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@mik.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,
    - előjeles:
      - 1-es komplemens,
      - 2-es komplemens 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** („bit/byte-sorrend” a jó fordítása) az a tulajdonság, ami bizonyos adatok - többnyire kisebb adategységek 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 bit/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 (LE)

- A 32-bites „`3A 4B 1C 2D`” értéket a következő módon **4×1byte**-os tárolási egységekben tárolják:
- `0x100 0x101 0x102 0x103...”2D 1C 4B 3A”...`
  Így, a kevésbé jellemző ("legkisebb") byte (az angol Least Significant Byte rövidítéséből *LSB* néven ismert) az első, és ez az `2D`, tehát a *kis vég kerül előre, legkisebb címen van tárolva (`0x100`):*

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

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

„Nagy a végén” - Big-endian (BE)

- Ekkor a 32-bites értéket „`3A 4B 1C 2D`”, a `0x100` címtől kezdve tároljuk a memóriában, **4×1 byte-os** tárolási egységeket feltételezve:
- 1 byte-onként növekvő címekkel rendelkezik

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

- Ebben az esetben, a „legjellemzőbb” byte - erre általában az ismert angolkifejezést „Most Significant Byte" használják a számítástechnikában (rövidítve *MSB*, ami itt a „`3A`”) - a memóriában az *legalacsonyabb címen van tárolva (`0x100`)*, míg a következő "jellemző byte" (`4B`) a következő, egyel nagyobb címen van tárolva, és így tovább.
- *Bit/byte-reversed format!*
- Pl: Tipikusan beágyazott rendszerek processzorai, pl. MCU – mikrovezérlők, programozható FPGA-k (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.

Number of Bits	*Number of Representable Values*	*Machines. Uses*
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!

- **<u>Pl:</u> (Legyen N:=6, LE, unsigned int)**

$101101_2 \Rightarrow$1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 45_{10}

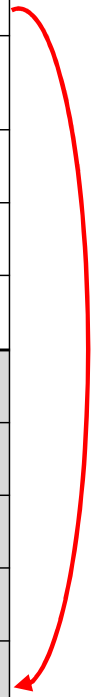
b.) 1’s komplemens rendszer:

- V értékű, N bites rendszer: $\mathbf{2^N–1–V}$.
- *„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 komplemens

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

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

c.) előjeles (2's komplemens) 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)**
- **V(2's)=**$101101_2 \Rightarrow$ –1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = -19_{10}
- Pl

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

* * * * * *

X	1000000	64	vagy	010011	
–Y	– 010011	– 19		101100	1's kompl.
				+ 1	+1
Z	–19 ≡ 101101	45		101101	–19 **V(2's)**

2.2 táblázat:

Table 2.2. 8-Bit Two's Complement Representations.

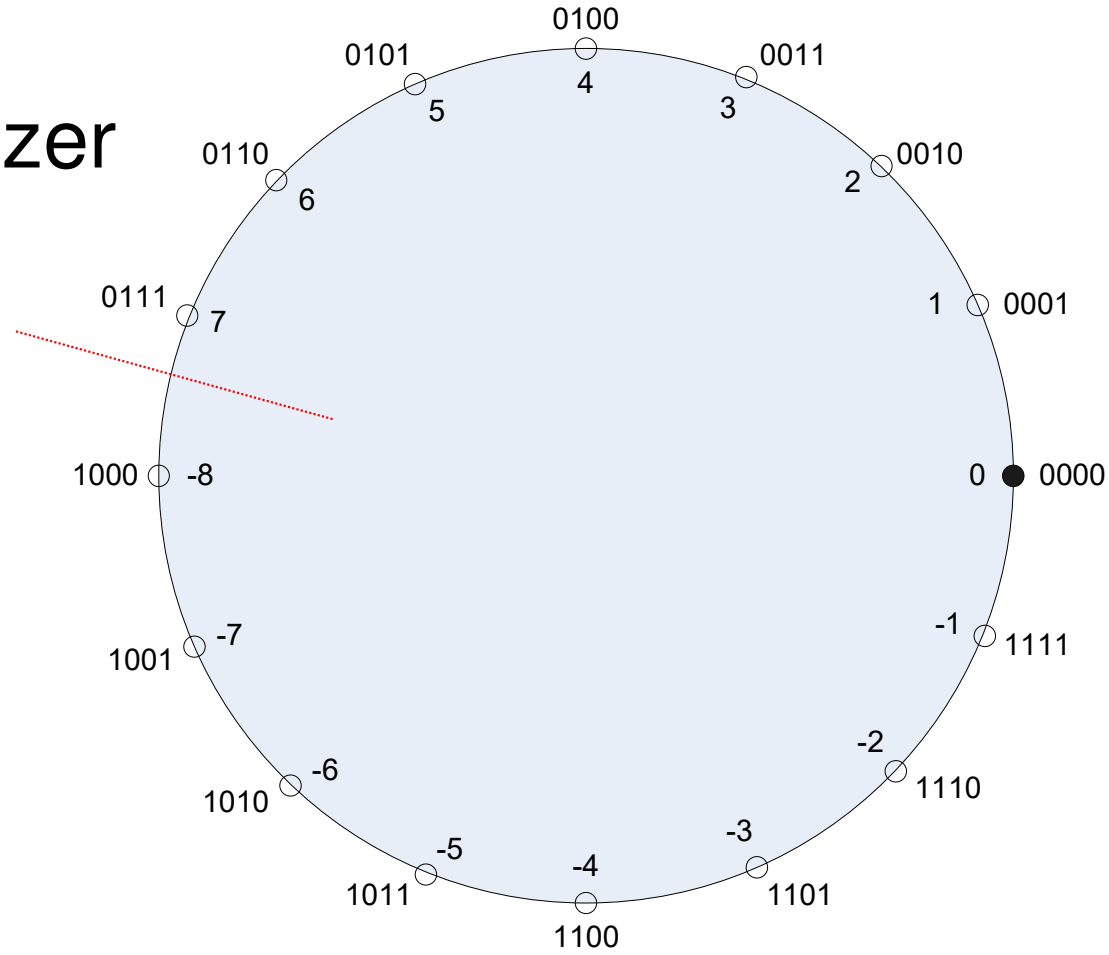
Bit Pattern	*Value*	*Note*
01111111	127	Largest representable value.
01111110	126	
01111101	125	
...	...	
...	...	
00000010	2	Note that leading zero indicates
00000001	1	positive number.
00000000	0	Unique representation of **zero.**
11111111	–1	Minus one is always all ones.
11111110	–2	Note that leading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

