



Számítógép Architektúrák II.

(MIVIB344ZV)

3. előadás: Aritmetikai műveletvégző egységek –
összeadás, kivonás

Előadó: Dr. Vörösházi Zsolt

voroshazi.zsolt@mik.uni-pannon.hu

Jegyzetek, segédanyagok:

- Könyvfejezetek:

- <http://www.virt.uni-pannon.hu> → Oktatás →
Tantárgyak → Számítógép Architektúrák II.
 - ([chapter03.pdf](#))

- Fóliák, óravázlatok .ppt (.pdf)

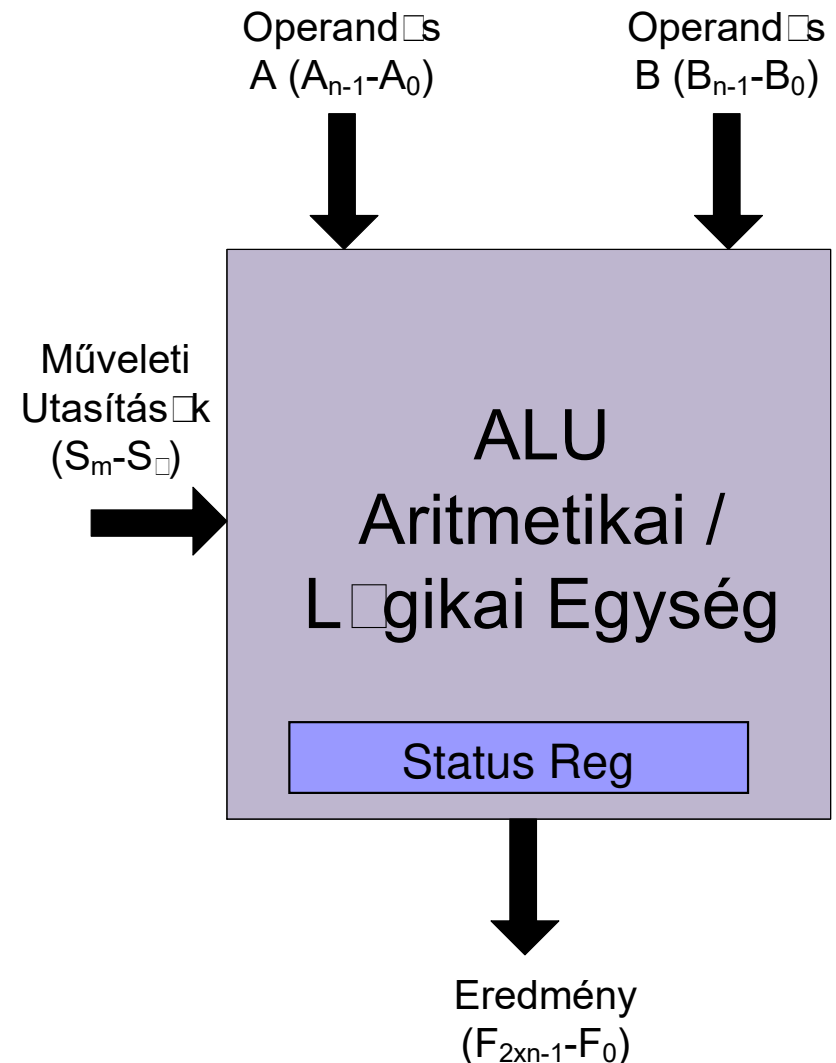
- Feltöltésük folyamatosan

Ismétlés

- Korai számítógépek teljesítményét főként ballisztikus számításoknál (hadászatban)
- Információ ábrázolás
- Használt utasításkészlet (RISC vs. CISC)
- Adatkezelő / műveletvégző egység: Alapvető ALU (Aritmetikai és Logikai funkciók)
 - Univerzális / funkcionális teljesség
 - Aritmetikai operátorok: $+$ \rightarrow $-$, $*$, $/$ (alapműveletek)
 - Logikai operátorok:
NAND, NOR \rightarrow NOT, AND, OR, XOR (AV), NXOR (EQ) – mai *CMOS VLSI technológia* esetén

ALU felépítése

- Utasítások hatására a (S_m-S_0) „vezérlőjelek” jelölik ki a végrehajtandó aritmetikai / logikai műveletet.
- További adatvonalak kapcsolódhatnak közvetlenül a **státusz regiszterhez**, amelyben fontos információkat tárolunk: pl.
 - *zero bit*
 - *carry-in, carry-out* átviteleket,
 - *előjel* bitet (sign),
 - *túlcsordulást* (overflow), vagy *alulcsordulást* (underflow) jelző biteket,
 - Paritás, stb.

