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

ASSEMBLY PROGRAMOZÁS

Dr. Varga Imre

Debreceni Egyetem

Informatikai Kar

Informatikai Rendszerek és Hálózatok Tanszék

2.0.0 verzió

2023. szeptember 3.

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

Figyelmeztetés

- A C programozási nyelv alapos ismerete és az algoritmizálás képessége nélkülözhetetlen a tárgy sikeres teljesítéséhez.
- Ezek felelevenítése nem feladata ennek a diasornak.
- Önálló felkészülés elvárt.

```
int FastSum(int *A, int N){
  int i, s=0;
  for(i=0;i<N;i++)
    s+=A[i];
  return s;
  }
```

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

Számábrázolás, adattípusok

Fixpontos számábrázolás

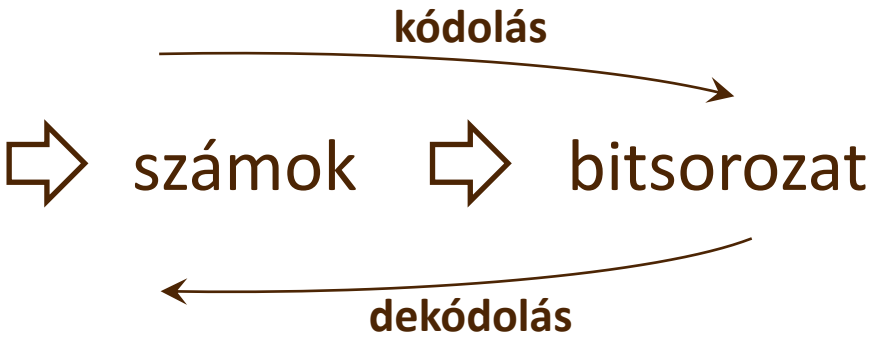
Lebegőpontos számábrázolás

Főbb adattípusok

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

Adatok

- Az információkat különböző típusú adatokként tároljuk.
- A számítógépen minden adat végül bitsorozat
  - numerikus érték
  - szöveg
  - kép
  - hang
  - videó
  - adatbázis
  - program



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

Bináris számok

- Csak két számjegy: 0 és 1
- Jelentőség: 2 állapot (igen/nem, van/nincs, stb.)
- Konverzió $69{,}375_{10} = 1000101{,}011_2$

69	*2*
34	1
17	0
8	1
4	0
2	0
1	0
0	1

2	**375**
0	750
1	500
1	**000**

64	32	16	8	4	2	1	1/2	**1/4**	**1/8**
2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

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

Hexadecimális számok

- 16 számjegy: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Szoros kapcsolat a binárissal, de rövidebb alak
- Jelölés: $1FA_{16}$, 0x1FA
- Konverzió

1*256	15*16	10*1	$=506_{10}$
16^2	16^1	16^0	
1	**F**	**A**	

0	0	0	1	1	1	1	1	1	0	1	0	
2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
2048	1024	512	256	128	64	32	16	8	4	2	1	$=506_{10}$

0x1FA = 0b 1 1111 1010 = 0772 = 506

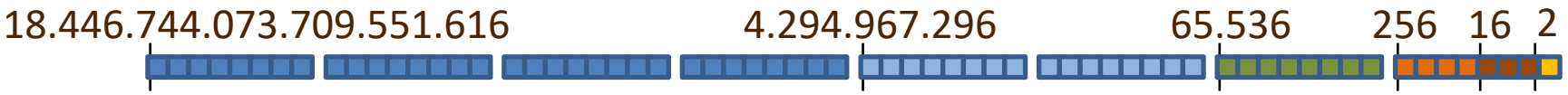
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Adatszervezés

Adategységek

- Bit (**bi**nary digi**t**): **két állapot** (0, 1)
- Nibble (fél bájt): 4 bit
- Bájt : 8 bit
- Szó: (általában) 16 bit
- Dupla szó: 32 bit
- Négyszeres szó: 64 bit

Eggyel több bit, kétszer annyi tárolható érték



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

Adattípus

Különböző esetekben eltérőek az adatokkal kapcsolatos elvárások, ezért **típus**okat vezetünk be

Típusok megadása

- Reprezentáció: hogyan kódoljuk
- Tartomány: milyen értékek lehetnek
- Műveletek: mire használhatóak

Főbb adattípusok

- egész, valós, karakter, mutató, logikai, tömb, rekord, sztring

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

Egész típus

Ábrázolás

- „Előjel-és-nagyság” (nem praktikus)
- **Fixpontos számábrázolás** (2, 4 vagy 8 bájt)
  - ➥ Előjeles vagy előjel nélküli
- Binárisan kódolt decimális (BCD)
  - ➥ Pakolt vagy pakolatlan

Megadás különböző módokon

- Pl. C nyelven:
  30=30u=30l=036=0x1E=0b11110*

*ISO C99

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

Fixpontos számábrázolás

- Egész értékek reprezentálása
- **Előjel nélküli** eset (csak nem negatív szám)
  - ábrázolandó szám kettes számrendszerben
  - adott számú biten
- Példa (1 bájt esetén)

  $41_{10} \rightarrow 101001_2 \rightarrow$

0	0	1	0	1	0	0	1

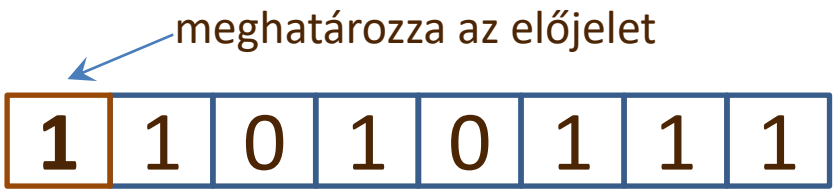
- N biten ábrázolható tartomány
  - legkisebb szám: 0
  - legnagyobb szám: 2^N-1

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

Fixpontos számábrázolás

- **Előjeles** eset (nem „előjel-nagyság” logika)
  - Nem negatív érték esetén: előjel nélküli logika
  - Negatív érték esetén: **kettes komplemens**
    - Abszolút érték bitjeinek invertálása (egyes komplemens)
    - A kapott érték növelése 1-gyel
- Példa (1 bájt esetén)

  $-41_{10} \rightarrow -101001_2 \rightarrow$



- N biten ábrázolható tartomány
  - Legkisebb szám: $-(2^{N-1})$
  - Legnagyobb szám: $2^{N-1}-1$

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

A C nyelv egész típusai

Fixpontos számábrázolás

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

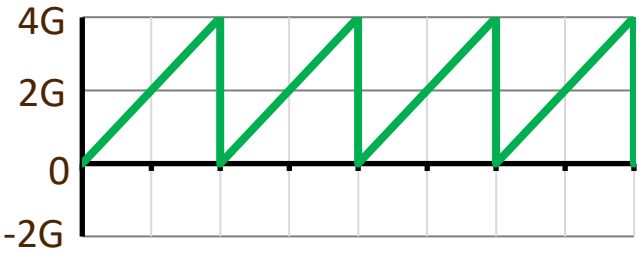
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Aritmetikai túlcsordulás

Két féle hibalehetőség fixpontos műveleteknél

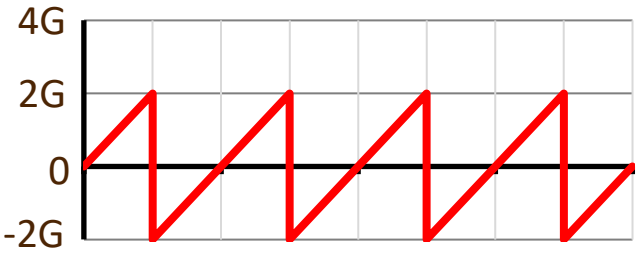
- Előjel nélkül: több bit lenne szükséges (**carry**)

```
  01011010     90
+11110001   +241
101001011     75 ≠331 >255
```



- Előjeles: nem várt előjelváltás (**overflow**)

```
  01011010     90
+01110001   +113
 11001011     -53 ≠203 > 127
```



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

Kiterjesztés

Néha szükség van egy adatot a korábbinál több biten ábrázolni úgy, hogy az értéke ne változzon

