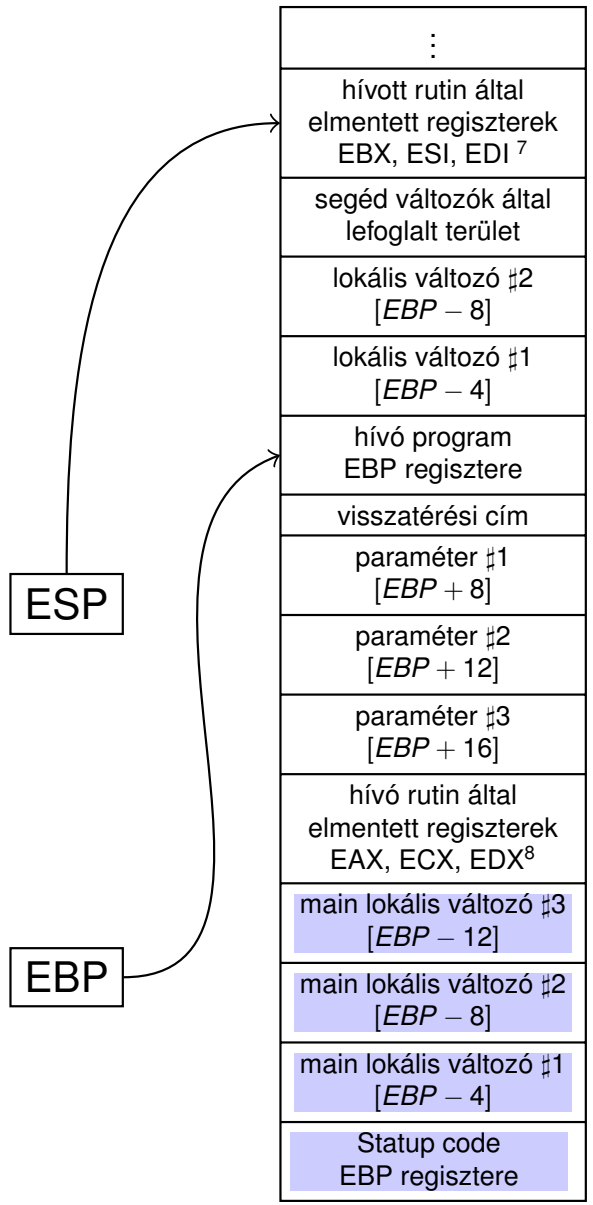
```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```



[7] csak ha szükséges

__cdecl hívási konvenció, hívott feladata

