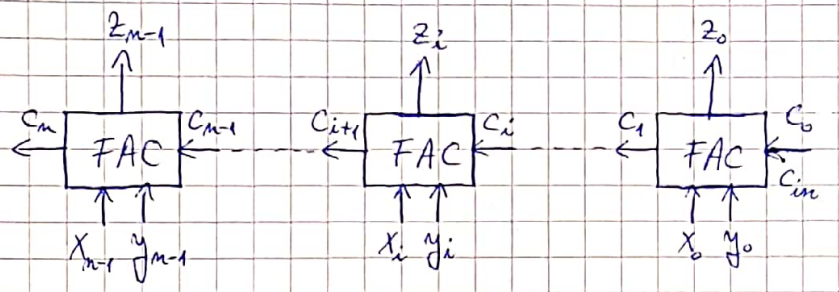
Curs 1

Cap. I. Analiza funcțională și sinteza dispozitivelor binare și zecimale de adunare și scădere

1.1. Sumatoare paralele cu propagare serială (Ripple Carry Adder)

Full Adder Cell (FAC)



inputs			outputs	
x_i	y_i	c_i	z_i	c_{i+1}
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

$$z_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$$

$$z_i = \bar{x}_i(\bar{y}_i c_i + y_i \bar{c}_i) + x_i(\bar{y}_i \bar{c}_i + y_i c_i)$$

$$z_i = \bar{x}_i(y_i \oplus c_i) + x_i(y_i \odot c_i)$$

$$z_i = x_i \oplus y_i \oplus c_i$$

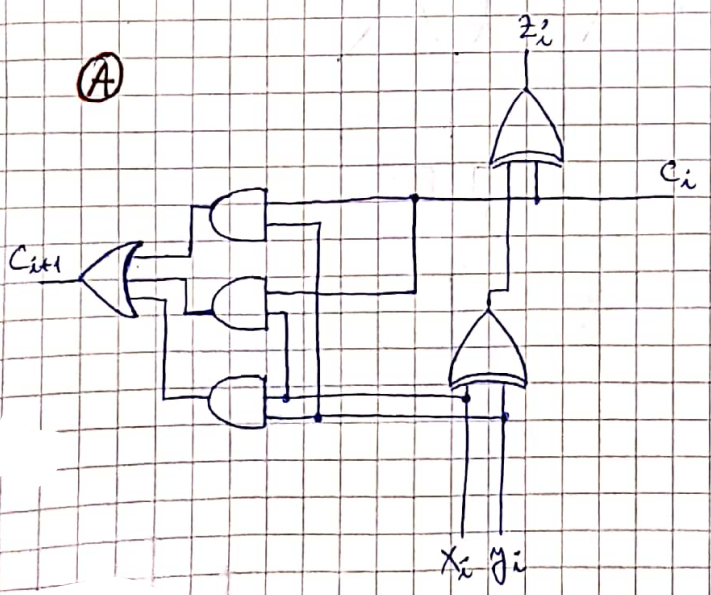
$$c_{i+1} = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$$

$$c_{i+1} = \bar{x}_i y_i c_i + x_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$$

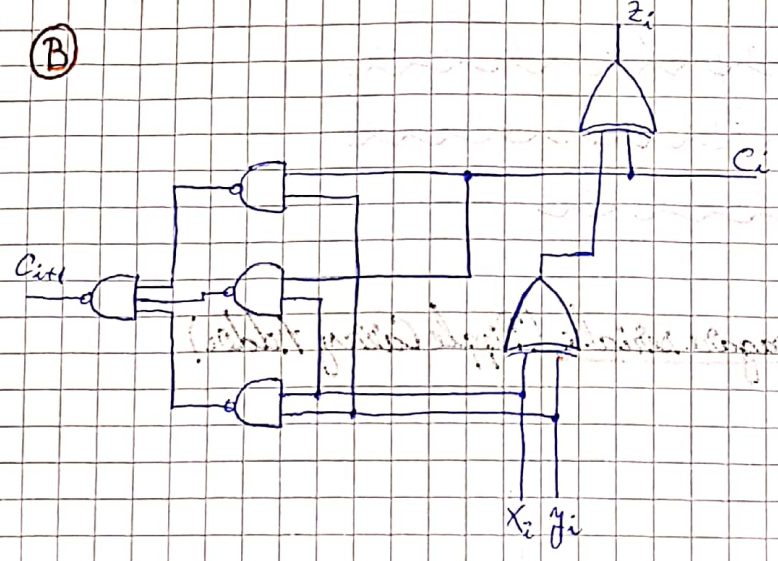
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$x \oplus y = \bar{x} y + x \bar{y}$$

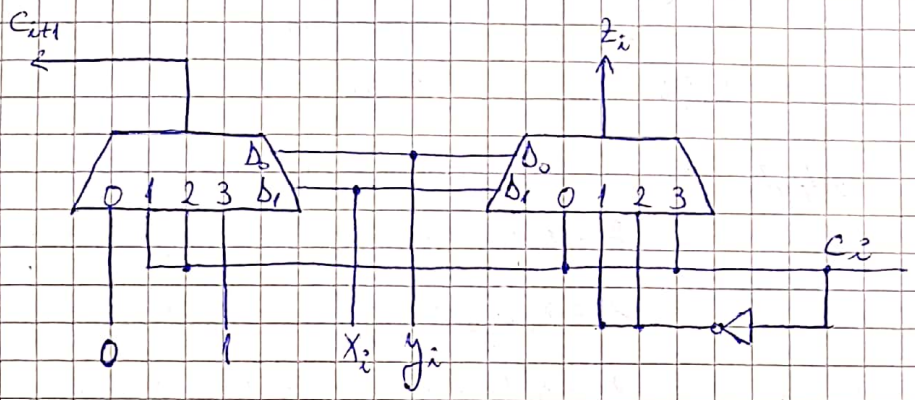
(A)



