Számrendszerek
Kódrendszerek

http://hazam.eu/

1 Tartalomjegyzék

2 Bevezetés

Kedves olvasóm,

A digitális áramkörök – ahogy szinte minden műszaki rendszer– működése végül is annyiból áll, hogy jeleket kapnak a külvilágból, és jeleket küldenek a külvilág felé.

Hogy ezekkel a jelfeldolgozó rendszerekkel megismerkedhessen, először egy kicsit közelebbről meg kell ismerkedne magukkal a „*jelekkel*”. Ha a jelekkel már kellően barátságos viszonyba került, még egy kitérőt kell tennünk a számrendszerek világába.

Ha a *számrendszereket* is megismertük, kezdhetjük a *kódrendszerek* megismerését (a kombinációs hálózatok ugyanis – mint ezt majd hamarosan látni fogjuk –mind valamiféle kódváltóként működő áramkörök).

A digitális áramkörök elemi műveletvégző egységeinek működése a logikai algebrán alapul, ezért a kombinációs hálózatok kezeléséhez a *logikai algebrát* is segítségül kell hívnia. Ezzel már el is jutott abba az állapotba, mely e modul végső célja. A végső cél, hogy *megbarátkozzon* az egyszerű kombinációs hálózatokkal.

Ha ezt eléri, két dologra lesz képes:

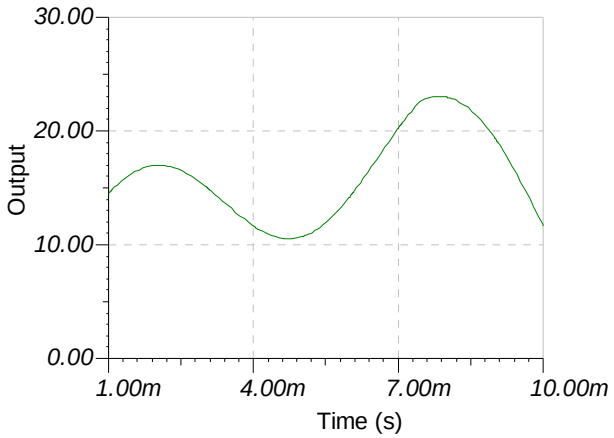
1. Képes lesz egy adott kombinációs hálózat *működését értelmezni*, vagy akár fordítva:
2. Egy adott működést megvalósító optimális kombinációs hálózatot képes lesz *megtervezni*.

Kalandra föl!

3 Jelfeldolgozási alapismeretek

A „jel” fogalmán általában valamilyen jellemző (súly, távolság, hőmérséklet, …) értékének vagy értékváltozásának nagyságát értjük. Valaminek a függését valami mástól.

A valós világban szinte mindig olyan jelekkel találkozunk, melyek az időtől függnek. Ha egy jel *időben és nagyságát tekintve is folytonos*, akkor **analóg** jelnek nevezzük.



3-1. ábra Analóg jel

A digitális rendszerek működését manapság időben és nagyságát tekintve is „darabos”, azaz nem folytonos értelmezési tartományú és értékkészletű– tehát mindkét értelemben diszkrét – jelek jellemzik. Egy analóg jel digitálissá alakítását két lépésben tudunk elvégezni.

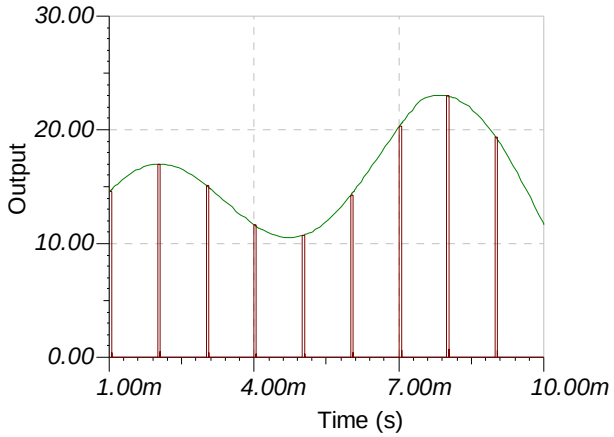
Először darabossá tesszük a jelet időben - ezt a műveletet nevezik *mintavételnek*. A mintavett jel már nem mérhető bármikor, de sajnos még felvehet bármilyen értéket.

A jel nagyságának végtelen sok lehetősége számítástechnikai eszközeinkkel sajnos nem kezelhető. Gondoskodunk tehát arról is, hogy a mintavett jel nagysága se vehessen fel bármilyen értéket – ezt a *kvantálással* érhetjük el.

3.1 Mintavétel

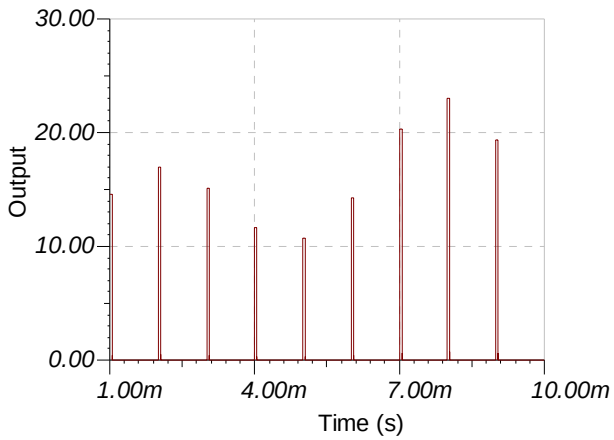
Azt a folyamatot, melynek során *az időben folytonosan változó jelből szakaszos lefolyású, időben „darabos”, diszkrét jelt képzünk*, mintavételnek nevezzük. A mintavétel során a vizsgált jel időben folyamatos jellege elvész, ez azonban nem jelenti azt, hogy az eredeti jel lefolyásáról két szomszédos mintavételi időpont között tudhatunk semmit. Ez első hallásra talán furcsának tűnik, hiszen a mintavétellel egy folyamatosan rendelkezésre álló információforrást szándékosan csak időnként teszünk elérhetővé. Hogy érzékeltesse a mintavétel előnyét, fontolja meg az alábbi példát.

Képzelje magát egy sörgyárba! A kellően finom végtermék létrehozásának érdekében egyszerre több tartályban is kell a készülő nedű hőmérsékletét figyelnie. Biztosan nem lesz szüksége arra, hogy másodpercenként minden egyes tartály hőfokát lekérdezze, hiszen ennyi idő alatt a tartályokban pihenő több hektoliteres folyadékmennyiség hőfoka szinte semmit sem tud változni. A tartályok hőmérsékleti *tehetetlensége* megengedi, hogy csak időnként vegyünk mintát a folyamatból. Általában elmondható, hogy a lehető legkisebb mintavételi frekvencia környezetében érdemes működtetni a mintavevő rendszereket, hiszen minden mintavétel költséggel jár.



3-2. ábra Analóg jel mintavételezése

3-3. ábra



3-4. ábra Mintavett jel

Ha telefonon keresztül beszélgetünk egy ismerősünkkel, akkor nincs szükségünk az audio CD-k esetén használatos, jó hangminőséget biztosító másodpercenkénti 44 056 darab mintára. A hangátvitelhez általában minden nemzet telefontársasága kb. 8 kHz-es mintavételi frekvenciát használ. Ha megelégszünk a kicsit gyengébb minőségű, tompább hangzást eredményező Internetes nemzetközi hívásszolgáltatással is, akkor – természetesen olcsóbb tarifák mellett – a mintavételi gyakoriság lecsökken 6,8 kilominta/sec-ra. Arra azonban vigyázni kell, hogy a vett minták gyakorisága ne legyen túl kicsi. Ekkor ugyanis a mintavett jel már nem állítható vissza megfelelő minőségben. Pl. a beszéd már nehezen érthetővé válik.

Minél gyakoribb a mintavétel, annál pontosabban helyreállítható majd a vett mintákból az eredeti folytonos jel. Nem kell azonban a mintavételi gyakoriságot a végletekig fokoznunk a tökéletes visszaállíthatóság eléréséhez. Meghatározható az a legkisebb mintavételi gyakoriság, ami semmiképpen sem jár információvesztéssel, a gyakorlatban azonban könnyen megeshet, hogy a technológia még ennél is ritkább mintavételt tesz lehetővé. **Shannon tétele** kimondja, hogy egy sávkorlátos, f_{max} maximális frekvenciájú összetevőt is tartalmazó jel mintavételezésének f_S gyakorisága mindig legalább kétszer akkora kell legyen, mint az f_{max} frekvencia. Ebben az esetben az eredeti jel a mintavett értékek sorozatából egyértelműen helyreállítható.

Tömören: a mintavett jel visszaállíthatóságának feltétele:

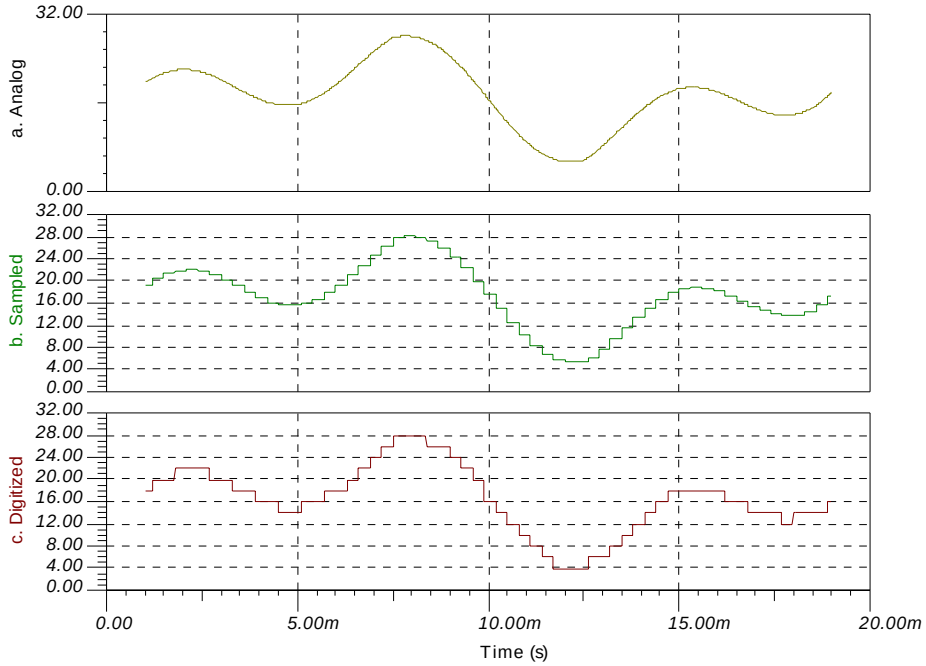
$$f_S \geq f_{max}$$

Az audio CD hangrendszerében azért alkalmazzák a 44 ksps (=kilo-samle-per-secundum) mintavételi gyakoriságot, mert még egy érzékeny fülű ember sem képes a 20 kHz-nél gyakoribb hangrezgéseket (magasabb hangokat) érzékelni. A választott f_s értéke Shannon tétele szerint lehetővé teszi az eredeti hangjel visszaállítását akár a 22 kHz-es komponenseivel együtt is.

A telefóniában azért használnak ennél jóval kisebb mintavételi gyakoriságot, mert nyilván nincs szükség a csengő hangok átvitelére; az emberi beszéd jól érthető akkor is, ha csak a 4 kHz-es és annál kisebb frekvenciájú komponenseit halljuk, sőt az érthetőség akkor is megmarad, ha csupán 3,4 kHz-es tartományig biztosítjuk az átvitelt.

3.2 Kvantálás

A kvantálás során nem a jel időbeni lefolyását módosítjuk; *a mintavett jel nagyságát tesszük „darabossá”*. A kvantálás a folytonos értékű jelből olyan értékfolyamot hoz létre, melynek értékkészlete egymástól elkülönült – más szóval diszkrét – értékekből áll. Az eredmény a digitális jel



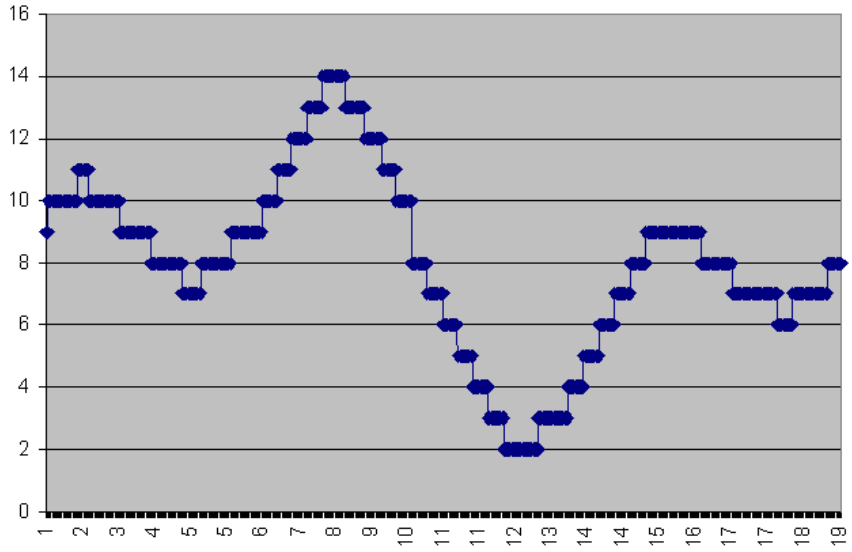
3-5. ábra

A kvantálás során a feladat ismeretében kiválasztjuk az érték feldolgozásához alkalmas *felbontást*. Ha például egy nagy fémtartályban reduktív technológiával készülő bor hőfokát kell megmérnünk, akkor minden bizonnyal felesleges lenne 1/100 °C pontossággal kvantálni a hőmérsékleti adatokat – ugyanakkor az is könnyen belátható, hogy ha csak 10 °C felbontással kvantáljuk a lé hőfokát, akkor lényeges hőfokváltozásokról sem veszünk majd tudomást, és az erjedés túl gyorsan vagy lassan következhet be.

