Computer Science & Engineering Department

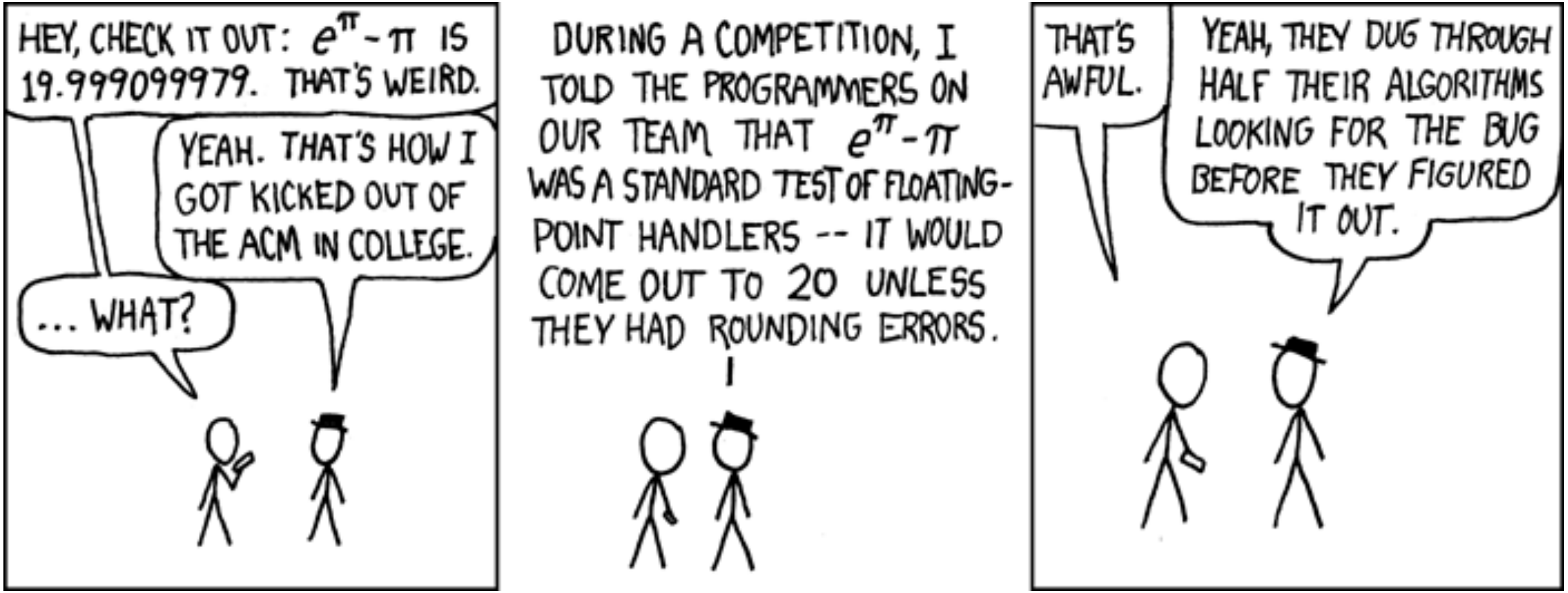
Calculatoare Numerice

– Cursul 3 –

Operații aritmetice

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

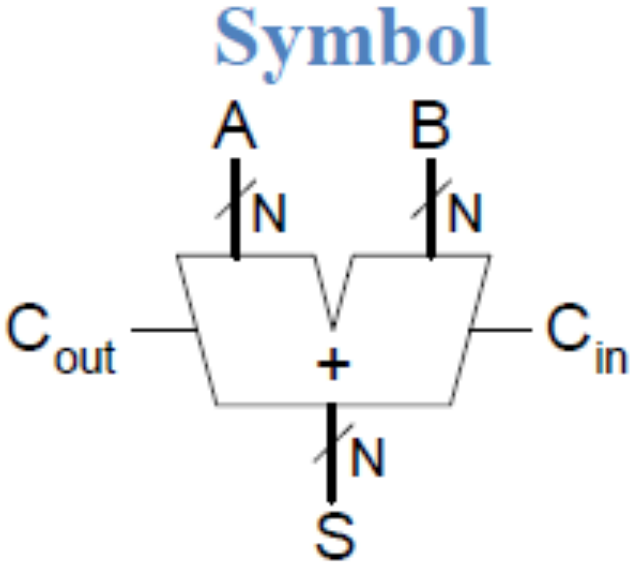
Comic of the day



http://xkcd.com/217/

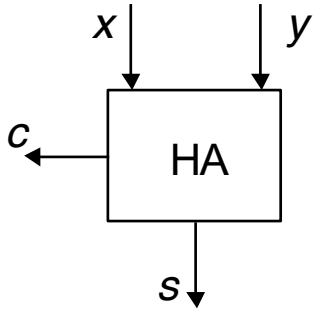
Sumatorul binar

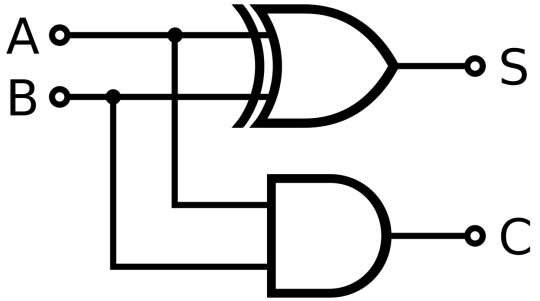
- Tipuri de sumatoare cu propagare de carry
  - Ripple-carry (lent)
  - Carry-lookahead (rapid)
  - Prefix (și mai rapid)
- Sumatoarele carry-lookahead și cu prefix sunt mai rapide pentru numere pe mai mulți biți dar necesită mai mult hardware



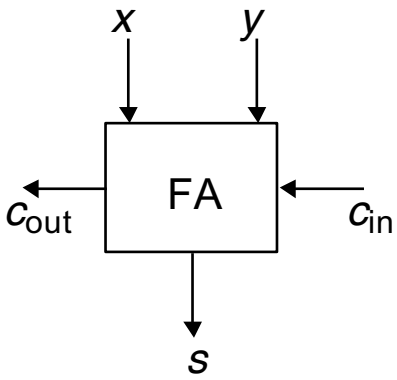
Binary half-adder (HA) & full-adder (FA)

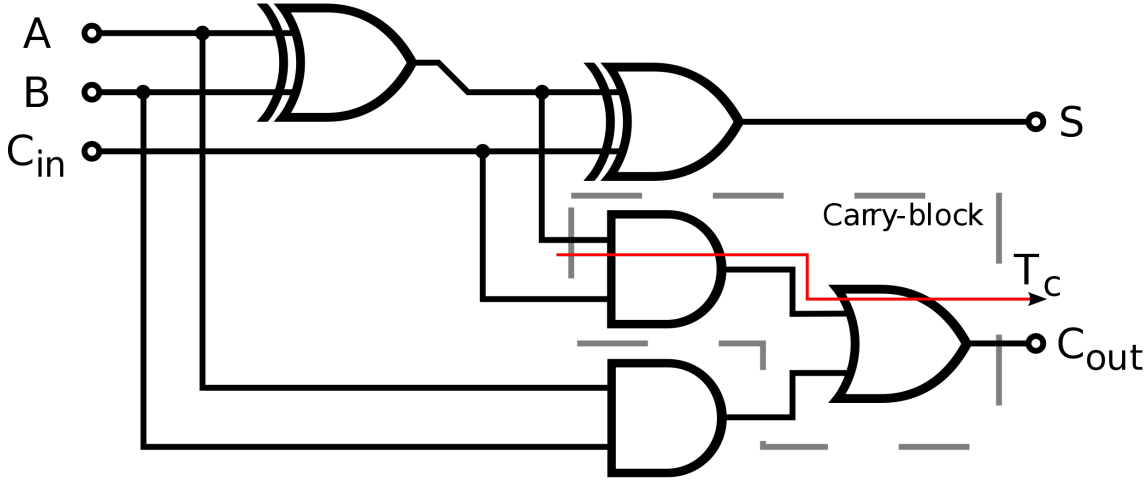
Inputs		Outputs	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