Pl: Körkörös számláló (circular nature):

- 4 bites 2's komplemens rendszer
- Overflow:

  0111 -> 1000
- Underflow:

  1000 -> 0111



II.) Fixpontos számrendszer

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

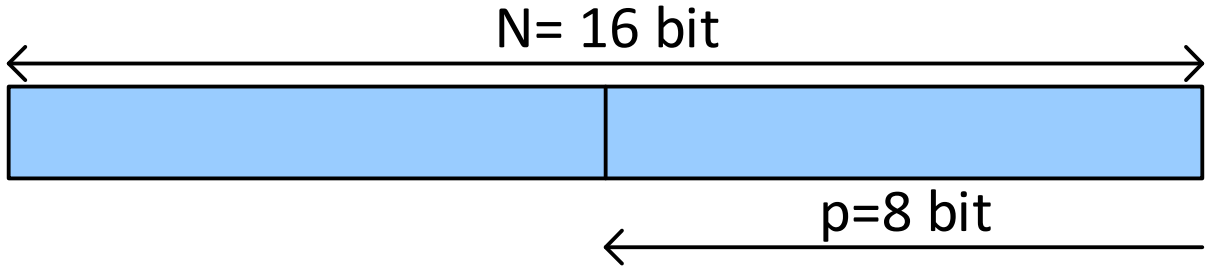
$$V_{\text{FIXED POINT}} = -b_{N-1} \times 2^{N-p-1} + \sum_{i=0}^{N-2} b_i \times 2^{i-p}$$

- p: radix (tizedes) pont helye, tizedes jegyek száma
- *differencia*, $\Delta r = 2^{-p}$ (számrendszer finomsága)
  - *Ha p=0 → Δr=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(smallest/largest absolute)=?, V(smallest/largest negative)=?, V(zero), $\Delta r = ?$ *(ahol lehet, dec. értékben/hatványalakban is megadva!)*

- Megoldás:



- Differencia $\Delta r = 2^{-8} = 0{,}390625 * 10^{-2}$
- V(zero)=**0**0000000.00000000 = 0.0

- V(smallest absolute)= **0**0000000.00000001= $2^{-8} = 0{,}390625 * 10^{-2}$
- V(largest absolute)= **0**1111111.11111111≈ $128 = 128 - \Delta r$
- V(smallest/"most" negative)= **1**0000000.00000000 = $-2^{7} = -128$
- V(largest/"least" negative)= **1**1111111.11111111 = $-2^{-8} = -0{,}390625 * 10^{-2}$

III. Lebegőpontos rendszer:
Exponens - Excess-kód?

- **Lebegőpontos** számok *kitevőjét* (*exponens*-ét) 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.
165	10100101	The representation of 37_{10} in excess 128 code.
+ 128	10000000	This is the excess.
− 23	00010111	The value to be represented.
105	01101001	The representation of -23_{10} in excess **128** code.
165	10100101	This is +37 in excess **128.**
+ 105	+ 01101001	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.**
− 128	− 10000000	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 (RCD) Representations.

Bit Pattern	*Value*
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:

Decimal	*Binary*	
+ 3 3 3	0011 0011 0011	This is the excess.
5 7 3	0101 0111 0011	And the number to be represented.
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.
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
		make output correct.
715	1100 10001 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 × 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.

(-3+1) (+3) (-3)

− 0010 + 0011 − 0011

1010 0100 1000

10 4 8

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

MSD: Most significant digit

LSD: Least significant digit

Lebegőpontos rendszer (FPN):

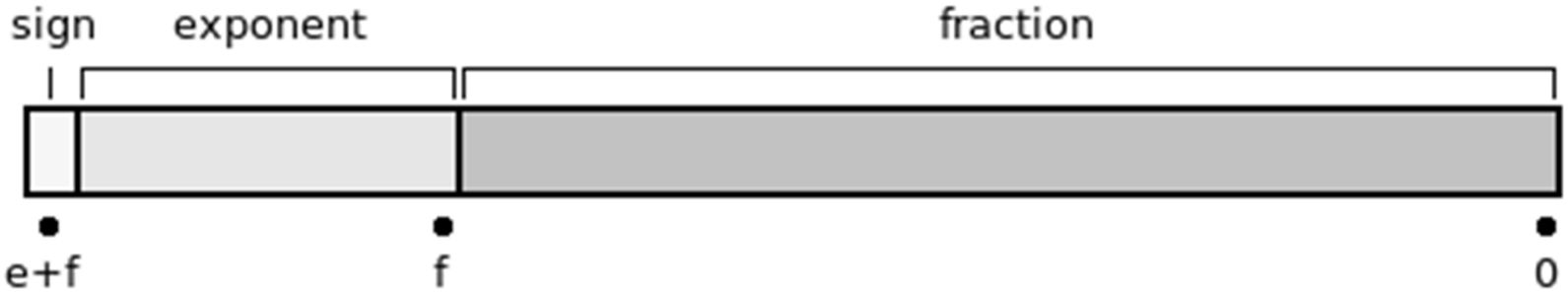
- 7 különböző tényező: *a számrendszer alapja, előjele és nagysága, a mantissza alapja, előjele és hosszúsága, ill. a kitevő alapja*.
- matematikai jelölés:

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

- Fixpontosnál nagyságrendekkel kisebb vagy nagyobb számok ábrázolására is mód van:
  - Pl: Avogadro-szám: 6.022*10^23
  - Pl: proton tömege 1.673*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…1 (két zérus: +0 is)
  - Normalizálatlan (denormal) számok
  - NaN: nem szám (pl. $\frac{\pm 0}{\pm 0} = \mathrm{NaN};$ vagy $\pm 0 \times \pm\infty = \mathrm{NaN}$)
  - $\pm\infty$



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

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: $NLE_{FPN} = V_{E(MAX)} + \left|V_{E(MIN)}\right| + 1_{ZERO}$
- Ábrázolható értékek száma: $NRV_{FPN} = NLM_{FPN} \times NLE_{FPN}$

