Pannon Egyetem
Villamosmérnöki és Információs Rendszerek Tanszék

Digitális Rendszerek és Számítógép Architektúrák (BSc államvizsga tétel)

2. tétel: Az információ reprezentációi és az ALU felépítése

Összeállította: Dr. Vörösházi Zsolt
voroshazi@vision.vein.hu

Jegyzetek, segédanyagok:

- Tétel: Informatika / Programtervező Informatikus / Gazdaságinformatikus BSc alapszakos hallgatóknak (2012. május)
- Könyvfejezetek:
  - http://www.virt.uni-pannon.hu
    → Oktatás → Tantárgyak → Digitális Rendszerek és Sz.gép Arc. / Számítógép Architektúrák
  - (chapter02.pdf)

Információ ábrázolás:

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

A) Numerikus 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.

„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 angolkifejezést "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-reversed format!
- Pl: mikrovezérlők, beágyazott processzorok FPGA-kon (MicroBlaze, PowerPC) stb.

„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 nem kötik ki külön, ezt feltételezzük! (pl. Intel, AMD, illetve ARM 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>**

101101 $\Rightarrow$ 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 45

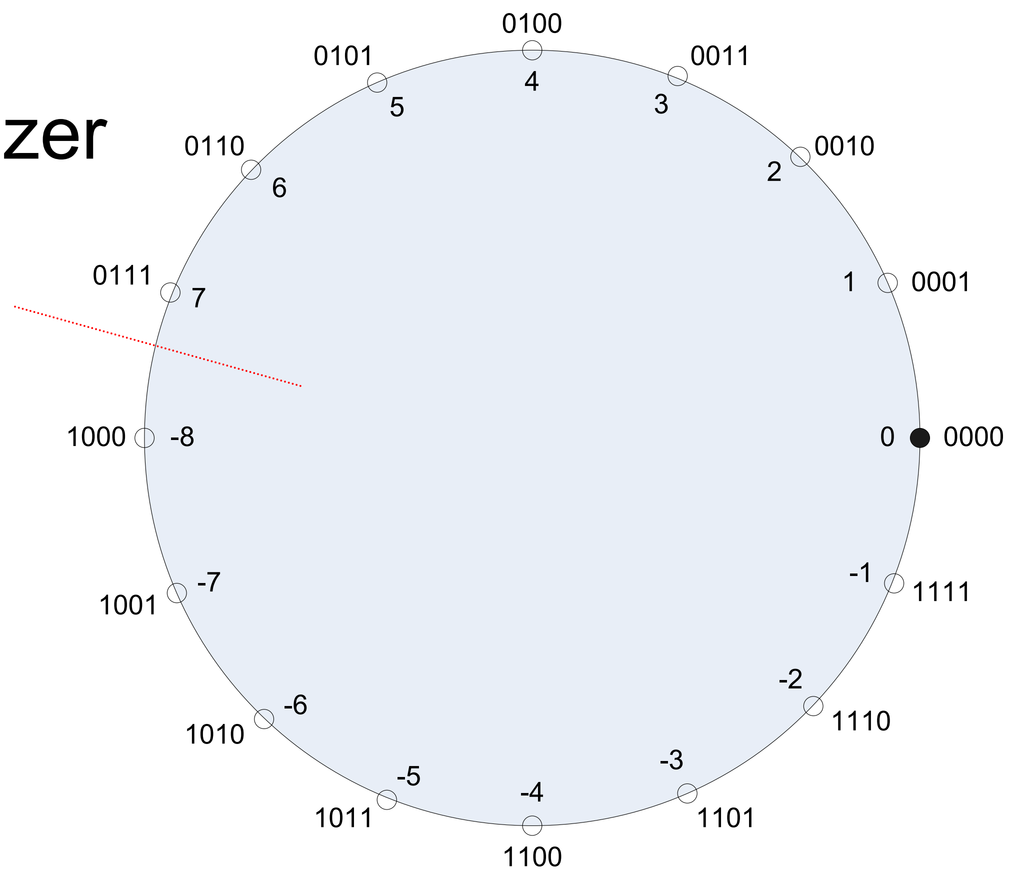
b.) előjeles (kettes komplemens) rendszer:

- 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. 101101 ⇒ –1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = –19
- Pl.

$$\begin{array}{r} 100000000 \\ -\ 01001100 \\ \hline 10110100 \end{array} \qquad \begin{array}{r} 256 \\ -\ 76 \\ \hline 180 \end{array} \qquad \Leftrightarrow \qquad \begin{array}{rl} 01001100 & \\ 10110011 & \text{1's kompl.} \\ +\qquad 1 & \text{+1} \\ \hline 10110100 & -76 \end{array}$$

PI: Körkörös számláló (circular nature):

- 4 bites 2’s komplemens rendszer
- Overflow:
  0111 -> 1000
- Underflow:
  1000 -> 0111



c.) 1-es 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,
- <u>Nem</u> helyiértékes rendszer,
- kétféleképpen is lehet ábrázolni a zérust!! (ellenőrzés szükséges)
- end-around carry: amelyben a részeredményhez kell hozzáadni a végrehajtás eredményét

II.) Fixpontos számrendszer

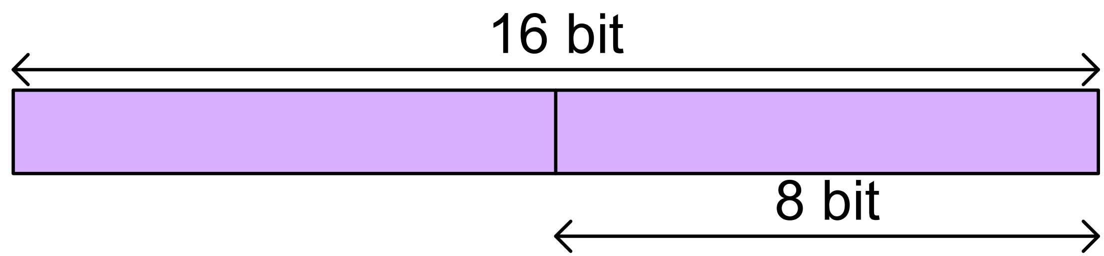
- Műveletek:
  - +, - : ugyanaz, mint az egész szám rdsz. esetén
  - *, / : meg kell bizonyosodni arról, hogy a tizedespont helyén maradt-e