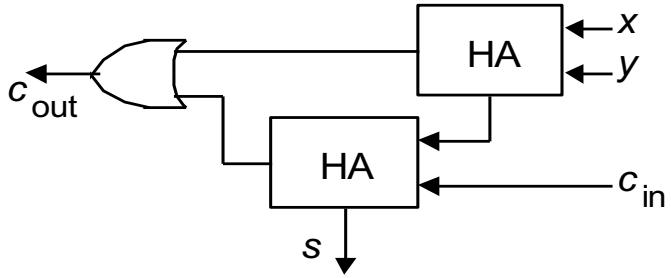


Inputs			Outputs	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

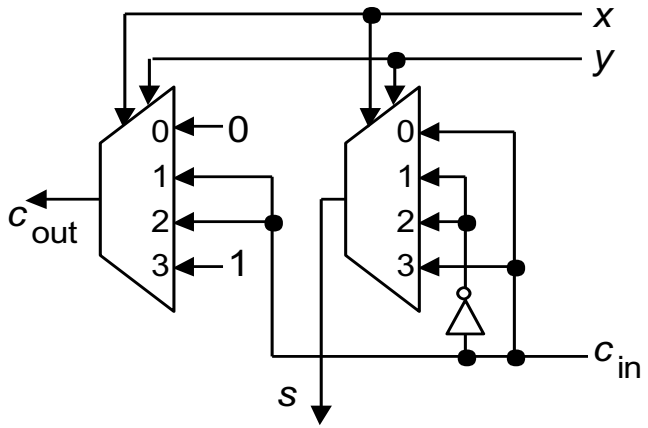




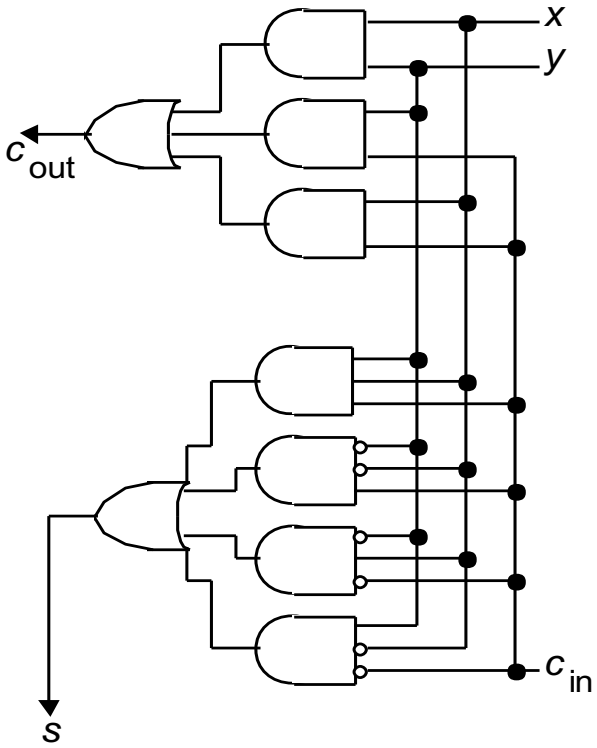
Implementarea unui sumator complet



(a) FA built of two HAs



(b) CMOS mux-based FA

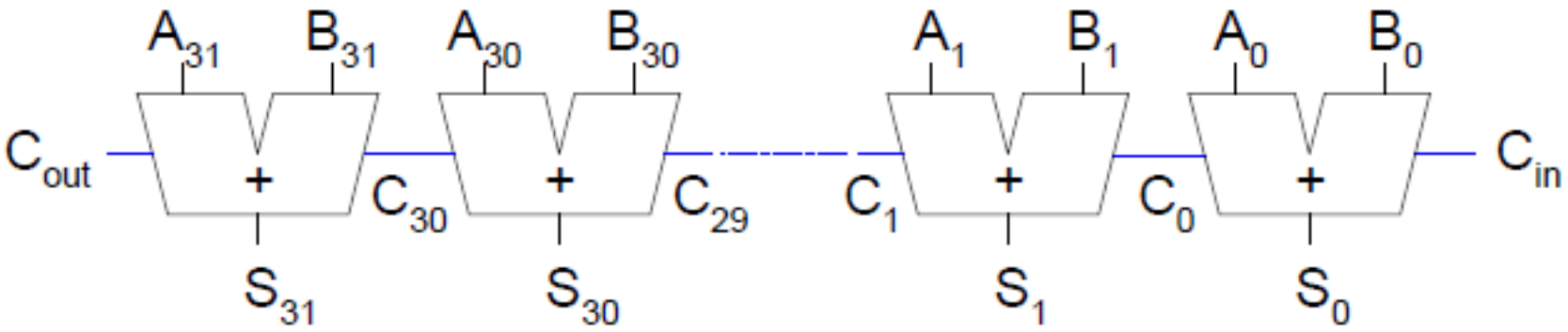


(c) Two-level AND-OR FA

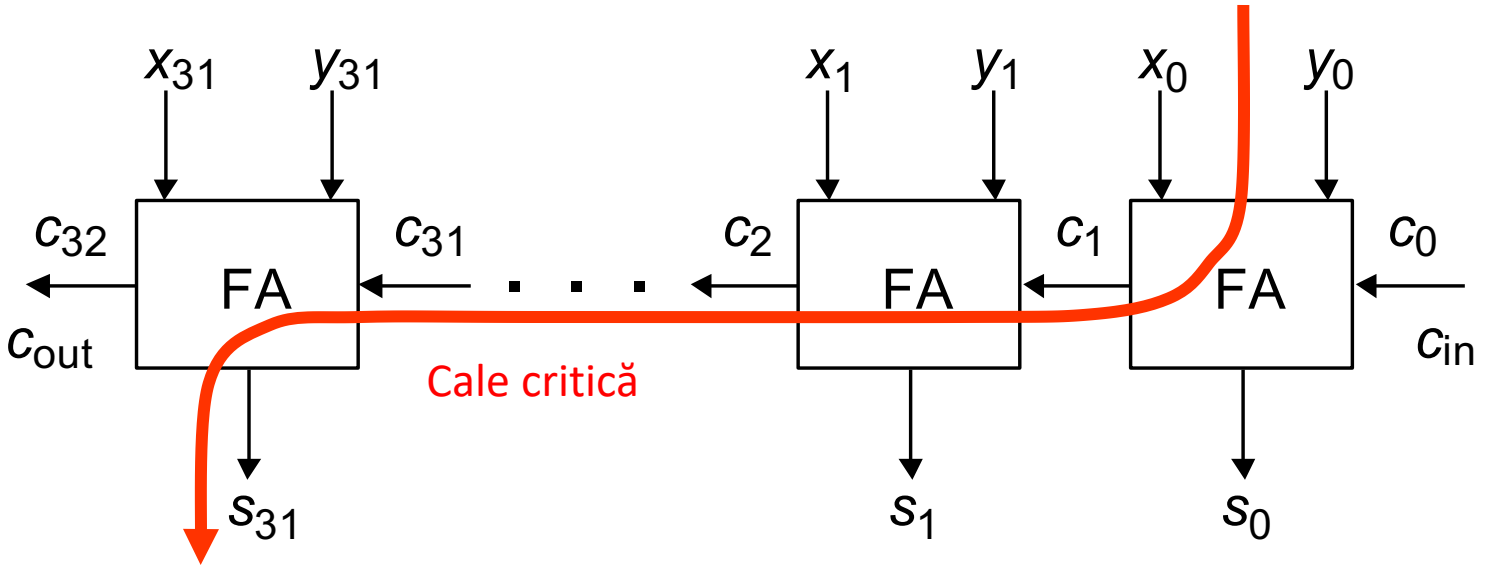
Sumator complet implementat folosind două sumatoare tip half-adder, cu ajutorul a două multiplexoare cu 4 intrări sau un circuit cu porți pe două niveluri.

Sumator Ripple-Carry

- Înlănțuim mai multe sumatoare de un bit
- Semnalul de carry se propagă prin întregul lanț
- Dezavantaj: **lent**



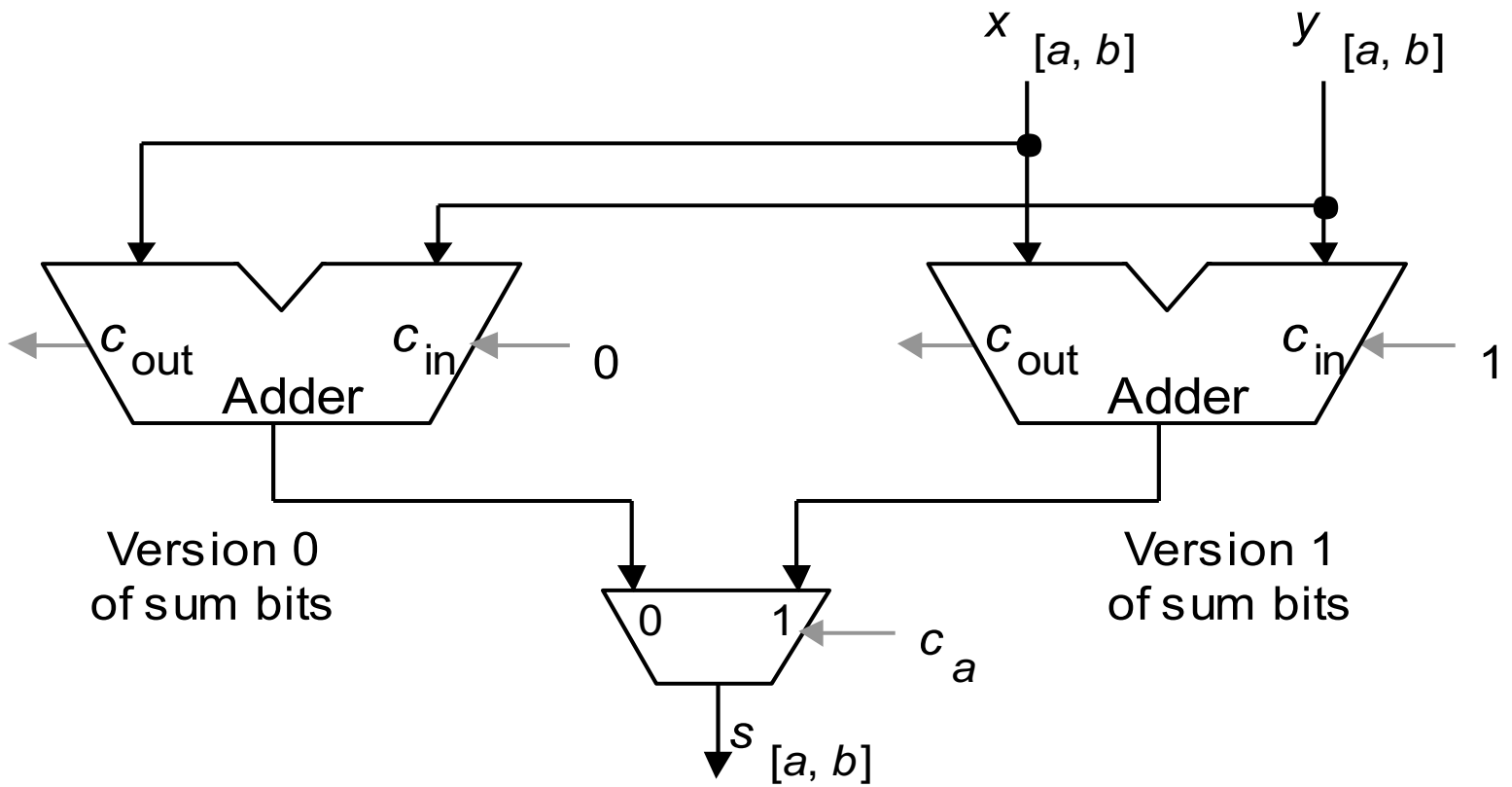
Sumator în riplu: foarte simplu dar lent



Sumator ripple-carry pentru numere de 32 de biți.

Sumator Carry Select

Presupune dublarea sumatorului inițial și introduce o întârziere suplimentară printr-un singur MUX



Principiul de funcționare al sumatorului Carry-select

Rangurile inferioare ale lui a, (0 la $a-1$) sunt sumate normal

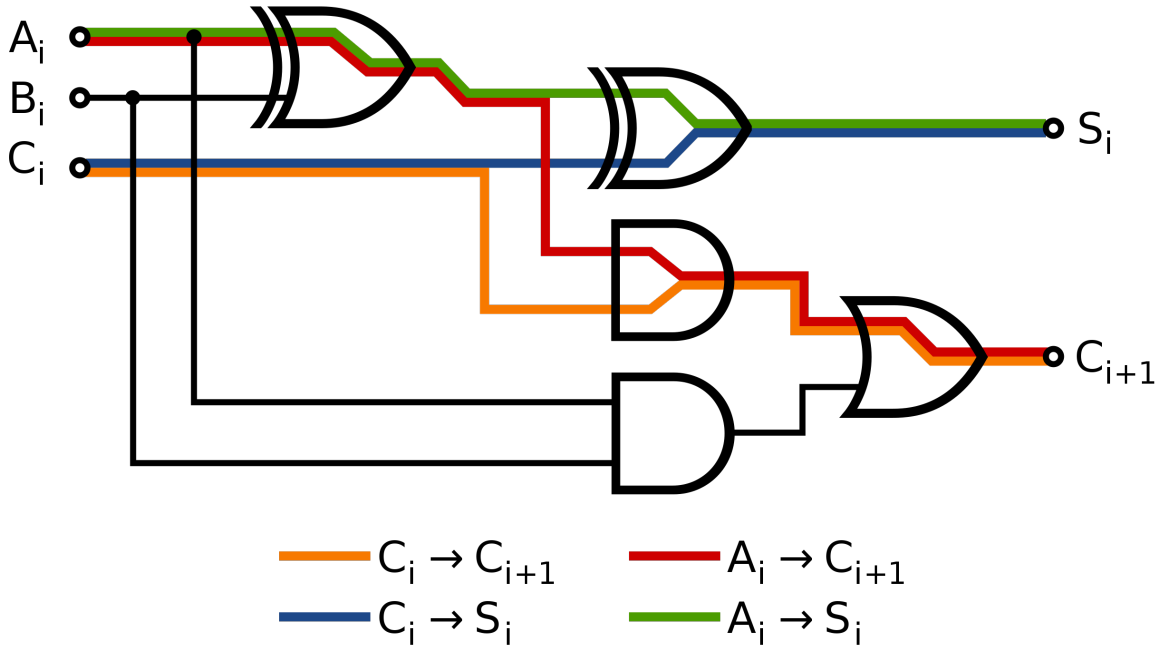
Întârzierea prin ripple-carry

Full-adder propagation delay:

- AND & OR au întârzieri similare (1-gate delay)
- XOR are întârziere dublă (2-gate delay) pentru că este facută din o combinație de porți AND și OR

Un sumator complet are următoarele întârzieri de propagare:

- De la A_i sau B_i la C_{i+1}: 4 gate-delays (XOR → AND → OR)
- De la A_i sau B_i to S_i: 4 gate-delays (XOR → XOR)
- De la C_i la C_{i+1}: 2 gate-delays (AND → OR)
- De la C_i la S_i: 2 gate-delays (XOR)

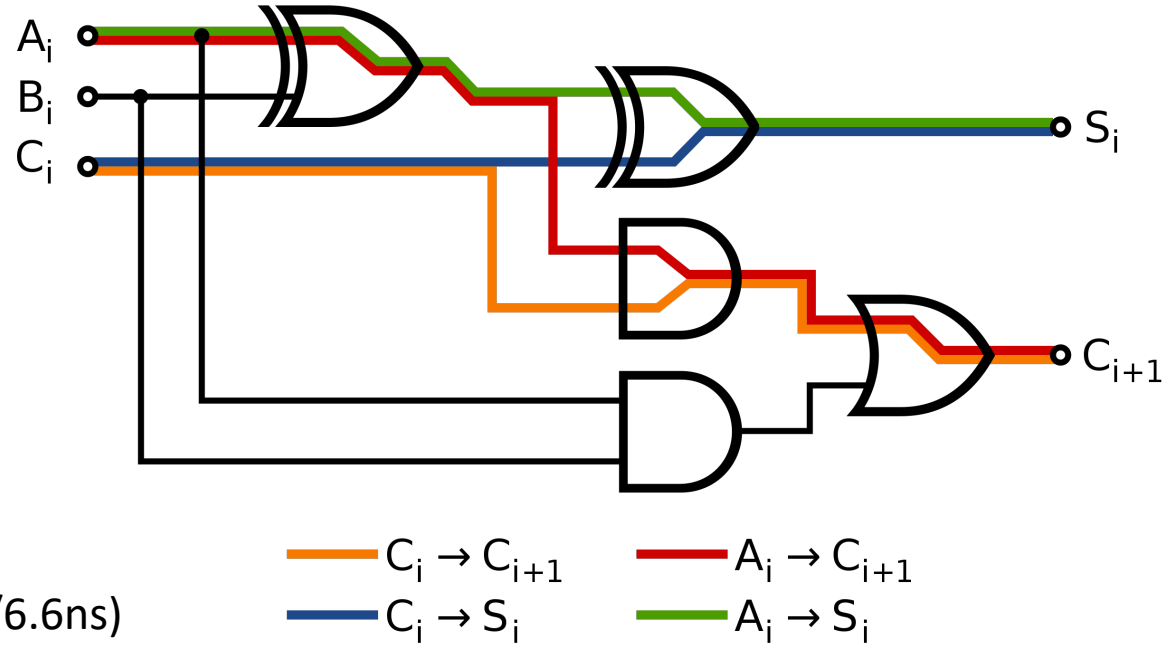


Întârzierea prin ripple-carry

Pentru că ieșirea de carry de la o etapă se leagă direct la intrarea de carry a următoarei etape, întârzierea totală pentru un sumator ripple-carry:

- 4 gate-delays din generarea primului semnal carry ($A_0/B_0 \rightarrow C_1$)
- 2 gate-delays pentru fiecare etapă intermediară ($C_i \rightarrow C_{i+1}$)
- 2 gate-delays la ultima etapă pentru producerea sumei și a ultimului carry-out ($C_{n-1} \rightarrow C_n$ și S_{n-1})

$$t_{ripple} = 4 + 2(n-2) + 2 = 2n + 2$$



Exemplu:

- Sumator pe 32 biți
- 2 gate-delay = 100ps
- Total delay = (2*32+2)*100ps = 6.6ns
- Dacă o adunare se execută într-un singur ciclu de ceas, frecvența procesorului trebuie limitată la maxim 151MHz (1/6.6ns)

Sumator Carry-Lookahead

- Calculează carry out (C_{out}) pentru blocuri de k biți folosind semnalele de *generare* și *propagare*
- **Definiții:**

– Coloana *i* produce un carry out fie prin
  - *generarea* unui carry out
  - *propagarea* unui carry in la ieșirea de carry out

– Semnalele Generate (G_i) și Propagate (P_i) petru fiecare coloană:
  - Coloana *i* va genera un carry out dacă A_i *ȘI* B_i sunt ambele 1.

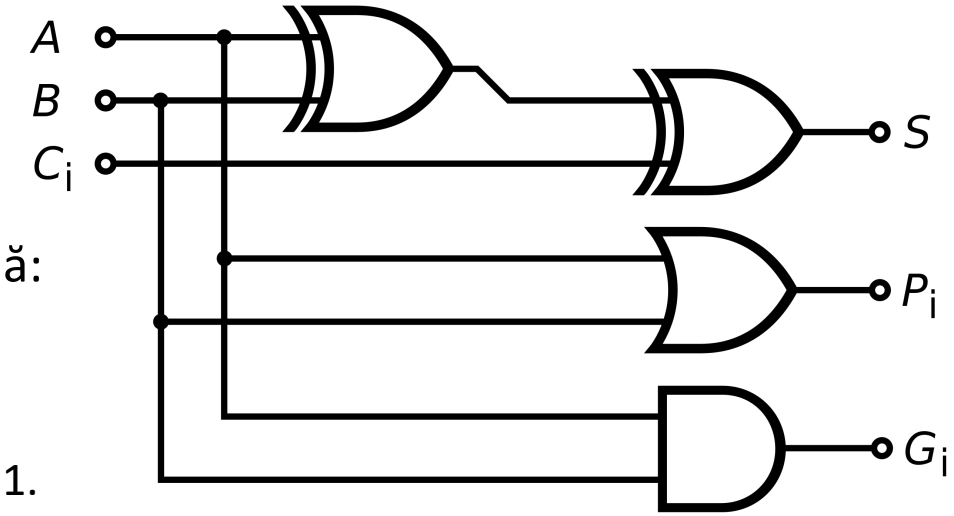
$$G_i = A_i B_i$$

  - Coloana *i* va propaga un carry in la carry out dacă A_i *SAU* B_i sunt 1.

$$P_i = A_i + B_i$$

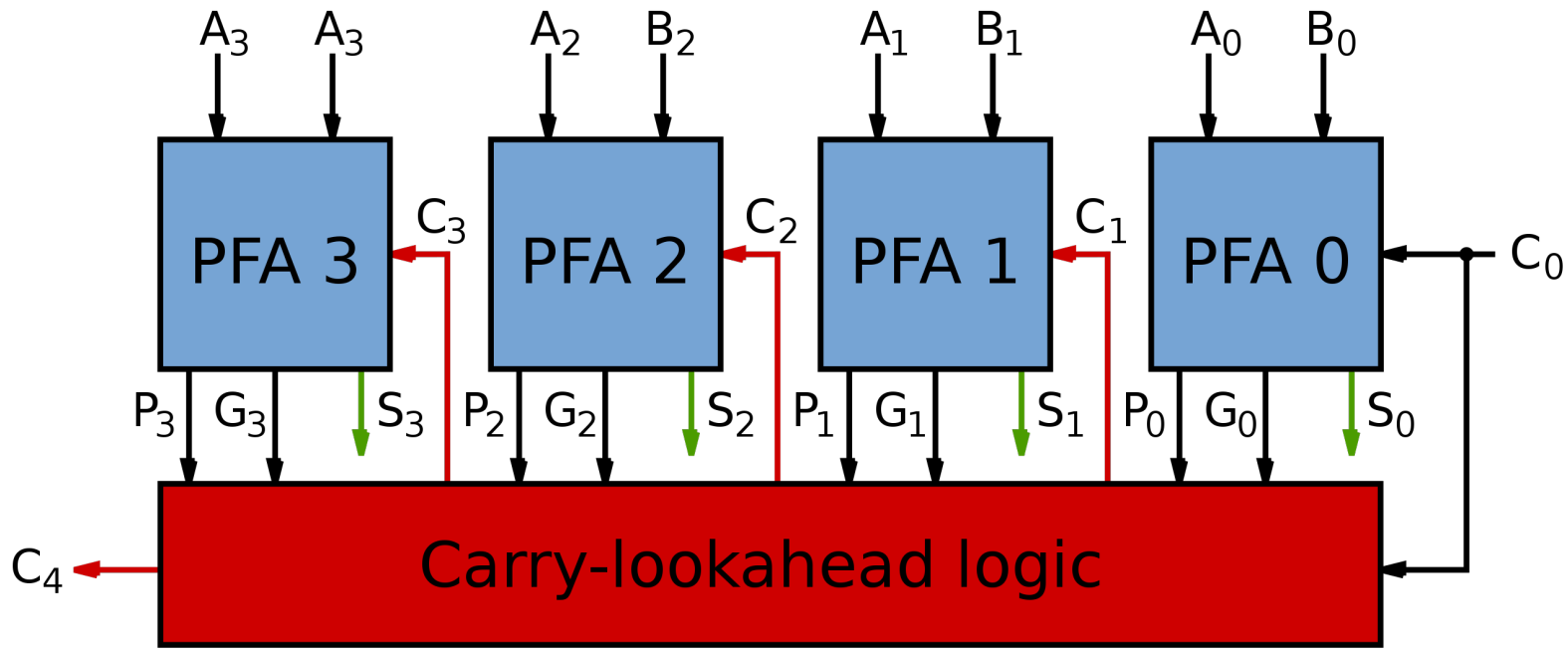
  - Semnalul carry out al coloanei *i* (C_i) este:

$$C_i = A_i B_i + (A_i + B_i)C_{i-1} = G_i + P_i C_{i-1}$$



Adunarea Carry-Lookahead

- **Pas 1:** Calculează G_i și P_i pentru toate coloanele
- **Pas 2:** Calculează G și P pentru blocuri de k-biți
- **Pas 3:** C_{in} se propagă prin fiecare bloc de k-biți de propagate/generate



Sumatorul Carry-Lookahead

- **Exemplu:** blocuri de 4 biți ($G_{3:0}$ și $P_{3:0}$):

