ASSEMBLY

1. előadás

Kitlei Róbert
kitlei.web.elte.hu

Jegyzet: Assembly programozás

www.inf.elte.hu
→ Karunkról
→ Kari digitális könyvtár

ASSEMBLY

Mi az assembly?

- gépi kód: a számítógép által közvetlenül értelmezett és végrehajtott jelsorozat
- assembly: a gépi kód emberek számára könnyen értelmezhető reprezentációja

ASSEMBLY

Miért érdemes assemblyt tanulni?

- a legalacsonyabb absztrakciós szint a gépi kód felett
- gyors, hatékony
- bizonyos feladatokhoz nélkülözhetetlen
- új rálátást ad

Milyen assemblert használunk?

- architektúra: x86-32
  - két fő gyártó: AMD, Intel
- operációs rendszer: Linux
- assembler: nasm
  - több platformra is megtalálható
- linker: gcc

ASSEMBLY

Milyen más tárgyak kapcsolódnak?

- Nagy hatékonyságú számítások RISC processzorokkal
- zSeries (IBM nagygépek)
- Operációs rendszerek
- Programnyelvek I. és II.

Platform C64

ASSEMBLY

```
start
ldx #$0d
cycle
lda hworld,x
sta $0400,x
dex
bne cycle
hworld
rts
.text "hello world!!!"
```

ASSEMBLY

Platform zSeries ”nagygép”

```
TITLE 'Hello World (VM/CMS)'
HELLO    START
         BALR  12,0
         USING *,12
*
         WRTERM 'Hello World!'
*
         SR    15,15
         BR    14
*
END   HELLO
```

Platform VAX

```
LL0:
        .data
        .text
L       .align  1
        .globl  _main
_main:
        .word   L12
        jbr     L14
L15:
        .data   1
L17:
        .ascii  "Hello, world.2"
        ..........
```

ASSEMBLY

Platform x86, DOS, MASM

```
.MODEL tiny
.CODE
        ORG 100h
HELLO           PROC
        MOV     AH,09h
        LEA     DX,msg
        INT     21h
        MOV     AX,4C00h
        INT     21h
HELLO           ENDP
        msg     DB      'Hello World$'
        END     HELLO
```

ASSEMBLY

Platform x86, DOS, TASM

```
; WRITTEN IN TASM (Turbo Assembler)
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
        mov ah,9
        mov dx,OFFSET Msg
        int 21h
        int 20h
        Msg DB 'Hello World',13,10,'$'
CODE ENDS
END START
```

ASSEMBLY

Platform x86, NetBSD, gas

```
.data
 msg:
     .string "Hello World\n"
 len:
     .long . - msg
 .text
 .globl _start
 _start:
     push $len          /* Laenge */
     push $msg          /* Adresse */
     push $1            /* Stdout */
     movl $0x4, %eax    /* write */
     ........
```

ASSEMBLY

Platform x86, Linux, nasm

```
    section .text
    global main
main
    mov eax, 4
    mov ebx, 1
    mov ecx, 11
    mov edx, H
    int 0x80
    mov eax, 1
    xor ebx, ebx
    int 0x80
    section .data
H   db   “Hello world”
```

Adatábrázolás — Logikai értékek

hamis 0 — bit — igaz 1

és and	0	1
0	0	0
1	0	1

vagy or	0	1
0	0	1
1	1	1

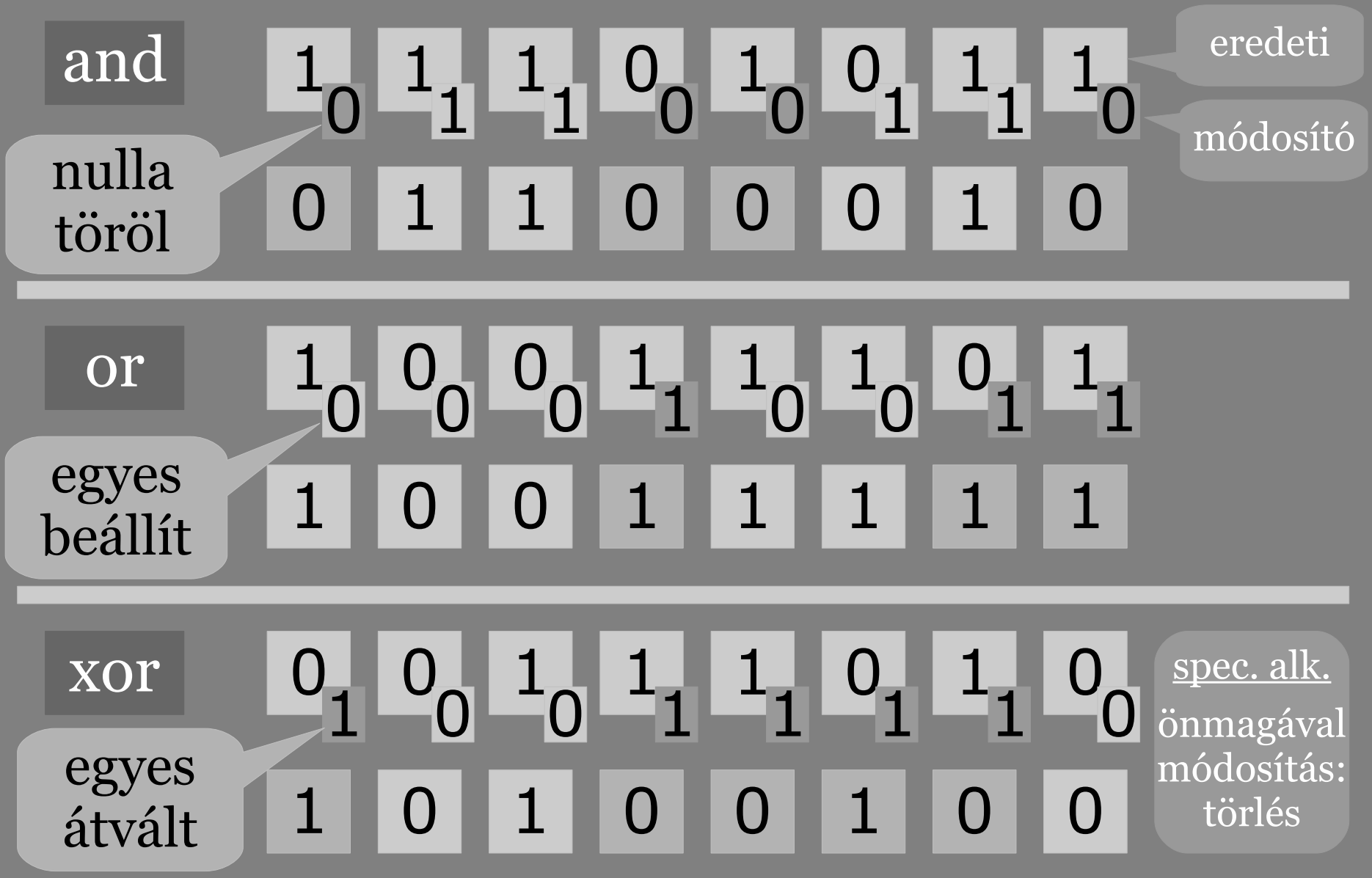
nem	not
0	1
1	0

kizáró vagy xor	0	1
0	0	1
1	1	0

ASSEMBLY

Adatábrázolás — Logikai értékek

bitmanipuláció



ASSEMBLY

Adatábrázolás Nemnegatív számok

számok – előjel nélkül

a bitek sorban

a bitek sorszáma

1	0	0	1	1	1	0	1
7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

$1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 +$
$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
a szám értéke: 157_{10}
leírása: 10011101b