```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

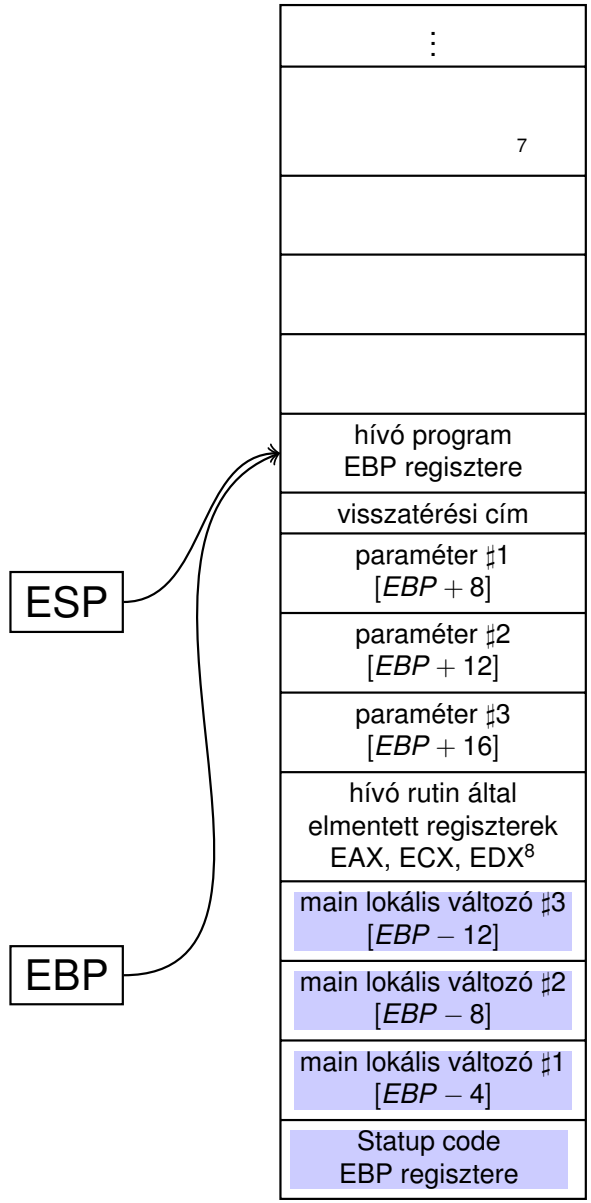


[7] csak ha szükséges

__cdecl hívási konvenció, hívott feladata

```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