- **Nulla kiterjesztés**: kitöltés nullával
  - Rövid előjel nélküli értékből hosszú előjelnélküli

    8 bit: 01101001 → 16 bit: 0000000001101001

    8 bit: 11010010 → 16 bit: 0000000011010010

    ```
    unsigned short x=123; unsigned int y=x;
    ```
- **Előjel tartó kiterjesztés**: kitöltés előjel bittel
  - Rövid előjeles értékből hosszú előjeles

    8 bit: 01101001 → 16 bit: 0000000001101001

    8 bit: 11010010 → 16 bit: 1111111111010010

    ```
    short int q=-123; long int z=q;
    ```

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

Bájtsorrend

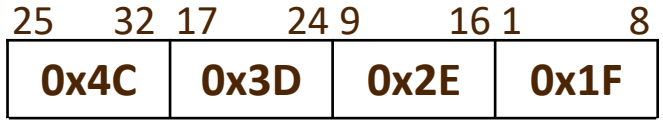
Több bájtos adatok esetén a bájtok sorrendje

- **Little-endian**: (gazdagép bájtsorrend)
  - ➥ Legkisebb helyi értékű bájt (LSB) van elől
- **Big-endian**: (hálózati bájtsorrend)
  - ➥ Legnagyobb helyi értékű bájt (MSB) van elől

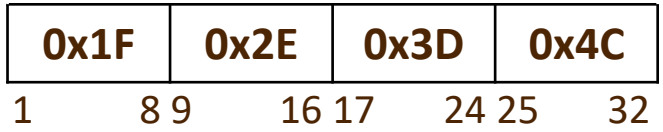
Példa:

523124044 = 0b11111 00101110 00111101 01001100 = **0x1F 2E 3D 4C**

Little-endian:



Big-endian:



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

Valós típus

- Ábrázolás: **lebegő pontos**
- Méret: 4, 8, 10 bájt
- Műveletek
  - Operátor túlterhelés
    Pl. a + operátor lehet egész-egész vagy valós-valós
  - Egyes műveletek nem használhatóak valós típussal
    Pl.: nincs maradékos osztás, léptetés, XOR
- Megadás különböző formában
  - Pl. C nyelven:
    0.25, 0.25f, 0.25L, .25, +0.25, 25e-2, 0.025e1

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

Lebegő pontos számábrázolás

- Valós számok ábrázolása
- Alapja a normalizálás: $123{,}45 = 1{,}2345 \cdot 10^2$
- Minden (2 számrendszerbeli) szám felírható így:

$$(-1)^S \cdot M \cdot A^K$$

  - ahol a **mantissza** (M) „1.H” alakú, azaz $1_2 \leq M < 10_2$
  - **karakterisztika** (K) pozitív és negatív is lehet
- Nem tároljuk: alap (A=2), rejtett bit, K előjele
  - E=K+B adott számú biten nem negatív (B: nulla pont)
- Tárolandó: S, H, E $(-1)^S \cdot 1.H \cdot A^{E-B}$

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

Lebegő pontos szabvány

IEEE 754 szabvány (ISO/IEC/IEEE 60559:2011)

- Előjel
  - Pozitív: 0; Negatív: 1
- Formátumok

	méret	S hossz	E hossz	H hossz	B
Egyszeres pontosság	32 bit	1 bit	8 bit	23 bit	127
Dupla pontosság	64 bit	1 bit	11 bit	52 bit	1023
Kiterjesztett pontosság	80 bit	1 bit	15 bit	63(+1) bit	16383
Fél pontosság	*16 bit*	*1 bit*	*5 bit*	*10 bit*	*15*

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

Lebegő pontos számábrázolás

Példa: -12.8125 egyszeres pontossággal

$-12.8125_{10} = -1100.1101_2 = -1^1 \cdot 1.1001101 \cdot 10^{11}$

$(-1)^S \cdot M \cdot A^K$

$(-1)^S \cdot 1.H \cdot 2^{E+127}$

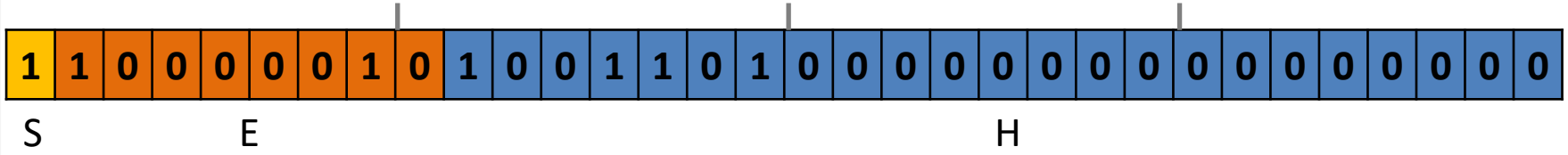
Előjel: S = **1**

Karakterisztika: $K = 3_{10} = 11_2$

Mantissza: M = 1.1001101

Hasznos bitek: H = **10011010000000000000000**…

Eltolt karakterisztika: $E = 3_{10}+127_{10} = \mathbf{10000010_2}$



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

Kerekítési hiba

- Nem mindig pontos az ábrázolás
- Ok: hasznos bitek tárolása véges számú biten
- Túl sok hasznos bit esetén kerekítés szükséges
  - Ha az utolsó tárolható hasznos bit után ’0’ áll, akkor a további biteket levágjuk
  - Különben felfelé kerekítünk
- Példa: 0.3 ábrázolása egyszeres pontossággal

  hasznos bitek: 00110011001100110011001100110…

  tárolt bitek: 00110011001100110011010

  0.3 → 0.300000011920929

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

Kerekítési hiba

- Valós számok:
  ‚folytonos' halmaz, végtelen sok elemmel
- Lebegőpontos számok:
  diszkrét halmaz, véges sok elemmel
- Az ábrázolható számok sűrűsége nem állandó
  4 bájton tárolható értékek például:
  `1.0000000000; 1.0000001192; 1.0000002384; 1.0000003576`
  `1000000000.0; 1000000064.0; 1000000128.0; 1000000192.0`
- A tárolt értékek csak néhány számjegyig pontosak
  π = 3,141592653… ≠ 3,141592741… (float)

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

A C nyelv valós típusai

- Lebegő pontos számábrázolás

típus	méret	tartomány		
		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

*C99

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

Speciális lebegőpontos értékek

- +0.0 és -0.0
  előjel bit ’0’ (+) vagy ’1’ (-), minden más bit ’0’
  Pl.: 0.0/-1.0, -1.0*0.0
- ±végtelen (inf, 1.#INF00)
  karakterisztika csupa ’1’, mantissza csupa ’0’
  Pl.: 1.0/0.0, inf+1.0, túl nagy érték
- „nem szám” (NaN, 1.#IND00)
  karakterisztika csupa ’1’, mantissza nem csupa ’0’
  Pl.: 0.0/0.0, inf-inf, 0.0*inf, NaN+1.0
- denormalizált szám (subnormal)
  karakterisztika csupa ’0’, mantissza nem csupa ’0’

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

Számolás lebegő pontos számokkal

- Összeadás: fixpontostól nagyon eltérő módszer
- Példa: 7,5 + 0,625

7,5 = 01000000 11110000 00000000 00000000

0,625 = 00111111 00100000 00000000 00000000

10000001 > 3 01111110

```
 11110000...
+   10100...
-----------
100000100...
```

1+10000001 = 10000010

01000001 000000100 00000000 00000000

$(-1)^0 * 1{,}000001 * 10^{11} = \mathbf{8{,}125}$

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

Példa: bitsorozat értelmezés

Mit jelenthet az alábbi big-endian bitsorozat?