A lineáris kvantálás során a folytonos értékű eredeti jelet összevetjük az ún. kvantummal (q), és meghatározunk azt, hogy a jel hányszorosa a kvantumnak. A kvantálás során tehát a jel nagyságához *egész számokat* rendelünk. A 3-5. ábra példája kvantálást szemléltet, ahol a feszültség-kvantum értéke 2 V. A kvantált jel valójában egy számsor – az ábrán azonban a szemléletesség kedvéért a számsornak a kvantummal megszorzott értéke található. Az egyes számértékek a 3-6. ábra mutatja.

A kvantálás során a jel nagyságának olyan amplitúdójú ingadozása, mely kisebb a kvantum felénél, elveszhet. A hibásan 10 °C-nyira választott kvantum lehetővé tenné a bor +- 5 °C hőmérsékletváltozását anélkül, hogy ezt érzékelnénk.

A kvantált jel helyreállítása gyakorlatilag azt jelenti, hogy a kvantum egész-számszorosait állítjuk elő. Eredményként tehát darabos értékkészletű, hirtelen értékugrásokat tartalmazó jeleket kapunk. A helyreállított jel „simítását” analóg szűrők végzik.



3-6. ábra

3.3 A/D és D/A konverzió

Az analóg-digitális átalakítás során az időben és értékben egyaránt folytonos lefolyású jelből mintákat veszünk, és a vett mintákat kvantáljuk. Ezt a műveletet az analóg-digitális átalakító (analog to digital converter = ADC) áramkörök végzik.

A digitális-analóg átalakítás során a jelet reprezentáló számsort alakítjuk át ismét időben és értékben egyaránt folytonos jellé. Ezt a műveletet a digitális-analóg átalakító (digital to analog converter = ADC) áramkörök végzik.

Az ADC és DAC egységek jellemzőivel, alkalmazásával a Digitális Jelfeldolgozás modul során ismerkedhet meg részletesebben.

A digitális áramkörök – mint ez az előzőekből is látható volt – számértékekkel dolgoznak. A számokat e rendszerek nem a tízes számrendszer használatával kezelik. Ismétlésként tekintse át és egészítse ki a számrendszerekről korábban (a számítástechnikai alapok modulban) tanultakat a következő fejezet áttanulmányozásával.

4 Számrendszerek

4.1 Tízes (DECimális) számrendszer

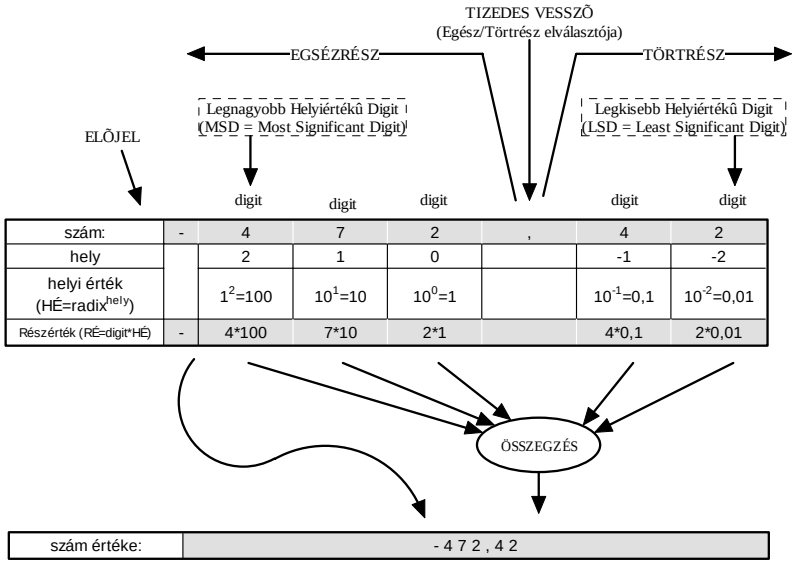
Ezt mindannyian jól ismerjük.

Alapszáma (radix-a) Tíz

Az érvényes számjegyek (digitek) halmaza {0;1;2;3;4;5;6;7;8;9}

4.1.1 Egy decimális szám értelmezése

Egy tetszőleges számrendszerben felírt szám értelmezése azt jelenti, hogy a számot felírjuk tízes számrendszerben. A decimális számok értelmezése során természetesen magukat az értelmezendő számokat kapjuk meg eredményként. A dolognak mégis sok haszna van, mert megismerhetjük a *hely*, a *helyi érték*, a *részérték* és a *számérték* fogalmát.



4-1. ábra

A felírt szám tartalmazza az egész- illetve törtrészt elválasztó szimbólumot. Magyarországon erre a vessző, angol nyelvterületen pedig a pont használatos. Ha ezt az elválasztó szimbólumot nem tartalmazza a felírt szám, akkor a szám jobb szélére képzeljük. (Az angol jelölésmód megengedi azt is, hogy a törteket oly módon ábrázoljuk, hogy a nulla értékű egészrészt nem tüntetjük föl, azaz a *0,42* számérték helyett *.42* is használatos.)

Megjegyzés: A „komoly” programozási nyelvek mindig tizedes pontot használnak.

A tizedes vesszőtől közvetlenül balra lévő digit a nulladik *helyen* található. A tőle balra lévő számjegy az egyes, ettől balra pedig a kettes helyen található és így tovább. A tizedes vesszőtől közvetlenül jobbra lévő számjegy a -1, a tőle jobbra lévő pedig a -2 helyen található, és így tovább. Ez ugyanígy van minden egyes számrendszer esetében.

Az egyes helyekhez tatozó *helyi értékeket* oly módon kapjuk meg, hogy a számrendszer alapszámát a helyedik hatványra emeljük: $HÉ=radix^{Hely}$.

A tízes számrendszer radixa (alapszáma) tíz, ezért az egyes helyiértékek egészrész esetén jobbról balra haladva sorra 1; 10; 100; 1000; …; törtrész esetén pedig (balról jobbra haladva) 0,1; 0,01; 0,001; …

Az egyes digitek a szám értékének egy-egy meghatározott *részértékét* képviselik. A részérték meghatározásához a számjegy saját értékét kell megszoroznunk a szám helye által meghatározott helyi értékkel. RÉ=digit*HÉ .

A *szám értékét* úgy kapjuk meg, hogy az egyes értékrészeket összegezzük. $\boxed{\text{Érték} = \sum \text{RÉ}.}$

A szám értékének *előjelét* minden egyes számrendszerben azonos módon ábrázoljuk

- Pozitív számok elé vagy nem írunk semmit, vagy egy „+” szimbólumot teszünk.
- Negatív számok elé mindig kiteszünk egy „-” szimbólumot.

4.2 Kettes (BINáris) számrendszer

Leginkább ezt használják a digitális áramkörökkel működő rendszerek. A számjegyek leírására csupán két szimbólumot használ. A kettes számrendszert alkotó számjegyeket *bináris digiteknek (bit-eknek)* nevezzük.

Alapszáma (radix-a) Kettő

Az érvényes számjegyek (digitek) halmaza {0;1}

A bináris digiteket *bit*-eknek nevezzük. A legnagyobb helyi értékű bit neve *MSB* (Most Significant Bit). A szabályosan felírt bitsorozatnak tehát bal oldali első bitjét nevezzük MSB-nek. A legkisebb helyiértékű bit angol elnevezésének rövidítése: *LSB* (Least Significant Bit).

Megjegyzés:

- A legtöbb assemblyben a bináris konstansokat egy „b” betűvel jelöljük, amit közvetlenül a szám LSB-je után kell írni. Pl. *10111001B, 111001b*

4.2.1 Egy bináris szám értelmezése [BIN → DEC]

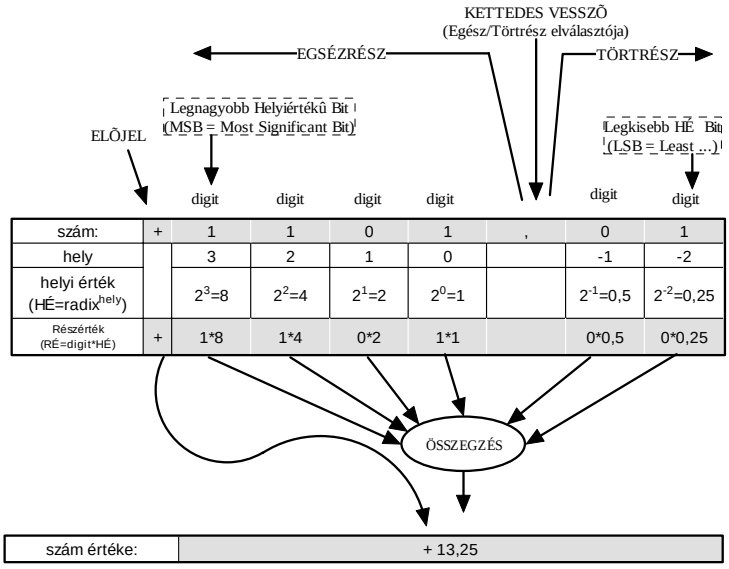
A szám értelmezését a 4-2. ábra mutatja. Ha fejben számolunk, akkor természetesen nem szokás a nullával szorzást elvégezni.

Sokkal praktikusabb módszer, ha minden *egyes értékű* bit alatt vagy fölött feltüntetjük a helyhez tartiozó helyi értéket, majd egyszerűen ezeket a helyiértékeket összegezzük.

Egy példa: az $101011{,}11_{(2)}$ szám értelmezése:

```
  1   0   1   0   1   1  ,  1   1
 32   -   8   -   2   1     0,5 0,25
```

Az eredmény tehát $32+8+2+1=43{,}75_{(10)}$–nek adódik.



4-2. ábra

4.2.2 Szám átalakítása kettes számrtendszerbe [DEC → BIN]

A modul teljesítése során feladata lesz az is, hogy kettes számrendszerben ábrázoljon decimális formában adott értékeket. Ez a művelet viszonylag egyszerűen megvalósítható, ha biztos benne, hogy hiba nélkül tud számokat maradékosan osztogatni kettővel. :-)

A tízes számrendszerben megadott számok egészrészét és törtrészét külön-külön kell átalakítania.

4.2.2.1 Egészek átalakítása

A számot addig osztogatjuk maradékosan 2-vel, amíg az 0-ra le nem fogy. Az osztás eredményét mindig az előző szám alá írjuk, a maradékot pedig egy külön oszlopba, az eredmény mellé. Az eredményt „visszafelé” olvassuk ki, tehát a legutoljára keletkezett maradékbit az MSB.

Pl. $177_{(10)}=?_{(2)}$

```
 177 | :2
-----------          Az eredmény: (…000) 1011 0001(2)
  88 | 1   LSB
  44 | 0     ↑
  22 | 0     |
  11 | 0     |
   5 | 1     |
   2 | 1     |
   1 | 0     |
   0 | 1   MSB
   0 |(0)   ← innnen már már felesleges folytatni!…
   0 |(0)
   0 |(0)
```

4.2.2.2 Törtek átalakítása

A szám törtrészét addig szorozgatjuk 2-vel, mígnem nullát kapunk eredményül. A duplázott értékek egész részei alkotják majd a kettes számrendszerben felírt szám számjegyeit.

Pl. $0,546875_{(10)}=?_{(2)}$

```
    2*  , 546875
    ----------          Az eredmény: 0,1000 111(0 00…)(2)
MSB  1 | 9375
     | 0 | 1875
     | 0 | 375
     | 0 | 75
     ↓ 1 | 5
       1 | 0
LSB    1 | 0
     (0)| 0  ← innnen már már felesleges folytatni!…
     (0)| 0
     (0)| 0
```

Rossz hír, hogy a folyamatnak nem mindig van vége: ha egy nulla és egy közötti szám valamely (pl. tízes) számrendszerben szép kerek formát mutat, az még nem garancia arra, hogy más számrendszerben is hasonlóan, kerek eredményt kapunk. Előfordul, hogy végtelen szakaszos tört keletkezik.

Gondoljon arra, hogy pl. az ⅓-ot tízes számrendszerben 0,333…-nak írjuk, ami egy végtelen hosszú tört, hármas számrendszerben viszont nyilván kereken: $0,1_{(3)}$ lesz!

