Bevezetés az informatikába

- Az összeadás, kivonás, szorzás algoritmusai. Prefixumok az informatikában
- Előjel nélküli egész számok ábrázolása a digitális számítógépeknél. Szorzás, összeadás, kivonás. Előjeles egész számok ábrázolása a digitális számítógépeknél. Összeadás, kivonás.
- Lebegőpontos számábrázolás különböző típusoknál. IEEE 754 szabvány.
- BCD, EBCDIC kódok. ASCII kódrendszer. Decimális számábrázolás ASCII kódban, EBCDIC kódban és patkolt zónás módban. A logikai típus ábrázolása, műveletek.
- Kifejezések kiértékelése. Műveletek precendenciája, balról jobbra szabály.

összeadás, szorzás, kivonás

Összeadás

decimális, bináris, oktális és hexadecimális számrendszerekben

Szorzás

decimális és bináris számrendszerekben

kivonás

decimális, bináris, oktális és hexadecimális számrendszerekben

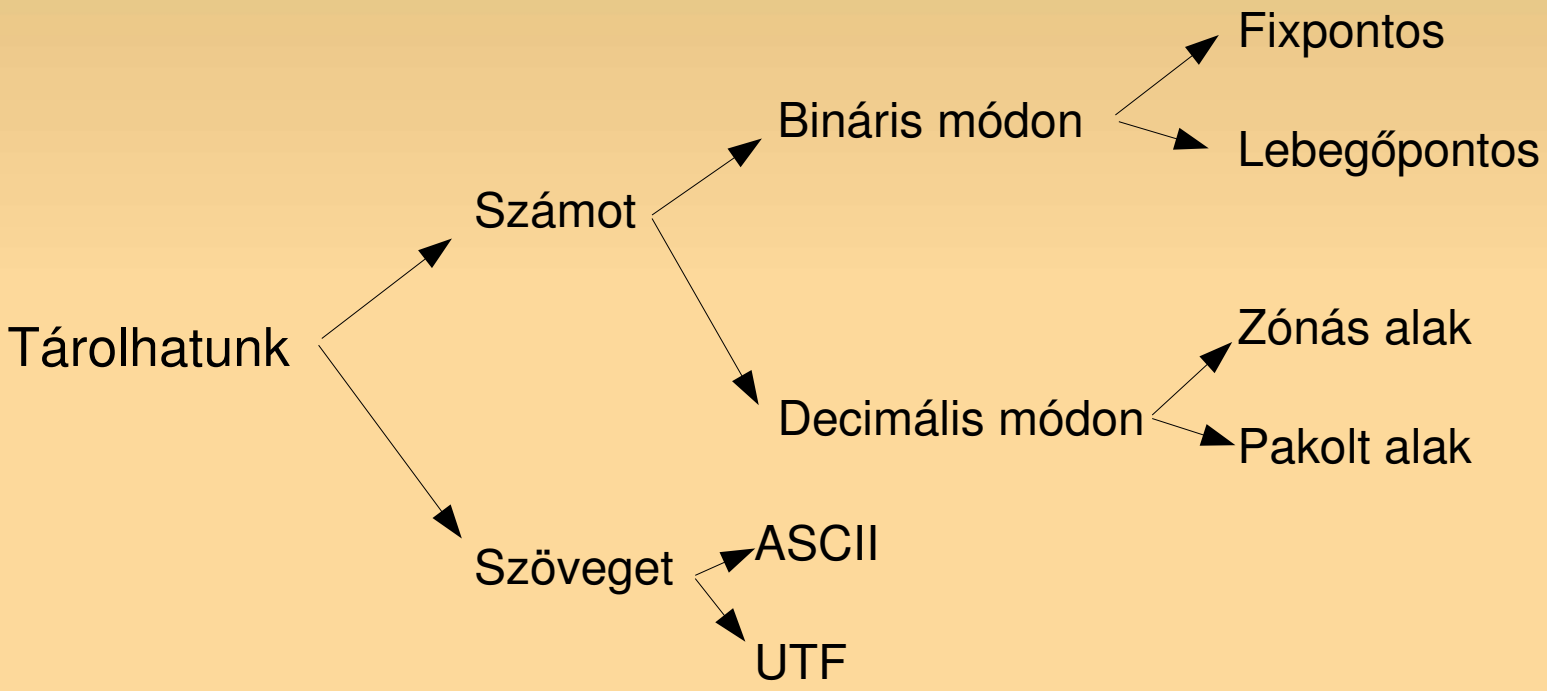
Prefixumok az informatikában

http://hu.wikipedia.org/wiki/Bin%C3%A1ris_prefixum

ubuntu

Adatábrázolás számítógépen

Minden adat 0 és 1 -ből álló bitsorozat.



Fixpontos számábrázolás

Valójában egészszámábrázolás

- bináris pont helye általában a szám után
  - elképzelhető máshol is, de fixen (külön nem jelöljük)

Előjel nélküli egész számok (természetes számok) ábrázolása

Eljárás: a számot átalakítjuk kettes számrendszerbe, és ha szükséges, nullákkal kiegészítjük egész számú bájt(ok)ra.

Ábrázoljuk az 53 tízes számrendszerbeli számot a szükséges (lehető legkevesebb) számú bájton!

Először átváltjuk a számot kettes számrendszerbe:

$$53_{(10)}=110101_{(2)}$$

Mivel ez csak hat darab bitet eredményezett, két nullával ki kell egészíteni, hogy nyolcbites egységet, azaz teljes bájtot kapjunk. Tehát az 53 előjel nélküli szám 1 bájton ábrázolva így néz ki:

00110101

Az eddigiekből következik, hogy 1 bájton a 00000000-tól 11111111-ig a létező összes lehetséges variációban 2^8=256 féle szám ábrázolható. A legkisebb értéke ezek közül a 0, a legnagyobb pedig 255.

Összefoglalva: n db biten 2^n féle szám ábrázolható, a legnagyobb 2^n-1. De ne felejtsük el, hogy az ábrázolás bájtos egységeken történik, tehát az n értéke csak a 8 többszöröse lehet (8, 16, 32,…).

Ábrázoljuk az 352 tízes számrendszerbeli számot a szükséges (lehető legkevesebb) számú bájton!

Először átváltjuk a számot kettes számrendszerbe:

$$352_{(10)}=101100000_{(2)}$$

Mivel az átváltás kilenc darab bitet eredményezett, ami már több, mint egy bájt, így hét nullával ki kell egészíteni, hogy 16 bites egységet, azaz két teljes bájtot kapjunk. Tehát a 352 előjel nélküli szám 2 bájton ábrázolva így néz ki:

00000001 01100000

Két bájton 2^{16}=65536 féle adat tárolható, a legnagyobb 2^{16}-1=65535. Értéket a 0..65535 tartományban vehet fel.

Pozitív egész számok ábrázolása

Az előjeles számok ábrázolásánál a számérték mellett valamilyen módon le kell tárolni a szám előjelét is. Pozitív számoknál az első bájt első bitjét használjuk fel az előjel tárolására. Ha ide 0-t írunk, akkor ez pozitív számot fog jelenteni. Természetesen ekkor már egy bittel kevesebb marad a szám értékének tárolására.

Eljárás: Az első bitre 0-t írunk, a maradék biteken pedig ábrázoljuk a számot. (Egy bájt esetén 8-1=7 db. bit marad a számnak, két bájtnál 16-1=15 db. és így tovább.)

Ha egy bájton ábrázolunk pozitív számot, akkor az első bitre (előjelbit) 0 kerül, és még marad 7 bit a számnak. Ezen a 2^7-1=127 a legnagyobb ábrázolható szám. Tehát a +127 így néz ki: 01111111.