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

- 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: pl. jelfeldolgozás (DSP – Texas Instruments)

Példa: fixpontos rendszer

- Kérdés: Legyen egy 16 bites 2’s comp. fixpontos rdsz. ahol p=8. V(smallest)=?, V(largest)=?, Δr *= ? (decimális értékben megadva)*
- Megoldás:



- V(smallest absolute)=00000000.00000001=2^-8=$\underline{0,390625*10^{-2}}$
- V(largest absolute)=01111111.11111111=$\underline{\sim 128}$
- V(largest negative)=10000000.00000000=−2^7=$\underline{-128}$
- Differencia Δr =2^-8=$\underline{0,390625*10^{-2}}$
- !!DE V(zero)=00000000.00000000

Excess-kód

- Lebegőpontos számok *kitevőit* (*exponens*-eit) tárolják / kódolják ezzel a módszerrel. Elkerülni: a kitevő NE legyen negatív, ezért eltolás történik.
  - S: a reprezentálni kívánt érték (eredmény), amit tárolnunk
  - V: a szám valódi értéke, E: az excess
  - 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

III.) Lebegőpontos rendszer:

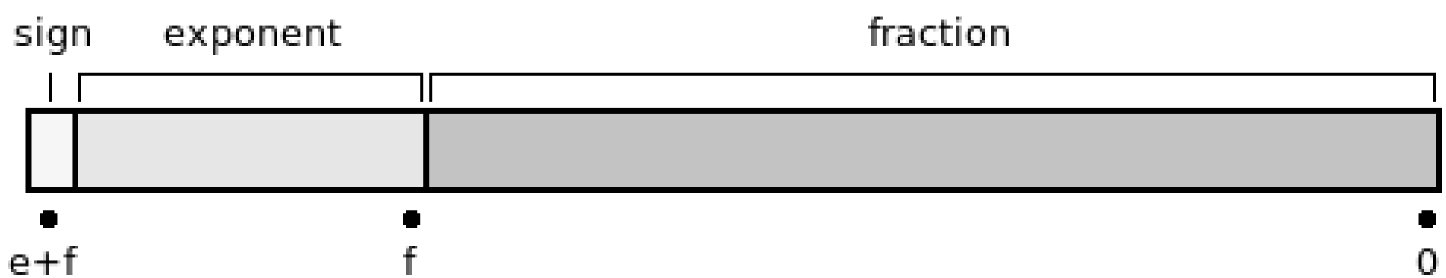
- 7 különböző tényező: *a számrendszer alapja, előjele és nagysága, a mantissza alapja, előjele és hosszúsága, ill. a kitevő alapja*.
- matematikai jelölés:

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

- 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

IEEE 754-1985

- Szabvány a bináris lebegőpontos számok tárolására, amely tartalmazza még:
  - negatív zérust: –0 = 111…1
  - Normalizálatlan számok
  - NaN: nem szám (Not a Number)
  - – /+ ∞



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

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: $V_{M_{max}} = 0.d_m d_m d_m \ldots = (1 - r_b^{-m})$
  - 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: $\boxed{V_{FPN} = (-1)^{SIGN} V_M \times r_b^{V_E}}$

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
- PI: $32\ 768_{10} = 0.32768*10^5 = 3.2768*10^4 = 32.768*10^3 = 327.68*10^2 = 3267.8 *10^1$ (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:

$$V_{M(MAX)} = 0.999 = 1.000 - 10^{-3}$$
$$V_{M(MIN)} = 0.100$$
$$\left.\begin{aligned} V_{E(MAX)} &= (r_e \wedge e - 1) = 99 \\ V_{E(MIN)} &= -(r_e \wedge e - 1) = -99 \end{aligned}\right\}$$

Itt még nincs megadva, és használva az Excess kódolás!

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

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

Rejtett bit

- Probléma: a zérus (közelítő) ábrázolása
- Hogy tárolható mégis a zérus? → Excess 2^{e-1} kódolással, de csak, **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)

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).

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

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

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

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

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

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

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

$$NRV_{FPN} = 2^{23} \times (2^8 - 1) = 2.139 \times 10^9$$

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).

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

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

$$\left.\begin{aligned} V_{E(MIN)} &= -(r_e^{\,e-1} - 1) = -(2^{7-1} - 1) = -63 \\ V_{E(MAX)} &= r_e^{\,e-1} - 1 = 2^{7-1} - 1 = 63 \end{aligned}\right\}$$

Excess-64

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

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

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!!

Példa: **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).

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

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

$$\left.\begin{aligned} V_{E(MIN)} &= -126 \\ V_{E(MAX)} &= 127 \end{aligned}\right\}$$

// Zérus pontosabb ábrázolása miatt +1

Excess-127

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

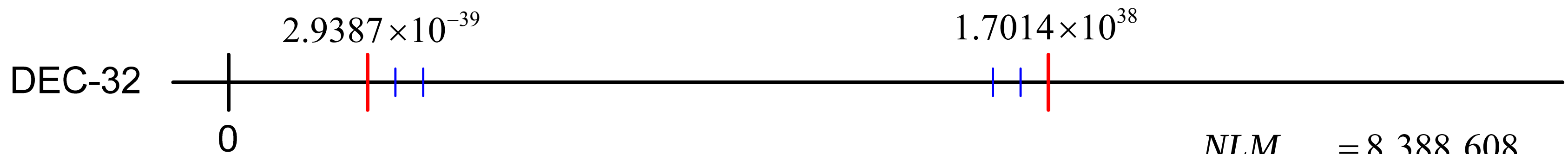
$$V_{FPN(MAX)} = 1.111\ldots_2 \times 2^{+127} = 3.4028 \times 10^{38}$$ // 2 x DEC!