az egyes bitek helyi értékei

Adatábrázolás Nemnegatív számok

ha páros a szám, nullát írunk, ha páratlan, egyet

... majd osztunk kettővel, és folytatjuk

... amíg maradt a számból

ekkor a végeredményt **alulról felfele olvasva** kapjuk: $61_{10} = 111101_2$

ASSEMBLY

Adatábrázolás Nemnegatív számok

hexadecimális (tizenhatos) számrendszer

0 .. 9		mint tízesben			
10_{10}	A	1	0	1	0
11_{10}	B	1	0	1	1
12_{10}	C	1	1	0	0
13_{10}	D	1	1	0	1
14_{10}	E	1	1	1	0
15_{10}	F	1	1	1	1

számok leírása

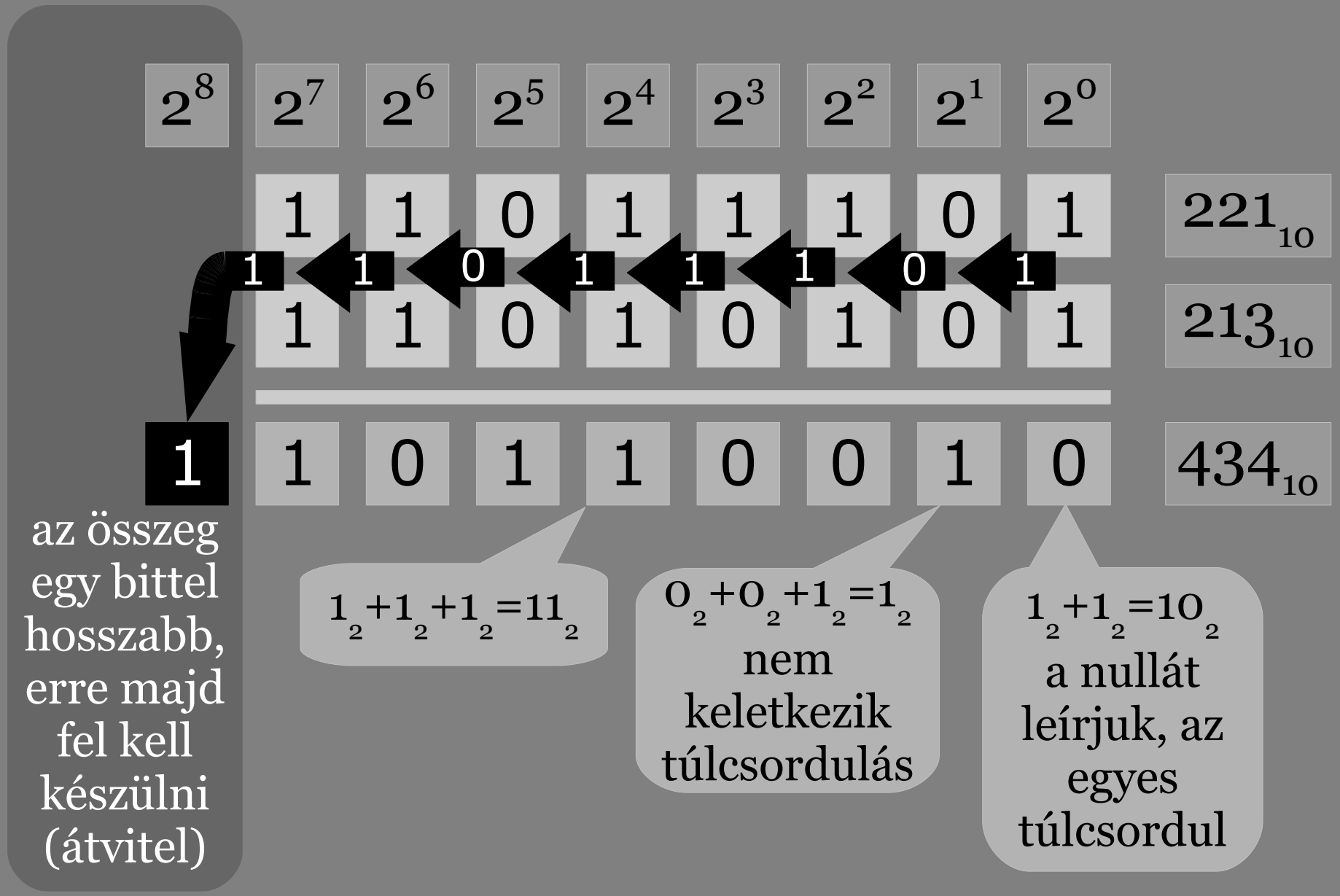
0xABCD $= 43981_{10}$

0Ah $= 10_{10}$