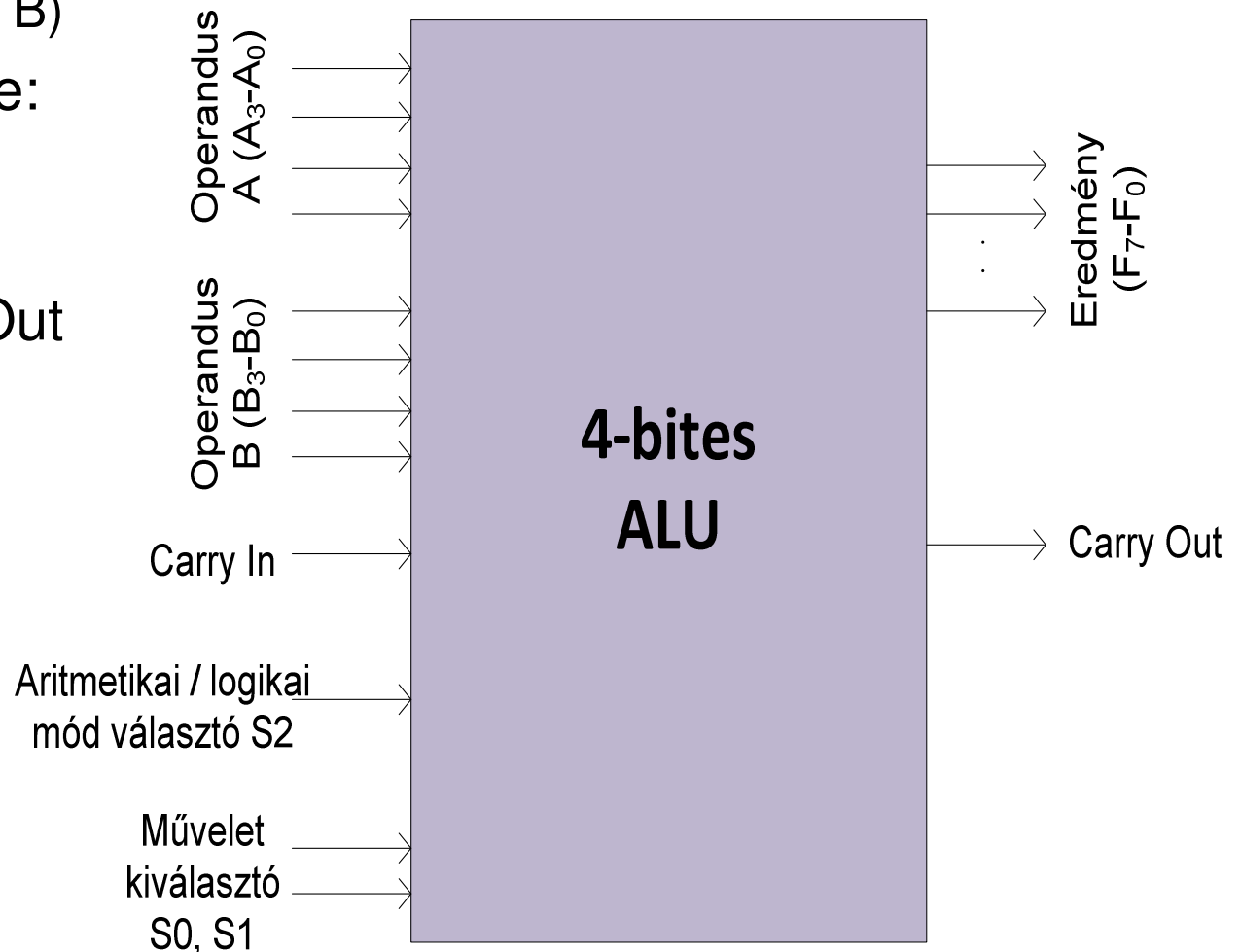


Státusz- (flag) jelzőbitek

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

Példa: N=4-bites ALU felépítése és működése

- Két N=4-bites operandus (A, B)
- Eredmény (F) bitszélessége:
 - $N + (1 \text{ CarryOut})$ bit, ha +;-
 - $2 \times N$ bites, ha *
- H.értékes átvitel: CarryIn/ Out
- S2: Aritmetikai/ logikai mód választó (MUX)
- S0, S1: művelet kiválasztó (S2 értékétől függően)
 - Aritmetikai vs. Logikai

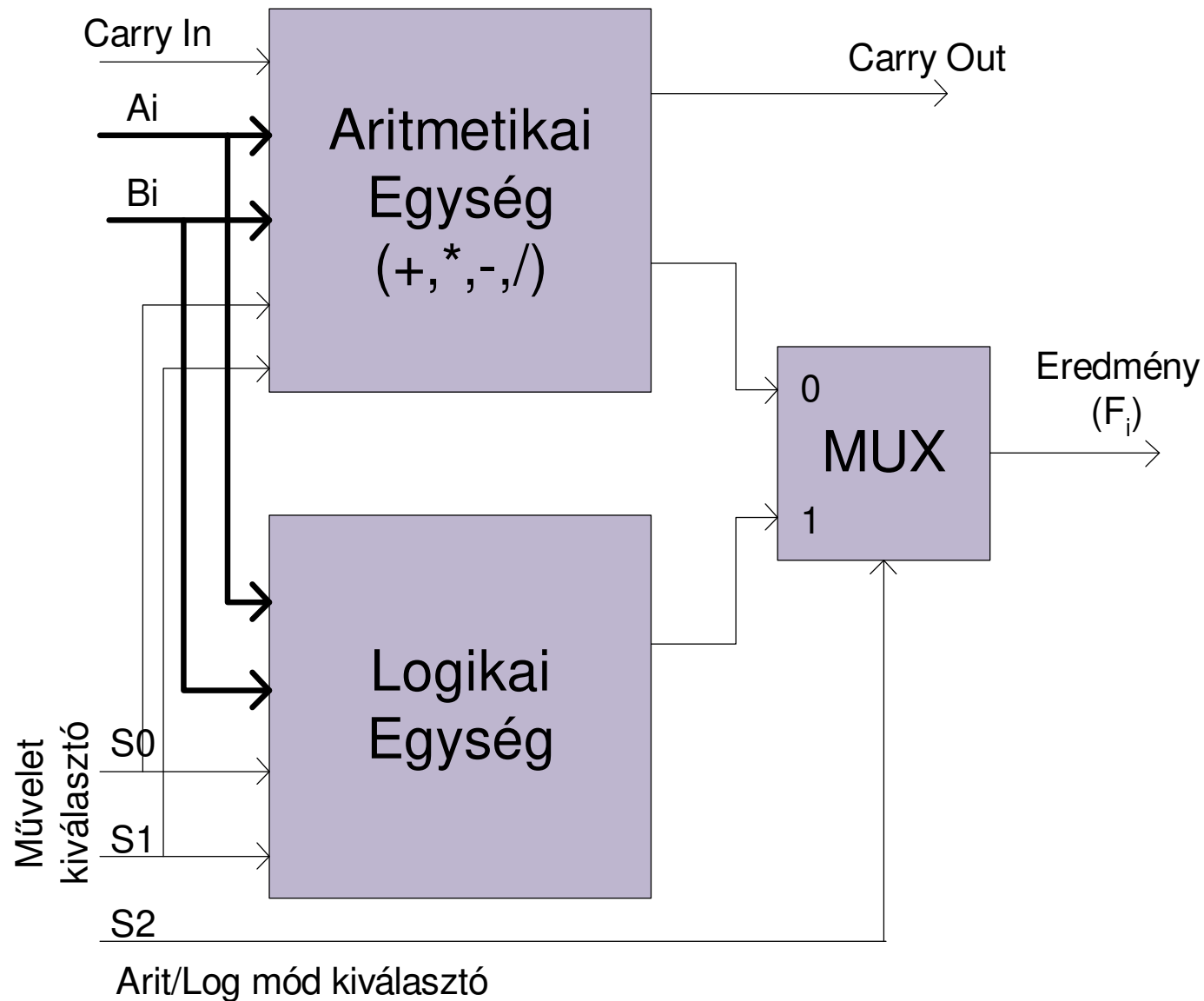



ALU működését leíró függvénytáblázat

(egy lehetséges működés, a funkciók bővíthetők):

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 komplement B
0	1	0	1	$F = A + \overline{B} + 1$	Kivonás = 2’s összeadás!
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)

ALU felépítése:





Lebegőpontos műveletvégző egységek

Lebegőpontos műveletvégző egységek

■ Probléma:

- Mantissa igazítás → Exponens beállítás
- Normalizálás, Utó-(Post) Normalizálás
 - (DEC-32, IEEE-32, IBM-32)