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

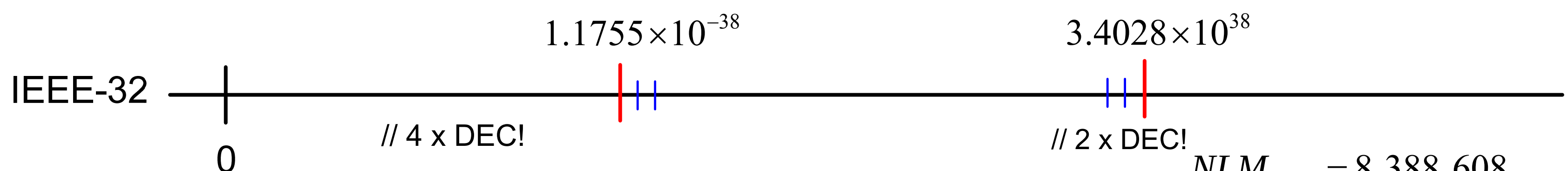
$$NLE_{FPN} = 127 + |-126| + 1_{ZERO} = 254$$

$$NRV_{FPN} = 2^{23} \times (2^8 - 1) = 2.131 \times 10^9$$

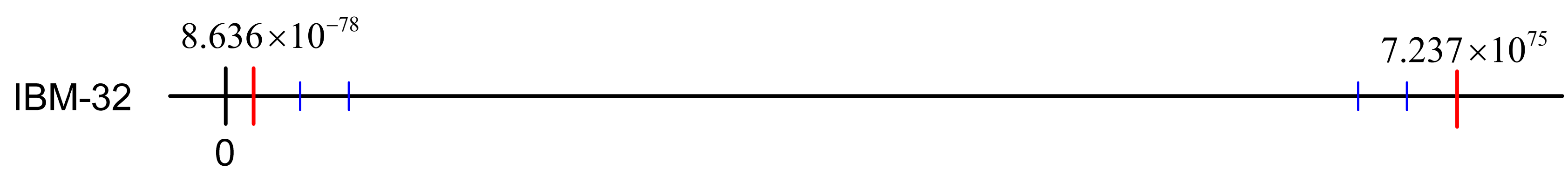
Lebegőpontos számrendszerek összehasonlítása (ha FPN előjele pozitív):



$$NLM_{FPN} = 8,388,608$$
$$NRV_{FPN} = 2.139\times10^{9}$$



$$NLM_{FPN} = 8,388,608$$
$$NRV_{FPN} = 2.139\times10^{9}$$



$$NLM_{FPN} = 15,728,640$$
$$NRV_{FPN} = 1.9975\times10^{9}$$

7%-al kevesebb mint a DEC!!

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

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

V_E	Előjel	Ábrázolás jelentése
$\neq 0$	1,0	Nem egy szám (NaN)
0	0	$+\infty$
0	1	$-\infty$

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)

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
- Redundancia: detektálás, hibajavítás

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. 4 adatbit esetén háromszor!
- Példa:

Kódszó Paritásbit

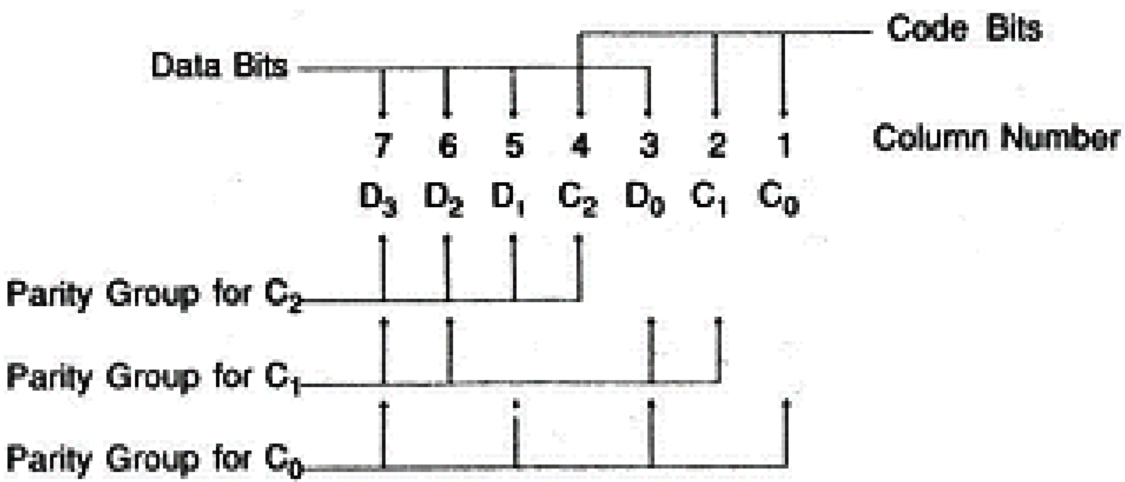
$$\left.\begin{array}{l} 0\,0\,0\,1 \rightarrow 0 \oplus 0 \oplus 0 \oplus 1 = 1 \\ 0\,1\,1\,0 \rightarrow 0 \oplus 1 \oplus 1 \oplus 0 = 0 \\ 1\,1\,1\,0 \rightarrow 1 \oplus 1 \oplus 1 \oplus 0 = 1 \end{array}\right\}$$ Páros paritás!

