Pannon Egyetem
Villamosmérnöki és Információs Tanszék

Digitális Áramkörök

(Villamosmérnök BSc / Mechatronikai mérnök MSc)

12. hét – Kombinációs hálózatok építőelemei MSI megvalósításban. Egyszerű aritmetikai áramkörök

Előadó: Dr. Vörösházi Zsolt
voroshazi.zsolt@virt.uni-pannon.hu

Kapcsolódó jegyzet, segédanyag:

- http://www.virt.uni-pannon.hu
  → Oktatás → Tantárgyak → Digitális Áramkörök (Villamosmérnök BSc / Mechatronikai mérnök MSc).
  - Fóliák, óravázlatok (.ppt)
  - Feltöltésük folyamatosan

Digitális tervezés építőelemei

- Az alacsonyabb absztrakciós szintről (kapuk) feljebb kell lépni a magasabb hierarchia szintek felé (összetett digitális építőelemek).
- Itt helyezkednek el a következő alapvető építőelemeink, pl.:
  - Összeadók (Adder),
  - Regiszterek, memóriák
  - Multiplexerek stb.

Biztosítani kell a tervezőnek:

- Adatmozgatás egyik elemtől a másikig (jelvezetékek, buszok)
- Több forrásból származó adat kiválasztása (multiplexálás)
- Forrásadat irányítása (routing) a cél felé (demultiplexálás)
- Adattranszformálás (ha szükséges): kódolás-dekódolás
- Adatok aritmetikai összehasonlítása (komparálás)
- Logikai, aritmetikai műveletvégzés adatokon (ALU)

Logikai (Kombinációs) hálózatok építőelemei

Logikai hálózatok legfontosabb építőelemei

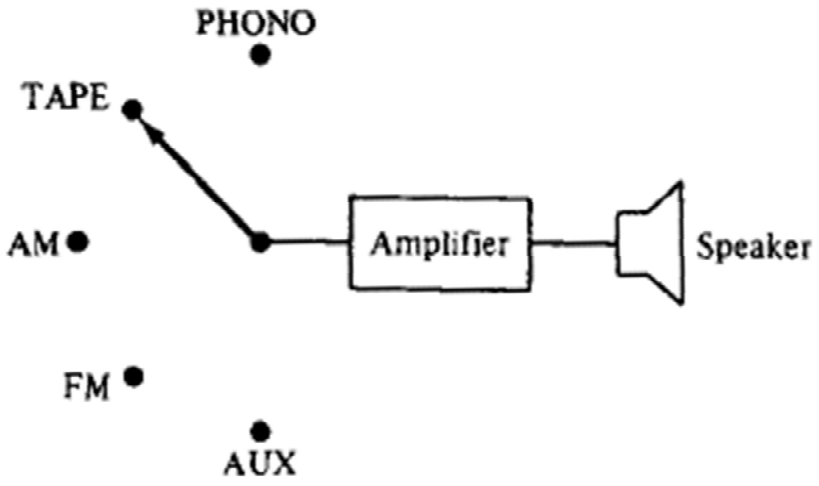
- Multiplexer (MUX)
- Demultiplexer (DEMUX)
- Dekódoló (decoder) áramkör
- Kódoló (encoder) áramkör
- Komparátor (comparator, CMP)
- Univerzális logikai áramkör
- Bináris műveletvégző egységek:
  - Összeadók, kivonók, ALU stb.

Multiplexer/Demultiplexer („útvonalválasztó”)

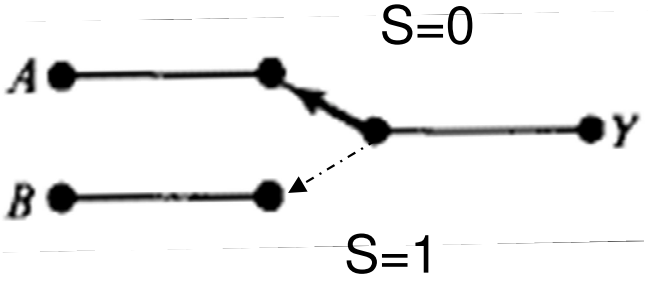
- Cél, az eszköz vagy eszközben lévő komponensek (K.H-ok) kivezetéseinek illetve bemeneteinek számának korlátozása
- Alkalmazás:
  - pl memóriák címzése, indexelése esetén
  - Buszok jeleinek multplexálása, demulitplexálása
- Adatlapok letöltése (ingyenesen):
  - pl: http://www.alldatasheet.com

1. Multiplexer („útvonalválasztó”)

- Olyan áramköri elem, amely *több* lehetséges bemenet közül *választ ki egyet*, a **selector** jel(ek) hatására, és a kiválasztott bemenetet a kimenettel köti össze. Működése hasonló a **mechanikus kapcsolókéhoz**. Például:



Sztereo erősítő kapcsolója

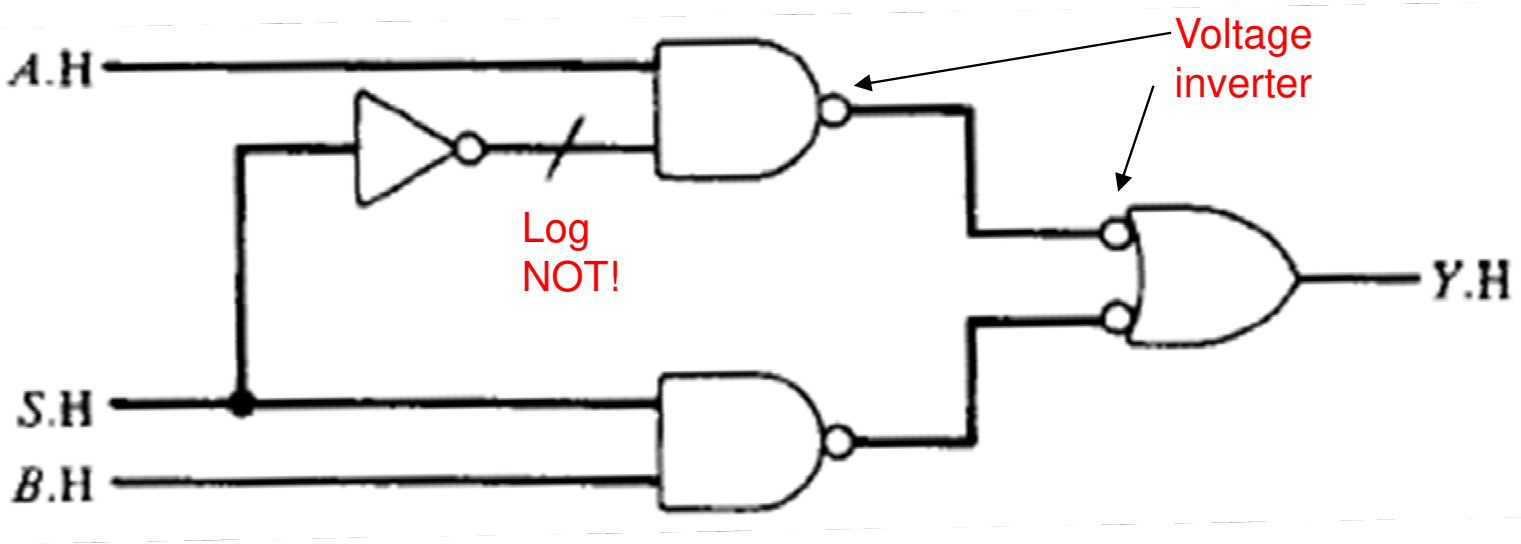


Két-állapotú mechanikus kapcsoló

$$Y = A \cdot \overline{S} + B \cdot S$$

Példa 1: Két-állapotú kapcsoló

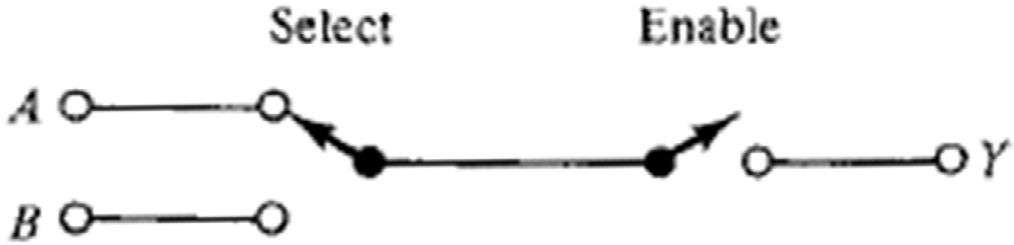
- Legyen Y: kimenet, A, B: bemenetek, amik közül S választ. Ekkor: $Y = A \cdot \overline{S} + B \cdot S$
- A két-állapotú kapcsolót leíró mixed-logic hálózat lehet (ha A.H, B.H, S.H és Y.H):



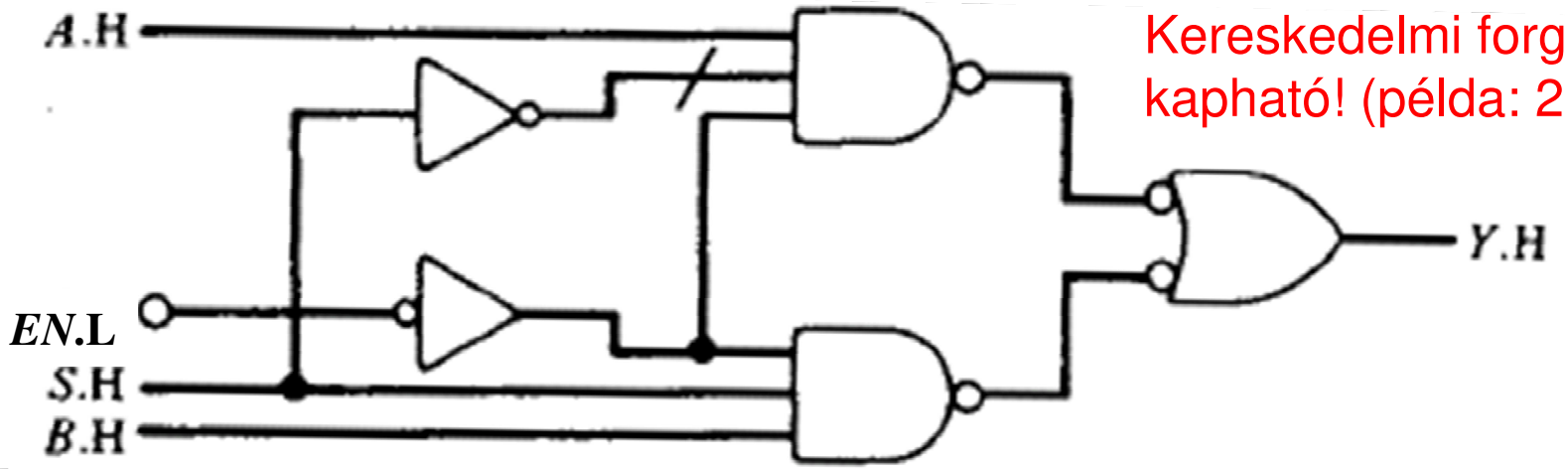
Példa 2: Két-állapotú, ***engedélyező*** bemenettel rendelkező kapcsoló

- EN (Enable vagy STROBE jel): engedélyező jel
- S: selector (kapcsoló állása)
- A, B: bemenetek, Y: kimenet

$$Y = EN \cdot (A \cdot \overline{S} + B \cdot S)$$

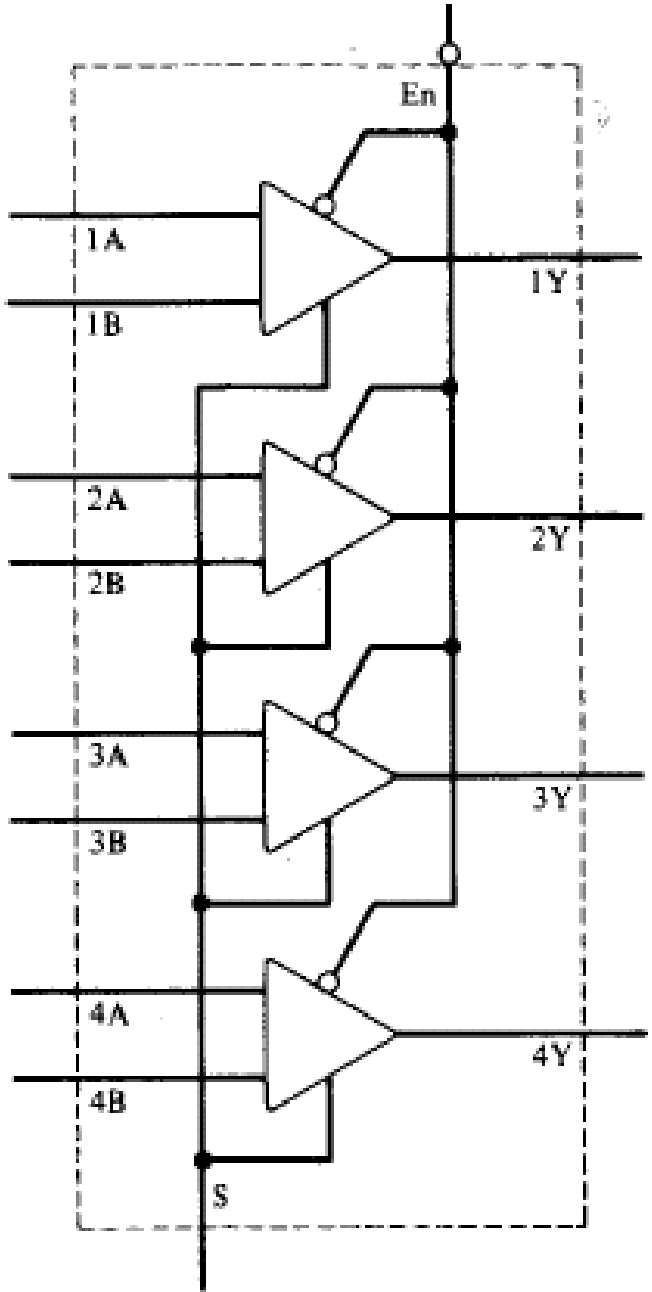


- Mixed-logic kapcsolási rajza (ha A.H, B.H, Y.H, de EN.L):

