

PANNON EGYETEM, Veszprém

**Villamosmérnöki és Információs Rendszerek
Tanszék**



Digitális Rendszerek és Számítógép Architektúrák (Levelező BSc)

1. előadás: Számrendszerek,
Nem-numerikus információ ábrázolása

Előadó: Dr. Vörösházi Zsolt

voroshazi.zsolt@virt.uni-pannon.hu

Jegyzetek, segédanyagok:

- Könyvfejezetek:

- <http://www.virt.uni-pannon.hu>

- Oktatás → Tantárgyak → Digitális Rendszerek és Számítógép Architektúrák (Levelező BSc képzés)

- ([chapter02.pdf](#))

- Fóliák, óravázlatok .ppt (.pdf)

- Feltöltésük folyamatosan

Információ ábrázolás:

- A) Számrendszerek (numerikus információ):
 - I.) Egész típusú:
 - előjel nélküli,
 - 1-es komplement,
 - előjeles 2-es komplement számrendszerek
 - II.) Fixpontos,
 - III.) Lebegőpontos (IBM-32, DEC-32, IEEE-32),
 - Excess kód (exponens kódolására)
- B) Nem-numerikus információ kódolása
- C) Hiba-detektálás, és javítás (Hamming kód)
 - SEC-DED



A) Számrendszerek

Endianitás (endianness)

- A számítástechnikában, az **endianitás** (byte-sorrend a jó fordítás) az a tulajdonság, ami bizonyos adatok - többnyire kisebb egységek egymást követő sorozata - tárolási és/vagy továbbítási sorrendjéről ad leírást (pl. két protokoll, vagy busz kommunikációja). Ez a tulajdonság döntő fontosságú az integer értékeknek a számítógép memóriájában byte-onként való tárolása (egy memória címhez relatívan), továbbítása esetében
- Byte sorrend megkötés:
 - Big-Endian formátum
 - Little-Endian formátum

Háttér: Az eredeti angol kifejezés az *endianness* egy utalás arra a háborúra, amely a két szembenálló csoport között zajlik, akik közül az egyik szerint a lágytojás nagyobb/vastagabb végét (big-endian), míg a másik csoport szerint a lágytojás kisebb végét (little-endian) kell feltörni. Erről Swift ír a *Gulliver Kalandos Utazásai* című könyvében.

„Kicsi a végén” - Little-endian

- Ekkor a 32-bites „4A 3B 2C 1D” értéket a következő módon tárolják
- 100 101 102 103...”1D 2C 3B 4A”...
Így, a kevésbé jellemző ("legkisebb") byte (az angol least significant byte rövidítéséből *LSB* néven ismert) az első, és ez az 1D, tehát a *kis vég kerül előre, legkisebb címen van tárolva (0x100)*:

0x103	0x102	0x101	0x100		[31:0]
4A	3B	2C	1D		
MSB			LSB		

- **Hagyományos, általánosan** használt formátum: ha külön nem kötik ki, ezt feltételezzük! (pl. Intel, AMD, illetve ARM stb.)

„Nagy a végén” - Big-endian

- A 32 bites egész értéket, (ami legyen „4A 3B 2C 1D”, a 0x100 címtől kezdve) tároljuk a memóriában, **1 byte-os** elemi tárolókból, 1 byte-onként növekvő címekkel rendelkezik, ekkor a tárolást a következők szerint végzi:

0x103	0x102	0x101	0x100		[0:31]
1D	2C	3B	4A		
LSB			MSB		

- Ebben az esetben, a „legjellemzőbb” byte - erre általában az ismert angol kifejezést "most significant byte" használják a számítástechnikában (rövidítve *MSB*, ami itt a „4A”) - a memóriában az *legalacsonyabb címen van tárolva (0x100)*, míg a következő "jellemző byte" (3B) a következő, egyel nagyobb címen van tárolva, és így tovább.
- Bit/byte-reversed format!
- Pl: mikrovezérlők, beágyazott processzorok FPGA-kon (MicroBlaze, PowerPC) stb.

I.) Egész típusú számrendszer:

- Bináris számrendszer: '1' / '0' (I / H, T / F)
- **N biten 2^N lehetséges érték reprezentálható**
- Összehasonlító táblázat:

Table 2.1. Number of Representable Values.

<i>Number of Bits</i>	<i>Number of Representable Values</i>	<i>Machines. Uses</i>
4	16	4004, control
8	256	8080, 6800, control, communication
16	65.536	PDP11, 8086, 32020
32	4.29×10^9	IBM 370, 68020, VAX11/780
48	1.41×10^{14}	Unisys
64	1.84×10^{19}	Cray, IEEE (dp)

a.) előjel nélküli egész:

- Unsigned integer: $V_{\text{UNSIGNED INTEGER}} = \sum_{i=0}^{N-1} b_i \times 2^i$

- ahol b_i az i -edik pozícióban lévő '0' vagy '1'
- Reprezentálható értékek határa: 0-tól 2^N-1 -ig
- Helyiértékes rendszer
- Negatív számok ábrázolása nem lehetséges!

- **Pl: (Legyen $N:=6$, LE, unsigned int)**

$$101101_2 \Rightarrow 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ = 45_{10}$$


b.) 1's komplement rendszer:

- V értékű, N bites rendszer: $2^N - 1 - V$.
- „0” lesz ott ahol „1”-es volt, „1”-es lesz ott ahol „0” volt (mivel egy szám negatív alakját, bitjeinek kiegészítésével kapjuk meg).
- csupán minden bitjét negálni, (gyors műveletet)
- Értékhatár: $2^{(N-1)} - 1$ től $-(2^{(N-1)} - 1)$ ig terjed,
- Nem helyiértékes rendszer,
- kétféleképpen is lehet ábrázolni a zérust!! (-0 / +0 ellenőrzés szükséges)
- *end-around carry*: amelyben a részeredményhez kell hozzáadni a végrehajtás eredményét

1's komplement

- Pl: **N:=6, LE,**
 $V(1's) = 10\ 1101_2 = ?$

V érték	V(Unsigned int)	V(1's comp)
01 1111	31	31
01 1110	30	30
...
010 010	18	18
...
00 0010	2	2
00 0001	1	1
00 0000	0	0
11 1111	63	-0
11 1110	62	-1
11 1101	61	-2
...
10 1101	45	-18
...
100001	33	-30
100000	32	-31



Példa:

Example 2.3: One's complement arithmetic: Consider a 6-bit one's complement system. Represent 15, -15, 13, and -13 in this system. Then perform the following additions: $15 + 13$, $15 + (-13)$, $13 + (-13)$, and $13 + (-15)$.

The numbers are derived in a simple fashion:

Decimal Value	One's Complement	Comment
15	001111	Positive numbers same as two's complement.
-15	110000	Complement bits to negate.
13	001101	Positive numbers same as two's complement.
-13	110010	Complement bits to negate.

Now for the additions:

$\begin{array}{r} 15 \\ + 13 \\ \hline 28 \end{array}$	$\begin{array}{r} 001111 \\ + 001101 \\ \hline 0\ 011100 \end{array}$	<p>This proceeds just like the two's complement version.</p> <p>No carry out: number is correct, and the result is as we expect.</p>
$\begin{array}{r} 15 \\ + -13 \\ \hline 2 \end{array}$	$\begin{array}{r} 001111 \\ + 110010 \\ \hline 1\ 000001 \\ + 000001 \\ \hline 000010 \end{array}$	<p>The addition is done in the normal fashion, but the result of one is incorrect; however, the presence of a carry says we should add that as a 1 in the LSB which gives the expected result.</p>
$\begin{array}{r} 13 \\ + -13 \\ \hline 0 \end{array}$	$\begin{array}{r} 001101 \\ + 110010 \\ \hline 111111 \end{array}$	<p>This time we will add a positive number to its negative (which is just-complement) and end up with all ones — a valid zero.</p>
$\begin{array}{r} 13 \\ + -15 \\ \hline -2 \end{array}$	$\begin{array}{r} 001101 \\ + 110000 \\ \hline 0\ 111101 \end{array}$	<p>Here the positive number is smaller than the negative number, so result is negative; no carry — the value is correct.</p>

end-around: cirkuláris carry, **körkörös átvitel** (átvitel az MSB-ről LSB-re)

c.) előjeles (2's komplement) rendszer:

- 2's comp:
$$V_{2'S\ COMPLEMENT} = -b_{N-1} \times 2^{N-1} + \sum_{i=0}^{N-2} b_i \times 2^i$$

- reprezentálható értékek határa: $-(2^{(N-1)})$ től $2^{(N-1)}-1$ ig

- Ha MSB='1', negatív szám, Fólia: 2.2 táblázat

- Pl. (Legyen N:=6, LE, signed int – 2's complement)**

- $101101_2 \Rightarrow -1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -19_{10}$

*: azt jelöli, amikor az adott helyiértéken '1'-et kell kivonni még az Xi értékéből (borrow from Xi)

- Pl * * * * *

		*		0 1 0 0 1 1	
X	1 0 0 0 0 0	64	vagy	1 0 1 1 0 0	1's kompl.
- Y	- 0 1 0 0 1 1	- 19		+ 1	+1
Z	1 0 1 1 0 1	45		1 0 1 1 0 1	-19