Hamming kód

- Több redundáns bittel nemcsak a hiba meglétét, és helyét tudjuk detektálni, hanem a hibás bitet javítani is tudjuk
- **Hamming kód**: egy biten tároljuk a bitmintázatok azonos helyiértékű bitjeinek különbségét, tehát egybites hibát lehet vele *javítani*.

Hamming kódú kódszó konstruálása (pl. 7 –bites kódszóra)

- **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)

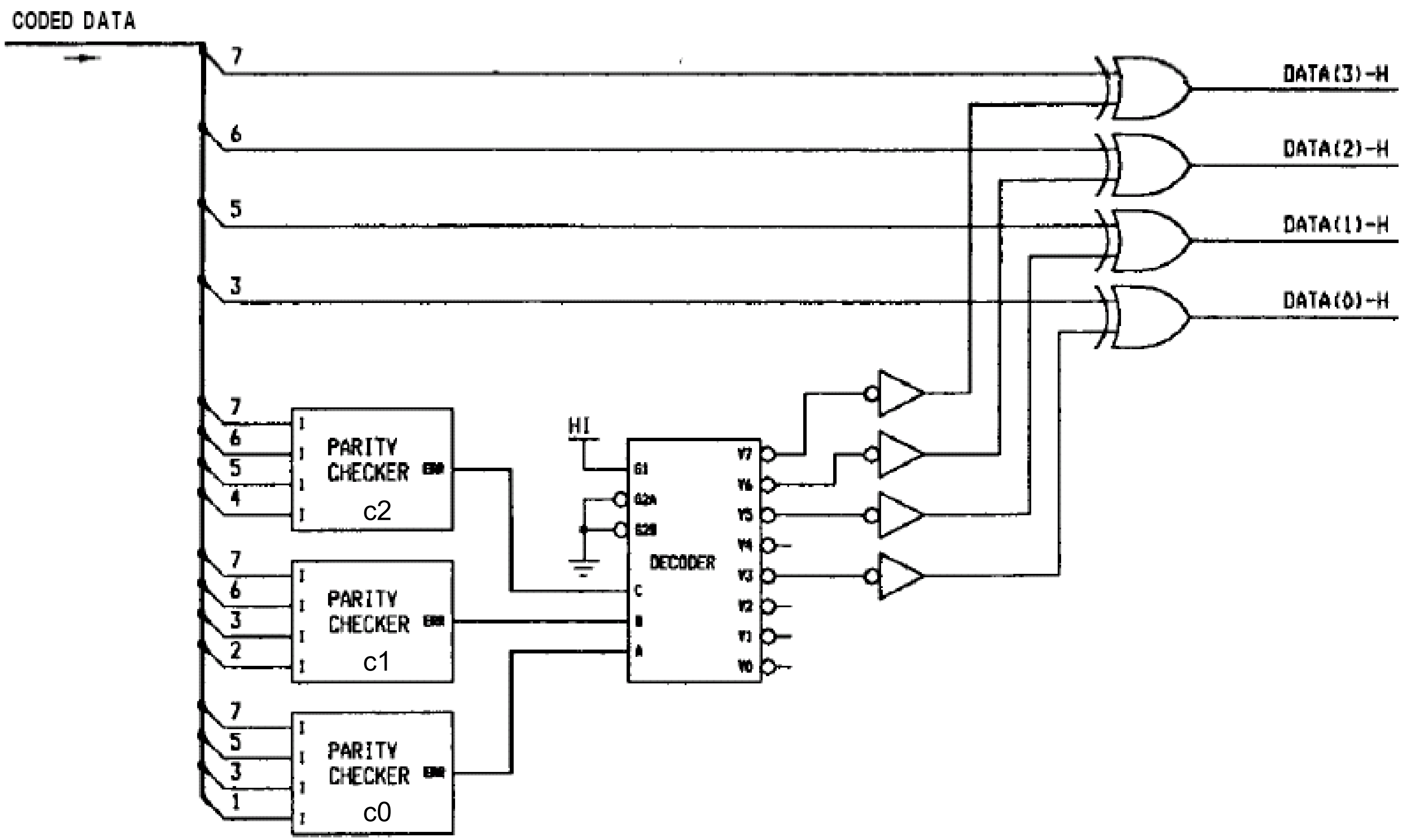


Paritás-csoportok	Bit pozíciók	Bitek jelölései
0	1, 3, 5, 7	C0, D0, D1, D3
1	2, 3, 6, 7	C1, D0, D2, D3
2	4, 5, 6, 7	C2, D1, D2, D3

Pl. 1) Hamming kódú hibajavító áramkör tervezése

- 3 kódbitünk van, így írásnál 3 paritásbit generáló-, míg olvasásnál 3 paritás-ellenőrző áramkör kell.
- Példa: 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).
  - Error syndrome: Hiba esetén például, tfh. az input mintázat 0100010 -ra változik, akkor a 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. 0100010 ⇒ 0100110. Ekkor a kódbitek a következőképpen módosulnak a páratlan paritásnak megfelelően: C2=0, C1=1 és C0=0.

