```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C NYELV ÉS AZ ASSEMBLY PROGRAMOZÁS

Dr. Varga Imre

Debreceni Egyetem

Informatikai Rendszerek és Hálózatok Tanszék

2021. augusztus 21.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A diasorozat célja

Ez a diasorozat nem minősül önálló tanulást támogató oktatási segédanyagnak.

Inkább tekinthető rövid összefoglalónak, emlékeztetőnek, gondolatébresztőnek.

Az „Assembly programozás” tantárgy óráinak keretében minden szükséges ismeretet, képességet megszerezhetsz.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C nyelv típusai

Aritmetikai (egyszerű) típusok

- [unsigned|signed] **char**
- [unsigned|signed] [short|long|long long*] **int**
- **enum**
- **float**
- [long*] **double**

Összetett/származtatott típusok

- **struct** és **union**
- *tömb*
- *mutató*

*ISO C99

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C egész típusai

típus	méret	tartomány	
[signed] char	1 bájt	-128	127
unsigned char		0	255
[signed] short	2 bájt	-32.768	32.767
unsigned short		0	65.535
[signed] int	4 bájt (2 bájt)	-2.147.483.648	2.147.483.647
unsigned int		0	4.294.967.295
[signed] long	8 bájt (4 bájt)	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned long		0	18.446.744.073.709.551.615
[signed] long long*	8 bájt	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned long long		0	18.446.744.073.709.551.615

*ISO C99

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Felsorolásos típus

- Alacsony szinten `int`-ként reprezentált
- C nyelven a két kód assembly szinten azonos

```
enum szam{egy=1,ketto,harom};
int main(){
  enum szam x;
  x=egy+harom+10;
  return x; }

#define egy 1
#define harom 3
int main(){
  int x;
  x=egy+harom+10;
  return x; }
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C nyelv valós típusai

Valós típusú konstansok formátuma

- 0.751
- .751f
- 0.751L
- +.75100
- 75.1e-2
- 0.0751E+1

Valós típusok paraméterei

	méret	minimum	maximum	pontosság
float	4 bájt	$1.18 \cdot 10^{-38}$	$3.40 \cdot 10^{+38}$	≈7 számjegy
double	8 bájt	$2.23 \cdot 10^{-308}$	$1.80 \cdot 10^{+308}$	≈15 számjegy
long double*	10 bájt	$3.36 \cdot 10^{-4932}$	$1.19 \cdot 10^{+4932}$	≈19 számjegy

*ISO C99

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Mutató típus

- A **C** nyelvben
  - Fontos szerepet játszik (mutatóorientált nyelv)
  - Előjel nélküli egész reprezentáció (32 vagy 64 bit)
  - NULL: csupa nulla bit
  - Értéke adatcím és kódcím (!) is lehet
  - Aritmetikai műveletek is végezhetőek velük

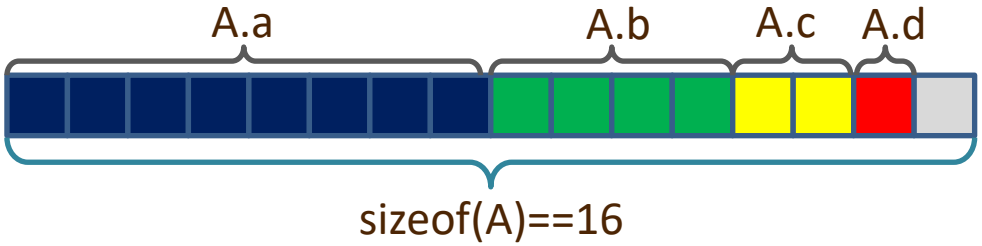
```
int a, b, c;
int *p=&b+sizeof(int);
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

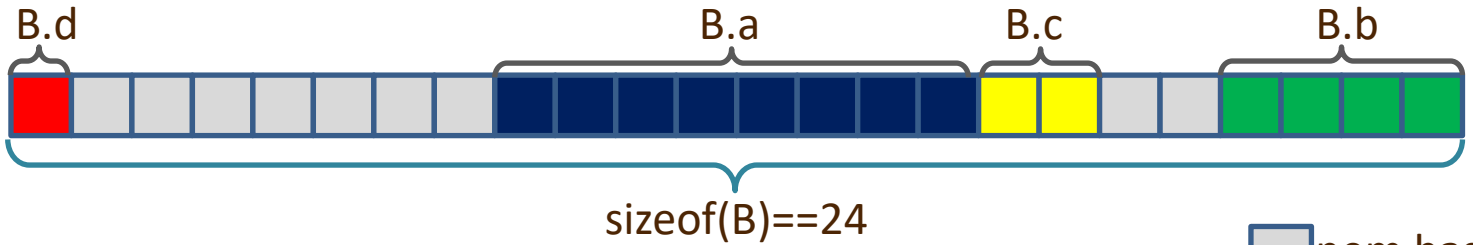
Rekord típus

- `struct` kulcsszó
- Nem biztos, hogy folytonos memóriaterületen