Tanulság: ha az átalakítás során egy kellemetlen végtelen szakaszos törteket kapjuk, akkor a konverziót abba kell hagyni! Ezt vagy a szakaszosság felismerésekor, vagy a kívánt pontosság elérésekor kell megtennünk.

Egy példa erre: $0,4_{(10)}=?_{(2)}$

```
    2*  , 4
    ----------          Az eredmény: 0,0110 0110 0110 0110 ...(2)
MSB  0 | 8
     1 | 6
     1 | 2
     0 | 4
     0 | 8← innnen már már felesleges folytatni!…
     1 | 6
LSB  1 | 2
     0 | 4
```

4.3 Tizenhatos (HEXadecimális) számrendszer

A tizenhatos számrendszer a lényegében a kettes számrendszerben felírt *bitsorozatok tömörített felírási módjának* tekinthető. Arra használjuk, hogy a viszonylag hosszú bitsorozatokat röviden, áttekinthetően jelenítsük meg.

Alapszáma (radix-a)	Tizenhat
Az érvényes számjegyek (digitek) halmaza	{0;1;2;3;4;5;6;7;8;9;A;B;C;D;E;F} (A = tíz, B = tizenegy, … F = tizenöt)

Megjegyzés: A kilencnél nagyobb értékű hexa digiteket szokás kisbetűkkel is jelölni.

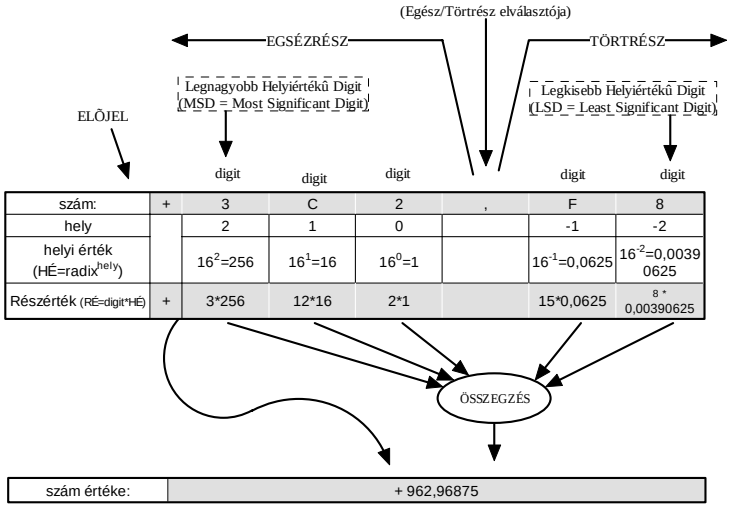
A *programozási nyelvekben* a hexadecimális számkonstansokat két módon szokás jelölni:

- *Bevezető szimbólumnal:* valamilyen különleges karakter vezeti be tizenhatos számrendszerbeli értéket. Turbo Pascal nyelvben: *$c4f2*; C nyelven: *0xc4f2*
- *Lezáró szimbólummal:* a hexa konstansok is decimális számkarakterrel kezdődnek, és a speciális – „h”– lezáró karakterrel fejeződnek be. A legtöbb assemblyben: *0c4f2h* egy hexa konstans (a szám elé írt nullára azért van szükség, hogy a fordítóprogram felismerje, hogy ez egy számkonstans – hiszen a bevezető nulla nélküli „c4f2h” minden programozási nyelven valamilyen objektum (pl. változó) neveként lenne értelmezhető.

4.3.1 Egy hexadecimális szám értelmezése [HEX → DEC]

Az HEX→DEC átalakítást hasonlóan, hajtjuk végre, ahogyan a BIN→DEC konverziót végeztük. (4-3. ábra)

Arra kell csupán figyelnie, hogy most az egyes helyi értékek a hexadecimális vesszőtől egészek felé (balra) elindulva rendre 1; 16; 256; 4096; 65536 … lesz. A törtrészekhez természetesen ezeknek a számoknak a reciprokjait rendeljük helyi értékként: 0,0625; 0,00390625; …



4-3. ábra

4.3.2 Szám átalakítása tizenhatos számrendszerbe [DEC → HEX]

Ez a művelet már nehezebb, mint a kettes számrendszerbe átírás. Csak akkor járhat sikerrel, ha biztos benne, hogy hiba nélkül tud számokat maradékosan osztani illetve osztani tizenhattal. Ez nem is olyan könnyű! Még szerencse, hogy vannak számológépek! ;-)

A számok egészrészét és törtrészét külön-külön kell átalakítania.

4.3.2.1 Egészek átalakítása

A számot addig osztogatjuk maradékosan 16-tal, amíg az 0-ra le nem fogy. Az osztás eredményét mindig az előző szám alá írjuk, a maradékot pedig egy külön oszlopba, az eredmény mellé. Az eredményt „visszafelé” olvassuk ki, tehát a legutoljára keletkezett maradékbit az MSD (Most significant Digit). Most arra is vigyáznia kell, hogy az esetleg decimálisan két digittel ábrázolt (10…15 közötti) maradékokat a nekik megfelelő egyetlen betűszimbólummal helyettesítse!

Pl. $962_{(10)}=?_{(16)}$

```
 962 | :16
---------          Az eredmény: (…000) 3C2(16)
  60 | 2    LSD
   3 | 12 C
   0 | 3    MSD
   0 |(0)   ← innnen már már felesleges folytatni!…
   0 |(0)
   0 |(0)
```

4.3.2.2 Törtek átalakítása

A szám törtrészét addig szorozgatjuk 16-tal, mígnem nullát kapunk eredményül. A szorzás eredményének egészrészeit külön oszlopba leírva megkapjuk a tizenhatos számrendszerben felírt szám számjegyeit. Most is vigyáznia kell, hogy az esetleg decimálisan két digittel ábrázolt (10…15 közötti) egészrészeket a nekik megfelelő egyetlen betűszimbólummal helyettesítse!

Pl. $0{,}96875_{(10)}=?_{(16)}$

```
          16*  , 96875
           ----------          Az eredmény: 0,F8(000…)(16)
MSD  |    F 15 | 5
LSD  ↓       8 | 0
            (0)| 0  ← innnen már már felesleges folytatni!…
            (0)| 0
            (0)| 0
```

Természetesen ilyenkor sem garantált, hogy a törtrész átalakításával véges számú lépésen belül végzünk. :-(

Pl. $0{,}62_{(10)}=?_{(16)}$

```
          16*  , 62
           ----------          Az eredmény: 0,9 EB851 EB851 EB...(16)
MSD          9 | 92
             E 14 | 72
             B 11 | 52
  |          8 | 32
  |          5 | 12
  ↓          1 | 92 ← innnen már már felesleges folytatni!…
             E 14 | 72
             B 11 | 52
```

Szép dolog mindez, – mondhatják, – de ha egyszer a digitális áramkörök működése a *kettes* számrendszeren alapul, akkor miért kell még a *tizenhatos* számrendszerrel is szenvednünk? Nos, a következő részek éppen erre ad választ.

4.4 A kettes és a tizenhatos számrendszer kapcsolata

A kettes és a tizenhatos számrendszer közötti megfeleltetés akkor a legegyszerűbb, ha csupán *egyetlen hexa digithez* kell kettes számrendszerbeli értékeket rendelnünk.

HEX	DEC	BIN 8-4-2-1
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
10	A	1 0 1 0
11	B	1 0 1 1
12	C	1 1 0 0
13	D	1 1 0 1
14	E	1 1 1 0
15	F	1 1 1 1

Ilyenkor a szakemberek a jobb oldalon látható segéd táblázatot használják.

Pl.: $C_{(16)} = 1100_{(2)}$

Megjegyzés: a segédtáblázatot egy szakember *fejből* használja. A bináris bitnégyeseket pillanatok alatt ki lehet találni. Azt kell, eldöntenünk, hogy az ábrázolandó 0 … 15 közötti számérték *bináris részértékei* között szerepelnek –e:

a 8-4-2-1 részértékek.

Ha igen, akkor egy 1-est írunk a megfelelő helyi értékhez, ellenben 0-t.

A kettes és bináris számrendszer kapcsolata más esetekben is hasonlóan egyszerű, olvassa csak el a hex→bin és a bin→hex átalakítás módszerét.

4.4.1 HEX→BIN átalakítás

Az átalakítás most is a jobb oldali segédtáblázat felhasználásával történik. Az egyes hexa digiteket átírjuk bináris *bitnégyesekké*, majd a bitnégyeseket egymáshoz illesztjük. (Az így képzett bitnégyeseket néha *tetrádnak* nevezik, angol elnevezése pedig *nibble* – a szó eredeti jelentése kb.: „egy kis harapás”.)

Az eredmény már kialakul a bitnégyesek egymáshoz illesztésével is, igazán akkor lesz szép, ha az egészrészhez tartozó bitnégyeseket baloldalon bevezető felesleges nullákat töröljük, és ugyanígy járunk el a törtrészt lezáró nullákkal is.

Pl. $-1AB2,4C_{(16)} = ?_{(2)}$

1	A	B	2	,	4	C
~~000~~1	1010	1011	0010	,	0100	11~~00~~

Eredmény: – 1 1010 1011 0010 , 0100 $11_{(2)}$

4.4.2 BIN→HEX átalakítás

Az átalakítás most is az említett segédtáblázat felhasználásával történik. Az egészrészt a törtrésztől elválasztó bináris vesszőtől jobbra és balra elindulva négyes bitcsoportokat alakítunk ki. Szükség esetén az egészrészt balról, a törtrészt pedig jobbról kiegészítjük nullákkal. A kialakított bitnégyesekhez tartozó hexa digiteket egymáshoz illesztve már ki is alakult a szám 16-os számrendszerben felírt alakja.

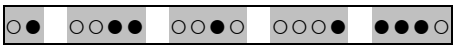
Pl.: + 110 101 011 00 10 , 01 00 $11_{(2)} = ?_{(16)}$

0001	1010	1011	0010	,	0100	1100
1	A	B	2	,	4	C

Eredmény: +1AB2,4C

Régebben a gyakorlatban is sokszor előfordult, hogy ehhez hasonló módon kellett egy számot értelmeznünk. Ha pl. egy 16 bites számértéket kell megjelenítenünk, akkor a legprimitívebb kijelzési mód 16 darab LED használata. (A LEDek világítanak, ha az adott helyiértéken egyes van, nem világítanak, ha 0.) Ilyen megjelenítést soha nem alkalmazhatunk oly módon, hogy a világító diódákat egyszerűen egymás mellé helyezzük – ilyenkor ugyanis a megjelenített érték nagyon nehezen lenne értelmezhető. A példában ● jelöli a világító, ○ pedig az inaktív fényforrást.

– ezt nem könyű értelmezni [hibásan kialakított kejelző!]



– ezt ránézésre is könnyebb felfogni: $1321E_{(16)}$

4.5 Nyolcas(OCTaális) számrendszer

A nyolcas számrendszer használata régebben volt elterjedt, manapság jelentősége egyre csökken. Alapszáma nyolc.

Az oktális számrendszert a binárissal *bithármasok (tripletek)* kapcsolják össze, éppen úgy, mint az előzőekben tárgyalt bitnégyesek a 16-os és a 2-es számrendszereket.

4.6 Gyakorló feladatok

Végezze el az alábbi konverziókat. Ha a törtrész számjegyeit legfeljebb nyolc digites pontosságig határozza meg!

Konverzió	Átalakítandó szám	Eredmény
DEC → BIN	1956	
DEC → BIN	- 1023	
DEC → BIN	0,8	
DEC → BIN	0,546875	
DEC → BIN	- 144,625	

Konverzió	Átalakítandó szám	Eredmény
DEC → HEX	-2000	
DEC → HEX	0,8	
DEC → HEX	65535	
DEC → HEX	+127	
DEC → HEX	-0,667 968 75	

Konverzió	Átalakítandó szám	Eredmény
DEC → OCT	-2001	
DEC → OCT	0,8	
DEC → OCT	65535	
DEC → OCT	+127	
DEC → OCT	-0,667 968 75	

Konverzió	Átalakítandó szám	Eredmény
BIN → HEX	1011 101 00	
BIN → HEX	0,101 111 01	
BIN → HEX	-11 0011 11	
BIN → HEX	0,1110 11	
BIN → HEX	+1010 1010,0101 0101	

Konverzió	Átalakítandó szám	Eredmény
HEX → BIN	ac76	
HEX → BIN	-3c	
HEX → BIN	0,b8	
HEX → BIN	-63,86	
HEX → BIN	+45	

Konverzió	Átalakítandó szám	Eredmény
HEX → DEC	Ac76	
HEX → DEC	-3c	
HEX → DEC	0,b8	
HEX → DEC	-63,86	
HEX → DEC	+45	

Konverzió	Átalakítandó szám	Eredmény
OCT → BIN	765	
OCT → BIN	-137	
OCT → BIN	0,42	
OCT → BIN	-53,4	
OCT → BIN	+45	

Konverzió	Átalakítandó szám	Eredmény
OCT → DEC	765	
OCT → DEC	-137	
OCT → DEC	0,42	
OCT → DEC	-53,4	
OCT → DEC	+45	

4.7 Gyakorló feladatok megoldása

Ellenőrizze az előzőleg megoldott feladatok eredményét! A helyes válaszokat az alábbi táblázatok tartalmazzák.