7-bites Hamming kódú hibajavító áramkör felépítése



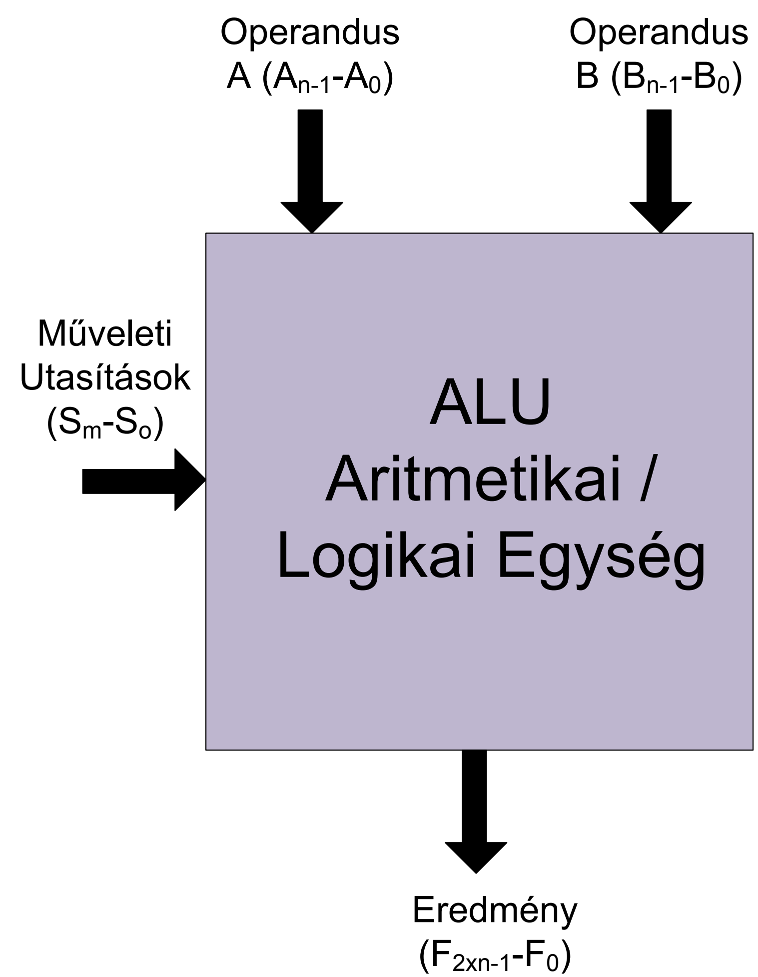
ALU: Aritmetikai Logikai Egységek

Tervezői célkitűzés

- Komplex funkció megvalósítása minimális kapu felhasználásával
  - Minimális késleltetés legyen az adat-úton
- Univerzálisan teljes leírás (K.H.):
  - NAND illetve,
  - NOR kapuk segítségével minden komplex logikai függvény felírható!
- Aritmetika univerzális építőeleme: összeadó
  - belőle a többi elem (-, *, /) származtatható!

ALU felépítése

- Utasítások hatására a (S_m-S_0) vezérlőjelek kijelöli a végrehajtandó aritmetikai / logikai műveletet. További adatvonalak kapcsolódhatnak közvetlenül a státusz regiszterhez, amely fontos információkat tárol el: pl.
  - *zero bit*
  - *carry-in, carry-out* átviteleket,
  - *előjel* bitet (sign),
  - *túlcsordulást* (overflow), vagy *alulcsordulást* (underflow) jelző biteket.

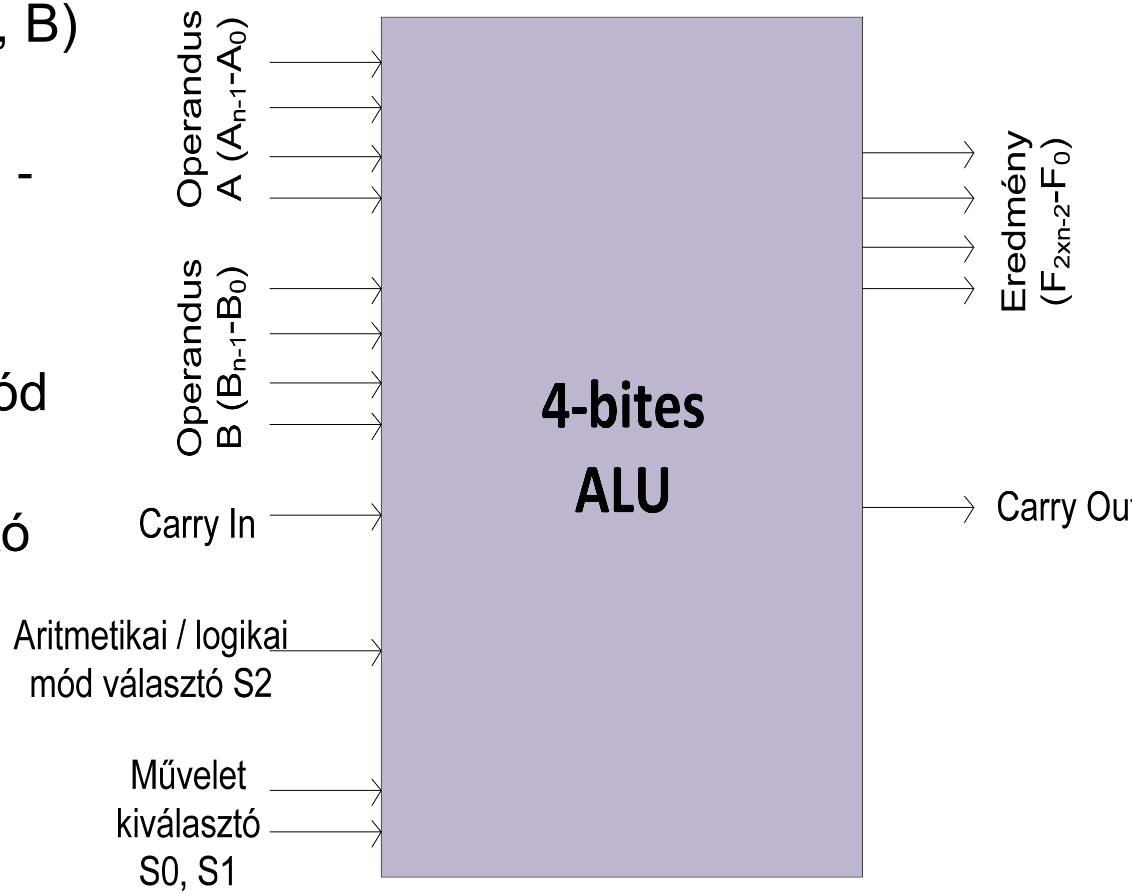


Státusz- (flag) jelzőbitek

- Az aritmetikai műveletek eredményétől függően hibajelzésre használatos jelzőbitek. Ezek megváltozása az utasításkészletben előre definiált utasítások végrehajtásától függ.
  - a.) Előjelbit (sign): 2's komplemens (MSB)
  - b.) Átvitel kezelő bit (carry in/out): helyiértékes átvitel
  - c.) Alul / Túlcsordulás jelzőbit (underflow / overflow)
  - d.) Zero bit: kimeneten az eredmény 0-e?
    - Pl: 0-val való osztás!
    - (szorzásnál egyszerűsíthetőség – adatfüggés)
  - e.) Paritás bit: páros, páratlan