h posztfixszel nullát kell írni elé, ha betűvel kezdődik

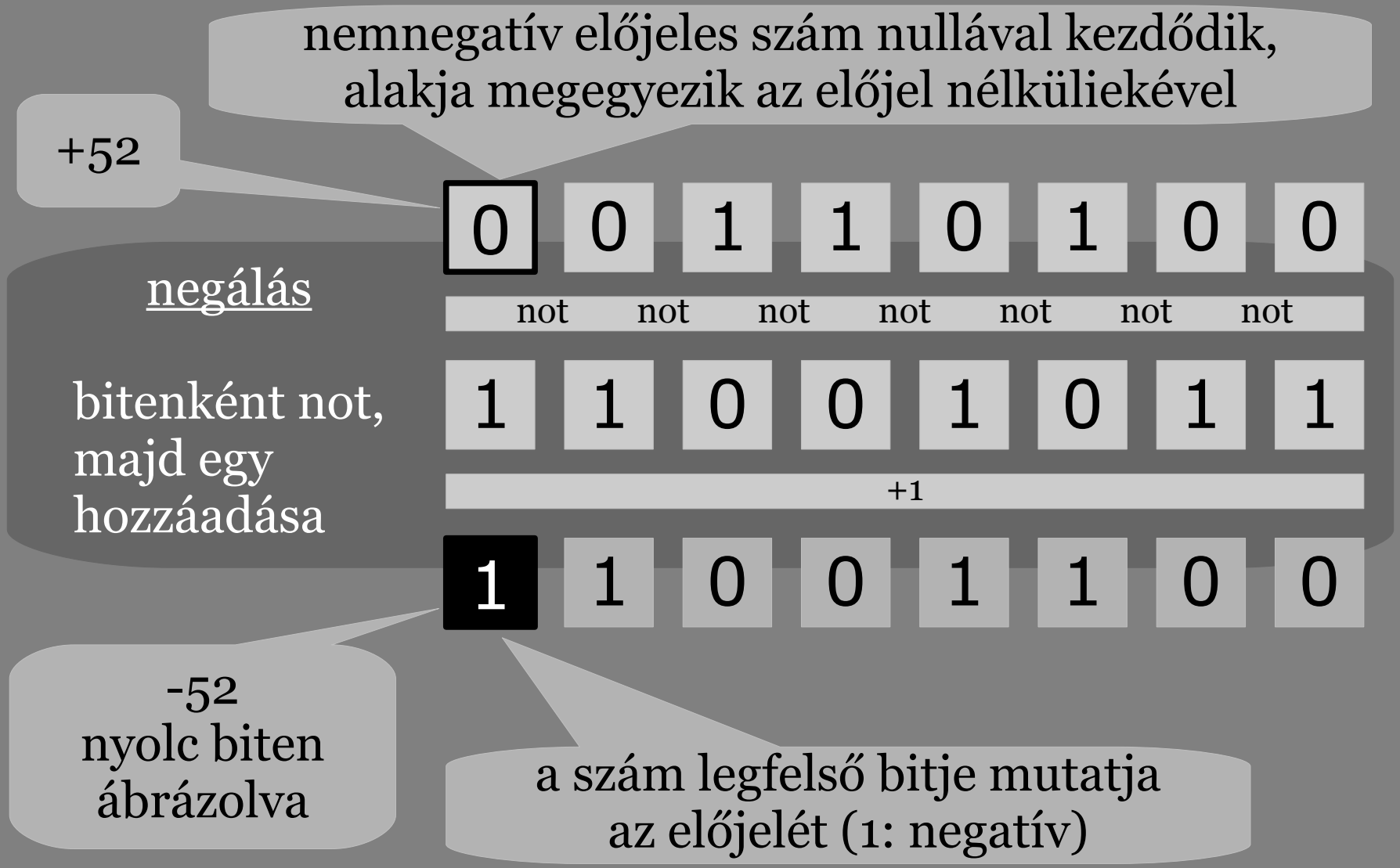
Adatábrázolás Nemnegatív számok

összeadás

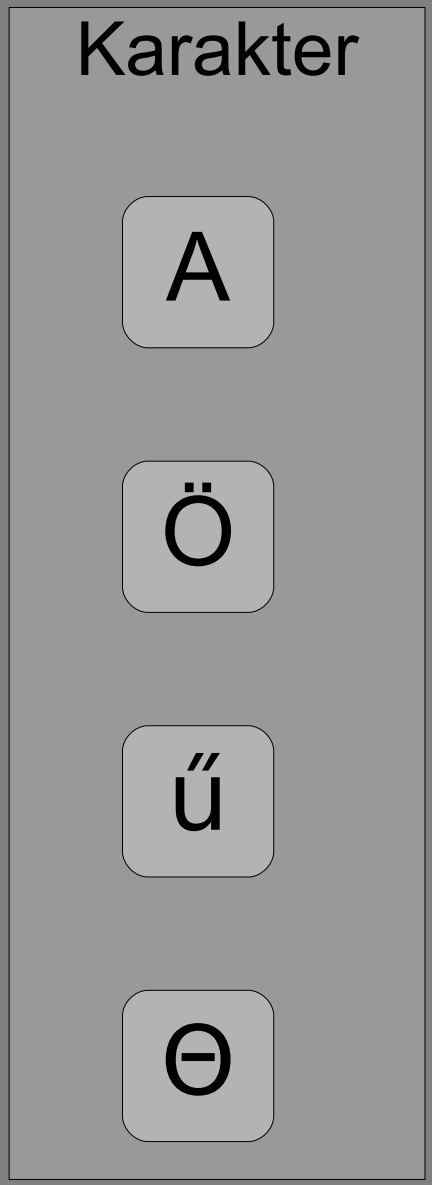


Adatábrázolás Előjeles számok

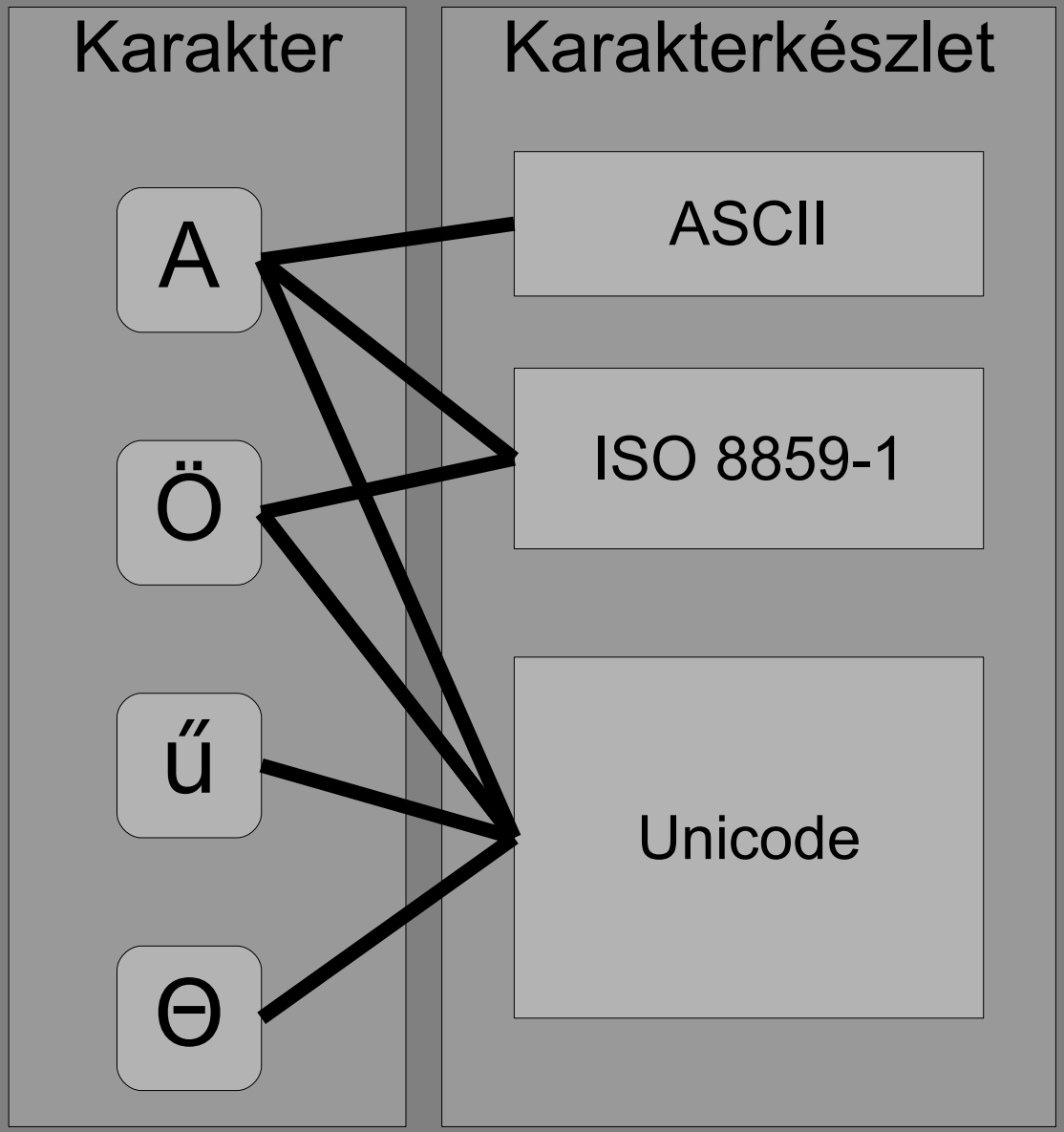
ún. kettes komplemens alak



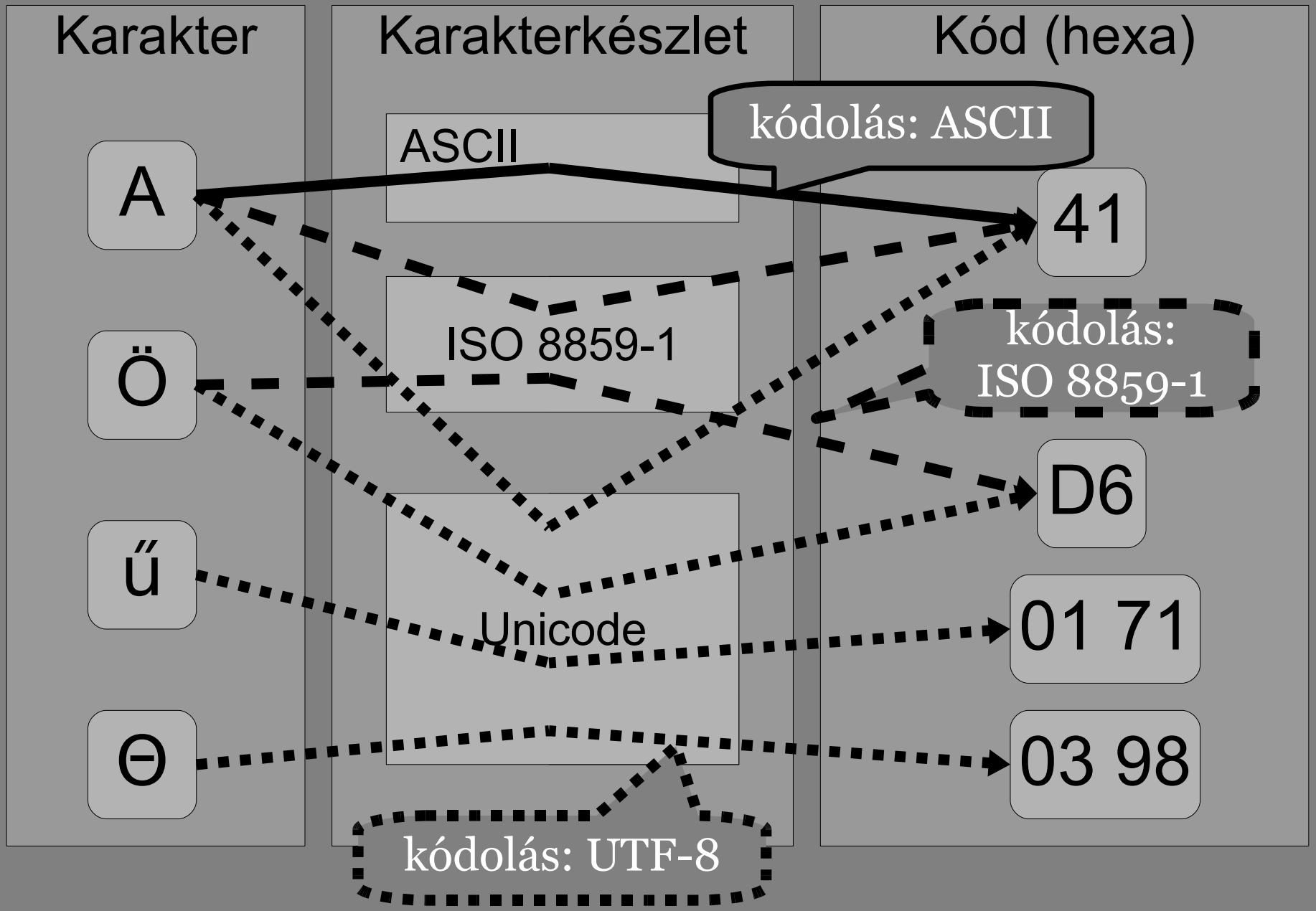
Adatábrázolás ASCII karakterek



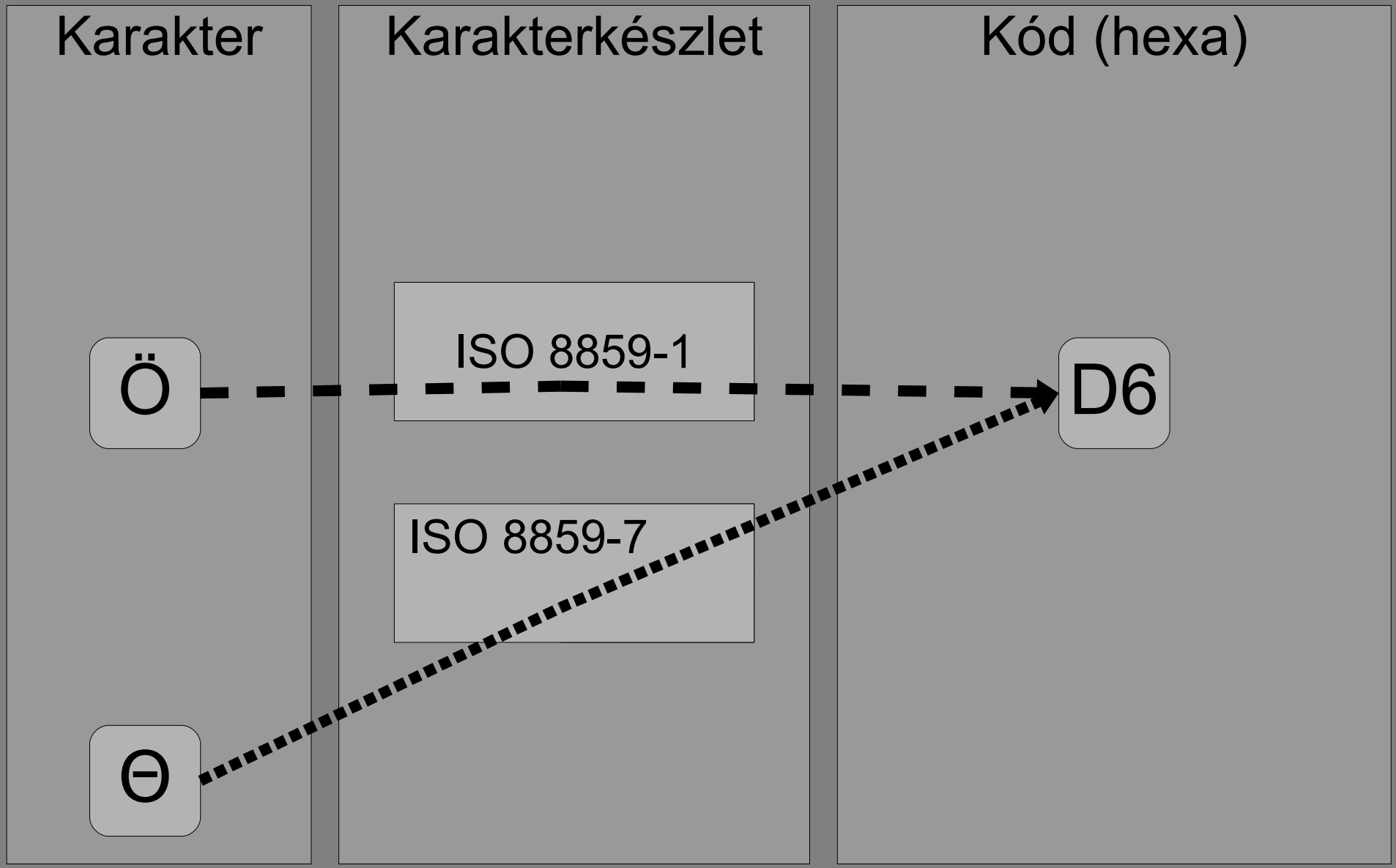
Adatábrázolás ASCII karakterek



Adatábrázolás ASCII karakterek



Adatábrázolás ASCII karakterek



ASSEMBLY

Adatábrázolás ASCII karakterek

karakter	kódja		megjegyzés
új sor	0A	10	a sorvége jelzése DOS, Windows 0D 0A, Unix, Linux alatt 0A
kocsivissza	0D	13	
szóköz	20	32	
0 .. 9	30..39	48..57	a számjegyek
A .. Z	41..5A	65..90	az angol ábécé betűi sorban, kihagyás nélkül
a .. z	61..7A	97..122	

sok karakter nincs benne, ezért fejlesztették ki a Unicode-ot

minden karaktert egy nyolc bites számmal kódolunk

ASSEMBLY

A memória felépítése

RAM

lineáris memóriamodell

a memória byte-ok sorozata

mindegyiknek egy
32 bites címe van

memóriacímek

a memória tartalma

ASSEMBLY

A memória felépítése RAM

lineáris memóriamodell

egy lépésben kezelhető adatok
- byte 8 bit
- word 16 bit szó
- dword 32 bit duplaszó