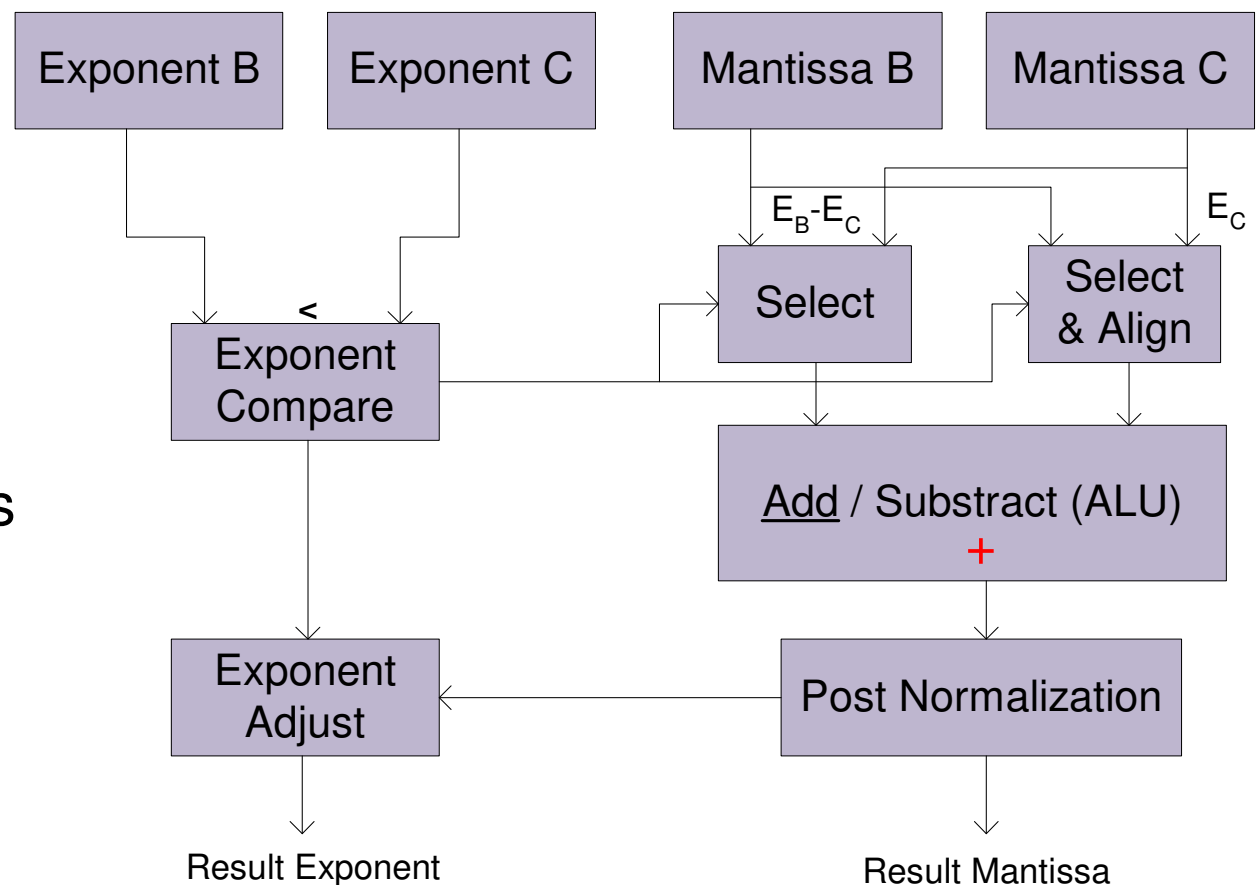
■ Műveletvégző elemek:

- Összeadó (adder)-,
- Kivonó (subtractor)-,
- Szorzó (multiplier)-,
- Osztó (divider) áramkörök.

a.) Lebegőpontos összeadó

■ **Művelet:** $A = B + C = M_B \times r^{E_B} + M_C \times r^{E_C} = (M_B \times r^{|E_B - E_C|} + M_C) \times r^{E_C}$

- Komplex feladat: a mantisszák hosszát egyeztetni kell (MSB bitek azonos helyiértéken legyenek)
- Legyen: $0 < B < C$
- $B \rightarrow C$ vagyis $|E_B - E_C|$ pozícióval jobbra igazítjuk az M_B mantisszát; ez változás az exponensben is
- ALU: Összeadás!: sign-magnitude formátumban
- Végül minimális post-normalizáció kell



Példa: Lebegőpontos összeadás (kivonás)

$r_b=2$, IEEE-754 (bináris 32-bites rendszer):

■ $A = B + C = 10.0001 + 1101.1 = //B < C//$

$$(1.00001 \times 2^{\mathbf{1}}) + (1.1011 \times 2^{\mathbf{3}}) =$$

$$(0.\underline{0}100001 \times 2^{\mathbf{3}}) + (1.1011 \times 2^{\mathbf{3}}) =$$

$$(0.0100001 + 1.1011) \times 2^{\mathbf{3}} =$$

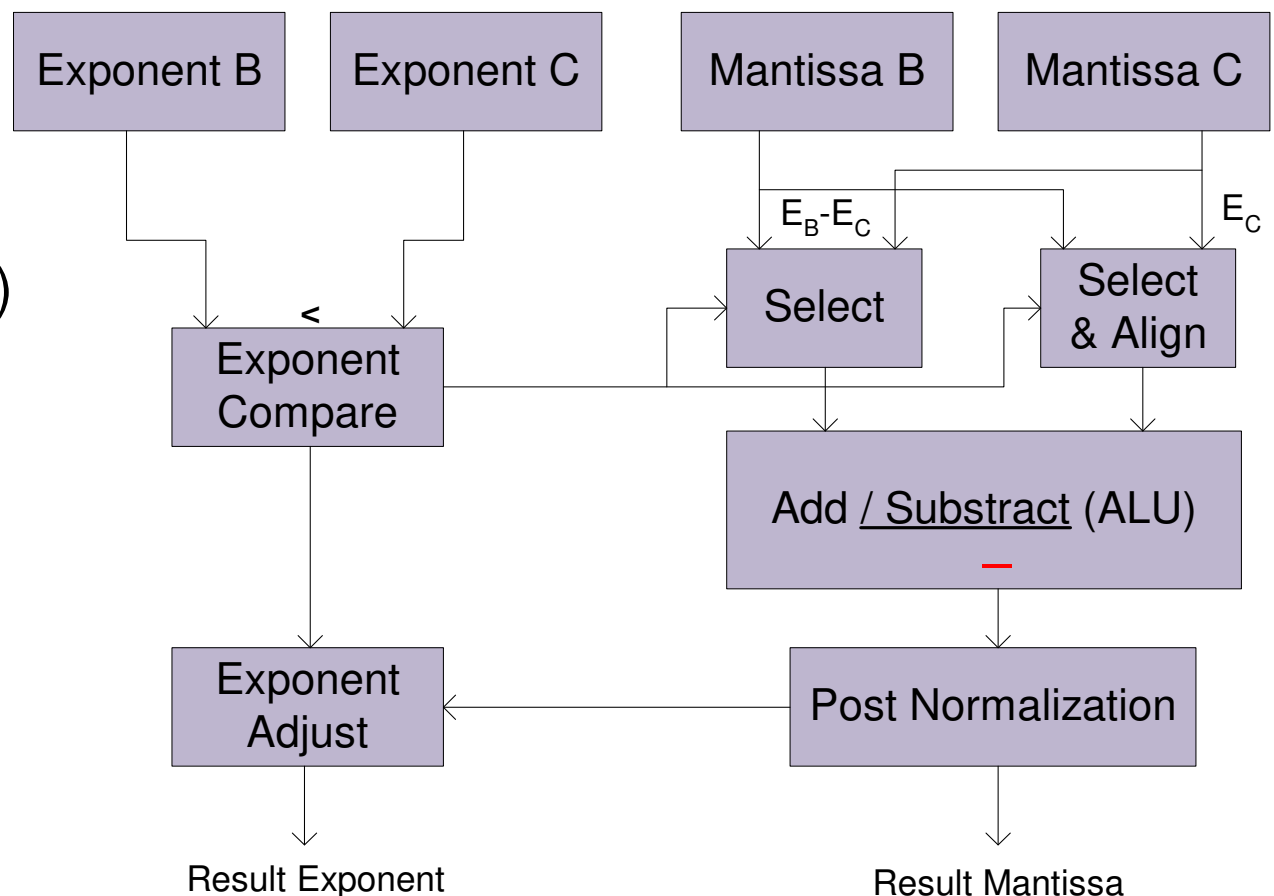
■ $1.1111001 \times 2^{\mathbf{3}} = A$

(itt éppen nem kell post-normalizálás)

b.) Lebegőpontos kivonó

■ **Művelet:** $A = B - C = M_B \times r^{E_B} - M_C \times r^{E_C} = (M_B \times r^{|E_B - E_C|} - M_C) \times r^{E_C}$