PI: 4-bites ALU felépítése és működése

- Két 4-bites operandus (A, B)
- Eredmény (F)!
  - N+(1 Carry) bites, ha + , -
  - 2*N bites, ha * , /
- Átvitel: Carry In/ Out
- S2: Aritmetikai/ logikai mód választó (MUX)
- S0, S1: művelet kiválasztó (*S*2 értékétől függően!)



IC: '181 ALU 4-bites (74LS181)

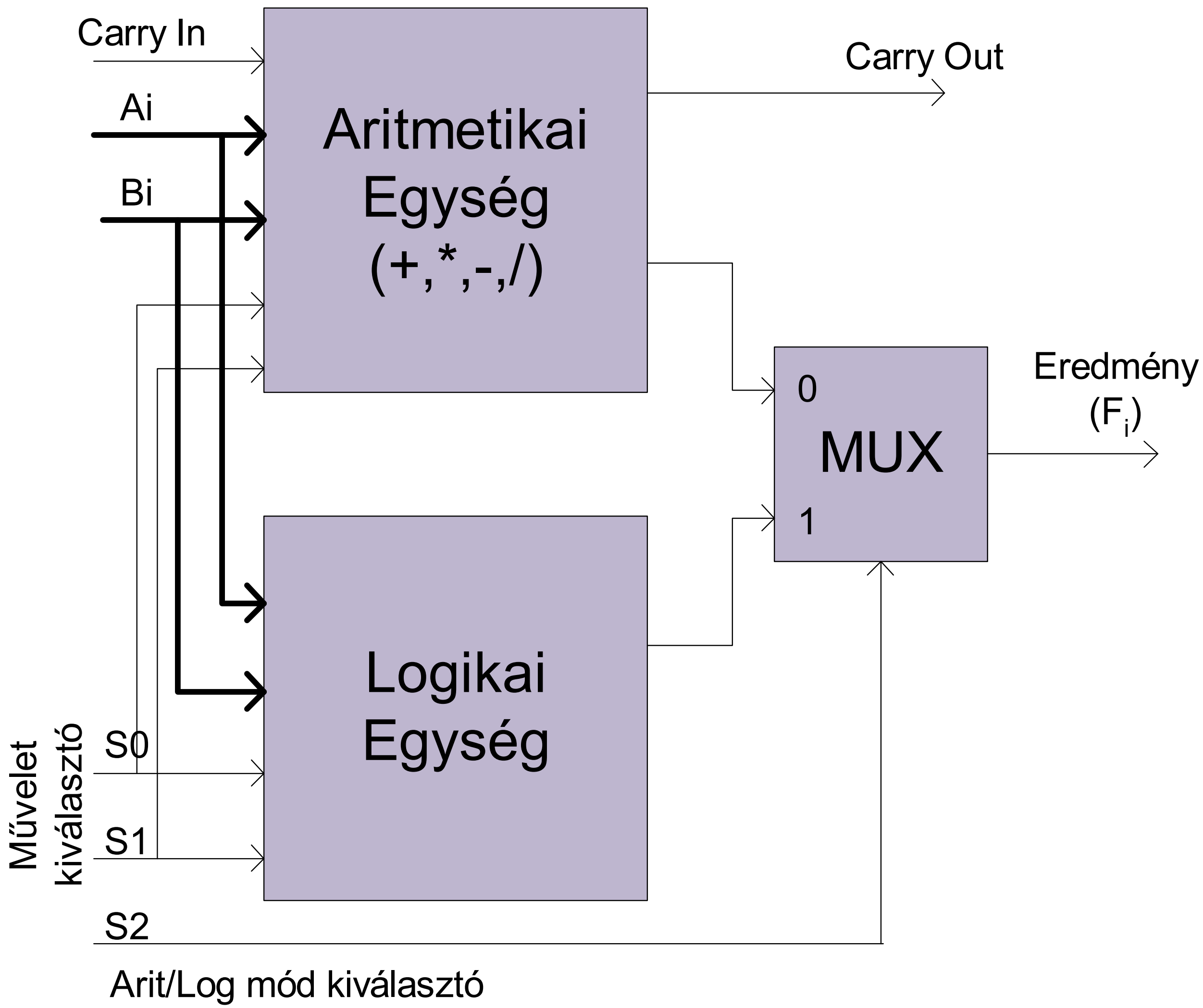
4-bites ALU szimbolikus rajza

ALU működését leíró függvénytáblázat:

Művelet kiválasztás:				Művelet:	Megvalósított függvény:
S2	S1	S0	Cin		
0	0	0	0	F=A	'A' átvitele
0	0	0	1	F=A+1	'A' értékének növelése 1-el (increment)
0	0	1	0	F=A+B	Összeadás
0	0	1	1	F=A+B+1	Összeadás carry figyelembevételével
0	1	0	0	$F = A + \overline{B}$	A + 1's komplemens B
0	1	0	1	$F = A + \overline{B} + 1$	Kivonás
0	1	1	0	F=A-1	'A' értékének csökkentése 1-el (decrement)
0	1	1	1	F=B	'B' átvitele
1	0	0	x	$F = A \wedge B$	AND
1	0	1	x	$F = A \vee B$	OR
1	1	0	x	$F = A \oplus B$	XOR
1	1	1	x	$F = \overline{A}$	'A' negáltja (NOT A)