- Normalizálás: mantissza értékét általában [0…~1] közé
- Pl: $32\ 768_{10} = 0.32768*10^5 = 3.2768*10^4 = 32.768*10^3 = 327.68*10^2 = 3267.8\ *10^1$ (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:

$$V_{M(MAX)} = 0.999 = 1.000 - 10^{-3}$$
$$V_{M(MIN)} = 0.100$$
$$V_{E(MAX)} = (r_e \text{ ^ } e - 1) = 99$$
$$V_{E(MIN)} = -(r_e \text{ ^ } e - 1) = -99$$

Egyszerűsítésként, itt még nincs használatban az Excess kódolás!

$$V_{FPN(MAX)} = 0.999 \times 10^{99}$$
$$V_{FPN(MIN)} = 0.100 \times 10^{-99}$$

$$NLM_{FPN} = 9 \times 10 \times 10 = 900 = 9 \times 10^2$$
$$NLE_{FPN} = 99 + |-99| + 1_{ZERO} = 199$$
$$NRV_{FPN} = 2 \times 900 \times 199 = 358{,}200$$

Normalizált lebegőpontos rdsz.

Table 2.4. 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	$\frac{15}{16}$	$\frac{15}{16}$	$1\frac{7}{8}$	$3\frac{3}{4}$	$7\frac{1}{2}$

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

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

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

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

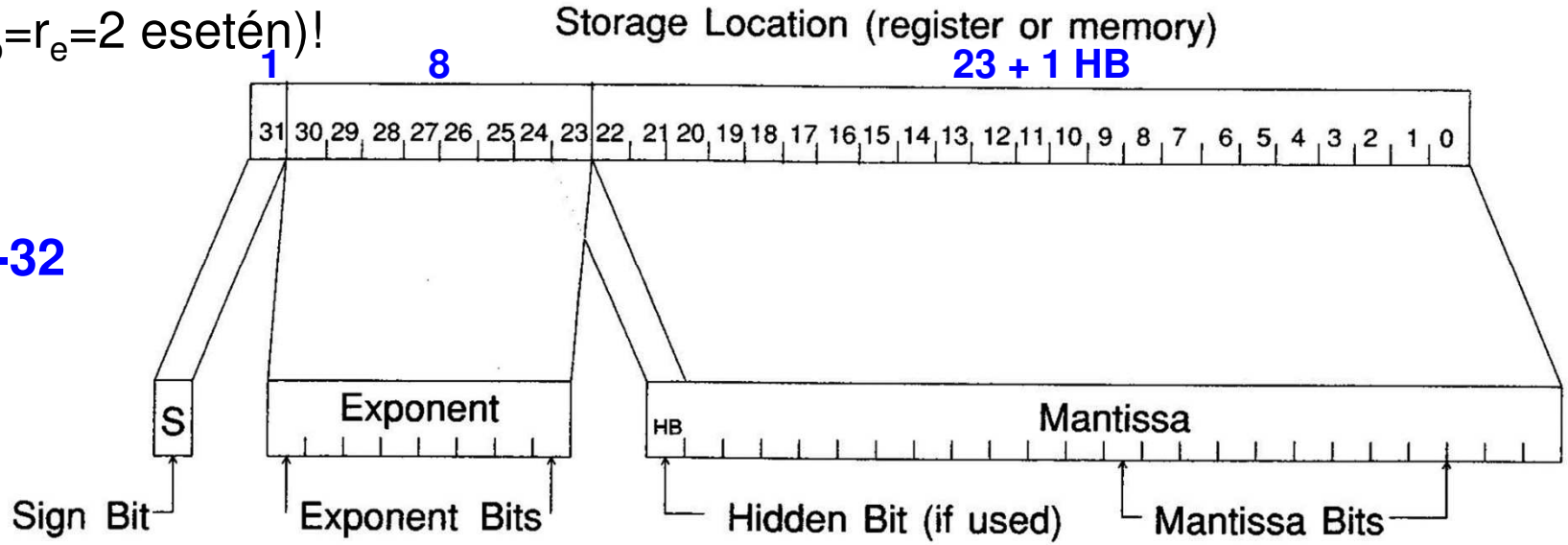
Number of fractions = $1 \times 2 \times 2 \times 2 = 8$

Number of values = 8×4 $= 32$

Normalizált lebegőpontos ábrázolás:
Rejtett (hidden bit) technika

- „Vezető” ‘1’-sek számát a legtöbb rendszer tervezésekor konstans értékként definiálják (HB=1), DE nem tárolják!
  - Ilyen a *mantissza 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 $1.000_2 \times 2^{-1} = 0.1000_2 \times 2^0 = ½$ -nek felel meg (m=4, p=3, e=2, $r_b=r_e=2$ esetén)!

IEEE-32



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), korábbi DEC (VAX, PDP gépei), korábbi IBM rendszerek gépei
  - Hidden bit: ma az IEEE-32 számrendszer (754-es formátum)

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).

$$V_{M(MIN)} = 0.\underbrace{1000\ldots_2}_{24db} = 1/2$$

$$V_{M(MAX)} = 0.1111\ldots_2 = 0.999999940395 = 1.0 - 2^{-24}$$

$$V_{E(MIN)} = -(r_e^{\,e-1} - 1) = -(2^{8-1} - 1) = -127$$

$$V_{E(MAX)} = r_e^{\,e-1} - 1 = 2^{8-1} - 1 = 127$$

} Excess-128

$$V_{FPN(MIN)} = 0.1000\ldots_2 \times 2^{-127} = 2.9387 \times 10^{-39}$$

$$V_{FPN(MAX)} = 0.1111\ldots_2 \times 2^{+127} = 1.7014 \times 10^{38}$$

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

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

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

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).

$$V_{M(MIN)} = 1.\underbrace{000\ldots_2}_{23db} = 1$$

$$V_{M(MAX)} = 1.111\ldots_2 = 1.99999988 = 2.0 - 2^{-23}$$

$$V_{E(MIN)} = -126$$

$$V_{E(MAX)} = 127$$

// Zérus pontosabb ábrázolása miatt -127+1

Excess-127 = Excess(128-1)

$$V_{FPN(MIN)} = 1.000\ldots_2 \times 2^{-126} = 1.1755 \times 10^{-38}$$ // 4 x DEC!

$$V_{FPN(MAX)} = 1.111\ldots_2 \times 2^{+127} = 3.4028 \times 10^{38}$$ // 2 x 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).