```
struct x{ long a; int b; short c; char  d;} A;
```




```
struct y{ char d; long a; short c; int  b;} B;
```



```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C nyelv tömbje

- Tömb index 0-tól (db-1)-ig
- Csak egydimenziós tömb van, viszont van tömbök tömbje `int M[3][2]={{1,2},{3,4},{5,6}};`
- Sztring: karaktertömb
  - ➥ Lezáró karakter ’\0’ (ASCII kód: 0)
- A tömb neve az első elemre mutató nevesített konstans (mutatóorientáltság erős)

```
int T[8];
  T[4]=2;
*(T+4)=2;
```

egyenértékű utasítások

```
int M[7][8];
int *p=&M[0][0];
M[3][5]=3;
*(p+(3*8)+5)=3;
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Konstans (literál)

- Közvetlen adat megadás (immediate)
- Kódba épített adat
- 8, 16, 32, 64 bit szélességű
- Fix vagy lebegőpontos ábrázolás
- Magas szinten:
  `123, 1.25f, 'a'`
- Alacsony szinten:
  `0x0000007b, 0x3fa00000, 0x61`

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Nevesített konstans

Komponensei: név, típus, érték

- `#define abc 123`

  Előfordító kicseréli a nevet az értékre

  Kódba épített adat

  Assembly szinten egyszerű konstans

- `const int abc=123;`

  Assembly szinten egyszerű változó

  A fordító nem engedi a változtatást

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Változó

- Komponensei: név, attribútum, cím, érték
- Deklaráció: explicit, implicit, automatikus
- Hatáskör: statikus, dinamikus
- Élettartam: statikus, dinamikus, programozó által vezérelt
- Értékadás: kezdőértékadás (automatikus, explicit), értékadó utasítás, paraméterátadás, input utasítás

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Változó

Alacsony szinten

- Adott méretű lefoglalt memóriaterület
- Ha van cím komponens mindig van érték is
- Érték: bitsorozat (tetszőleges értelmezéssel)
- Értékadás: memóriacímre adat mozgatás
- Statikus változó adat szegmensben
- Dinamikus változó verem szegmensben
- Programozó által vezérelt a heap-ben
- Mutató: előjel nélküli egész változó C-ben

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Kifejezés

- Komponensei: típus, érték
- Formálisan: operátor, operandus és zárójel
- Operátor: unáris, bináris, ternáris
- Alak: **infix** (2+3)*4
  prefix * + 2 3 4
  postfix (RPN) 2 3 + 4 *
- Infix alak esetén nem egyértelmű kiértékelés: precendencia és kötésirány kell
- Típusegyenértékűség vagy típuskényszerítés

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C precedencia táblázata

	Operátor	Leírás	Kötés-irány
1.	()	Zárójel, függvény paraméter lista	→
	[]	Tömb indexelés	
	.	Mező hivatkozás	
	->	Mező hivatkozás mutatóval	
	++ --	Utólagos inkrementálás/dekrementálás	
2.	++ --	Előzetes inkrementálás/dekrementálás	←
	+ -	Előjel	
	! ~	Logikai/bitenkénti tagadás	
	(típus)	Típus kényszerítés	
	*	Mutatott terület	
	&	Címképzés	
	sizeof	Méret	
3.	* / %	Szorzás/osztás/maradékos osztás	→
4.	+ -	Összeadás/kivonás	→
5.	<< >>	Shiftelés balra/jobbra	→
6.	< <= > >=	Relációs operátorok (< ≤ > ≥)	→
7.	== !=	Relációs operátorok (= ≠)	→
8.	&	Bitenkénti ÉS	→
9.	^	Bitenkénti XOR	→
10.	|	Bitenkénti VAGY	→
11.	&&	Logikai ÉS	→
12.	||	Logikai VAGY	→
13.	?:	Feltételes operátor	←
14.	= += -= *= /= %= <<= >>= &= ^= |=	Értékadó operátorok	←
15.	,	Vessző operátor	→

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Kifejezés kiértékelés

- A kifejezés értékének, típusának meghatározása
- Konstans kifejezést a fordító értékel ki
- Nem konstans infix kifejezést a fordító postfix alakra hozza (figyelembe véve a zárójeleket, precedenciát és a kötésirányt) és az alapján állítja elő a gépi kódot

eax + ebx * ecx & edx

```
imul ebx, ecx
add  eax, ebx
and  eax, edx
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Utasítások

Deklarációs utasítás

- Nem (biztos, hogy) áll mögötte gépi kód
- A fordítóprogramnak szól

Végrehajtható utasítás

- Egy magas szintű utasításból több gépi kódú utasítást is előállíthat a fordító
- Csoportjai: üres, értékadó, ugró, elágaztató, ciklusszervező, hívó, egyéb

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Elágaztató utasítás

Kétirányú elágaztató utasítás C nyelven:

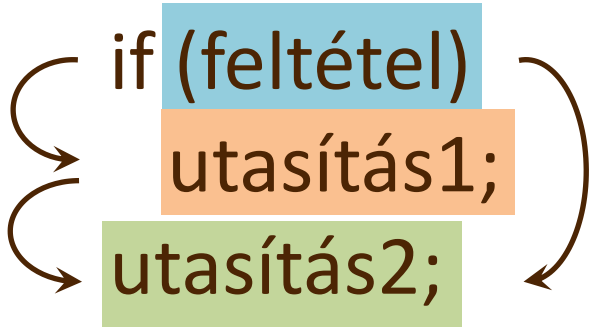
```
if (kifejezés)
utasítás1;
[else
utatsítás2;]
```

Többirányú elágaztató utasítás C nyelven:

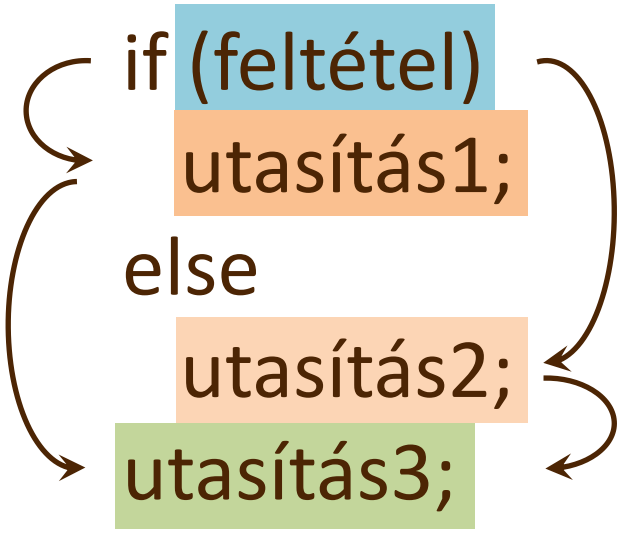
```
switch(kifejezés){
case egész_konstans_kif_1: [utasítás1;]
[case egész_konstans_kif_2: [utasítás2;]]
[default: utasítás3;]
}
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A feltételes utasítás alacsony szinten



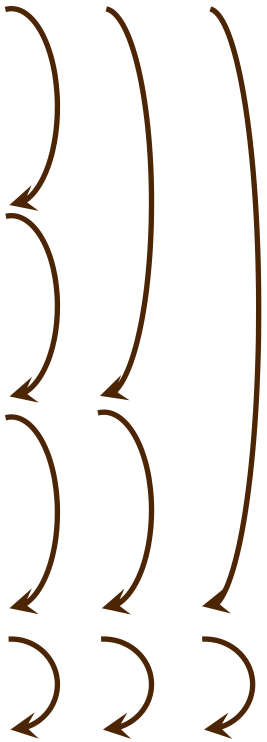

```
        cmp  eax, ebx
        jne  .L0
        mov  ecx, 1
.L0:    add  edx, 1
```




```
        cmp  eax, ebx
        jne  .L2
        mov  ecx, 1
        jmp  .L3
.L2:    mov  ecx, 2
.L3:    add  edx, 1
```

A switch utasítás alacsony szinten

```
switch (kifejezés){
  case kifejezes1:
    utasítás1;
  case kifejezes2:
    utasítás2;
  default:
    utasítás3;}
utasítás4;
```

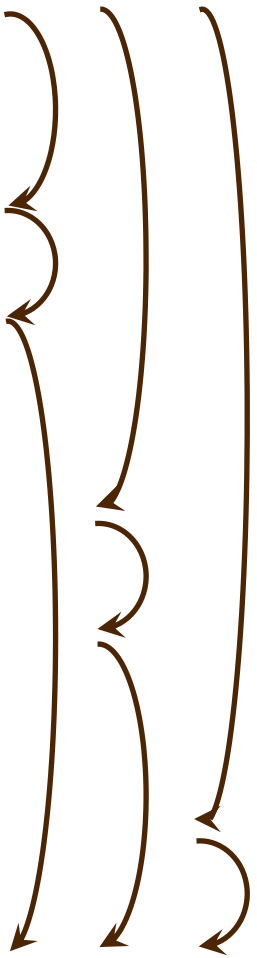


```
      cmp  eax, 1
      je   .L3
      cmp  eax, 2
      je   .L4
      jmp  .L2
.L3:  mov  ebx, 1
.L4:  mov  ebx, 2
.L2:  mov  ebx, 3
      add  edx, 1
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A break hatása alacsony szinten

```
switch (kifejezés){
  case 1:
    utasítás1;
    break;
  case 2:
    utasítás2;
    break;
  default:
    utasítás3;}
utasítás4;
```



```
      cmp   eax, 1
      je    .L3
      cmp   eax, 2
      je    .L4
      jmp   .L2
.L3:  mov   ebx, 1
      jmp   .L5
.L4:  mov   ebx, 2
      jmp   .L5
.L2:  mov   ebx, 3
.L5:  add   edx, 1
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Ciklusszervező utasítások

Felépítés: fej + mag + vég

Működés szempontjából: üres, ‘normál’, végtelen

Fajtái:

- Feltételes
  - Kezdőfeltételes
  - Végfeltételes
- Előírt lépésszámú
  - Előltesztelő
  - Hátultesztelő
- Felsorolásos
- Végtelen

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A C nyelv ciklusai

- Kezdőfeltételes (nem 0 feltétel esetén ismétel)