Konverzió	Átalakítandó szám	Eredmény
DEC → BIN	1956	111 1010 0100
DEC → BIN	- 1023	- 11 1111 1111
DEC → BIN	0,8	0,1100 1100 …
DEC → BIN	0,546875	0,1000 11
DEC → BIN	- 144,625	- 1001 0000,101

Konverzió	Átalakítandó szám	Eredmény
DEC → HEX	-2000	-7d0
DEC → HEX	0,8	0,ccc …
DEC → HEX	65535	ffff
DEC → HEX	+127	+7f
DEC → HEX	-0,667 968 75	-0,ab

Konverzió	Átalakítandó szám	Eredmény
DEC → OCT	-2001	-3721
DEC → OCT	0,8	0,6314 6314 …
DEC → OCT	65535	177777
DEC → OCT	+127	+377
DEC → OCT	-0,667 968 75	-0,526

Konverzió	Átalakítandó szám	Eredmény
BIN → HEX	1011 101 00	174
BIN → HEX	0,101 111 01	0,bd
BIN → HEX	-11 0011 11	-cf
BIN → HEX	0,1110 11	0,ec
BIN → HEX	+1010 1010,0101 0101	+Aa,55

Konverzió	Átalakítandó szám	Eredmény
HEX → BIN	ac76	1010 1100 0111 0110
HEX → BIN	-3c	-11 1100
HEX → BIN	0,b8	0,1011 1
HEX → BIN	-63,86	-110 0011,1000 011
HEX → BIN	+45	+10 0101

Konverzió	Átalakítandó szám	Eredmény
HEX → DEC	Ac76	44 150
HEX → DEC	-3c	-60
HEX → DEC	0,b8	0,718 75
HEX → DEC	-63,86	-99,523 437 5
HEX → DEC	+45	+69

Konverzió	Átalakítandó szám	Eredmény
OCT → BIN	765	111 110 101
OCT → BIN	-137	-1 011 111
OCT → BIN	0,42	0,100 01
OCT → BIN	-53,4	-101 011,1
OCT → BIN	+45	+100 101

Konverzió	Átalakítandó szám	Eredmény
OCT → DEC	765	501
OCT → DEC	-137	-95
OCT → DEC	0,42	0,503 906 25
OCT → DEC	-53,4	-43,5
OCT → DEC	+45	+37

5 Egyszerű alapműveletek különböző számrendszerekben

5.1 Műveletek decimális számokkal

5.2 Bináris számok összeadása

5.3 Bináris számok kivonása

5.4 Bináris számok szorzása

5.5 Műveletek hexadecimális számokkal

6 Kódok

A kódok megismeréséhez feltétlenül tisztáznunk kell néhány alapfogalmat.

6.1 Kódolási alapfogalmak

Kezdjük mindjárt magának a kódnak a meghatározásával. *Kód* = egy szimbólumhalmazhoz rendelt, azt egyértelműen leképező szimbólumhalmaz.

A kód egy másik, valószínűleg könnyebben emészthető meghatározása: egy – az érdekeltek mindegyike által – egyértelműen meghatározott és ismert adatábrázolási módszer. A kód tehát egy megállapodás az érdekeltek között az adatok ábrázolásának, tárolásának, továbbításának módjáról.

Aki ismeri a kódot, az képes a rendelkezésére álló adatokat a kódrendszerben érvényes szimbólumokká alakítani. Ez a művelet a *kódolás*.

A kódolt formájú adatok jelentésének megismeréséhez a kódszimbólumokat vissza kell alakítani a kódolás előtti jelekké, azaz el kell végezni a kódolás inverz műveletét, a *dekódolást*.

Természetesen a kódolás/dekódolás csupán szemlélet kérdése: az eredeti információ (esemény) is csak akkor értelmezhető, ha ismerjük a jelentését – ha tehát közelebbről szemügyre veszünk egy kódolást/dekódolást, minden esetben azt vehetjük észre, hogy valójában csupán *kódváltás* történt.

Kódoláskor a kódot alkotó elemi jelek halmazából, azaz a *szimbólumkészlet*ből választjuk ki az éppen szükséges elemeket. Azokat a kódokat, melyeknek szimbólumkészlete kételemű, *bináris kód*oknak nevezzük. A digitális technika jelenleg főként bináris, azaz kétértékű jelrendszert használ. Ezeket a gyakran a „0” és „1” szimbólumokkal szokták ábrázolni, ugyanakkor talán sejthető, hogy fizikailag ezeket a „lapos karika” és „álló kampó” jeleket nem tudjuk könnyedén feldolgozni az informatikai eszközökkel.

A szimbólumok mögött általában mindig valamilyen technikailag jól kezelhető állapot (esetleg állapotváltozás) megléte vagy hiánya – azaz valamilyen jel – áll. A *jelek* valamilyen fizikai jellemző érékei vagy értékváltozásai valaminek a függvényében. Leggyakrabban az *idő* függvényében szokás a fizikai jellemző értékeit jelnek tekinteni.

Van azonban olyan eset is, amikor valami más jellemző, pl. a *hely* függvényében változik a jellemző, és ezt nevezzük jelnek. Gondoljon csak erre a szövegre, amit olvas éppen. Jeleknek ebben a szövegben is a papír vagy képernyő felületének különböző helyén lévő fényességérték-minták tekinthetők.

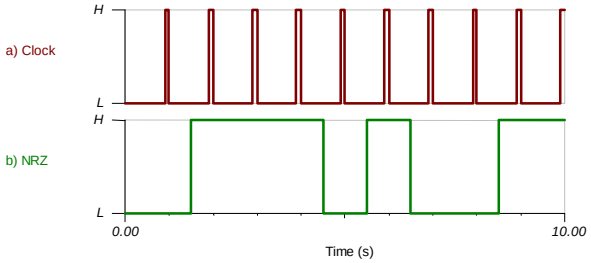
Ha a kódszimbólumokat a fizikai jellemző értéke közvetlenül hordozza, *statikus szimbólumábrázolásról* beszélünk. Néhány példa arra, hogy milyen fizikai jellemzők hordozhatják a jeleket:

- Feszültség [pozitív TTL logika]: 0 : 0V … 0,8V
  1 : 2,4V … 5V

  Egy ilyen módon ábrázolt 0 111 0 1 00 1 bitsorozat látható az 6-1. ábra NRZ jele. Az ábrán szereplő Clock jel ahhoz kell, hogy kijelöljük azokat a pillanatokat, amikor az NRZ jel szintjét egy új szimbólumnak kell tekintenünk.

- Mágneses mező [adathordozón] 0 : észak → dél
  1 : észak ← dél

- Anyag-folytonosság [adathordozón] 0 : nincs anyaghiány az adott helyen
  1 : lyuk van a szalagon

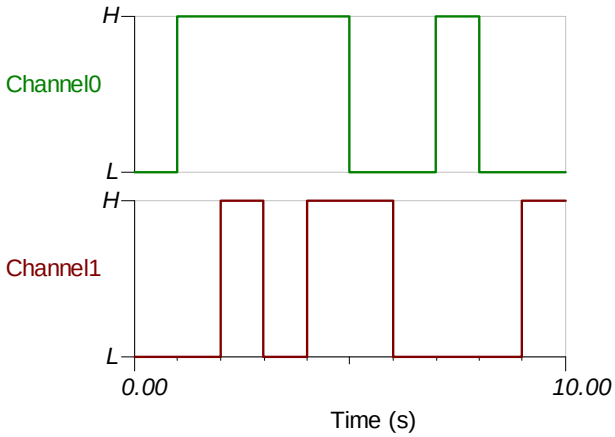


6-1. ábra

Ha a *jelek változását* tekintjük információnak, akkor azt *dinamikus szimbólumábrázolásnak* nevezzük. Például:

- Kétcsatornás adatábrázolás: 0 : Állapotváltozás a 0. adatcsatornán
  1 : Állapotváltozás az 1. adatcsatornán

  Az 6-2. ábra egy ilyen kétcsatornás szimbólumábrázolást szemléltet.
  Az ábrán a felső csatorna jelszintjének változása jelenti az 1-et, az alsó csatorna jelének változása pedig a 0-t.
  A jelek által hordozott szimbólumsorozat: 0 111 0 1 00 1.



6-2. ábra

- CD fizikai adatábrázolása 0 : folyamatosan land vagy folyamatos pit
  1 : pit/land vagy land/pit átmenet

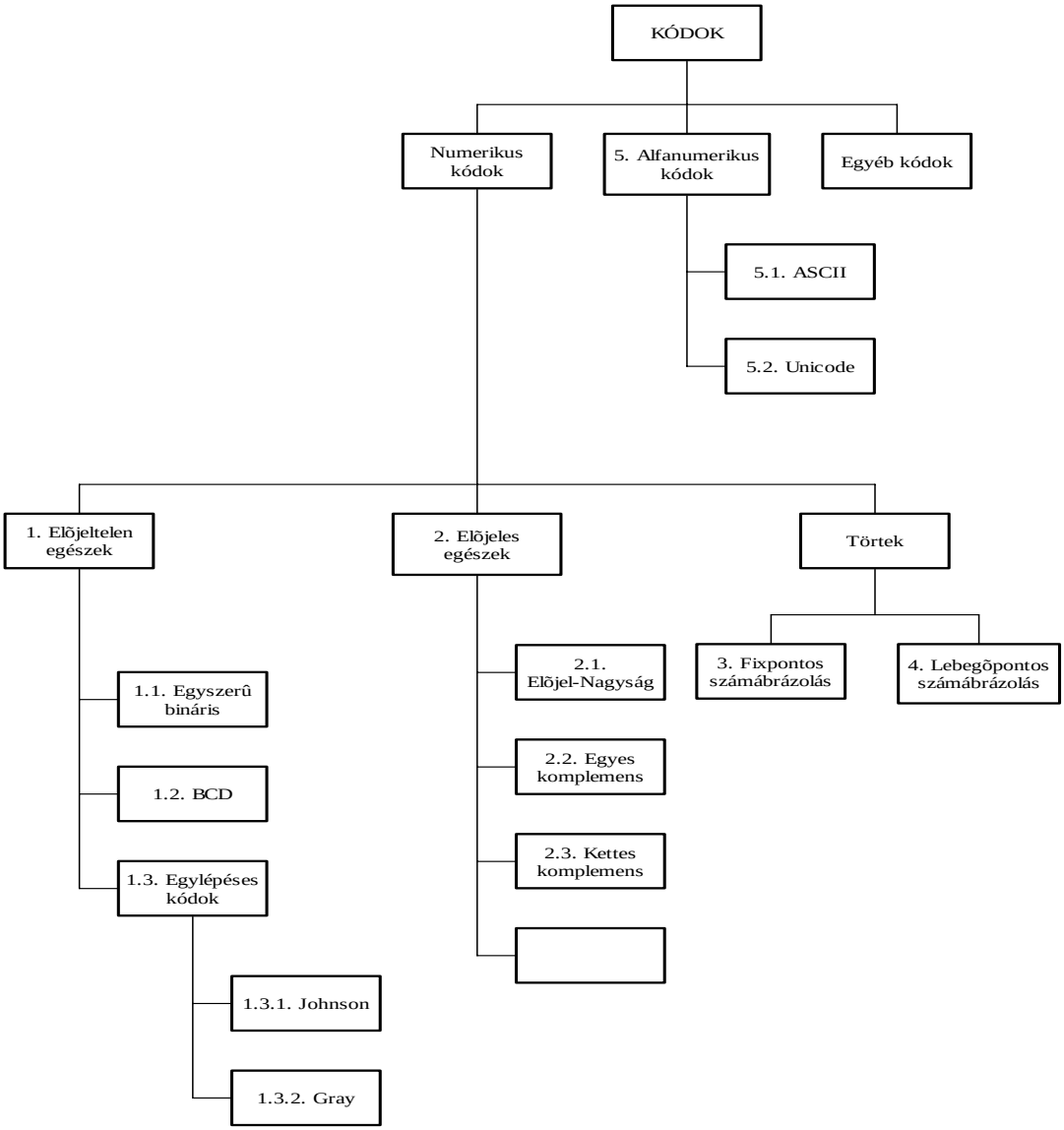
Függetlenül attól, hogy a szimbólumokat milyen módon ábrázoljuk, minden esetben ezek a szimbólumok hordozzák az adatokat. Pontosabban: az adatokat a szimbólumok egymásutánja jelzi.

Ne gondolja azonban azt, hogy az elemi szimbólumok közvetlen jelentéssel bírnak. A szimbólum-folyam nem egy tagolatlan massza. A szimbólumsorozat mindig kódszavakra tagolható.

Kódszónak nevezzük a kódszimbólumokból alkotott egybefüggő sorozatot. A kódszó az üzenet legkisebb, már értelmezhető jelentéssel bíró egysége.

A kódszavakat alkotó szimbólumok száma a *kódszó hossz*. Ha egy kód bináris, akkor a kódszavak hosszát bitekben adjuk meg.