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

$$V_{M(MAX)} = 0.\mathrm{FFFFFF}_{16} = 0.999999940395 = 1.0 - 16^{-6}$$

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

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

} Excess-64

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

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

Bővebb tartomány mint a DEC!

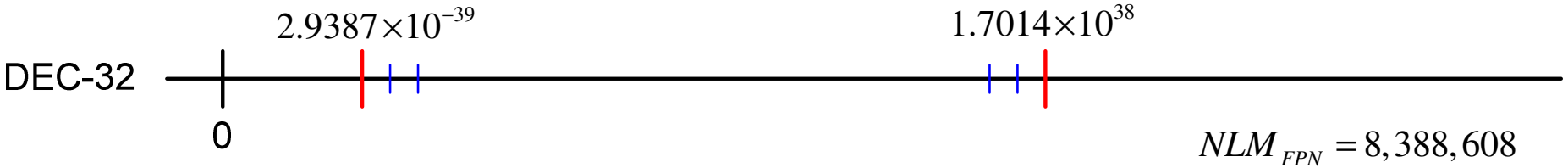
$$NLM_{FPN} = 15 \times 16^5 = 15,728,640$$

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

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

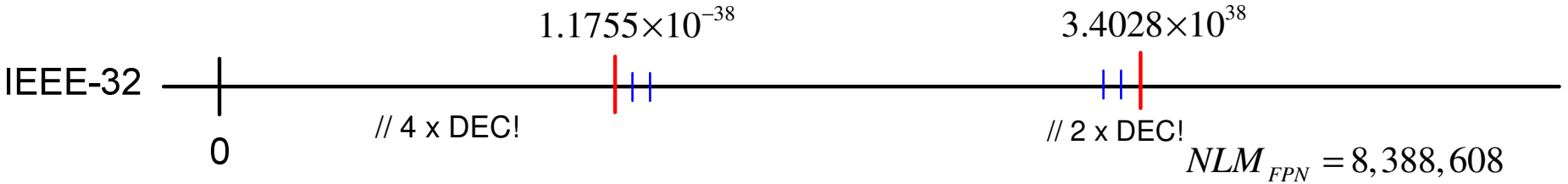
7%-al kevesebb mint a DEC!!

Lebegőpontos számrendszerek összehasonlítása (ha FPN előjele pozitív):