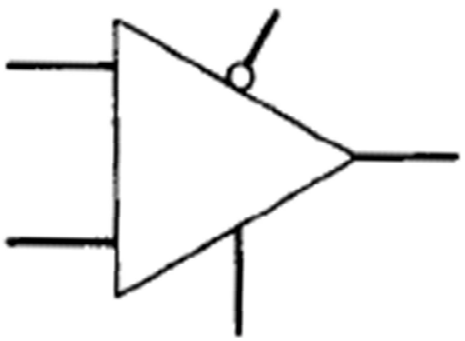


Kereskedelmi forgalomban kapható! (példa: 2:1 MUX)

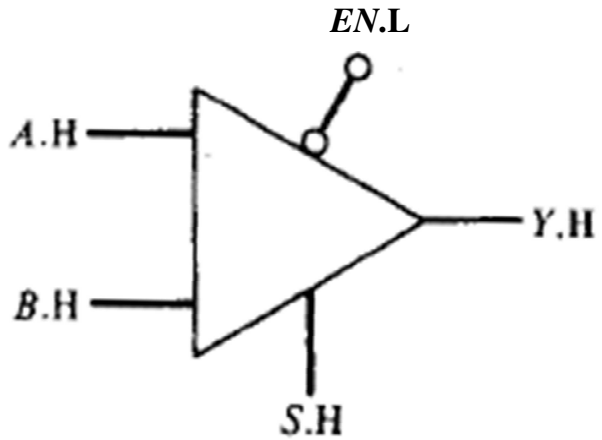
TTL'74LS157- 2:1 MUXok alkalmazása



- Quad (4 db) 2-bemenetű 2:1 MUX-ból áll.
- Közös S, EN jelek.
- 4 db Y(1,2,3,4) kimenet



2:1 MUX szimbóluma

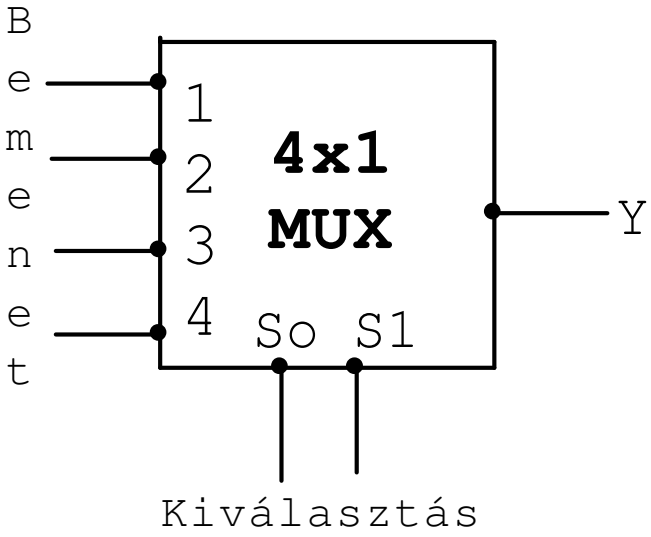


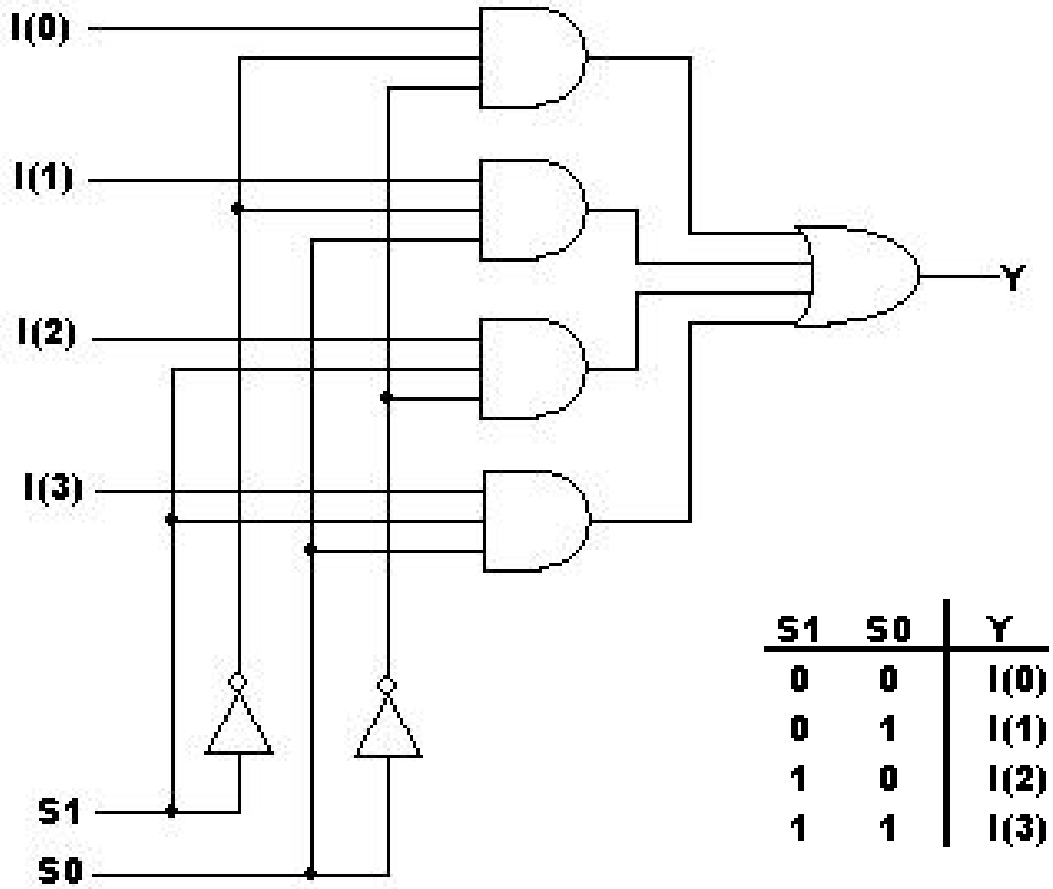
2:1 MUX áramköri kivezetések jelöléseivel

$$Y = EN \cdot (A \cdot \overline{S} + B \cdot S)$$

Példa: 4:1 Multiplexer

- N kiválasztó jel -> 2^N bemenet, 1 kimenet
- Példa: 4:1 MUX





S1	S0	Y
0	0	I(0)
0	1	I(1)
1	0	I(2)
1	1	I(3)

2^N számú bemenet közül választ egyet (Y), mint egy kapcsoló. Rendelkezhet EN bemenettel is.

Multiplexer: általános szabályok

- *2^n* bemenet esetén *n* db selector jelet kell definiálni (bináris kódban megadva)
- Pl. 4:1 Mux igazságtáblázata:
  - S0, S1: selector jelek
  - Input 1…4: 4 db bemenet

S1	S0	Kiválasztott input pozíció
0	0	Input 1
0	1	Input 2
1	0	Input 3
1	1	Input 4

- Kereskedelmi forgalomban 2-,4-,8-,16-, 32-bites MUX-ok is kaphatóak.

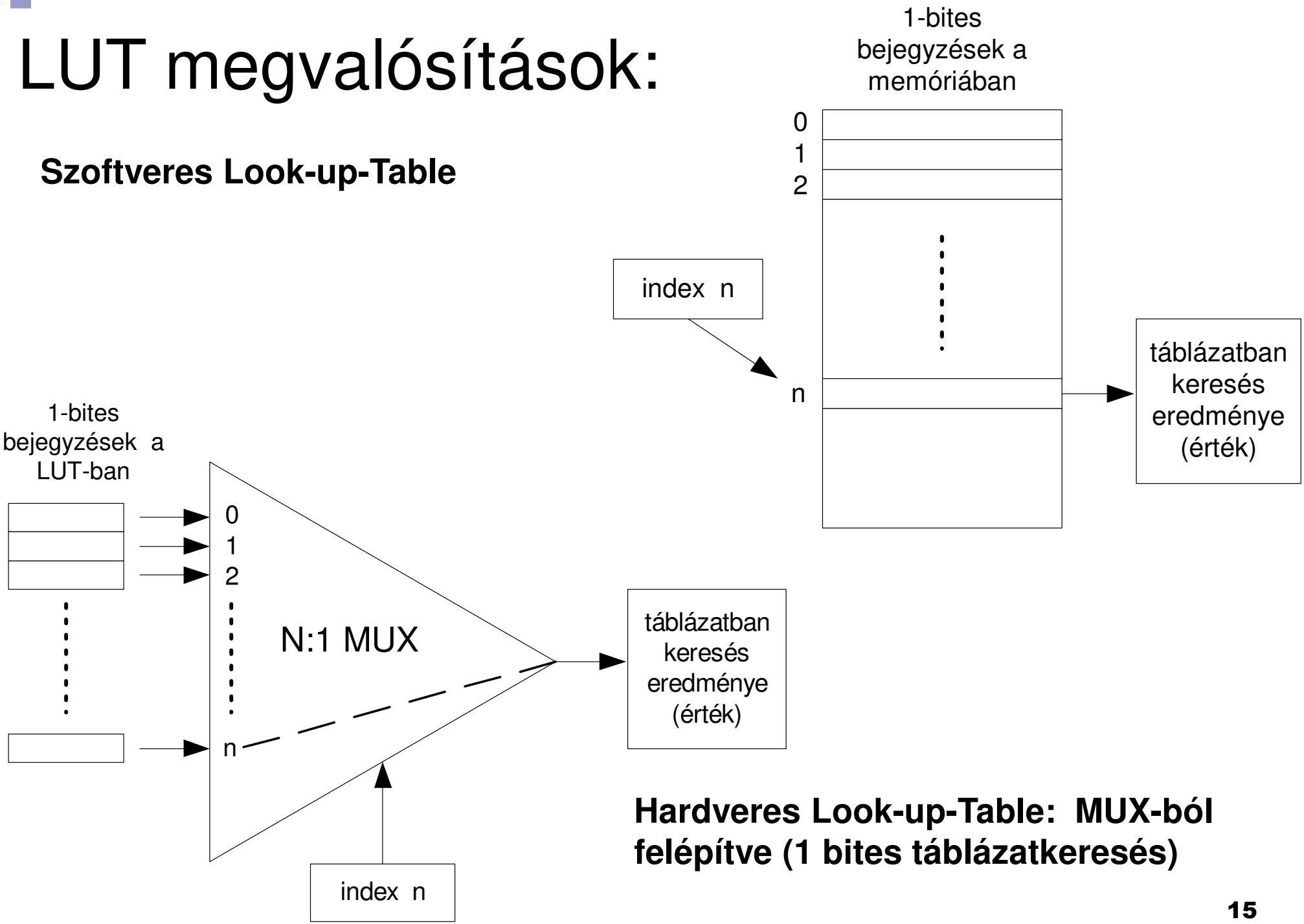
MUX: Look-Up-Table (LUT)

Táblázatban keresés – input kiválasztás

- MUX selector (S) jeleinek bináris kódjai, mint a bemenetetek *címzése* (*address, vagy index)* jelennek meg
- Az adatbemenetek (inputok) táblázatos vagy vektoros formában adottak.
- A MUX tehát egy 1-bites hardveres megvalósítása a „szoftveres” LUT táblázat kiolvasásnak.
- Fontos! Memóriákkal való műveletvégzés (*címzésük, indexelés* stb.) a MUX-ok segítségével történik, nem LUT-val.