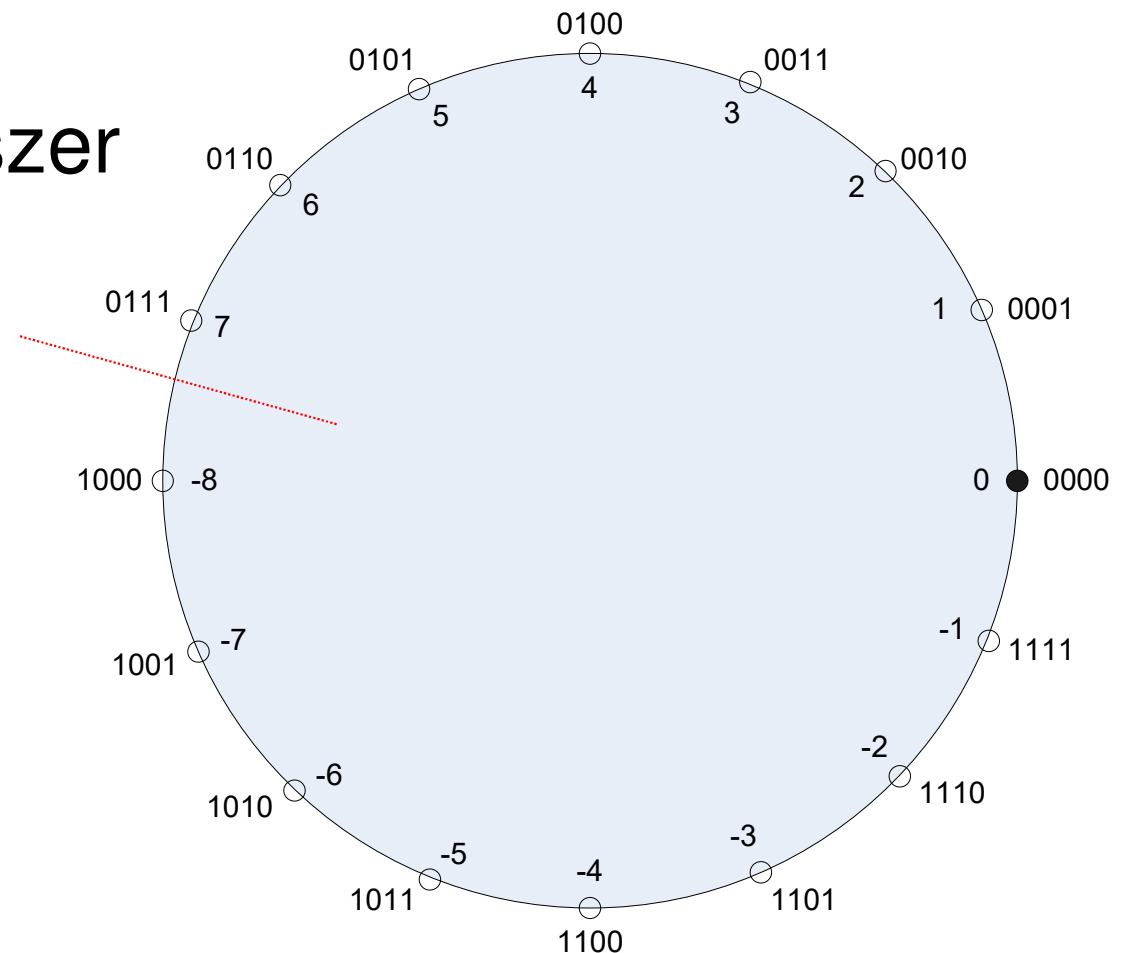
2.2 táblázat:

Table 22. 8-Bit Two's Complement Representations.

<i>Bit Pattern</i>	<i>Value</i>	<i>Note</i>
01111111	127	Largest representable value.
01111110	126	
01111101	125	
...	...	
...	...	
00000010	2	Note that leading zero indicates positive number.
00000001	1	
00000000	0	Unique representation of zero .
11111111	-1	Minus one is always all ones.
11111110	-2	Note that leading one indicates negative number.
11111101	-3	
...	...	
...	...	
10000010	-126	
10000001	-127	
10000000	-128	Smallest (most negative) representable value.

PI: Körkörös számláló (circular nature):

- 4 bites 2's komplement rendszer
- Overflow:
0111 \rightarrow 1000
- Underflow:
1000 \rightarrow 0111



II.) Fixpontos számrendszer

■ Műveletek:

- +, - : ugyanaz, mint az egész szám rdsz. esetén
- *, / : meg kell bizonyosodni arról, hogy a tizedespont helyén maradt-e

$$V_{\text{FIXED POINT}} = -b_{N-1} \times 2^{N-p-1} + \sum_{i=0}^{N-2} b_i \times 2^{i-p}$$

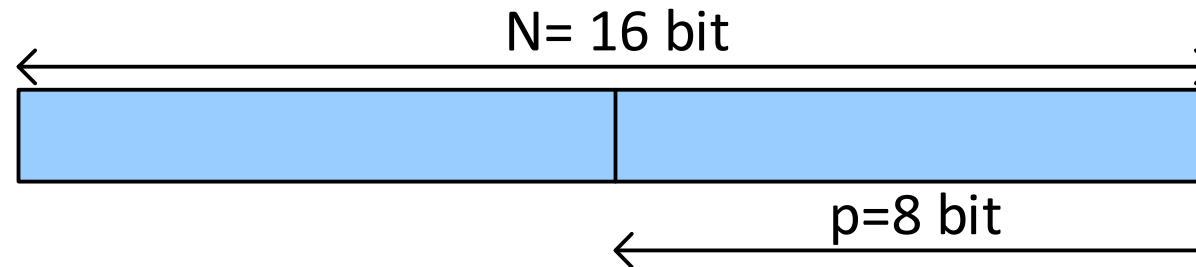
- p: radix (tizedes) pont helye, tizedes jegyek száma
- *differencia*, $\Delta r = 2^{-p}$ (számrendszer finomsága)
 - Ha $p=0 \rightarrow \Delta r=1$, egész rendszer, különben fixpontos
- Alkalmazás: fixpontos műveletvégző egységek
 - pl. jelfeldolgozás (Ti: DSP – [Texas Instruments](#)),

Példa: fixpontos rendszer

- Legyen egy **N:=16 bites, LE, 2's comp.** fixpontos rdsz. ahol **p:=8.**

Kérdés: $V(\text{smallest/largest absolute})=?$, $V(\text{smallest/largest negative})=?$, $V(\text{zero})$, $\Delta r = ?$ (*ahol lehet, dec. értékben megadva*)

- Megoldás:



- Differencia $\Delta r = \underline{2^{-8}} = 0,390625 * 10^{-2}$
- $V(\text{zero}) = \mathbf{00000000.00000000} = 0.0$
- $V(\text{smallest absolute}) = \mathbf{00000000.00000001} = \underline{2^{-8}} = 0,390625 * 10^{-2}$
- $V(\text{largest absolute}) = \mathbf{01111111.11111111} \approx \underline{128} = 128 - \Delta r$
- $V(\text{smallest/"most" negative}) = \mathbf{10000000.00000000} = \underline{-2^7} = \underline{-128}$
- $V(\text{largest/"least" negative}) = \mathbf{11111111.11111111} = \underline{-2^{-8}} =$
 $- 0,390625 * 10^{-2}$

III. Lebegőpontos rendszer: Excess-kód

- **Lebegőpontos** számok *kitevőit* (*exponens*-eit) tárolják / kódolják ezzel a módszerrel. Cél: a kitevő NE legyen negatív, ezért eltolással oldják meg.
 - S: a reprezentálni kívánt érték (kódolt eredmény), amit tárolunk
 - V: a szám (exponens) valódi értéke, E: az excess (offset/eltolás)
 - $S = V + E$.
- Két számot összeadunk, akkor a következő történik:
$$S1 + S2 = (V1 + E) + (V2 + E) = (V1 + V2) + 2 \times E$$
- pontos eredmény: $[(V1+V2) + E]$ (ki kell vonnunk E-t!)
- Fólia: 2.4, 2.5, 2.6-os példák

Példa 2.4:

Example 2.4: Number representation in excess codes: What is the representation of $+37_{10}$ in an 8-bit excess 128 code? What is the representation of -23_{10} in an 8-bit excess 128 code? What is the sum of the two numbers, in the 8-bit excess 128 code?

An 8-bit unsigned number can represent values between 0 and 255. The excess representation can then represent values from -128 to +127.

+ 128	10000000	This is the excess.
+ 37	00100101	The value to be represented.
<u>165</u>	10100101	The representation of 37_{10} in excess 128 code.
+ 128	10000000	This is the excess.
- 23	00010111	The value to be represented.
<u>105</u>	01101001	The representation of -23_{10} in excess 128 code.
165	10100101	This is +37 in excess 128.
<u>+ 105</u>	<u>+ 01101001</u>	This is -23 in excess 128.
270	1 00001110	Note the carry out in this operation. 270 is too big to represent in 8 bits; to correct for the 2 x E that is in this sum, subtract 128.
<u>- 128</u>	<u>- 10000000</u>	In binary, is this add or subtract?
142	10001110	This is the representation of 14, the correct result in excess 128.

Táblázat

2.3: BCD

Table 2.3. Binary Coded Decimal
(BCD) Representations.

<i>Bit Pattern</i>	<i>Value</i>
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	Not valid
1011	Not valid
1100	Not valid
1101	Not valid
1110	Not valid
1111	Not valid

Példa 2.5:

Example 2.5: BCD excess 3 system: Consider a system that works with 3-digit decimal numbers, and it stores the digits in excess 3 format. What is the representation of **573**? What is the representation of **142**? Add the two numbers, and give the correct result in excess 3 format.

The numbers are handled on a digit-by-digit basis, with the excess being included with each digit:

<i>Decimal</i>	<i>Binary</i>	
+ 3 3 3	0011 0011 0011	This is the excess.
5 7 3	0101 0111 0011	And the number to be represented.
<hr/>	<hr/>	
8 10 6	1000 1010 0110	The excess 3 representation.
+ 3 3 3	0011 0011 0011	This is the excess.
1 4 2	0001 0100 0010	This is the number to be represented.
<hr/>	<hr/>	
4 7 5	0100 0111 0101	The excess 3 representation.
573	1000 1010 0110	Do the addition in decimal and in
+ 142	0100 0111 0101	binary. Correct as needed to
<hr/>	<hr/>	make output correct.
715	1100 1001 1011	

Carry out of second set of 4 bits indicates that the most significant digit should be incremented by one. Also indicates that this value is **correct** as it stands (since $2 \times E = 6$ and the **carry** out indicates that the number overflowed into the next digit) so we need to add 3. Therefore, the MSD needs to be incremented by one and decremented by 3; the middle digit needs to have 3 added; and the LSD needs to be decremented by 3.

$$\begin{array}{r}
 (-3+1) \quad (+3) \quad (-3) \\
 \hline
 -0010 + 0011 - 0011 \\
 \hline
 1010 \quad 0100 \quad 1000 \\
 10 \quad 4 \quad 8
 \end{array}$$

Which is the correct excess 3 representation for 715_{10} .