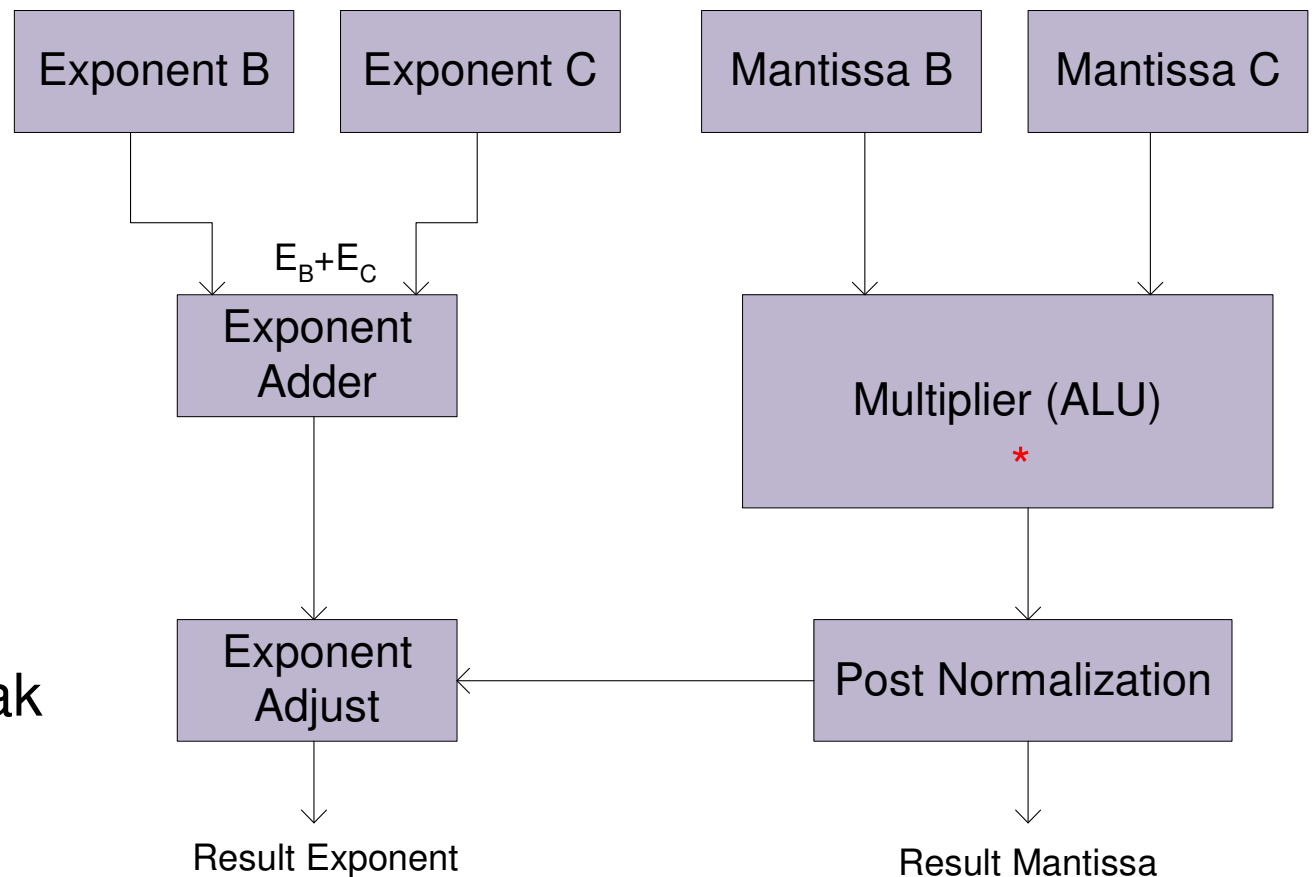
- Komplex feladat: a mantisszák hosszát egyeztetni kell (MSB bitek azonos helyiértéken legyenek)
- Legyen: $0 < B < C$
- $B \rightarrow C$ vagyis $|E_B - E_C|$ pozícióval jobbra igazítjuk az M_B mantisszát, ez változás az exponensben is
- Kivonás! (ALU)



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
- ALU: szorzás!