Ábrázoljuk a +53 tízes számrendszerbeli számot a szükséges (lehető legkevesebb) számú bájton!

Először átváltjuk a számot kettes számrendszerbe:

$$53_{(10)}=110101_{(2)}$$

Egy bájt első bitjére 0-t írunk, marad 7 bit. Mivel átváltás után csak hat darab bitet foglal el a szám, még egy nullával ki kell egészíteni, hogy nyolcbites egységet, azaz teljes bájtot kapjunk. Tehát a +53 szám 1 bájton ábrázolva így néz ki:

00110101

Látszólag az eredmény nem különbözik az előjel nélküli 53 szám bitmintájától, valójában a program a deklarálás miatt tudni fogja, hogy itt az első biten nem lehet értékes számjegy.

Ábrázoljuk a 169 előjel nélküli, és a +169 tízes számrendszerbeli számokat a szükséges (lehető legkevesebb) számú bájton!

Megoldás: Először átváltjuk a számot kettes számrendszerbe:

$$169_{(10)}=10101001_{(2)}$$

Az előjel nélküli szám esetén már készen is vagyunk, hiszen a 8 bitet most nem kell kiegészíteni, teljes a bájt:

10101001

A +169 ábrázolásához most viszont egy bájton az első bitre nem tudunk 0-t írni (az előző példával ellentétben), mert akkor elveszítenénk az ott lévő értékes 1-est. Tehát újabb bájttal kell kiegészíteni az eredményt. Ezért a +169 szám ábrázolásához legalább 2 bájt kell, ahol az első 0-át, azaz a + előjelet 7 kitöltő 0

követi, és utána a szám első értékes jegye már a második bájton található:

00000000 10101001

Hexadecimális alakja: 0 0 A 9

Két bájton 2^{15}-1=32767 lesz az ábrázolható maximum: 01111111 11111111, ami a +32767-nek felel meg, mert a 16 bitből az elsőn az előjelet jelző 0 lesz, és marad 15 bit a számnak, amelyekre a maximális értékhez mindenhová 1-et kell írnunk.

Ábrázoljuk a +4573 tízes számrendszerbeli számot a szükséges (lehető legkevesebb) számú bájton!

Negatív egész számok ábrázolása

A negatív szám ábrázolása azonban nem csak az előjelben különbözik a pozitív számoktól hanem az ún. 2-es **komplemens kód**ban.

Komplemensképzés

pl. 3 pozición 10-esben

a legnagyobb szám: 999

Vegyünk egy tetszőleges számot: 354

vonjuk ki 999- ből: 645

354 kilences komplementre (kiegészítője) **645**

<u>Általában:</u> *p* alapú számrendszerben egy szám *(p-1)*-es alapú komplementere az a szám amely minden helyiértékben *(p-1)*-re egészíti ki a számot.

Kettes számrendszerben egy szám 1-es komplementere minden helyiértéken 1-re egészíti ki a számot. (eljárás: minden biten 0->1 ill. 1->0)

pl. 0 1 1 1 0 0 0 1

1 0 0 0 1 1 1 0

ubuntu

Komplemensképzés folyt.

Ha egy szám (p-1) -es komplementeréhez 1-et hozzáadunk p-es komplementert kapunk.

pl. 354 9-es komplementere 645

10-es komplementere 646

kettes komplemens

Negatív számokat leggyakrabban a kettes komplemens képzésével ábrázolunk.

- **A képzés lépései:**
- Tekintsük a szám abszolút értékének kettes számrendszerbeli alakját
- A szám elejére írjunk annyi 0-t, hogy az kiegészüljön annyi jegyűre, mennyi jegyen ábrázolunk
- Az így kapott szám minden bináris jegyét váltsuk az ellenkezőjére
  - /0-át 1-re, 1-et 0-ra/
- Adjunk hozzá a számhoz egyet.

Ábrázoljuk a -53 tízes számrendszerbeli számot a szükséges (lehető legkevesebb) számú bájton!

Először képezzük a -53 abszolút értékét:

$$|-53| = +53.$$

Átváltjuk a számot kettes számrendszerbe:

$$53_{(10)}=110101_{(2)}$$

Egy bájt első bitjére 0-t írunk, marad 7 bit. Mivel átváltás után csak hat darab bitet foglal el a szám, még egy nullával ki kell egészíteni, hogy nyolcbites egységet, azaz teljes bájtot kapjunk. Tehát a +53 szám 1 bájton ábrázolva így néz ki:

00110101

Ezután meghatározzuk a szám komplemensét:

11001010

Ehhez még egyet hozzá kell adni, hogy megkapjuk a kettes komplemenst:

11001010+1=11001011

Tehát a -53 szám a memóriában egy bájton így néz ki:

11001011

Feladat: Ábrázoljuk 1 bájton a -131-et.

A legnagyobbat 01111111 bitminta a +127 értéknek felel meg.

Induljunk ki ebből, hiszen ennek komplemensét, és a kettes komplemensét könnyen meg tudjuk határozni, ami a -127 érték lesz: 10000000+1=10000001.

Ez tehát a -127 bitmintája, amin viszont látszik, hogy még eggyel csökkenthetjük az értékét úgy, hogy még mindig negatív (1-el kezdődő) bitmintánk legyen: 10000001-1=10000000. Ez tehát a -127-1=-128 érték lesz.

- két bájton -32768-tól +32767-ig
- négy bájton -2147483648..+2147483647 -ig ábrázolhatunk előjeles egész számokat

Kettes komplemens kód használatával nincs szükség kivonásra.

$$a-b=a+(-b)$$

$$-a-b=(-a)+(-b)$$

- Példa: 5 -18 ill. -23-2 (két bájton)
- -30+30

Lebegőpontos számábrázolás

- megnöveli a számtartományt
- lehetővé teszi a törtszámok ábrázolását

Valamely *N* szám **számítástechnikai** normál alakja egy szorzat, amelynek első tényezője 0 és 1 értékek közé esik. A másik tényező a számrendszer alapjának valahányadik hatványa. Például a 351.26 számot így alakíthatjuk át:

$$351.26=0.35126*10^{3}$$

Az első tényező (0.35126) neve mantissza, a 10 kitevője (3) pedig a karakterisztika.

Általános alakban:

$$N=m*a^{k}$$

ahol *N* a szám, *m* a mantissza, *a* a számrendszer alapja, *k* a karakterisztika, és *m*-re teljesül a következő feltétel:

A matematikai normál alak esetén a mantissza értéke tízes számrendszert feltételezve 1 és 10 közé esik. Pl. a fenti szám alakja: $3.5126*10^{2}$.

Törtre normalizálás

A bináris pontot addig toljuk el, amíg a mantissza értéke 1/2 és 1 közötti értékű nem lesz.

Például:

$N_2 = 0{,}00010110012 = 0{,}1011001 \cdot 2^{-3}$

Mivel a 2^{-1} helyértéken lévő bit mindig 1 értékű, ezért a szám eltárolása előtt kiveszik. Ezt **implicitbit**nek hívják. Így a tárolt mantissza (m) értéke:

m: 011001000..

Egészre normalizálás

Ez esetben a normalizált mantissza értéke 1 és 2 közé esik.

Például:

$N_2 = 0{,}0001011001_2 = 1{,}011001 \cdot 2^{-4}$

Itt az egészek helyén áll mindig 1, ezért tárolása szükségtelen. A tárolt mantissza azonos az előzővel:

m: 011001000..

Természetesen műveletvégzés előtt mindkét esetben a nem tárolt biteket vissza kell helyezni, hiszen ellenkező esetben hibás eredményt kapnánk.