MSD: Most significant digit

LSD: Least significant digit

Lebegőpontos rendszer (FPN):

- 7 különböző tényező: *a számrendszer alapja, előjele és nagysága, a mantissza alapja, előjele és hosszúsága, ill. a kitevő alapja.*

- matematikai jelölés:

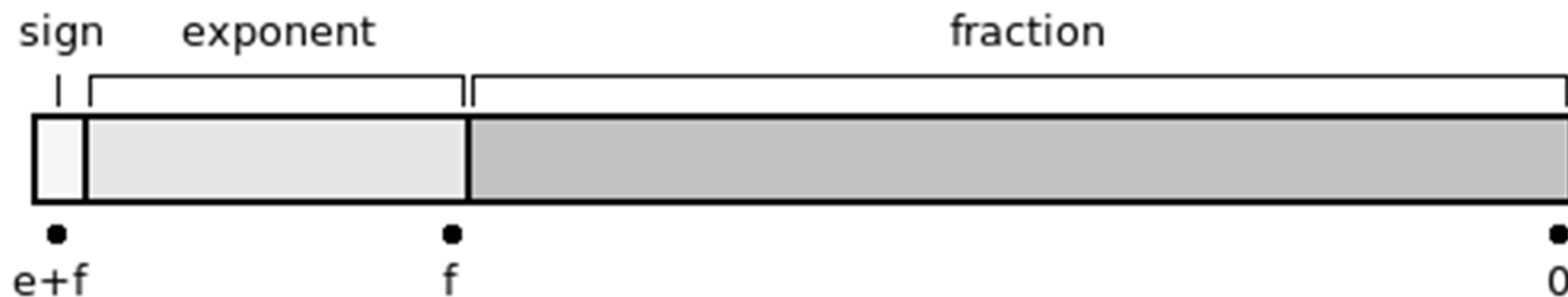
$$(\text{előjel}) \text{ Mantissza} \times \text{Alap}^{\text{Kitevő}}$$

- Fixpontosnál nagyságrendekkel kisebb vagy nagyobb számok ábrázolására is mód van:

- Pl: Avogadro-szám: $6.022 \cdot 10^{23}$
- Pl: proton tömege $1.673 \cdot 10^{-24}$ g
- Normalizált rendszerek: pl. DEC-32, IEEE-32, [IBM-32](#)
 - **32-bites, vagy egyszeres pontosságú – float (C)**
 - 64-bites, vagy dupla pontosságú – double (C) – nem tárgyaljuk

IEEE 754-1985

- Szabvány a bináris lebegőpontos számok tárolására, amely tartalmazza még:
 - negatív zérust: $-0 = 111\dots1$ (két zérus: $+0$ is)
 - Normalizálatlan (denormal) számok
 - NaN: nem szám (pl. $\frac{\pm 0}{\pm 0} = \text{NaN}$; vagy $\pm 0 \times \pm \infty = \text{NaN}$)
 - $\pm \infty$



Sign-magnitude format („előjel-hossz” formátum): az előjel külön biten kerül tárolásra (MSB), exponens kódolt (Excess-el „eltolt”), majd a törtrész következik végül. Fontos a sorrendjük!

IEEE-754 szabvány

- Lebegőpontos szám tárolási formátumai:
 - 16-bites, (half) pontosságú,
 - **32-bites (single), vagy egyszeres pontosságú,**
 - 64-bites (double), vagy dupla pontosságú,
 - 128-bites (quadruple), vagy négyszeres, pontosságú
 - Kibővített (extended).

Lebegőpontos rendszer jellemzői

- Számrendszer / kitevő alapja: r_b r_e
- Mantissa értéke: $V_M = \sum_{i=0}^{N-1} d_i \times r_b^{i-p}$
 - Maximális: $V_{M_{\max}} = 0.d_m d_m d_m \dots = (1 - r_b^{-m})$
 - Minimális: $V_{M_{\min}} = 0.100 \dots = 1/r_b$
 - Radix pont helye: p
 - (a p helye az exponens értékével van összefüggésben!)
 - Mantissa bitjeinek száma: m
- Exponens értéke (max / min): V_E $V_{E_{\max}}$ $V_{E_{\min}}$
- Lebegőpontos szám értéke: $V_{\text{FPN}} = (-1)^{\text{SIGN}} V_M \times r_b^{V_E}$

Normalizált lebegőpontos rendszer jellemző paraméterei:

- Ábrázolható maximális érték: $V_{FPN(MAX)} = V_{M(MAX)} \times r_b^{V_{E(MAX)}}$
- Ábrázolható minimális érték: $V_{FPN(MIN)} = V_{M(MIN)} \times r_b^{V_{E(MIN)}}$
- Legális mantisszák száma: $NLM_{FPN} = (r_b - 1) \times r_b^{m-1}$
- Legális exponensek száma: $NLE_{FPN} = V_{E(MAX)} + |V_{E(MIN)}| + 1_{ZERO}$
- Ábrázolható értékek száma: $NRV_{FPN} = NLM_{FPN} \times NLE_{FPN}$

■ Normalizálás: mantissza értékét általában $[0... \sim 1]$ közé

■ Pl: $32\,768_{10} = 0.32768 \cdot 10^5 = 3.2768 \cdot 10^4 = 32.768 \cdot 10^3 = 327.68 \cdot 10^2 = 3267.8 \cdot 10^1$ (érvényes alakok)

Példa: normalizált lebegőpontos rendszer

- Adott: Legyen $r_b = 10$, $r_e = 10$, $m = 3$, $e = 2$
 - Tfh. $p=m$, ill. a legbaloldalibb jegye a mantisszának '1'
- Kérdés: jellemző paraméterek?

- Megoldás: $V_{M(MAX)} = 0.999 = 1.000 - 10^{-3}$

$$V_{M(MIN)} = 0.100$$

$$V_{E(MAX)} = (r_e^e - 1) = 99$$

$$V_{E(MIN)} = -(r_e^e - 1) = -99$$

$$V_{FPN(MAX)} = 0.999 \times 10^{99}$$

$$V_{FPN(MIN)} = 0.100 \times 10^{-99}$$

} Egyszerűsítésként,
itt még nincs
használatban az
Excess kódolás!

$$NLM_{FPN} = 9 \times 10 \times 10 = 900 = 9 \times 10^2$$

$$NLE_{FPN} = 99 + |-99| + 1_{ZERO} = 199$$

$$NRV_{FPN} = 2 \times 900 \times 199 = 358,200$$

Normalizált lebegőpontos rdsz.

Table 24. 6-Bit Normalized Floating Point System, Base 2.

$$r_b = 2, r_e = 2, m = 4, e = 2$$

				$V_E \rightarrow$	00	01	10	11
				$2^{V_E} \rightarrow$	1	2	4	8
V_M base 2				V_M	$V_M \times 2^{V_E}$			
1	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	1	2	4
1	0	0	1	$\frac{9}{16}$	$\frac{9}{16}$	$1\frac{1}{8}$	$2\frac{1}{4}$	$4\frac{1}{2}$
1	0	1	0	$\frac{5}{8}$	$\frac{5}{8}$	$1\frac{1}{4}$	$2\frac{1}{2}$	5
1	0	1	1	$\frac{11}{16}$	$\frac{11}{16}$	$1\frac{3}{8}$	$2\frac{3}{4}$	$5\frac{1}{2}$
1	1	0	0	$\frac{3}{4}$	$\frac{3}{4}$	$1\frac{1}{2}$	3	6
1	1	0	1	$\frac{13}{16}$	$\frac{13}{16}$	$1\frac{5}{8}$	$3\frac{1}{4}$	$6\frac{1}{2}$
1	1	1	0	$\frac{7}{8}$	$\frac{7}{8}$	$1\frac{3}{4}$	$3\frac{1}{2}$	7
1	1	1	1	$\frac{15}{16}$	$\frac{15}{16}$	$1\frac{7}{8}$	$3\frac{3}{4}$	$7\frac{1}{2}$