A kódokat szokás osztályozni a kódszavak hossza alapján: a gyakorlatban használt kódrendszerek leginkább *rögzített kódszó hosszúságúak*, bár előfordulnak *változó szóhosszúságú kódok* is. A jelenleg elterjedten alkalmazott kódrendszerek szinte mind rögzített szószélességű


KÓDOK
Numerikus kódok
5. Alfanumerikus kódok
Egyéb kódok
5.1. ASCII
5.2. Unicode
1. Elõjeltelen egészek
2. Elõjeles egészek
Törtek
1.1. Egyszerû bináris
1.2. BCD
1.3. Egylépéses kódok
1.3.1. Johnson
1.3.2. Gray
2.1. Elõjel-Nagyság
2.2. Egyes komplemens
2.3. Kettes komplemens
3. Fixpontos számábrázolás
4. Lebegõpontos számábrázolás

Az ábrázolandó számot le kell írnia kettes számrendszerben, majd balról kiegészíteni annyi darab nullával, hogy a bitsorozat a kívánt kódszó-szélességnek megfeleljen.

Példák:

A $44_{(10)}$ számot ábrázoljuk előjeltelen bináris kódban 8 biten.

$44_{(10)} = 10\ 1100_{(2)}$ → a megoldás: 0010 1100

A $81_{(10)}$ számot ábrázoljuk előjeltelen bináris kódban 16 biten.

$81_{(10)} = 101\ 0001_{(2)}$ → a megoldás: 0000 0000 0000 0101 0001

A $175_{(10)}$ számot ábrázoljuk előjeltelen bináris kódban 6 biten.

$175_{(10)} = 1010\ 1111_{(2)}$ → a megoldás: nincs megoldás!!!

Természetesen a kódszó szélességből rögtön adódik az ábrázolható számtartomány is. Ha a kódolás szószélessége *n*, akkor az *n* biten ábrázolható számtartomány 2^n darab számból áll. Az ábrázolható értékek: 0; 1; 2; … (2^n-1). Az alábbi táblázat néhány gyakoribb a szószélesség esetén mutatja az ábrázolható számtartományt.

Szószélesség	Lehetséges értékek darabszáma	Ábrázolható számtartomány	
		DEC	HEX
n	2^n	$0 \ldots 2^n - 1$	
8	256 (0.25 k)	0; 1; … 255	00; 01; … FF
16	65536 (64k)	0; 1; … 65535	0000; 0001; … FFFF

6.2.2 BCD kód

A BCD kód az angol *Binary Coded Decimal* kifejezés kezdőbetűinek rövidítéséből adódik. A kifejezés magyar jelentése *Binárisan Kódolt Decimális*. A BCD kód nem a számok értékét, hanem a számok tízes számrendszerbeli alakját kódolja bináris formában.

A BCD kifejezés valójában nem is egyetlen nem egyetlen kódot, hanem egy kód-családot takar. Az alábbiakban megismerkedünk, a különböző BCD kódokkal.

A kódolás módszere mindig ugyanazt a sémát követi. A számunkra barátságos tízes számrendszerben megadott számértéket úgy tudjuk átírni BCD kódba, hogy a felírt szám egyes *számjegyeit egyenként, külön-külön kódoljuk* valamilyen szabály szerint bitnégyesekké, majd az így kapott értékeket valamilyen módon *egymáshoz illesztjük*.

Az egyes BCD kódok között az a különbség, hogy

(a) milyen módon állítjuk elő a bitnégyeseket az egyes decimális digitekből, és hogy

(b) milyen módszerrel illesztjük egymáshoz a digitek kódolásakor előállított kódszó-darabokat.

(a) Bitnégyesek előállítási módja szerint megkülönböztetett BCD rendszerek.

- Normál BCD, nBCD, 8-4-2-1 BCD

  Ez a módszer az egyes digitek értékének megfelelő bináris értékekkel kódol. A példák különböző decimális számokhoz tartozó bitnégyeseket szemléltetik.

  44 esetén ezek: 0100 és 0100

81 esetén ezek: 1000 és 0001

175 esetén ezek: 0001 és 0111 és 0101

- 3 többletes BCD, Excess-3 BCD

  Ez a módszer az nBCD bitnégyesek értékét még meg is növeli hárommal. A példák különböző decimális számokhoz tartozó bitnégyeseket szemléltetik.

  44 esetén ezek: 0111 és 0111

  81 esetén ezek: 1011 és 0100

  175 esetén ezek: 0100 és 1010 és 1000

Vannak más módszerek is, azonban ezeknek a jelentősége egyre kisebb. A 3 többletes módszer is sokkal kevésbé elterjedt, mint a a normál BCD; a műszaki informatikai gyakorlatban általában igaz, hogy ha BCD kódról beszélünk, akkor ez alatt nBCD módszerrel előállított bitsorozatot értünk. Az nBCD kifejezés helyett ezért a továbbiakban egyszerűen csak BCD szerepel mostantól.

(a) Bitnégyesek egymáshoz illesztése szerint megkülönböztetett BCD rendszerek.

- Tömör(ített) BCD, *Packed BCD*

  Ilyenkor a bitnégyeseket közvetlenül egymáshoz illesztjük. Egy bájton tehát két decimális digitet tudunk ábrázolni.

- Zónázott BCD, *Zoned BCD*

  Ilyenkor a bitnégyeseket nem közvetlenül egymáshoz illesztjük; minden egyes bájton csupán egyetlen hasznos bitnégyest helyezünk el úgy, hogy a bájt felső négy bithelyét nullákkal töltjük fel.

  Ezzel elérjük, hogy minden egyes bájt egyetlen decimális számjegyet tartalmaz. A zónázott BCD kódolás természetesen több – kétszer annyi – bithelyen ábrázolja ugyanazt a , mint a

Leginkább a számítógépek memóriában tárolt számok esetén szokás tömörített/zónázott formáról beszélni.

A 175 tömör BCD kódja 16 biten:	0000 0001 0111 0101
A 175 zónázott BCD kódja 3 bájton:	0000 0001 0000 0111 0000 0101

Ha a BCD kód hardver egységek összekapcsolására alkalmazott, akkor nem szokás értelmezni a tömörített/zónázott megkülönböztetést. Ha pl. egy elektronikus mérleg 5 digites hétszegmenses kijelzővel van ellátva, akkor a kijelzőt vezérlő elektronika az egyes digitek állapotát digitenként négy (összesen tehát 20) vezetéket használva határozhatja meg.

A BCD fontos tulajdonsága, hogy *tartalmaz tiltott kódszavakat*. A hasznos bitnégyeseknek tíz különböző értéke lehet (0…9) – négy biten azonban 16 lehetőség van összesen. A kódolás tehát nem használja ki az összes lehetséges bitkombinációt: hat tiltott értéket is tartalmaz, az 1010, 1011, 1100, 1101, 1110, és 1111 értékeket nem vehetik fel az nBCD bitnégyesek.

Az ábrázolható számtartomány *n* bites tömörített BCD kód esetén: 0, 1, … $(10^{n/4}-1)$. Természetesen *n* értéke mindig négynek egész számú többszöröse kell legyen. Az alábbi táblázat néhány gyakoribb a szószélesség esetén mutatja az ábrázolható számtartományt.

Szószélesség	Lehetséges értékek darabszáma	Ábrázolható számtartomány	
		DEC	HEX
N	$10^{n/4}$	$0 \ldots 10^{n/4}-1$	
8	100	00; 01; … 99	00; 01; … 99
16	10 000	0000; 0001; … 65535	0000; 0001; … 9999

6.2.3 Egylépéses kódok

Az egylépéses kódokat leginkább a közvetlenül digitális jeleket szolgáltató *érzékelők* használják. Az egylépéses kódok megismerése előtt azonban meg kell még ismerkednünk a kódszavak távolságának fogalmával.

Két kódszó távolságának nevezzük, azt, hogy két kódszó hány szimbólum-pozíción tér el egymástól. A kódszavak távolsága természetesen mindig egész szám.

Pl. a 0100 és a 0101 kódszavak közötti távolság éppen egy. Háromszimbólumnyi távolságra vannak egymástól az 11110011 és az 11110100 kódszavak.

Az alábbi táblázat mutatja az egyszerű bináris kód első néhány szomszédos kódszavának távolságát. Az egyszerű bináris kód láthatóan *nem* egylépéses kód!

Jelentés	Kódszó	Táv
0	0000	1
1	0001	2
2	0010	1
3	0011	3
4	0100	1
5	0101	2
6	0110	1
7	0111	4
8	1000	1
9	1001	2
10	1010	

A gyakorlatban nagy jelentőséggel bírnak azonban azok a kódrendszerek is, melyekben a szomszédos kódszavak távolsága mindig éppen egy. Ezeket a kódokat egylépéses kódoknak (Hamming kódoknak) nevezzük. Az egyszerű bináris kód nem ilyen, ezért nem alkalmazható pl. kódlécek helyzetkódolására. Természetesen a 8-4-2-1 BCD sem ilyen egylépéses kód.

Van azonban két egylépéses kód, mellyel érdemes megismerkednünk: ezek a Johnson és a Gray kódok.

6.2.3.1 Jonhson kód

A Johnson kódszavak képzése *a lehető legegyszerűbben* történik, annak szem előtt tartásával, hogy a szomszédos kódszavak csakis egyetlen bithelyen térjenek el egymástól.

Jelentés	Kódszó
0	000000
1	000001
2	000011
3	000111
4	001111
5	011111
6	111111
7	111110
8	111100
9	111000
10	110000
11	100000

Az n-bites Johnson kód a nulla értéket n darab nullával kódolja, majd a kódszó jobb oldalról fokozatosan feltöltődik egyesekkel. Ha az egyesek teljesen feltöltötték a kódszót, akkor kezdődik a kódszó feltöltése ismét nullákkal szintén az LSB felől, mígnem egyetlen egyes marad az MSB helyén. A táblázat egy 6-bites Johnson kódot szemlélt.

A Johnson kód nem használja ki az n biten rendelkezésre álló 2^n lehetőséget. *Rengeteg tiltott kódszót tartalmaz* ez a kódrendszer. Az n bites Johnson kód csupán 2n darab különböző állapot kódolására alkalmas. Annak ellenére, hogy ilyen „*pazarló*" a kódrendszer, a gyakorlatban időnként szokták alkalmazni egyszerűsége miatt.

6.2.3.2 Gray kód

Ez a kódrendszer *nem tartalmaz tiltott kódokat*, azaz képes kihasználni az *n* biten rendelkezésre álló 2^n darab állapotot.

A kódrendszert legegyszerűbben úgy alkothatjuk meg, hogy az *egy bites szószélességű Gray kódból* indulunk ki. Ebből kiindulva aztán fokozatosan szélesítjük a kódszavakat a kívánt mértékben. Az egy bites Gray kód meglehetősen egyszerű.

Jelentés	Kódszó
0	0
1	1

A *kétbites Gray kódszavakat* az egy-bites kódok ismeretében állíthatjuk elő az alábbi módszerrel. Az ismert kódrészeket tükrözzük a táblázatban jelölt tengelyre, majd a tengely fölötti kódszavak elé nullákat, a tengely alatti kódszavak elé pedig egyeseket írunk.

Jelentés	Szó	
0	0	0
1	0	1
	← tengely	
2	1	1
3	1	0

A *hárombites Gray kódszavakat* az egy-bites kódok ismeretében állíthatjuk elő az alábbi módszerrel. Az ismert kódrészeket tükrözzük a táblázatban jelölt tengelyre, majd a tengely fölötti kódszavak elé nullákat, a tengely alatti kódszavak elé pedig egyeseket írunk.

Jelentés	Kódszó		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
		← tengely	
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

A *négybites Gray kódszavakat* a három-bites kódok ismeretében állíthatjuk elő az alábbi módszerrel. Az ismert kódrészeket tükrözzük a táblázatban jelölt tengelyre, majd a tengely fölötti kódszavak elé nullákat, a tengely alatti kódszavak elé pedig egyeseket írunk. Próbálja kitölteni az alábbi négybites Gray kódhoz tartozó kódtáblázatot!

Kódszó LSB … MSB																
Jelentés	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

6.2.4 Gyakorló feladatok

6.2.5 Gyakorló feladatok megoldása

6.3 Előjeles egészek kódolása

Az előjeles számok kódolására már komolyabb feladat: *az előjelt is a 0 és 1 szimbólumokkal* kell közölnünk! Az *előjelbit* mindig a bitsorozat bal szélére kerül. A következőkben megismerkedünk azzal, hogyan szokás a számértékeket előjelükkel együtt kódolni. Az összes általunk megismerendő előjeles egész kódolási módszer a szám nagyságának kettes számrendszerben felírt alakjából indul ki.

6.3.1 Előjel-Nagyság kód

Ez legprimitívebb számábrázolási módszer. A kódszó MSB-jét nem tekintjük az érték, hordozójának. A kódszó első bitjét kinevezzük *előjelbitnek (S = signum)*. Ez az S bit jelzi a szám előjelét az alábbi megállapodás szerint:

- pozitív számok esetén S = 0
- negatív számok esetén S = 1

A további bitek természetesen a szám nagyságát (abszolút értékét) jelzik egyszerű bináris kódban. Tehát a nagyság MSB-je a kódszó második bitje (balról).