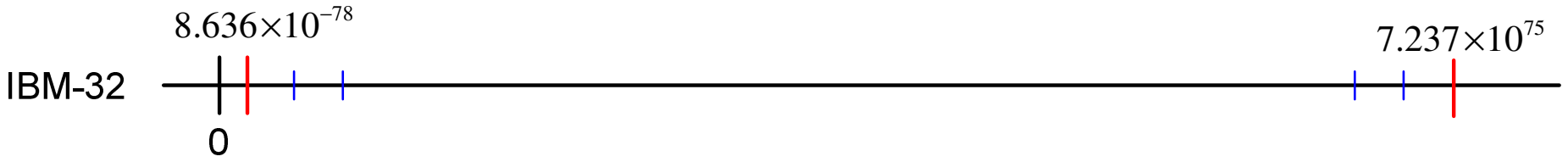
$NLM_{FPN} = 8,388,608$

$NRV_{FPN} = 2.139\times10^{9}$



$NLM_{FPN} = 8,388,608$

$NRV_{FPN} = 2.139\times10^{9}$



$NLM_{FPN} = 15,728,640$

$NRV_{FPN} = 1.9975\times10^{9}$

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

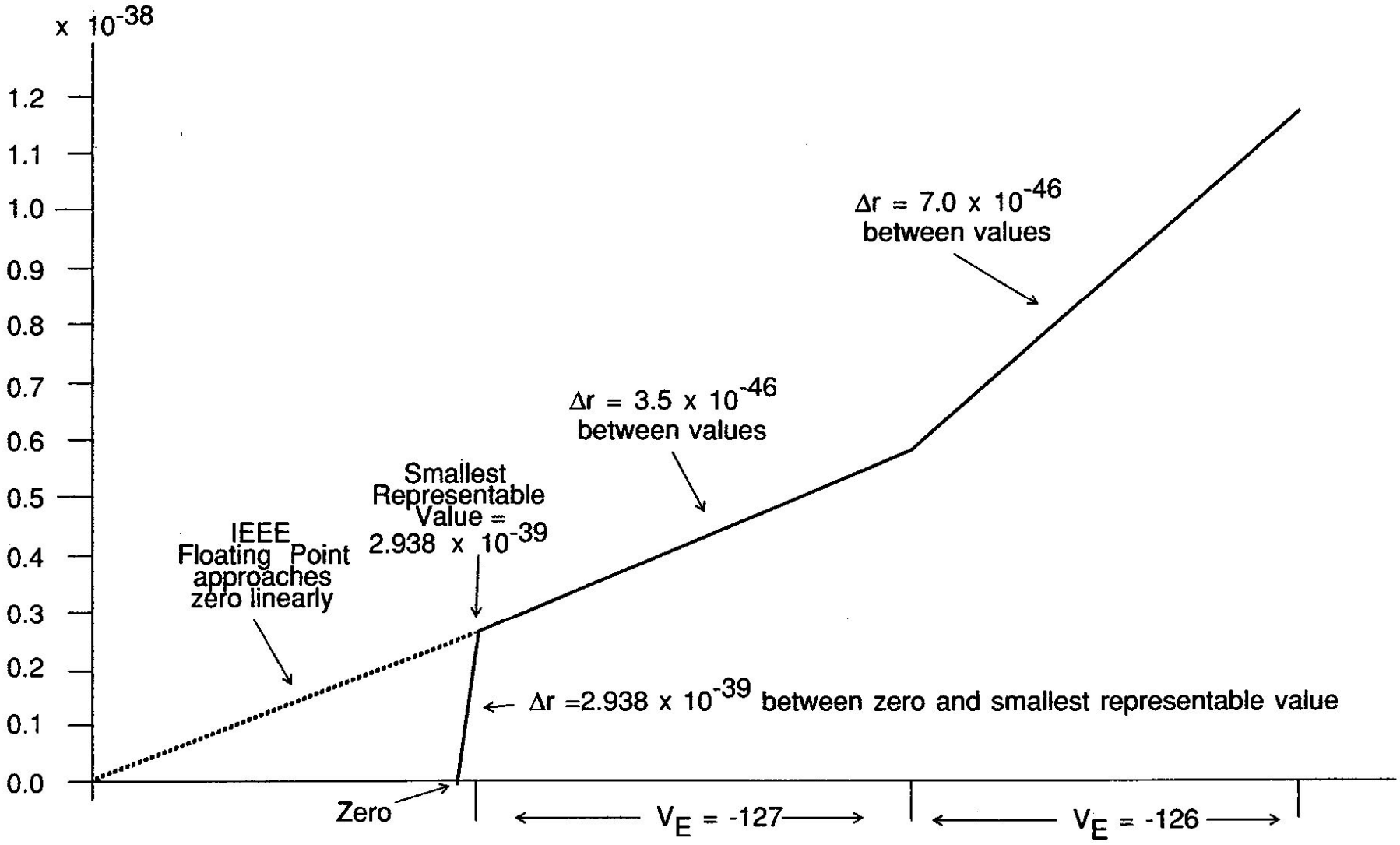


Figure 2.3. Values of the DEC Normalized Floating Point System Near Zero.

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

- V_E = [-126, 127] → [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:

```
(+∞) + (+7) = (+∞)
(+∞) × (−2) = (−∞)
(+∞) × 0 = NaN
0 / 0    = NaN
Sqrt(−1) = NaN
```

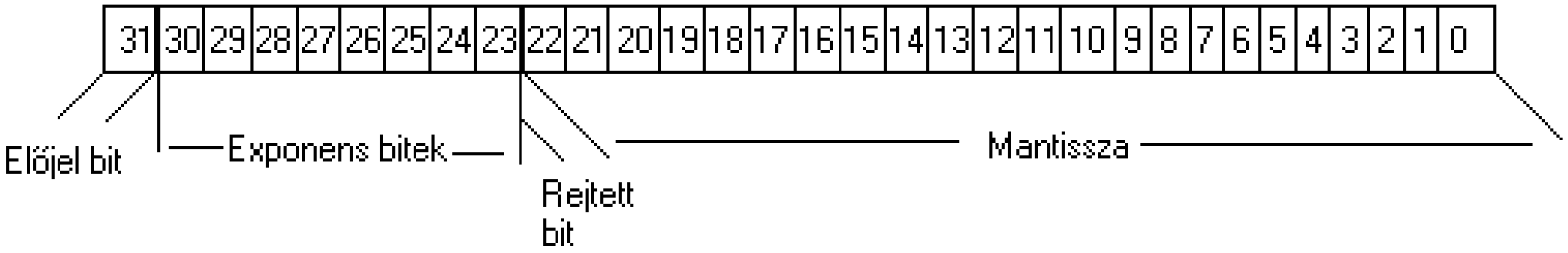
V_E [255]	S (előjel)	Ábrázolás jelentése
≠ 0	X	Nem egy szám (NaN)
0	0	+∞
0	1	-∞

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.**

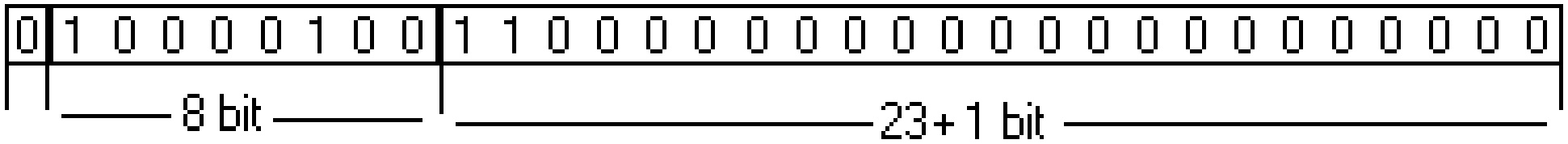


Példa (folyt) **DEC-32**

- DEC-32 lebegőpontos számrendszerben: $r_e = r_b = 2$, $p = m = 24$ (vagyis a mantissza hossza p=24, a rejtett bit helyén is tárolunk, a mantisszák tartománya 1/2-től 1-ig terjedhet, HB=0), e=8 az exponens bitek száma Excess-128 kódban tárolva.

- $12_{10} = 1100_2 = 0.1100 * 2^4 \Rightarrow$ a kitevő $4_{10} = 100_2$ Excess-128 kódja: 10000000+100= 10000100.

- Tehát a fenti formának megfelelően:

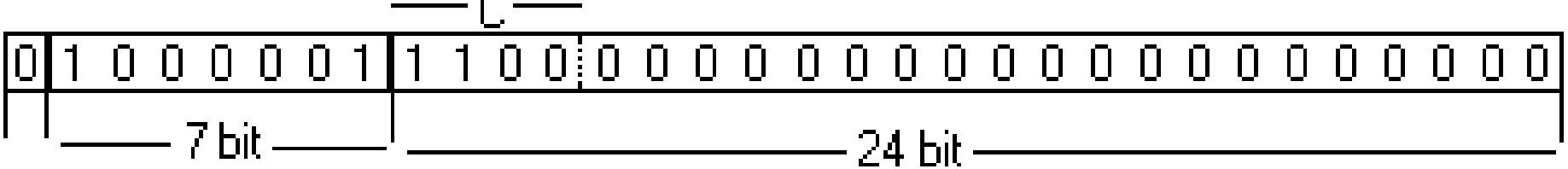


Példa (folyt) **IBM-32**

- IBM-32 lebegőpontos rendszerben: r_e=2, r_b=16 (a rdsz. alapja hexadecimális), p=m=6 a rejtett bittel együtt, HB=0 (vagyis a mantissza hossza ugyan 6 számjegynyi, de egy hexadecimális érték tárolásához 4 bit szükséges, tehát 4*6=24), e=7 (az exponens bitek száma egyel kevesebb!)
  Excess-64 kódban tárolva.

- $12_{10} = C_{16} = 0.C * 16^1 \Rightarrow$ a kitevő $1_{10} = 1_2$ Excess-64 kódja: 1000000+1= 1000001.