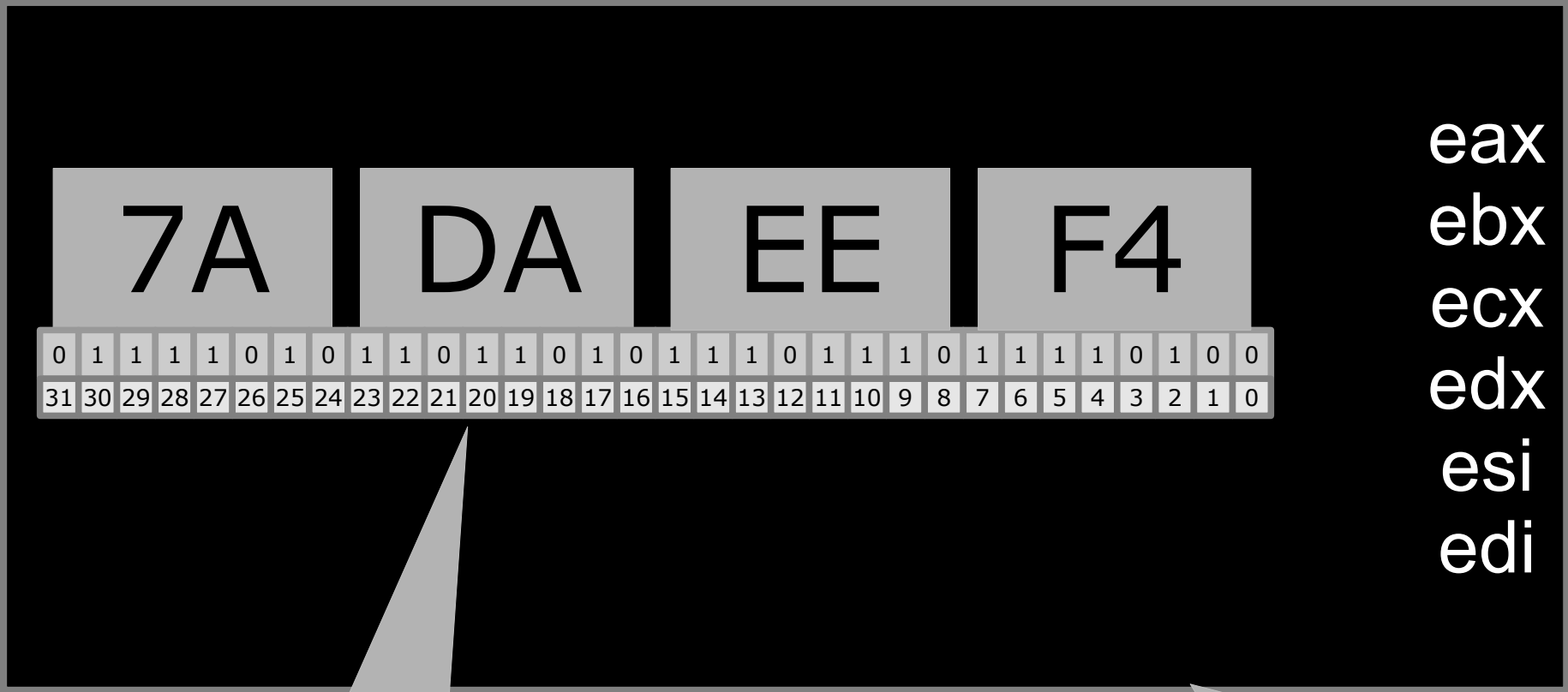
a memóriában byte-onként fordított sorrendben tárolódnak (little-endian)

az A110CA7E címen található duplaszó értéke: $FE5A11FA_{16}=4267315706_{10}$

az A110CA81 címen található szó értéke: $57FE_{16}=22526_{10}$

A memória felépítése Regiszterek

az általános célú regiszterek



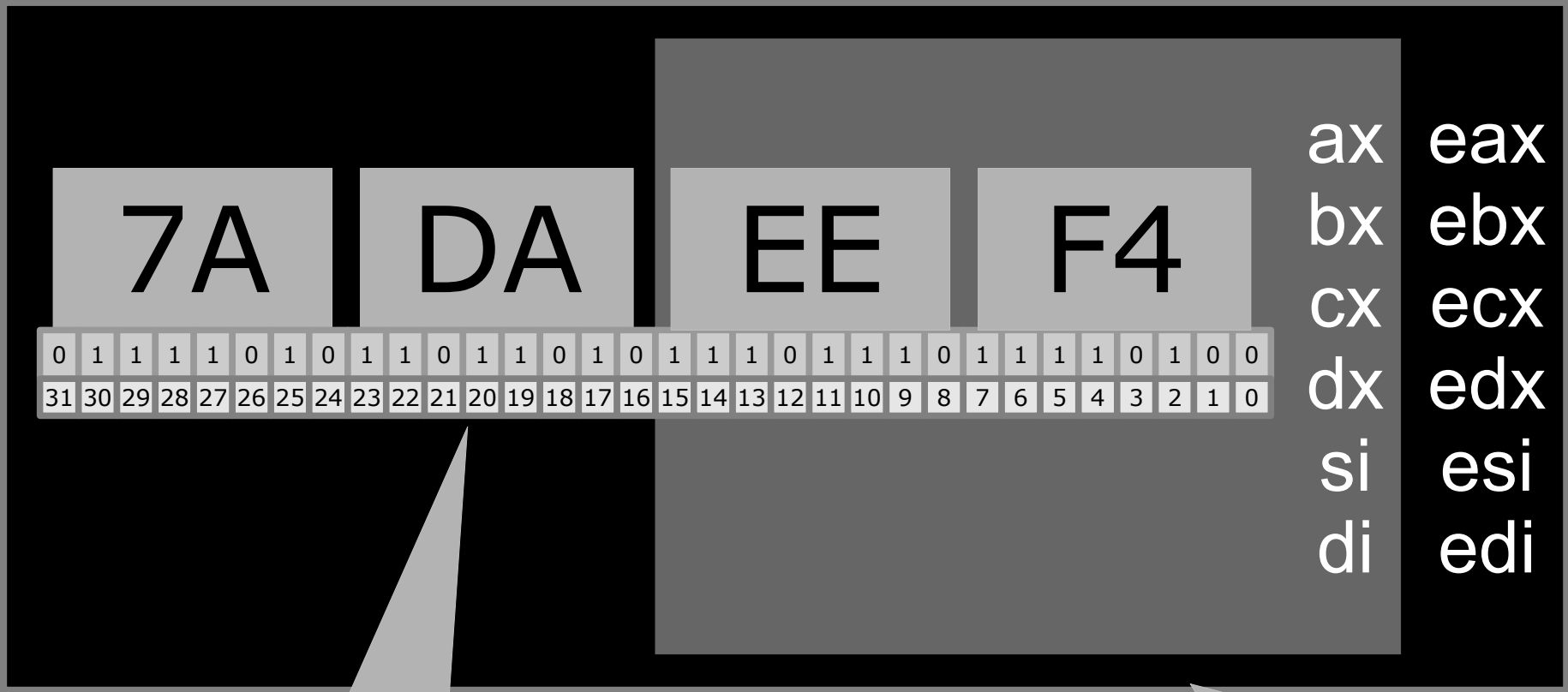
a regiszter bitjei sorszámmal (vö. helyi érték)

az általános célú regiszterek: **eax**, **ebx**, **ecx, edx, esi, edi**

ASSEMBLY

A memória felépítése Regiszterek

az általános célú regiszterek



a regiszter bitjei sorszámmal (vö. helyi érték)