Az előjelbit bevezetéséért azzal kell fizetnünk, hogy az ábrázolható legnagyobb abszolút érték természetesen lecsökken: *n* tites kódszó esetén az ábrázolható számtartomány:

$$-(2^{n-1}-1)\ ;\ \dots\ ;\ -2\ ;\ -1\ ;\ -0\ ;\ +0\ ;\ +1;\ +2\ ;\ \dots\ ;\ +(2^{n-1}-1)$$

Az alábbi táblázat néhány gyakoribb a szószélesség esetén mutatja az ábrázolható számtartományt. Vegye észre, hogy az előjel-nagyság kód külön képes ábrázolni a +0 és -0 értékeket!

Bitszám	Lehetséges értékek darabszáma	Ábrázolható számtartomány	
		DEC	HEX
N	2^n	$-(2^{n-1}-1)$ … -0; +0… $+(2^{n-1}-1)$	
8	256	-127 … -0; +0… +127	-7F … -00; +00 … +7F
16	65536	-32767 … -0; +0 … +32767	-7FFF…-0; +0…+7FFF

Lássunk néhány példát az előjel-nagyság kódra!

Kódolandó	Keletkező kódszó
+44 előjel-nagyság kódban 8 biten →	a) átírjuk a 44-et kettes számrendszerbe: 10 1100 b) a számot balról feltöltjük nullákkal – ezzel kialakul a nagyság: s010 1100 c) a pozitív előjel miatt S = 0 0010 1100
-44 előjel-nagyság kódban 8 biten →	1010 1100 (S=1 lett)
+44 előjel-nagyság kódban 12 biten →	0000 0010 1100
-44 előjel-nagyság kódban 12 biten →	1000 0010 1100

A kódrendszer hátránya, hogy különböző számok összevonásakor az előjeleket külön figyelnünk kell a műveletvégzés során. Szinte soha nem is alkalmazzák.

6.3.2 Egyes komplemens kód

Ez már egy kicsit trükkösebb. :-)

A pozitív számokat ugyanúgy kódoljuk, mintha Előjel-nagyság kódban ábrázolnánk a számot.

A *negatív* számokat azonban nem csupán az előjel megfordításával képezzük: negatív szám esetén *minden bitet megfordítunk*: tehát nemcsak az előjel bit változik 1-re, hanem az egyes értékbitek is megfordulnak!

Kódolandó	Keletkező kódszó
+44 előjel-nagyság kódban 8 biten →	a) átírjuk a 44-et kettes számrendszerbe: 10 1100 b) a számot balról feltöltjük nullákkal – ezzel kialakul a nagyság: s010 1100 c) a pozitív előjel miatt S = 0 0010 1100
-44 előjel-nagyság kódban 8 biten →	1101 0011 (minden bit megfordult!)
+44 előjel-nagyság kódban 12 biten →	0000 0010 1100
-44 előjel-nagyság kódban 12 biten →	1111 1101 0011

Az ábrázolható számtartomány megegyezik az előjel-nagyság kódhoz tartozóval:

$$-(2^{n-1}-1)\ ;\ \ldots\ ;\ -2\ ;\ -1\ ;\ -0\ ;\ +0\ ;\ +1;\ +2\ ;\ \ldots\ ;\ +(2^{n-1}-1)$$

Ebben a kódrendszerben is külön kódszó jelzi a +0 és a -0 értékeket. Különböző előjelű számok összevonásakor az egyszerű bináris összeadással is helyes eredményt kapunk szerencsés esetben (ha az összeg nulla), de máskor sajnos nem: mindig eggyel kevesebb jön ki!

	0010 1100	(+44)
+	1101 0011	(-44)
	1111 1111	(-0)

	0011 0010	(+50)
+	1101 0011	(-44)
[1]	0000 0101	(+5)

6.3.3 Kettes komplemens kód

Ez a számítástechnikában *legelterjedtebben* (szinte kizárólagosan) alkalmazott kódolási forma.

A pozitív számokat ugyanúgy kódoljuk, mint az előző kódok esetén.

A negatív számok esetén a pozitív megfelelőből képzett kettes komplemenst ábrázoljuk. A kettes komplemens képzés műveletét háromféleképpen tudjuk elvégezni: matematikai módszerrel, negálást követő inkrementálással, valamint „kézi” módszerrel.

- **Matematikai módszer**

Ilyenkor a kiindulási (pozitív) számot kivonjuk a csupa nulla bitből álló kódszóból. Az esetleg keletkező végső áthozatot eldobjuk. A módszer nagy hátránya, hogy a kivonást el kell végeznünk. Általában az emberek még a tízes számrendszerben is kerülik a kivonást – a kettes számrendszerben ugyanez a művelet még jobban megvisel minket. A példán 44 kettes komplemensének képzése látható 8 biten.

	0000 0000	(0)
-	0010 1100	(-44)
	1101 0100	

- **NOT + INC**

A módszer onnan kapta nevét, hogy a mai ALU egységek rendkívül gyorsan képesek egy bitsorotat minden egyes bitjét negálni (NOT), majd ez után szintén pillanatok alatt tudják az eredményt növelni eggyel (INCrementálás = növelés). Az első művelet az eredeti szám egyes komplemensét képezi, a második ezt növeli eggyel. Az eredmény a kettes komplemens. Ezzel a módszerrel dolgoznak a számítógépek ALU egységei.

	0010 1100	(+44)
	1101 0011	1es k.
+	0000 0001	(+1)
	1101 0100	

- **„Kézi" kettes komplemens képzés**

Ezt a módszert egy átlagos képességű emberi teremtmény képességeihez igazították. A módszer rendkívül egyszerű: az eredeti szám alá oly módon írhatja fel a kettes komplemenst ügyesen, hogy

- jobbról balra másolja a biteketet, mígnem elér az első egyes értékű bithez.
- még ezt a bitet is lemásolja változatlanul,
- a további biteket azonban negálja (0 ⇔ 1).

0010 1**1**00	(+44)
← erre	
1101 0**1**00	

Eredményképpen magát a szignumot is helyesen fogja megkapni.

A számábrázolási tartomány csak egyetlen 0 értékeket tartalmaz; a tartomány negatív irányban eggyel nagyobb kiterjedésű. Az alábbi táblázat néhány gyakoribb a szószélesség esetén mutatja az ábrázolható számtartományt.

Bitszám	Lehetséges értékek darabszáma	Ábrázolható számtartomány	
		DEC	HEX
N	2^n	-2^{n-1}) … -0; +0… $+(2^{n-1}-1)$	
8	256	-128 … -0; +0… +127	-80 … 0 … +7F
16	65536	-32768 … -0; +0 … +32767	-8000 … 0 …+7FFF

A kettes komplemens kódban végzett különböző előjelű számok összevonása mindig helyes eredményt ad! Ezért szeretjük ezt a kódot.

	0010 1100	(+44)
+	1101 0100	(-44)
[1]	0000 0000	(0)

	0011 0010	(+50)
+	1101 0100	(-44)
[1]	0000 0110	(+6)

6.3.4 Eltolt bináris kód

Ez a kód egészen más filozófiát követ, mint az előzőek – érdekes módon mégis hasonló eredményre jut, mint a kettes komplemens kódolás. Az eltolt bináris kódolás egy nagyon egyszerű megfontoláson alapul, amit a 4 bites szószélességű kódrendszer példáján keresztül ismerhetünk meg. Az egyes számértékekhez tartozó kódszavakat a 6-5. ábra utolsó oszlopában tekintheti meg.

Az elv nagyon frappáns: ha 4 biten *abszolút bináris* – tehát előjeltelen – kódban 0-tól 15-ig tudok elszámolni, akkor úgy tehetem a módszert előjeles számok ábrázolására is alkalmassá, hogy egyszerűen *eltolom* a bitsorozat jelentését az abszolút bináris értelmezéshez képest. Ha négy biten kell a számokat kódolnom, akkor az eltolást úgy végzem, hogy *az abszolút számtartomány „közepén" található, binárisan éppen kerek* 1000 *kódszót tekintem nullának.* Az 1000 fölötti bitsorozatok (abszolút értelmezés szerint: a 8-nál nagyobbak) mind pozitívak. A 0000 … 0111 kódszavak tartománya természetesen negatív számokat reprezentál. A négybites ofszet bináris kód eltolása tehát éppen 8. Az *n* bites kódszóhoz mindig 2^{n-1} értékű eltolás tartozik. (Léteznek természetesen olyan furcsa, „ezoterikus" OB kódok is, melyeknek az eltolása *n* bites kódszavak esetén *nem* 2^n, ezeket azonban tekintsük csak kivételeknek, melyek erősítik a szabályt. :-))

Számábrázolási tartománya megegyezik a kettes komplemens kódolásával.

Az ofszet bináris kódokat a gyakorlatban leginkább az ADC-k és DAC-k *(lásd: 3.3. rész: A/D és D/A konverzió)* használják.

6.3.5 Előjeles egész-ábrázolások összevetése

Az ismertetett négy különböző előjeles egész ábrázolási módszer összehasonlító táblázatát a 6-5. ábra tartalmazza.

Jelentés	Ábrázolási módszer			
	Előjel-nagyság (E-N)	Egyes komplemens (1-es K)	Kettes komplemens (2-es K)	Ofszet bináris (O.B.)
+7	0111	0111	0111	1111
+6	0110	0110	0110	1110
+5	0101	0101	0101	1101
+4	0100	0100	0100	1100
+3	0011	0011	0011	1011
+2	0010	0010	0010	1010
+1	0001	0001	0001	1001
0 +0	0000	0000	0000	1000
0 -0	1000	1111		
-1	1001	1110	1111	0111
-2	1010	1101	1110	0110
-3	1011	1100	1101	0101
-4	1100	1011	1100	0100
-5	1101	1010	1011	0011
-6	1110	1001	1010	0010
-7	1111	1000	1001	0001
-8	-	-	1000	0000

6-5. ábra

Vegye észre az alábbi összefüggéseket a táblázatban.

- Az EN és 1K kódokkal külön jelölhetjük a +0 és –0 értékeket, a 2K és OB kódok csak egy nullát tartalmaznak.
- A 2K és OB kódoknak negatív irányban eggyel hosszabb a számábrázolási tartománya.
- Az EN, 1K és 2K kódok a pozitív számokat egyaránt 0 előjelbittel ábrázolják.
- Az EN, 1K és 2K kódok a pozitív számokat teljesen egyformán ábrázolják; csak a negatív számok ábrázolásában van eltérés.
- Az EN kód nagyság-része a +0|-0 határra tengelyesen szimmetrikus
- Az 1K kód minden bitjéhez a +0|-0 határra tengelyesen szimmetrikus kódszó fordított bitje tartozik.
- A 2K kód negatív számokhoz tartozó minden kódszava azonos az eggyel balra és fölötte lévő 1K kódszóval.
- Az *OB kód* kódszavai felfoghatók *fordított előjelbites kettes komplemens kódként* is! (Tehát van egy újabb, immár negyedik módszer is a kettes komplemens kód képzésére: állítsuk elő az eltolt bináris kódot, majd fordítsuk meg a kódszó előjelbitjét!

A fenti észrevételek egy része talán nyilvánvalónak tűnik – mégis javaslom, hogy egy kicsit merengjen el a megjegyzéseken és, tanulmányozza önállóan is a 6-5. ábra táblázatát. Más, az előzőekben fel nem sorolt összefüggéseket is észrevehet rajta! Egy igazi bitvadász számára ez mindig nagy örömöt jelent…

6.3.6 Gyakorló feladatok

6.3.7 Gyakorló feladatok megoldása

6.4 Valós számok ábrázolása

A valós számok egész- és törtrészt is tartalmazhatnak.

6.4.1 Fixpontos számábrázolás

Fixpontos számábrázolás esetén mindig egy kissé módosított egész ábrázolási módszert használunk. Annyi a különbség csupán, hogy a képzeletbeli tizedes (illetve kettedes) vesszőt nem a számjegy-sorozat végére, hanem valahová előrébb helyezzük.

Ismerkedjünk meg ezzel a számábrázolással egy példán keresztül!

Rendelkezésünkre áll összesen 32 bit a számok kódolására, és e=24 bitet használunk a számok egészrészének ábrázolásához, t=8 bitet pedig a törtrész számára tartunk fönn. A helyzet az alábbi:

$n_e = 23$ → ábrázolható számok egészrészének tartománya kb. +- 2^{23}, azaz +-8 388 608.

$n_t = 8$ → az egyes értékek egymástól 1/256 =0,00390625 távolságra helyezkednek el a teljes tartományban. Az ábrázolás *felbontása a teljes számtartományban egyenletes,* állandóan ekkora.

6.4.2 fix-point-reals.sdr

Ha nagyobb pontosságot (felbontást) akarunk, akkor a törtrész bitszámát (n_t-t) kell növelnünk.