A karakterisztikához egy egész számot adnak hozzá, és így tárolják.

Ezt a megoldást **eltolt vagy ofszet** karakterisztikának hívják. Az eltolásra azért van szükség, hogy a karakterisztikát eltoljuk a pozitív számok tartományába, és így nem kell az előjelét ábrázolni. Az eltolás mértékére (d) két megoldás használatos:

$$d = 2^{k-1} - 1$$

$$d = 2^{k-1}$$

ahol k a karakterisztika ábrázolására szánt bitek száma. Így az eltolt karakterisztikát (c)

$$c = E + d$$

összefüggéssel számíthatjuk ki.

Példa

Ábrázoljuk a -1001101.1101 kettes számrendszerbeli számot 4 bájton (4*8=32 biten) oly módon, hogy egy bitet használunk az előjelnek, nyolcat a karakterisztikának, és a többit a mantisszának!

Átalakítás számítástechnikai normál alakba:

$$-1001101.1101=-0.10011011101*2^{7}$$

A szám előjele negatív, tehát az első bit 1-es lesz (a könnyebb érthetőség kedvéért a 32 bitet a feladatban szereplő felosztásban táblázatba foglalva ábrázolva):

előjel 1 bit	karakterisztika 8 bit	mantissza 23 bit

ubuntu

A karakterisztika jelen esetben **7**, de ehhez az eltolt ábrázolás miatt 128-at hozzá kell adni: 7+128=135. Természetesen ezt az értéket kettes számrendszerbe is át kell alakítanunk:

$$135_{(10)}=10000111_{(2)}$$

Most ugyan nem kell, de ha szükséges, nyolc bitre is ki kell egészíteni a kettes számrendszerbeli alakot, és utána már beírhatjuk a helyére:

Ezután következik a mantissza ’maradó’ része. Azért nevezzük így, mert nem a teljes mantissza kerül letárolásra, hanem csak az elején lévő 0.1 érték elhagyása utáni:

ubuntu

Azért hagyható el a 0.1 a szám elejéről, mert minden szám normál alakja így kezdődik, így ezt fölösleges lenne letárolnunk, de természetesen a számmal végzett műveleteknél ezt a 0.1-et visszakapja. Viszont még mindig nem vagyunk készen, mert a 32 bitből 1 bitet az előjelre, 8-at a karakterisztikára használtunk fel, és a maradék 23 bitből most csak 10-et foglalt le a mantissza maradó része. Ezért a további bitekre 13 db. 0-t kell írnunk:

A bitminta tehát bájtos csoportosításban:

11000011 10011011 10100000 00000000

IEEE Standard 754 Floating Point

- 1977-ben kezdték a kidolgozását, 1985 -ben jelent meg
- - célja: architektúrák között az adatszintű kompatibilitásnak megteremtése
- - minden architektúrából összegyűjtötték a legjobb megoldásokat
- A lebegőpontos számok ábrázolásának egységesítésére született az ANSI/IEEE 754 szabvány, amellyet a nagy processzorgyártók (INTEL, MOTOROLA, stb.) is használnak. Ez a szabvány háromféle lebegőpontos formát ír elő:
- egyszeres pontosság 32 bit
- dupla pontosság 64 bit
- kiterjesztett pontosság 80 bit

Short real (32 bit)

A mantissza explicit bites egyes normalizált: m = 1.F

Bitértékek: s = 1 bit, k = 8 bit, F = 23 bit, b értéke: 127

Ábrázolandó számtartomány:

$$8,43 \times 10^{-37} < |N| < 3,37 \times 10^{3}$$

Long real (64 bit)

A mantissza explicit bites egyes normalizált: m = 1.F

- Bitértékek: s = 1 bit, k = 11 bit, F = 52 bit, b értéke: 1023
- Ábrázolandó számtartomány:
- $$4,19 \times 10^{-307} < |N| < 1,67 \times 10^{308}$$

kiegészítés: m = 1.F, ahol az egyes helyiértéken lévő egyest nem tartalmazza a mantissza (mivel értéke egyértelmű), csak az adott szám tört része van tárolva.

temporary real (80 bit)

A mantissza explicit bites egyes normalizált: **m = 1.F**

Bitértékek: **s = 1** bit, **k =15** bit, **F = 64** bit, **b értéke: 16383**

Ábrázolandó számtartomány: $3{,}4 \cdot 10^{-4932} < |N| < 1{,}2 \cdot 10^{4932}$

quad real (128 bit)

A mantissza explicit bites egyes normalizált: **m = 1.F**

Bitértékek: **s = 1** bit, **k =15** bit, **F = 112** bit, **b értéke: 16383**

Ábrázoljuk lebegőpontosan 32 biten a $38,29_{10}$ számot!

$38_{10} = 100110_2$

$0,29_{10} = 0,01001010001111010111000 0_2$

$38,29_{10} = 100110,01001010001111010111000 0_2 = A$

Egyes normalizált alak:

$$A = 1,00110010010100011110101110000 \times 2^5$$

Így a mantissza: (23 bit)

$$m = 00110010010100011110101110000$$

Nézzük a karakterisztikát: (8 bit)

$$k = 5 + 127 = 132_{10} = 10000100_2$$

Az előjelbit: (1 bit) e=0

ubuntu

0	10000100	00110010010100011110101

Hexadecimális számrendszerbe is felírhatjuk:

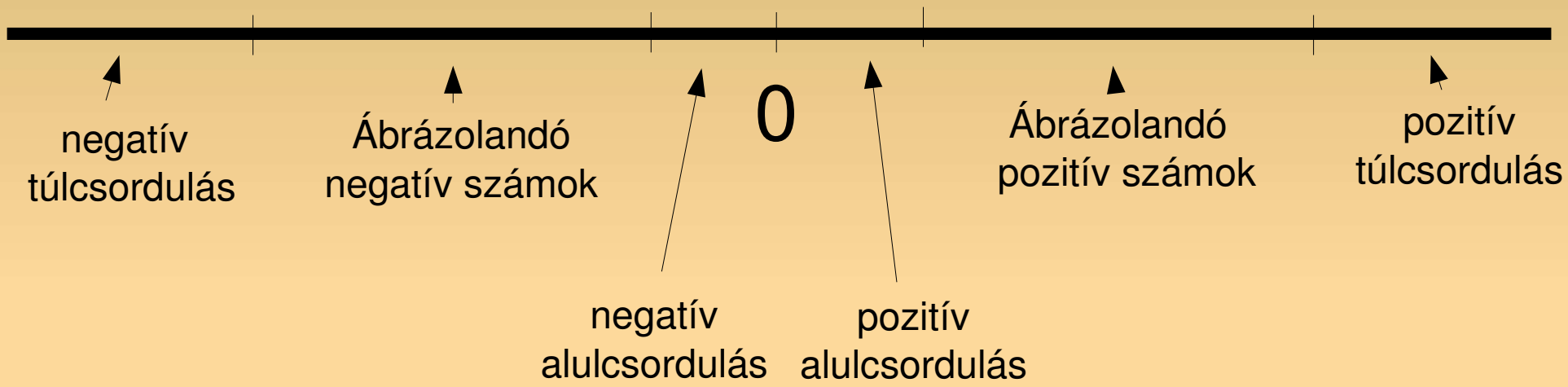
$0100\ 0010\ 0001\ 1001\ 0010\ 1000\ 1111\ 0101_{2}$

$N: 421928F5_{16}$

Természetesen ez nem azonos az N szám tizenhatos számrendszerbeli értékével,

ezért nem használtunk egyenlőségjelet.