`11101001 10110000 10100101 01000110`

- 32 bites előjeles egész: -374 299 322
- 32 bites előjel nélküli egész: 3 920 667 974
- 16 bites előjeles egészek: -5 712; -23 226
- 16 bites előjel nélküli egészek: 59 824; 42 310
- lebegőpontos: -266939282492416730000000000.0
- „Latin 1” szöveg: é°¥F
- „Latin 2” szöveg: é°ĽF
- Unicode szöveg UTF8 kódolással: 鰥F
- Pakolt BCD: ?9?0?546 (érvénytelen) vagy +9-0+546

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

Tömb típus

- Azonos típusú elemek **folytonosan** tárolva
- Minden elem az adott reprezentációval
- A tömb neve
  - C nyelven: az első elemre mutató nevesített konstans
  - Egyes nyelveken: az összes elemre hivatkozás
- Egydimenziós tömb *i.* elemének a címe (A_i)

$$A_i = A_1 + (i - 1)E$$

ahol E egy adatelem mérete és A_1 a tömb kezdőcíme.
Ezért kezdődik a tömbindexelés gyakran 0-val.

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

Assembly programozás

Assembly nyelv

Utasítás készlet

Címzési módok

Gépi kód

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

Assembly programozás

- Alacsony absztrakciós szint, elemi utaítások
- Hardverismeret szükség
  - ↳ Platformfüggő
- Programozói szabadság
- Feladatra optimalizálható kód
- Nagyobb program teljesítmény
- Nehezebben átlátható kód
- PC, mikrokontroller

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

Utasítás szerkezet

- 4 címes utasítás

Műveleti kód	1. Operandus cím	2. Operandus cím	Eredmény cím	Következő utasítás cím

- 3 címes utasítás

Műveleti kód	1. Operandus cím	2. Operandus cím	Eredmény cím

- 2 címes utasítás

Műveleti kód	1. Operandus + eredmény cím	2. Operandus cím

- 1 címes utasítás

Műveleti kód	2. Operandus cím

- 0 címes utasítás (pl. verem-alapú architektúrák)

Műveleti kód

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

Program és utasítás felépítés

Forrás fájl

- utasítás_1
- utasítás_2
- utasítás_3
- utasítás_4
- utasítás_5
- …

Címke	Művelet	Operandus(ok)	Megjegyzés
.L1:	mov	eax, 10	# ten into eax

Címke: Azonosító, általában kettősponttal zárul

Művelet: Az elvégzendő művelet **mnemonic**-ja

Operandus(ok): Adat(ok) vagy adat(ok)ra hivatkozás(ok)

Megjegyzés: Sor végéig a fordító figyelmen kívül hagyja

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

Utasításkészlet architektúra

ISA (Instruction Set Architecture)

A számítógép programozáshoz kötődő részletei

- Regiszter* készlet
- Bitszélesség
- Gépi utasítások
- Címzési módok
- Memória architektúra
- Megszakítás kezelés

*Kis kapacitású, gyors elérésű tároló a processzorban.

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

Regiszterkészlet

- Regiszterek: Kis kapacitású, gyors elérésű tárolók a processzorban, amelyek például műveletek operandusait eredményeit tárolják.
- Fontos:
  méret, darabszám, elnevezés, szerep, stb.
- Néhány példa:
  - ➥Az x86 architektúra főbb 32 bites regiszterei:
    `eax, ebx, ecx, edx, esp, ebp, edi, esi, eip, eflags`
  - ➥Az ARM architektúra főbb 32 bites regiszterei:
    `r0, r1, r2, r3, … r12, SP, LR, PC, CPSR`

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

Utasítás típusok

- Adatmozgató utasítások
- (Egész) aritmetikai utasítások
- Bitenkénti utasítások
- Vezérlésátadó utasítások
- Sztringkezelő utasítások
- BCD és valós aritmetikai utasítások
- Fordítási direktívák
- Egyéb utasítások

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

Utasítás típusok

Adatmozgató utasítások

- Regiszter-regiszter (mov, xchg)
- Regiszter-memória (mov)
- Regiszter-I/O port (in, out)
- Regiszter-verem (push, pop)
- Kiterjesztés (bitszélesség fontos) (cbw, cwd, cdqe)
- Státusz bit állítás (sti, cli)

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

Utasítás típusok

Aritmetikai utasítások

- Összeadás (átvitel nélkül/átvitellel) (add, adc)
- Kivonás (átvitel nélkül/átvitellel) (sub, sbb)
- Inkrementálás, dekrementálás (inc, dec)
- Kettes komplemens (neg)
- Előjeles/előjel nélküli szorzás (imul, mul)
- Előjeles/előjel nélküli osztás (idiv, div)
- Összehasonlítás (cmp)

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

Utasítás típusok

Bitenkénti logikai utasítások

- ÉS művelet (and)
- VAGY művelet (or)
- KIZÁRÓ VAGY művelet (xor)
- Egyes komplemens (not)
- Logikai/előjel nélküli balra léptetés (shl)
- Logikai/előjel nélküli jobbra léptetés (shr)
- Aritmetikai/előjeles jobbra léptetés (sar)
- Jobbra/balra forgatás (ror, rol)
- Jobbra/balra forgatás carry-n keresztül (rcr, rcl)

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

Utasítás típusok

Vezérlésátadó utasítások

- Feltétel nélküli ugrás (jmp)
- Feltételes ugrás (je, jne, jg, jge, jl, jle, ja, jb, jc, jo, …)
- Ciklusszervező (loop, loopz, loopnz, …)
- Hívó utasítás (call)
- Szubrutinból visszatérés (ret)
- Szoftveres megszakítás (int)
- Megszakításból visszatérés (iret)

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

Utasítás típusok

Stringkezelő (bájtsorozat kezelő) utasítások

- Sztring(elem) mozgató (movs, movsw, movsd)
- Sztring(elem) összehasonlító (cmps)
- Keresés sztring(elem)ben (scas)

Egyéb végrehajtható utasítások

- Lebegőpontos (fld, fst, fadd, fsqrt, …)
- Rendszervezérlő (hlt)
- Üres utasítás (nop)
- Információ (cpuid)

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

Utasítás típusok

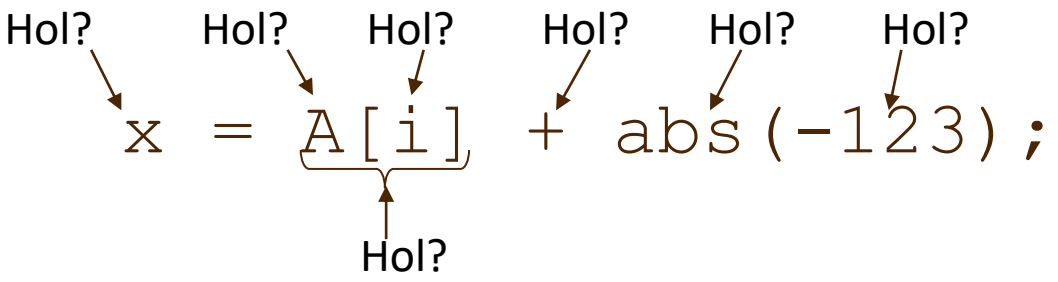
Fordítási direktívák: gépi kód nincs mögöttük, a fordítás menetét befolyásolják

- Helyfoglalás (.byte, .comm, .zero)
- Láthatóság megadás (.globl, .local)
- Szintaxis megadás (.intel_syntax)
- Szegmens definiálás (.text, .data, .bss)
- Helyettesítő szimbólum létrehozás (.set)
- …

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

Memória elérés szükségessége

- Műveletvégzés a CPU-ban történik
- Az utasítások és az adatok a RAM-ban vannak
- A CPU és a RAM közötti átvitelhez kell cím
  - A processzornak tudnia kell, mi hol van a memóriában
  - A címek vagy regiszterben vannak tárolva vagy számítás eredményei



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

Címzési módok

Adat (vagy utasítás) hogyan érhető el.