Ha elegendően nagy számtartományt kívánunk a kóddal lefedni, akkor n_e értékét, azaz az egészrészeknek fenntartott bitszámot kell megnövelnünk.

Ha azonban mind az egészrészek, mind pedig a törtrészek bitszámát megnöveljük, akkor bármely *konkrét* számot is ábrázoljuk a kódrendszerben, mindenképpen pocséklunk:

- kis számok esetén a nagy bitszámú egészrész felesleges
- nagy számok esetén a törtrészre nincs szükség

6.4.3 Lebegőpontos számábrázolás

A lebegőpontos számábrázolás megszünteti a fixpontos kódolás bitpazarlását.

A lebegőpontos ábrázolási forma a valós számokat két fixpontos értékkel ábrázolja. Ezek egyikét *mantisszának (m),* a másikat pedig *karakterisztikának (k)* nevezzük. A *k* karakterisztikát exponensnek is szokás nevezni. Az így kódolt szám értelmezése:

$$\boxed{szám = m \cdot r^k},$$

ahol *r* az ábrázolás úgynevezett *alapszáma (radix).* A radix értékét megállapodásban rögzítjük: a számot ábrázoló jelsorozat csak m és k értékeit tartalmazza – a radixot nem.

A mantissza értékét *normálni* szokták – azaz rögzítik a benne tárolt egészrész számjegyeinek darabszámát.

A lebegőpontos ábrázolást a legtöbb valamennyire is „igényes” számológép ismeri.

6.4.3.1 *m* és *k* szószélességének hatása az ábrázolt számtartományra

Az *m* és *k* értékek az ábrázolt szám értékét határozzák meg:

- *m* a szám előjelét és konkrét értékét határozza meg, míg
- *k* a szám nagyságrendjére van hatással.

Ha a *mantissza* tárolására több számjegynyi helyet tartunk fenn, akkor az ábrázolható számtartomány finomodik, azaz az egyes értékek ábrázolásának relatív pontosságát növeljük meg. Tehát *a mantissza tárolására fenntartott digitek száma* a számtartományon belüli számábrázolás finomságát, a felbontását, az *egyes ábrázolt értékek relatív pontosságát* határozza meg.

Ha a *karakterisztika szószélességét* növeljük, akkor az ábrázolható számok tartományát szélesítjük ki.

Nullától egyre távolabbi, *egyre nagyobb számokat* leszünk képesek ábrázolni a számegyenesen pozitív és negatív irányban egyaránt. A legkisebb ábrázolható nem-nulla szám is egyre közeledik a nullához: *egyre kisebb számokat* tudunk tehát ábrázolni.

Tehát a *k* tárolására fenntartott digitek száma a számtartomány kiterjedését, az ábrázolható számok nagyságrendjét határozza meg.

A fixpontos és lebegőpontos ábrázolási módszereket az alábbi ábra szemlélteti.

6.4.4 floating-point-reals.sdr

6.4.4.1 Tudományos normál alak

Ha egy szám túlságosan nagy, akkor mindig célszerű a karakterisztika-mantissza módszerrel történő megadása. Pl. -423000000000000000000000000000 helyett sokkal célszerűbb $-4{,}23 \cdot 10^{28}$-t írnunk.

A normálás (vagy normalizálás) a matematikusok számára azt jelenti, hogy *a mantissza egészrészét egy számjegyűvé kell tennünk.* Ezt a normalizált alakot tudományos alaknak nevezzük. Jele a számológépeken általában **SCI** (scientific = tudományos). A normalizálás során a tizedespontot (vesszőt) jobbra vagy balra mozgatjuk, mígnem a szám egészrésze egy számjegyűvé válik. Eközben a karakterisztika értékét természetesen mindig növeljük vagy csökkentjük eggyel, hogy a szám értéke változatlan maradjon.

Példák a tudományos normalizálásra:

Eredeti alak	Lebegőpontos ($m \cdot r^k$) alak	Normalizált alak (SCI)
64535.87	$64535.87 \cdot 10^0$	$6.453587 \cdot 10^{+4}$
0.000 010 458	$0.000\,010\,458 \cdot 10^0$	$1.045\,8 \cdot 10^{-5}$
-5 100 000	$-5\,100\,000 \cdot 10^0$	$-5.1 \cdot 10^{+6}$

6.4.4.2 Mérnöki normál alak

Egy másik tipikus üzemmód a számológépek esetén a mérnöki (**ENG** = engineering) normálalak. Ilyenkor a normálás megengedi, hogy a mantissza egy, kettő vagy három digites egészrészű legyen.

Mérnöki normáláskor célunk az, hogy *a karakterisztika mindig hárommal osztható legyen*. Ez akkor célszerű, ha a kapott jellemző nagyságrendjét jól érzékelhetővé akarjuk tenni. A mérnöki normálalakban megadott számokat könnyedén átírhatjuk szabványos SI prefixumokká.

Példák a mérnöki normalizálásra:

Eredeti alak	Lebegőpontos ($m \cdot r^k$) alak	Normalizált alak (SCI)
64535.87	$64535.87 \cdot 10^0$	$64.53587 \cdot 10^3$ [64.535 87 kilo]
0.000 010 458	$0.000\,010\,458 \cdot 10^0$	$10.45\,8 \cdot 10^{-6}$ [10.45 mikro]
-5 100 000	$-5\,100\,000 \cdot 10^0$	$-5.1 \cdot 10^6$ [5.1 Mega]

6.4.4.3 Számítástechnikai normál alak

A számítástechnikai normál alakot jelenleg szabványok rögzítik (IEEE 754 és IEEE 854 szabványok). Az ábrázolás *alapszáma (radix): kettő*.

A számokat bináris alakjuk szerint, egy bites egészrészre (azaz 1-re!) normálva ábrázolja. A számokat több pontossági szinten – azaz több szabványos szószélességet használva – lehet kódolni. A lehetséges teljes szószélességek: 32, 64, 80 és 128 bit. Minden esetben az alábbi mezőkből indul ki egy lebegőpontos szám ábrázolása.

$$(-1)^{\mathbf{S}} \cdot 2^{\mathbf{E}} \cdot \mathbf{1.F}$$, ahol

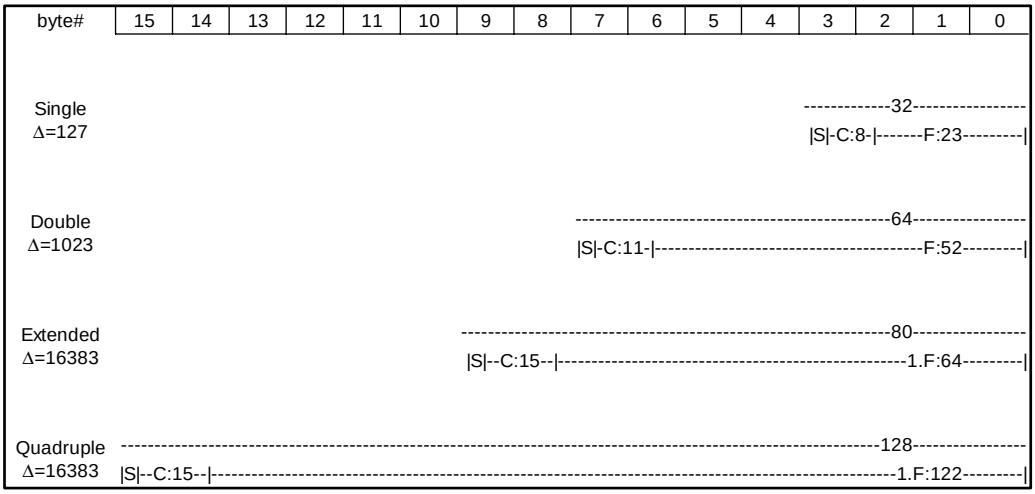
- *S* = a szám előjelbitje (Sign)
- *F* = a mantissza törtrésze (Fractional part). *F* ábrázolása – az előjel értékétől függetlenül – mindig abszolút bináris kódban történik.
- *E* = az ábrázolandó szám exponense (karakterisztikája)

A bináris jelsorozatban az Exponens előjeles ábrázolása helyett egy eltolt karakterisztikát kell megjeleníteni: $\boxed{C = E + \Delta}$, ahol

> Δ = egy konstans, melynek értéke *E* bitszámától függ. Az eltolás azért szükséges, mert a szabvány csak a mantissza előjelét (tehát az ábrázolt szám előjelét) kódolja külön előjelbit használatával. A karakterisztika (exponens) kódolásakor eltolt bináris kódot használnak. És természetesen *nem* szabályos

Az ábrázolás érdekessége, hogy a 32 és 64 bites szószélesség esetén a kódszó nem tartalmazza az "1.F" formájú mantisszának az "1." bevezető egészrészét. Ezt a *bennfoglalt egyest* az FPU természetesen minden műveletvégzéskor automatikusan „hozzá gondolja” a számhoz.

A szabvány az alábbi pontossági osztályokat definiálja: Single (egyszeres), Double (dupla), Extended (kiterjesztett) és Quadruple (négyszeres). Minden egyes pontossági osztályhoz rögzítve van az egyes mezők szélessége és a Δ érték is. Az egyes lebegőpontos számformátumokat egy táblázatban összefoglalva tekintheti meg.



A szabvány lehetővé teszi a nagyon kis számok denormalizált ábrázolását, rögzíti továbbá a +0 és -0 értékeket, a +∞ és -∞ értékeket valamint a NaN (Not a Number) értéket is. További információk a szabványból tudhatók meg.

6.4.5 Gyakorló feladatok

6.4.6 Gyakorló feladatok megoldása

6.5 Alfanumerikus kódok

Az Alfanumerikus kódok mindegyike egy-egy elterjedt, elfogadott megállapodáson alapuló kódtáblázatot jelent. A ma használt alfanumerikus kódok egyik őse a 6 bites telex kód volt. Az EBCDIC kódolás – a másik nagy öreg – 8 biten kódolt egy-egy karaktert – nagygépes környezetben ez utóbbi kódrendszer még ma is előfordul. A műszaki alkalmazások szempontjából legfontosabb két alfanumerikus kódrendszer jelenleg az ascii kód és a unicode – ezeket vizsgáljuk a továbbiakban.

6.5.1 ASCII

Az ASCII (ejtsd: eszki) kód az egyik legrégebbi – és egyben az egyik legelterjedtebb alfanumerikus kódrendszer. A szabványos *American Code for Information Interchange* az ANSI által elfogadott, karakterenként *7 bitet használó* kódtáblázatot jelent. A kódszó két legnagyobb helyiértékű bitje a kódszó típusát adja meg, a további öt bit pedig konkréttá teszi a karakter jelentését.

? | x x | x x x x x

- 00 → vezérlő karakter
- 01 → matematikai és egyéb jelek, számok
- 10 → Angol ábécé NAGYbetűi
- 11 → Angol ábécé kisbetűi

A „?” helyén az eredeti ASCII kódban soha nem szerepelhetett semmi; csupán azt jelzi számunkra, hogy a 8 bites kódszóhoz még hiányzik egy bit! Az ASCII kódtábla felépítését a 6-6. ábra jelzi. Az táblázat peremezésén szereplő $ szimbólummal kezdődő számok az egyes kódszavak ASCII kódjának felső illetve alsó 4 bitjére utalnak értelemszerűen. Pl. a „G” karakter a $40 oszlop $07 sorában található, tehát ASCII kódja hexadecimális 47, azaz 1000111.

	$00	$10	$20	$30	$40	$50	$60	$70
$00	NUL	DLE	space	0	@	P	`	p
$01	SOH	DC1	!	1	A	Q	a	q
$02	STX	DC2	"	2	B	R	b	r
$03	ETX	DC3	#	3	C	S	c	s
$04	EOT	DC4	$	4	D	T	d	t
$05	ENQ	NAK	%	5	E	U	e	u
$06	ACK	SYN	&	6	F	V	f	v
$07	BEL	ETB	'	7	G	W	g	w
$08	BS	CAN	(	8	H	X	h	x
$09	HT	EM	)	9	I	Y	i	y
$0A	LF	SUB	*	:	J	Z	j	z
$0B	VT	ESC	+	;	K	[	k	{
$0C	FF	FS	,	<	L	\	l	|
$0D	CR	GS	-	=	M	]	m	}
$0E	SO	RS	.	>	N	^	n	~
$0F	SI	US	/	?	O	_	o	

6-6. ábra

A vezérlő karakterek közül néhány fontosabb jelentésével szintén érdemes tisztában lennünk:

NUL = Null = Semmi (üres karakter)
Ennek a karakternek a vételekor abszolút semmi sem történik. Kizárólag „hely-kitöltő” szerepe van.