Túl- és alulcsordulás kezdeti érték, nulla érték problémája



Az IEEE 754/1985 szabvány speciális lebegőpontos számformái: **Megnevezés**	**Előjel**	**Kitevőrész**	**Mantisszarész**
Normalizált érték	s	Tetszőleges kitevő	Tetszőleges számérték
Denormalizált érték	s	0	Nem nulla számérték
Nulla	s	0	0
Negatív végtelen	1	111…111	0
Pozitív végtelen	0	111…111	0
Jelző 'nem szám' (s NaN)	x	111…111	Nem nulla számérték
Egyszerű 'nem szám' (s NaN)	x	111…111	Nem nulla számérték

http://www6.uniovi.es/~antonio/uned/ieee754/IEEE-754.html

Decimális számok ábrázolása

A bináris ábrázolás előnye a gyors aritmetikai művelet. Sok esetben kell a számokat beolvasni, nyomtatni megjeleníteni (I/O) művelet, és kevesebb az egyéb művelet.

Ilyenkor célszerűbb természetes, decimális formában tárolni (a bináris alakjával).

Általában 1 számjegy 1 bájt.

Két kódrendszer használatos:

- ASCII
- BCD (EBCDIC)

BCD
Binárisan Kódolt Decimális

Az előzőekben láthattuk, hogy a törtszámok kettes számrendszerbe való átváltásakor a legritkább esetben kapunk pontos eredményt. Ha az előbb tárgyalt módokon tároljuk a számokat, ez a nagy pontosságot igénylő számításoknál lényeges lehet. Akár a tizedik helyen álló törtjegy értéke is fontos lehet, miközben már az első tizedes jegy értéke sem pontos.

Erre a problémára adhat megoldást a BCD ábrázolási mód.

Ennél a tárolási módnál nem a számot, hanem a számjegyeket tároljuk.

A tízes alapú számrendszerben 0-tól 9-ig vannak számjegyek, azaz egy-egy számjegyjegy tárolására 4 bit (fél bájt) elégséges.

Ennek a számábrázolási módnak több fajtája van, attól függően, hogy az ASCII vagy az EBCDIC kódtáblára alapul.

Mindkettőn belül van még normál és tömörített ábrázolás is.

ASCII

	0	1	2	3	4	5	6	7	8	9
16	30	31	32	33	34	35	36	37	38	39

BCD (EBCDIC)

	0	1	2	3	4	5	6	7	8	9
16	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9

ASCII

Előjeles ábrázolás

1. byte az előjel $2B_{16}$ + / $2D_{16}$ -

A további byte-okon ASCII kódban a számjegyek.

$3i_{16}$ i=0,1,2,..,9

2	B+ / D-	3	i	..	3	i
0010	1011 1101	0011	i_{16}	..	0011	i

ubuntu

pl. +31 052

-31 052

ubuntu

Zónás ábrázolás

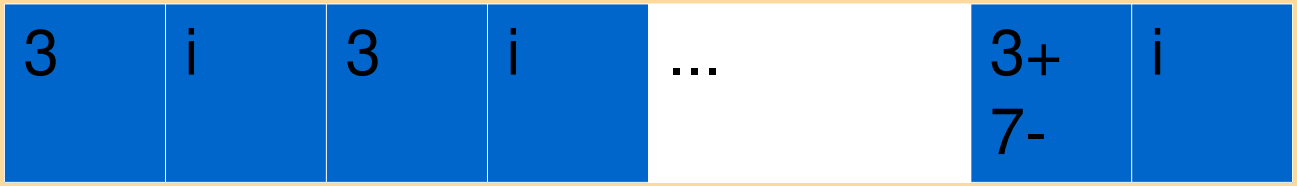
Az előjelet beépítjük az utolsó számjegybe (így az nem foglal el külön byte-ot)

-

az utolsó byte-on lévő előjel és jegy.

Ha $i >= 0$ akkor $3i_{16}$

ha $i <= 0$ akkor $7i_{16}$



ubuntu

pl. +31 052

- 31 052

3	3	3	1	..	3+ 7-	2 2
0011	0011	0011	0001	..	0011 0111	0010

ubuntu

Pakolt tömörített forma

Egy byte-on két számjegy kerül ábrázolásra, és az előjel az utolsó félbájtra kerül.

+ helyett C

- helyett D

Ha a számjegyek száma páros, akkor a szám elé egy 0-át kell írni, hogy az ábrázolás az előjellel együtt egész bájtokon képződjön.

Ezzel a formával aritmetikai művelet is végezhető.

pl. + 31 052 ==> 31 05 2C

- 31 052 ==> 31 05 2D

- 453107 ==> - 04 53 10 7D

0000	0100	0101	0011	0001	0000	0111	1101
0	4	5	3	1	0	7	D

BCD

Binárisan kódolt decimális: (csak számjegyeket tárol)

A hexadecimális számjegyeket 4 biten ábrázolja, tetrádokká képezi.

HEX	BIN
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Példa:

$123=76_{16}$=0111 0110 tetrádok

ubuntu

BCD (EBCDIC) kódban

Zónás forma

Az utolsó byte-on

+i helyett Ci_{16}

-i helyett Di_{16}

a többi helyen i helyett Fi_{16}

+31 052

F3 F1 F0 F5 C2

1111 0011 1111 0001 1111 0000 1111 1001 1100 0010

- 31 052 ==> F3 F1 F0 F5 D2

Pakolt alak (aritmetikai művelet itt is végezhető)

- 453107 ==> - 04 53 10 7D

0000	0100	0101	0011	0001	0000	0111	1100 1101
0	4	5	3	1	0	7	+C -D

vagy i

Összeadás NBCD-ben

ubuntu

Szöveges információ ábrázolása

A szöveges információ karakterekből áll. A *karakter* a kódolásra használt jelrendszer legkisebb (tovább nem bontható) eleme. A karakterekhez egyértelműen rendelünk egy-egy számot, amely majd a memóriában az adott karaktert képviseli (reprezentálja). A karakterek kódolására különböző szabványok születtek.

- ASCII
- UNCODE

ASCII

Az ASCII (American Standard Code for Information Interchange) egy bájton ábrázolja a karaktereket. Mivel 1 bájton 256 féle szám tárolható, így 256 kód áll rendelkezésünkre. Ebből az első 128 jel az ún. standard ASCII, amely mindig ugyanaz. Ebben az első 32 jel vezérlőkód (pl. lapdobás, enter, escape), továbbá az angol ábécé nagy- és kisbetűi, számjegyek, írásjelek és egyéb jelek találhatók benne:

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

A hétköznapi életben, és matematikában is használt műveleti jelek nevei mindenki számára ismertek (pl.: ’*’ – szorzás, ’+’ – összeadás, ’<’ – kisebb, ’=’ – egyenlő stb.), de gyakran megesik, hogy az egyéb jelek neveit nem ismerik. Ezek közül pl. a számítástechnikában a gyakrabban használtak nevei (és fantázianevei): ’#’ – Hash-mark (kettős kereszt), ’@’ – At-mark (kukac, printkukac, farkincás a), ’~’ – Tilde (kishullám, tilda), ’$’ – Dollárjel, ’\’ – Back slash (vissza perjel), ’&’ – Ampersand, And-mark (és jel).

A másik 128 féle jel tartalma *kódlap*októl (*code page*) függően változik. Pl.: az amerikaiak a 437-es kódlapot használják, a magyarok pedig a 852 jelűt. Ez utóbbiban vannak például az ékezetes karaktereink is. A karaktereket a billentyűzeten, az ’*Alt* + *kód*’ kombinációval is előállíthatjuk. (A kódot a numerikus billentyűzeten kell megadni!)