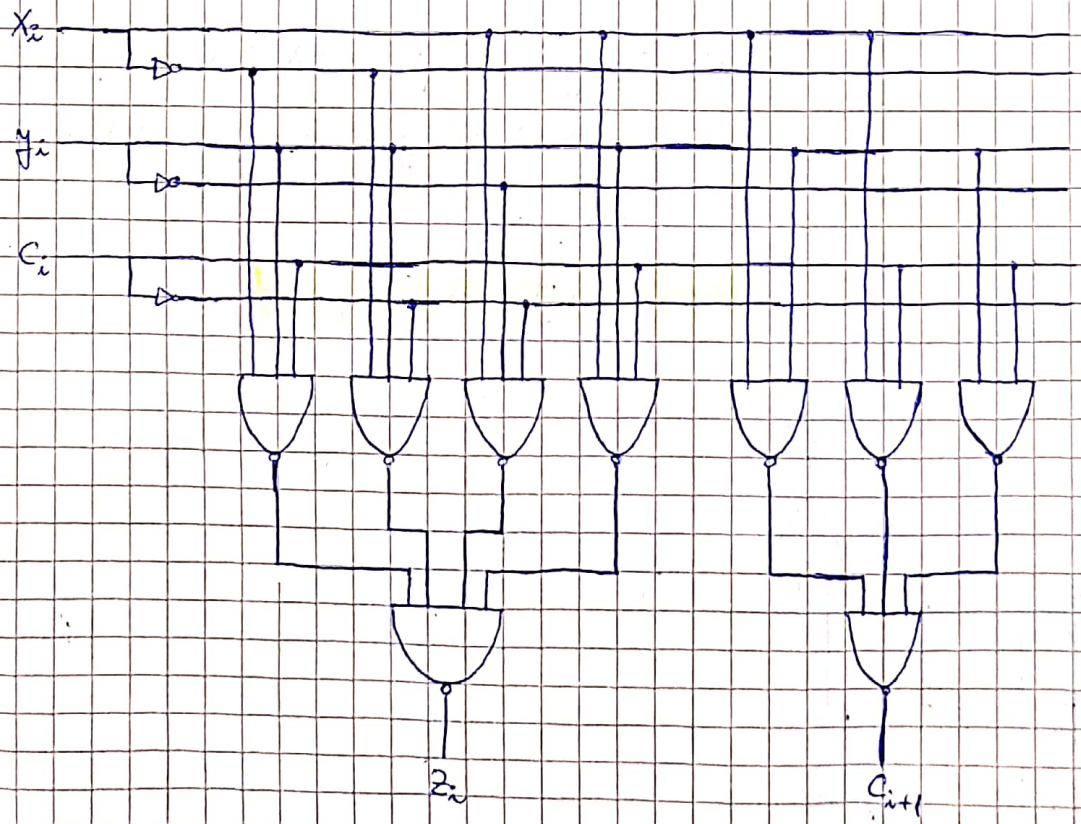
B


z_i
c_i
c_{i+1}
x_i y_i

C

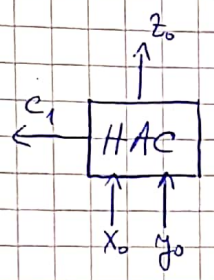

c_{i+1}
z_i
D_0
D_1
0 1 2 3
D_0
D_1
0 1 2 3
c_i
0
1
x_i y_i

D

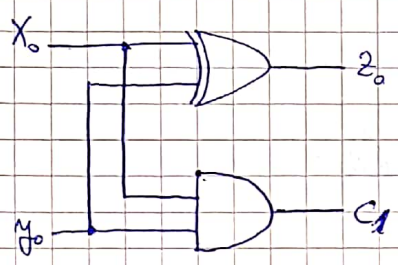

x_i
y_i
c_i
z_i
c_{i+1}

Când $c_0 = c_{in} = 0 \Rightarrow$ Half Adder Cell (HAC)

$$\begin{cases} z_0 = x_0 \oplus y_0 \oplus 0 \\ c_1 = x_0 \cdot 0 + y_0 \cdot 0 + x_0 \cdot y_0 \end{cases} \Leftrightarrow \begin{cases} z_0 = x_0 \oplus y_0 \\ c_1 = x_0 \cdot y_0 \end{cases}$$



Ⓐ XOR & AND

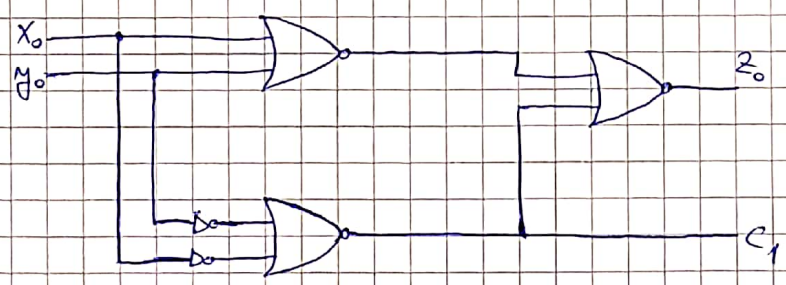


Ⓑ XNOR

$$z_0 = x_0 \oplus y_0 = \overline{x_0} y_0 + x_0 \overline{y_0} = \overline{\overline{(x_0 + y_0)(\overline{x_0} + \overline{y_0})}}$$

$$z_0 = \overline{\overline{x_0 + y_0} + \overline{\overline{x_0} + \overline{y_0}}}$$

$$c_1 = \overline{\overline{x_0 \cdot y_0}} = \overline{\overline{x_0} + \overline{y_0}}$$

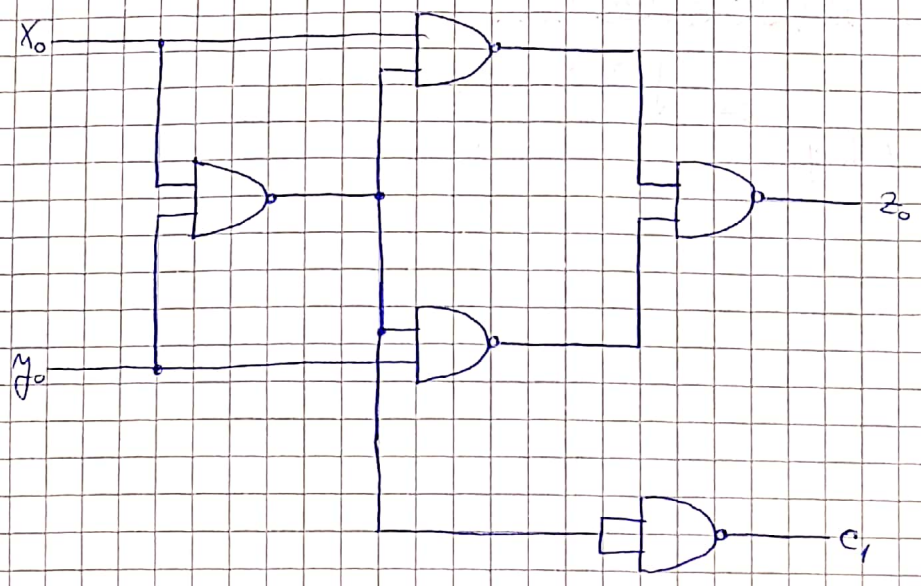


Ⓒ NAND

$$z_0 = (x_0 + y_0)(\overline{x_0} + \overline{y_0}) = \overline{\overline{x_0(\overline{x_0} + \overline{y_0}) + y_0(\overline{x_0} + \overline{y_0})}}$$

$$z_0 = \overline{\overline{x_0 \cdot \overline{x_0 \cdot y_0}} \cdot \overline{y_0 \cdot \overline{x_0 \cdot y_0}}}$$