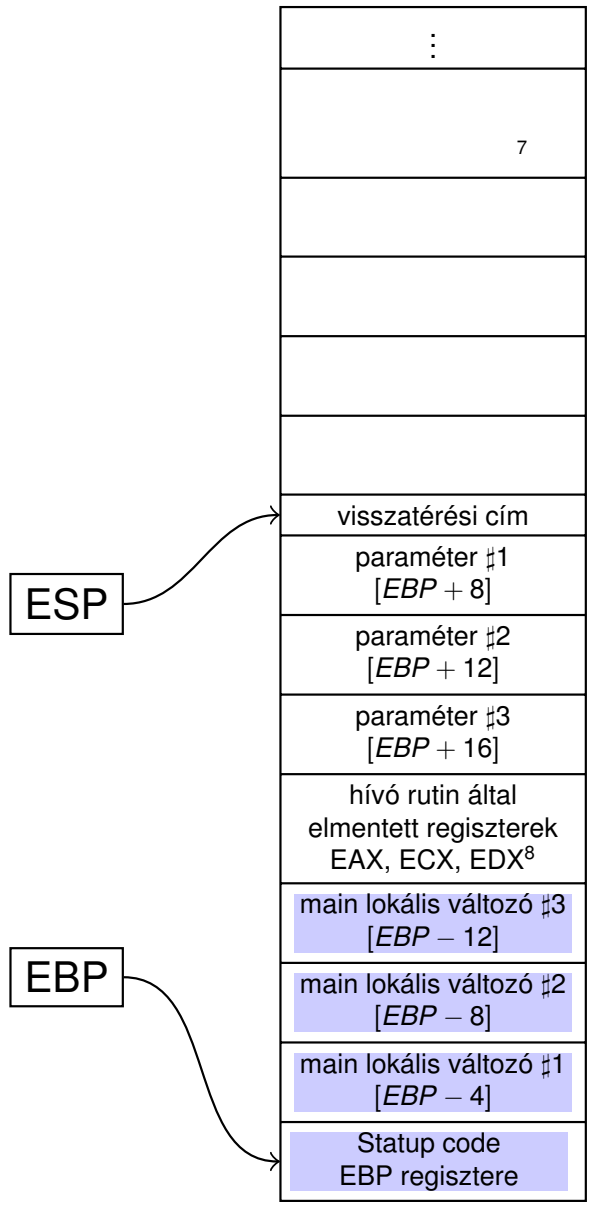
[7] csak ha szükséges



__cdecl hívási konvenció, hívott feladata

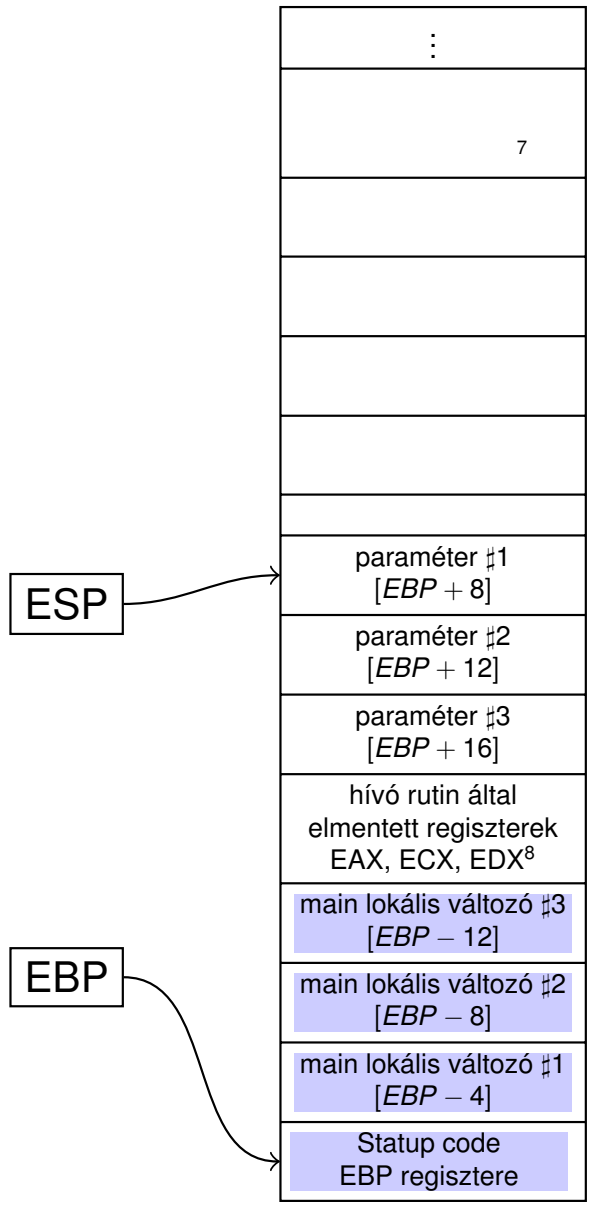
```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