* Forrás: Könyv függeléke (appendix)

ALU felépítése:



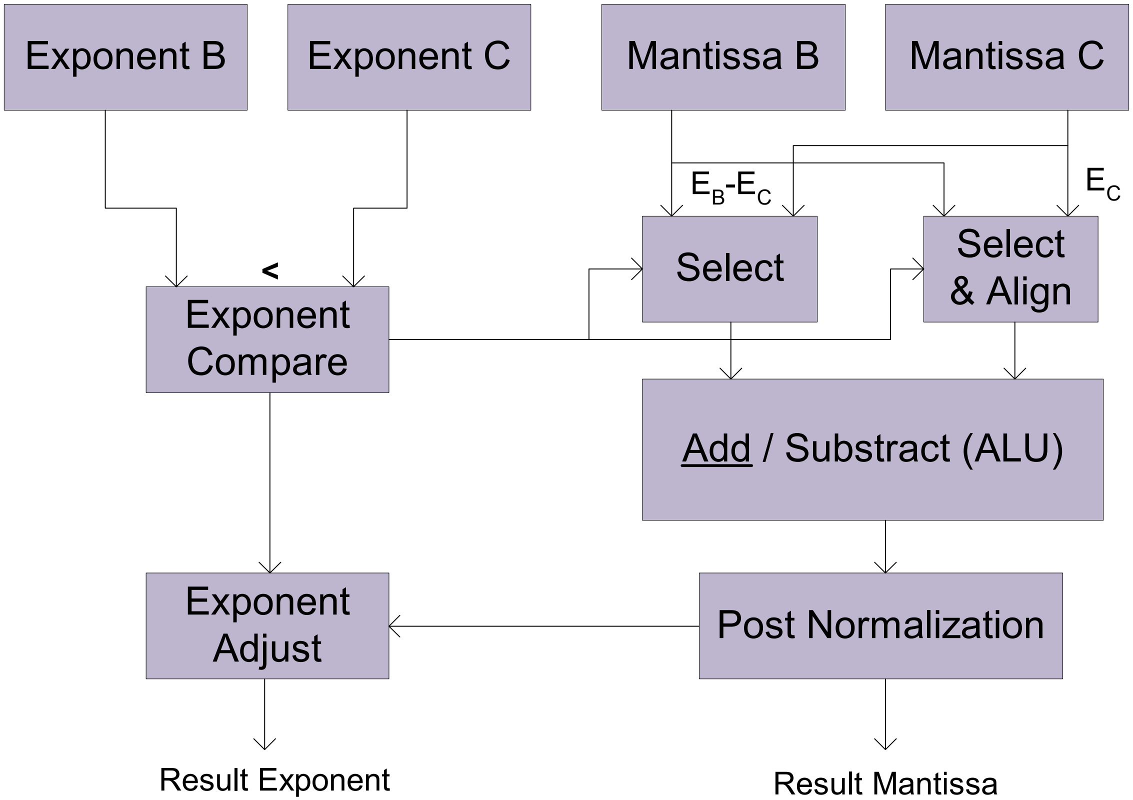
Lebegőpontos műveletvégző egységek

Lebegőpontos műveletvégző egységek

- Probléma:
  - Mantissza igazítás → Exponens beállítás
  - Normalizálás (DEC-32, IEEE-32, IBM-32)
- Műveletvégző elemek:
  - Összeadó-,
  - Kivonó-,
  - Szorzó-,
  - Osztó áramkörök.