- Rejtett (implicit, implied)
- Közvetlen adat megadás (immediate)
- Közvetlen cím (direct, absolute)
- Regiszteres cím (register direct)
- Közvetett cím (indirekt)
- Regiszter indirekt
- Indexelt cím
- Regiszter relatív cím
- …

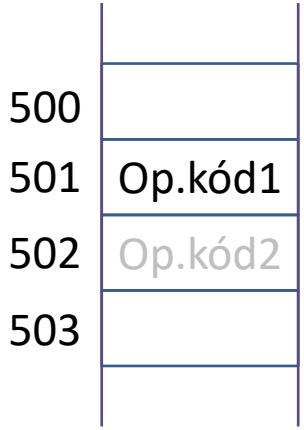
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Rejtett címzés

- Nincs igazi cím megadás
- Pl. ha nincs operandus

RAM:

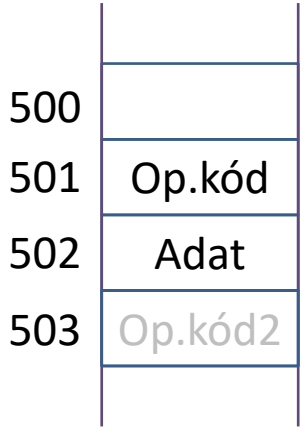


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Közvetlen adat megadás

- Műveleti kód után közvetlenül található adat
- Ez lesz az operandus
- Konstans

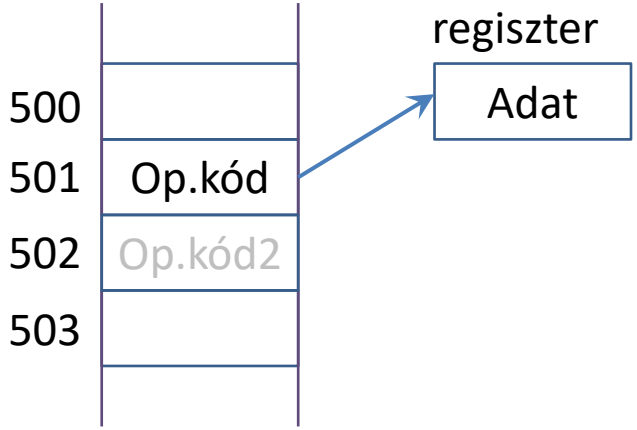


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Regiszteres címzés

- A műveleti kód hivatkozik egy regiszterre
- A regiszterben található az operandus értéke

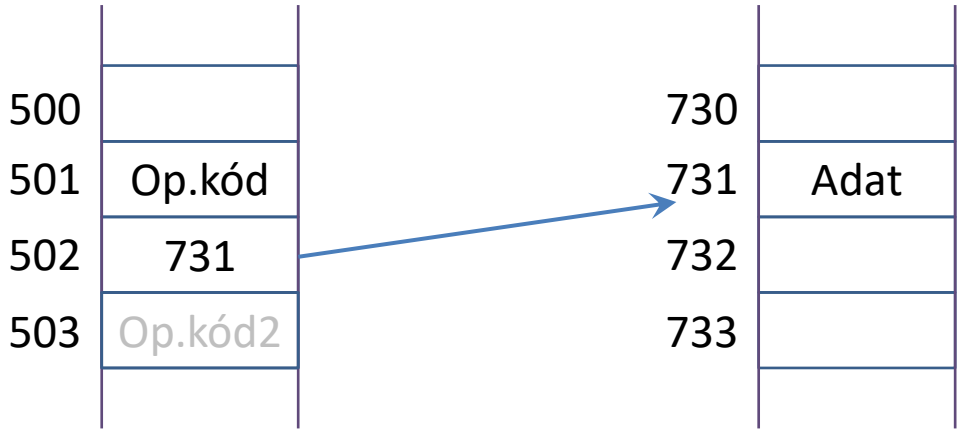


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Közvetlen cím

- Műveleti kód után található egy memóriacím
- Itt helyezkedik el az operandus

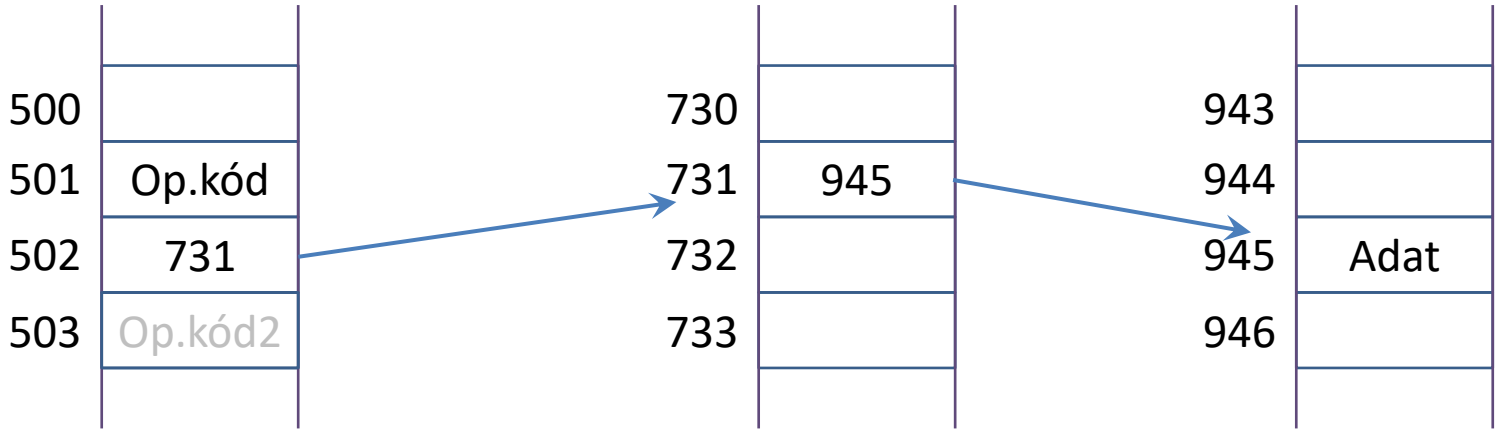


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Közvetett cím

- A műveleti kód után található egy cím
- Ezen a címen található az operandus igazi címe

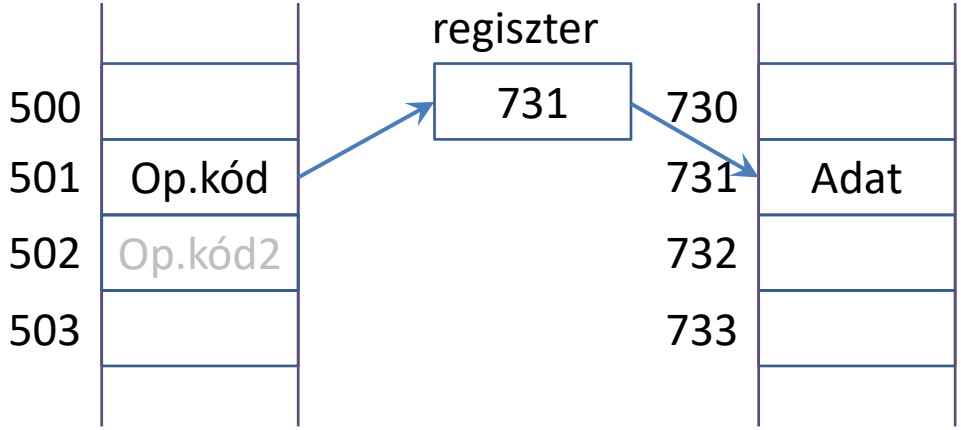


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Regiszter indirekt

- A műveleti kód hivatkozik egy regiszterre
- A regiszterben található címen helyezkedik el az operandus

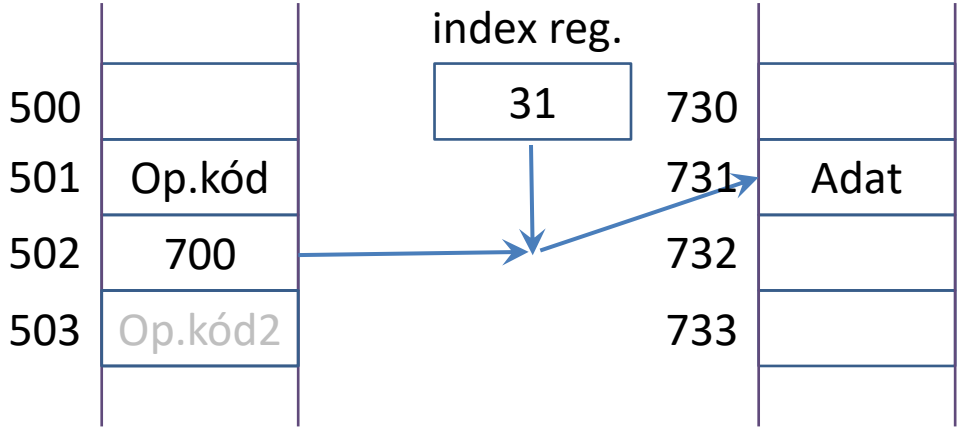


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Indexelt

- A műveleti kód után található egy cím, ehhez hozzáadva az index regiszterben található értéket megkapjuk az operandus memóriacímét