LUT megvalósítások:

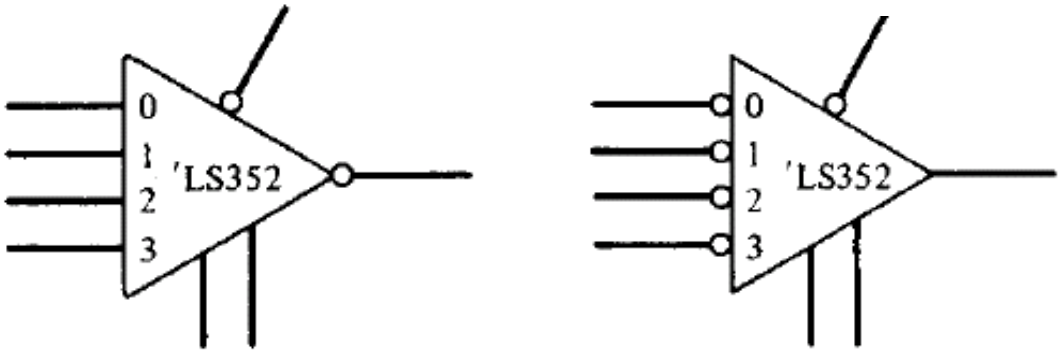
Szoftveres Look-up-Table



Hardveres Look-up-Table: MUX-ból felépítve (1 bites táblázatkeresés)

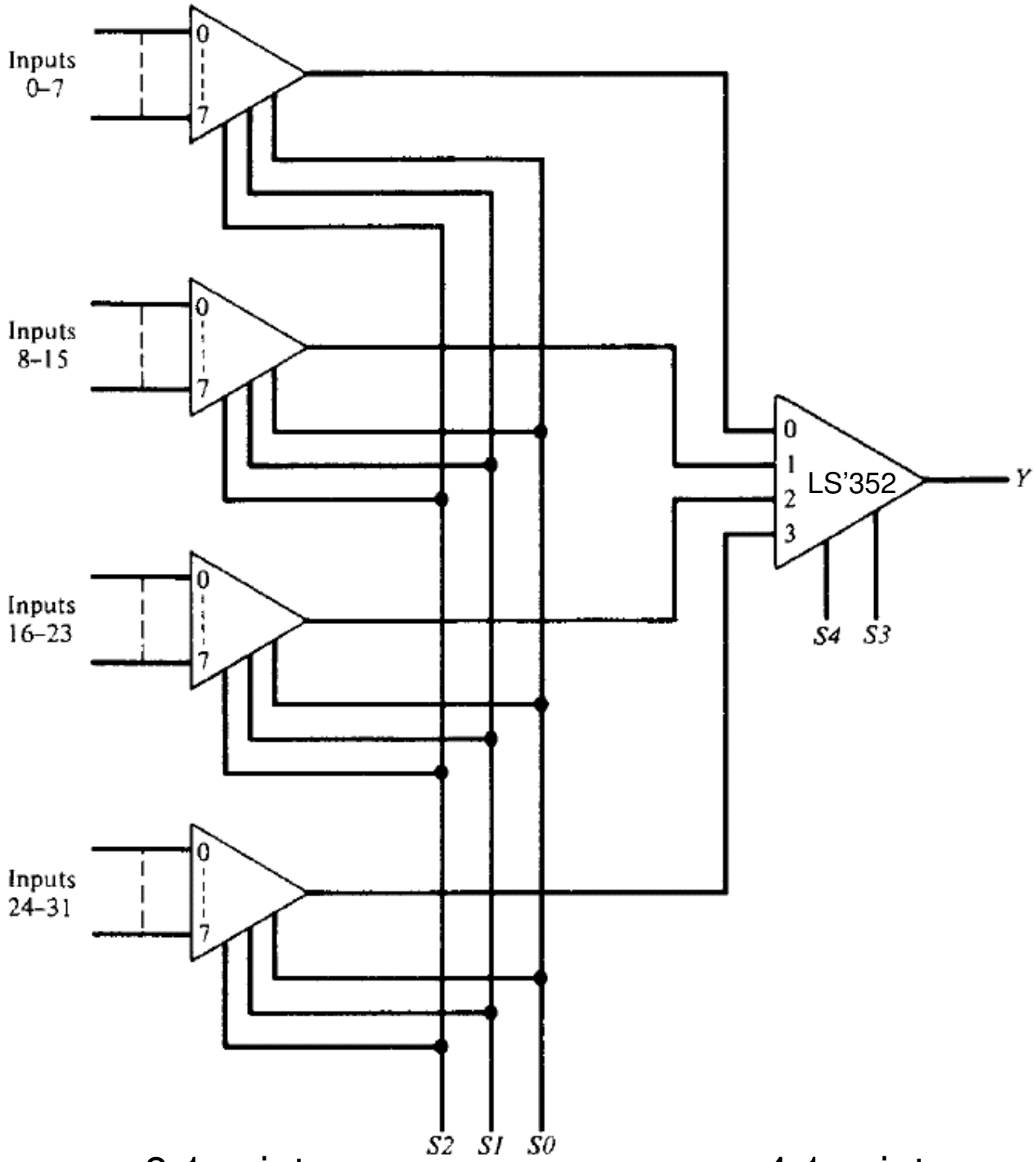
Példa: 32-elemű LUT

- 32 input bejegyzés -> 5 kiválasztó jel szükséges
  - 4 db 8:1 MUX csoportra osztva (kimeneten 4:1-es MUX)
  - 2 csoportja a selector jeleknek
    - Csoportok közül választás (S0,S1,S2)
    - Kiválasztott csoporton belüli választás (S3,S4)
- Selection index 5-biten (S4,S3,S2,S1,S0)
- Bemenetek magas aktív (H)
- Kimenetek magas / alacsony aktív áll. (H/L)
  - Fizikai eszköz: TTL 74LS352



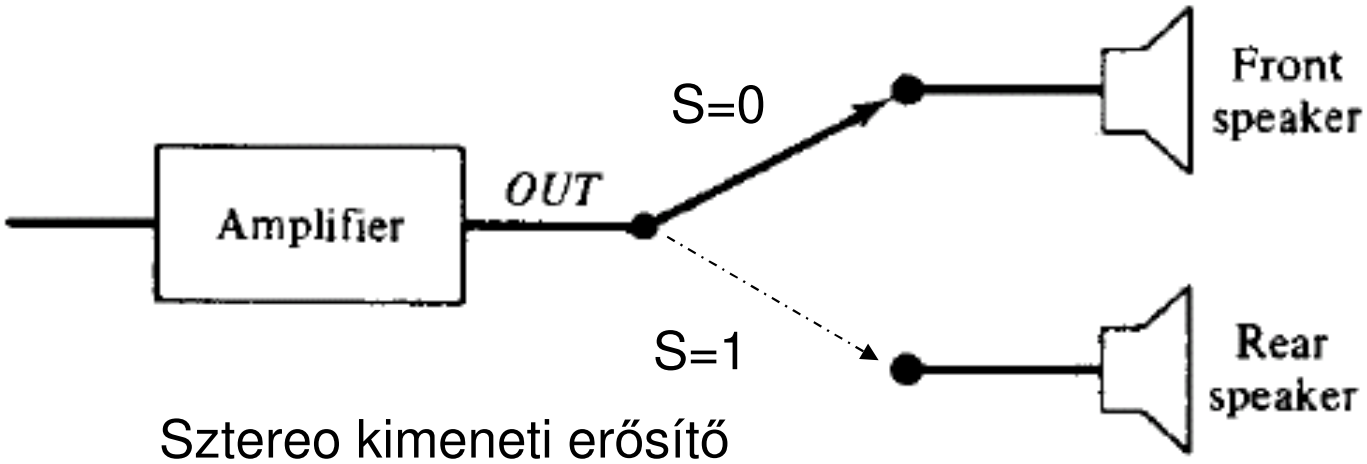
32-elemű LUT

- 32-bemenet
  - (4x8 csoport)
  - 8:1 MUX
- Kimeneten szinten
  - 4:1 MUX
- 5 szelektor jel
  - 3+2
  - (S0,…S4)



2. Demultiplexer (DEMUX)

- Olyan áramköri elem, amely több kimenet közül választ ki egyet, a *selector* jel(ek) hatására, és a kiválasztott kimenetet a bemenettel köti össze. (Multiplexerrel ellentétes funkciót tölt be.)
- Működése hasonló a **mechanikus elosztó (distributor) kapcsolókéhoz**. Például:



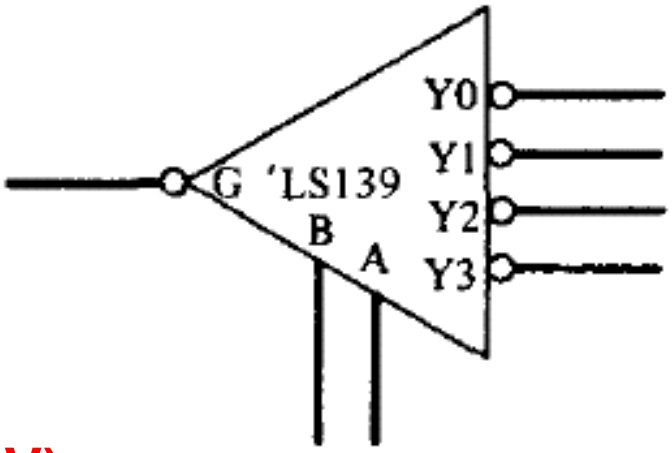
Sztereo kimeneti erősítő

$$\text{Front sp.} = OUT \cdot \overline{S}$$

$$\text{Rear sp.} = OUT \cdot S$$

Példa - 1:4 Demultiplexer

- TTL 74'LS139 duál 1:4 demultiplexer
  - Kereskedelmi forgalomban kapható
  - G: egyetlen demultiplexálandó bemenet
  - A,B: routing control/selector jelek (bináris kód)
  - 4-kimenet mindegyike False/Hamis, egyet kivéve, amelyik a kiválasztott (annak az értéke a bemenettől függően lehet T/F)



Mixed logika
T=L ! (~0V)

Példa - 1:4 Demultiplexer (folyt)

- Kanonikus táblázat

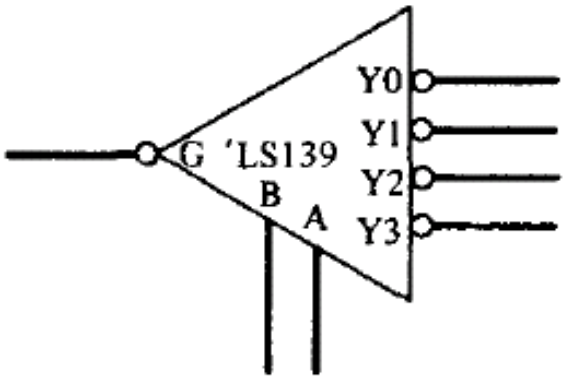
Demultiplexer logika						
G	B	A	Y0	Y1	Y2	Y3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

T=L !

Feszültség-logikai tábla

74'LS139 feszültség-logika						
G.L	B.H	A.H	Y0.L	Y1.L	Y2.L	Y3.L
H	x	x	H	H	H	H
L	H	H	L	H	H	H
L	H	L	H	L	H	H
L	L	H	H	H	L	H
L	L	L	H	H	H	L

- Demultiplexer logikai egyenletei:



$$Y0 = \overline{B} \cdot \overline{A} \cdot G$$

$$Y1 = \overline{B} \cdot A \cdot G$$

$$Y2 = B \cdot \overline{A} \cdot G$$

$$Y3 = B \cdot A \cdot G$$

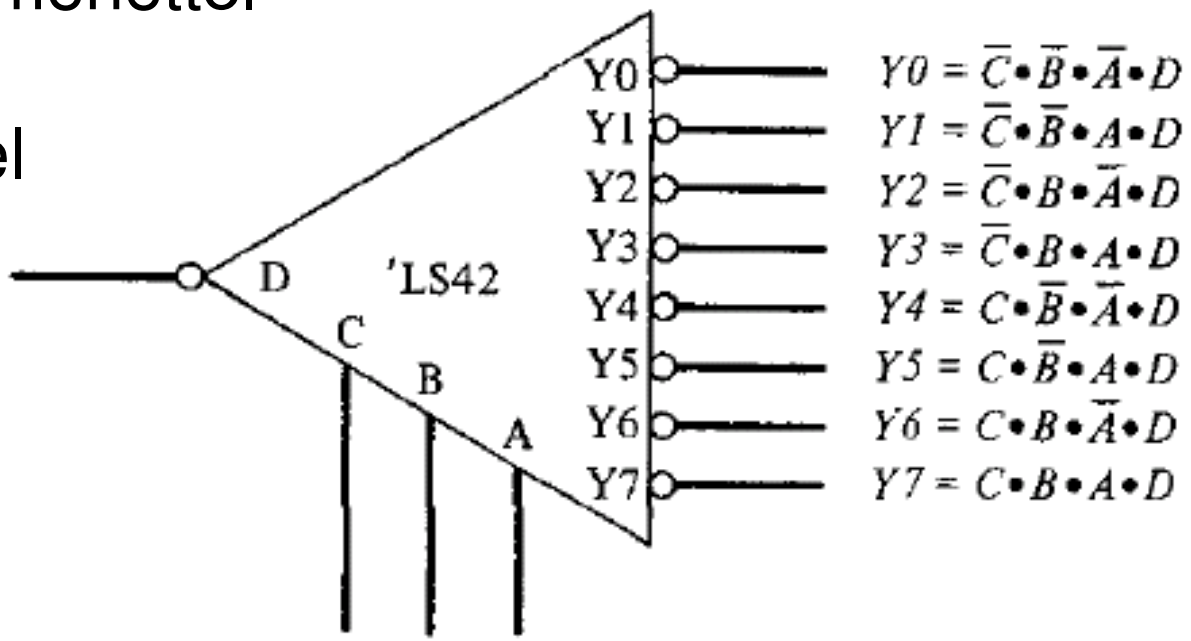
Példa 2. - 1:8 Demultiplexer

- TTL 74'LS42 - 1:8 demultiplexer (egyben „dekódoló")
  - Kereskedelmi forgalomban kapható
  - D: egyetlen demultiplexálandó bemenete
  - A,B,C: routing control jelek (bináris kód)
  - 8-kimenet (Y0,…Y7)mindegyike False, egyet kivéve, amelyik a kiválasztott (annak az értéke a bemenettől függően lehet T/F)
  - (További két kimenettel „dekódoló" áramkör készíthető belőle:)

$$Y8 = \overline{C} \cdot \overline{B} \cdot \overline{A} \cdot \overline{D}$$

$$Y9 = \overline{C} \cdot \overline{B} \cdot A \cdot \overline{D}$$

T=L !