$$\text{Smallest fraction} = 0.1000_2 = \frac{1}{2}$$

$$\text{Largest fraction} = 0.1111_2 = \frac{15}{16}$$

$$\text{Smallest number} = 0.1000_2 \times 2^0 = \frac{1}{2}$$

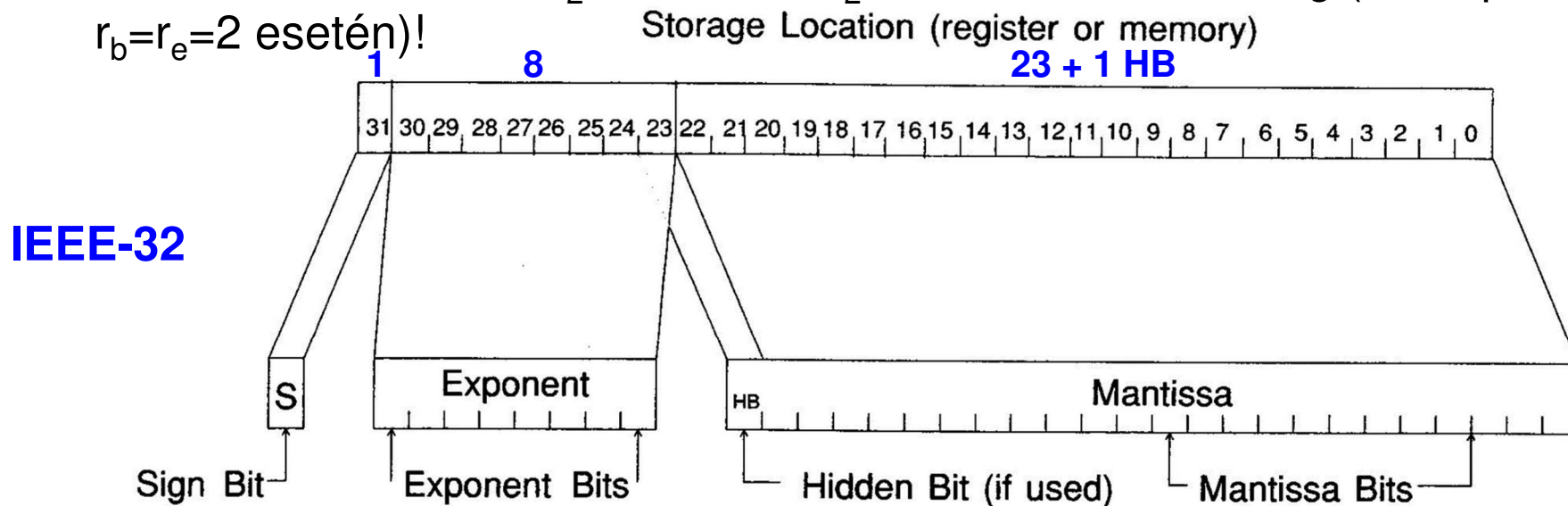
$$\text{Largest number} = 0.1111_2 \times 2^3 = 7\frac{1}{2}$$

$$\text{Number of fractions} = 1 \times 2 \times 2 \times 2 = 8$$

$$\text{Number of values} = 8 \times 4 = 32$$

Normalizált lebegőpontos ábrázolás: Rejtett (**hidden bit**) technika

- „Vezető” ‘1’-sek számát a legtöbb rendszer tervezésekor konstans értéként definiálják (HB=1), DE nem tárolják!
 - Ilyen a *mantissa legmagasabb helyiértékű bitje*, **rejtett (hidden/implicit: HB)** bitnek hívunk, közvetlenül az exponensbitek mögött helyezkedik el.
 - Ez, ha HB=1 van beállítva (bizonyos rendszerek esetén, pl. IEEE, rögzített), duplájára nő a legális mantisszák, így az ábrázolható értékek száma is.
 - Viszont a nullát nem könnyen tudnánk reprezentálni, mivel a legkisebb ábrázolható formátum a „0 00 000”, ami a HB ‘1’-es konstansként való definiálása miatt $1.000_2 \times 2^{-1} = 0.1000_2 \times 2^0 = \frac{1}{2}$ -nek felel meg (m=4, p=3, e=2, $r_b=r_e=2$ esetén)!



Rejtett bit / Zérus érték

- Probléma: a zérus (közelítő) ábrázolása
- Hogy tárolható mégis a zérus? → **Excess** 2^{e-1} kódolás esetén, **ha $r_e=2$!**
 - Bias tartománya: $-(2^{e-1}-1) - (2^{e-1}-1)$ –ig terjed, ha $r_e:=2$!
 - Ha az **exponens bitek mindegyike zérus ($V_E=0$)** → az ábrázolt lebegőpontos számot ($V_{FPN} = 0$) is **zérusnak** tekintjük!
- Rendszer tervezése során definiálják a használatát
 - No hidden bit: Intel Pentium, Motorola 68000 (CISC)
 - Hidden bit: IEEE-32 számrendszer (754-es formátum)
 - DEC (VAX, PDP gépei)

Példa: 2-es alapú DEC 32-bites, normalizált lebegőpontos rendszer

- Adott: $r_b=2$, $r_e=2$, $m=24$ (HB nélkül), $/p=24$ ($m=p!$), $e=8$, az exponenst tároljuk Excess-128 kódolással, és a számokat tároljuk "előjel-hossz" formátumban (~tekintsük a mantisszát pozitívnak).

$$V_{M(MIN)} = 0.\underbrace{1000\dots_2}_{24db} = 1/2$$

$$V_{M(MAX)} = 0.1111\dots_2 = 0.999999940395 = 1.0 - 2^{-24}$$

$$\left. \begin{aligned} V_{E(MIN)} &= -(r_e^{e-1} - 1) = -(2^{8-1} - 1) = -127 \\ V_{E(MAX)} &= r_e^{e-1} - 1 = 2^{8-1} - 1 = 127 \end{aligned} \right\} \text{Excess-128}$$

$$V_{FPN(MIN)} = 0.1000\dots_2 \times 2^{-127} = 2.9387 \times 10^{-39}$$

$$V_{FPN(MAX)} = 0.1111\dots_2 \times 2^{+127} = 1.7014 \times 10^{38}$$

$$NLM_{FPN} = 2^{23} = 8,388,608$$

$$NLE_{FPN} = 127 + |-127| + 1_{ZERO} = 255 = (r_e^e - 1) = 2^8 - 1$$

$$NRV_{FPN} = 2^{23} \times (2^8 - 1) = 2.13909 \times 10^9$$

Példa: 2-es alapú IEEE-32 bites normalizált lebegőpontos rendszer

- Adott: $r_b=2$, $r_e=2$, $m=24$, de $p=23!$ (**+HB='1'**!), Tehát a rejtett bitnek itt lesz szerepe! $e=8$, az exponenst tároljuk Excess-127 kódolással, és a számokat tároljuk "előjel-hossz" formátumban (~tekintsük a mantisszát pozitívnak).

$$V_{M(MIN)} = \underbrace{1.000\dots_2}_{23db} = 1$$

$$V_{M(MAX)} = 1.111\dots_2 = 1.99999988 = 2.0 - 2^{-23}$$

$$\left. \begin{array}{l} V_{E(MIN)} = -126 \\ V_{E(MAX)} = 127 \end{array} \right\} \begin{array}{l} // \text{Zérus pontosabb ábrázolása miatt } -127+1 \\ \text{Excess-127} = \text{Excess}(128-1) \end{array}$$

$$V_{FPN(MIN)} = 1.000\dots_2 \times 2^{-126} = 1.1755 \times 10^{-38} \quad // 4 \times \text{DEC!}$$

$$V_{FPN(MAX)} = 1.111\dots_2 \times 2^{+127} = 3.4028 \times 10^{38} \quad // 2 \times \text{DEC!}$$

$$NLM_{FPN} = 2^{23} = 8,388,608$$

$$NLE_{FPN} = 127 + |-126| + 1_{ZERO} = 254 = (r_e^e - 2) = 2^8 - 2$$

$$NRV_{FPN} = 2^{23} \times (2^8 - 2) = 2.1307 \times 10^9$$

Példa: 16-os alapú IBM-32 bites normalizált lebegőpontos rendszer

- Adott: $r_b=16$, $r_e=2$, $m=6$ (HB nélkül), $/p=m=6!/, e=7$, az exponenst tároljuk Excess-64 kódolással, és a számokat tároljuk "előjel-hossz" formátumban (~tekintsük a mantisszát pozitívnak).

$$V_{M(MIN)} = 0.100000_{16} = 1/16$$

$$V_{M(MAX)} = 0.FFFFFFFF_{16} = 0.999999940395 = 1.0 - 16^{-6}$$

$$V_{E(MIN)} = -(r_e^{e-1} - 1) = -(2^{7-1} - 1) = -63$$

$$V_{E(MAX)} = r_e^{e-1} - 1 = 2^{7-1} - 1 = 63$$

Excess-64

$$V_{FPN(MIN)} = 0.100000 \times 16^{-63} = 8.636 \times 10^{-78}$$

$$V_{FPN(MAX)} = 0.FFFFFFFF_{16} \times 16^{+63} = 7.237 \times 10^{75}$$

Bővebb tartomány mint a DEC!