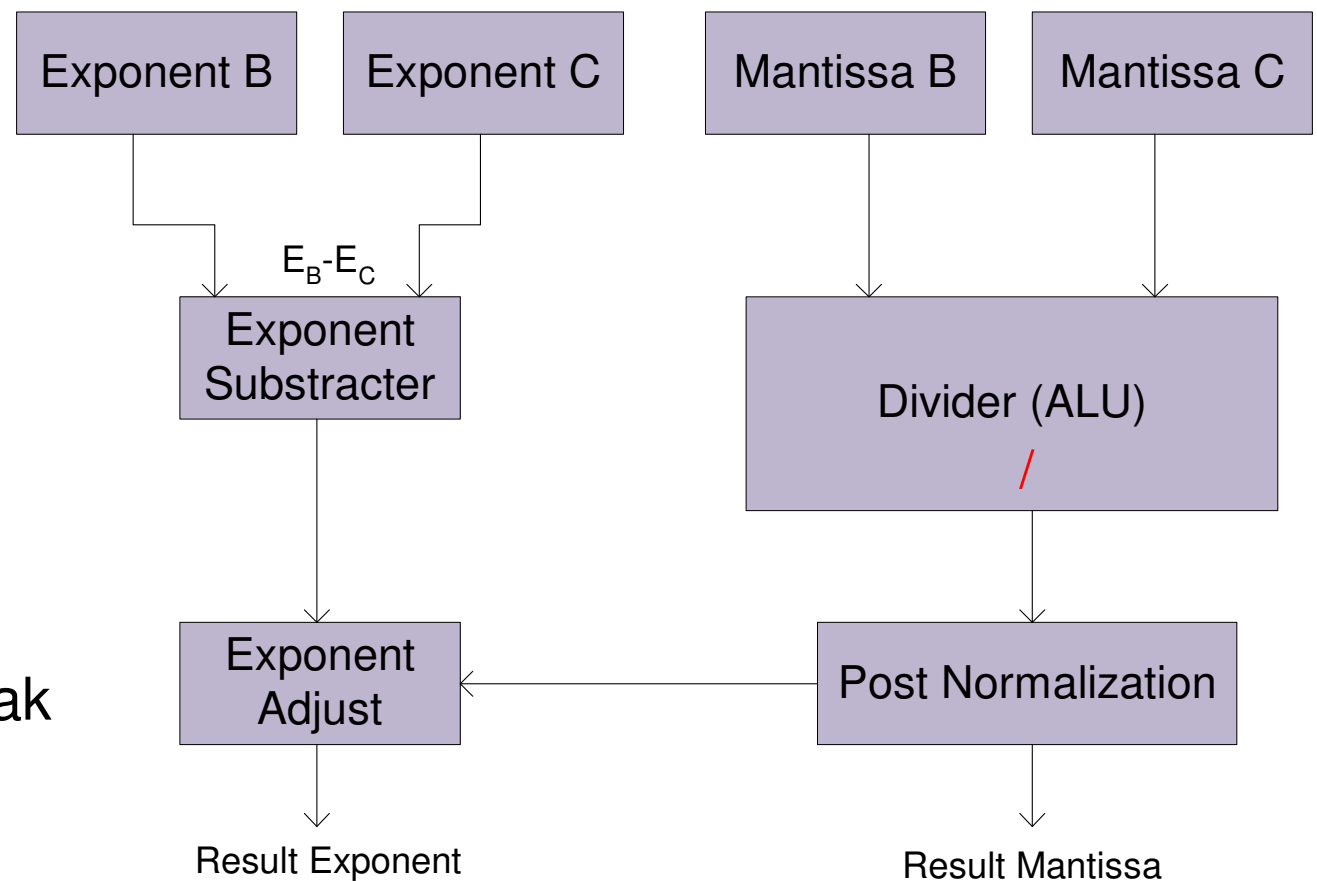


d.) Lebegőpontos osztó

■ 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)





Összeadó áramkörök

a.) Fél-összeadó – Half Adder

■ HA: 1-bites Half Adder

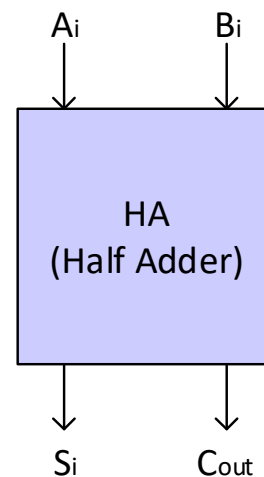
igazságtáblázat

A _i	B _i	Cout	Si
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

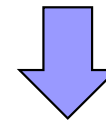
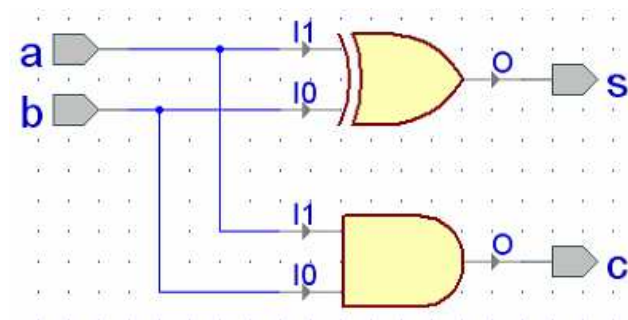
Nem kezeli a Cin-t !

$A + B = \text{Cout} | S$

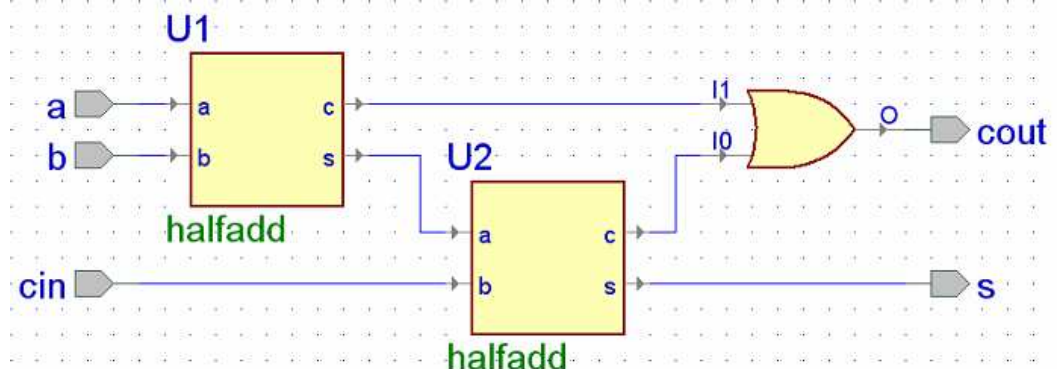
szimbólum



$$T_{HA} = 1G$$



1 bites FA felépítése 2 db HA segítségével:



Karnaugh táblái:

C_{out}:

	B	0	1
A	0	0	0
A	1	0	1

$$C_{out} = A_i \cdot B_i$$

S_i:

	B	0	1
A	0	0	1
A	1	1	0

$$S_i = A_i \oplus B_i$$

Kimeneti fgv-ei:

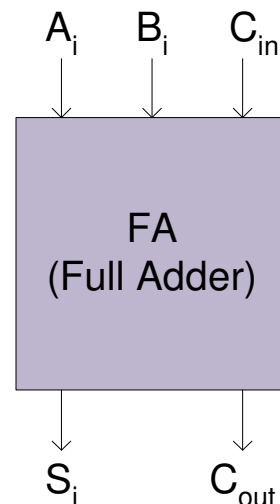
b.) Teljes összeadó – Full Adder

■ FA: 1-bites Full Adder

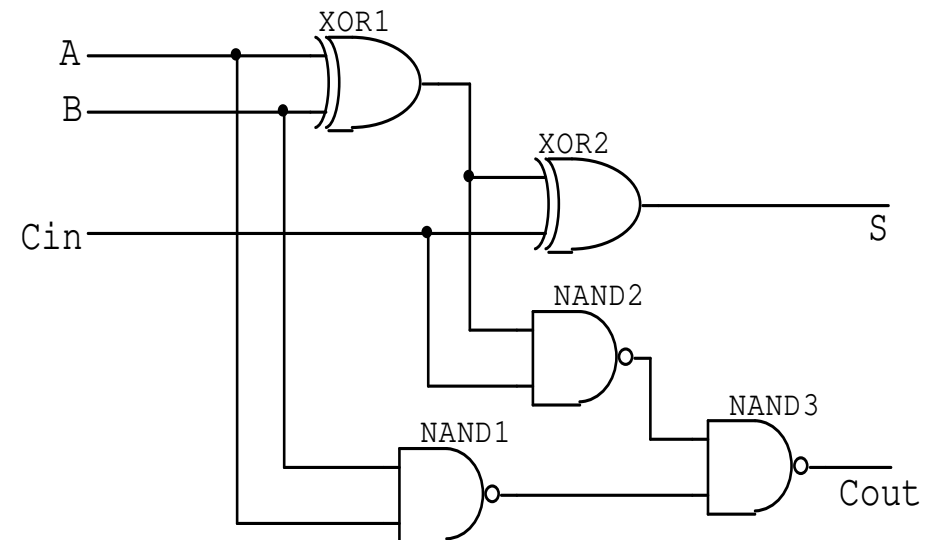
igazságtáblázat

A_i	B_i	C_{in}	C_{out}	Sum_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

szimbólum



Ez a FA egy lehetséges CMOS kapcsolási rajza: (itt $T_{FA} = 3G$!)