3. Dekódoló (decoder) áramkör

- Kódolt információ dekódolása (konverzió)
  - Egyidőben, egyszerre csak egy logikai kimeneti változó (tehát a dekódolt) lehet igaz, a többi hamis!
  - 2^N kimenet dekódolásához N bemenet szükséges! Gyakran alkalmazott eszköz. Példák:
    - numerikus memória-cím dekódolásával azonosíthatunk pontosan egy adott memóriacellát!
    - opcode=műveleti kóddal azonosítunk egy kívánt funkciót (utasítást) pl. számítógép ISA (utasítás készlete)
  - Kapható tetszőleges 2-,3-,4-…több bemenetű IC formájában

Példa: DEC PDP-8 (egycímű gép) dekódoló logikája

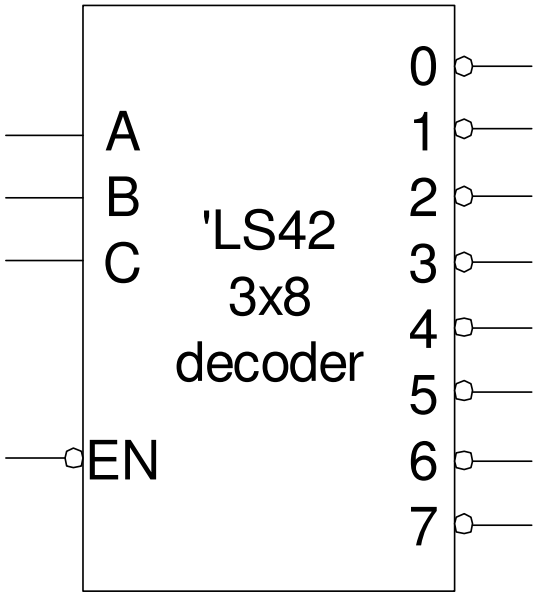
- 12-bites szóhossz: 3-bites opcode (8 művelet) + 9-bit utasítás cím (speciális operandus címzési módokat tett lehetővé)

Operation code	Bits C	B	A	Instruction	
0	0	0	0	AND	And,
1	0	0	1	TAD	ADD,
2	0	1	0	ISZ	IncSkipZero,
3	0	1	1	DCA	DepositClearAcc
4	1	0	0	JMS	
5	1	0	1	JMP	Jump
6	1	1	0	IOT	
7	1	1	1	OP	

Egyidőben csak egy utasítás teljesülhet!

TTL’74LS42 dekóder áramkör

- **3-8** dekóder áramkör (kibővíthető BCD-to-DECIMAL dekóderré ha 4 bemenet van):
  - (A,B,C) 3 bemenet, (1…7) 8 kimenet
- EN: engedélyező jel, (T=L) alacsony aktív

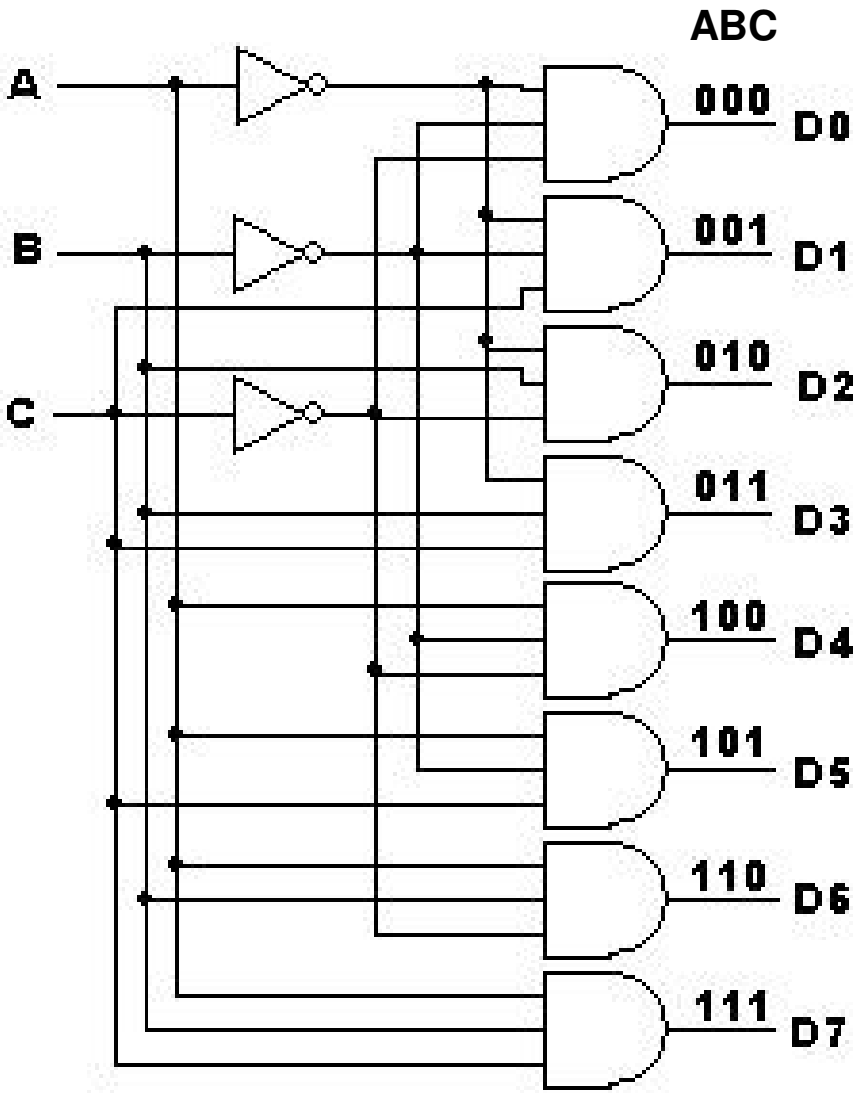


Mixed logic szimbólum

T=L!

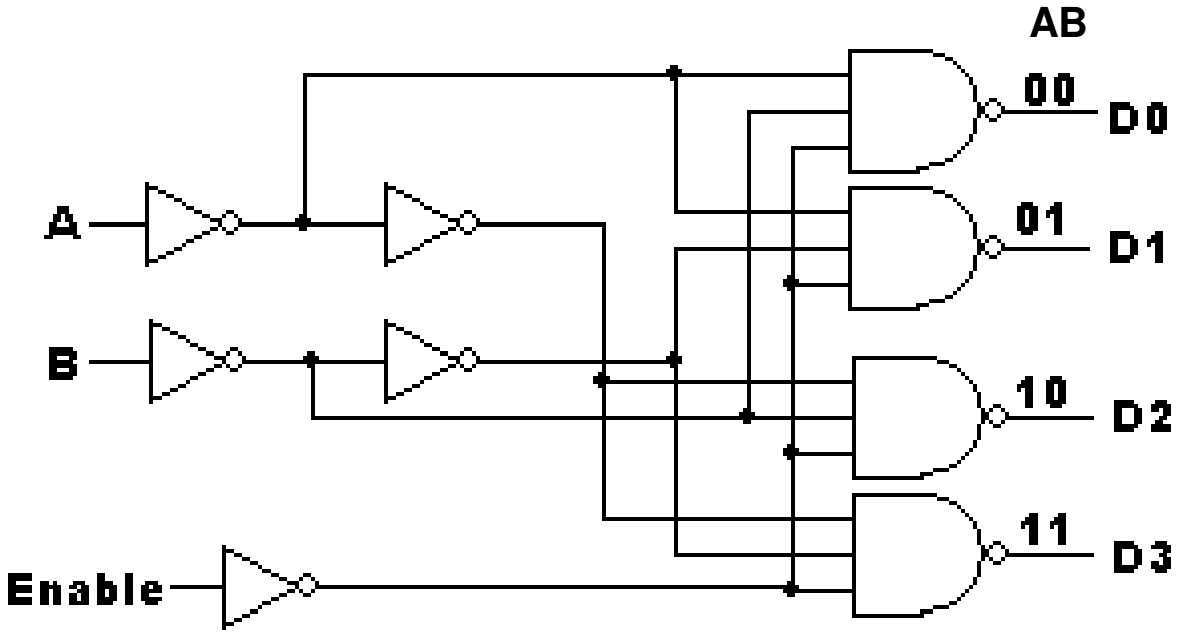
Példa: 3x8 Dekódoló áramkör kapcsolási rajza

- EN: engedélyező jel nélkül
- N bemenet esetén 2^N kimenete van
- Példa: 3x8 dekóder áramkör
  - Példa: Hamming-kódú hibajavító áramkör



Példa: 2x4 Dekódoló áramkör kapcsolási rajza (engedélyező bemenettel)

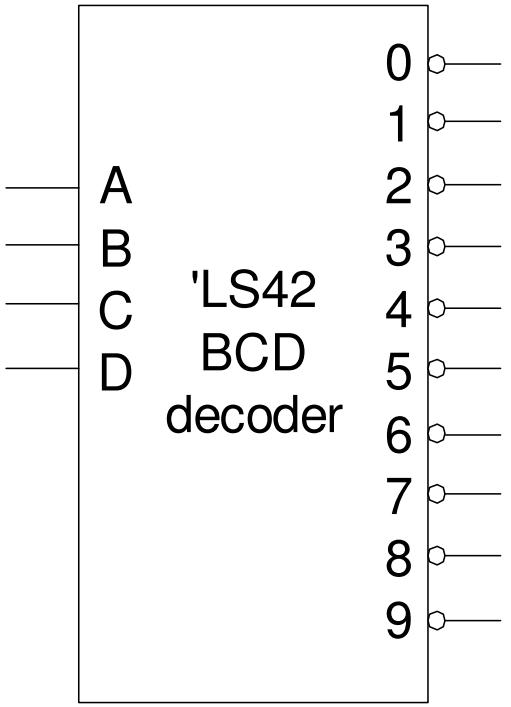
- EN: alacsony aktív állapotban működik
- 2 bemenő bit (A,B)
- 4 kimenő (dekódolt) bit (D0…D3)



En	A	B	D0	D1	D2	D3
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Példa: BCD -> Decimális dekóder ’74LS42 áramkör felhasználásával

- TTL ’74LS42-ból 4 BCD bemenet -> 10 kimenet
  - Decimális számjegyeket dekódolja (0..9)
  - //0xA = (10) ”1010” … 0xF = (15) ”1111” dont’care – nincs bekötve
    - Így csak 10 kimenete van
  - EN-lábat, mint a „D” legnagyobb helyiértékű bitjeként használjuk, False-nak definiáljuk!!



Mixed logic szimbólum

T=L!

4. Kódoló (encoder) áramkör

- A dekódoló áramkör ellentéte: bemenetek kódolt ábrázolásának egy formája
  - ***Hagyományos encoder***: csak egy bemenete lehet igaz egyszerre
  - ***Priority encoder***: több bemenete is igaz lehet egyszerre, de azok közül a *legnagyobb bináris értékű*, azaz *prioritású* bemenethez generál kódot! (kód: address, index lehet)
    - I/O, IRQ jelek generálásánál használják leggyakrabban

Probléma: Priority encoder esetén

- Mi van akkor, ha még sincs igaz bemenete (mindegyik hamis)? Két megoldás van:
  - *1.) módszer*: Input vonalak megszámozása 1-től (D1) kezdődően, és a 0 kimeneti kód (itt: B;A = F;F) jelenti, hogy mind „hamis” volt. (X – don’t care)
  - *2.) módszer*: input vonalak megszámozása 0-tól (D0) kezdődően, és egy külön vezérlőjelet (W) biztosítani arra, hogy nincs „igaz” bemenet. (X – don’t care)

Method 1

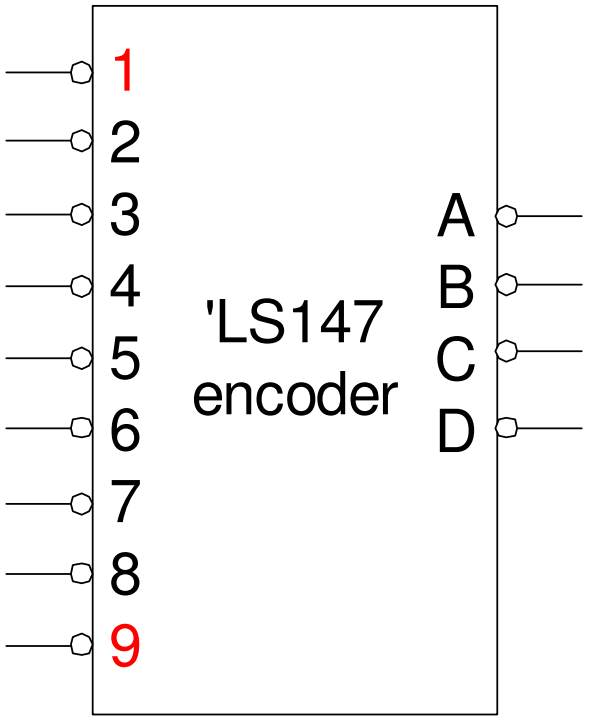
D3	*D2*	*D1*	*B*	*A*
F	F	F	F	F
F	F	T	F	T
F	T	X	T	F
T	X	X	T	T

Method 2

D3	*D2*	*D1*	*D0*	*B*	*A*	*W*
F	F	F	F	–	–	T
F	F	F	T	F	F	F
F	F	T	X	F	T	F
F	T	X	X	T	F	F
T	X	X	X	T	T	F

TTL’74LS147 Priority encoder - kódoló áramkör

- 10-input -> 4-output encoder
- (8 input -> 3 output is egyben)
  - „0” nincs jelölve: amikor az összes bemenet False (lefoglalt)
  - Alkalmazás:
    - Cím, indexgenerálás
    - LUT választás



Mixed logic szimbólum

T=L!