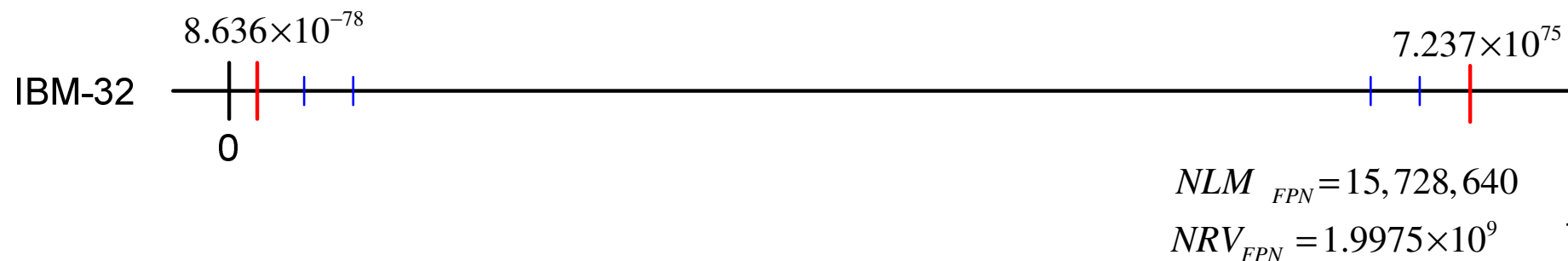
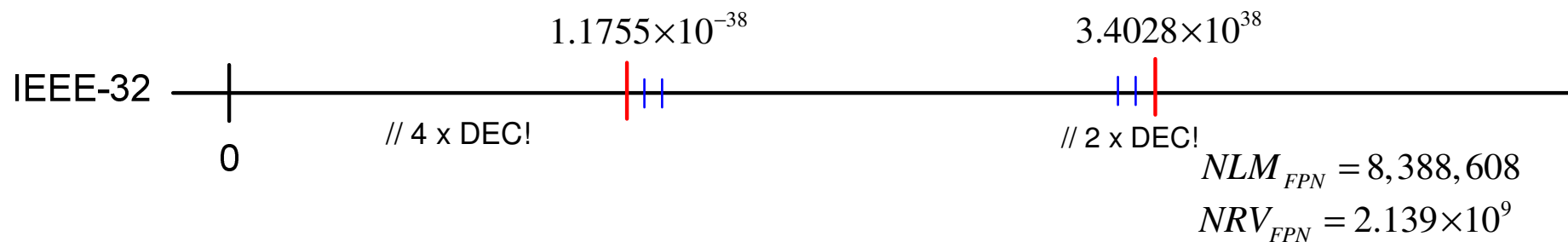
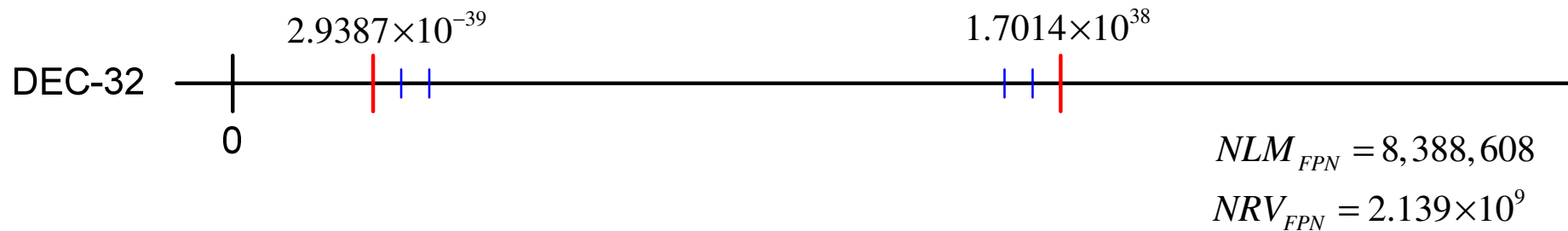
$$NLM_{FPN} = 15 \times 16^5 = 15,728,640$$

$$NLE_{FPN} = 63 + |-63| + 1_{ZERO} = 127 = 2^7 - 1$$

$$NRV_{FPN} = 15 \times 16^5 \times (2^7 - 1) = 1.9975 \times 10^9$$

7%-al kevesebb mint a DEC!! **33**

Lebegőpontos számrendszerek összehasonlítása (ha FPN előjele pozitív):



7%-al kevesebb mint a DEC!!

- *DEC*: zérushoz közelítve szakadás (az első érték aránytalanul messze van)
- Ezt küszöböli ki az *IEEE* rendszer (Normalizálatlan tartományban lineárisan tart a zérushoz): ezért használja fel a $V_E = 126$ értéket

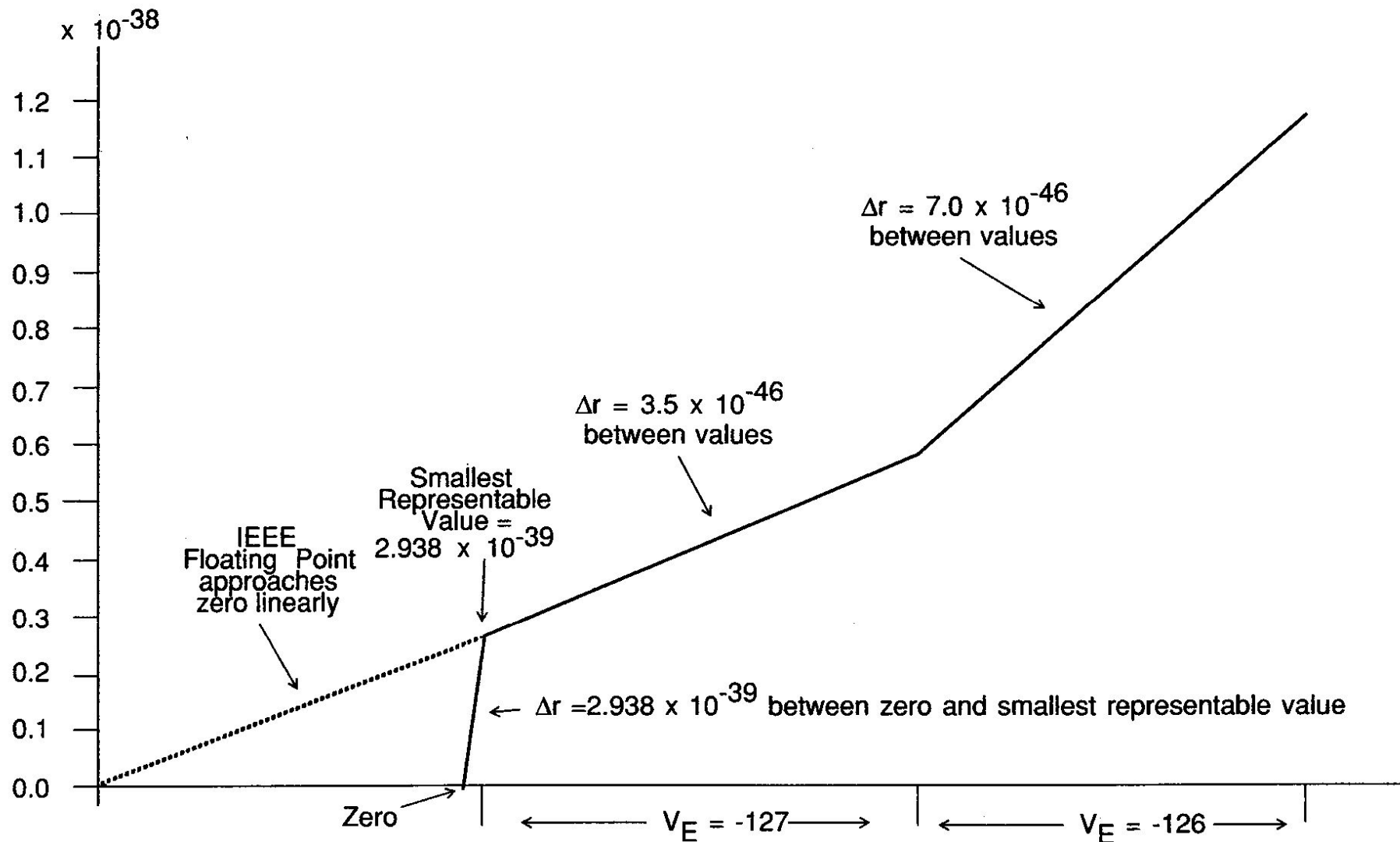


Figure 2.3. Values of the DEC Normalized Floating Point System Near Zero.

Példa: IEEE-32 bites normalizált lebegőpontos rendszer (folyt.)

- $V_E = [-126, 127] \rightarrow [1, 254]$ az Excess127-el eltolt exponens tartomány
- Speciális jelentőség:
 - $V_E = 0$ értékénél (zérus ábrázolása)
 - $V_E = [255]$ értékénél lehetőség van bizonyos információk tárolására:

$$(+\infty) + (+7) = (+\infty)$$

$$(+\infty) \times (-2) = (-\infty)$$

$$(+\infty) \times 0 = \text{NaN}$$

$$0 / 0 = \text{NaN}$$

$$\text{Sqrt}(-1) = \text{NaN}$$

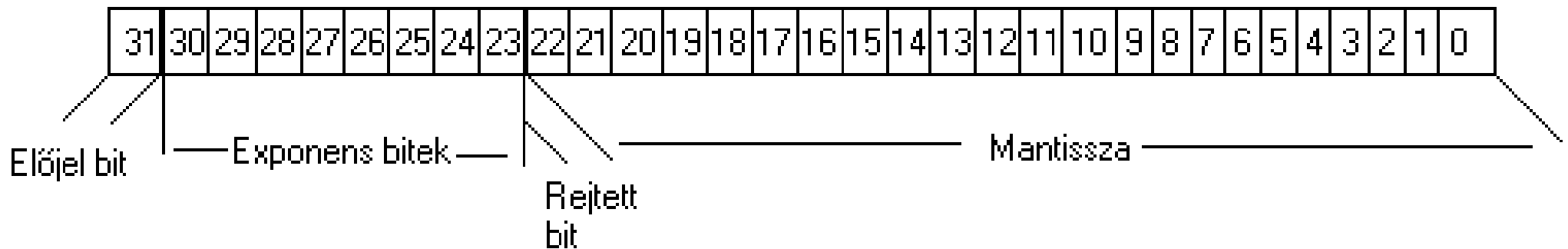
V_E [255]	S (előjel)	Ábrázolás jelentése
$\neq 0$	X	Nem egy szám (NaN)
0	0	$+\infty$
0	1	$-\infty$

Különböző pontosságú IEEE lebegőpontos rendszerek

Típus	Sign (s)	Exponens (e)	Excess-kód (Exc)	Mantissza (p < m)	Teljes szóhossz
Half (IEEE 754r)	1	5	15	10	16
Single	1	8	127	23	32
Double	1	11	1023	52	64
Quad	1	15	16383	112	128

Példák:

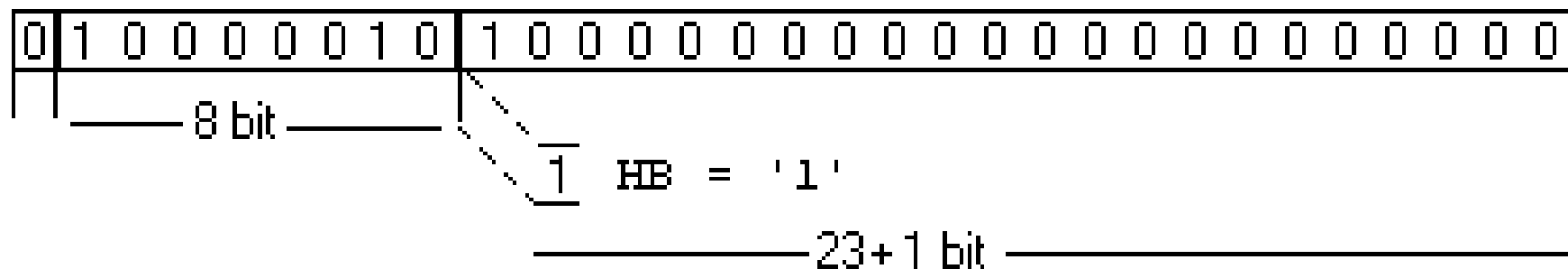
- Adja meg a következő szám (decimális '12') bináris bitmintázatát a különböző 32-bites DEC, IEEE és IBM lebegőpontos formátumokban.