$$c_1 = \overline{\overline{x_0 \cdot y_0}}$$



Lab 1

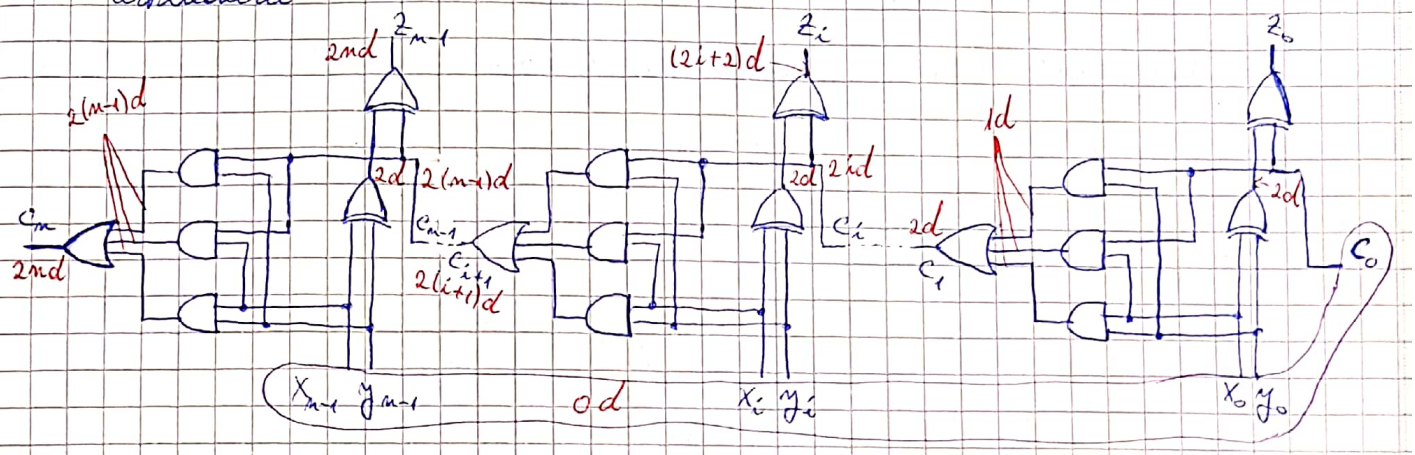
1. Cd. următorii operanzi fără semn să se calculeze:

- 300 + 279 =
- 300 − 279 =
- 790 + 607 =
- 790 − 607 =

Curs 2

1.1.1 Critical path

= se referă la calea cea mai lungă de propagare a semnalelor într-o arhitectură



- Se cd. că orice poartă primitivă (SAU, ȘI) are o întârziere de 1d (u.m. folosită pt. întârzieri)
- Invertoarele au întârziere 0d
- Porțile SAU Exclusiv au întârzieri 2d
- Toți biții lui x, y și c_0 sunt conectați în paralel, simultan, la momentul inițial al circuitului

Întârzierea pt un sumator RCA: $D_{RCA}^{z/c_{out}} = 2nd$

- Condițiile speciale ale operațiilor de adunare
  - rezultat zero
  - rezultat negativ
  - c_{out} din MSB (Most Significand bit)
  - overflow:

→ unsigned: X = +9   1001   01001

y = +12   1100   01100

1 0101 = 5!   10101 = 21

Overflow la adunarea nr. fără semn apare atunci când se generează carry din cel mai semnificativ rang (MSB)

→ signed: $X = +4$ 0100 00100

$y = +7$ 0111 00111

1011 = -5! 01011 = +11

$X = -5$ 1011 11011

$y = -6$ 1010 11011

1 0101 = +5! 1 10101 = -11

01011

Pt. a extinde nr. pe mai mulți biți se dublează bitul de semn.

Condiția de overflow la nr. cu semn apare at. când la adunarea a 2 nr. cu același semn rezultă un nr. de semn opus.

Luăm $X = x_{n-1}x_{n-2}\ldots x_1x_0$ și $Y = y_{n-1}y_{n-2}\ldots y_1y_0$ și vrem să vedem când apare overflow (nr. sunt signed).

overflow $\overset{not.}{=} v$

Input			Output	
x_{n-1}	y_{n-1}	c_{n-1}	z_{n-1}	v
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

← biți de semn

$$v = \overline{y_{n-1}}\,\overline{x_{n-1}}\,c_{n-1} + x_{n-1}y_{n-1}\overline{c_{n-1}}$$

$c_{n-1} \oplus 1$

$$v \overset{I_2}{=} \overline{x_{n-1}}\,\overline{y_{n-1}}\,c_{n-1} \oplus x_{n-1}y_{n-1}\overline{c_{n-1}} \overset{I_1}{=}$$

$$= (\overline{x_{n-1}}\,\overline{y_{n-1}} \oplus x_{n-1}y_{n-1})c_{n-1} \oplus x_{n-1}y_{n-1} \overset{I_2'}{=}$$

$$= (\underbrace{\overline{x_{n-1}}\,\overline{y_{n-1}} + x_{n-1}y_{n-1}}_{x_{n-1} \oplus y_{n-1} \oplus 1})c_{n-1} \oplus x_{n-1}y_{n-1} \overset{I_1}{=}$$

$$= x_{n-1}c_{n-1} \oplus y_{n-1}c_{n-1} \oplus c_{n-1} \oplus x_{n-1}y_{n-1} \overset{I_2'}{=}$$

$$= (x_{n-1}c_{n-1} + y_{n-1}c_{n-1} + x_{n-1}y_{n-1}) \oplus c_{n-1}$$

$$= c_n \oplus c_{n-1}$$

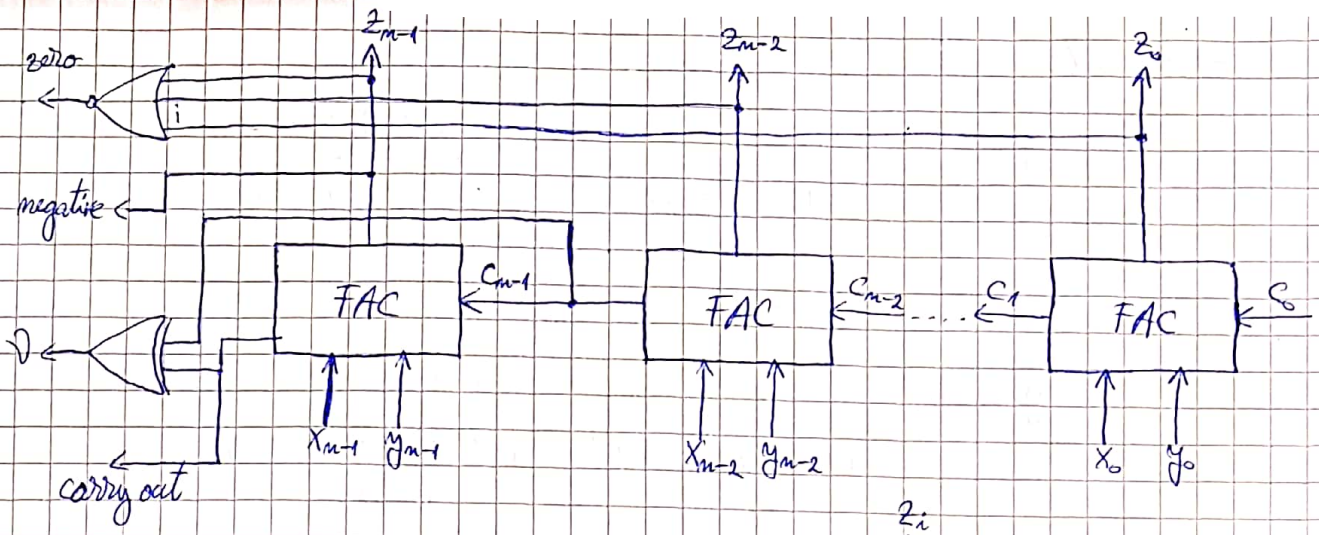
$$v = c_n \oplus c_{n-1}$$

Identități logice:

I_1: $(A \oplus B)C = AC \oplus BC$

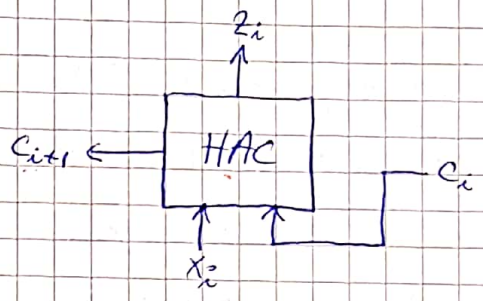
I_2: $A + B = A \oplus B \oplus AB$

I_2': $A \oplus B = (A+B) \oplus AB$



$X = x_{n-1} x_{n-2} \dots x_1 x_0$

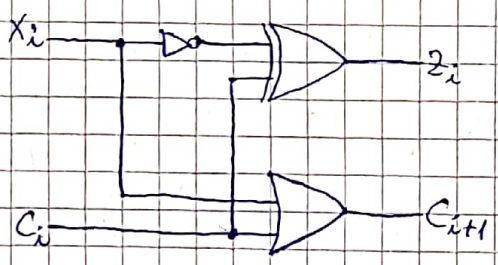
$Y = y_{n-1} y_{n-2} \dots y_1 y_0$

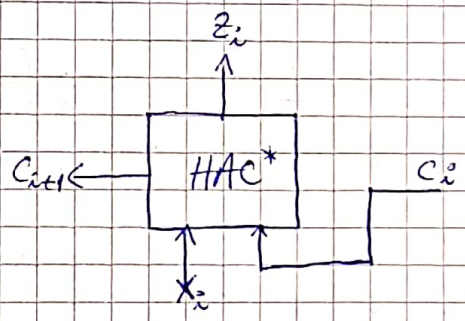


Y = constantă (impare)

$$y_i = 0 \Rightarrow \begin{cases} z_i = x_i \oplus \overset{0}{\overset{\shortparallel}{y_i}} \oplus c_i = x_i \oplus c_i \\ c_{i+1} = x_i \cdot c_i + 0 \cdot (x_i + c_i) = x_i \cdot c_i \end{cases} \Bigg\} \boxed{HAC}$$

$$y_i = 1 \Rightarrow \begin{cases} z_i = x_i \oplus 1 \oplus c_i = \overline{x_i} \oplus c_i \\ c_{i+1} = \underbrace{x_i \cdot c_i}_{abs} + x_i + c_i = x_i + c_i \end{cases} \Bigg\} \boxed{HAC^*}$$

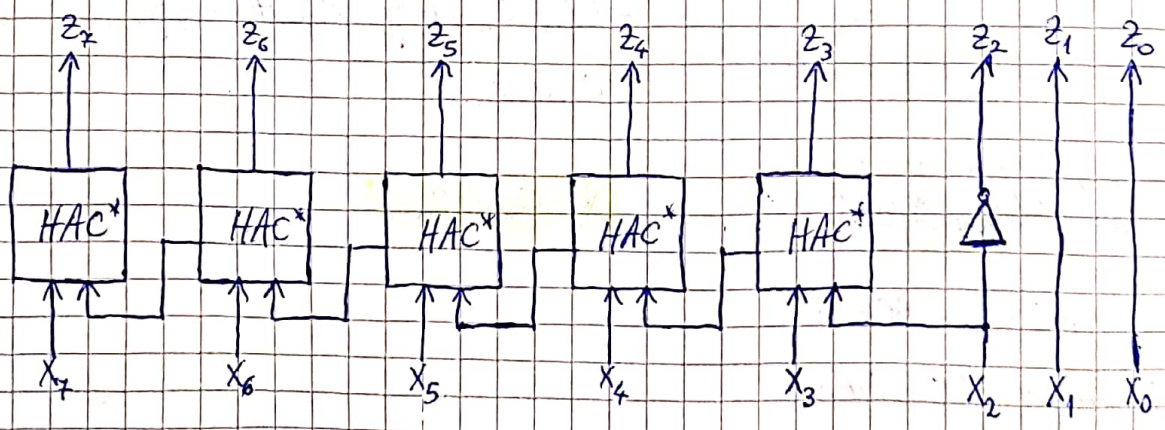




Se adune un nr de forma:

ex: $X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$

cu 1 1 1 0 1 1 0 0



1.2 Sumatoare zecimale cu propagare serială a transportului

1.2.1. BCD 8421

$X_i = x_3 x_2 x_1 x_0$

$Y_i = y_3 y_2 y_1 y_0$

$Z = z_3 z_2 z_1 z_0$

$$X_i + y_i = \begin{cases} z_i = \text{cifra sumă} \\ c_{i+1} \end{cases}$$

→ if $X_i + y_i < 10$: $\begin{cases} z_i = X_i + y_i \\ c_{i+1} = 0 \end{cases}$

→ if $X_i + y_i \geq 10$: $\begin{cases} z_i = X_i + y_i - 10 \\ c_{i+1} = 1 \end{cases}$

$X_i + y_i = c' z_3' z_2' z_1' z_0'$

$X_i + y_i \geq 10$ $\begin{cases} X_i + y_i \in [10, 15] & \cdot\ C1 \text{ (condiția 1)} \\ X_i + y_i \geq 16 & \cdot\ C2 \end{cases}$

- $C2$: $c' z_3' z_2' z_1' z_0' \geq 16 \Leftrightarrow c' = 1$
- $C1$: $10 \leq c' z_3' z_2' z_1' z_0' \leq 15 \Leftrightarrow c' = 0$ și $10 \leq z_3' z_2' z_1' z_0' \leq 15$

$z_3'z_2'$ \ $z_1'z_0'$	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12: 1	13: 1	15: 1	14: 1
10	8	9	11: 1	10: 1

$\overline{c}(z_3' \cdot z_2' + z_3' \cdot z_1') = 1$

$C_1 \cup C_2$: $c' + \underbrace{\overline{c'}}_{\text{ales}}(z_3' z_2' + z_3' z_1') = 1 \Rightarrow c' + z_3' z_2' + z_3' z_1' = 1$ pt. $X_i + y_i \geq 10$

Pt. scădere:

$(X_i + y_i - 10) \bmod 2^4 = (X_i + y_i - 10 + 16) \bmod 2^4 = (X_i + y_i + 6) \bmod 2^4$

* Scăderea lui 10 pe 4 cifre este echivalentă cu adunarea lui 6.