[7] csak ha szükséges



__cdecl hívási konvenció, hívott feladata

```
_func:
    push    ebp
    mov     ebp,esp
    sub     esp,10h
    mov     dword ptr [ebp-4],0
    mov     eax,dword ptr [ebp+8]
    mov     dword ptr [ebp-4],eax
    mov     ecx,dword ptr [ebp-4]
    add     ecx,dword ptr [ebp+0Ch]
    mov     dword ptr [ebp-4],ecx
    mov     edx,dword ptr [ebp-4]
    add     edx,dword ptr [ebp+10h]
    mov     dword ptr [ebp-4],edx
    mov     eax,dword ptr [ebp-4]
    mov     esp,ebp
    pop     ebp
    ret
```

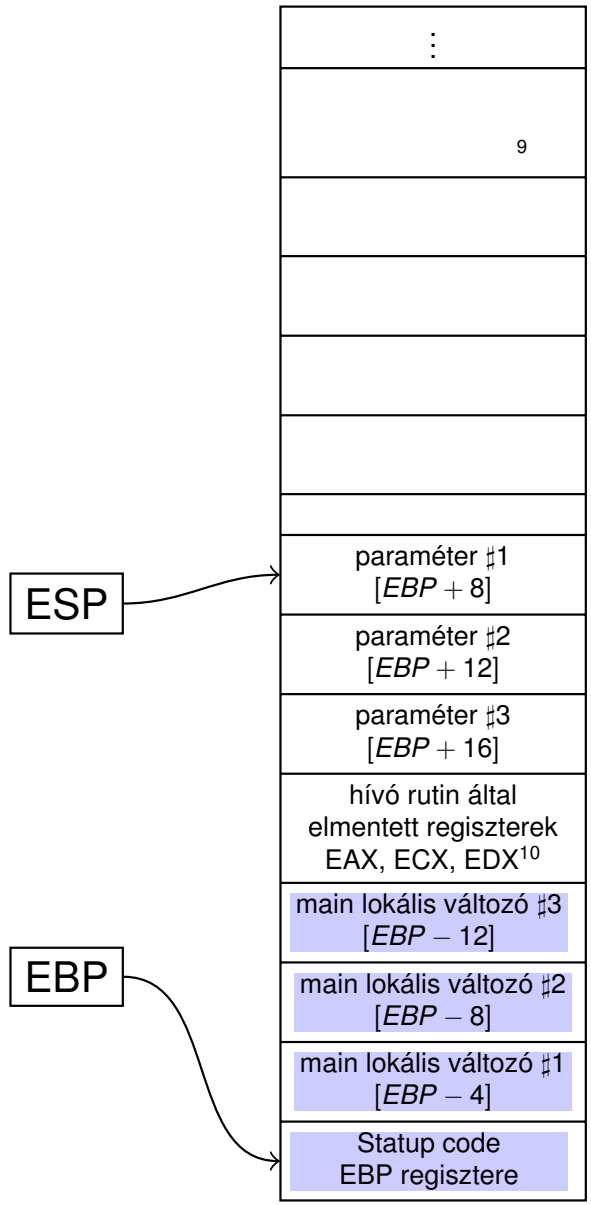


[7] csak ha szükséges

__cdecl hívási konvenció, visszatérés a hívóhoz

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

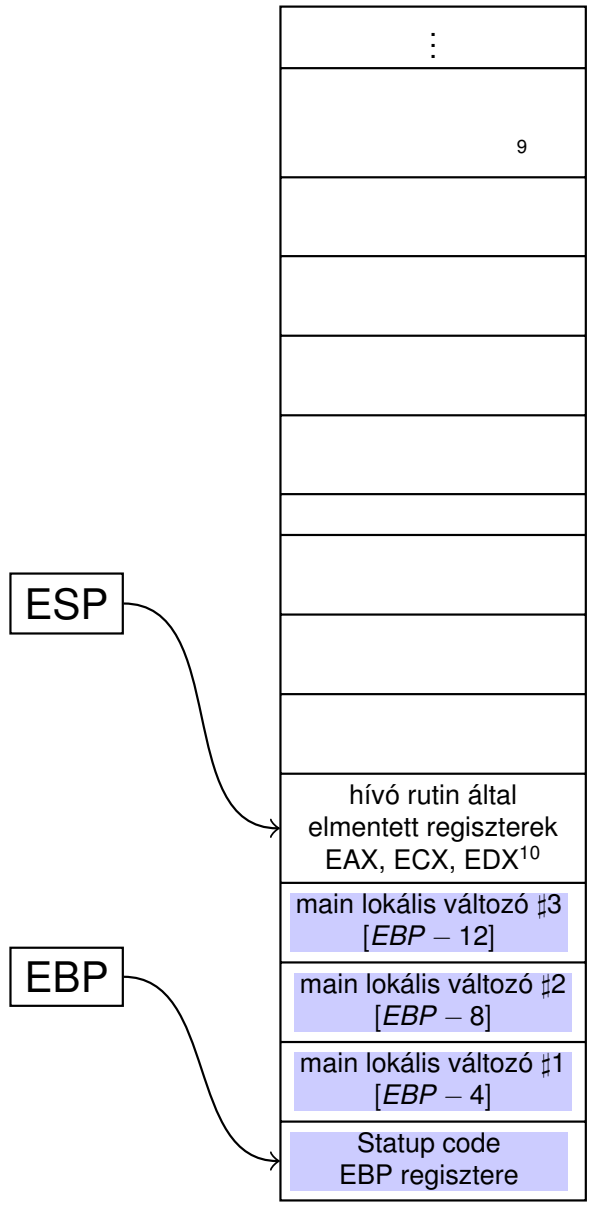
[9] csak ha szükséges



__cdecl hívási konvenció, visszatérés a hívóhoz

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

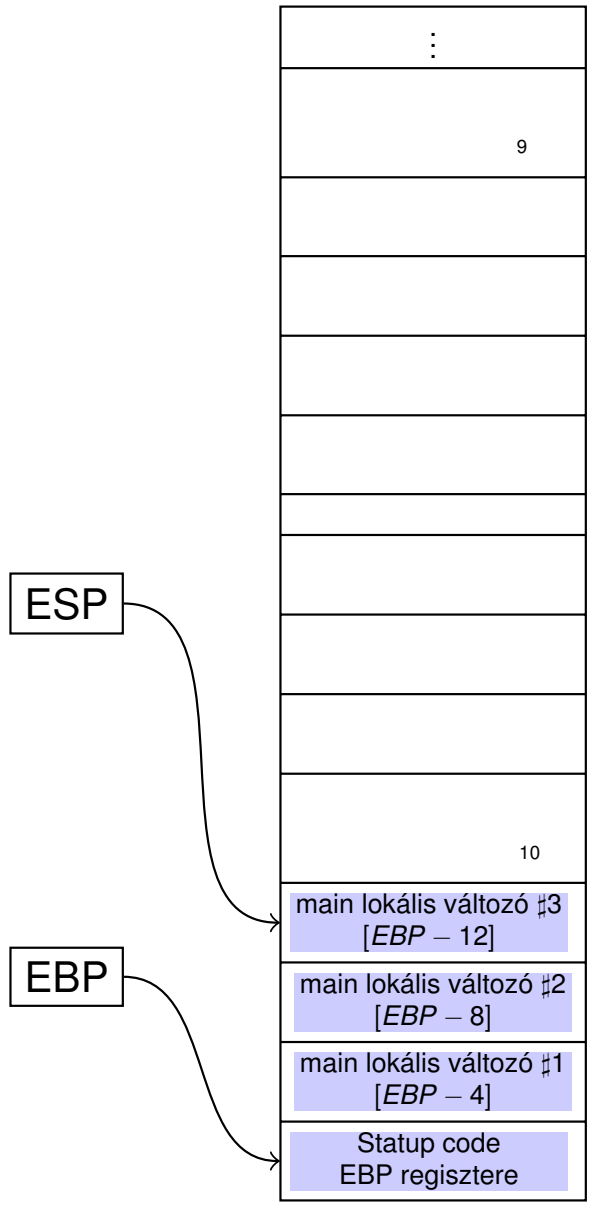
[9] csak ha szükséges



__cdecl hívási konvenció, visszatérés a hívóhoz

```
_main:
    push    ebp
    mov     ebp,esp
    sub     esp,0Ch
    mov     dword ptr [ebp-4],1
    mov     dword ptr [ebp-8],2
    mov     dword ptr [ebp-0Ch],3
    mov     eax,dword ptr [ebp-0Ch]