Példa (folyt) IEEE-32

- IEEE-32 lebegőpontos számrendszerben: $r_e = r_b = 2$, $m = 24$, $p = 23$ ($p < m$) (és a rejtett bit beállítva, $HB = '1'$, de nem tároljuk el), (ekkor a mantisszák tartománya 1-től majdnem 2-ig terjedhet), $e = 8$ az exponens bitek száma Excess-127 kódban tárolva.
- $12_{10} = 1100_2 = 1.100 * 2^3 \Rightarrow$ a kitevő $3_{10} = 11_2$ Excess-127 kódja: $1111111 + 11 = 10000010$.

Tehát a fenti formának megfelelően:





B) Nem-numerikus információ kódolása



Nem-numerikus információk

- Szöveges,
- Logikai (Boolean) információt,
- Grafikus szimbólumokat,
- és a címeket, vezérlési karaktereket értjük alattuk.

Szöveges információ

- Minimális: 14 karakterből álló halmazban: számjegy (0-9), tizedes pont, pozitív ill. negatív jel, és üres karakter.
- + ábécé (A-Z), a központozás, címkék és a formátumvezérlő karakterek (mint pl. vessző, tabulátor, (CR: Carriage Return) kocsivissza, soremelés (LF: Line Feed) , lapemelés (FF: From Feed), zárójel)
- Így elemek száma 46: 6 biten ábrázolható
$$\lceil \log_2 46 \rceil = 6 \text{ bit}$$
- De 7 biten tárolva már kisbetűs, mind pedig a nagybetűs karaktereket is magába foglalja.

Szöveges információ kódolás

- BCD (Binary Coded Decimal): 6-biten
 - nagybetűk, számok, és speciális karakterek
- EBCDIC (Extended Binary Coded Decimal Interchange Code): 8-biten (A. Függelék)
 - + kisbetűs karaktereket és kiegészítő-információkat
 - 256 értékből nincs mindegyik kihasználva
 - Továbbá I és R betűknél szakadás van!
- ASCII (American Standard Code for Information Interchange): (A függelék) – alap 7-biten / extended 8-biten
- UTF-n (Universal Transformation Format): változó hosszúságú karakterkészlet (többnyelvűség támogatása)

EBCDIC

HEX DIGITS	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	SP SP010000	& SM030000	- SP100000	ø LOE10000	Ø LOE20000	° SM190000	μ SM170000	€ SC040000	{ SM110000	}	\ SM070000	0 ND100000
-1	9SP SP300000	é LE150000	/ SP120000	É LE120000	a LA010000	j LJ010000	~	£	A	J	÷	1
-2	â LA100000	ê LE100000	Ã LA180000	Ê LE180000	b LB010000	k LK010000				K	S	2
-3	ã LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000				L	T	3
-4	ä LA130000	è LE130000	Å LA140000	È LE140000	d LD010000	m LM010000				M	U	4
-5	á LA110000	í LI100000	Á LA120000	Í LI120000	e LE010000	n LN010000	ˆ LV010000	˚ SM240000	˘ LE020000	N	V	5
-6	â LA180000	î LI180000	Ã LA200000	Î LI180000	f LF010000	o LO010000	w LW010000	ŕ SM250000	F LF020000	O LO020000	W LW020000	6
-7	ã LA270000	ï LI170000	Ä LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	Œ LO220000	G LG020000	P LP020000	X LX020000	7
-8	ç LC410000	ì LI130000	Ç LC420000	Ï LI140000	h LH010000	q LQ010000	y LY010000	œ LO910000	H LH020000	Q LQ020000	Y LY020000	8
-9	ñ LN180000	ß LS010000	Ñ LN200000	ˆ SD130000	i LI010000	r LR010000	z LZ010000	ÿ LY180000	I LI020000	R LR020000	Z LZ020000	9
-A	Ý LY120000	! SP020000	Š LS220000	: SP130000	« SP170000	© SM210000	ı SP030000	¬ SM980000	ſ SP020000	ı ND011800	2 ND021800	3 ND031800
-B	. SP110000	Š SC030000	, SP080000	# SM010000	» SP180000	® SM200000	ı SP180000	š LS210000	ó LO150000	û LU150000	ô LO180000	ù LU180000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LOE30000	æ LA010000	Ð LOE40000	˘ SD010000	ó LO170000	û LU170000	ô LO180000	ù LU180000
-D	(SP080000) SP070000	ˉ SP090000	˙ SP050000	ý LY110000	ž LZ010000	[SM080000] SM090000	ò LO130000	ú LU130000	ó LO140000	ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT020000	Æ LA020000	Þ LT040000	Ž LZ020000	ó LO110000	ú LU110000	ó LO120000	ù LU120000
-F	SM130000	^ SD100000	? SP150000	" SP040000	± SA020000	€ SC200000	® SM030000	× SA070000	ó LO190000	ÿ LY170000	ô LO200000	€



Extended ASCII (1 byte)

	Standard							Extended							
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		SP	0	@	P	`	p	■	ك	SP	0	آ	ذ	ا	ف
1		!	1	A	Q	a	q	ل	و	،	+	ء	ر	ن	ق
2		"	2	B	R	b	r	،	و	¢	z	آ	ز	هـ	،
3		#	3	C	S	c	s	f	"	E	a	آ	س	م	،
4		\$	4	D	T	d	t	"	"	H	'	و	ثي	ن	ô
5		%	5	E	U	e	u	...	•	¥	µ	ا	ص	هـ	"
6		&	6	F	V	f	v	†	-	.	¶	ئ	ض	و	،
7		'	7	G	W	g	w	‡	-	§	.	ا	x	ç	ا.
8		(8	H	X	h	x	^	-	:	.	ب	ط	è	"
9)	9	I	Y	i	y	‰	™	©	ا	هـ	ظ	é	ù
A		*	:	J	Z	j	z	§	™	®	؛	ن	ع	ê	°
B		+	:	K	[k	{	<	>	«	»	ث	ع	ë	û
C		,	<	L	\	l		œ	œ	¬	¼	ج	ا	ى	ü
D		-	=	M]	m	}	€		-	½	ح	ف	ي	ı
E		.	>	N	^	n	~	ž	ř	®	¾	خ	ق	î	ı
F		/	?	O	_	o		■	ÿ	-	?	د	ك	i	ÿ

Legend



Code point contains a non-printable control character.



Code point is unoccupied; reserved for future use.



C) Hamming hibakódolás – Hiba-detektálás, és javítás

Hibakódolás - Hibadetektálás és Javítás

- N bit segítségével 2^N különböző érték, cím, vagy utasítás ábrázolható
- 1 bittel növelve (N+1) bit esetén: 2^N -ről 2^{N+1} -re: tehát megduplázódik az ábrázolható értékek tartománya
- *Redundancia*: többlet bitek segítségével lehet a hibákat detektálni, ill. akár javítani is

Paritás bit

- Legegyszerűbb hibafelismerési eljárás, a paritásbit átvitele. Két lehetőség:

	Kód	Paritásbit
<input type="checkbox"/> páros paritás	1 1 0 1	1
<input type="checkbox"/> páratlan paritás	1 1 0 1	0

- **Páros paritás:** az '1'-esek száma páros.
 - A kódszóban lévő '1'-esek számát '1' vagy '0' hozzáadásával **páros**sá egészítjük ki. '0' a paritásbit, ha az '1'-esek száma páros volt.
- **Páratlan paritás:** az '1'-esek száma páratlan.
 - A kódszóban lévő '1'-esek számát '1' vagy '0' hozzáadásával **páratlan**ná egészítjük ki. '1' a paritásbit, ha az '1'-esek száma páros volt.

Paritás bit generáló áramkör

■ Paritásbit képzése:

- ANTIVALENCIA (XOR) művelet alkalmazása a kódszó bitjeire, pl. 'n' adatbit esetén „n-1”-szer!

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

■ Példa:

Kódszó

Paritásbit(P)



$$0\ 0\ 0\ 1\ ? \rightarrow 0 \oplus 0 \oplus 0 \oplus 1 = 1$$

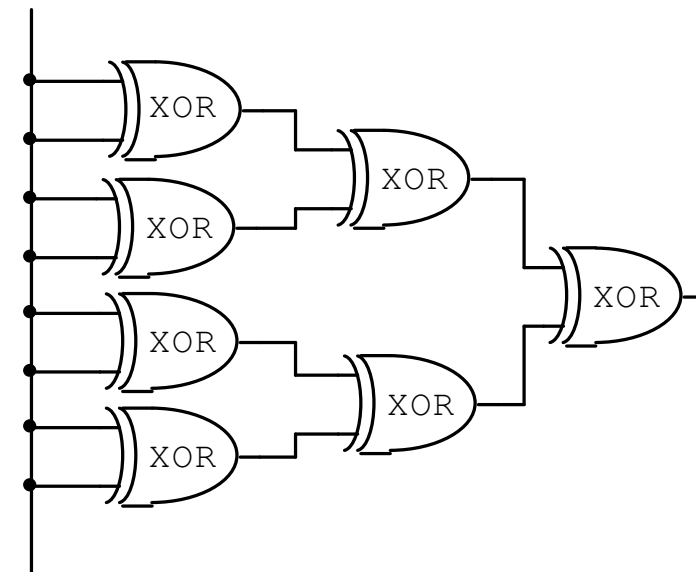
$$0\ 1\ 1\ 0\ ? \rightarrow 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$1\ 1\ 1\ 0\ ? \rightarrow 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

Páros paritás!