5. Komparátor

- Katalógus: TTL 74LS86
- Logikai kifejezés – *referencia* kifejezés (bináris számok) *aritmetikai kapcsolatának* megállapítására szolgáló eszköz.
  - Pl: Kettő n-bites szám összehasonlítása
- **compare** = **összehasonlítás**! Az azonosság eldöntéséhez a EQ/XNOR/Coincidence operátort használjuk. Jele: $A.EQ.B = A \odot B$
- n-bites minták esetén:

$$A.EQ.B = (A0 \odot B0) \cdot (A1 \odot B1) \cdot \ldots \cdot (An \odot Bn)$$

Ismétlés: EQ/XNOR/Coincidence operátor

- Logikai egyenlet: $A.EQ.B = A \odot B = A \cdot B + \overline{A} \cdot \overline{B}$
- Referenciabit szerinti megkülönböztetés:
  - ha a referencia bit (B), amihez hasonlítunk **konstans**
  - ha a referencia bit (B) egy **változó** mennyiség
- Példa: ha B referencia konstans -> egyszerűsítése A-nak

$$A.EQ.B = A \quad if \quad B = T$$

$$A.EQ.B = \overline{A} \quad if \quad B = F$$

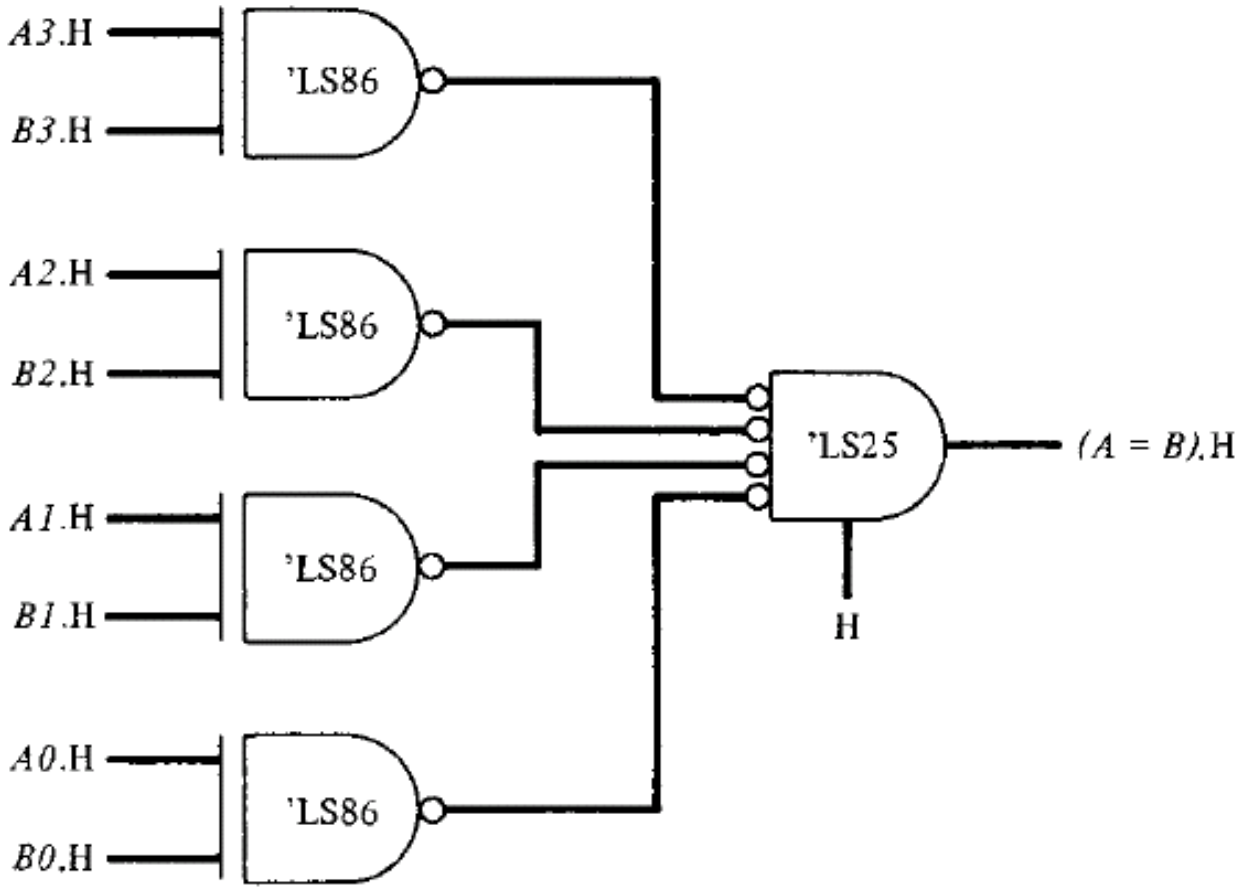
- Példa: legyen B egy 4-bites konstans mennyiség (B=TFFT), és A tetszőleges, akkor:

$$A.EQ.B = A0 \cdot \overline{A1} \cdot \overline{A2} \cdot A3$$

Példa: 4-bites komparátor

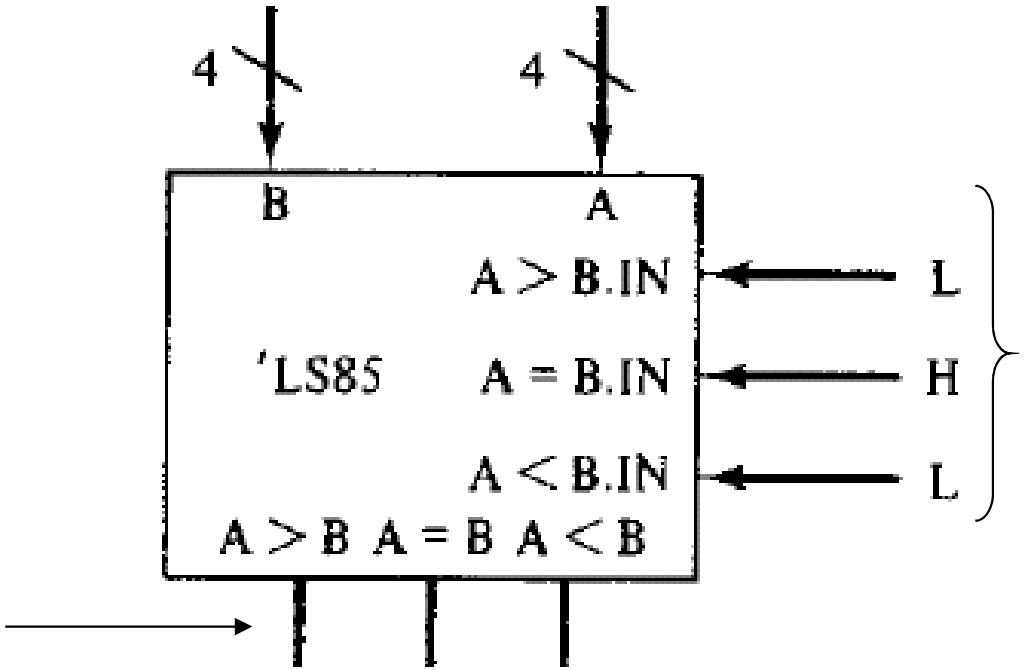
- Mixed-logic kapcsolási rajza, és log. egyenlete:

$$A.EQ.B = (A0 \odot B0) \cdot (A1 \odot B1) \cdot (A2 \odot B2) \cdot (A3 \odot B3)$$



’74LS85 4-bit Magnitude Comparator

- „Magnitude comparing” (~nagyságrend összehasonlítás):
  - két kifejezés **nagyságának** összehasonlítása (A<B; A=B; A>B stb.) egyszerre



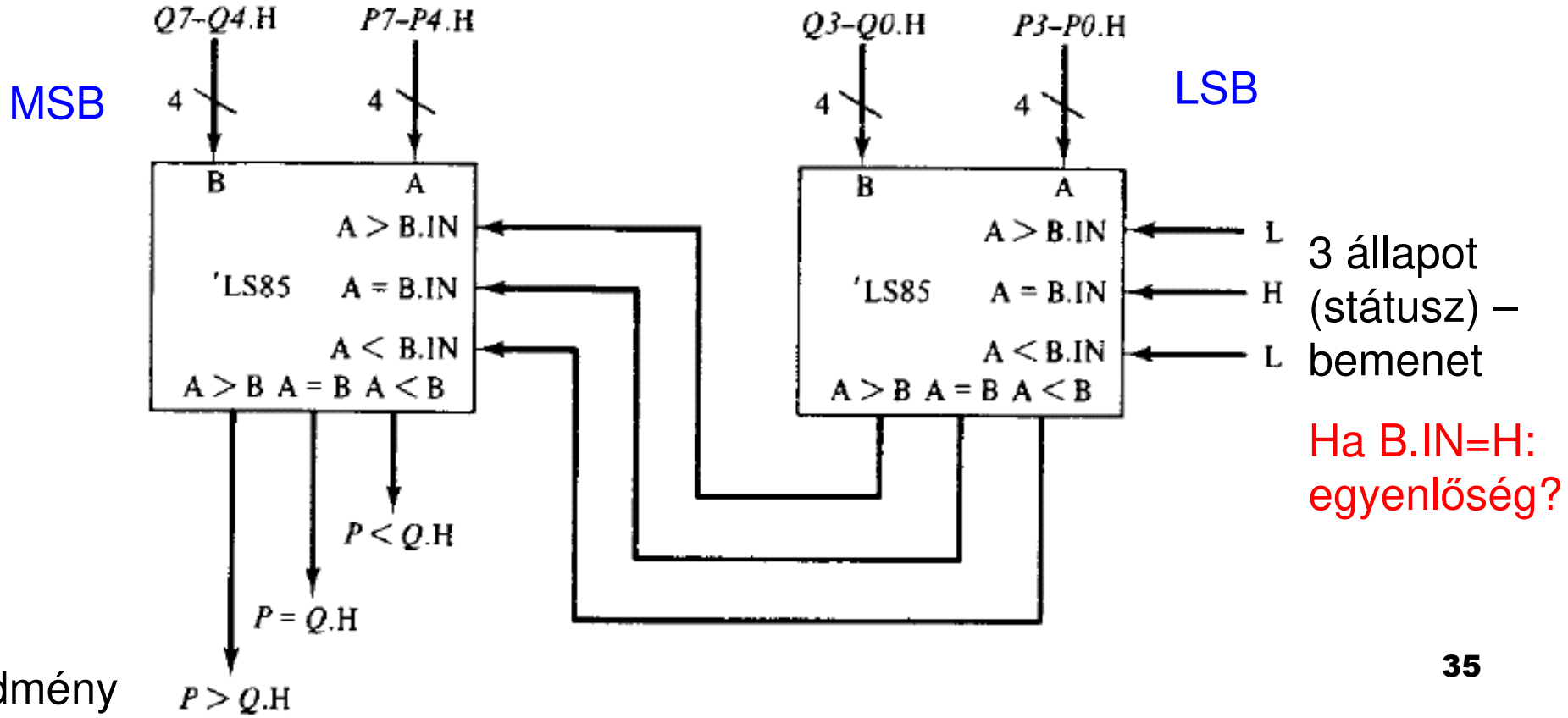
3 állapot (státusz) –bemenet

Ha B.IN=H: egyenlőség van A és B-re?