ex: $(8+7+6) = 5 \Leftrightarrow (8+7+6) \bmod 2^4 = 21 \bmod 2^4 = 5$

În binar $10_{10} = 1010_2$ și în C2 este $0110_{C_2} = 6_{10}$

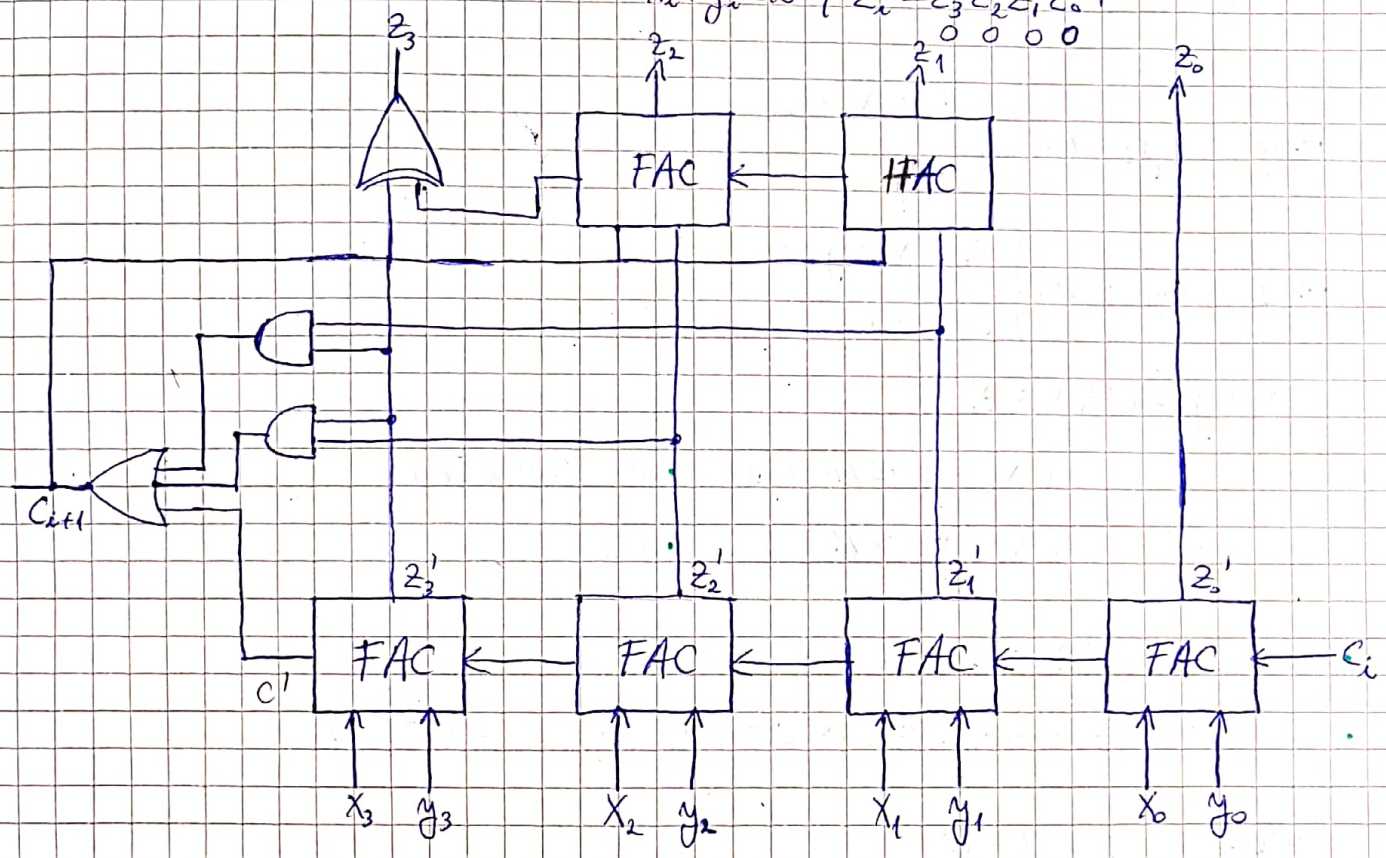
▷ Corecția lui z_i':

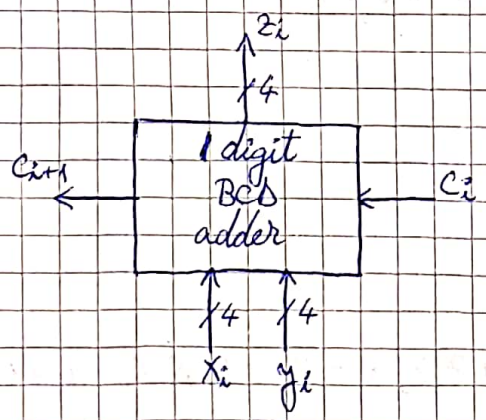
$C_{i+1} = c' + z_3' z_2' + z_3' z_1'$

$\rightarrow 1$, $x_i + y_i \geq 10$: $\begin{cases} C_{i+1} = 1 \\ z_i = z_3 z_2 z_1 z_0 + 0110 \end{cases}$

$\rightarrow 0$, $x_i + y_i < 10$: $\begin{cases} C_{i+1} = 0 \\ z_i = z_3' z_2' z_1' z_0' + 0000 \end{cases}$

$\rightarrow z_3' z_2' z_1' z_0' + 0\, C_{i+1} C_{i+1} 0$





ex:

	4	7	5 +
	2	9	8
7	17	13	
10	7	10	3

În BCD:

$$\begin{array}{r} 0100 + \\ 0010 \\ \hline 0111 \end{array} \qquad \begin{array}{r} 0111 + \\ 1001 \\ \hline 10001 \geq 16 \\ 0110 \\ \hline 0111 \\ 7 \end{array} \qquad \begin{array}{r} 0101 + \\ 1000 \\ \hline 1101 + \geq 10 \\ 0110 \\ \hline 10011 \\ 3 \end{array}$$

1.2.2. Excas de 3

$X_{iE_3} = x_3 x_2 x_1 x_0 \qquad Y_{iE_3} = y_3 y_2 y_1 y_0 \qquad Z_{iE_3} = z_3 z_2 z_1 z_0$

$X_{iE_3} = x_i + 3 \qquad Y_{iE_3} = y_i + 3 \qquad Z_{iE_3} = z_i + 3$

$$X_{iE_3} + Y_{iE_3} \begin{cases} Z_{iE_3} \\ C_{i+1} \end{cases}$$

→ if $X_i + Y_i < 10$: $\begin{cases} C_{i+1} = 0 \\ z_i = x_i + y_i \big|_{+6} \end{cases} \qquad Z_{iE_3} = X_{iE_3} + Y_{iE_3} - 3 \equiv +13$

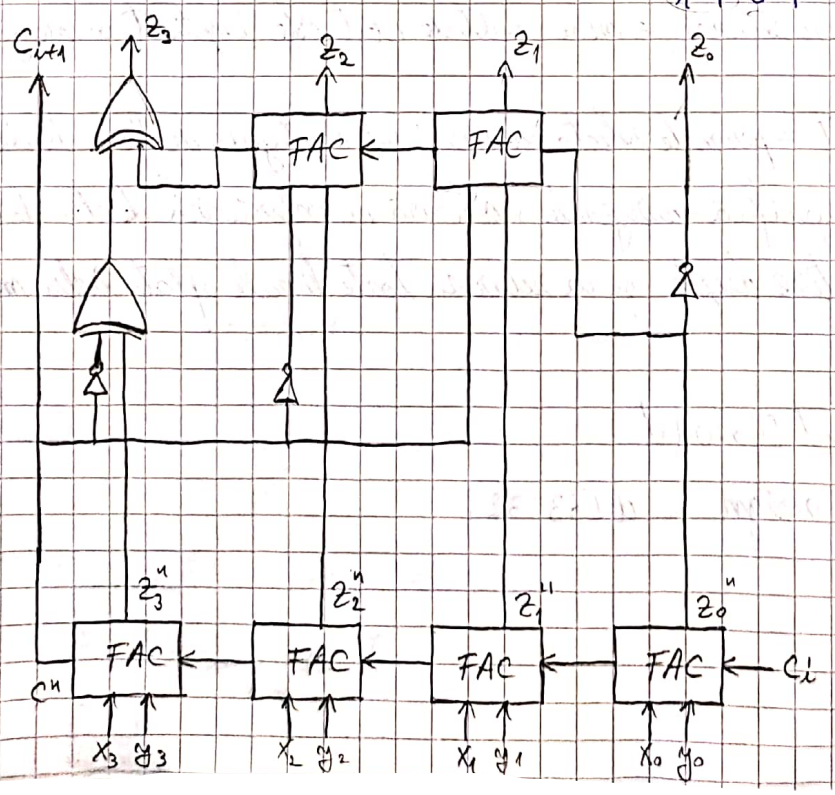
→ if $X_i + Y_i \geq 10$: $\begin{cases} C_{i+1} = 1 \\ z_i = x_i + y_i - 10 \big|_{+6} \end{cases} \qquad Z_{iE_3} = X_{iE_3} + Y_{iE_3} - 13 \equiv +3$