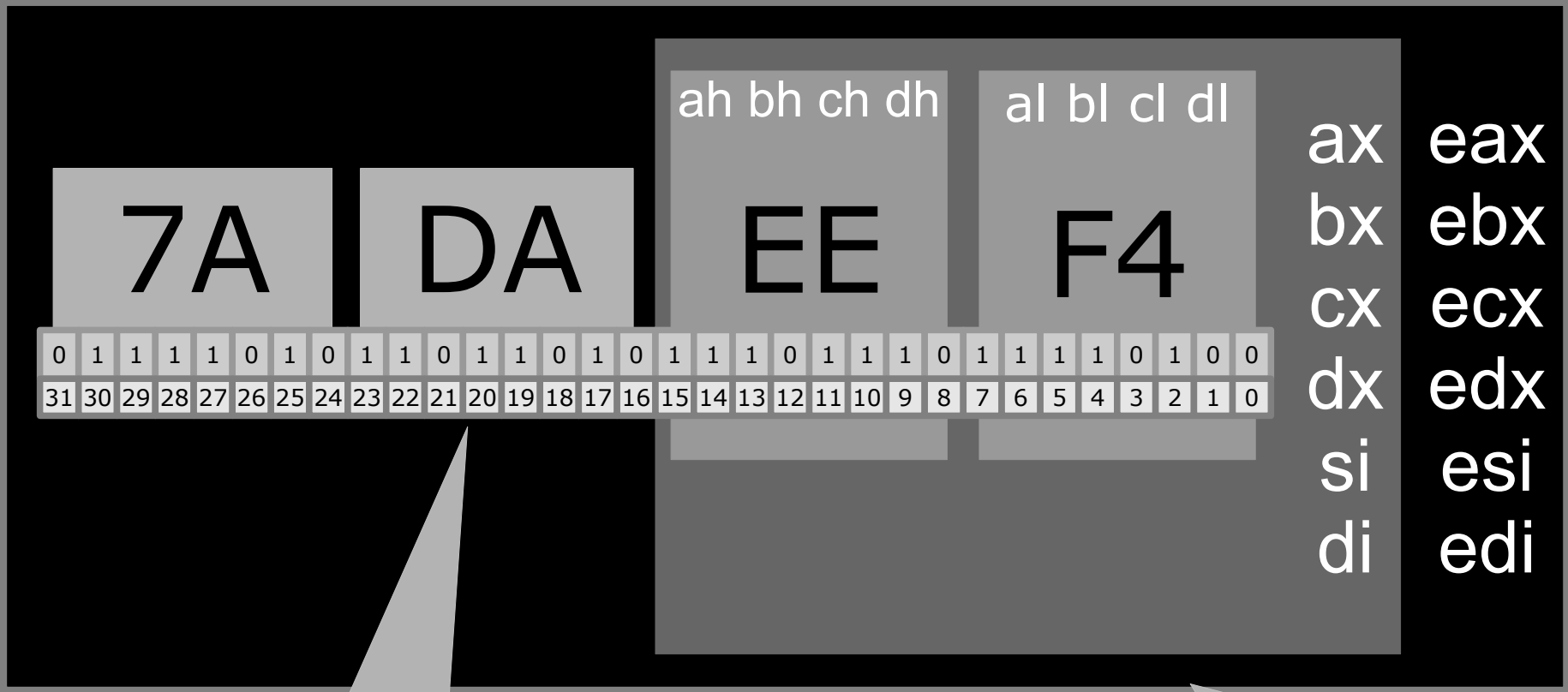
az általános célú regiszterek:
eax, **ebx**, **ecx**, **edx**, **esi**, **edi**

32 bitesek, egyes részeik külön névvel rendelkeznek

ASSEMBLY

A memória felépítése Regiszterek

az általános célú regiszterek



a regiszter bitjei sorszámmal (vö. helyi érték)

az általános célú regiszterek: **eax**, **ebx**, **ecx, edx, esi, edi**

32 bitesek, egyes részeik külön névvel rendelkeznek

A memória felépítése Regiszterek

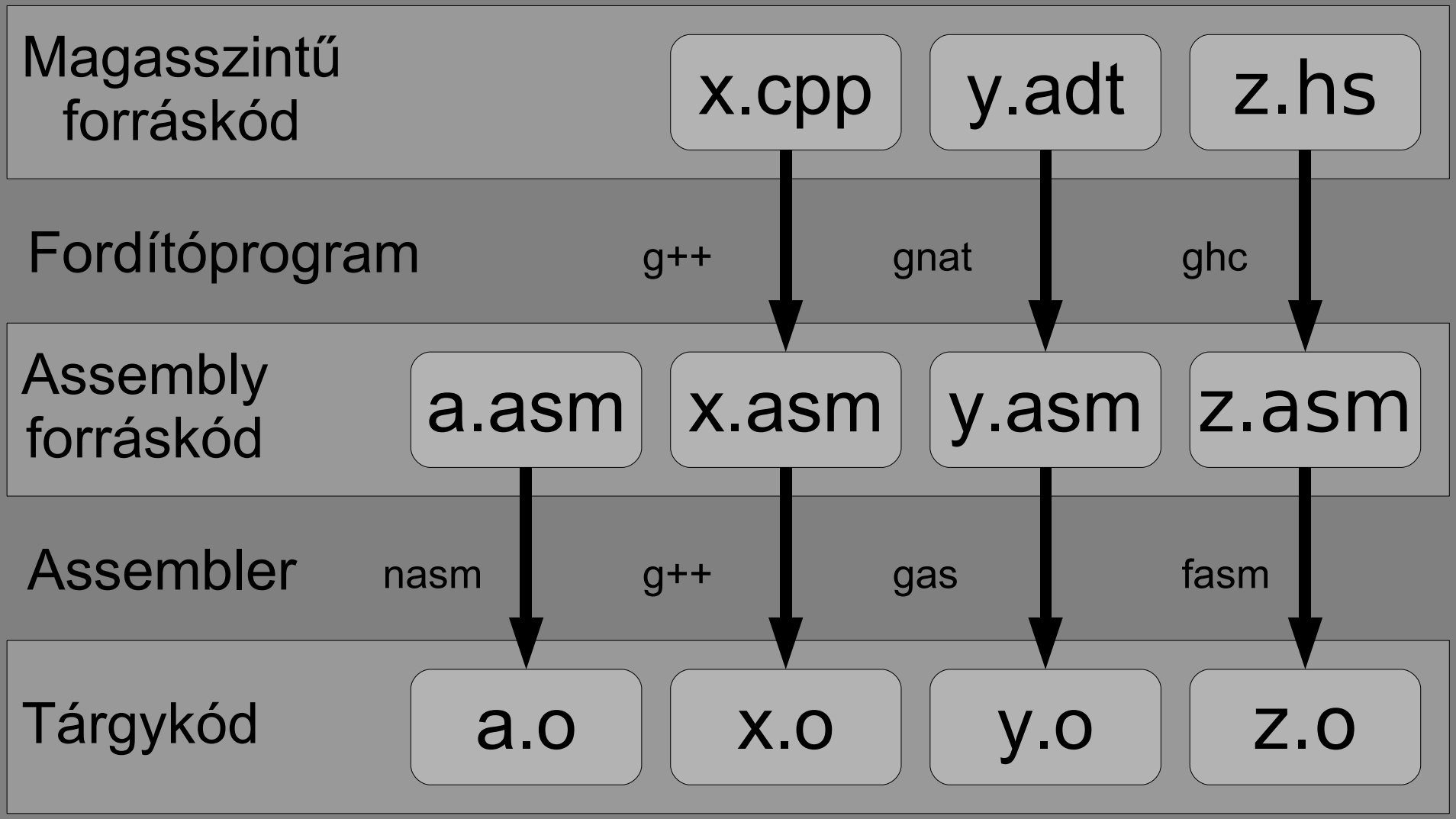
további 32 bites regiszterek

jelzőbitek regisztere
- átviteli jelzőbit (carry)