```
while (feltétel)
    utasítás;

for ([kif1]; [kif2]; [kif3])
    utasítás;
```

- Végfeltételes (nem 0 feltétel esetén ismétel)

```
do
    utasítás;
while (feltétel);
```

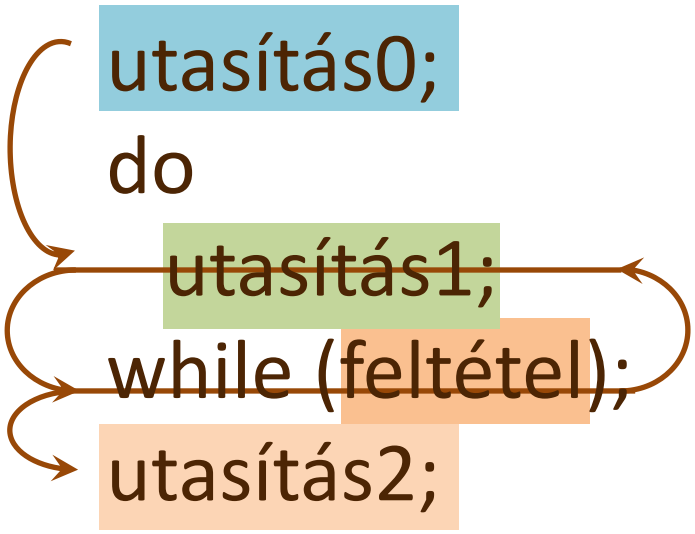
A while és for ciklus alacsony szinten


```
   x=5;
   goto test;
loop:
   y=y+1;
   x=x-1;
test:
   if (x>0) goto loop;
```

```
        mov  eax, 10
        jmp  .L2
.L3:    sub  eax, 1
.L2:    cmp  eax, 0
        jne  .L3
        add  ebx, 1
```

A for és a while ciklus assembly szinten egyenértékű.

A do-while ciklus alacsony szinten




```
    mov  eax, 10

.L3: sub  eax, 1
    cmp  eax, 0
    jne  .L3
    add  ebx, 1
```

A do-while ciklus nem lehet üres ciklus.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Alprogram

Az újra felhasználhatóság és a procedurális absztrakció eszköze

Komponensei:

- Név
- Formális paraméterlista
- Törzs
- Környezet

Fajtái:

- Eljárás
- Függvény

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Eljárás

- Tevékenységet hajt végre
- Aktiválás utasításszerűen lehet
- Befejezés a törzs végére érve vagy befejeztető utasítással
- Folytatás a hívás utáni utasítással

```
void procedure(int a, char b){
  printf(”%d %c”,a,b);
  }     // C nyelven nincs eljárás
…
procedure(3,’A’);
…
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Függvény

- Értéket határoz meg
- Aktiválás kifejezésben
- Befejezés általában befejeztető utasítás révén visszatérési érték megadásával
- Folytatás a kifejezés kiértékelésnél

```
int function(int a, char b){
   return a+b;
   }
…
x=2*function(3,'A')+1;
…
```