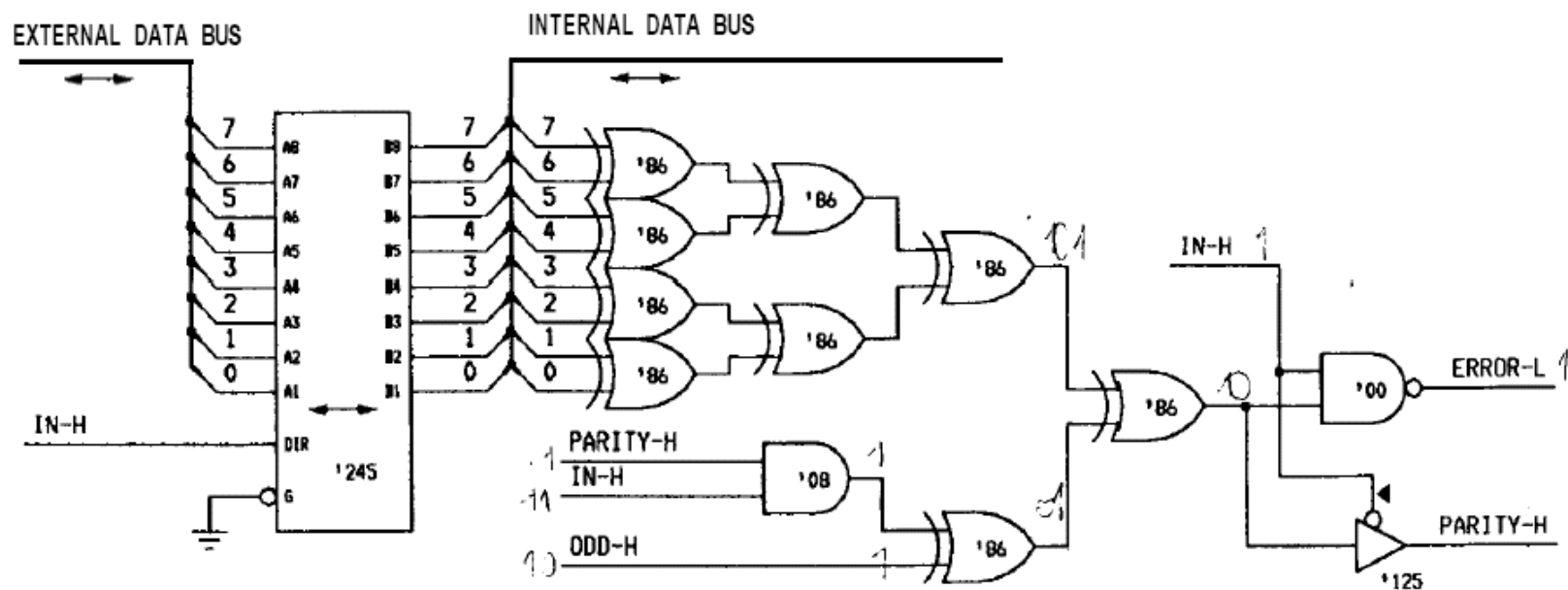
Paritás bit ellenőrzés

- Páros v. páratlan paritás: N bites információ egy kiegészítő bittel bővül → *egyszeres hiba* felismerése
- Hibák lehetséges okai:
 - leragadásból: '0'-ból '1'-es lesz, vagy fordítva
 - ideiglenes, tranziens jellegű hiba
 - áthallás (crosstalk)
- Pl: 8-adatbithez *páros* paritásbit generálás
IC 74'180 / 74'280. <http://alldatasheet.com> 9-bites paritás ellenőrző
(XOR gate: IC 74'86)



8 bites adatút kétirányú paritásbit ellenőrzéssel

- Paritásbit dekódoló és generáló képesség
(Megj: L/H jelöli, hogy adott vezérlő jel mely feszültségszinten aktív)
- IN-H=1 →
 - bemeneten PARITY-H=1 paritás ellenőrzés (engedélyezés)
 - kimeneten PARITY-H a generált paritás bit
- Ha ODD-H=1 páratlan paritást van beállítva (ODD-H=0 páros)
- Hiba esetén: ERROR-L=1 ! (ERROR_L=0 nincs hiba)

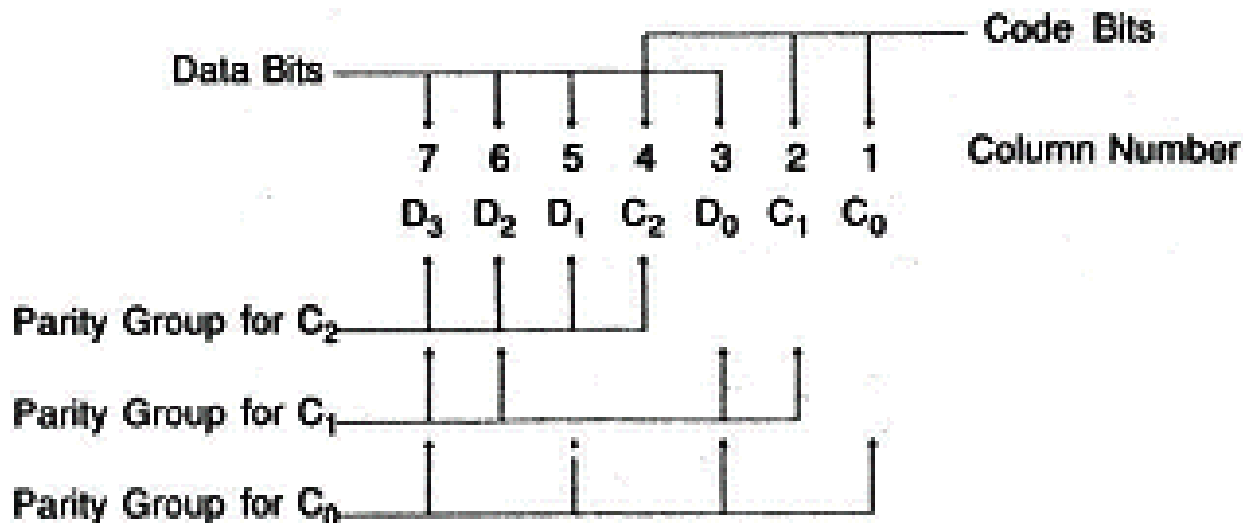


Hamming kód

- Háttér: több *redundáns* bittel nemcsak a hiba meglétét, és helyét tudjuk detektálni, hanem akár a hibás bitet javítani is tudjuk
- **Hamming kód**: egy biten tároljuk a bitmintázatok azonos helyiértékű bitjeinek különbségét, tehát egybites hibát lehet vele *javítani*.
 - SEC-DED: Single Error Correction / Double Error Detection
 - Bitcsoportokon történő paritás ellenőrzésen alapul

7-bites Hamming kódú kódszó konstruálása (pl. 4 – bites adatszóra)

- 2^N-1 bites Hamming kód: N kódbit, 2^N-N-1 adatbit
- Összesen pl. 7 biten *4 adatbitet* (D_0, D_1, D_2, D_3), *3 kódbittel* (C_0, C_1, C_2) kódolunk
- C_i kódbitek a bináris súlyuknak megfelelő bitpozíciókban
- A maradék pozíciókat rendre adatbitekkel töltjük fel (D_i).



Paritás-csoportok	Bit pozíciók	Bitek jelölései
0	1, 3, 5, 7	C_0, D_0, D_1, D_3
1	2, 3, 6, 7	C_1, D_0, D_2, D_3
2	4, 5, 6, 7	C_2, D_1, D_2, D_3

Pl. 1/a) Hamming kódú hibajavító áramkör tervezése (Little Endian)

- 3 kódbitünk van, így írásnál 3 paritásbit generáló-, míg olvasásnál 3 paritás-ellenőrző áramkör kell.
- Példa: bemeneti adatbit-mintázatunk 0101 (D_3 - D_0). **LE!**
 - Mivel páratlan paritást alkalmazunk, a megfelelő helyen szereplő kódbitekkel kiegészítve a következő szót kapjuk: 0100110. Ha nincs hiba, a paritásellenőrzők (C_2 , C_1 , C_0) kimenete '000' (nem-létező bitpozíciót azonosít, azaz nincs hiba), így megegyezik a kódolt mintázat paritásbitjeinek értékével, minden egyes paritáscsoportra (küldött és az azonosított C_i -minták bitenkénti XOR kapcsolata).
 - **Error syndrome:** Hiba esetén például, tfh. az input mintázat 0100010 -ra változik, ekkor a vevő oldali paritásellenőrző hibát észlel. Ugyan C_2 . paritásbitcsoport rendben ('0'), DE a C_1 ('0') és C_0 ('1') változott, tehát hiba van:
 - Ekkor 011 = 3 az azonosított minta, ami a 3. oszlopot jelenti (\rightarrow **D0** helyén).
 - Javításként *invertálni kell* a 3. bitpozícióban lévő bitet. 0100010 \Rightarrow 0100110. Ekkor a kódbitek a következőképpen módosulnak a páratlan paritásnak megfelelően: $C_2=0$, $C_1=1$ és $C_0=0$.