Összehasonlítás eredménye, mint kimenet

Példa: 8-bites Magnitude Comparator – egyenlőség esetén

- Kettő 4-bites ’74LS85 Magnitude komparátor sorbakötéséből („cascading”) kapjuk a 8-bites (P,Q) értékek összehasonlítását

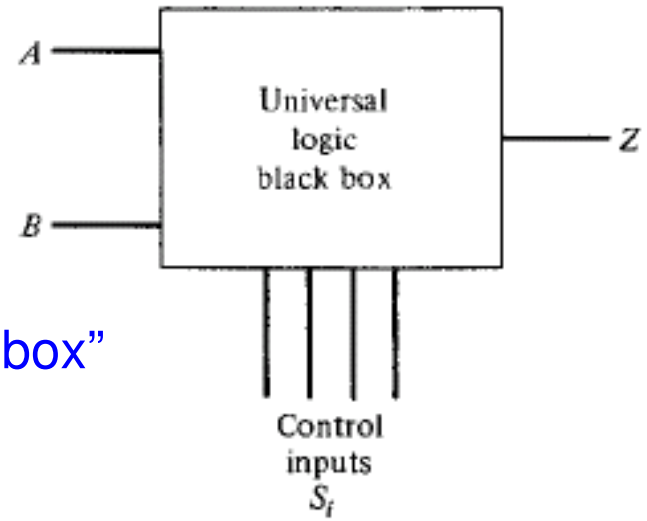


6. Univerzális logikai áramkör

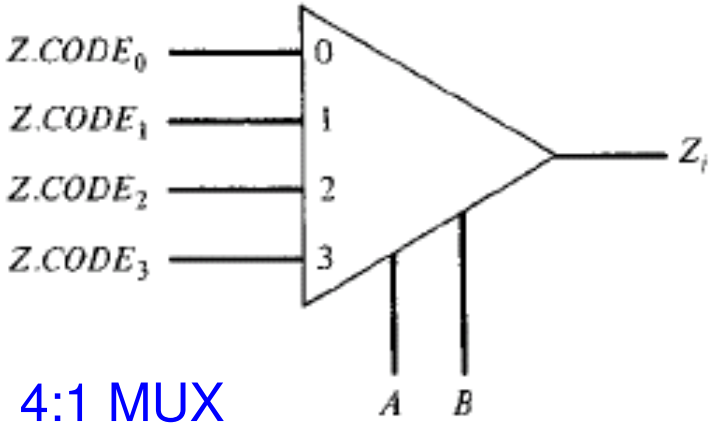
- Ritkán használt áramköri elem
- 2-bemenő bit (A,B) – 16-kimenet (Z0-Z15)

Lásd: Arató könyv
I. fejezet

A	B	Z 0	Z 1	Z 2	Z 3	Z 4	Z 5	Z 6	Z 7	Z 8	Z 9	Z 10	Z 11	Z 12	Z 13	Z 14	Z 15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1



„black box” modell



4:1 MUX

7. Bináris műveletvégző egységek

- a.) Hald Adder (HA) Fél összeadó → FA
- b.) Full Adder (FA) – Teljes összeadó
- c.) Ripple Carry Adder (RCA) – Átvitelkezelő összeadó
- d.) Look-Ahead-Carry Adder (LACA): átvitelgyorsító összeadó
- e.) Full Subtractor (FS) – Teljes kivonó
- f.) Arithmetic Logic Unit (ALU)

a.) Fél-összeadó – Half Adder

- 1-bites Half Adder

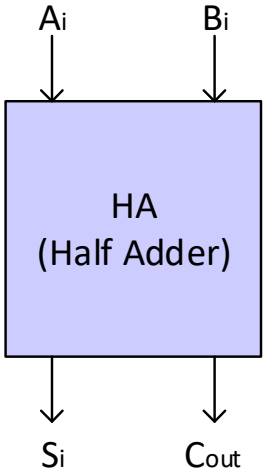
T_{HA} = 1G

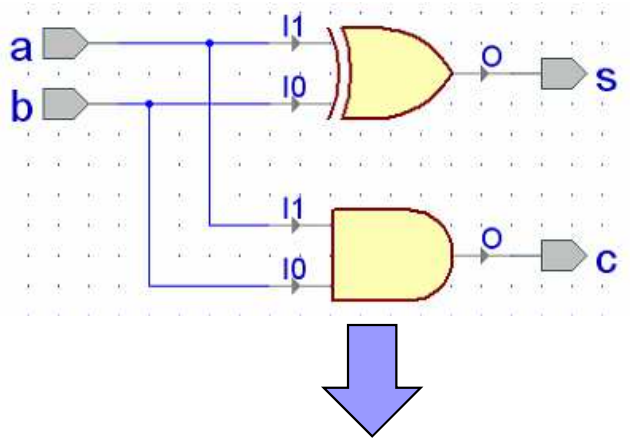
igazságtáblázat

Ai	Bi	Cout	Si
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

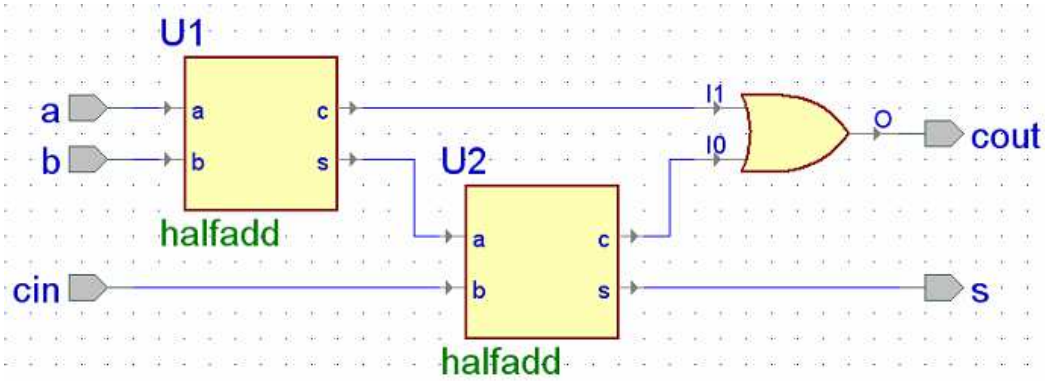
Nem kezeli a Cin-t !

szimbólum

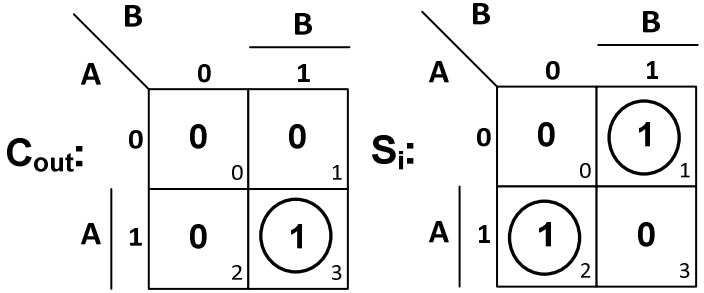




1 bites FA felépítése 2 db HA segítségével:



Karnaugh táblái:



Kimeneti fgv-ei:

$$C_{out} = A_i \cdot B_i \qquad S_i = A_i \oplus B_i$$

b.) Teljes összeadó – Full Adder

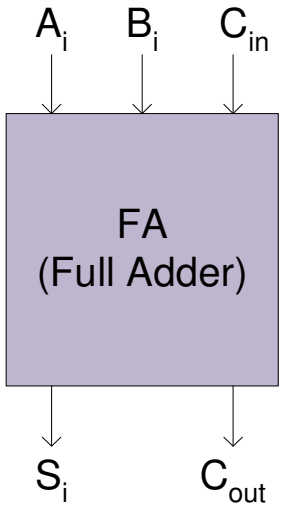
- FA: 1-bites Full Adder

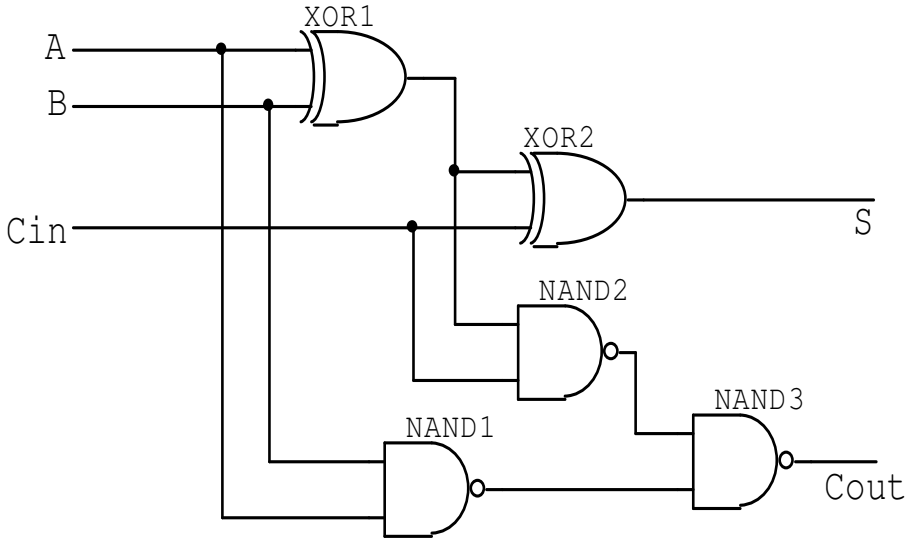
Ez a FA egy lehetséges CMOS kapcsolási rajza: (itt $T_{FA} = 3G$!)

igazságtáblázat

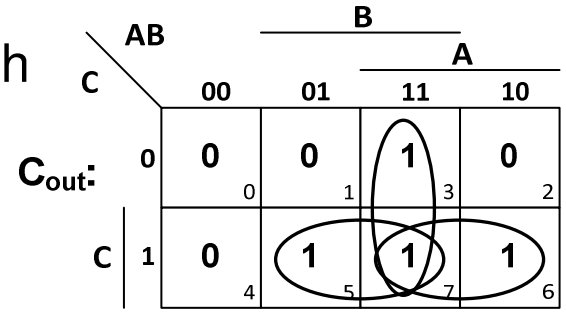
A_i	B_i	Cin	Cout	Sum_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

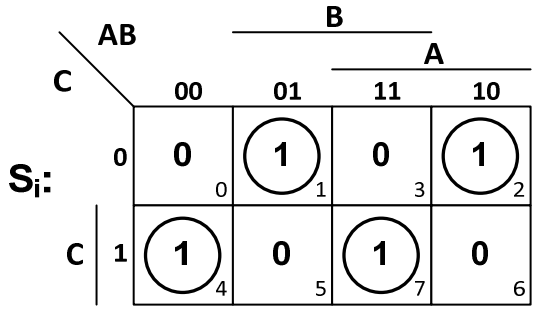
szimbólum





Karnaugh táblái:





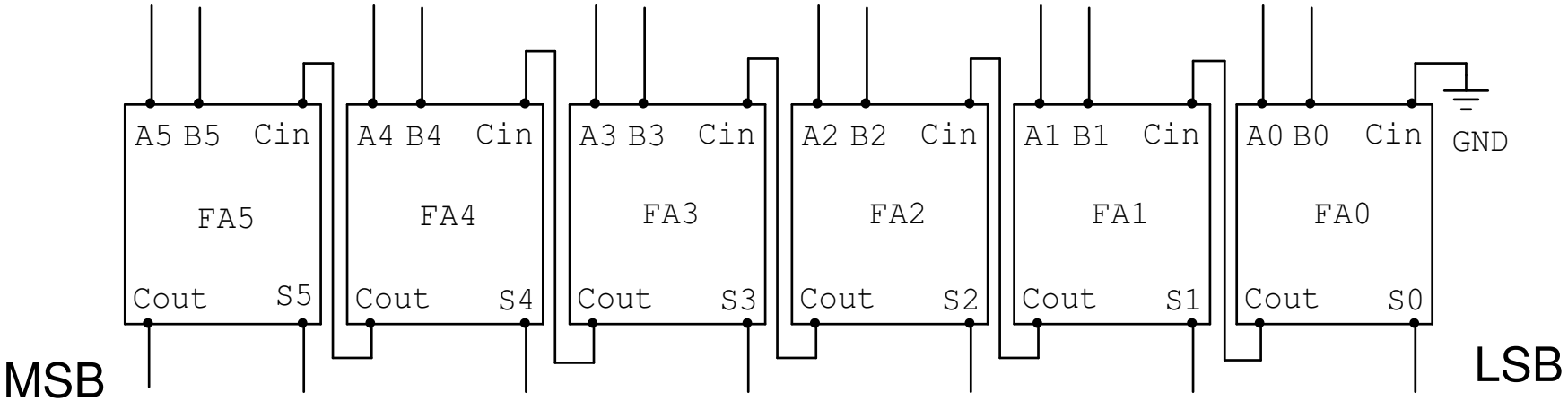
Kimeneti fgv-ei:

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

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

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

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

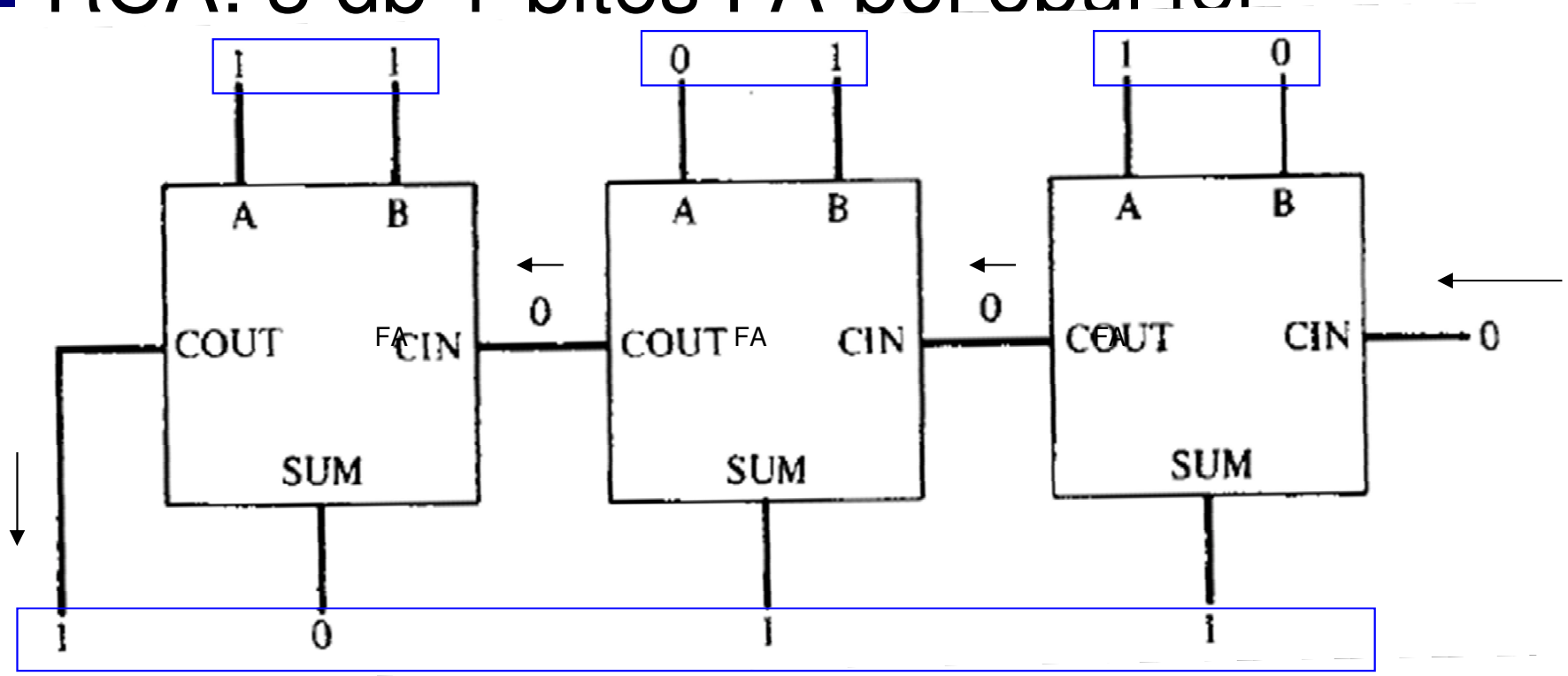


- Számítási időszükséglet (RCA):