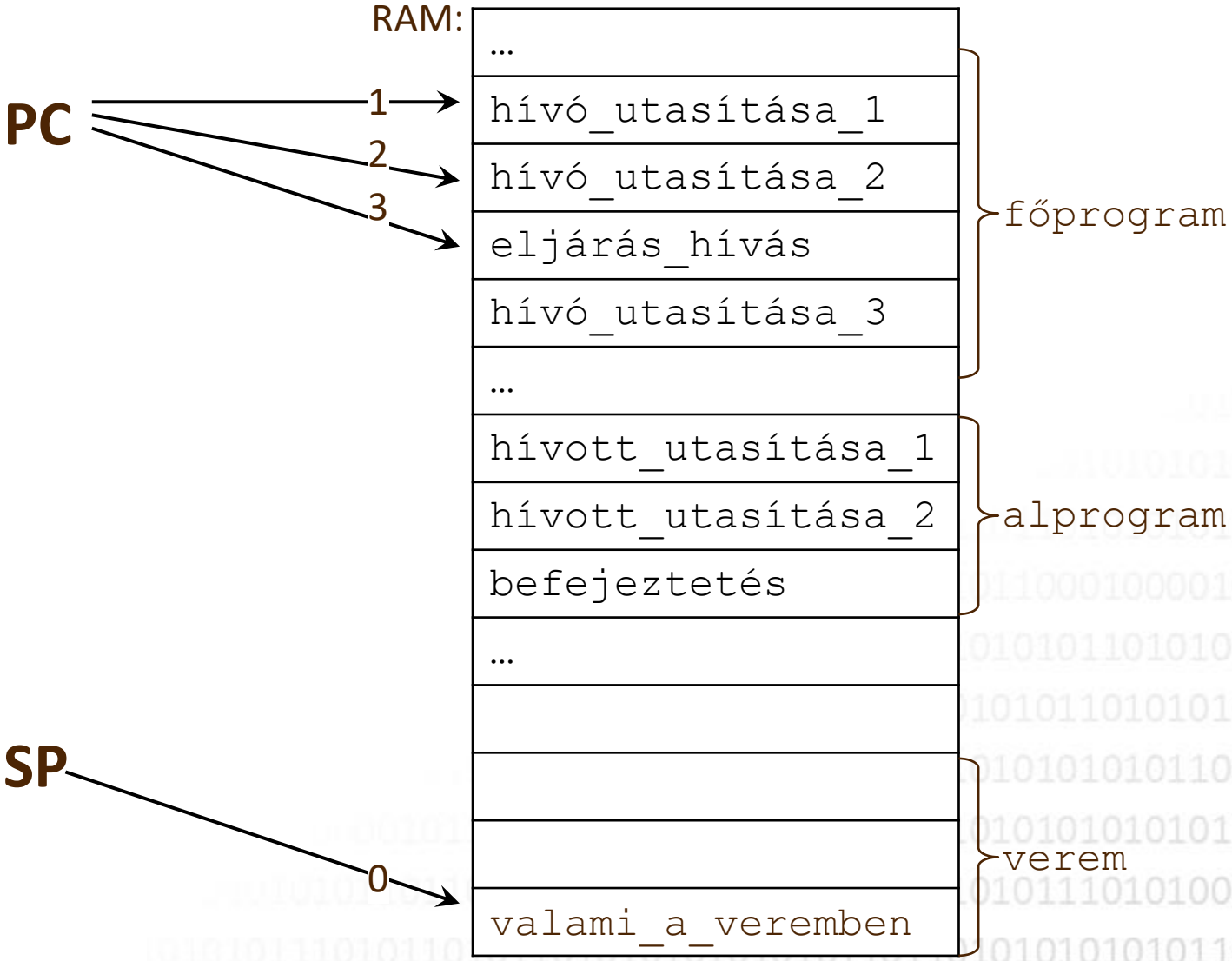
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

A verem

- Last In First Out tároló
- Tetejének a memóriacímét a verem mutató regiszter (SP) tárolja
- Push és Pop művelet
- A verem mérete korlátos
- Veremhez gyakran hozzá kell férni
- Általában gyorsítótárazott (cache)
- Adat kivétele nem fizikai törlés
- Külön memóriaszegmensben van

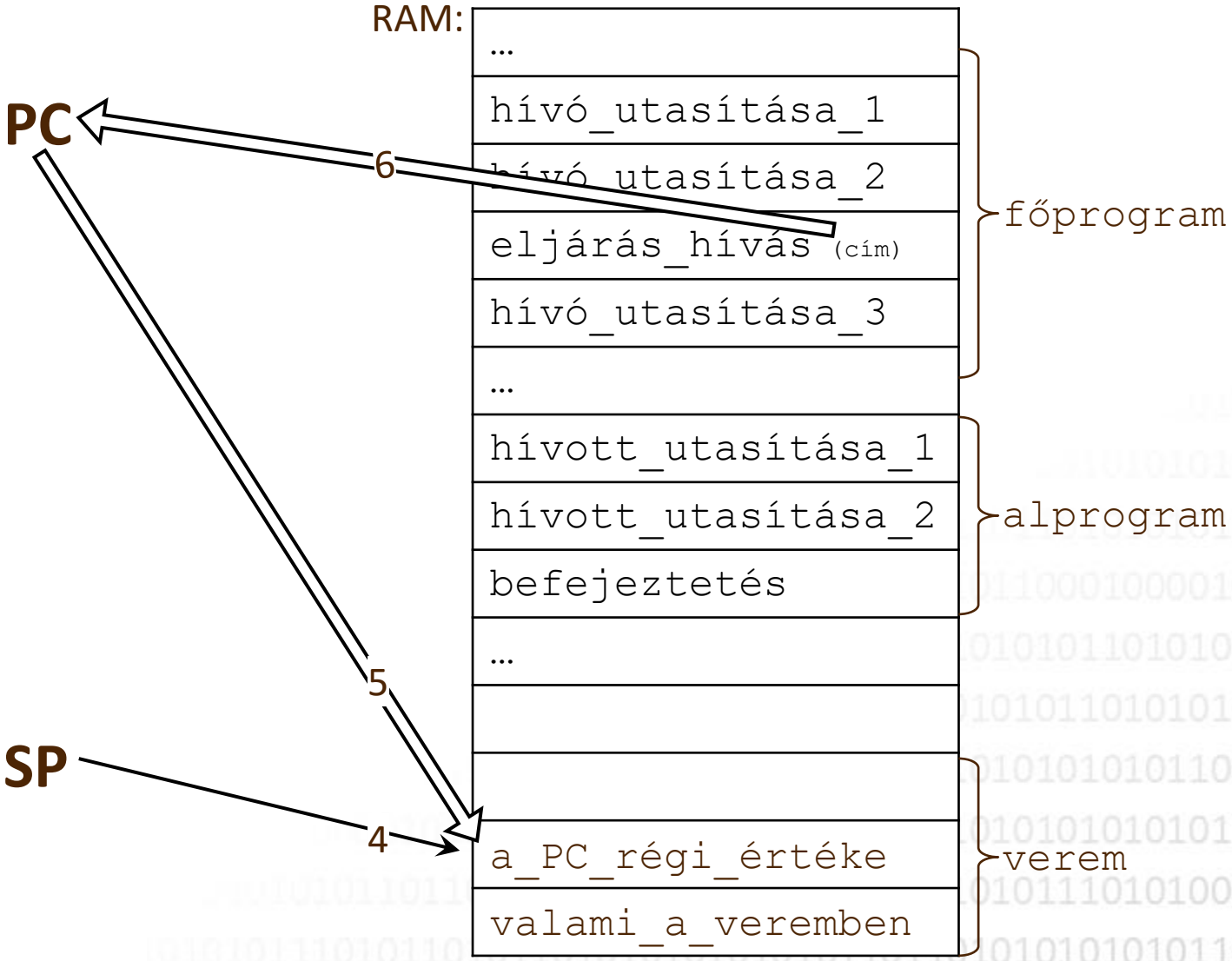
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Eljárás hívás alacsony szinten