$$A + B + C_{in} = C_{out} | S$$

Karnaugh táblái:

C_{out} :

		C_{in}			
		B			
A	C_{in}	00	01	11	10
	0	0	0	1	0
A	1	0	1	1	1

$$C_{out} = A_i \cdot B_i + A_i \cdot C_{in} + B_i \cdot C_{in}$$

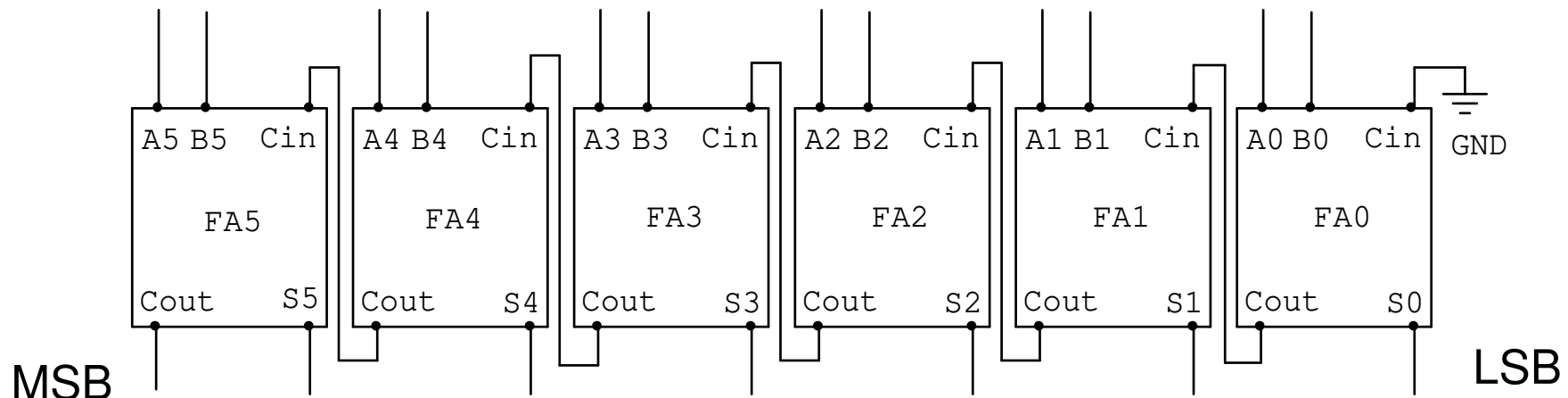
S_i :

		C_{in}			
		B			
A	C_{in}	00	01	11	10
	0	0	1	0	1
A	1	1	0	1	0

$$S_i = A_i \oplus B_i \oplus C_{in}$$

c.) Átvitelkezelő összeadó – Ripple Carry Adder (RCA)

- Példa: 6-bites RCA: [5..0] (LSB Cin = GND!)



- Számítási időszükséglet (RCA):

$$T_{(RCA)} = N \cdot T_{(FA)} = N \cdot (2 \cdot G) = 12 \text{ G (6-bites RCA esetén)}$$

ahol a min. 2G az 1-bites FA kapukésleltetése ([ns], [ps])

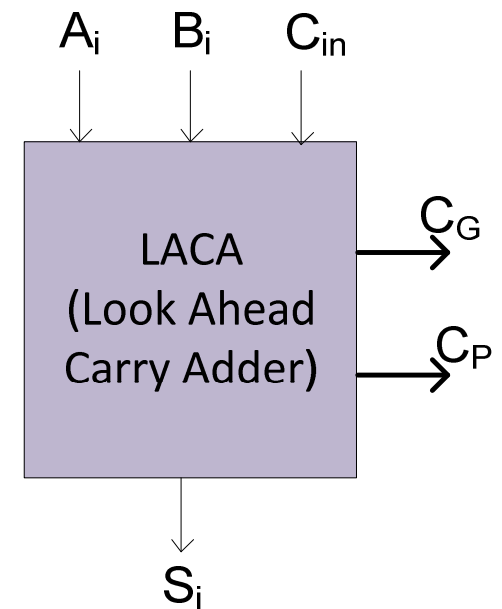
d.) LACA: Look Ahead Carry Adder Átvitelgyorsító összeadó

- Képlet (FA) átalakításából kapjuk:

$$C_{out} = A_i \cdot B_i + A_i \cdot C_{in} + B_i \cdot C_{in}$$

$$\Rightarrow \underbrace{A_i \cdot B_i}_{\text{CarryGenerate}} + C_{in} \cdot \underbrace{(A_i + B_i)}_{\text{CarryPropagate}} = C_G + C_{in} \cdot C_P$$

$$S_i = A_i \oplus B_i \oplus C_{in}$$



LACG: Look Ahead Carry Generator egy **b** bites ALU-hoz kapcsolódik, mindenegybes állapotban a C_{in} generálásáért felel a C_P és C_G (LACA-tól) érkező jeleknek megfelelően („LACG looks at C_P and C_G from adders”).

N-bites LACA számítási időszükséglete: $T_{LACA} = 2 + 4 \times (\lceil \log_b(N) \rceil - 1)$

ahol **N**: bitek száma, **b**: LACG bitszélessége (hány LACA-hoz tartozik egy LACG)

Megjegyzés: LACA – CG átírása XOR kapcsolatra (nem triviális forma)

- CG előállítás: alkalmazott másik érvényes forma a XOR kapcsolattal megadott kifejezés

igazságtáblázat

A_i	B_i	C_{in}	C_{out}	Sum_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Karnaugh tábla:

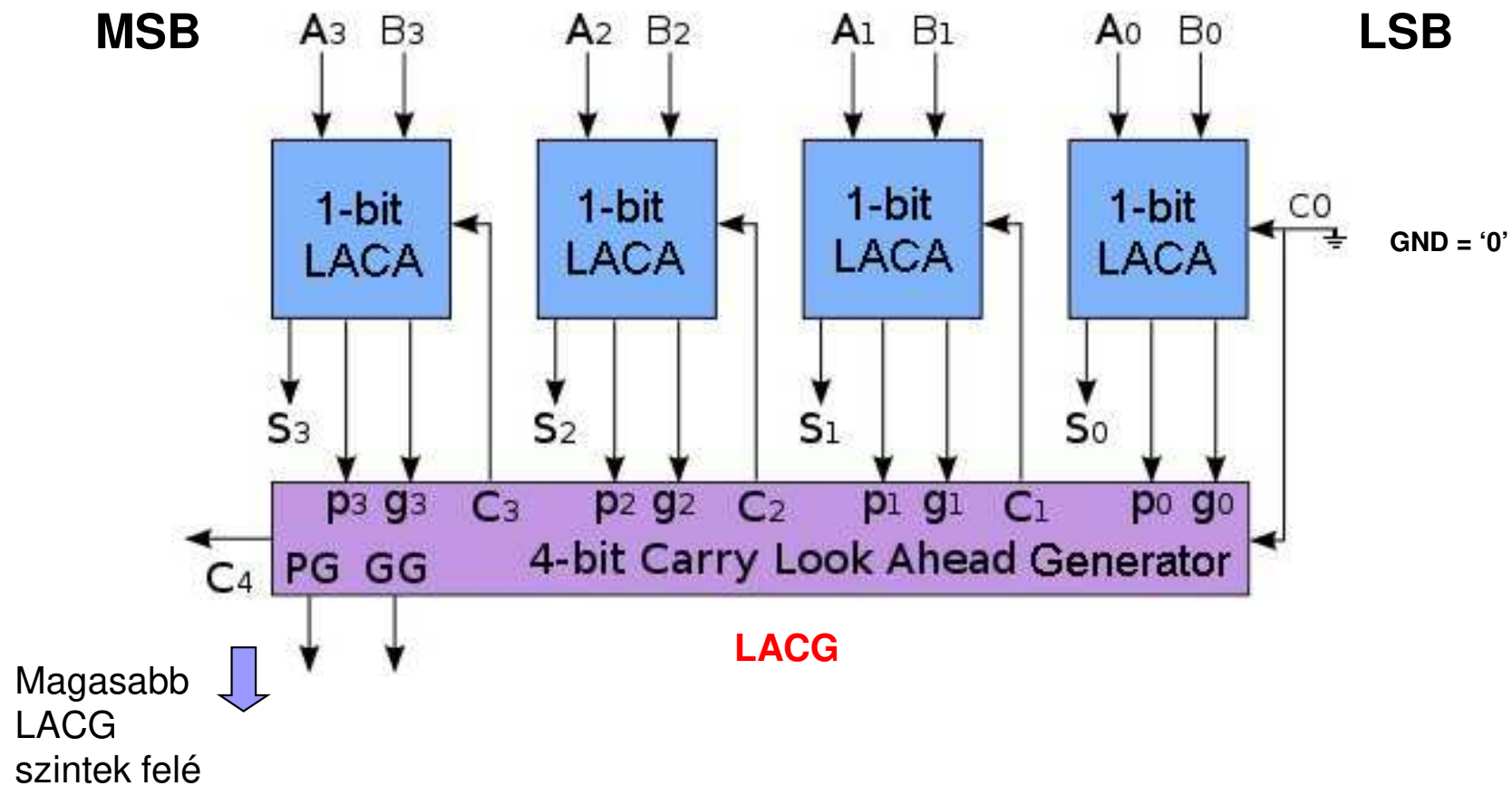
Kimeneti fgv:

		C_{in}			
		B			
A	BC_{in}	00	01	11	10
	C_{out}	0	0	1	0
1		0	1	1	1

$$\begin{aligned}
 C_{out} &= \overline{A_i} \cdot B_i \cdot C_{in} + A_i \cdot \overline{B_i} \cdot C_{in} + A_i \cdot B_i \\
 &= A_i \cdot B_i + C_{in} \cdot (A_i \oplus B_i) = \\
 &= C_G + C_{in} \cdot C_P
 \end{aligned}$$

Példa: 4-bites LACA

- Legyen **b=4 (LACG)**, és **N=4 (LACA)**. Áramkör felépítése, és időszükséglete?



$$T_{LACA} = 2 + 4 \times (\underbrace{\lceil \log_4(4) \rceil}_1 - 1) = 2$$

Példa (folyt.): 4-bites LACA számítási műveletei (carry terjesztés)

- LSB → MSB felé az összeadások

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1$$

$$C_3 = G_2 + P_2 \cdot C_2$$

$$C_4 = G_3 + P_3 \cdot C_3$$

- Behelyettesítések: adott C_i -t → a C_{i+1} -be

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1$$

$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2$$

$$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

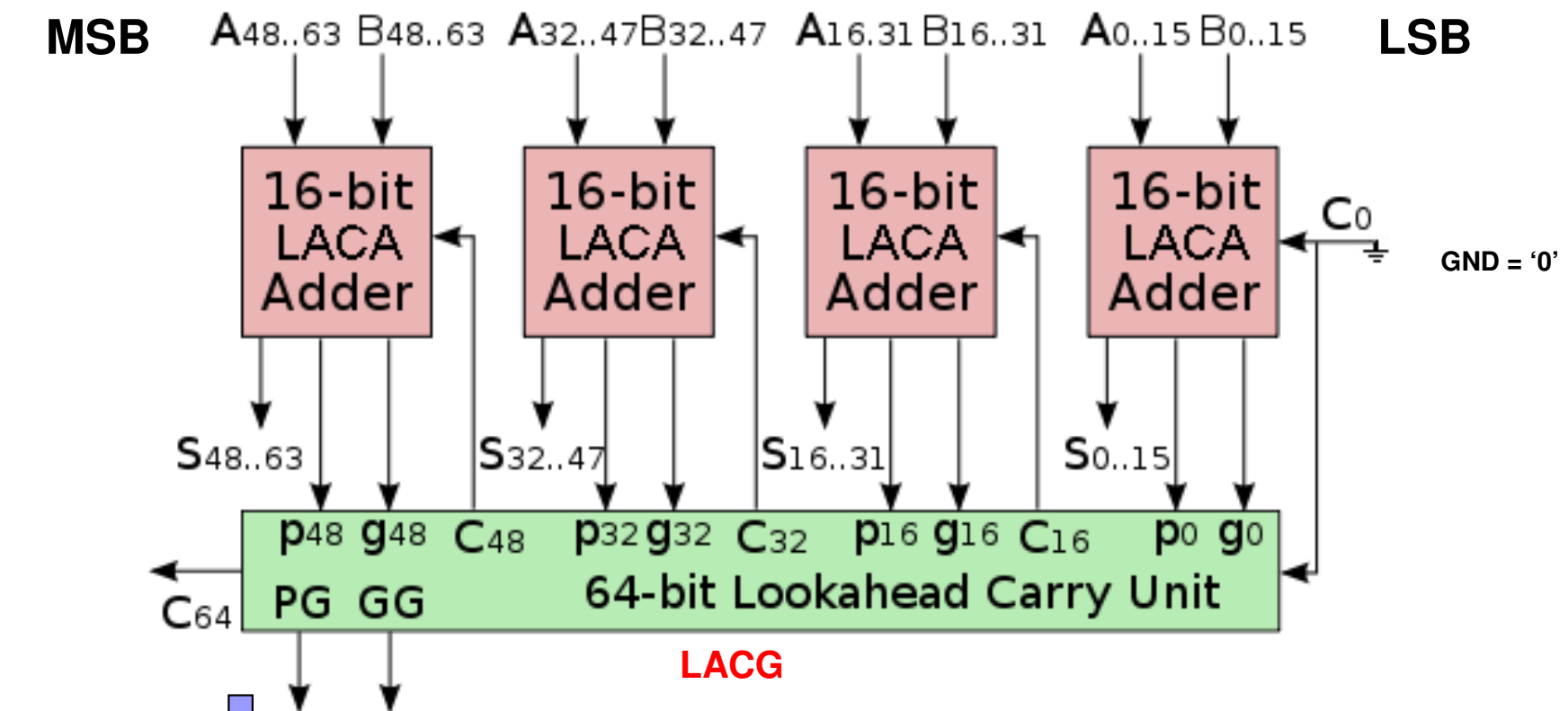
- Magasabb b-bites LACG hierarchia szintek felé GP (group propagate) és GG (group generate) számítása:

$$GG = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

$$PG = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

Példa: 4x16-bites LACA

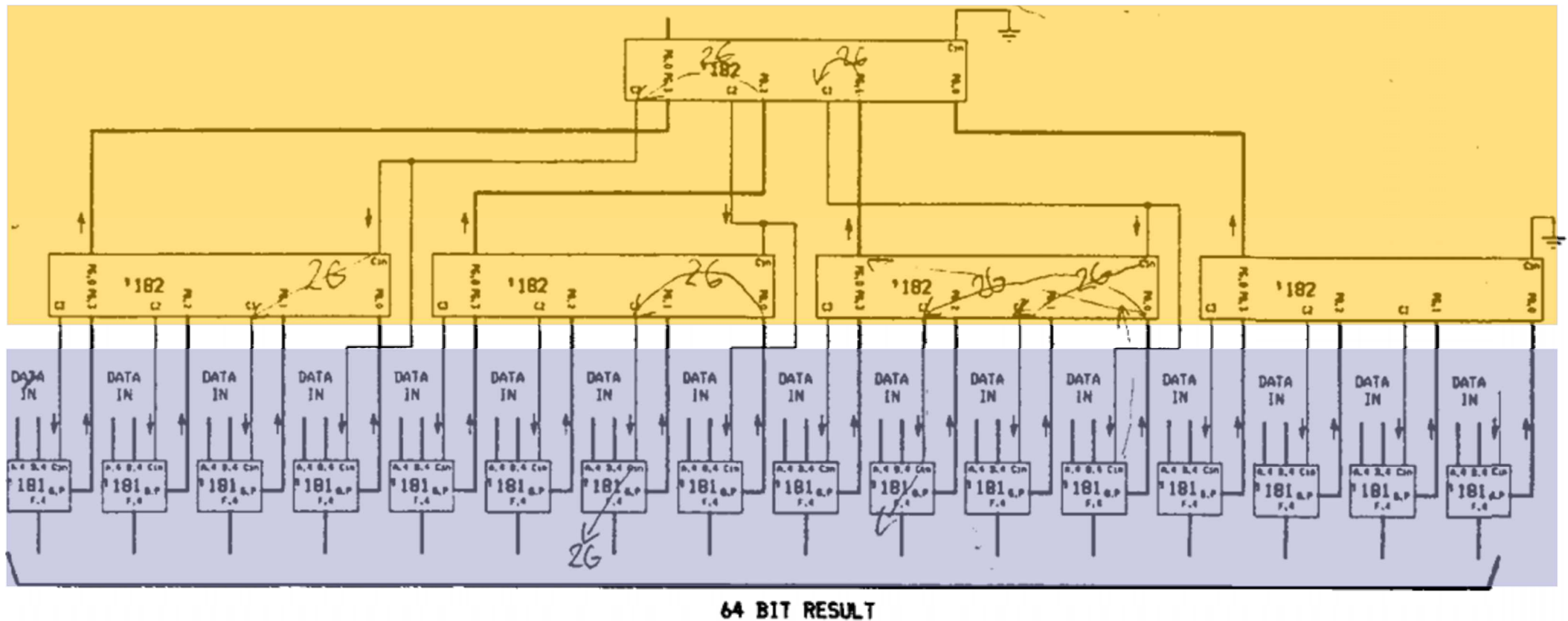
- Legyen **b=64 (LACG)**, és **N=4x16 (LACA)**. Áramkör felépítése, és időszükséglete?



Magasabb
LACG
szintek felé

$$T_{LACA} = 2 + 4 \times (\underbrace{\lceil \log_{64}(64) \rceil}_1 - 1) = 2$$

64-bites (16×4) LACA összeadó



$$T_{LACA} = 2 + 4 \times (\underbrace{\lceil \log_4(64) \rceil}_3 - 1) = 10$$

■ **Komponensek:**

- '182 = SN74LS181: **N=4**-bites ALU (összeadás)
- '181 = SN74LS182: **b=4** bites LACG generátor



Kivonó áramkörök

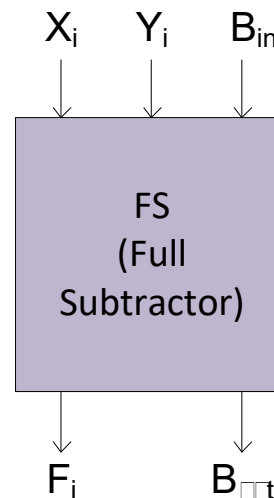
Teljes kivonó - Full Subtractor (FS)

■ FS: 1-bites Full Subtractor

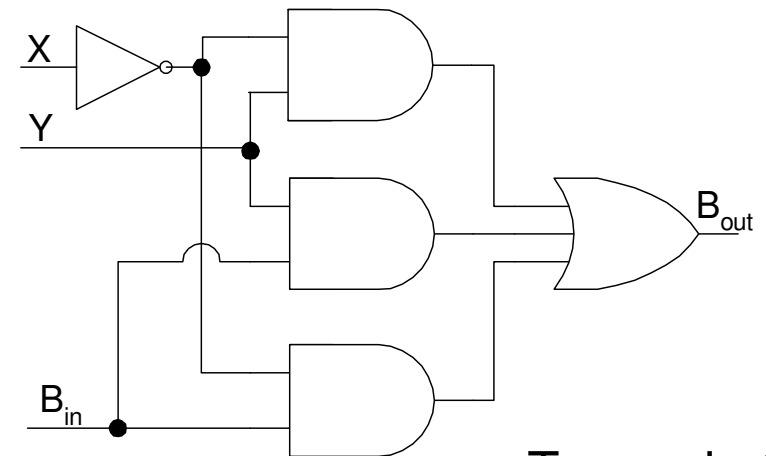
igazságtáblázat

X_i	Y_i	B_{in}	B_{out}	F_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

szimbólum



Logikai kapcsolási rajz Bout-ra
(F előállítása ugyanaz, mint FA-nál):



$$T_{FS} = \min 2G$$

Karnaugh táblái:

B_{out} :

$X \backslash Y \ B_{in}$	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Kimeneti fgv-ei:

$$B_{out} = \overline{X_i} \cdot Y_i + \overline{X_i} \cdot B_{in} + Y_i \cdot B_{in}$$

F_i :

$X \backslash Y \ B_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$F_i = X_i \oplus Y_i \oplus B_{in}$$

Bináris kivonás módszerei

I. módszer:

Bináris kivonás *FS* segítségével

$$\square \frac{X_i - Y_i \rightarrow F_i}{}$$

$$\square 0 - 0 \rightarrow 0$$

$$\square 0 - 1 \rightarrow 1, \text{ borrow bit '1'}$$

$$\square 1 - 0 \rightarrow 1$$

$$\square 1 - 1 \rightarrow 0$$

$ \begin{array}{r} \text{* * * * *} \\ \cancel{100000000} \\ - 01001100 \\ \hline 10110100 \end{array} $	$ \begin{array}{r} \text{*} \\ \cancel{256} \\ - 76 \\ \hline 180 \end{array} $
--	---

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

II. módszer: Kivonás visszavezetése az *univerzálisan teljes* bináris *összeadás* segítségével (2's komplement alak – korábban tanultuk):

□ FA, RCA, vagy LACA

$$F_i = X_i + 2's \text{ comp}(Y_i)$$