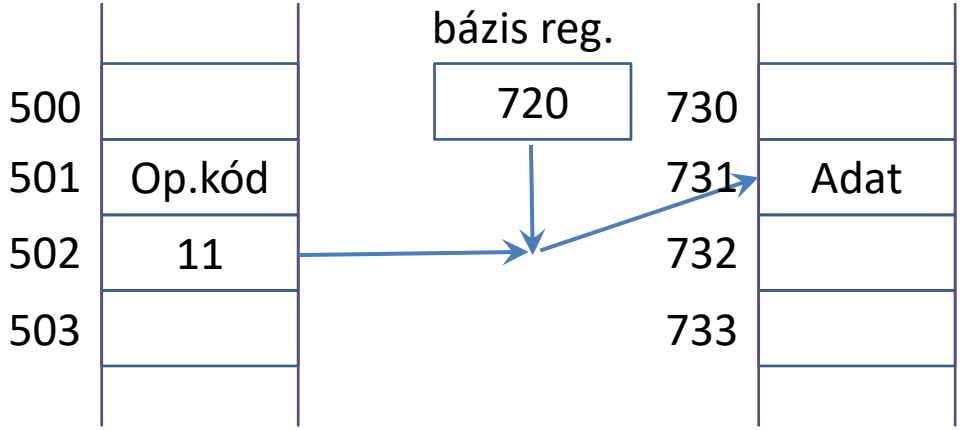


```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Címzési módok

Regiszter relatív

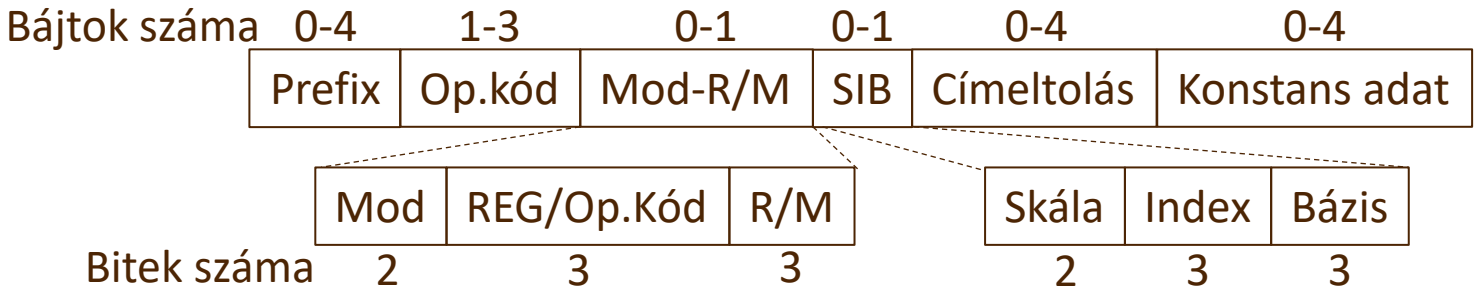
- A műveleti kód után található egy eltolási érték, ehhez hozzáadva a bázis regiszterben található kezdőcímet megkapjuk az operandus címét



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

Gépi kód

- Az egyetlen nyelv, amit a processzor megért
- Bináris formátum
- Művelet, címzési mód, operandus méret, operandus típus (reg./mem./konst.), stb.
- Processzoronként változó lehet
- Könnyen dekódolható legyen a hardver számára
- Formátum x86 architektúra esetén:



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

Gépi kód példa

C nyelvű kód

```
y=5;
z=6;
x=z+y*7;
```

Assembly kód (x86)