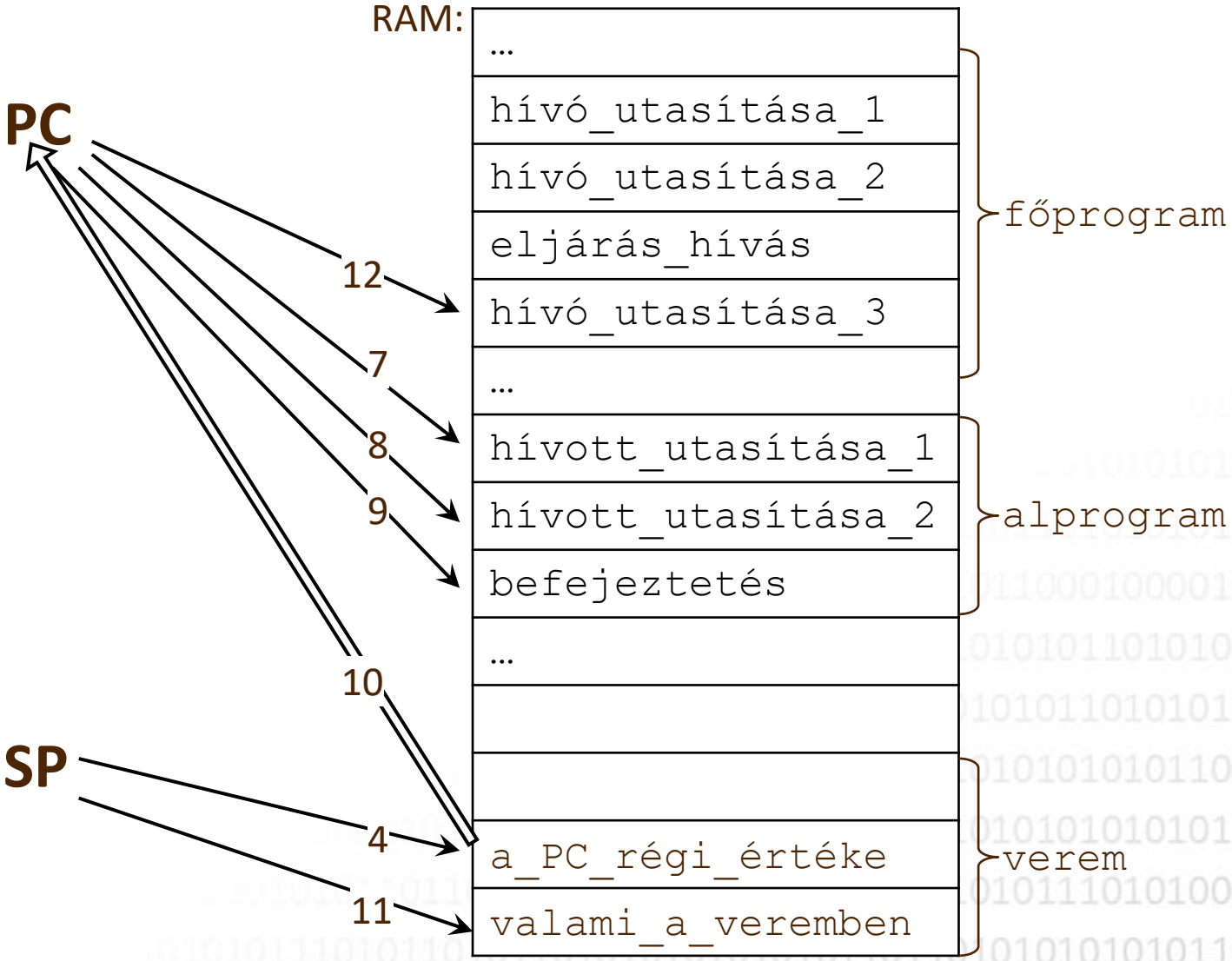
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Eljárás hívás alacsony szinten



```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Eljárás hívás alacsony szinten