Karakter tárolása

Az ’A’ betű kódja az ASCII kódrendszerben 65, ezért az ’A’ betűnek fenntartott memóriarekeszben a 01000001 bitminta található, mert ez a 65 tízes számrendszerbeli szám kettes számrendszerbe átváltott alakja. Néha tizenhatos (hexadecimális) alakban találkozunk a kódokkal.

$$65_{(10)}=01000001_{(2}=41_{(16)}$$

Unicode Transformation Format UTF 8 ill. 16

Mi a UNICODE?

A Unicode a különböző írásrendszerek egységes kódolását és használatát leíró nemzetközi szabvány. A Unicode nem csak a kódolással, hanem a karakterek osztályozásával, megjelenítésével és használatával is részletesen foglalkozik.

ubuntu

A nemzetköziséget is egyre jobban támogató szoftverek kifejlesztése során teljesen egyértelműen kiderült, hogy a karakterkészletek tarkasága a számítástechnika fejlődésének egyik hatalmas zsákutcája, hiszen nem képes kielégítő megoldást nyújtani számtalan problémára. A megoldást csakis egy olyan kódolás nyújthatja, amely egymagában képes az összes nyelv összes karakterét ábrázolni.

- Meg is született, sőt, állandóan fejlődésben van egy ilyen kódkészlet, a Unicode, UCS (Universal Character Set), avagy ISO-10646 szabvány.

Nyilván ez csak úgy lehetséges, ha átlépjük a 256-os határt, vagyis azt mondjuk, hogy minden karakternek megfeleltetünk egy egyedi pozitív egész azonosító számot, ami (szinte) tetszőlegesen nagy lehet. Eleinte még úgy képzelték, hogy 216 (65536) elég lesz, de később letettek erről, és most a bombabiztos 231 (bő kétmilliárd) az elvi határ, ugyanakkor a becslések szerint 221 (bő kétmillió) fölé nem fognak eljutni a számokkal, ennyi jócskán elég lesz az összes elő, halott és mesterséges kultúra írásjeleinek ábrázolására.

Az összes korábbi 8 bites kódkészletben megtalálható karakter belefért a Unicode kezdeti alsó 65536-os tartományába, amelyet Basic Multilingual Plane-nek (BMP) is neveznek.

Az alsó 128 érték megegyezik a hagyományos ASCII-val. Sőt, az alsó 256 megegyezik a Latin-1-gyel. A magyar ő és ű betűk tehát 256-nál nagyobb azonosítót kaptak. Önálló azonosítót kapott minden egyes írásjel, melyekkel például a jelen leírás nulladik fejezetében találkoztunk, így például van alsó 99-es idézőjel, van felső 99-es, felső 66-os stb., mind-mind különböző azonosítóval.

A Unicode értékeket általában hexadecimálisan, nagy ritkán decimálisan adjuk meg. Sokszor U+ bevezetés után írjuk le a hexa értéket legalább 4 számjegyen (itt látszik még a régi idők szele, amikor úgy képzelték, ennyi elég lesz). Például az ó betű kódja U+00F3 (decimális 243), az ő betűé pedig U+0151 (decimális 337).

A Unicode egy ennél sokkalta bonyolultabb szabvány. A különféle egzotikusabb betűírásokon (cirill, héber, arab stb.) túl tartalmazza a kínai, japán, koreai (ezeket együtt szokták angolul CJK-nak rövidíteni) írásjeleket, és számos vezérlő karaktert, melyekkel például a jobbról balra írás kapcsolható be és ki, vagy éppen a sortörés lehetséges helyei adhatók meg.

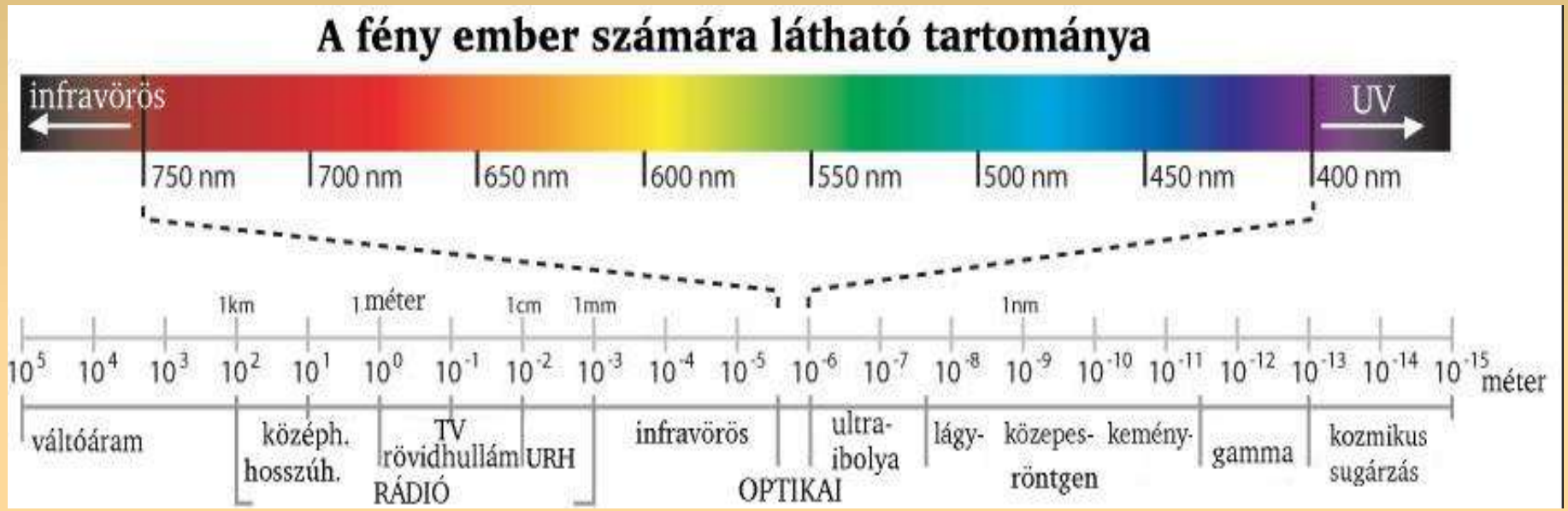
Unicode szabvány UTF-8 ábrázolási módja

Az UTF-8 ábrázolási mód a következőképpen fest: Az ASCII karaktereket egy byte-on, önmagukkal reprezentáljuk, ezek tehát 128 alatti értékű byte-ok. A 128-nál nagyobb vagy egyenlő kódú Unicode karaktereket viszont több egymást követő 128-nál nagyobb vagy egyenlő byte ábrázol. Érdekességképpen a táblázat, így kell a biteket átrendezni:

Színábrázolás

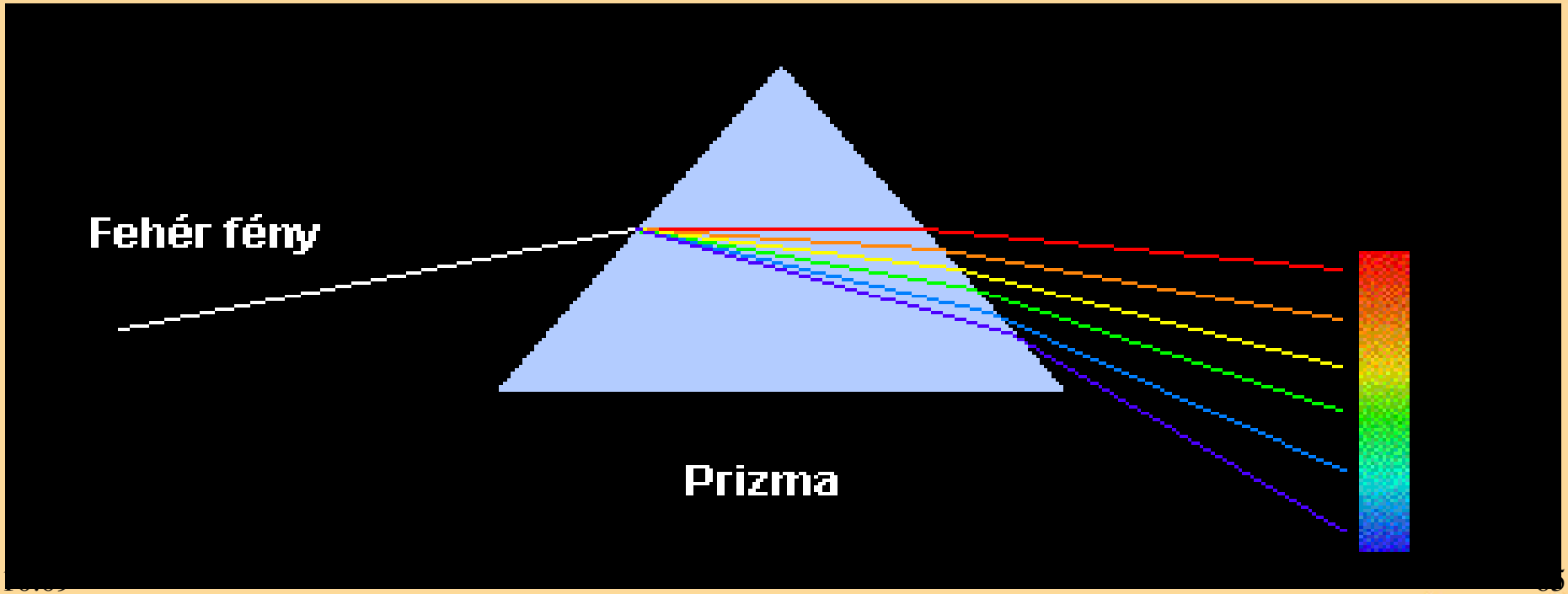
A fény tulajdonságait meghatározó három fő szempont

- **intenzitás vagy amplitudó**, amelyet az ember fényerőként, fényességként érzékel,
- **frekvencia** (és ezzel összefüggésben a hullámhossz), amelyet az ember színként érzékel, és
- **polarizáció**, azaz az elektromágneses rezgés iránya, ezt az átlagember normál körülmények között nem érzékeli, de például bizonyos rovarok igen.


A fény ember számára látható tartománya
infravörös
UV
750 nm
700 nm
650 nm
600 nm
550 nm
500 nm
450 nm
400 nm
1km
1méter
1cm
1mm
1nm
10^{5} 10^{4} 10^{3} 10^{2} 10^{1} 10^{0} 10^{-1} 10^{-2} 10^{-3} 10^{-4} 10^{-5} 10^{-6} 10^{-7} 10^{-8} 10^{-9} 10^{-10} 10^{-11} 10^{-12} 10^{-13} 10^{-14} 10^{-15} méter
váltóáram
középh. hosszúh.
TV rövidhullám URH
RÁDIÓ
infravörös
ultra- ibolya
OPTIKAI
lágy- közepes- kemény-
röntgen
gamma
kozmikus sugárzás

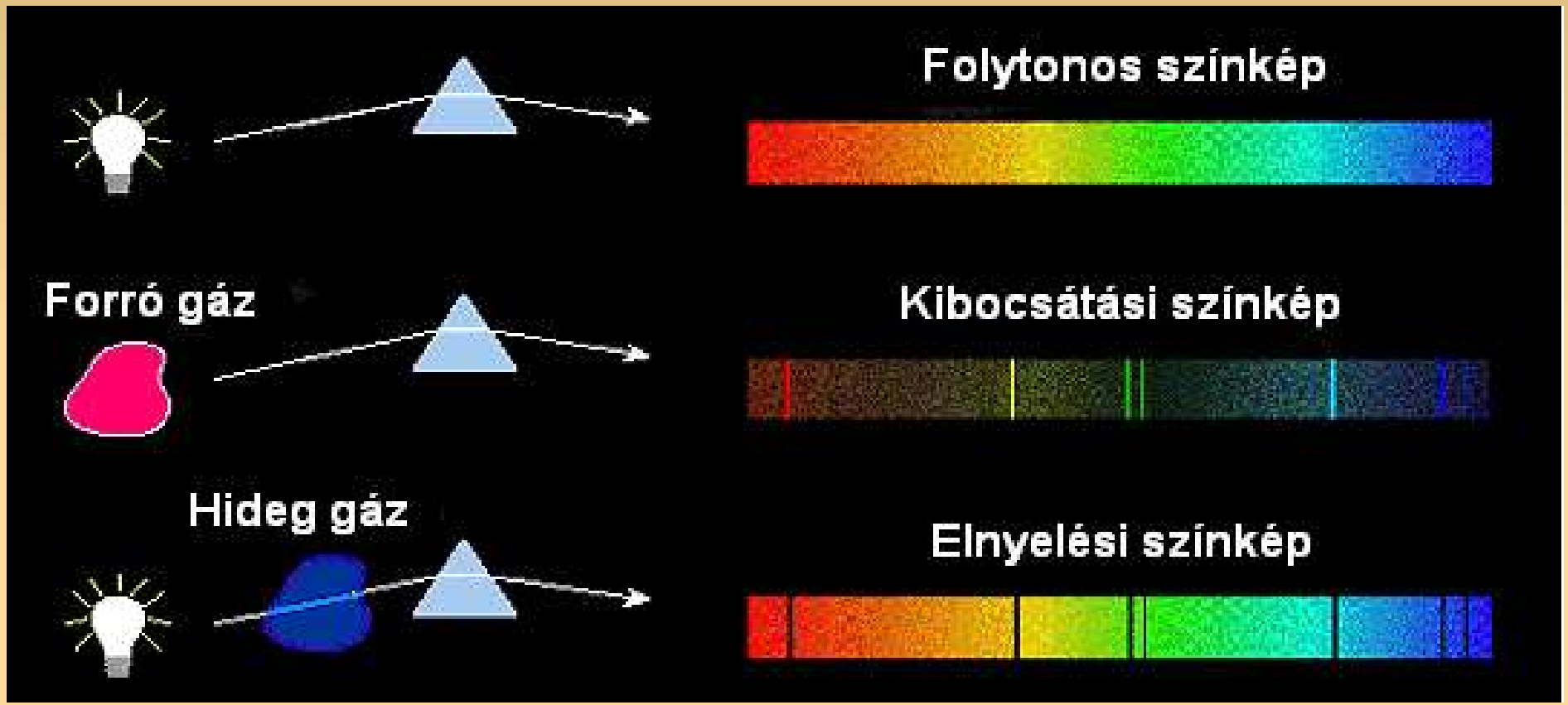
ubuntu

- A fehér fény színei csökkenő hullámhossz szerint rendezve a következők:
- vörös, narancs, sárga, zöld, kék és ibolya.



Három -féle színképet különböztetünk meg:

-
- - folytonos kibocsátási színkép az izzó szilárd anyagokra jellemző
- - vonalas kibocsátási színképet az izzó gázok hoznak létre
- - vonalas elnyelési színkép keletkezik hideg gázokon áthaladó fehér fény esetén


Folytonos színkép
Forró gáz
Kibocsátási színkép
Hideg gáz
Elnyelési színkép

Érzékelés

- 1. SZÍNEZET - melyet hétköznapi életben egyszerűen csak színnek nevezünk, ez a fény hullámhosszától függ. Szemünk kb. 200 féle színezetet tud megkülönböztetni.