$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0)$$

$$P_{3:0} = P_3 P_2 P_1 P_0$$

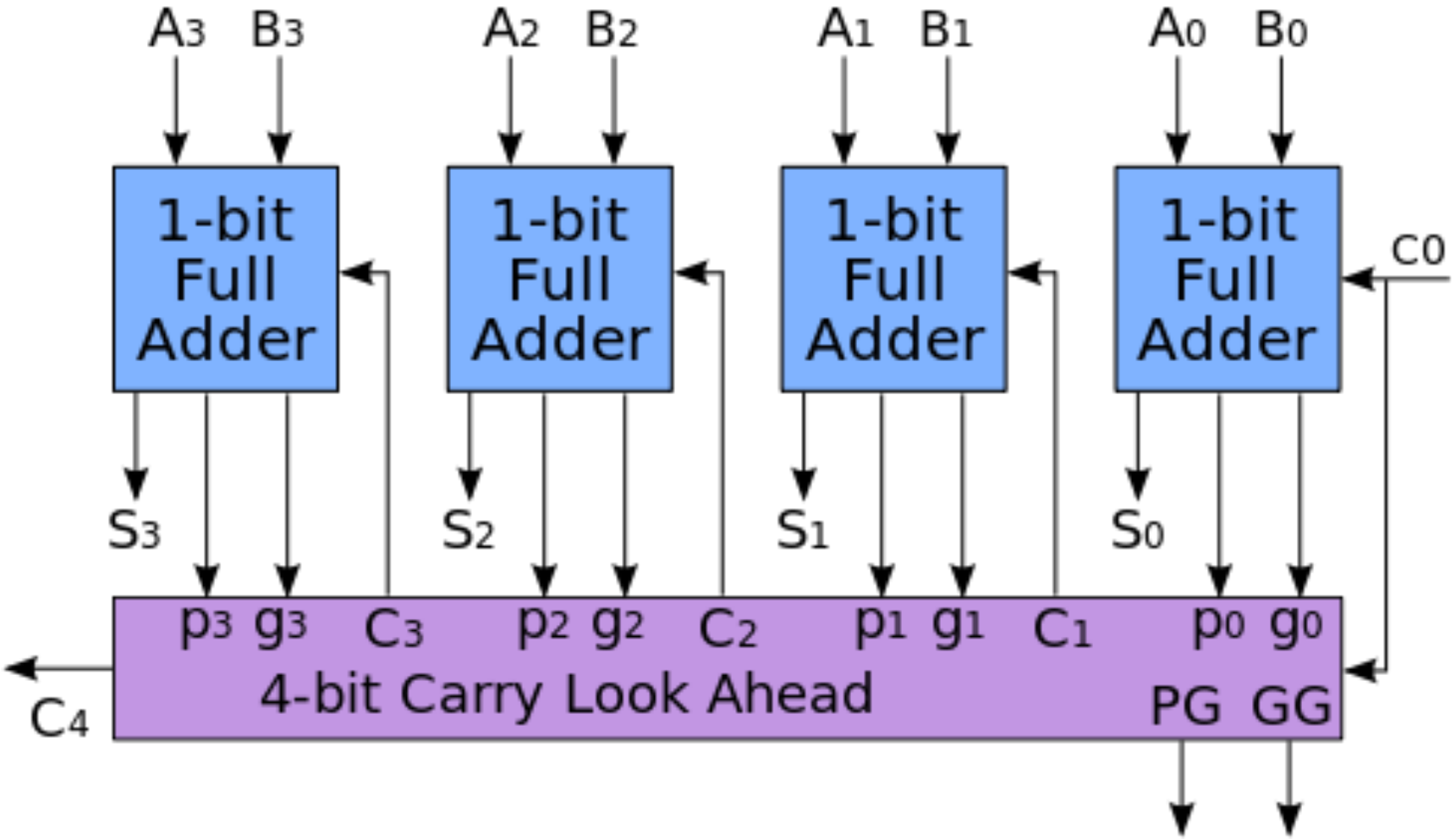
- **În general,**

$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} G_j)$$

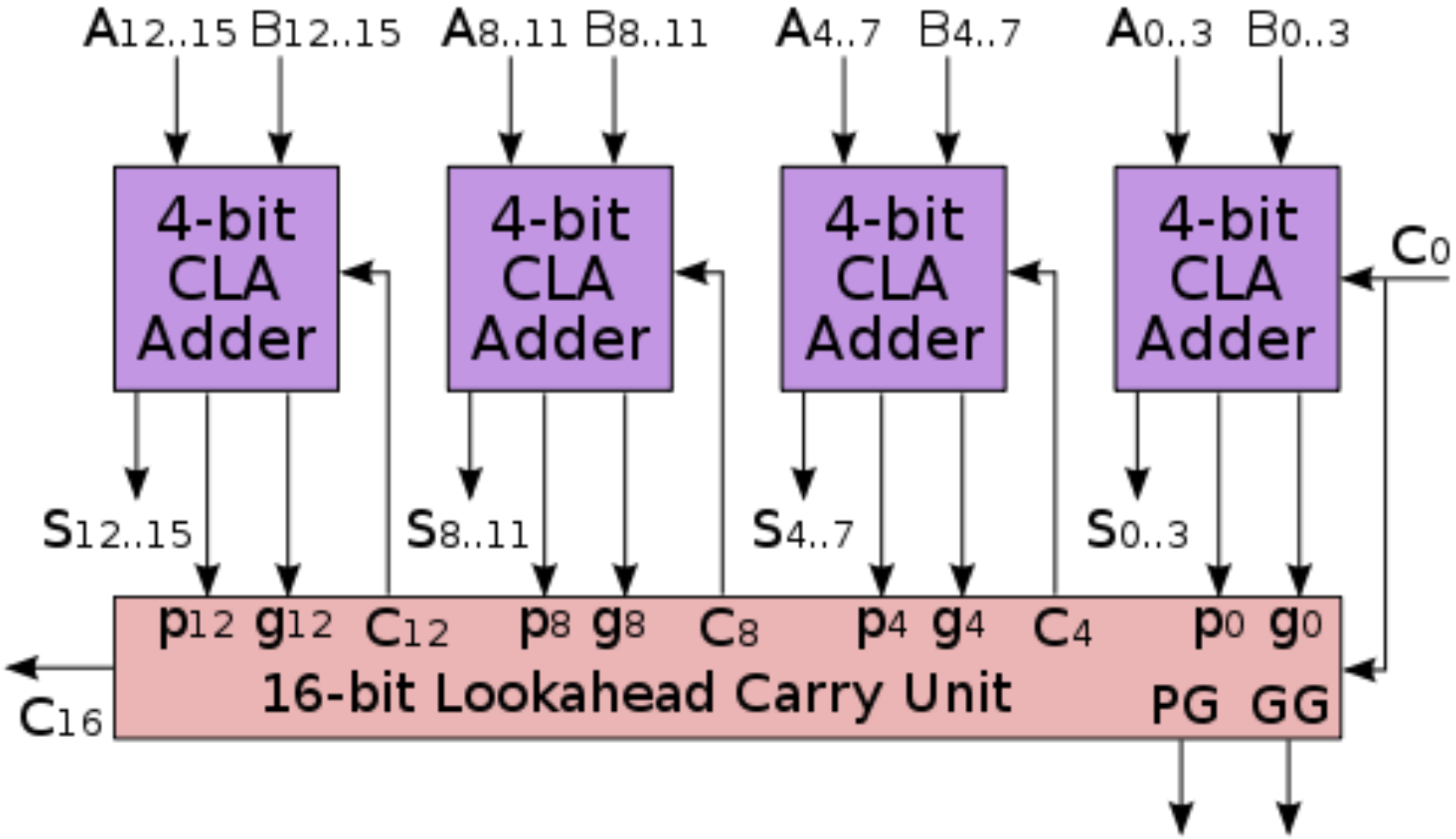
$$P_{i:j} = P_i P_{i-1} P_{i-2} P_j$$

$$C_i = G_{i:j} + P_{i:j} C_{i-1}$$

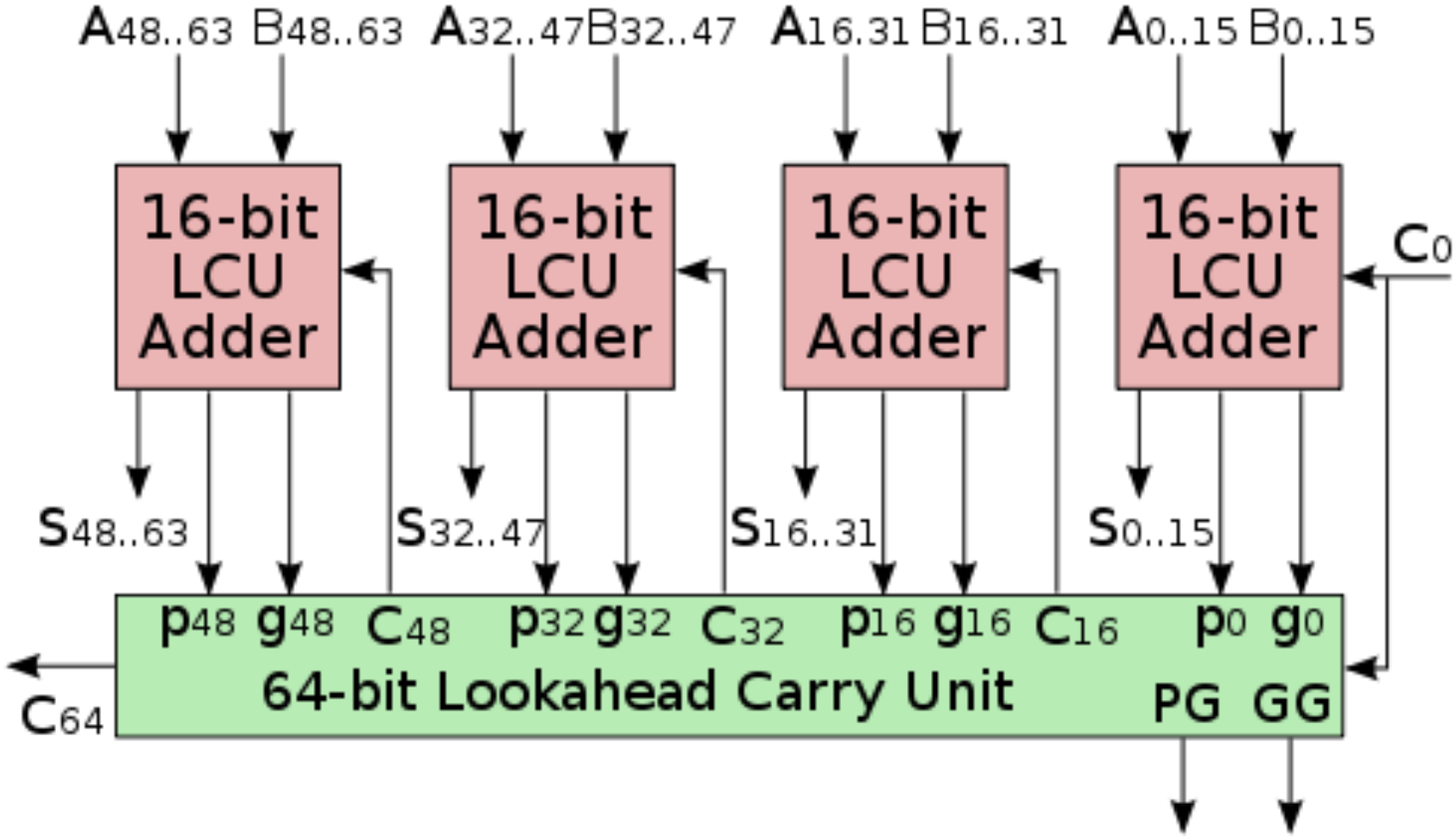
Carry-Lookahead 4-biți



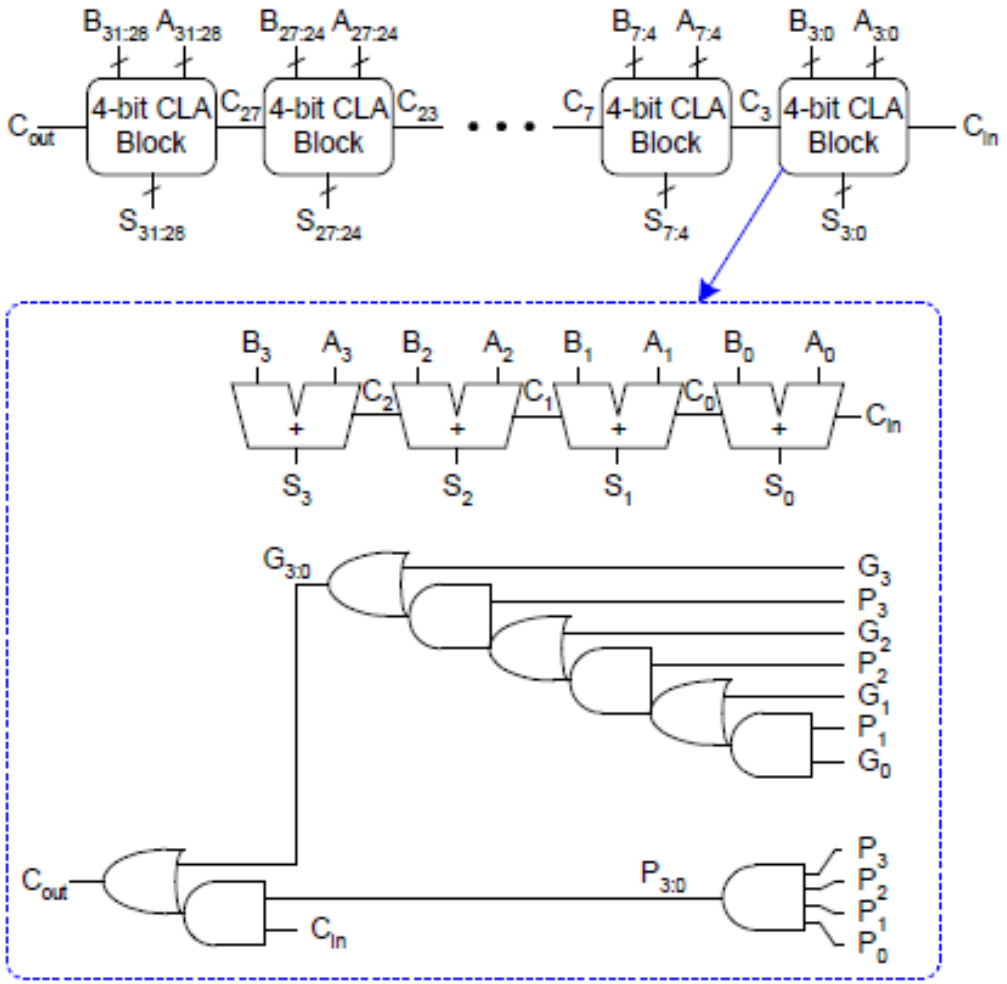
Carry-Lookahead 16-biți



Carry-Lookahead 64-biți



CLA pe 32-biți cu blocuri de 4-biți



Întârzierile printr-un sumator Carry-Look Ahead

Pentru un CLA pe N biți din blocuri de k biți:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