```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Hívási lánc

- Az alprogramok hívhatnak újabb alprogramokat, azok továbbiakat, …
- A visszatérési címek folyamatosan a verem tetejére kerülnek (A verem mérete nő.)
- A hívási lánc dinamikusan épül fel, bomlik le
- A lánc minden tagja aktív, de csak a legutóbbi működik
- Rekurzió: egy aktív alprogram meghívása

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Lokális változók

Az programegységben deklarált nevek (változók) a programegység lokális nevei (változói)

Nem elérhetőek a programegységen kívülről

C nyelvnél (alapesetben):

- Statikus hatáskörkezelés
- Dinamikus élettartam kezelés
- Nincs automatikus kezdőérték

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Lokális változó alacsony szinten

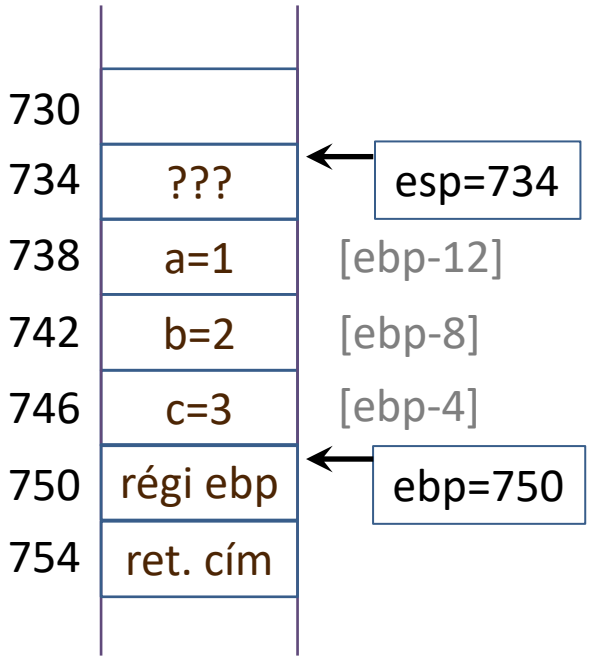
- Az alprogramba lépéskor a verem tetején a visszatérési cím található
- A verembe mentjük a bázis regiszter értékét
- A veremmutató (SP) értékét átmásoljuk a bázis regiszterbe (BP)
- Átállítva a veremmutató értékét hagyunk helyet a lokális változók számára a veremben
- A verem nem csak LIFO módon kezelhető
  A lokális változók ‚bázis relatív' címzéssel elérhetőek

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Lokális változó alacsony szinten

```
void eljar(){
    int a=1;
    int b=2;
    int c=3;
    … *
    }
```

* A RAM tartalma:



```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Lokális változó alacsony szinten

```
eljar:  push    ebp
        mov ebp, esp
        sub esp, 16
        mov DWORD PTR [ebp-12], 1
        mov DWORD PTR [ebp-8], 2
        mov DWORD PTR [ebp-4], 3
        …
        mov esp, ebp
        pop ebp
        ret
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Paraméter kiértékelés

Formális- és aktuális paraméter összerendelés

- Sorrendi kötés
- Név szerinti kötés

Számbeli egyeztetés

- Azonos paraméterszám
- Eltérő paraméterszám

Típusegyeztetés

- Típusegyezés
- Típus konverzió

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Paraméterátadás

- Érték szerinti
- Cím szerinti
- Eredmény szerinti
- Érték-eredmény szerinti
- Név szerinti
- Szöveg szerinti

Az adatmozgás iránya fontos

C nyelvben: csak érték szerinti paraméterátadás

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Érték szerinti paraméterátadás

- Formális paraméternek van címkomponense a hívott területén.
- Aktuális paraméternek van érték komponense.
- Az aktuális paraméter értéke átkerül a hívott alprogram területén lefoglalt címkomponensre.
- Az információátadás egy irányú.
- Az alprogram a saját területén dolgozik.
- A hívott alprogram nem tudja hol van a hívó.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Függvényhívás alacsony szinten

1. A hívó a verembe teszi az aktuális paramétereket fordított (!) sorrendben. (sorrendi kötés, számbeli egyeztetés)
2. A verembe bekerül a visszatérési cím (PC aktuális értéke).
3. PC megkapja a hívott alprogram kezdőcímét.
4. Szekvenciálisan lefutnak a hívott utasításai. Hívott alprogram menti a regisztereket, felhasználja a veremben lévő paramétereket . Meghatározódik a visszatérési érték.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Függvényhívás alacsony szinten

5. Visszatérési érték betétele egy meghatározott általános regiszterbe (rax vagy st0/(x)mm0).
6. Lokális változók felszabadítása.
7. Veremből visszatérési cím (hívást követő utasítás címe) átmásolása a PC-be (visszatérés).
8. Visszatérési érték a meghatározott regiszterben.
9. Paraméterek kitakarítása a veremből.
10. Végrehajtás folytatása a következő utasítással.