    push    eax
    mov     ecx,dword ptr [ebp-8]
    push    ecx
    mov     edx,dword ptr [ebp-4]
    push    edx
    call    _func
    add     esp,0Ch
    mov     dword ptr [ebp-4],eax
    xor     eax,eax
    mov     esp,ebp
    pop     ebp
    ret
```

[9]csak ha szükséges



__cdecl hívási konvenció X86

1. Hívó feladata a hívás előtt
   - szükség esetén elmenteni az **EAX**, **ECX EDX**-t;
   - átadni a függvény paramétereit;
   - meghívni a függvényt (**call**).
2. Hívott feladata az eljárás elején
   - elmenteni az **EBP** regisztert;
   - a lokális adatoknak helyet foglalni;
   - szükség esetén elmenteni az **EBX**, **ESI** és **EDI**-t.
3. Hívott feladata a visszatérés előtt
   - az **EAX** regiszterbe betenni a visszatérési értéket;
   - visszaállítani az elmentett **EBX**, **ESI**, **EDI** regisztereket;
   - **ESP** stack pointert az **EBP** által mutatott helyre állítani;
   - az elmentett stack báziscím visszaállítása az **EBP**-be;
   - visszatérni a hívóhoz (**ret**).
4. Hívó feladata a hívás után
   - felszabadítani a függvény paramétereit;
   - az **EAX**-ben lévő visszatérési érték elmentése;
   - az **EAX**, **ECX**, **EDX** regiszterek visszaállítása.