```
mov   DWORD PTR [ebp-12], 5
mov   DWORD PTR [ebp-8], 6
mov   ebx, DWORD PTR [ebp-12]
mov   eax, ebx
sal   eax, 3
sub   eax, ebx
add   eax, DWORD PTR [ebp-8]
mov   DWORD PTR [ebp-16], eax
```

Gépi kód

c7	45	f4	05	00	00	00	c7	45	f8	06	00	00	00	
8b	5d	f4	89	d8	c1	e0	03	**29**	**d8**	03	45	f8		
89	45	f0												

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

A számítógép vázlatos felépítés

Architektúra szintek

Processzor

Buszrendszer

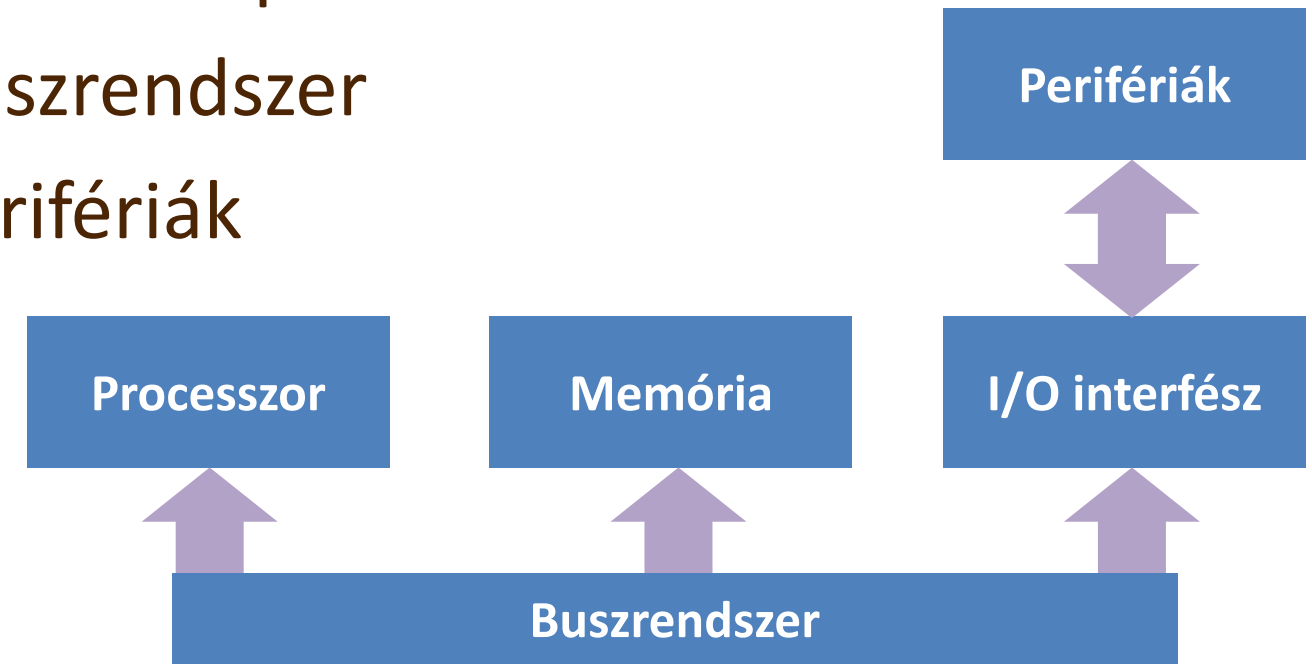
Memória

Perifériák

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

A számítógép alapvető felépítése

- Processzor
- Memória
- Input-Output interfész
- Buszrendszer
- Perifériák



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

Processzor

- **Központi feldolgozó egység** (CPU)
- A számítógép „agya”
  - Vezérlés
  - Utasítás végrehajtás

AMD Sempron processzor

- (Mikro)processzor integrált áramköröket tartalmaz
- Műveletek és az egységek szinkronizálásához órajelet használ


Intel Core i3 processzor

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

Memória

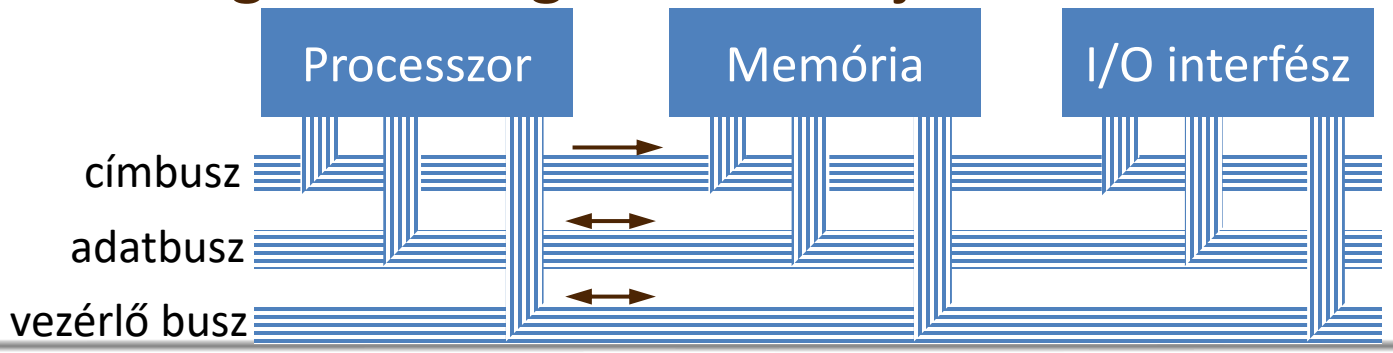
- Operatív tár
  - Aktuális műveletekhez szükséges „rövidtávú” memória
  - Elsődleges tároló, nem háttértár
- Címezhető adattároló „rekeszek”
- Adatok és utasítások tárolása
- Elektronikus működés
  - Integrált áramkör

DDR RAM

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

Buszrendszer

- Busz (sín): vezetékek egy csoportja, amelyeken digitális jelet lehet továbbítani
- Buszrendszer logikai részei
  - Címbusz: a címek bitsorozatának átvitelét biztosítja
  - Adatbusz: adatok átviteléért felelős
  - Vezérlő busz: részegységek működésének összehangolását segítő vezérlő jelek továbbítása



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

A processzor felépítése és működése

CPU

Regiszterek

Fetch-execute ciklus

RISC / CISC processzorok

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

Processzor

- Központi Feldolgozó Egység
  - Central Processing Unit (CPU)
- Részei:
  - Vezérlő egység (CU)
  - Aritmetikai és logikai egység (ALU)

  } Végrehajtó egység (EU)
  - Címző egység (AU) és Busz illesztő egység (BIU)
  - Regiszterek
  - Belső buszrendszer
  - Belső gyorsítótár
  - Egyéb (pl. órajel generátor)

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

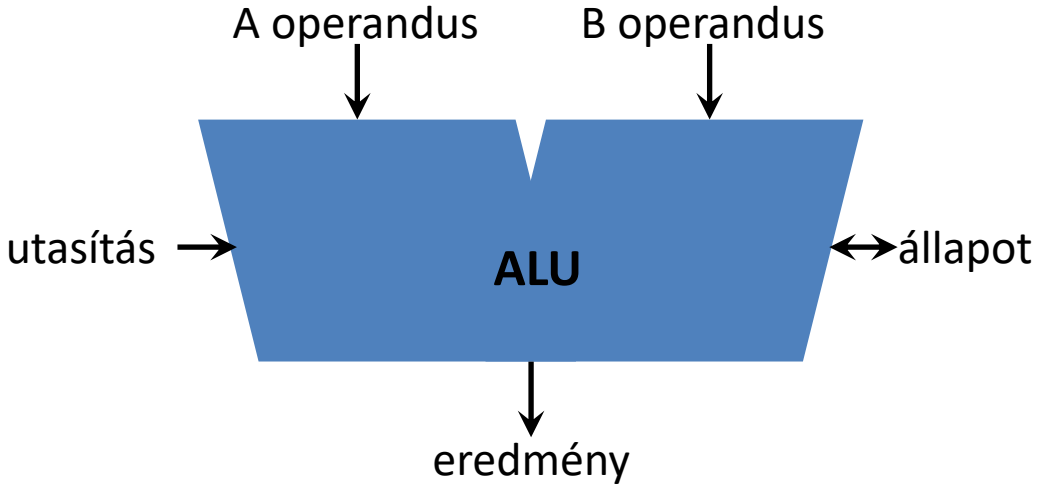
Regiszterek

- Kis méretű tároló áramkör
  - Mérete általában a busz szélességével egyenlő
  - Általában 8-512 bit tárolására alkalmas
- Gyors hozzáférésű (elérési idő < 1ns)
- Darabszámát a processzor határozza meg (10-100)
- Regisztertömb elemi
- Van nevük (néha átnevezhetőek)
- 3 kategória:
  - Rendszer- , általános célú- és speciális célú regiszter

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

Aritmetikai és logikai egység

- Számítási műveleteket végez
- Tartalmaz: fixpontos összeadót, komplemens képzőt, léptető regisztereket, bitenkénti logikai művelet végző, stb. áramköröket



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

Vezérlő egység

- Az IR-ben lévő érték alapján irányítja, szabályozza a többi egység (pl. ALU) működését
- Fontos regiszterei: IR, PC, SR
- A vezérlés történhet
  - Huzalozott logikával (közvetlenül)
    Minden utasításhoz létrehozott bonyolult digitális áramkörök segítségével
  - Mikroprogramozott (közvetett) módon
    Minden műveleti kód elindít egy apró (ROM-ban tárolt) mikroprogramot

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

Címző/címgeneráló egység

Addressing Unit (AU), Address Generation Unit (AGU)

- Az utasítások sokféle címzési mód segítségével határozhatják meg hol található az operandus
- Az AU az operandus címét állítja elő és helyezi el az MAR-ban
- Az utasításban található címet képezi le „fizikai” memóriacímre
- Tárolóvédelmi hibák felismerése
- Kapcsolat: BIU

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

A CPU működése

- Egy elemi művelet sorozat ismételgetése
  - Fetch-execute ciklus
- Órajel szinkronizált
- CU vezényel
- Végtelen, monoton, gépies ismétlés során
  - Adatmozgatás
  - Műveletvégzés
- Fontos a regiszterek (PC, IR, SR, ACC, MAR, MDR, stb.) tartalma és annak változása

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

Fetch-Execute ciklus

1. Utasítás beolvasása (IF)