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Függvényhívás alacsony szinten

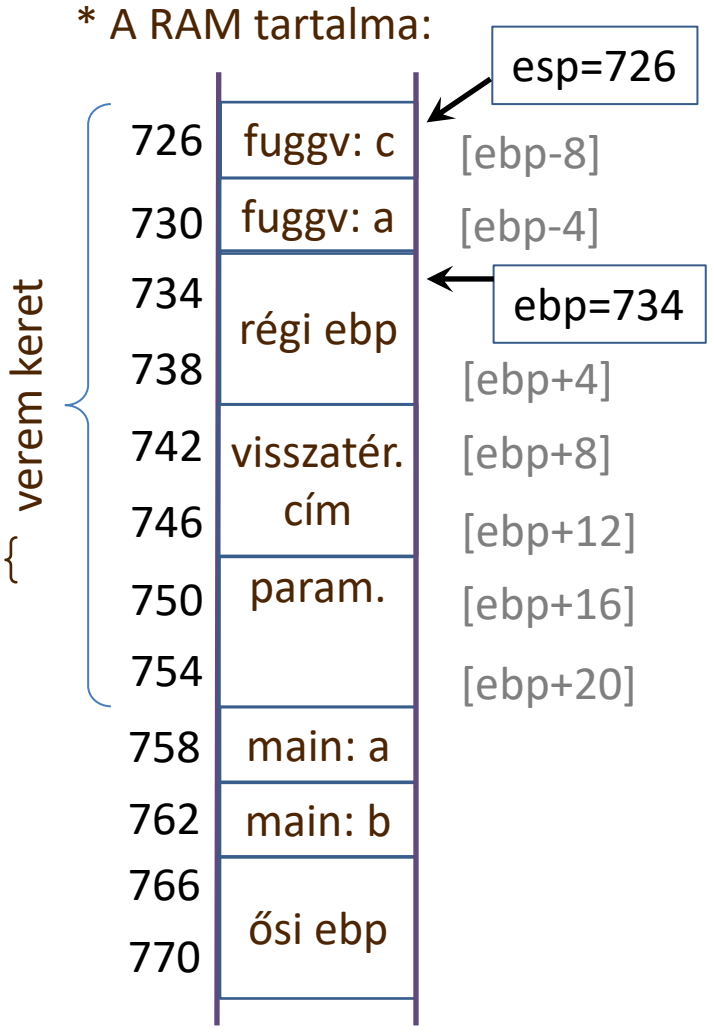
Egyes esetekben…

- A paraméterek meghatározott sorrendben regiszterekbe kerülnek, itt adódnak át. (El kell menteni a korábbi tartalmat. Egész és valós külön.)
- Float/double paraméterek száma az eax-ben.
- A visszatérési érték néha a verembe kerül a visszatérési cím alá.
- A paraméterek kitakarítása a veremből lehet a hívó vagy a hívott feladata is

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Példa C nyelven

```
int fuggv(int c){
   int a;
   a=c+1;   *
   return a;
   }
int main(int argc,
                char *argv[]){
   int a, b;
   a=argc;
   b=fuggv(a);
   return b;
   }
```



```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Példa assembly-ben #1

```
fuggv:  push    rbp
        mov     rbp, rsp
        sub     rsp, 8
        mov     eax, DWORD PTR [rbp+16]
        mov     DWORD PTR [rbp-8], eax
        mov     eax, DWORD PTR [rbp-8]
        add     eax, 1
        mov     DWORD PTR [rbp-4], eax
        mov     eax, DWORD PTR [rbp-4]
        mov     DWORD PTR [rbp+16], eax
        mov     rsp, rbp
        pop     rbp
        ret
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Példa assembly-ben #2

```
main:   push    rbp
        mov     rbp, rsp
        sub     rsp, 8
        mov     DWORD PTR [rbp-8], edi
        mov     eax, DWORD PTR [rbp-8]
        push    rax
        call    fuggv
        pop     rax
        mov     DWORD PTR [rbp-4], eax
        mov     eax, DWORD PTR [rbp-4]
        mov     rsp, rbp
        pop     rbp
        ret
```

```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Ajánlott irodalom

- Joseph Cavanagh:
  X86 Assembly Language and C Fundamentals,
  CRC Press, 2013
- Richard Blum:
  Professional Assemby Language,
  Wiley, 2005
- R. E. Bryant, D. R. O'Hallaron:
  Computer Systems – A programmer's perspective,
  Pearson, 2016