$$\left.\begin{array}{l} X_{iE_3} + Y_{iE_3} = C'' z_3'' z_2'' z_1'' z_0'' \\ x_i + y_i \geq 10 \big|_{+6} \Rightarrow X_{iE_3} + Y_{iE_3} \geq 16 \end{array}\right\} C'' = 1$$

Deci C'' → 1 ($x_i + y_i \geq 10$): $\begin{cases} C_{i+1} = 1 \\ Z_{iE_3} = z_3'' z_2'' z_1'' z_0'' + \\ \quad 0\ 0\ 1\ 1 \end{cases}$

→ 0 ($x_i + y_i < 10$): $\begin{cases} C_{i+1} = 0 \\ Z_{iE_3} = z_3'' z_2'' z_1'' z_0'' + \\ \quad 1\ 1\ 0\ 1 \end{cases}$

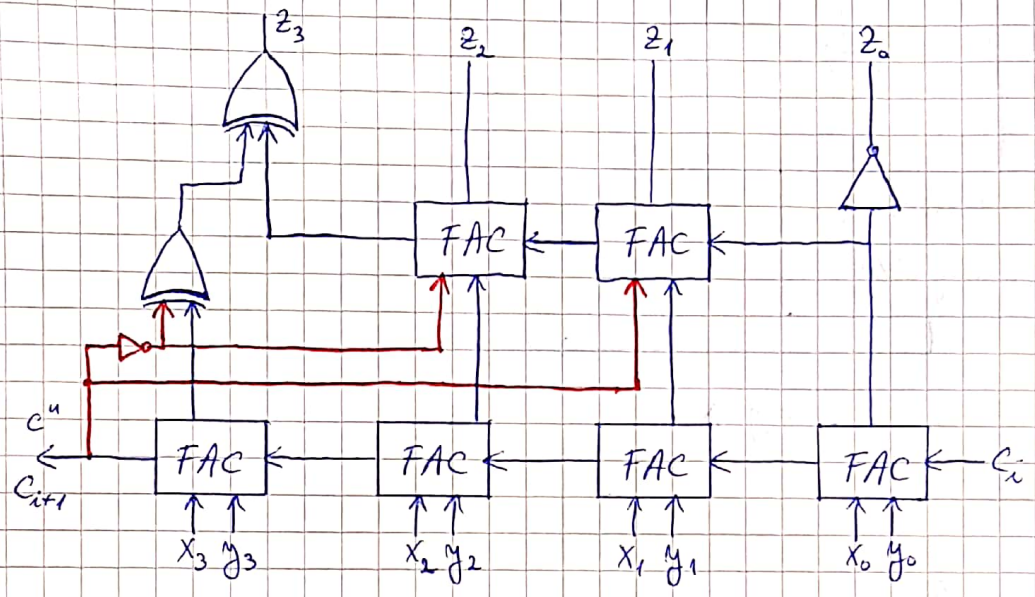
→ $\boxed{z_3'' z_2'' z_1'' z_0'' + \overline{C_{i+1}}\, \overline{C_{i+1}}\, C_{i+1}\, 1}$



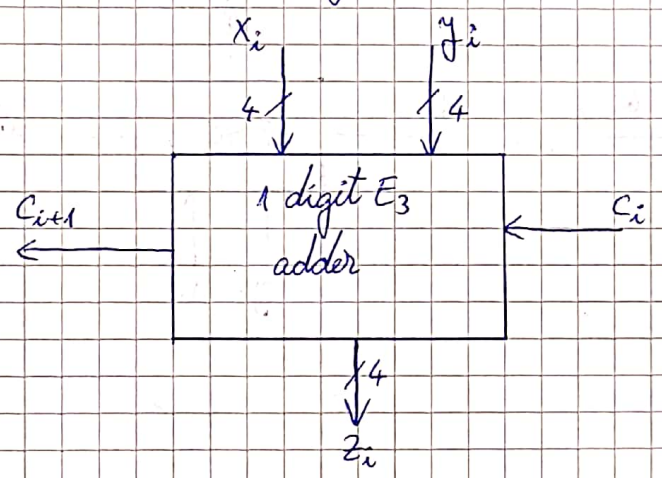
Curs 3

1.2.2 z_{iE_3}'s correction stage

$$z_{iE_3} = z_3^u z_2^u z_1^u z_0^u + \overline{c}_{i+1}\overline{c}_{i+1}c_{i+1}1$$



$\Updownarrow$



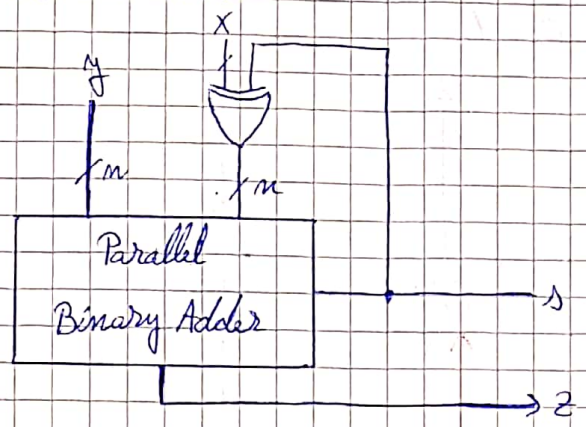
Ex: 475 +
298
773

$\Leftrightarrow$

1	1	
0111	1010	1000 +
0101	1100	1011
0 1101	1 0111	1 0011
↳ 1101	↳ 0011	↳ 0011
1010	1010	0110
7	7	3