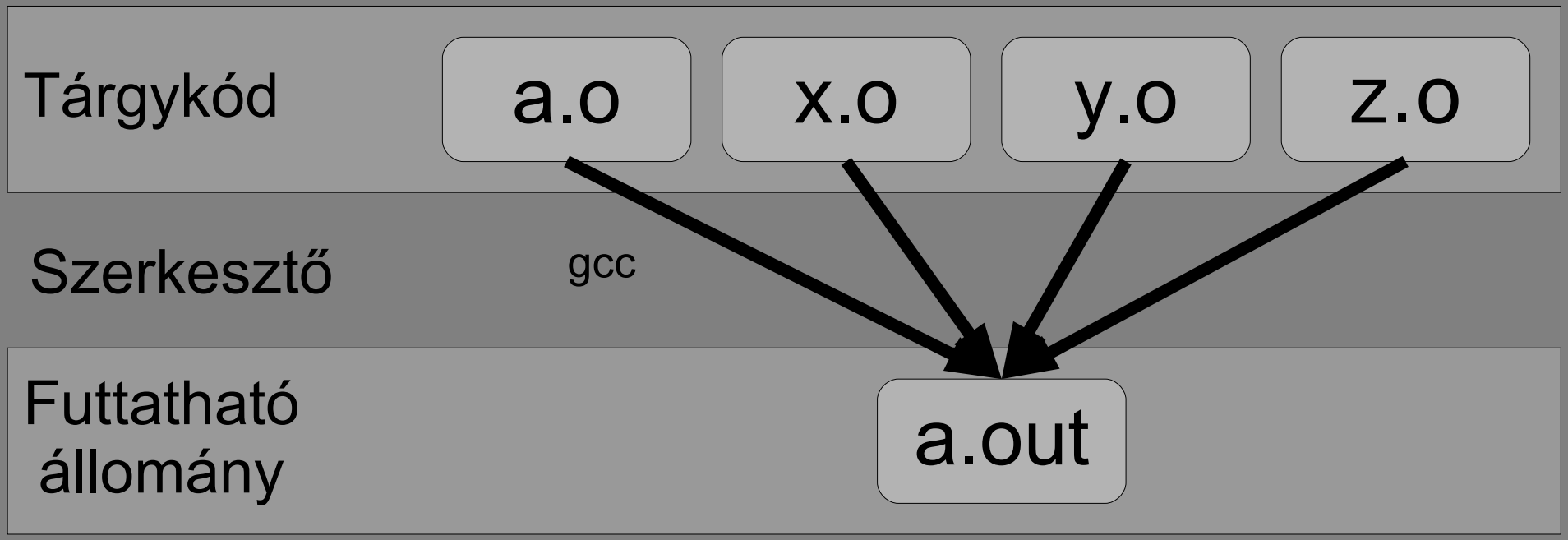
veremkezelő regiszterek: **esp**, **ebp**
- ezek is általános célúak
- csak a futási idejű verem kezelésére használjuk őket

ASSEMBLY

A fordítás menete



A fordítás menete



ASSEMBLY

Az első program Programkód

```
    section .text   ; a kódszegmens kezdete
    global main     ; a program legyen elindítható

main                ; a program belépési pontját jelző címke
    mov   eax, 4                ; a rendszerszolgáltatás
    mov   ebx, 1                ; paramétereinek  beállítása:
    mov   ecx, szoveg           ; szöveg kiírása a képernyőre
    mov   edx, 6                ; a szoveg címkéről 6 hosszan
    int   0x80      ; rendszerszolgáltatás hívása

    mov   eax, 1            ; a kilépés paramétereinek
    mov   ebx, 0            ; beállítása
    int   0x80      ; rendszerszolgáltatás hívása: kilépés

    section .data   ; az adatszegmens kezdete
szoveg  db   "Hello", 0xA   ; a szöveg, végén egy újsorral
```

Az első program Fordítás, futtatás

fordítás és futtatás

```
> nasm hellovilag.asm -f elf
                                    hellovilag.o
> gcc hellovilag.o -o hellovilag
                                        hellovilag
> ./hellovilag
Hello
>
```

Az első program — Lista

memóriacím	a generált bájtok	az eredeti forrásszöveg
		section .text ; ez a 3 sor
		global main ; nem generál
		main ; kódot
00000000	B8 04000000	mov eax, 4
00000005	BB 01000000	mov ebx, 1
0000000A	B9 00000000	mov ecx, szoveg
0000000F	BA 06000000	mov edx, 6
00000014	CD 80	int 0x80
00000016	B8 01000000	mov eax, 1
0000001B	BB 00000000	mov ebx, 0
00000020	CD 80	int 0x80
		section .data
00000000	48 65 6C 6C 6F	szoveg db ”Hello”
00000005	0A	db 0xA

az utasítás kódja

a paraméterek kódolt alakja

a memóriában sorban megjelenő bájtok

Utasítások — Adatmozgatás

mov
feltétel nélküli
adatmozgatás

mnemonik

edx a **cél**operandus

ebx a **forrás**-operandus

```
mov edx, ebx
mov bh, 19
mov ah, 0Ah ; 0 nélkül más
mov ecx, 0xFEDCBA
; a teteje nullákkal töltődik fel
```

általános elv
a forrás és a cél hosszának meg kell egyeznie

```
mov edi, címke
; edi címke értékét veszi fel, ami egy szám

mov ebx, [címke]
; ebx a címke által jelölt
; memóriacímen található
; duplaszó értékét veszi fel
```

HELYTELEN

```
mov   eax, bl
mov   [c1], [c2]
```

Utasítások Adatmozgatás

mov
feltétel nélküli
adatmozgatás

Tegyük fel, hogy *címke* értéke 0x01DE7EDD. Ekkor

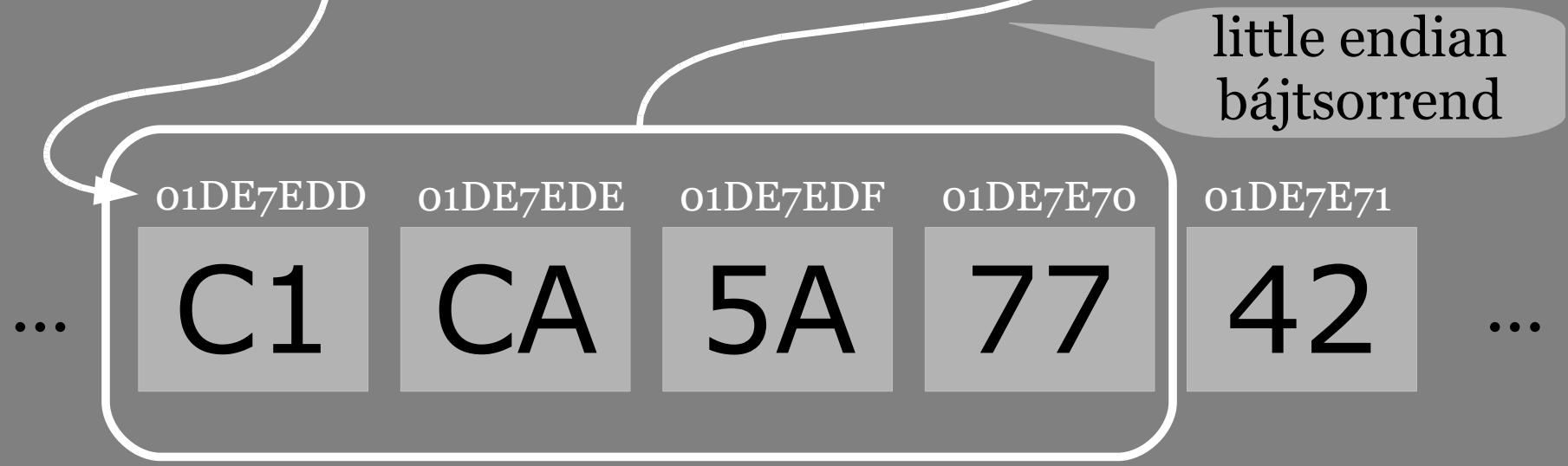
mov ecx, *címke* hatása olyan, mint mov ecx, 0x01DE7EDD