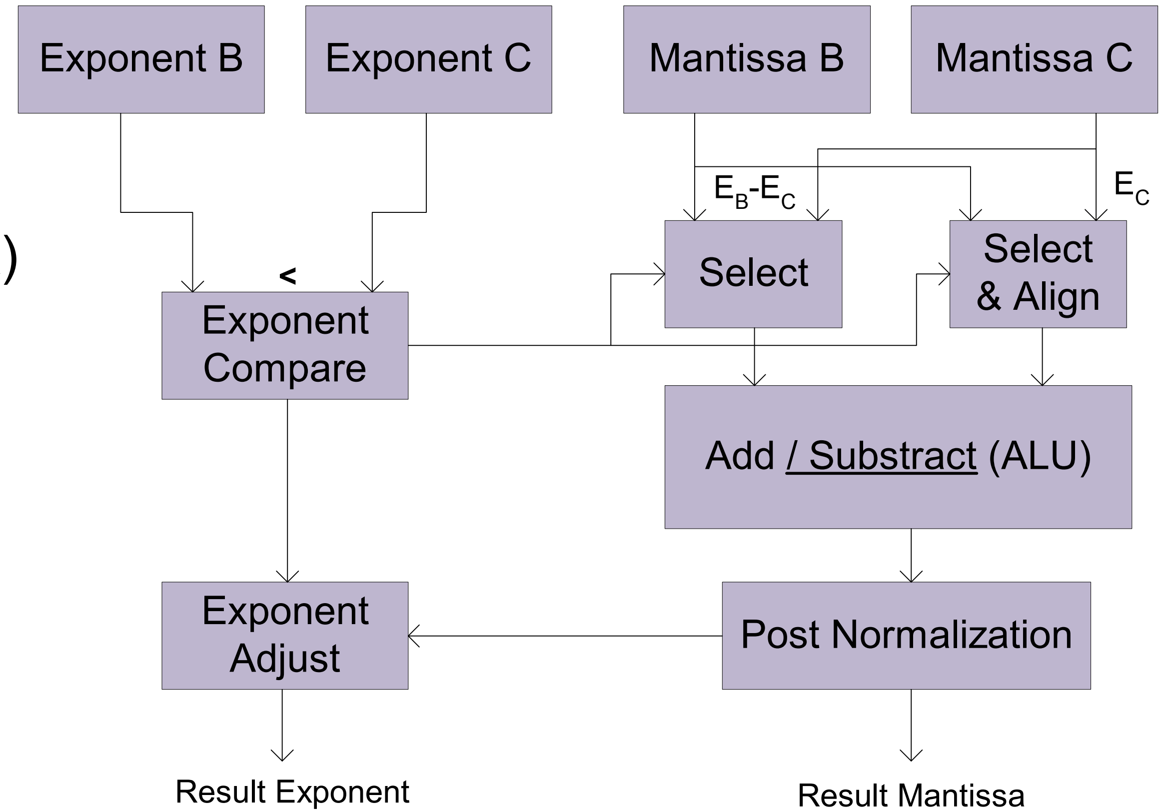
a.) Lebegőpontos összeadó

- Művelet: $A = M_B \times r^{E_B} + M_C \times r^{E_C} = (M_B \times r^{|E_B - E_C|} + M_C) \times r^{E_C}$
  - Komplex feladat: a mantisszák hosszát egyeztetni kell (MSB bitek azonos helyiértéken legyenek)
  - Legyen: 0<B<C
  - B → C vagyis $|E_B – E_C|$ vel jobbra igazítjuk a mantisszát; ez változás az exponensben is
  - ALU: Összeadás!: sign-magnitude formátumban
  - Végül minimális post-normalizáció kell



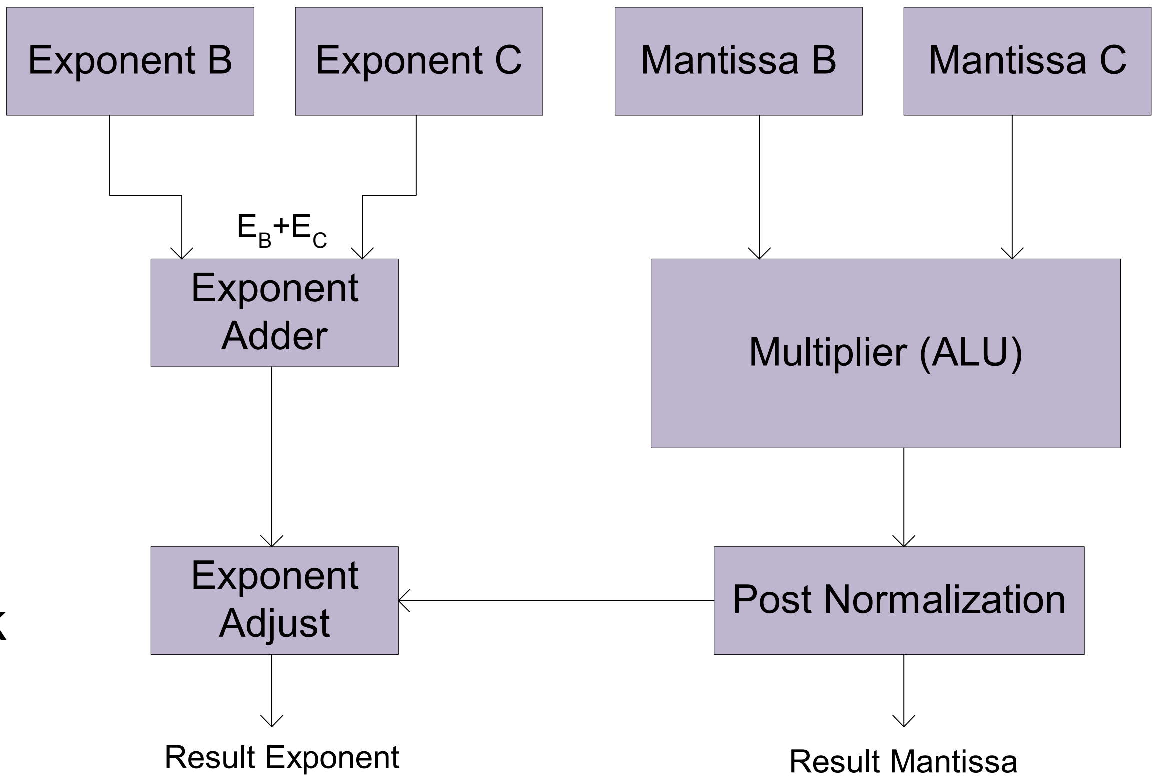
b.) Lebegőpontos kivonó

- Művelet: $A = M_B \times r^{E_B} - M_C \times r^{E_C} = (M_B \times r^{|E_B - E_C|} - M_C) \times r^{E_C}$
  - Komplex feladat: a mantisszák hosszát egyeztetni kell (MSB bitek azonos helyiértéken legyenek)
  - Legyen: 0<B<C
  - B → C vagyis $|E_B - E_C|$-vel jobbra igazítjuk a mantisszát, ez változás az exponensben is
  - Kivonás! (ALU)



c.) Lebegőpontos szorzó

- Művelet: $A = B \times C = M_B \times r^{E_B} \times M_C \times r^{E_C} = (M_B \times M_C) \times r^{E_B + E_C}$
  - A: szorzat
  - B: szorzandó
  - C: szorzó
  - Könnyű végrehajtani
  - Nincs szükség az operandusok beállítására
  - Minimális post-normalizációt kell csak végezni



d.) Lebegőpontos osztó

- Művelet: $A = B/C = M_B \times r^{E_B} / M_C \times r^{E_C} = (M_B / M_C) \times r^{E_B - E_C}$
  - A: hányados
  - B: osztandó
  - C: osztó
  - Könnyű végrehajtani
  - Nincs szükség az operandusok beállítására
  - Minimális post-normalizációt kell csak végezni
  - Osztás! (ALU)