- Tehát a fenti formának megfelelően: ($C_{16} = 1100_2$)

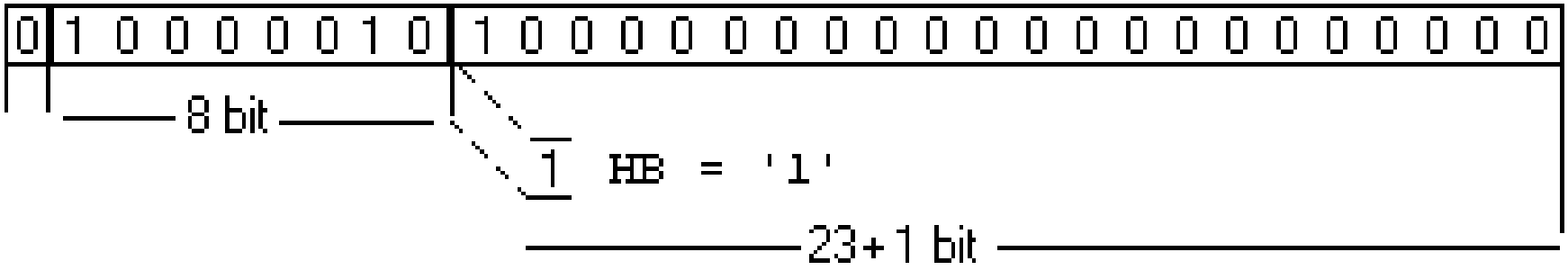


Példa (folyt) **IEEE-32**

- 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) kocsi-vissza, 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

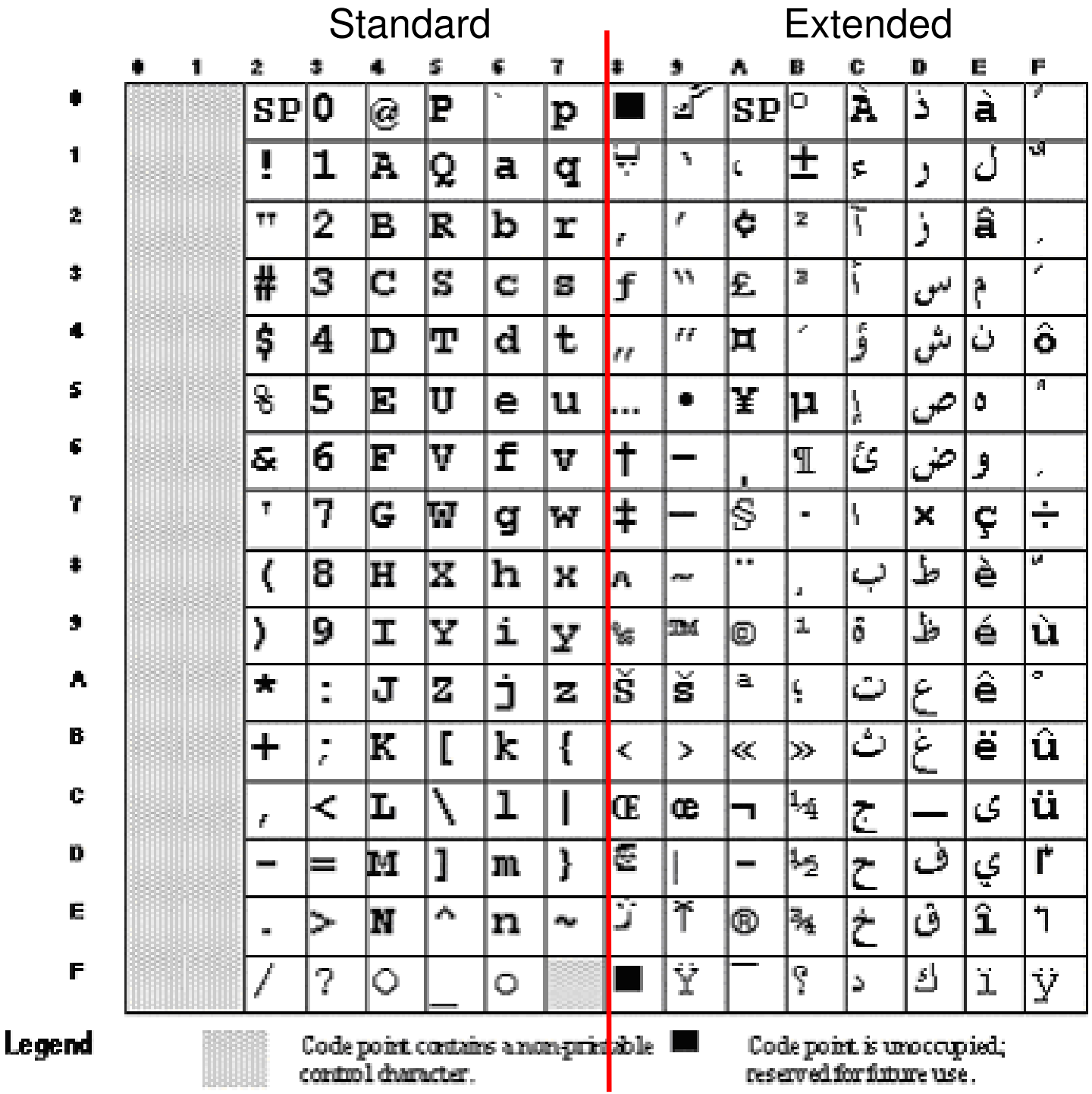
- BCD (Binary Coded Decimal): 6-biten
  - nagybetűk, számok, és speciális karakterek
- EBCDIC (Extended Binary Coded Decimal Interchange Code): 8-biten (A. Függelék)
  - + kisbetűs karaktereket és kiegészítő-információkat
  - 256 értékből nincs mindegyik kihasználva
  - Továbbá I és R betűknél szakadás 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 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO810000	Ø LO820000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(NSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~	£	A 000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000			000	K LK020000	S LS000000	2 ND020000
-3	ã LA170000	ë LE170000	Ã LA180000	Ë LE180000	c LC010000	l LL010000			000	L LL020000	T LT000000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000			000	M LM020000	U LU000000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	' LV010000	≥ SM340000	⌐ LE020000	N LN020000	V LV000000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	Œ LO830000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	œ LO810000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	Ÿ LY180000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	Ý LY120000	! SP020000	Š LS220000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	(SHY) SP320000	¹ ND011000	² ND021000	³ ND031000
-B	. SP110000	$ SC030000	, SP060000	# SM010000	» SP180000	º SM220000	¿ SP160000	š LS210000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD830000	æ LA510000	Ð LD840000	¯ SD310000	õ LO170000	ü LU170000	Õ LO180000	Ü LU180000
-D	(SP080000	) SP070000	_ SP090000	' SP050000	ý LY110000	ž LZ210000	[SM080000	] SM090000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT810000	Æ LA520000	Þ LT840000	Ž LZ220000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	| SM130000	^ SD150000	? SP150000	" SP040000	± SA020000	€ SC200000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

A
LA020000

Extended ASCII (1 byte)



Unicode

Nyelvkészletek: Alap, - Latin 1/2, görög, cirill, héber, arab stb.