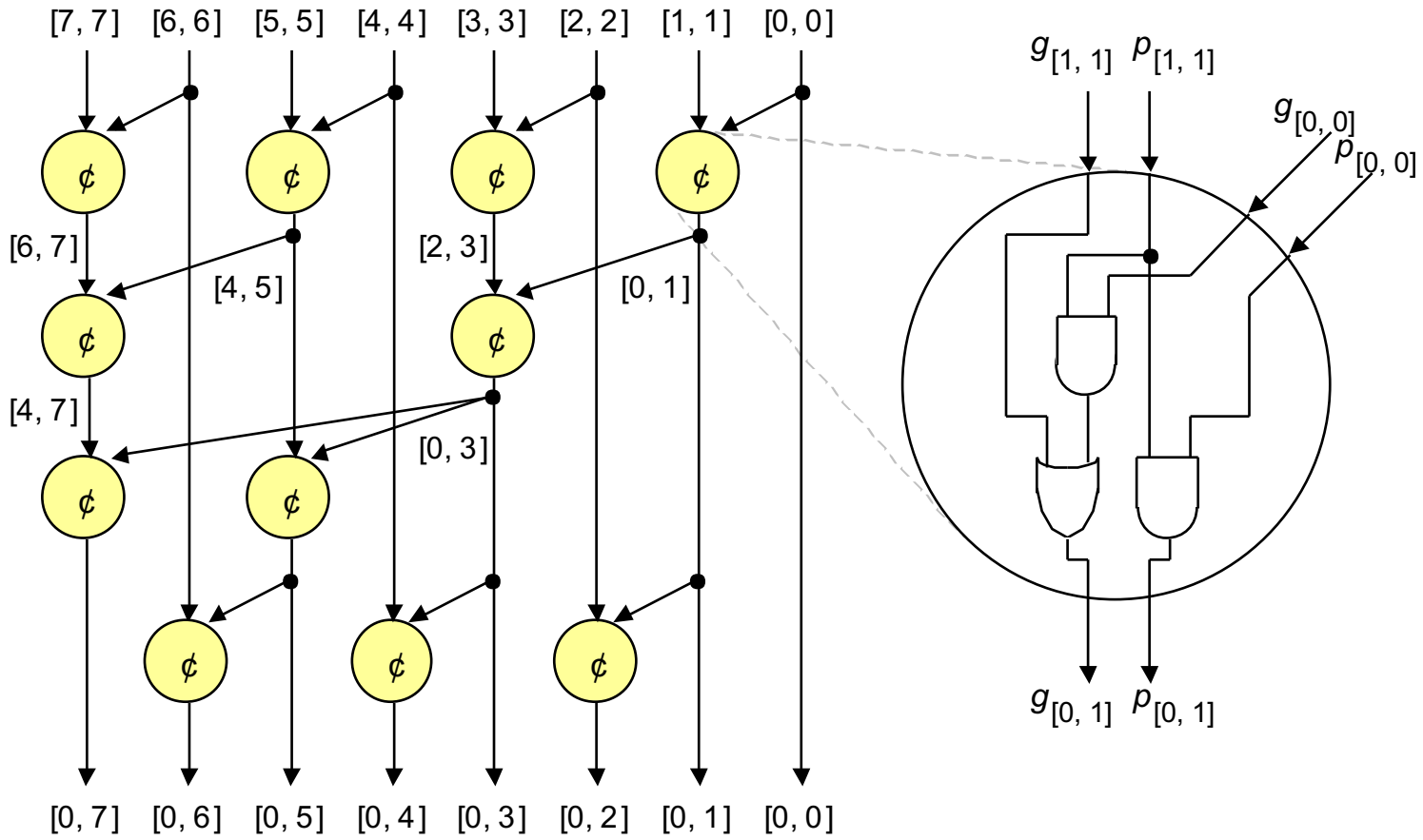
– t_{pg} : delay pentru generarea P_i, G_i

– t_{pg_block} : delay pentru generarea tuturor $P_{i:j}$, $G_{i:j}$

– t_{AND_OR} : delay de la C_{in} la C_{out} a ultimei porți AND/OR din blocul de k-biți al CLA

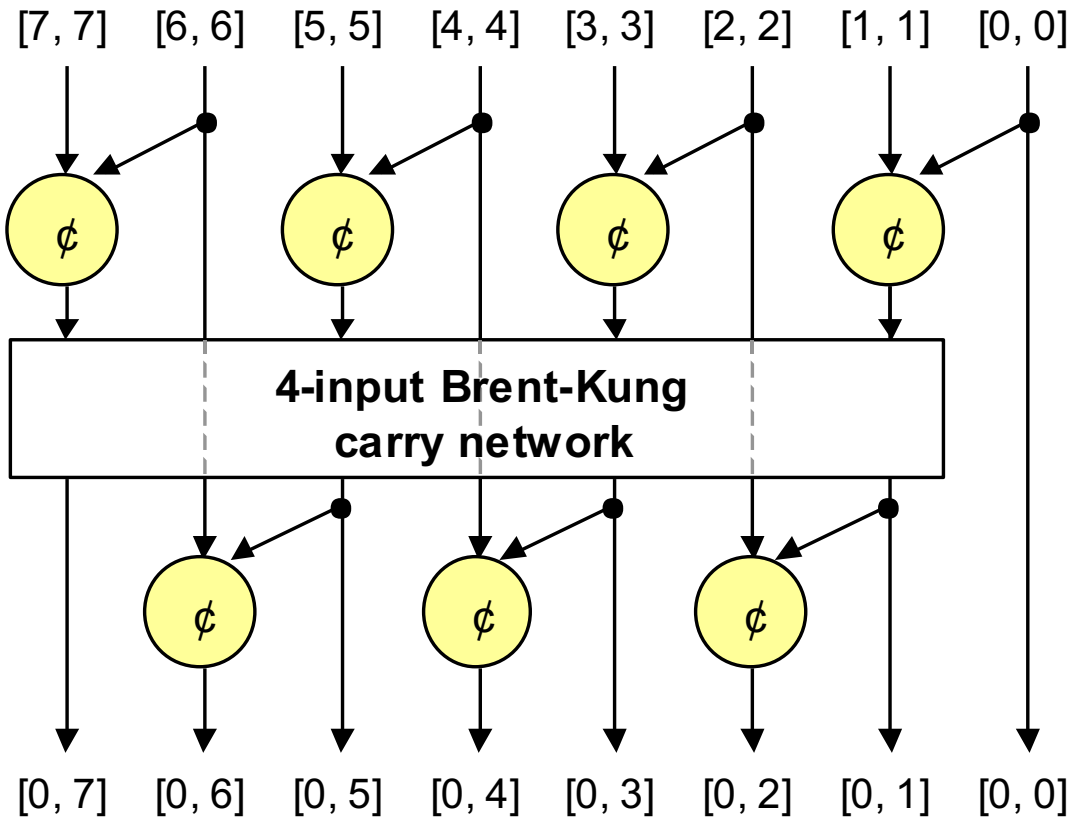
Un sumator carry-lookahead de N biți este mult mai rapid decât un sumator ripple-carry dacă $N > 16$

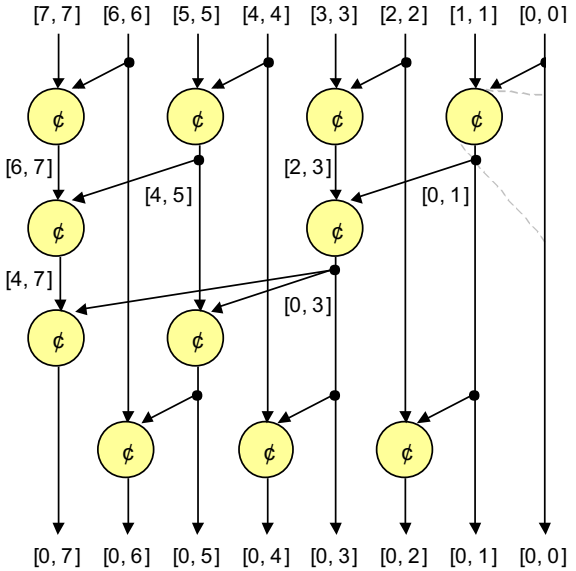
Rețele Carry - Carry Lookahead



Rețea carry lookahead Brent-Kung pentru un sumator pe 8 biți și detaliul de implementare pentru un bloc de carry.