Pl. 1/b) Hamming kódú hibajavító áramkör tervezése (Big Endian)

- 3 kódbitünk van, így írásnál 3 paritásbit generáló-, míg olvasásnál 3 paritás-ellenőrző áramkör kell.
- Példa: bemeneti adatbit-mintázatunk 0101 (D_0 - D_3). **BE!**
 - Mivel páratlan paritást alkalmazunk, a megfelelő helyen szereplő kódbitekkel kiegészítve a következő szót kapjuk: 1001101. Ha nincs hiba, a paritásellenőrzők (C_0 , C_1 , C_2) kimenete '000' (nem-létező bitpozíciót azonosít, azaz nincs hiba), így megegyezik a kódolt mintázat paritásbitjeinek értékével, minden egyes paritáscsoportra (küldött és az azonosított C_i -minták bitenkénti XOR kapcsolata).
 - **Error syndrome:** Hiba esetén például, tfh. az input mintázat 1011101 -ra változik, ekkor a vevő oldali paritásellenőrző hibát észlel. Ugyan C_2 . paritásbitcsoport rendben ('1'), DE a C_1 ('1') és C_0 ('0') változott, tehát hiba van:
 - Ekkor (110) azaz 011 = 3 az azonosított minta, ami a 3. oszlopot jelenti (→ **D0** helyén).
 - Javításként *invertálni kell* a 3. bitpozícióban lévő bitet. 1011101 \Rightarrow 1001101. Ekkor a kódbitek következőképpen módosulnak⁵⁸ a páratlan paritásnak megfelelően: $C_0=1$, $C_1=0$ és $C_2=1$.

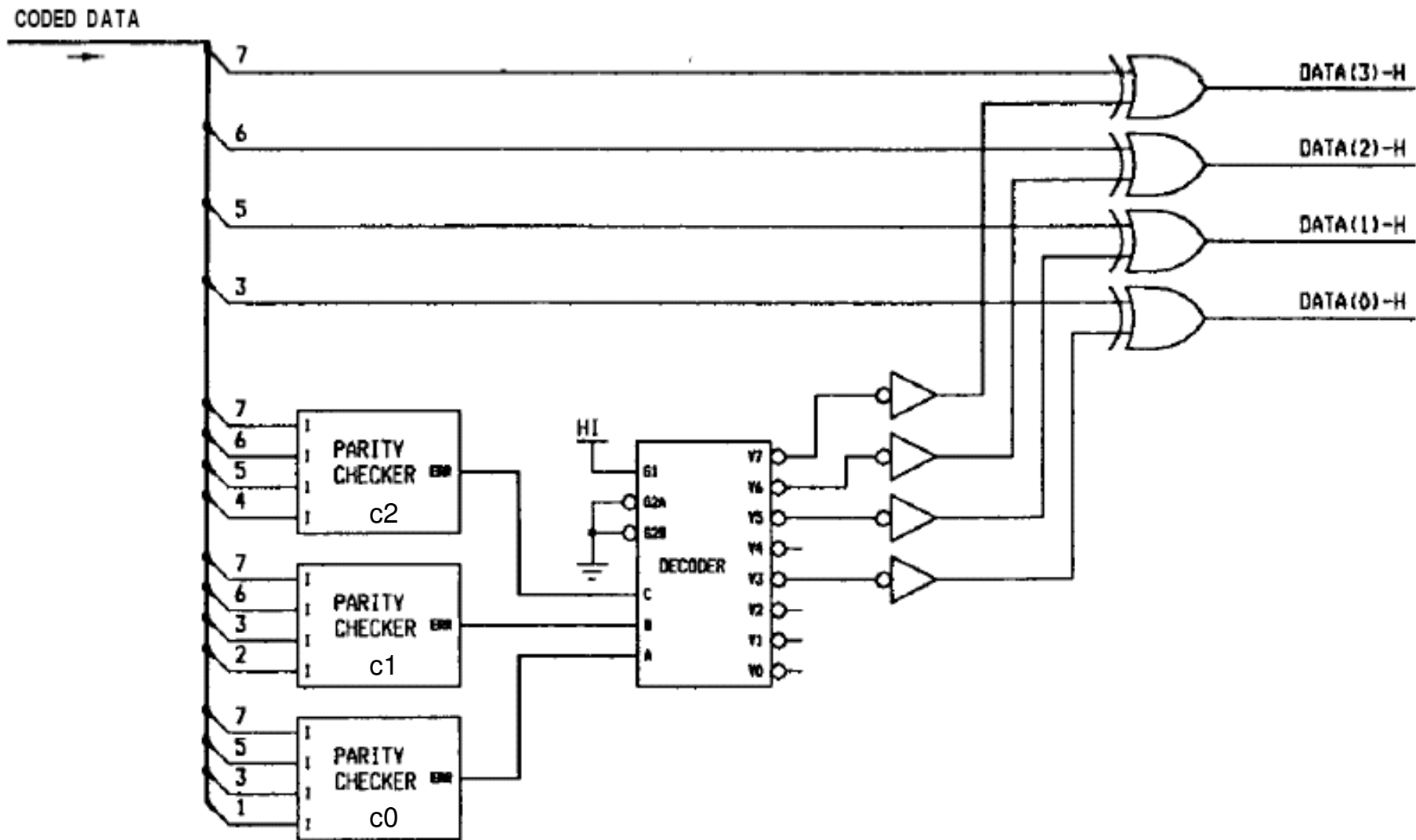
PI 2.) Hamming kódú hibajavító áramkör tervezése (Little Endian)

- 3 kódbitünk van, így írásnál 3 paritásbit generáló-, míg olvasásnál 3 paritás-ellenőrző áramkör kell.
- Változatlan a bemeneti adatbit-mintázatunk 0101 (D_3-D_0).
 - Mivel páratlan paritást alkalmazunk, a megfelelő helyen szereplő kódbitekkel kiegészítve a következő szót kapjuk: 0100110. Ha nincs hiba, a paritásellenőrzők (C2, C1, C0) kimenete '000' (nem-létező bitpozíciót azonosít, azaz nincs hiba), így megegyezik a kódolt mintázat paritásbitjeinek értékével, minden egyes paritáscsoportra (küldött és vett C_i -k bitenkénti XOR kapcsolata).
 - Hiba esetén például, ha az input mintázat 0110110 -ra változik, akkor a paritásellenőrző hibát észlel. Mindhárom paritásbit ellenőrző megváltozott, C2 ('1'), a C1 ('1') és C0 ('1') hibát észlel, tehát:
 - Ekkor 101 = 5 az azonosított minta, ami a 5. oszlopot jelenti (→ **D1** helyén).
 - Javításként *invertálni kell* a 5. bitpozícióban lévő bitet. 0110110 ⇒ 0100110. Ekkor a kódbitek a következőképpen módosulnak a páratlan paritásnak megfelelően: C2=0, C1=1 és C0=0.

PI 3.) Hamming kódú hibajavító áramkör tervezése (Little Endian)

- 3 kódbitünk van, így írásnál 3 paritásbit generáló-, míg olvasásnál 3 paritás-ellenőrző áramkör kell.
- Változatlan a bemeneti adatbit-mintázatunk 0101 (D_3-D_0).
 - Mivel páratlan paritást alkalmazunk, a megfelelő helyen szereplő kódbitekkel kiegészítve a következő szót kapjuk: 0100110. Ha nincs hiba, a paritásellenőrzők (C2, C1, C0) kimenete '000' (nem-létező bitpozíciót azonosít, azaz nincs hiba), így megegyezik a kódolt mintázat paritásbitjeinek értékével, minden egyes paritáscsoportra (küldött és vett C_i -k bitenkénti XOR kapcsolata).
 - Hiba esetén például, tfh. az input mintázat 0100100 -ra változik, akkor a paritásellenőrző hibát észlel. C2. paritásbit ellenőrző változatlan ('0'), és a C0 is ('0'), DE a C1 ('0') hibát észlel, tehát:
 - Ekkor 010 = 2 az azonosított minta önmaga, ami a 2. oszlopot jelenti (→ **C1** paritásbit! helyén).
 - Javításként itt már a dupla hibaelenőrzést (DEB / vagy SECDEC) kell alkalmazni, amely a **paritásbiteket is kódolja**.

7-bites Hamming kódú hibajavító áramkör felépítése



Példa: SEC-DED-dupla paritáshiba ellenőrzés (Little Endian)

DEB: extra bit, a teljes kódszóra vonatkozóan (Ci-eket is kódolja)

Hamming kód (DEB-el) 8 adatbitre: mi a helyes ábrázolása 8 biten a 01011100 adatbit mintázatnak. Szükséges 8 adatbit (D0-D7), 4 kódbit (C0-C3) és egy kettős hibajelző bit (DEB). Páratlan paritást alkalmazunk. (BW-binary weight jelenti az egyes oszlopok bináris súlyát, 1,2, 4 ill 8 biten).

13	12	11	10	9	8	7	6	5	4	3	2	1	Oszlopszám
DEB	D7	D6	D5	D4	C3	D3	D2	D1	C2	D0	C1	C0	
	1	1	1	1	1	0	0	0	0	0	0	0	BW, 8bit
	1	0	0	0	0	1	1	1	1	0	0	0	BW, 4 bit
	0	1	1	0	0	1	1	0	0	1	1	0	BW, 2 bit
	0	1	0	1	0	1	0	1	0	1	0	1	BW, 1 bit

Paritáscsoportok	Bit pozíciók	Bitek jelölései
0	1, 3, 5, 7, 9, 11	C0, D0, D1, D3, D4, D6
1	2, 3, 6, 7, 10, 11	C1, D0, D2, D3, D5, D6
2	4, 5, 6, 7, 12	C2, D1, D2, D3, D7
3	8, 9, 10, 11, 12	C3, D4, D5, D6, D7

13	12	11	10	9	8	7	6	5	4	3	2	1	Oszlopszám
DEB	D7	D6	D5	D4	C3	D3	D2	D1	C2	D0	C1	C0	
-	0	1	0	1	-	1	1	0	-	0	-	-	Adatbitek
1	0	1	0	1	1	1	1	0	1	0	0	0	Hozzáadott

kódbitek. Tehát a helyes ábrázolása 01011100-nek a következő:
1010111101000.