Általános írásjelek, matematikai, pénzügyi, mértani szimbólumok stb.

*részlet

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: $2^{\wedge N}$ -ről $2^{\wedge N+1}$ –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
páros paritás	1 1 0 1	1
páratlan paritás	1 1 0 1	0

- Páros paritás: az ‘1’-esek száma páros.
  - A kódszóban lévő ‘1’-esek számát ‘1’ vagy ‘0’ hozzáadásával párossá 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 páratlanná 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!
- Példa:

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

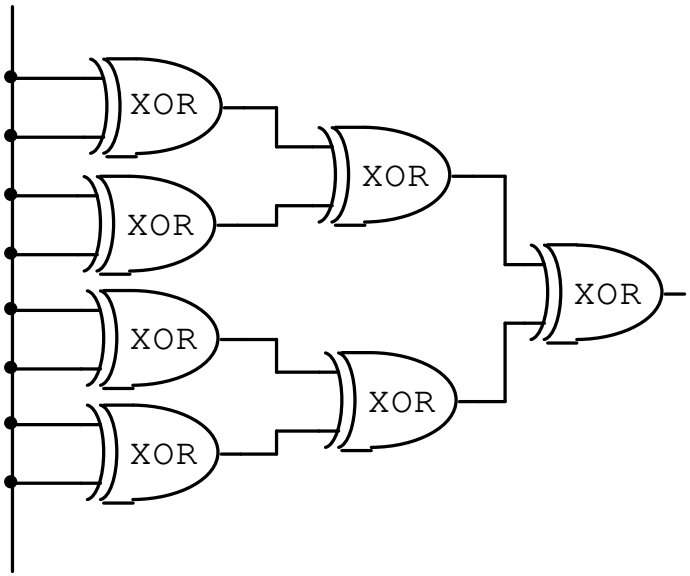
Kódszó Paritásbit(P)

0 0 0 1 ?→ 0 ⊕0 ⊕0 ⊕1 = 1
0 1 1 0 ?→ 0 ⊕1 ⊕1 ⊕0 = 0
1 1 1 0 ?→ 1 ⊕1 ⊕1 ⊕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)

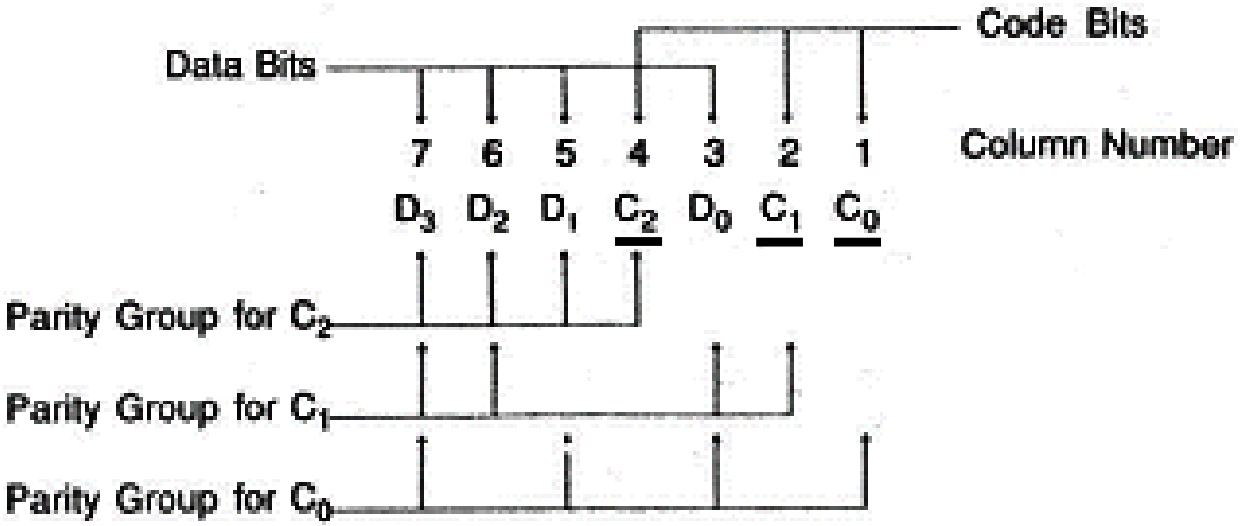


Hamming kód

- 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 (D0,D1,D2,D3),* ***3*** *kódbittel (C0,C1,C2)* 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	C0, D0, D1, D3
1	2, 3, 6, 7	C1, D0, D2, D3
2	4, 5, 6, 7	C2, D1, D2, D3

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: 010**0**1**10**. 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 az azonosított Ci-minták bitenkénti XOR kapcsolata).
  - Error syndrome: Hiba esetén például, tfh. az input mintázat 010**0**0**10** -ra változik, ekkor a vevő oldali paritásellenőrző hibát észlel. Ugyan C2. paritásbitcsoport rendben ('0'), DE a C1 ('0') és C0 ('1') változott, tehát hiba van:
    - Ekkor 011 = 3 az azonosított minta, ami a 3. oszlopot jelenti (→ **D0** helyén).
    - Javításként *invertálni kell* a 3. bitpozícióban lévő bitet. 010**0**0**10** ⇒ 010**0**1**10**. Ekkor a kódbitek a következőképpen módosulnak a páratlan paritásnak megfelelően: C2=0, C1=1 és C0=0.