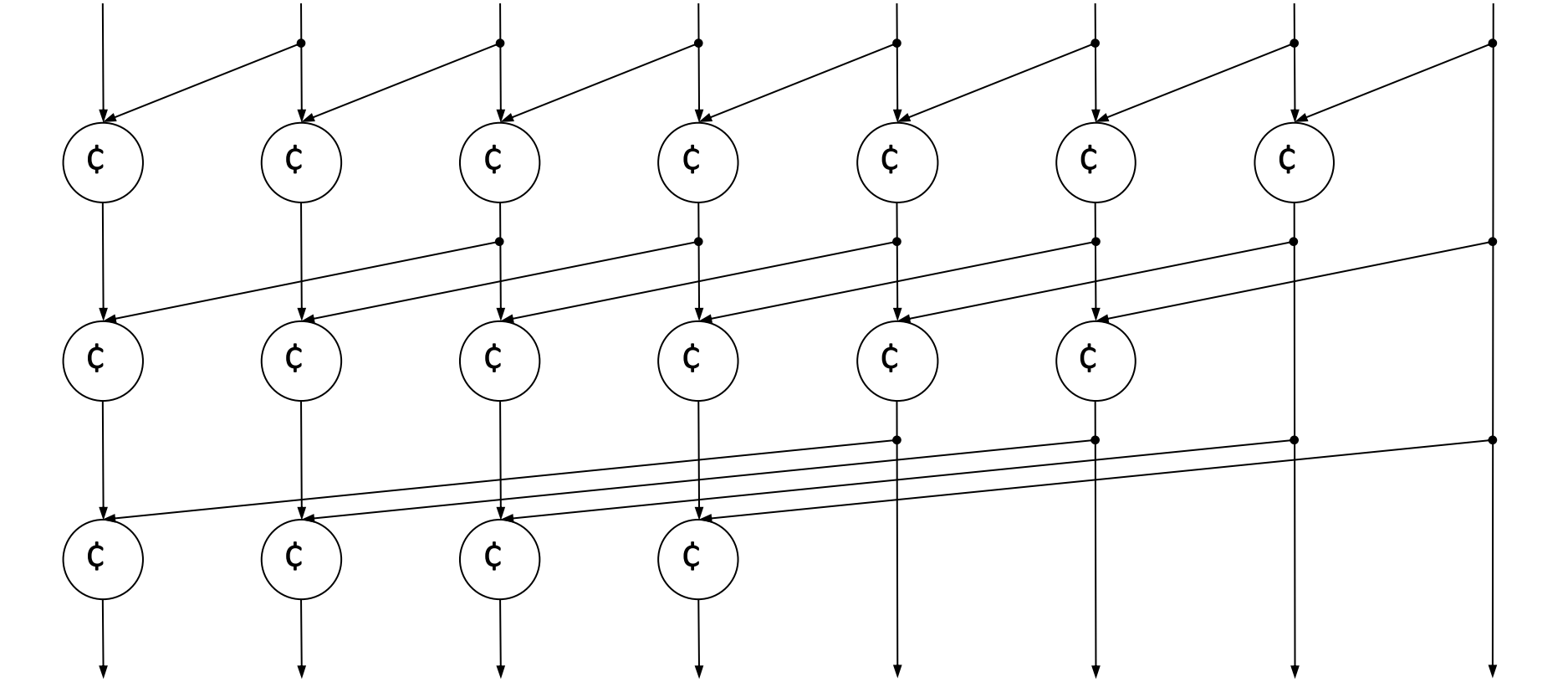
Structura recursivă a unei rețele carry Brent-Kung





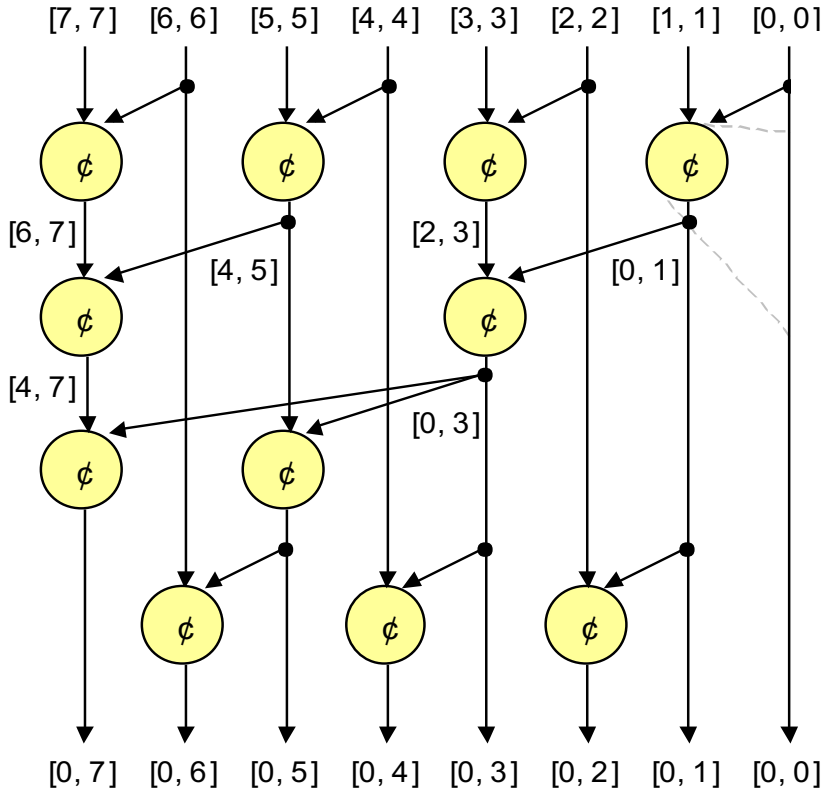
Rețea carry-lookahead Brent-Kung pentru un sumator pe 8 biți. Numai primul și ultimul nivel de operatori carry sunt evidențiați.

Un alt design: Rețeaua Kogge-Stone

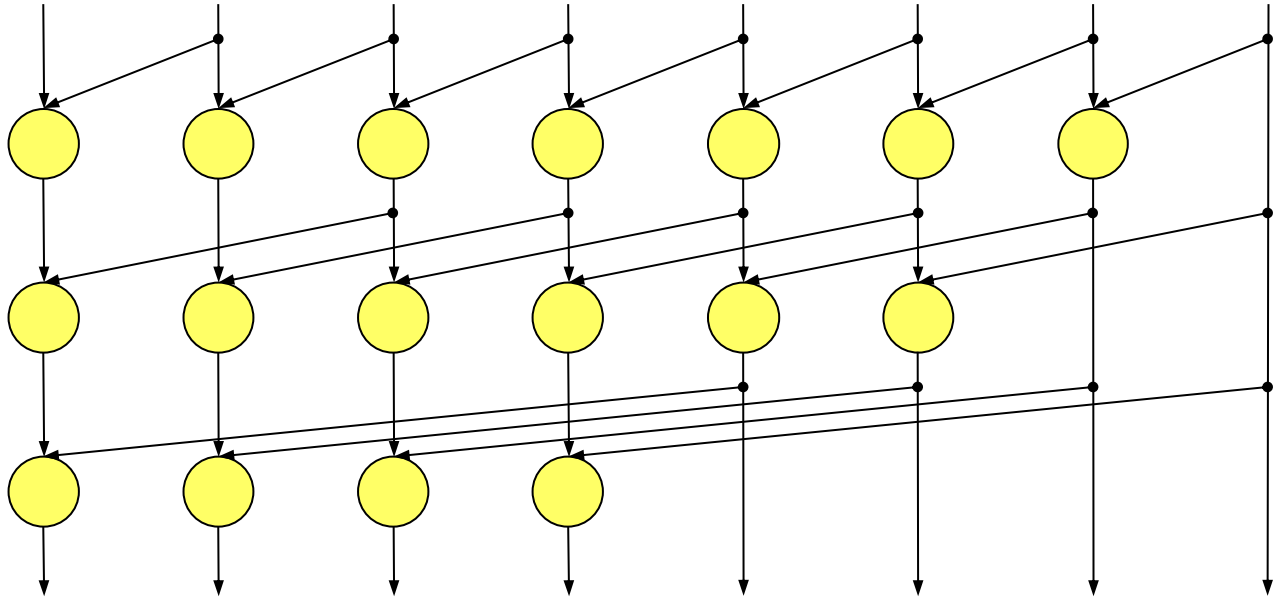


Rețea carry-lookahead Kogge-Stone pentru un sumator pe 8 biți.

Brent-Kung vs. Kogge-Stone Carry Network



11 operatori carry
4 niveluri



17 operatori carry
3 niveluri

http://xkcd.com/676/

Sumator cu prefixe

- Calculează carry in (C_{i-1}) pentru fiecare coloană apoi calculează suma:

$$S_i = (A_i \text{ XOR } B_i) \text{ XOR } C_i$$

- Calculează G și P pentru blocuri de 1-, 2-, 4-, 8-biți, etc.

Până când toate G_i (carry in) sunt cunoscute

- $\log_2 N$ etape

Sumator cu prefixe

- Carry in este ori *generat* de o coloană sau *propagat* de la o coloană anterioară
- Coloana -1 ține C_{in}, așa că:

$$G_{-1} = C_{in},\ P_{-1} = 0$$

- Carry in al coloanei *i* = carry out al coloanei *i-1*:

$$C_{i-1} = G_{i-1:-1}$$

$G_{i-1:-1}$: semnal generate de la coloana *i*-1 la -1

- Ecuația sumei:

$$S_i = (A_i \text{ XOR } B_i) \text{ XOR } G_{i-1:-1}$$

- **Scop: Calculează rapid** $G_{0:-1}$, $G_{1:-1}$, $G_{2:-1}$, $G_{3:-1}$, $G_{4:-1}$, $G_{5:-1}$, …(numite ***prefixe***)

Sumator cu prefixe

- Semnalele Generate și Propagate pentru un bloc de la biții *i*:*j*:

$$G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$

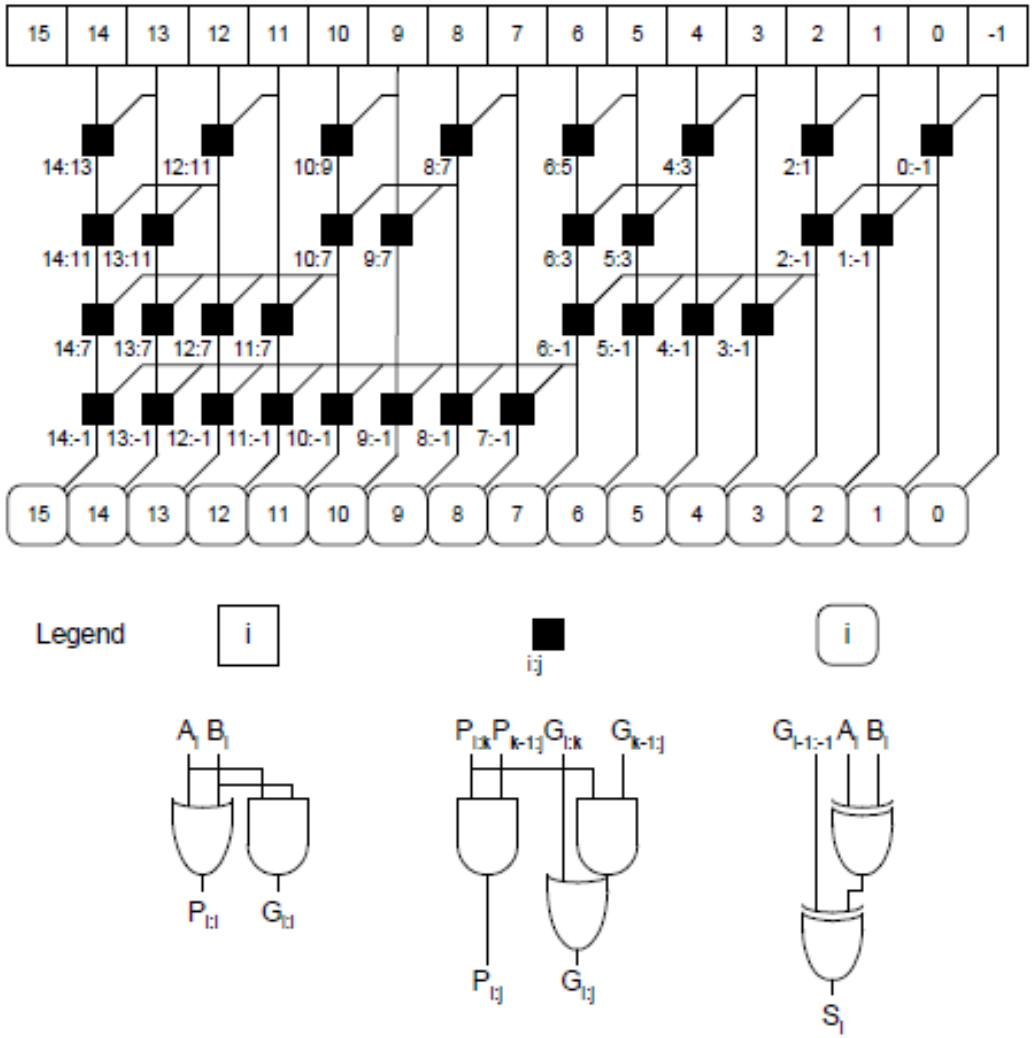
- În limba română:

– **Generate:** blocul *i*:*j* va genera carry dacă:

- partea superioară (*i*:*k*) generează carry, sau
- partea superioară propagă un carry generat de partea inferioară (*k*-1:*j*)

– **Propagate:** blocul *i*:*j* va propaga carry dacă și blocul superior și cel inferior propagă carry-ul

Schema unui sumator cu prefixe



$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_prefix}) + t_{XOR}$$

– t_{pg}: delay pentru a produce $P_i G_i$ (poartă AND sau OR)

– t_{pg_prefix}: delay introdus de celula (neagră) pentru calculul semnalelor combinate (i,j) (porți AND-OR)

Compararea performanțelor

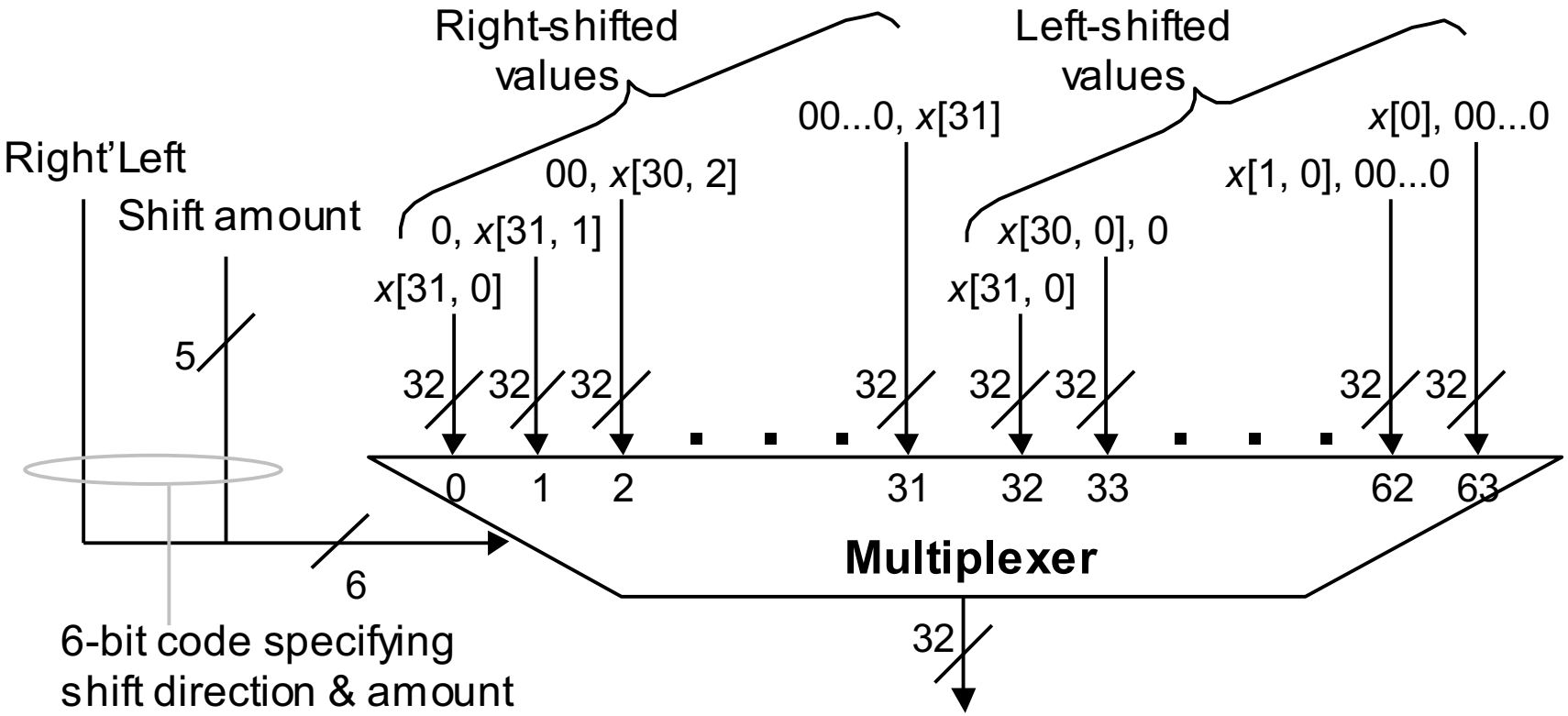
Comparați întârzierile prin trei sumatoare de 32 de biți cu ripple-carry, carry-lookahead și prefixe

- CLA are blocuri de 4-biți
- 2-input gate delay = 100 ps; full adder delay = 300 ps

$$t_{\text{ripple}} = Nt_{FA} = 32(300 \text{ ps})$$
$$= \mathbf{9.6 \text{ ns}}$$
$$t_{CLA} = t_{pg} + t_{pg\_\text{block}} + (N/k - 1)t_{\text{AND\_OR}} + kt_{FA}$$
$$= [100 + 600 + (7)200 + 4(300)] \text{ ps}$$
$$= \mathbf{3.3 \text{ ns}}$$
$$t_{PA} = t_{pg} + \log_2 N(t_{pg\_\text{prefix}}) + t_{\text{XOR}}$$
$$= [100 + \log_2 32(200) + 100] \text{ ps}$$
$$= \mathbf{1.2 \text{ ns}}$$

Operații logice și de shiftare

Conceptual, shiftările pot fi implementate pur combinațional prin multiplexoare

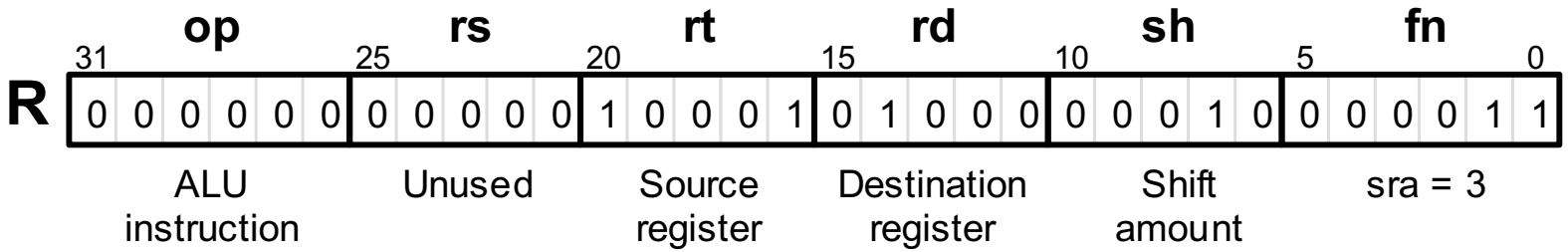


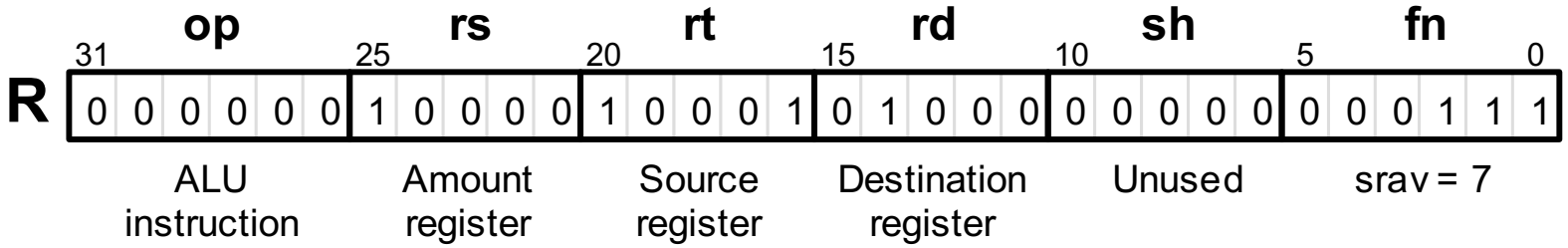
Unitate de shiftare logică folosind multiplexoare.

Shiftari aritmetice

Scop: înmulțirea și împărțirea cu puteri ale lui 2

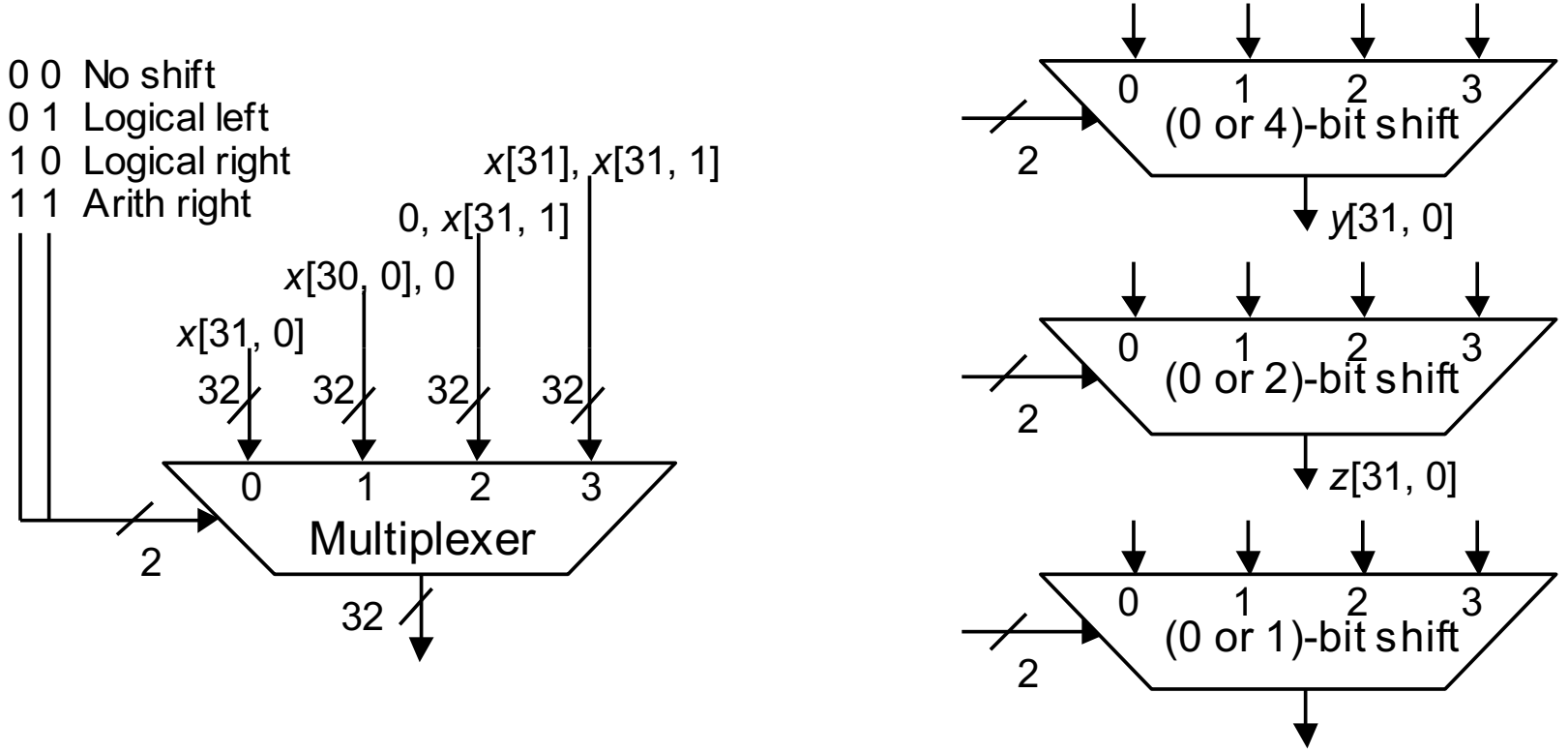
```
sra  $t0,$s1,2     # $t0 ← ($s1) right-shifted by 2
srav $t0,$s1,$s0   # $t0 ← ($s1) right-shifted by ($s0)
```





Operații de shiftare în cod mașină.