   A PC-ben található érték megmondja, hol található a memóriában a következő utasítás. Ezt be kell olvasni az IR-be. PC aktualizálása.
2. Dekódolás (ID)
   A beolvasott műveleti kódot értelmezni kell. Milyen művelet? Milyen adatokon? Eredmény hová kerül? (Melyek a használt regiszterek?)
   Utasításkészlet architektúra (ISA) definiálja
   Lehet huzalozott vagy mikroprogramozott

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

Fetch-Execute ciklus

3. Művelet végrehajtása vagy címszámítás (EX)
   Az ALU dolgozik, az eredmény belső átmenti regiszterbe kerül, Load/Store utasítás esetén kiszámolódik a memóriacím
4. Adatmemória hozzáférés (MEM)
   Load/Store utasítás esetén adatmemória olvasás/írás a megfelelő címen
5. Visszaírás (WB)
   A cél regiszterbe kerül a művelet eredménye vagy a beolvasott adat

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

Az x86 architektúra

Processzor felépítés

Regiszterkészlet

Memóriakezelés

Assembly nyelv

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

x86 architektúra kezdetek

- Az Intel 1976-78 között fejlesztette ki az „új” **Intel 8086** névre hallgató CISC processzorát
- Később ezt fokozatosan tovább fejlesztették
  - Intel 80186 (1982)
  - Intel 80286 (1982)
  - Intel 80386 (1986)
  - Intel 80486 (1989)
  - …, a folyamat máig tart
  - Az új processzorok visszafelé kompatibilisek
- Ma a processzorcsaládra **x86** néven hivatkozunk
- Főbb gyártók: Intel, AMD, VIA

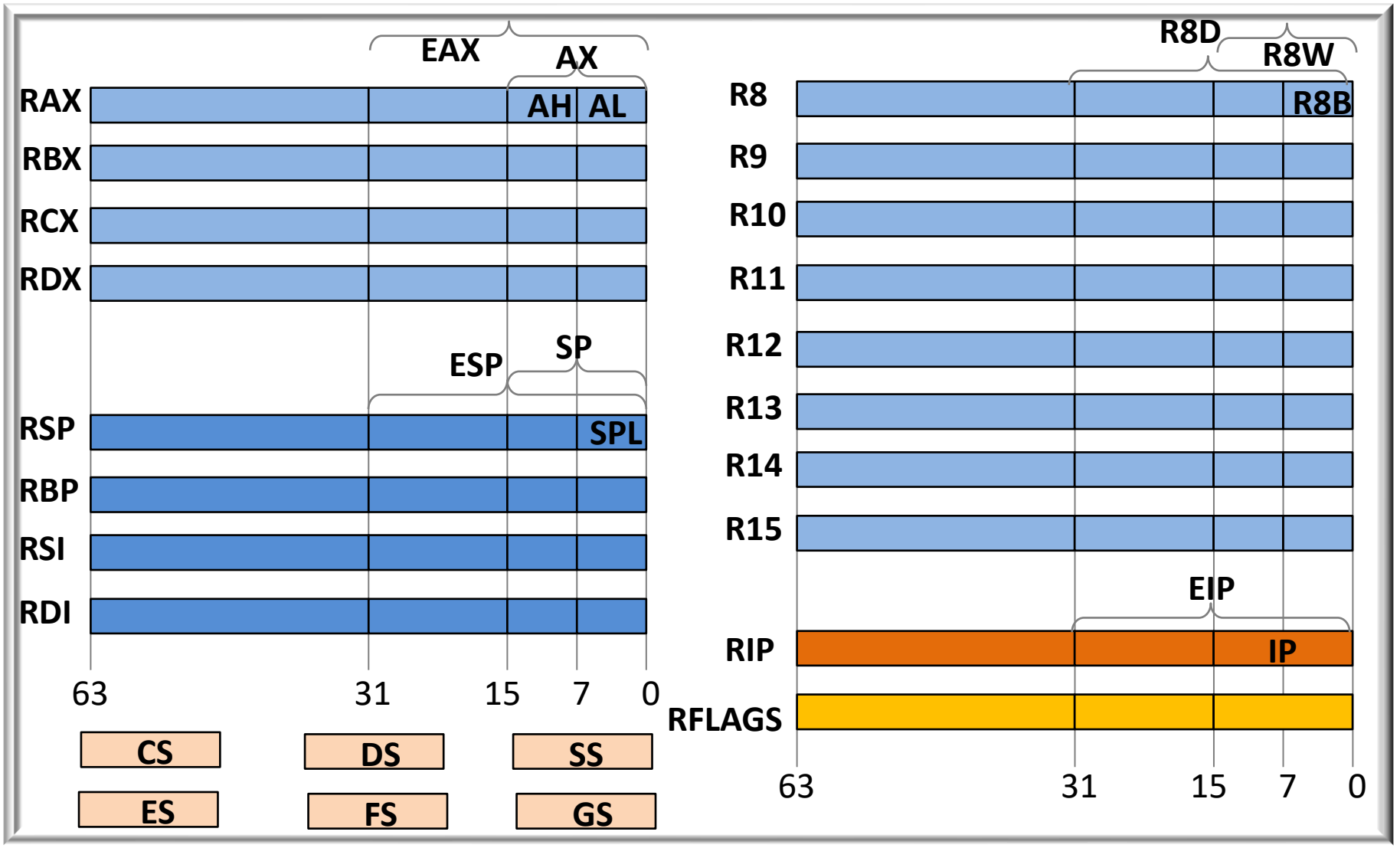
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

Memória szegmentáció

- A memória logikai részekre van osztva
  - Kód szegmens
  - Adat szegmens
  - Verem szegmens
- A címzést külön regiszterek segítik (CS, DS, SS, ES)
- Címzés: szegmens kezdőcím + eltolás
- Memóriakezelés
  - Valós-, védett-, virtuális-, hosszú mód

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

x86-64 regiszter készlet



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

x86 regiszterkészlet

Fő regiszterek (általános célú regiszterek, GPR)

- EAX
  - Elsődleges munka regiszter, szorzás, osztás, visszatérési érték
- EBX
  - Munka regiszter, bázis mutató DS-ben
- ECX
  - Munka regiszter, (ciklus)számláló, 4. paraméter
- EDX
  - Munka regiszter, input/output, szorzás, osztás, 3. paraméter

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

x86 regiszterkészlet

Index regiszterek (általános célú regiszterek, GPR)

- ESI (forrás index regiszter)
  - Sztring műveletek forrás indexe, DS regiszterrel, 2. paraméter
- EDI (cél index regiszter)
  - Sztring műveletek cél indexe, ES regiszterrel, 1. paraméter
- ESP (verem mutató regiszter)
  - Verem tetején lévő elemet címzi, SS regiszterrel
- EBP (bázis/keret regiszter)
  - Lokális változókhoz, paraméterekhez, SS regiszterrel

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

x86 regiszterkészlet

EFLAGS regiszter

- Állapot bitek
- Vezérlő bitek
- Rendszer bitek

31	30	29	28	27	26	25	24
0	0	0	0	0	0	0	0

23	22	21	20	19	18	17	16
0	0	ID	VIP	VIF	AC	VM	RF

15	14	13	12	11	10	9	8
0	NT	IOPL		OF	DF	IF	TF

7	6	5	4	3	2	1	0
SF	ZF	0	AF	0	PF	1	CF

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

x86 regiszterkészlet

Példa néhány EFLAGS bitre

- CF=1, ha aritmetikai/logikai művelet átvitelt eredményezt nem ábrázolható bitpozícióba
- OF=1, ha egy művelet túlcsordulás okozott
- SF=1, ha az eredmény legnagyobb helyiértékén 1-es bit található (negatívként is értelmezhető)
- ZF=1, ha a legutóbbi eredmény értéke nulla
- PF=1, ha az eredményben szereplő 1-es bitek száma páros
- IF=1, a maszkolható megszakítás engedélyezett

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

x86 regiszterkészlet

Utasítás számláló

- EIP (Instruction pointer)
  - CS regiszter révén a kód szegmensben található utasításokat címzi
  - Minden „fetch-execute” ciklus során inkrementálódik az adott utasítás hosszával (kivéve vezérlésátadás)

Egyéb regiszterek

- Vannak további működést segítő regiszterek
- Programozó elől rejtettek

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

x86-64 utasítások, operandusok

Többszáz féle utasítás

Utasításonként 0, 1 vagy 2 operandus

- Regiszter (8, 16, 32, 64 bites)
- Konstans (8, 16, 32, 64 bites)
- Memória tartalom
  - Memória cím és méret kényszerítés