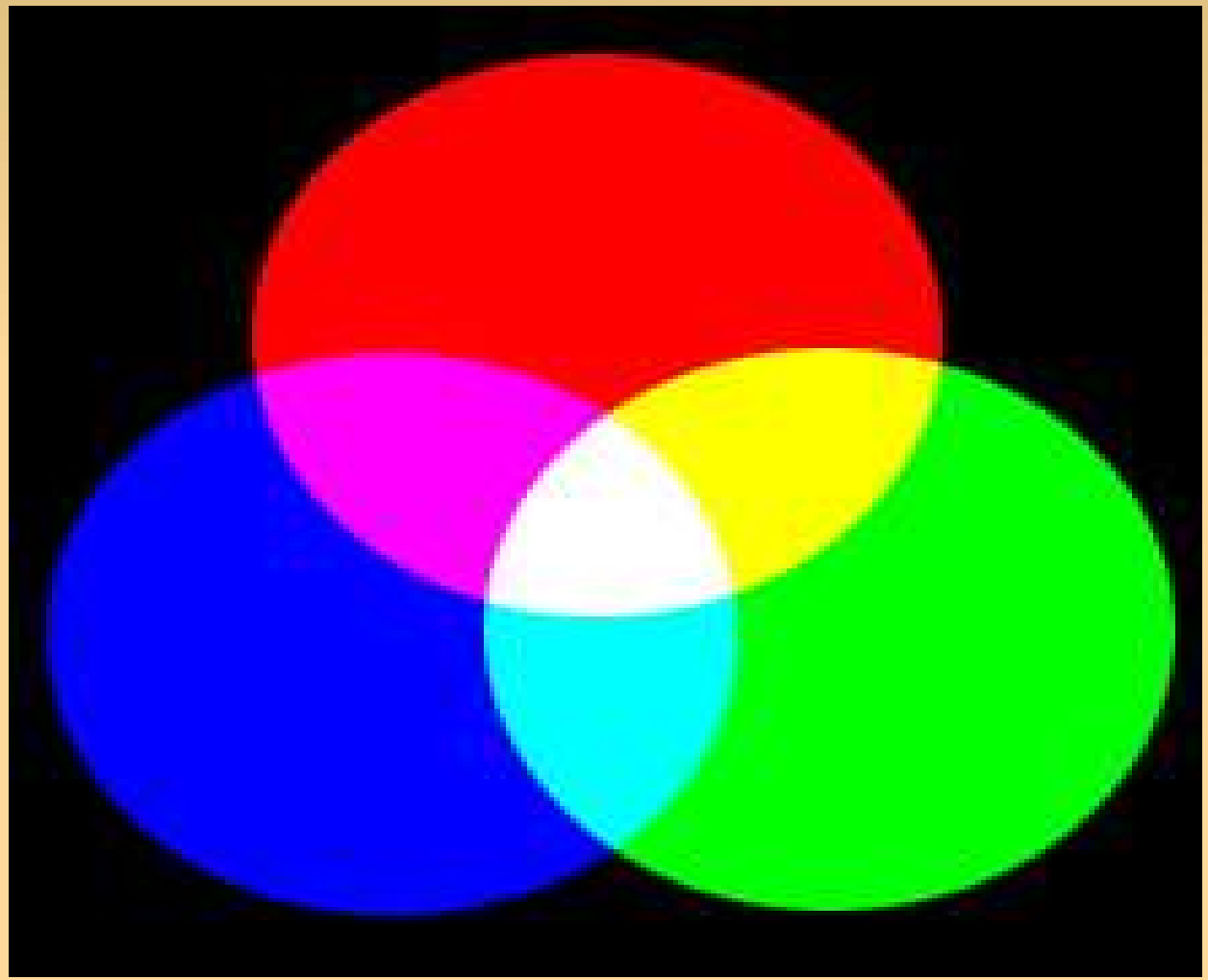
- 2. TELÍTETTSÉG - mely attól függ, hogy mekkora a fehér fény összetevője a többi összetevőhöz képest. Szemünk átlagosan kb. 20 telítettségi fokozatot tud megkülönböztetni.

- 3. VILÁGOSSÁG - mely az egységnyi térszögben a szemünkre érkező fényenergia mennyiségétől függ. A világosság szerinti felbontóképessége a szemnek erősen hullámhosszfüggő, általában 500 fokozatot tudunk megkülönböztetni.

színkeverésnek két módja van

- **additív színkeverés:** a keverékszín monokróm fényforrások fényének egymásra vetítésével áll elő. (pl.: TV) Alapszínei: vörös, zöld, kék.

- RGB (vagy 24 Bit Color): Egy képpont a piros, a kék és a zöld 256-256-256 féle árnyalatából áll össze, összesen 16 millió színárnyalattal. 24 biten tárolja az információt. Ez additív színrendszer, tehát a három alapszín egyforma keverése fehér, hiányuk fekete színt eredményez. Ezeket a színeket használja minden elektronikus kivetítőeszköz (monitor, kivetítő).