$T_{(RCA)} = N*T_{(FA)min} = N*(2*G) = 12\ G$ (6-bites RCA esetén)
ahol a min. 2G az 1-bites FA kapukésleltetése ([ns], [ps])

Példa: 3-bites RCA áramkör

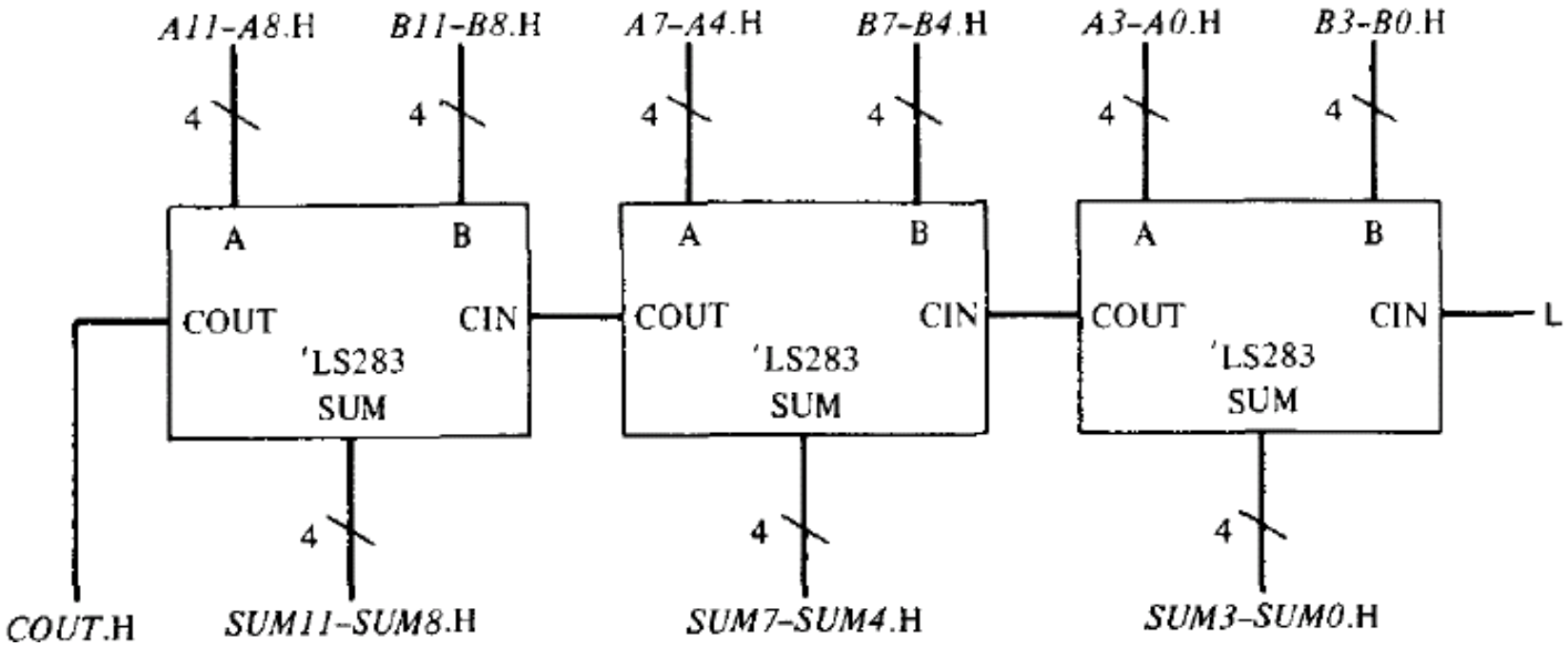
- RCA: 3 db 1-bites FA-ból épül fel



  - A+B=101+110=1011

Példa: 12-bites RCA áramkör

- 3 db 4-bites RCA-ból (’74LS283) épül fel



  - 2’s komplemens összeadó: előjeles aritmetika
    - 12 biten: 1-előjelbit + 11 bit (MSB: előjel)

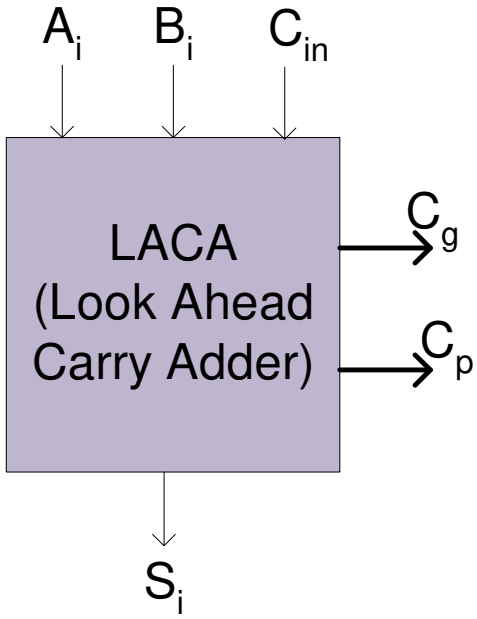
d.) LACA: Look Ahead Carry Adder:

- Képlet (FA) átalakításából kapjuk:

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

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

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



LACG: Look Ahead Carry Generator egy **b** bites ALU-hoz kapcsolódik, mindenegyes állapotban a Cin generálásáért felel a CP és CG (LACA-tól) érkező jeleknek megfelelően („*LACG looks at CP and CG 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)

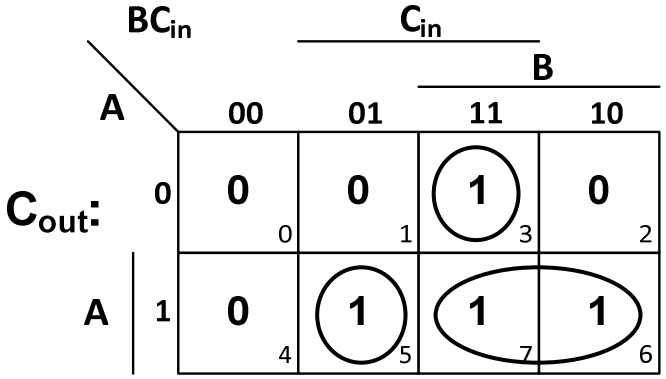
Megjegyzés: LACA – CG átírása XOR kapcsolatra (nem triviális forma)

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

igazságtáblázat

A_i	B_i	Cin	Sum_i	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Karnaugh tábla:



Kimeneti fgv:

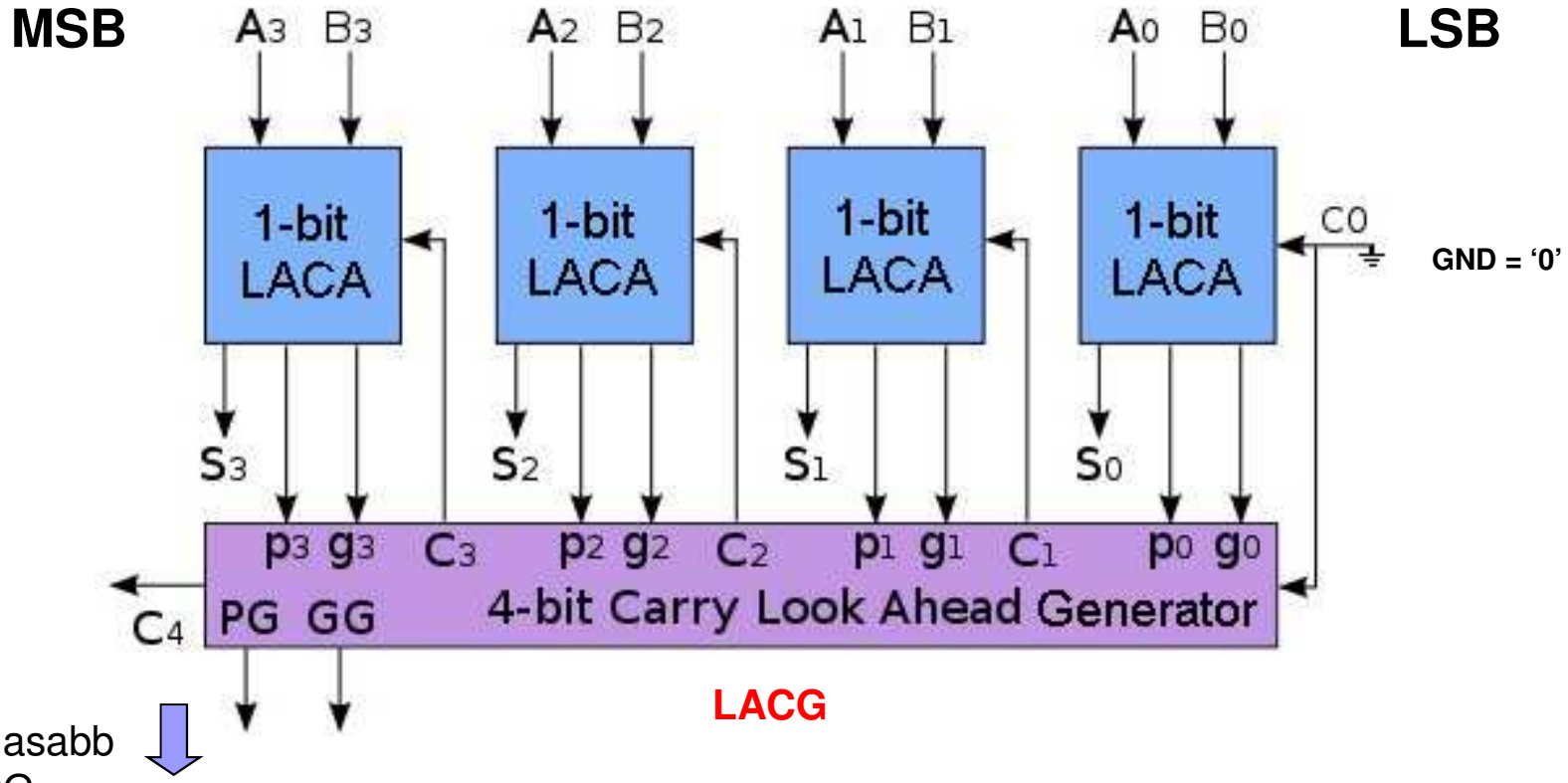
$$C_{out} = \overline{A_i} \cdot B_i \cdot C_{in} + A_i \cdot \overline{B_i} \cdot C_{in} + A_i \cdot B_i =$$