Utasítások Adatmozgatás

mov
feltétel nélküli
adatmozgatás

Tegyük fel, hogy *címke* értéke 0x01DE7EDD. Ekkor

mov ecx, [*címke*] hatása olyan, mint mov ecx, 0x775ACAC1

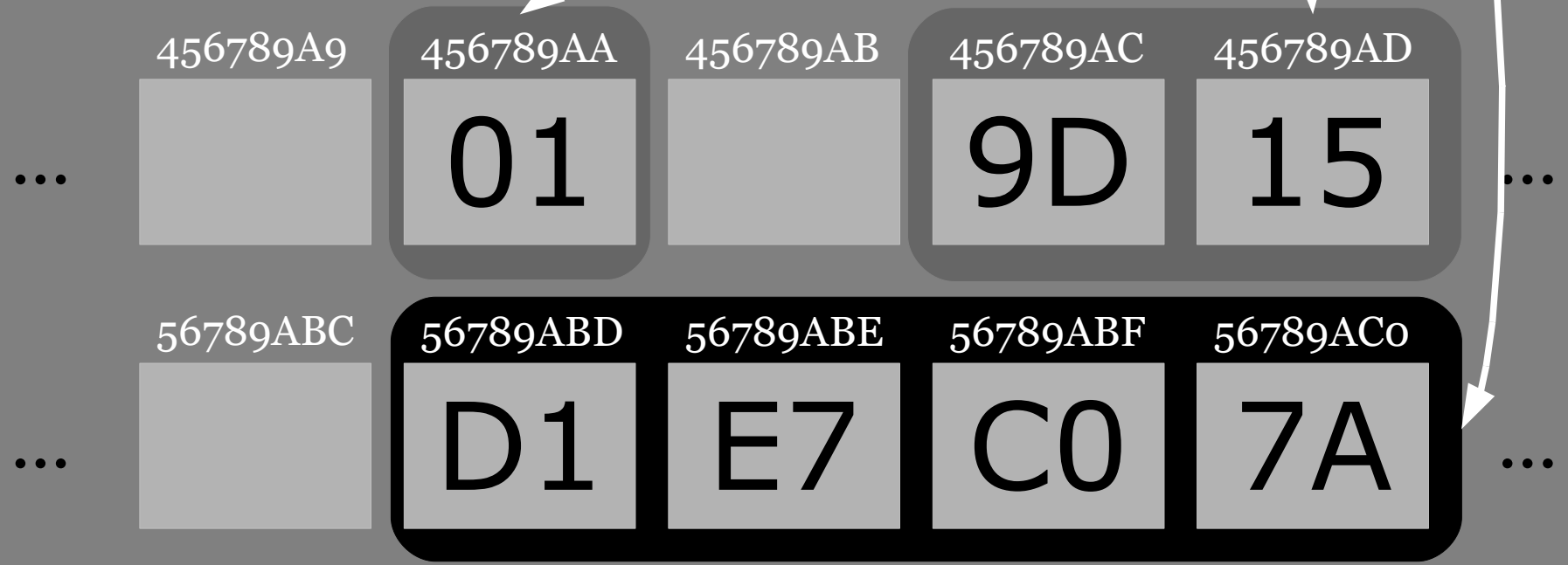


Utasítások Adatmozgatás

meg kell adni a méretet,
csak a címből nem derül ki

mov
feltétel nélküli
adatmozgatás

```
mov     dword [56789ABDh], 0x1AC0E7D1
mov     word [456789ACh], 159Dh
mov     byte [456789AAh], 1
```



ASSEMBLY

Utasítások Adatmozgatás

movzx	movsx	xchg
előjel nélküli hosszkiterjesztés	előjeles hosszkiterjesztés	adatok felcserélése

movsx és movzx: a cél hosszabb, mint a forrás

- movzx: nullákkal terjeszt ki
- movsx: a felső bittel terjeszt ki

```
mov    al, 10000000b
movzx  bx, al
  ; bx = 0000000010000000b
movsx  cx, al
  ; cx = 1111111110000000b
xchg       bx, cx
  ; bx = 0xFF80, cx = 0x0080
```

HELYTELEN

```
movzx  eax,  ebx
movsx  ax,   edi
xchg   bl,   edx
```

ASSEMBLY

Utasítások Logikai műveletek

not	and	or	xor
logikai nem	logikai és	logikai vagy	logikai kizáró vagy

```
mov al,          11001011b
not  al   ; al = 00110100b
and  al,         10010010b
          ; al = 00010000b
or   al,         10101001b
          ; al = 10111001b
xor  al,         01001101b
          ; al = 11110100b
```

ASSEMBLY

Utasítások Aritmetikai műveletek

inc	add	dec	sub
érték növelése	összeadás	érték csökkentése	kivonás

neg
negáció

```
mov  al,  6
mov  dl,  5
sub  al,  dl    ; al = 1
inc  al         ; al = 2
add  al,  al    ; al = 4
dec  al         ; al = 3
neg  al         ; al = -3 előjelesen
                          előjel nélkül: 253
```

hatása ekvivalens ezzel:

```
not al
inc al
```

ASSEMBLY

Utasítások Aritmetikai műveletek

mul	imul	div	idiv
előjel nélküli szorzás	előjeles szorzás	előjel nélküli osztás	előjeles osztás

egyoperandusúak, ennek hosszától függően a másik:

- byte: ax illetve al (osztás maradéka: ah)
- szó: dx:ax illetve ax (osztás maradéka: dx)
- duplaszó: edx:eax illetve eax (osztás maradéka: edx)

A dx:ax és edx:eax regiszterek úgy tekintendőek, mintha egy 32 (64) bites regisztert alkotnának összeolvasva.

HELYTELEN

```
div     2
div     byte 2
mul     eax,  3
```

Utasítások Aritmetikai műveletek

mul	imul	div	idiv
előjel nélküli szorzás	előjeles szorzás	előjel nélküli osztás	előjeles osztás

```
mov  al,  5
mov  dx, 0x0AFE  ; dl = -2, dh = 10
mov  bx, 20000
mul  dh          ; ax = 50 = 0x0032, dl marad
imul dl          ; ax = 0xFF9C = -100
div  bx
  ; ez nem ax-ot osztja, hanem dx:ax-ot:
  ; 0x0AFEFF9C = 184 483 740
  ; ax = 9224, dx = 3740 = 0x0E9C, dl = -100
idiv dl          ; al = -92 = 0xA4, ah = 24
```

ASSEMBLY

Utasítások Bitforgatás

shl és shr bitléptetés balra/jobbra	sal és sar aritmetikai léptetés balra/jobbra	rol és ror bitforgatás balra/jobbra

```
mov  eax, 0xB38F  ; eax = 1011001110001111b
shr  eax, 3       ; eax = 000101100111000 1b
shl  eax, 5       ; eax = 1100111000100000b
ror  eax, 7       ; eax = 0100000110011100b
rol  eax, 13      ; eax = 1000100000110011b
sar  eax, 4       ; eax = 1111100010000011b
sal  eax, 2       ; eax = 1110001000011000b
```

kettőhatvánnyal szorzás
- shl, shr: előjel nélkül
- sal, sar: előjelesen

ASSEMBLY

Utasítások Feltételvizsgálat

cmp összehasonlítás	jmp feltétel nélküli ugrás	ja, jb, jc, ... feltételes ugrások

egy feltételvizsgálat két sorból áll

```
cmp  eax, ebx   ; beállítja a jelzőbiteket
je   cimke      ; a jelzőbitek állása szerint ugrik
                ; ha a feltétel nem áll fenn, a program
                ; a következő utasítással folytatódik
```

a feltételes ugrások alakja

	<	≤	>	≥
nem előjeles	jb, jnae	jbe, jna	ja, jnbe	jae, jnb
előjeles	jl, jnge	jle, jng	jg, jnle	jge, jnl

jelzőbitek

je, jne = jz, jnz egyenlőség
jc, jnc átvitel

Utasítások Logikai műveletek

bt bittesztelés	bts bittesztelés és beállítás	btc bittesztelés és törlés	btr bittesztelés és fordítás

seta, ...
feltételes
értékbeállítás

```
mov al,                                  00100101b
bt  al, 0  ; átviteli bit = 1, al = 00100101b
bts al, 3  ; átviteli bit = 0, al = 00101101b
btc al, 5  ; átviteli bit = 1, al = 00001101b
btr al, 7  ; átviteli bit = 0, al = 10001101b
cmp al, 10001101b
sete dl      ; dl = 1
```