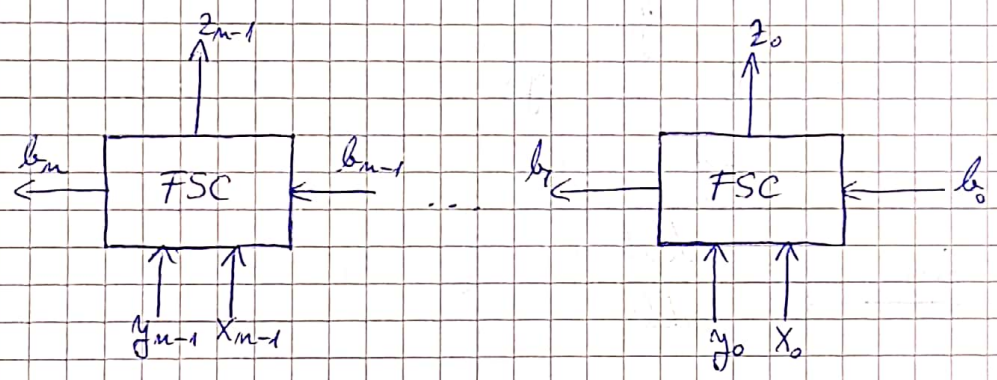
1.3 Scăzătoare binare și zecimale

Ⓐ Sumatorul scăzător

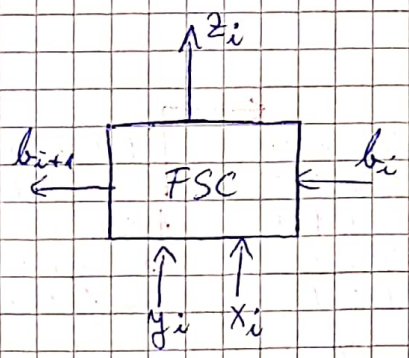


$$s \begin{cases} 1: z = y + \bar{x} + 1 = y - x \\ 0: z = y + x \end{cases}$$

Ⓑ Full Substractor Cell



$\Updownarrow$

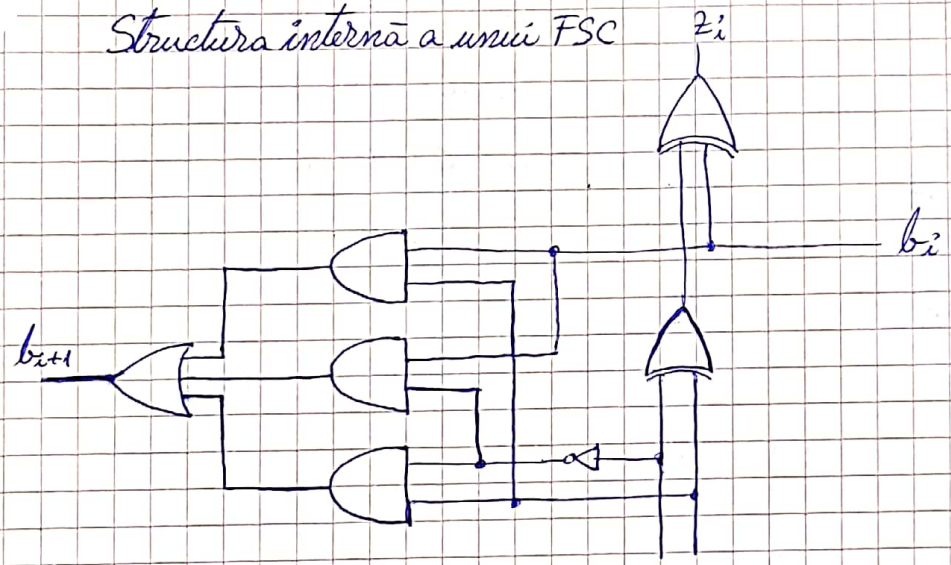


unde $y_i - x_i - b_i \begin{cases} \to z_i \\ \to b_{i+1} \end{cases}$

Input			Output	
y_i	x_i	b_i	z_i	b_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$y_i - x_i - b_i = \begin{cases} z_i = y_i \oplus x_i \oplus b_i \\ b_{i+1} = \bar{y}_i x_i + \bar{y}_i b_i + x_i b_i \end{cases}$$

Structura internă a unui FSC



© Scăderea numerelor în BCD

$y^{(k)}, x^{(k)} \rightarrow$ BCD cu k digits (se reprez. pe 4k biți)

$y^{(k)} - x^{(k)}$

$y^{(k)} = y_{k-1} y_{k-2} \dots y_1 y_0$

$x^{(k)} = x_{k-1} x_{k-2} \dots x_1 x_0$

complementul de 9 al nr. x pe k cifre

$\overline{x}^{*(k)} = \overline{x}^*_{k-1} \overline{x}^*_{k-2} \dots \overline{x}^*_1 \overline{x}^*_0$, $x_i^* = 9 - x_i$

$\overline{x}^{*(k)} = \underbrace{99\dots99}_{k} - x_{k-1}x_{k-2}\dots x_1 x_0 = 10^k - 1 - x^{(k)}$ ← nu generează borrow

$$(y^{(k)} - x^{(k)}) \bmod 10^k = (y^{(k)} \underbrace{+ 10^k - 1 - x^{(k)}}_{\overline{x}^{*(k)}} + 1) \bmod 10^k \Rightarrow (y^{(k)} + \overline{x}^{*(k)} + 1) \bmod 10^k$$

! Obținem compl de 9 doar având compl de 9 al fiecărei cifre

$\overline{X_i^*} = X_3^* X_2^* X_1^* X_0^* = 1001 - X_3 X_2 X_1 X_0$

$X_i = X_3 X_2 X_1 X_0$

Input

X_3	X_2	X_1	X_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

etc.

pt. 1010 - 1111 avem dddd

Output

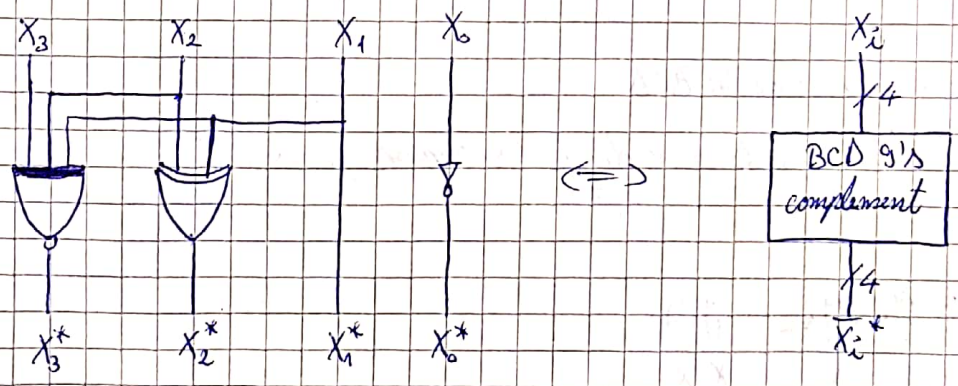
X_3^*	X_2^*	X_1^*	X_0^*
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0