Implementarea shiftării pe mai multe niveluri



(a) Single-bit shifter

(b) Shifting by up to 7 bits

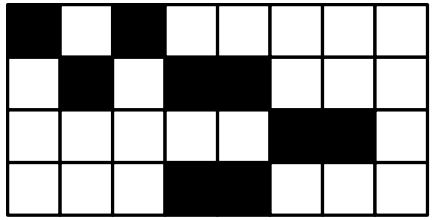
Shiftare pe mai multe niveluri într-un barrel shifter.

Manipularea biților prin shiftări si operații logice

Bits 10-15

ȘI cu o mască izolează un câmp: 0000 0000 0000 0000 1111 1100 0000 0000
Shiftează rezultatul cu 10 poziții pentru a alinia câmpul la dreapta cuvântului de date
Rezultatul poate să aibă valori între 0 și 63, în funcție de valoarea izolată

32-pixel (4 × 8) block of black-and-white image:



Representation as 32-bit word:

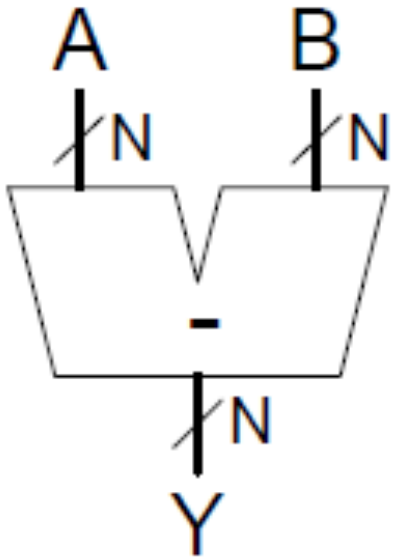
Row 0	Row 1	Row 2	Row 3
1010 0000	0101 1000	0000 0110	0001 0111

Hex equivalent: 0xa0a80617

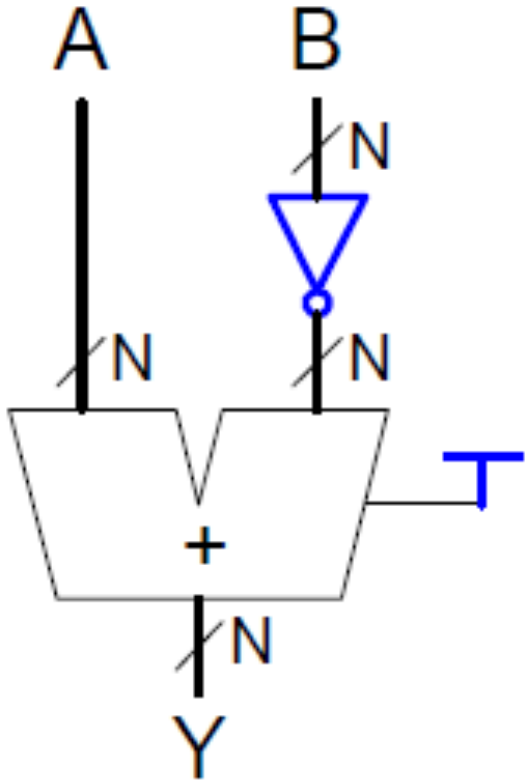
Un bloc de imagine alb-negru de 4 × 8 pixeli reprezentat ca un cuvânt de 32 de biți.

Scăzător

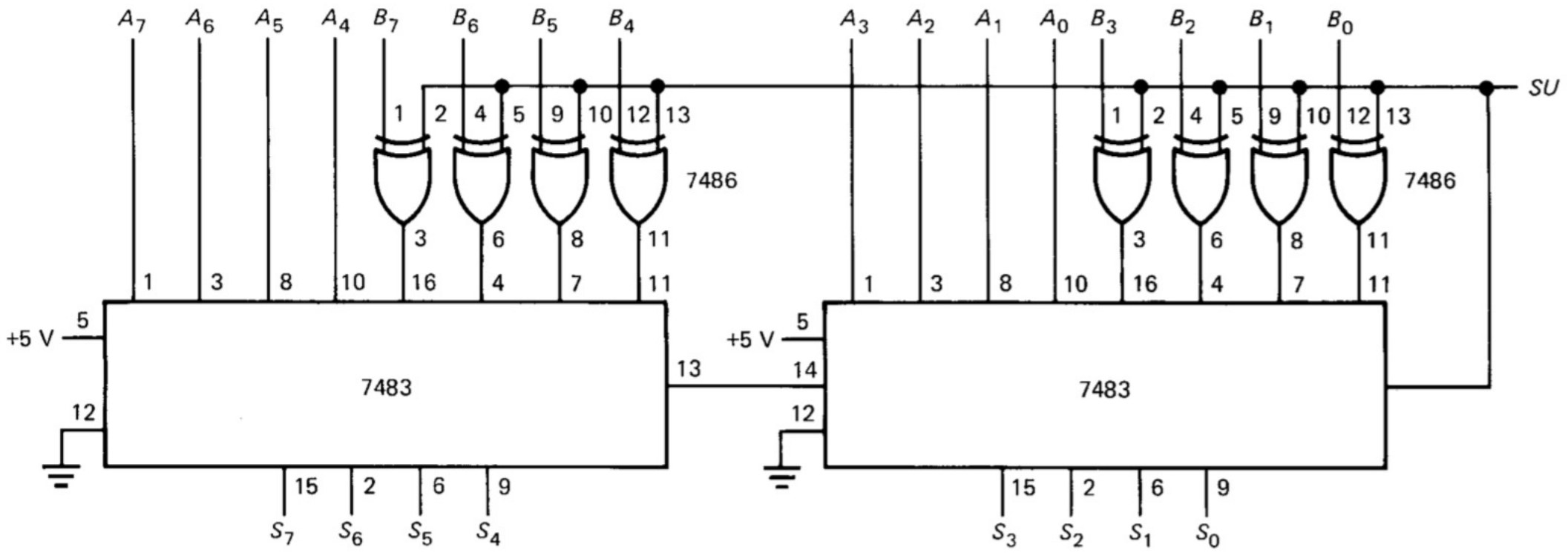
Simbol



Implementare



Sumator / Scăzător binar pe 8 biți

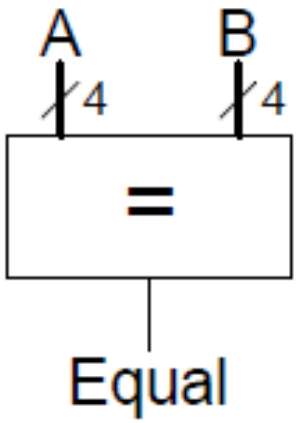


74LS83 – Sumator binary complet pe 4 biți cu logică de fast carry

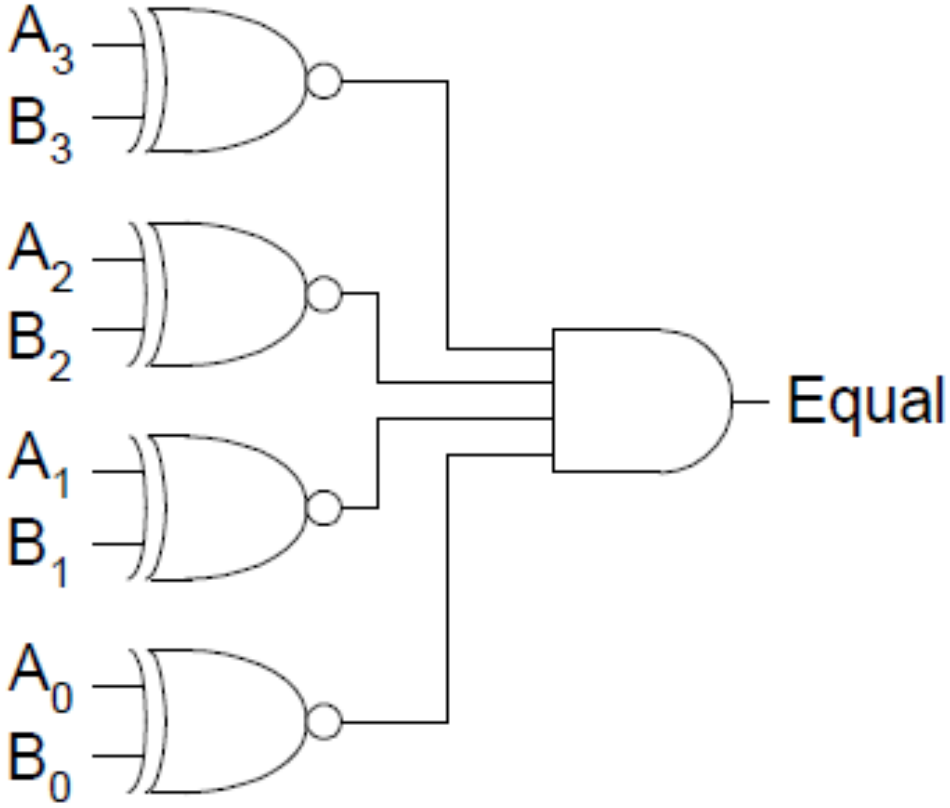
74HC86 – Quad XOR 2-input gate

Comparator: Egalitate

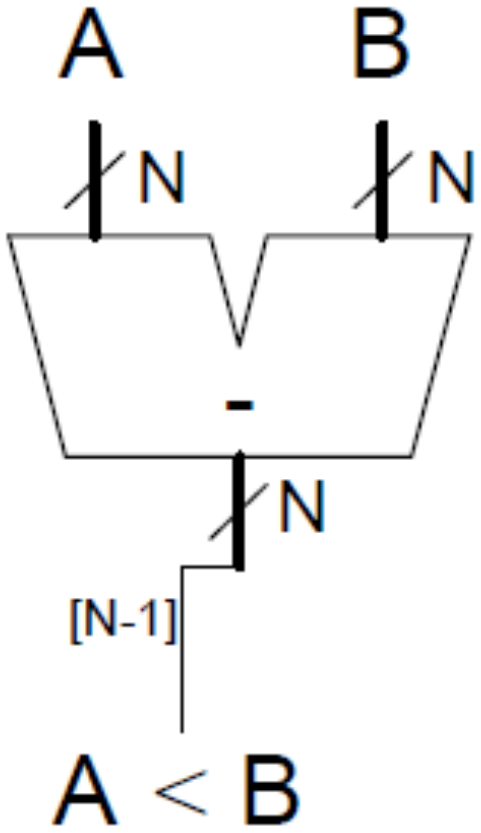
Simbol



Implementare



Comparator: “Mai mic decât”



Acknowledgements

- Aceste slide-uri conțin materiale aparținând:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Behrooz Parhami (UCSB)

- MIT material derived from course 6.823
- UCB material derived from course CS252