$$= A_i \cdot B_i + C_{in} \cdot (A_i \oplus B_i) =$$

$$= C_G + C_{in} \cdot C_P$$

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

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

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

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

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

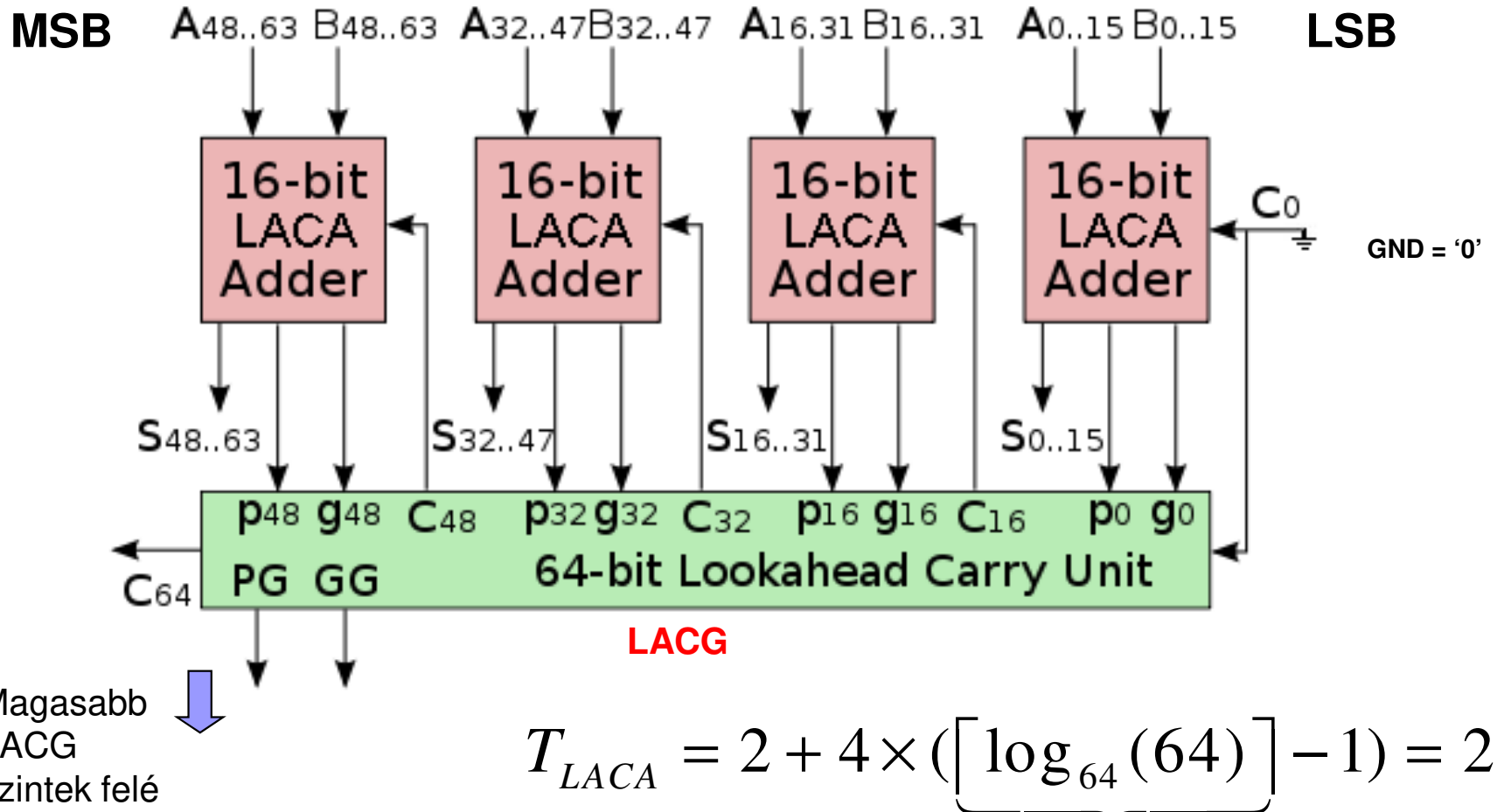
- Magasabb b-bites LACG hierarchia szintek felé GP (group propagate) és GG (group generate) számítása:

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

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

Példa: 4x16-bites LACA

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



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

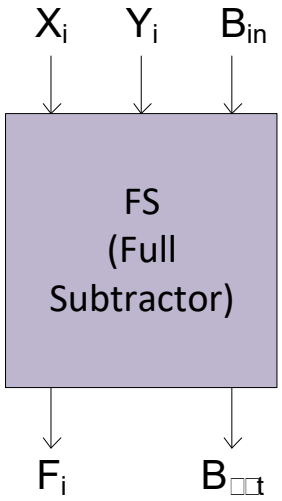
e.) Teljes kivonó - Full Subtractor (FS)

- FS: 1-bites Full Subtractor

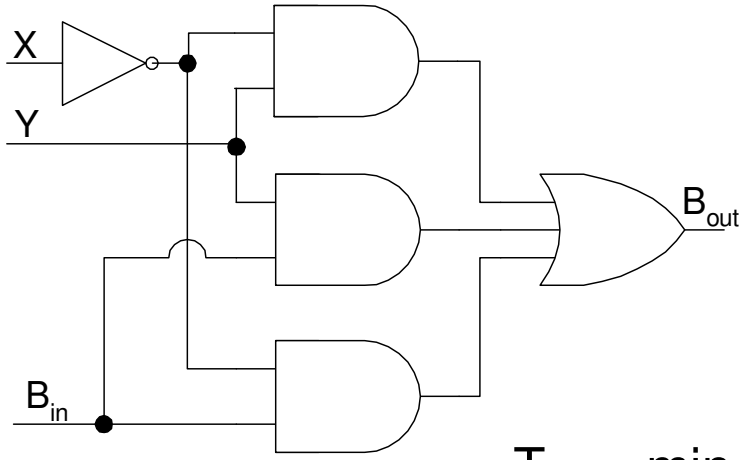
igazságtáblázat

X_i	Y_i	Bin	Bout	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):



T_{FS} = min 2G

Karnaugh táblái:

B_{out}:

X \ Y Bin	00	01	11	10
0	0 (0)	1 (1)	1 (3)	1 (2)
1	0 (4)	0 (5)	1 (7)	0 (6)

F_i:

X \ Y Bin	00	01	11	10
0	0 (0)	1 (1)	0 (3)	1 (2)
1	1 (4)	0 (5)	1 (7)	0 (6)

Kimeneti fgv-ei:

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

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

Bináris kivonás végrehajtása (folyt)

- **I. módszer:**
  Bináris kivonás *FS* segítségével
  - $X_i - Y_i \rightarrow F_i$
  - $0 - 0 \rightarrow 0$
  - $0 - 1 \rightarrow 1$, borrow bit '1'
  - $1 - 0 \rightarrow 1$
  - $1 - 1 \rightarrow 0$

```
******          *
1000000000      256
-001001100    - 76
----------    -----
 010110100      180
```

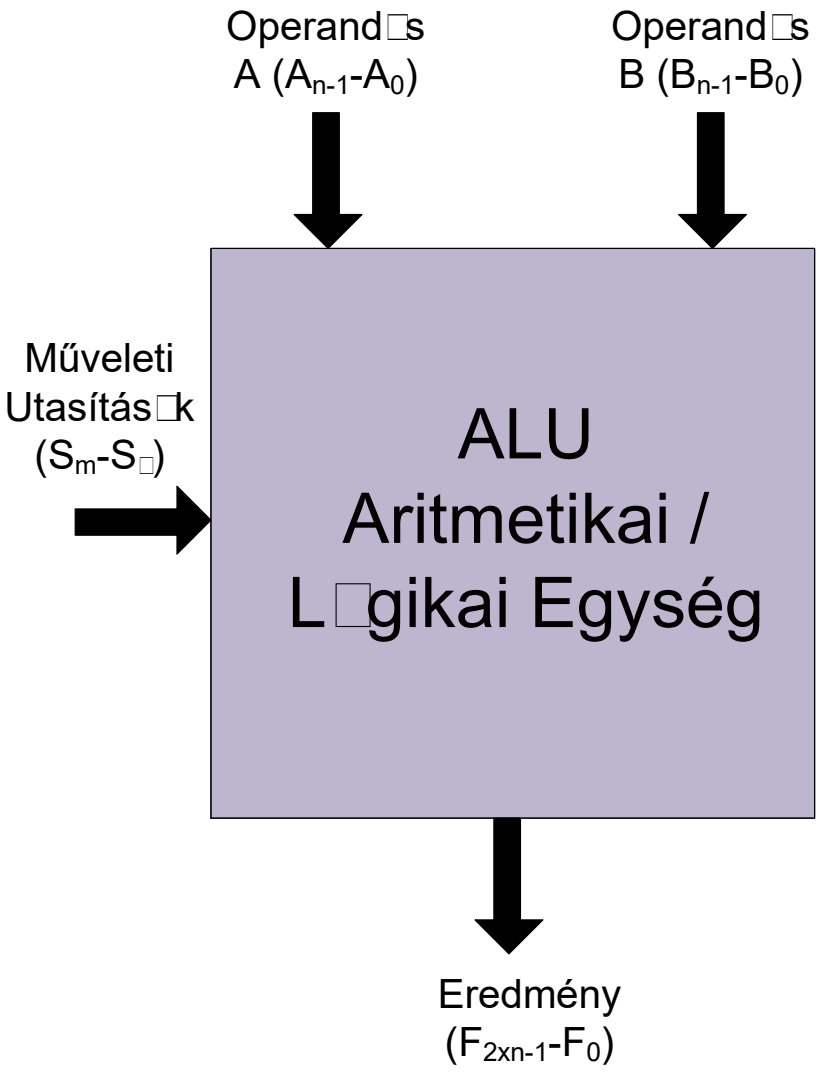
*: azt jelöli, amikor az adott helyiértéken '1'-et kell kivonni még az X_i értékéből (borrow from X_i)

- **II. módszer:** Kivonás visszavezetése az *univerzálisan teljes* bináris *összeadás* segítségével (2's komplemens alak):
  - FA, RCA, vagy LACA

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

f.) 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.

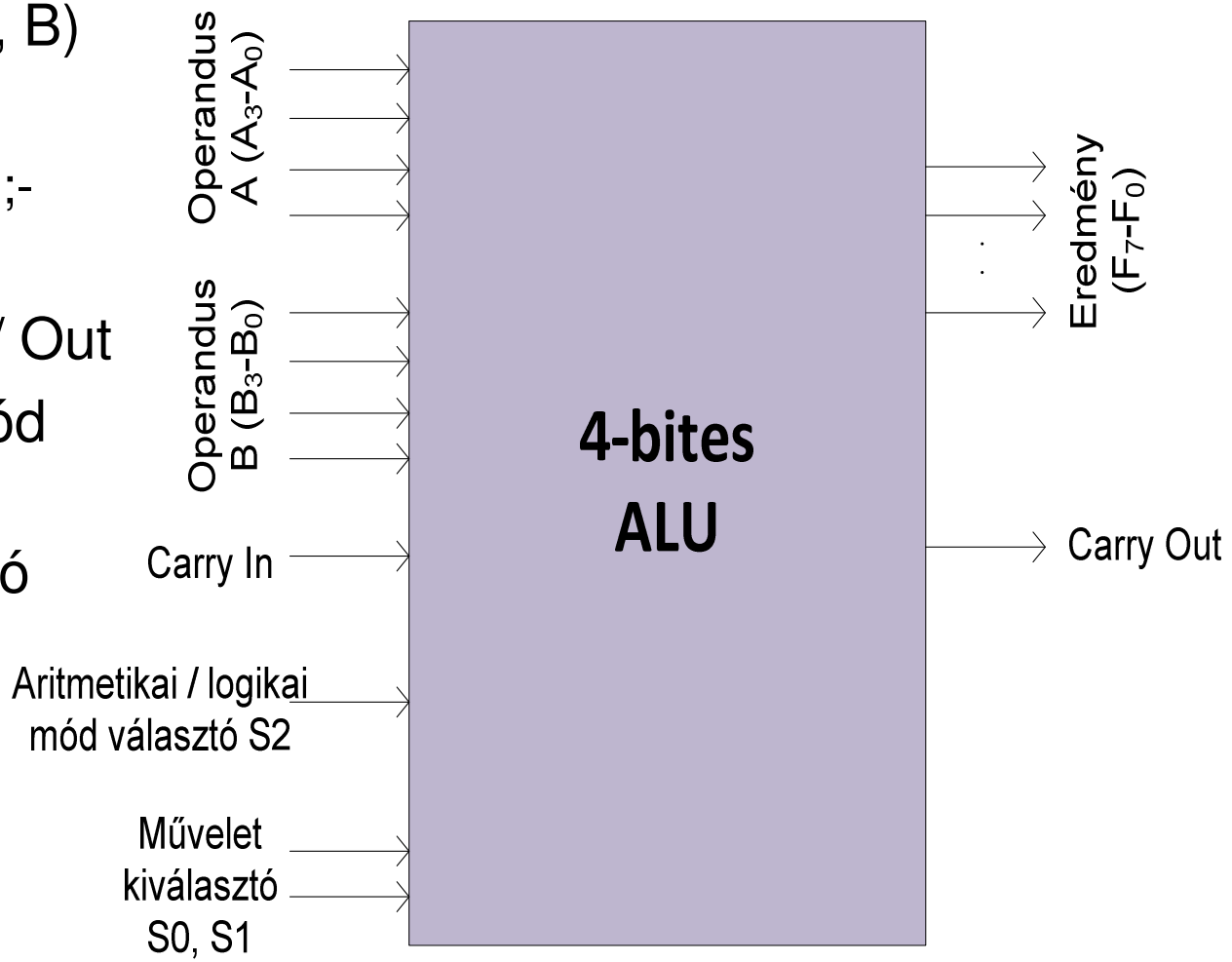


Státusz- (flag) jelzőbitek

- Az aritmetikai műveletek eredményétől függően hibajelzésre használatos jelzőbitek. Ezek megváltozása az utasításkészletben előre definiált utasítások végrehajtásától függ.
  - a.) Előjelbit (sign): 2's 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

Pl: 4-bites ALU felépítése és működése

- Két 4-bites operandus (A, B)
- Eredmény (F)!
  - N+(1 CarryOut) bit, ha +;-
  - 2×N bites, ha *; /
- H.értékes átvitel: CarryIn/ Out
- S2: Aritmetikai/ logikai mód választó (MUX)
- S0, S1: művelet kiválasztó (*S2* értékétől függően
  - Aritmetikai vagy Logikai



4-bites ALU szimbolikus rajza

IC: ‘181 ALU 4-bites (74LS181)

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 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)

* L.Howard Pollard: Computer Desing and Architecture c. könyv függeléke (appendix)

ALU felépítése