```
mov  al, BYTE  PTR [v]
mov  ax, WORD  PTR [v]
mov  eax,DWORD PTR [v]
mov  rax,QWORD PTR [v]
```

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

x86 címzési módok

Effektív cím (EA) megadás alakjainak összegzése

$$\left[\left\{\begin{array}{c}CS:\\SS:\\DS:\\ES:\\FS:\\GS:\\egyiksem\end{array}\right\}\left\{\begin{array}{c}EAX\\EBX\\ECX\\EDX\\ESI\\EDI\\EBP\\ESP\\egyiksem\end{array}\right\}\left\{+\begin{array}{c}EAX\\EBX\\ECX\\EDX\\ESI\\EDI\\EBP\\egyiksem\end{array}\right\}\left\{*\begin{array}{c}1\\2\\4\\8\\egyiksem\end{array}\right\}\left\{+\begin{array}{c}eltolás\\semmi\end{array}\right\}\right]$$

Szegmens szelektor | *Bázis* | *Index* | *Skálázó faktor* | *Eltolás*

Példa:

```
mov EAX, [DS:EBP+EDI*4+16]
```

vagy ugyanaz más írásmódban:

```
mov EAX, DS:10h[EBP][EDI*4]
```


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

x86 assembly szintaxis

Intel szintaxis

```
.intel_syntax noprefix
.globl main
main: push ebp
      mov  ebp, esp
      sub  esp, 16
      mov  DWORD PTR [ebp-16], 2
      mov  DWORD PTR [ebp-12], 3
      cmp  DWORD PTR [ebp-16], 4
      jne  .L2
      mov  eax, DWORD PTR [ebp-12]
      mov  DWORD PTR [ebp-8], eax
      jmp  .L3
.L2:  mov  DWORD PTR [ebp-8], 4
.L3:  mov  eax, DWORD PTR [ebp-8]
      add  esp, 16
      pop  ebp
      ret
```

AT&T szintaxis

```
.att_syntax noprefix
.globl main
main: pushl %ebp
      movl  %esp, %ebp
      subl  $16, %esp
      movl  $2, -16(%ebp)
      movl  $3, -12(%ebp)
      cmpl  $4, -16(%ebp)
      jne   .L2
      movl  -12(%ebp), %eax
      movl  %eax, -8(%ebp)
      jmp   .L3
.L2:  movl  $4, -8(%ebp)
.L3:  movl  -8(%ebp), %eax
      addl  $16, %esp
      popl  %ebp
      ret
```

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

x86 alprogram hívási konvenció

Hívó szabályai

- Paraméterek adott sorrendben regiszterekbe: `edi`, `esi`, `edx`, `ecx`, …
  - ➥ Vagy fordított sorrendben a rendszerverembe helyezése
  - ➥ Lebegő pontos paraméterek a más regiszterekbe (`eax`-be a float paraméterek száma)
- Alprogram hívás (visszatérési cím a verembe kerül, programszámláló átáll az alprogram kezdőcímére)
- Visszatérés után veremből paraméterek kivétele
- Visszatérési érték az `eax` regiszterben van

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

x86 alprogram hívási konvenció

Hívott szabályai

- Bázis pointer (`EBP`) verembe mentés
- Verem mutató (`ESP`) mentése bázis pointerbe
- Helyfoglalás lokális változóknak a veremben
- Használandó regiszterek mentése a verembe

- Visszatérési érték az `eax` regiszterbe tétele
- Mentett regiszterek és verem visszaállítás
- Visszatérés (a veremben lévő címre)

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

Intel Core i9

- Megjelenési év 2017
- 64 bites regiszterek (x86-64)
- Mikroarchitektúra: Skylake, Coffee Lake, Rocket Lake
- 14nm technológia
- FCLGA aljzat (2066 kontaktus)
- 6-18 mag, 13-24MB L3 smart cache, AVX-512, HT, DMI 3.0, Virtualization Technology (VT-x, VT-d), DDR4, Turbo Boost Max 3.0 (5,2GHz), DL Boost, AES-NI

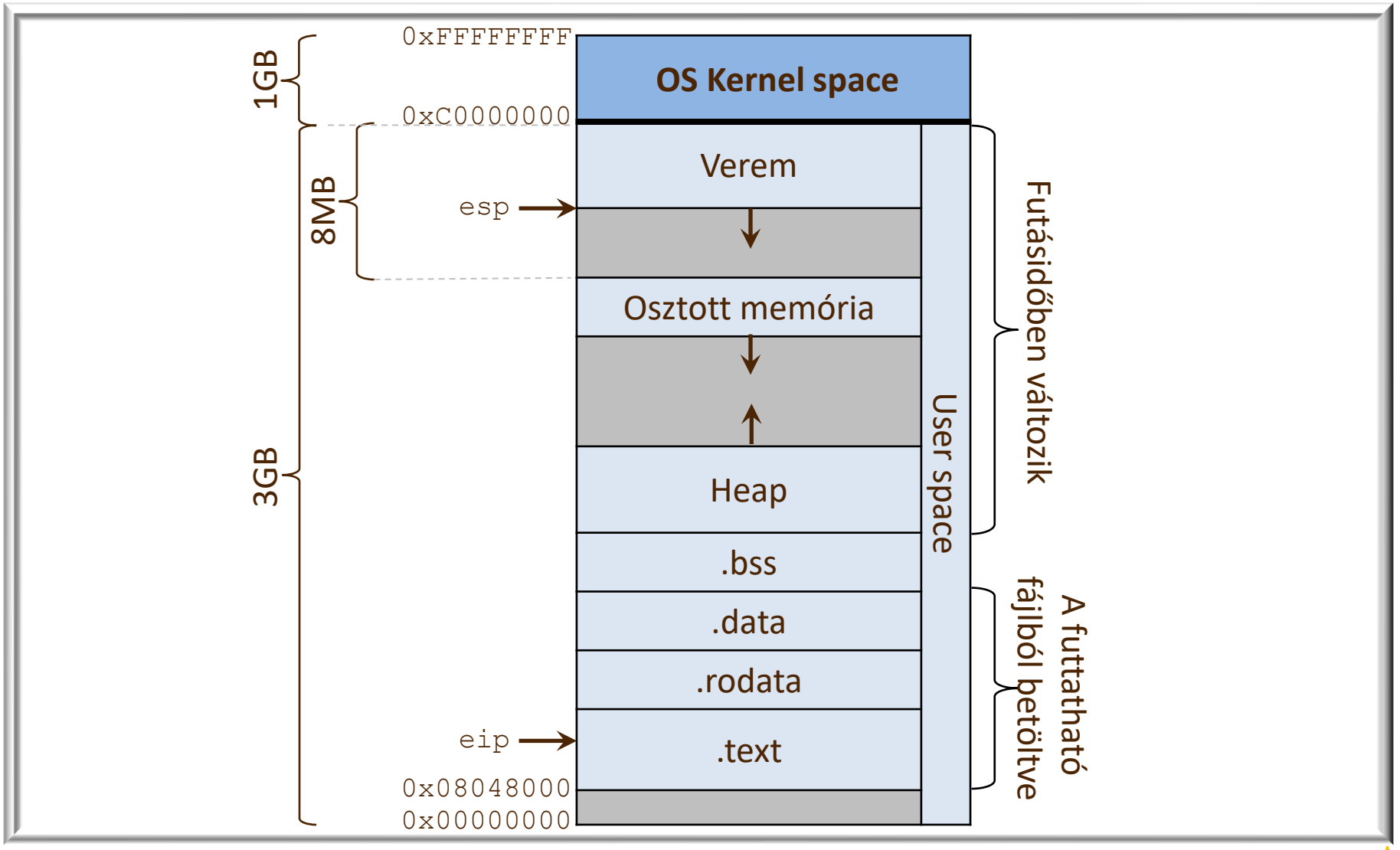
```
mov eax, DWORD PTR [rbp+16]
and eax, 0FFFFFD00h
mov rsp, rbp
pop rbp
ret
```

AMD Ryzen 9

- Megjelenési év 2019
- 64 bites regiszterek (x86-64)
- Mikroarchitektúra: Zen2, Zen3
- 7-14nm technológia
- Kb. 10 milliárd tranzisztor
- AM4, FP6 aljzat (1331, 1140 kontaktus)
- 8-16 mag, 32-64MB L3 cache, AVX2, HT, PCIe 4.0, APU, DDR4, AMD-V virtualizáció, unlocked, Turbo Core (4,9GHz)

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

Egy program címtere Linux-on



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

Ajánlott irodalom

- Andrew S. Tanenbaum: *Számítógép architektúrák*, Panem, 2006
- Richard Blum: *Professional Assembly Language*, Wiley, 2005
- Sarah L. Harris, David M. Harris: Digital design and computer architecture (ARM edition), Morgan Kaufman, 2016
- Nicholas Charter: *Computer architecture*, McGraw-Hill, 2001
- David A Patterson, John L Hennessy: Computer organization and design, Morgan Kaufman, 2014
- R. E. Bryant, D. R. O'Hallaron: *Computer Systems – A programmer's perspective*, Pearson, 2016
- Joseph Cavanagh: *X86 Assembly Language and C Fundamentals*, CRC Press, 2013