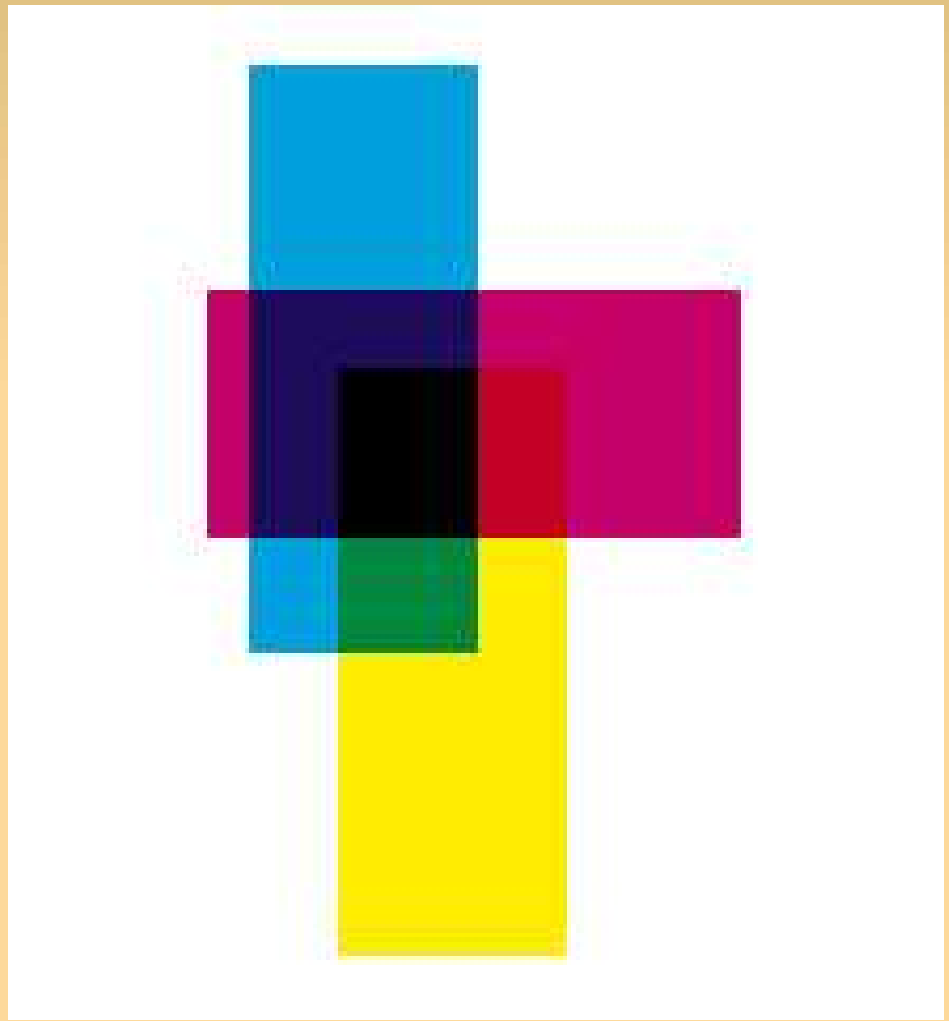


ubuntu

- **szubtraktív színkeverés:** széles sávban sugárzó fényforrás fényéből -kivonunk- egyes hullámhosszokat, tartományokat. A maradék lesz a keverékszín. (pl.: színes nyomtatás) Alapszínei: cián, magenta (bíbor), sárga, fekete (latin nevéből ered a K rövidítés). (**CMYK**) A színeket a feketéből vonjuk ki.

- CMYK (vagy 32 Bit Color): Egy képpont a türkiz (Cyan), a bíbor (Magenta) a sárga (Yellow) (másodlagos alapszínek) és a fekete (blacK) 256*4 féle árnyalatából áll össze. 32 biten (4 byte) tárolja az információt. 4,3 milliárd árnyalata lehet egy képpontnak. A képszerkesztő programokban gyakran 0 és 100 közötti értékek adhatóak meg ezekből a színekből.

- CYM: Ugyanaz, mint az előző, csak fekete nélkül. A fekete nehezen keverhető ki (ezért veszik bele gyakran az alapszínek közé). A CYM alapszínei az RGB alapszíneinek komplemens színei. Ez szubtraktív színrendszer. A színek hiánya fehéret eredményez.



ubuntu

HSB színmegadás

- (Hue – színárnyalat, Saturation – telítettség, Brightness – fényesség): A színárnyalat 0 és 359 közötti értékben egy színt határoz meg a színkeréken, a telítettség és a fényesség megadása százalékban történik. A telítettség jelentése: mennyire keskeny sávot határoz meg a színkerékből. Nagyobb érték esetén a megadott szín távolabbi szomszédai is részt vesznek a szín kikeverésében, a szín pasztell, majd szürkés árnyalatú lesz. Minimális érték esetén a szín "tiszta".

HLS

- A HLS színmodell ugyanezeket az értékeket tartalmazza, csak a fényesség helyett a L – Luminancia szerepel. Mindegyik 0 és 255 közötti értéket vehet fel, vagy az előzőhöz hasonlóan a színárnyalat 0 és 359 között, a másik kettő százalékban adható meg

Kifejezések kiértékelése. Műveletek precendenciája, balról jobbra szabály.

Egy operandusú (unáris) műveletek

a) +: **A változatlanul hagyás jele** (nem mindig előjel, mert egyes típusok nem tárolják az előjelet).

b) -: **Kettes komplemensképzés jele** (amennyiben előjeles típus, akkor a -1-szerese, azaz az ellentettje).

c) **NOT: Bitenként 1-es komplemensképzés jele** (átbillentés). Csak fixpontosan ábrázolt operandusokra használható.

Végezzük el az alábbi műveletet 1 byte-on előjelesen:

-5+(-3)+NOT3

5=00000101

-5=11111011

3=00000011

-3=11111101

NOT3=11111100

-5+(-3)=1|11111000

(overflow)

-5+(-3)+NOT3=1|11110100

Két operandusú multiplikatív műveletek

a) *: Szorzás. Fixpontos és lebegőpontos típusokra is alkalmazható. Az eredmény akkor fixpontos, ha mindkét operandus (tényező) fixpontos, különben lebegőpontos.

b) /: Osztás. 0-val a gép sem tud osztani. Fixpontos és lebegőpontos típusokra is alkalmazható. Az eredmény mindig lebegőpontos.

c) DIV: (Valamiféle) Egészrészű osztás – egy szám egészrésze az a legnagyobb egész szám, amelyik még nem nagyobb a számnál (pl. 0,8–>0, -1,7–>-2). A DIV B=sign(A/B)*[|A|/|B|]

Sign(x)={+1, ha x>0, 0, ha x=0, -1, ha x<0). A és B is csak fixpontos lehet.

Példa

9DIV2

9DIV2=+1*[4,5]=+4

Példa

-7DIV2

-7DIV2=-1*[3,5]=-3

Példa

7DIV(-2)

7DIV(-2)=-1*[3,5]=-3

d) MOD: A DIV maradéka. A MOD B=A-(A DIV B*B). A és B is csak fixpontos lehet.

7MOD(-2)=7-(-3)*(-2)=+1

-7MOD2=-7-(-3)*2=-1

e) **AND:** Az eredmény akkor 1, ha mindkét szám 1. Csak fixpontos típusokra alkalmazható. :

f) **SHL**: Bitenkénti balra tolás (shift left). ASHLB esetén B-vel eltoljuk az A bitjeit balra. Elől lehet túlcsordulás, jobbról 0-ákkal kell feltölteni az A-t. Az ASHL1=A*2, ha nincs túlcsordulás, különben hibás eredmény lesz. A és B is csak fixpontos lehet.

5SHL2

00000101–>00010100

g) **SHR:** Bitenkénti jobbra tolás (shift right). ASHRB esetén B-vel eltoljuk az A bitjeit jobbra. Hátul lehet túlcsordulás, balról 0-ákkal kell feltölteni az A-t. Az ASHR1=A/2, ha nincs túlcsordulás, különben ADIV2 (egészrészű osztás) lesz – mindig csak egész szám lesz az eredmény. A és B is csak fixpontos lehet.

5SHR2

00000101–>00000001

Két operandusú additív műveletek

a) +: Összeadás. Fixpontos és lebegőpontos típusokra is alkalmazható. Az eredmény akkor fixpontos, ha mindkét operandus (tag) fixpontos, különben lebegőpontos.

b) -: Kivonás. Fixpontos és lebegőpontos típusokra is alkalmazható. Az eredmény akkor fixpontos, ha mindkét operandus (kisebbítendő és kivonandó) fixpontos, különben lebegőpontos.

c) OR: Az eredmény akkor 1, ha vagy az egyik, vagy mindkét szám 1. Csak fixpontos típusokra alkalmazható.

Aritmetikai kifejezések kiértékelés

Aritmetikai operandusokból és aritmetikai operációkból áll az aritmetikai kifejezés.

Egy programnyelv kifejezés-számításának lépései:

1. Ha kell, kiszámolja az operandusokat (pl. függvényeket).

2. Kiszámolja a műveleteket a precedencia-szabály szerint

- unáris,
- multiplikatív,
- additív.

A számtani műveletek prioritása

- hatványozás,gyökvonás
- szorzás, osztás
- összeadás, kivonás

Amennyiben azonos precedenciájú műveletek vannak, akkor a balról jobbra felírt sorrendben számítja ki a műveleteket. Zárójellel befolyásolhatjuk a számítás sorrendjét.

Ha vannak, kiértékeli a relációkat (összehasonlító műveleteket). Minden reláció értéke logikai érték.

Ha a műveletek sorrendjét befolyásolni akarjuk, akkor zárójelezhetünk. A programnyelvek redundáns (feleslegesen ismétlődő, terjengő) zárójeleket is megengednek, de ilyenkor operandusoknak nézi, így lassítja a számítást.