Termiológia

- Függvény prológus (előszó) a függvény elején azok az utasítások, amelyek elmentik a használt regisztereket, és előkészítik a stack keretet.

```
push ebp
ebp = esp
esp = esp - <frame space>
```

- Függvény epilógus (utószó) a függvény végén azok az utasítások, amelyek törlik a stack keretet, és visszaállítják az elmentett regisztereket.

```
leave       // esp = ebp, pop ebp
ret         // pop eax, jump eax
```

Egyéb hívási konvenciók

stdcall esetén a hívott rutin szabadítja fel a paramétereknek lefoglalt helyet. Windows API függvények ezt használja.

```
int __stdcall func(int i, int j, int k)
{
.......
}
```

fastcall esetén egyes paraméterek a regiszterekbe kerülnek.

```
int __fastcall func(int i, int j, int k)
{
.......
}
```

__stdcall hívási konvenció

Illusztráció!

```
_func@12:
    push ebp
    mov ebp,esp
    sub esp,10h
    mov dword ptr [ebp-4],0
    mov eax,dword ptr [ebp+8]
    mov dword ptr [ebp-4],eax
    mov ecx,dword ptr [ebp-4]
    add ecx,dword ptr [ebp+0Ch]
    mov dword ptr [ebp-4],ecx
    mov edx,dword ptr [ebp-4]
    add edx,dword ptr [ebp+10h]
    mov dword ptr [ebp-4],edx
    mov eax,dword ptr [ebp-4]
    mov esp,ebp
    pop ebp
    ret 0Ch
```

```
_main:
    push ebp
    mov ebp,esp
    sub esp,0Ch
    mov dword ptr [ebp-4],1
    mov dword ptr [ebp-8],2
    mov dword ptr [ebp-0Ch],3
    mov eax,dword ptr [ebp-0Ch]
    push eax
    mov ecx,dword ptr [ebp-8]
    push ecx
    mov edx,dword ptr [ebp-4]
    push edx
    call _func@12
    mov dword ptr [ebp-4],eax
    xor eax,eax
    mov esp,ebp
    pop ebp
    ret
```

Paramétereket a hívott szabadítja fel. Gyorsabb a __cdecl-nél, de nem lehet változó paraméterszámú függvényt írni.

__fastcall hívási konvenció

Illusztráció!

```
@func@12:
    push ebp
    mov ebp,esp
    sub esp,18h
    mov dword ptr [ebp-18h],edx
    mov dword ptr [ebp-14h],ecx
    mov dword ptr [ebp-4],0
    mov eax,dword ptr [ebp-14h]
    mov dword ptr [ebp-4],eax
    mov ecx,dword ptr [ebp-4]
    add ecx,dword ptr [ebp-18h]
    mov dword ptr [ebp-4],ecx
    mov edx,dword ptr [ebp-4]
    add edx,dword ptr [ebp+8]
    mov dword ptr [ebp-4],edx
    mov eax,dword ptr [ebp-4]
    mov esp,ebp
    pop ebp
    ret 4
```

```
_main:
    push ebp
    mov ebp,esp
    sub esp,0Ch
    mov dword ptr [ebp-4],1
    mov dword ptr [ebp-8],2
    mov dword ptr [ebp-0Ch],3
    mov eax,dword ptr [ebp-0Ch]
    push eax
    mov edx,dword ptr [ebp-8]
    mov ecx,dword ptr [ebp-4]
    call @func@12
    mov dword ptr [ebp-4],eax
    xor eax,eax
    mov esp,ebp
    pop ebp
    ret
```

Hasonlít az __stdcall-ra, de pár paramétert regiszterben ad át.