Pl. 1/a) folyt. Hamming kódú kódszó (Little Endian)

- 4 adatbithez (D_3-D_0)=0101 → 3 paritásbit (C_2-C_0)
  - azaz 7-bites Hamming kódú hibajavító kódszó

7	6	5	4	3	2	1	poz
D3	D2	D1	C2	D0	C1	C0	
0	1	0	?	1	?	?	
0	1	0	0	1	1	0	
|	.	|	.	|	.	|	
|	|	.	.	|	|		
|	|	|	|				

Error syndrome

	C2	C1	C0
Adó	0	1	0
Vevő	0	1	0
XOR	0	0	0

Azaz 000 = 0. pozíció
(nem létezik,
tehát nincsen hiba)

Pl. 1/a) folyt. Hamming kódú kódszó (Little Endian)

- Tfh. van hiba, a D0 megváltozik ’1’ → ’0’

7	6	5	4	3	2	1
D3	D2	D1	C2	D0	C1	C0
0	1	0	?	0	?	?
0	1	0	0	0	0	1
|	.	|	.	|	.	|
|	|	.	.	|	|	
|	|	|	|			

*

poz

Error syndrome

	C2	C1	C0
Adó	0	1	0
Vevő	0	0	1
XOR	0	1	1

Azaz 011 = 3. pozíció (tehát D0 a hibás!)

Javítás = hibás D0 invertálása ’0’ → ’1’

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: **10**0**1**101. Ha nincs hiba, a paritásellenőrzők (C0, C1, C2) kimenete '000' (nem-létező bitpozíciót azonosít, azaz nincs hiba), így megegyezik a kódolt mintázat paritásbitjeinek értékével, minden egyes paritáscsoportra (küldött és az azonosított Ci-minták bitenkénti XOR kapcsolata).
  - Error syndrome: Hiba esetén például, tfh. az input mintázat 1011101 -ra változik, ekkor a vevő oldali paritásellenőrző hibát észlel. Ugyan C2. paritásbitcsoport rendben ('1'), DE a C1 ('1') és C0 ('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 ⇒ 1001101. Ekkor a kódbitek következőképpen módosulnak a páratlan paritásnak megfelelően: C0=1, C1=0 és C2=1.

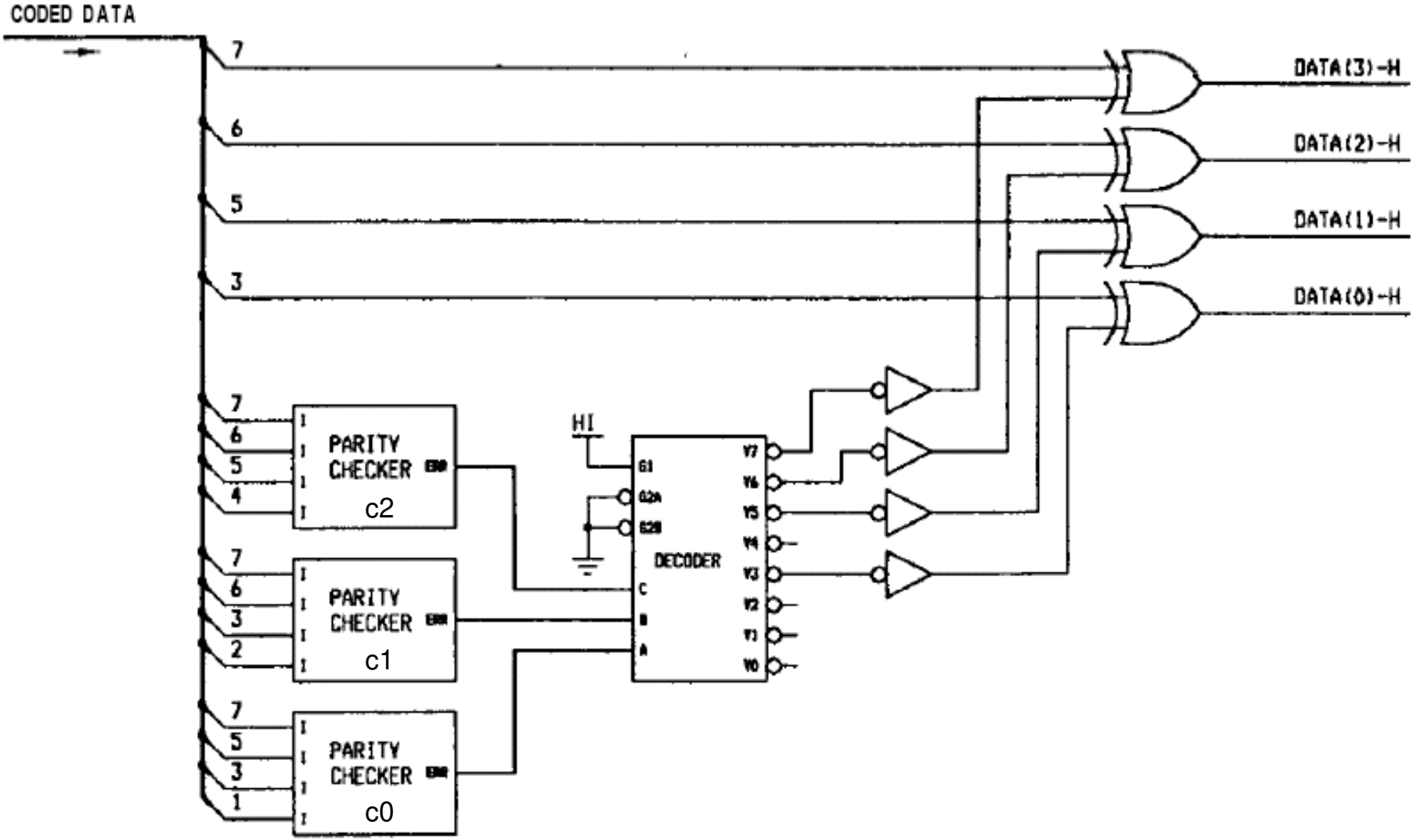
Pl 2.) Hamming kódú hibajavító áramkör tervezése (Little Endian)

- 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: 010**0**1**10**. 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 Ci-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.

Pl 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: 010**0**1**10**. 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 Ci-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 hibaellenőrzést (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-ket 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.