BEL	=	Bell	=	Csengetés Hatására a vevő „csenget” egyet [sípol a nyomtató vagy a konzol]. Figyelemfelhívásra használják.
HT	=	Horizontal Tabulator	=	Vízszintes tabulátor A nyomtatófej/kurzor a legközelebbi tabulátor pozícióhoz ugrik.
LF	=	Line Feed	=	Soremelés A nyomtatófej/kurzor a jelenlegi pozíció alatti helyre, azazz egy sorral lejjebb ugrik.
CR	=	Carriage Return	=	Kocsi vissza A nyomtatófej/kurzor a pillanatnyi sor elejére ugrik.
FF	=	Form Feed	=	Lapdobás A nyomtatás/megjelenítés a következő üres oldal/képernyő kezdő pozíciójánál folytatódik. Lap-nyomtatók ennek a karakternek a vételekor kezdik a puffer memóriában összegyűjtött adatok papírra nyomtatását.
BS	=	Back-Space	=	Visszatörlés A kurzor egyel balra lép, és a kurzor alatti karakter helyére szóköz (space) kerül.
ESC	=	Escape	=	„Menekülés” – kilépés az ASCII kódrendszerből Az ESC jelzi, hogy a további karakterek *nem* ASCII karakterekként értelmezhetők. A nyomtatókat grafikus üzemmódba kapcsoló parancs például mindig ESC-vel kezdődik. (Vigyázat! A számítógép billentyűzetének ESC gombja más funkciót tölt be!)

A "?xxxxxx" felépítésű ASCII kód MSB-jét a gyakorlatban szinte mindig beállítják valamilyen szabály szerint. A „?” bit kitöltésére három módszer kínálkozik.

- Az MSB értéke *mindig legyen 0!*
- Az MSB értéke tartalmazzon valami olyan információt, ami fokozza a küldött karakterek átvitelének megbízhatóságát: az MSB *legyen paritásbit!*
- *Bővítsük ki az ASCII kódtáblázatot* 8 bitessé!

Mindhárom ötletet érdemes részletesebben tanulmányoznunk! Lássuk őket sorban…

1. Az MSB értéke mindig legyen 0!

   Ezt a módszert pl. akkor tapasztalhatjuk meg, ha egy régebbi SMTP protokolt (nem pedig a nyolc bites karakterek átvitelét is lehetővé tevő ESMTP-t) használó szerveren áthalad a 8 bites kódolást használó, ékezeteket is tartalmazó e-mail. Mivel az USA-ban nincs szükség ékezetes karakterekre, az ottani szerverek számára természetes, hogy egy írott szövegben csak 0 … 127 közötti értékű ASCII karakterek találhatók – egy ilyen szerver számára tehát abszolút természetes, hogy az ékezetes magyar szöveg rajta áthaladva torzul. Egy példaszöveg:

   ```
   Íme, jól látható, amint az ÉKEZETES BETŰK megjelennek
   KÜLÖBBÖZŐ KÓDOLÁSSAL egy magyar nyelvű szövegben.
   ```

```
Ebben a négysornyi írásban fellelhető a magyar nyelv minden
különleges betűje. Úgyhogy nem is húzom tovább...
```

Torzulás után ugyanaz a szöveg.

```
 me, jsl lathats, amint az IKEZETES BET[K megjelennek
K\LVBBVZU KSDOLASSAL egy magyar nyelv{ szvvegben.
Ebben a nigysornyi mrasban fellelhetu a magyar nyelv minden
k|lvnleges bet{je. Zgyhogy nem is hzzom tovabb...
```

2. az MSB *legyen paritásbit!*

   Ez a módszer a kódszóban lévő 1-esek darabszámát az úgynevezett paritásbit felhasználásával kiegészíti egy oly módon, hogy a paritással kiegészített kódszóban az 1-esek darabszáma mindig páros vagy páratlan legyen. Ha a kiegészítés páros darabszámra történik, akkor páros paritású rendszerről van szó, ellenben páratlan paritási rendszerről beszélünk.

   A „DIGIT” karaktersort páros paritású esetben az alábbiak szerint kódolhatjuk.

D	**1**1100100
I	**0**1101001
G	**1**1000111
I	**0**1101001
T	**0**1110100
k.par.	**0**1110111

   Az átvitel során jól felismerhető az egyetlen bithiba.

   Ha a bitfolyam bármely kódszava megsérül, akkor a vevő ezt rögtön észre is veszi a paritás ellenőrzésekor. Ilyenkor kérheti újra az adást.

   Ha keresztirányú paritást (a példában: k.par.) is tartalmaz a rendszer, akkor még a hiba helye is meghatározható, és a hibás bitet a vevő automatikusan kijavíthatja a teljes adatblokk újraküldésének kérése nélkül. Ezt a módszert régebben használták modemes kommunikációhoz. Jelenleg nem ebben a formában alkalmazzák ugyan, a paritásos adatellenőrzés módszere azonban nagyon elterjedt máig is a műszaki informatikai alkalmazásokban.

3. *Bővítsük ki az ASCII kódtáblázatot* 8 bitessé!

   Ezt a módszert használja a kiterjesztett ASCII szabvány. Egy külön címet is megérdemel.

6.5.2 Kiterjesztett ASCII; kódlapok

A 7 bites ASCII kód 128 elemű karakter-készletébe nem fért bele a fontosabb latin betűs írást használó nyelvek minden ékezetes karaktere. Ha 8 bitesre bővítjük a kódteret, akkor a 128 … 255 közötti kódszavakhoz hozzá lehet rendelni a különböző ékezetes betűket, sőt egyéb jelekre is marad hely.

A kiterjesztett ASCII lehetővé teszi az ékezetes karakterek használatát, de sajnos a 128 karakter nem elegendő arra, hogy minden nemzet ékezetes betűi beleférjenek. A „megoldást” az jelenti, hogy az ASCII kód kiterjesztett kódszavait a világ – és az alkalmazások – más-más területein másképpen értelmezik. Egy értelmezési rendszert egy kódlapnak szoktak hívni. Egy megjelenítő eszköz egyszerre csak egy kódlap szerint képes a karaktereket megjeleníteni. Új kódlap használatához kijelző-üzemmódot kell váltani.

Néhány elterjedtebb kódlap:

- DOS 437 (IBM PC 8 bites alapértelmezett ASCII kódja)
- DOS 852 (Latin 2 kiterjesztés, DOS esetén a Magyar Szabvány szerint kötelezően előírt)

- ANSI 1250 (Windows – Magyar Szabvány)
- ANSI 1252 (Windows – USA)
- ISO 8859-1, ISO 8859-2

Az egyes kódlapok sok esetben hasonló betűket kódolnak ugyanazzal a bitsorozattal, azonban ez nem mindig teljesül; például az 1101 0101 kódszó hatására

- a magyar Windows „Ő” karaktert,
- az amerikai Windows „Õ” karaktert,
- a magyar DOS „Ñ” karaktert,
- az amerikai DOS pedig „ ╒” karaktert jelenít meg,

Példaképpen az ANSI 1250 kódlap felépítése (Magyar Windows):

	$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
$00	space		space	°	Ŕ	Đ	ŕ	đ
$01		‘	ˇ	±	Á	Ń	á	ń
$02	˛	’	˘	˛	Â	Ň	â	ň
$03		“	Ł	ł	Ă	Ó	ă	ó
$04	„	”	¤	´	Ä	Ô	ä	ô
$05	…	•	Ą	µ	Ĺ	Ő	Í	ő
$06	†	–	¦	¶	Ć	Ö	ć	ö
$07	‡	—	§	·	Ç	×	ç	÷
$08			¨	¸	Č	Ř	č	ř
$09	‰	™	©	ą	É	Ů	é	ů
$0A	Š	š	Ş	ş	Ę	Ú	ę	ú
$0B	‹	›	«	»	Ë	Ű	ë	ű
$0C	Ś	ś	¬	Ľ	Ě	Ü	ě	ü
$0D	Ť	ť	-	˝	Í	Ý	í	ý
$0E	Ž	ž	®	ľ	Î	Ţ	î	ţ
$0F	Ź	ź	Ż	ż	Ď	ß	ď	˙

Ez a módszer tehát nem oldja meg teljesen az ékezetes betűk problémáját: egyrészt mindig a megfelelő kódlap szerint kell a karaktereket kódolnunk és megjelenítenünk is. – azt pedig nem mindig lehet egyértelműen kideríteni, hogy a szöveg milyen kódlap szerint készült; másrészt a többnyelvű szövegek esetén nem oldható meg a helyes megjelenítés. (Egy megjelenítő eszköz egyszerre csak egy kódlap szerint képes a karaktereket megjeleníteni. Új kódlap használatához üzemmódot kell váltani.)

6.5.3 UNICODE

A 16 bites UNICODE karaktertáblázatot az Apple és a Xerox által alapított Unicode Consortium hozta létre azzal a szándékkal, hogy egyetlen kódtáblában foglalják össze a világ népei által használt valamennyi ábécé összes írásjelét.

Tizenhat biten összesen 65536 karaktert lehet ábrázolni, ami azt jelenti, hogy a különféle latin, görög, héber, arab, örmény és ciril ábécéken kívül a kódtábla tartalmazza az ősi szanszkrit nyelv betűit, matematikai jeleket, a kínai, japán és koreai képírás jeleit, sőt még az egyiptomi hieroglifákat is. (Ez utóbbi még nincs a szabványban, pedig hely lenne rá.)

A 64 kilobájtos kódlap 256 bájtos blokkokra van felosztva: a különféle nyelvek ábécéinek blokkonként osztják ki a helyet. A legelső blokkot az ISO Latin 1 szabvány kapta, vagyis a csak

Latin 1 karaktereket használó szöveg karaktereinek egyik bájtja mindig nulla, a másik pedig a Latin 1 kód. (A Latin 1 kódtábla alsó 128 karaktere megegyezik a 7 bites ASCII kódtábla karaktereivel, az ASCII kompatibilitás tehát magától értetődik.)

Hatvanötezer karaktert persze nem lehet egyik napról a másikra kiosztani, így a táblázat egy része (az ábrán "lefoglalt terület" néven szerepel) egyelőre még üres.

A legnagyobb feladatot a távol-keleti ábécék kódolása jelenti: itt a szabvány készítői a CJK Research Group (CJK = China, Japan, Korea) munkájára támaszkodtak. A "Han Unification" program keretében a CJK csoport kutatói összegyűjtik a három nyelv közös írásképeit. Eddig tizenegyezer jelet kategorizáltak és helyeztek el a Unicode kódtáblában. (Azok az írásképek, melyeket csak egy nyelv használ, külön blokkban kapnak helyet.)

A kódtábla utolsó két blokkjában van egy "saját használatú" terület, aminek kiosztásáról a szabvány nem rendelkezik, ezt az alkalmazások használhatják saját céljaikra (például vezérlőkarakereknek). A kódtábla utolsó két helye (U+FFFE és U+FFFF) verziószámot tartalmaz, amit a programok a kompatibilitás ellenőrzésére használhatnak fel.

Az alapítók után hamarosan a számítástechnikai ipar meghatározó vállalatai (Adobe, Aldus, Borland, Digital, IBM, Lotus, Microsoft, NeXT, Novell, Sun, stb.) is csatlakoztak a konzorciumhoz, nemrégiben pedig az ISO is elfogadta az Unicode-ot ISO 10646 szabványának részeként. Úgy tűnik, semmi sem áll a Unicode elterjedésének útjában.

Egycsapásra megoldódott volna a szoftverek nemzektöziesítésének kérdése? Nos, úgy tűnik, hogy elvileg igen. Van azonban pár gyakorlati nehézség. Az első mindjárt az, hogy ha 8 bitesről 16 bitesre növeljük a karaktereink méretét, szöveges dokumentumaink mérete azonnal a kétszeresére növekszik. (Ez persze így csak a tiszta szöveges fájlokra igaz, a legtöbb szöveg- és kiadványszerkesztő a szövegen kívül mindenféle misztikus bináris információt is tárol a dokumentumban, amire a méretnövekedés természetesen nem vonatkozik Az ilyen dokumentumok mérete a 16 bites karakterekre való áttéréssel csak mintegy 20 - 25 százalékkal növekszik.)

A kettes számú nehézség az, hogy nem elég egy karaktert tárolni, azt meg is kell tudni jeleníteni. Ez pedig azt jelenti, hogy szükségünk van egy olyan fontkészletre, amely az összes karaktert tartalmazza, vagy legalábbis közülük azokat, amiket használni szeretnénk. Ellenkező esetben ugyanis csak egy téglalap vagy egy szóköz fog megjelenni, mivel itt nem arról van szó, hogy egy létező karaktertáblára rádefiniáltak egy másikat, mint ahogyan tették azt a 437, 850, 852 8 bites kódtáblák esetén. Azt pedig a szabvány nem írja elő, hogy mi jelenjen meg egy karakter helyett, ha az ábrázoláshoz szükséges képernyő-betűtípus nem áll rendelkezésre.

További nehézséget okozhat a fontfájlok mérete. A Windows 3.1-ben egyetlen 256 karaktert tartalmazó betűkészlet megjelenítéséhez egy 64 kilobájtos True Type fontfájl szükséges. A 16

Hosszú távon minden bizonnyal az Unicode jelenti a jövőt, a vezető számtástechnikai cégek stratégiai döntései legalább is erre engednek következtetni. A kezdetektől fogva Unicode alapú a Windows NT teljes belső rendszere (a fájlok és objektumok nevei, stb.), és Unicode karaktereket használ a SUN sikeres programozási nyelve, a Java is.