$\Rightarrow X_3^* = \overline{X_3 + X_2 + X_1}$

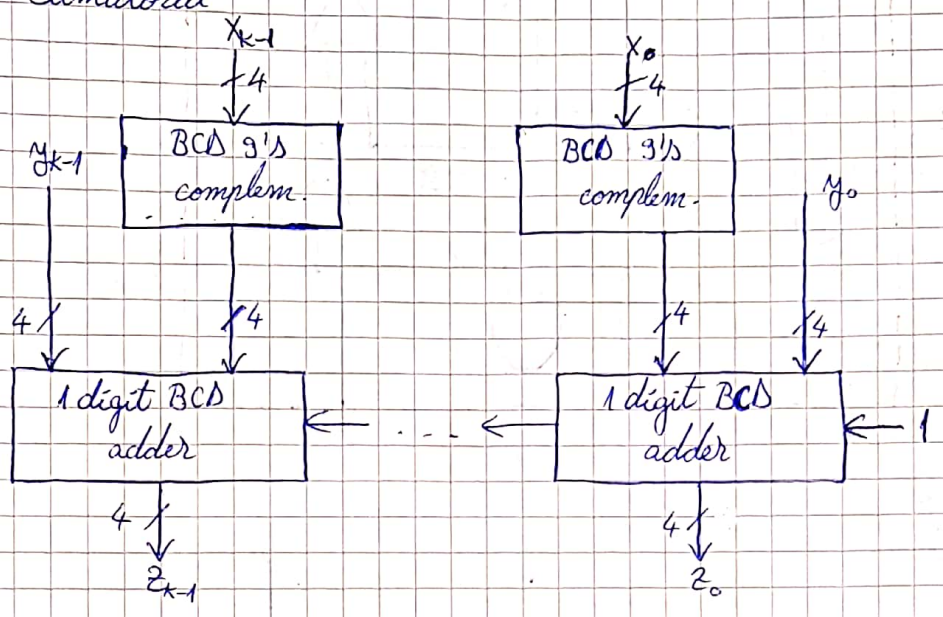
$X_2^* = X_2 \oplus X_1$

$X_1^* = X_1$

$X_0^* = \overline{X_0}$



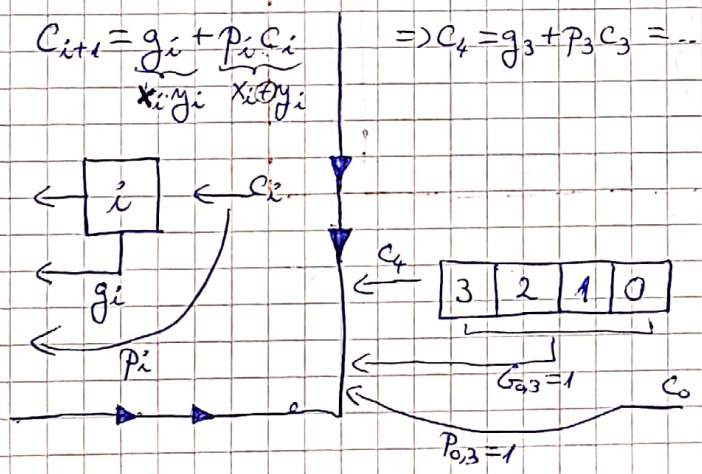
Sumatorul



1.4. Metode de calculare a sumei de performanță ridicată

1.4.1. Sumatoare cu anticipare a transportului multi-nivel (Multi-level Full CLA)

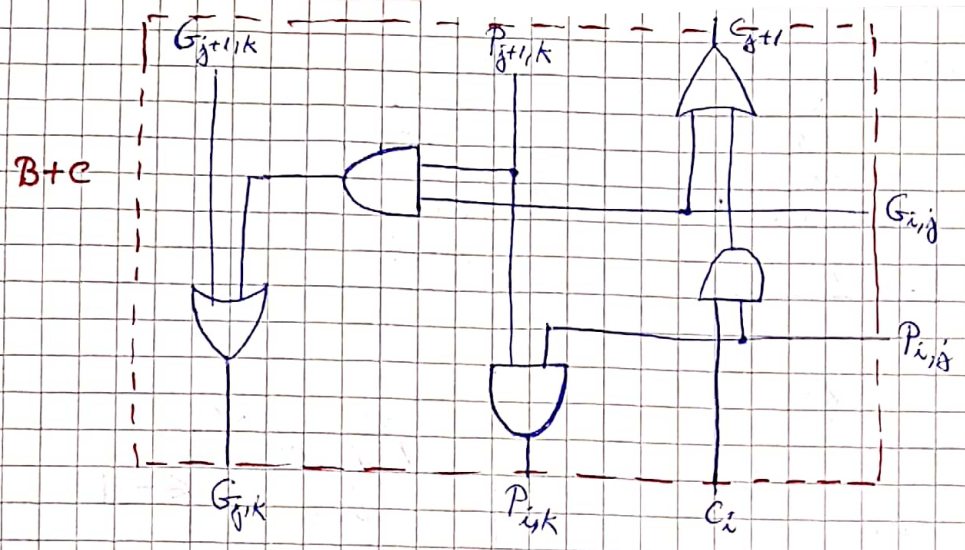
$$C_{i+1} = \underbrace{g_i}_{x_i \cdot y_i} + \underbrace{p_i \cdot c_i}_{x_i \oplus y_i} \Rightarrow C_4 = g_3 + p_3 c_3 = \dots = g_3 + p_3 g_2 + p_3 p_2 g_1 + \underbrace{p_3 p_2 p_1 g_0}_{G_{0,3}} + \underbrace{p_3 p_2 p_1 p_0}_{P_{0,3}} c_0$$

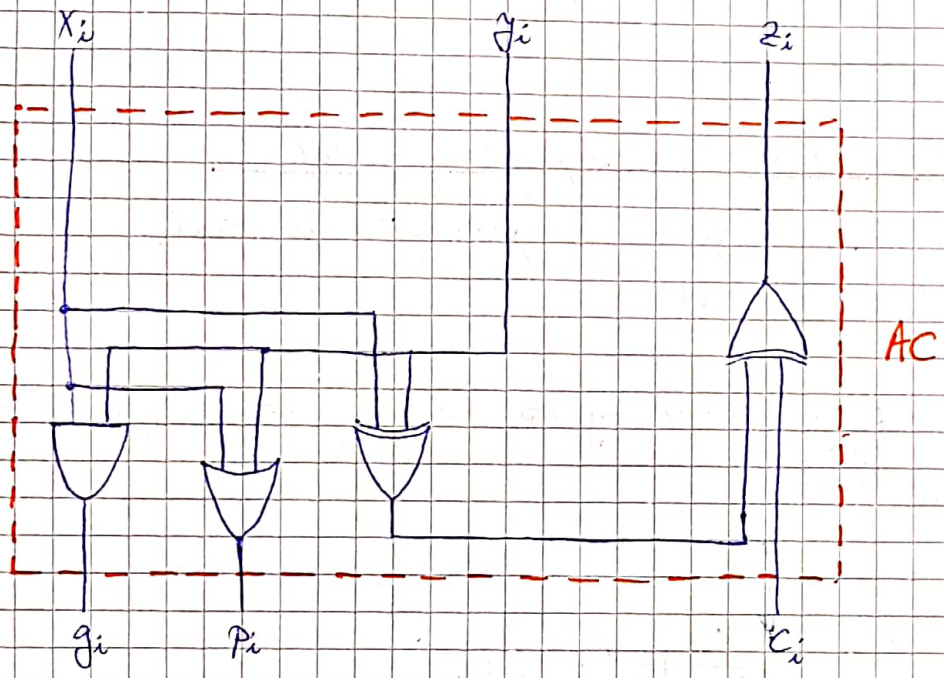


$$G(0,3) + P(0,3) = \underbrace{g_3 + p_3 g_2}_{G_{2,3}} + \underbrace{p_3 p_2}_{P_{2,3}} \cdot \underbrace{(g_1 + p_1 g_0)}_{G_{0,1}} + \underbrace{\underbrace{p_3 p_2}_{P_{2,3}} \cdot \underbrace{p_1 p_0}_{P_{1,0}}}_{P_{0,3}} \cdot c_0$$

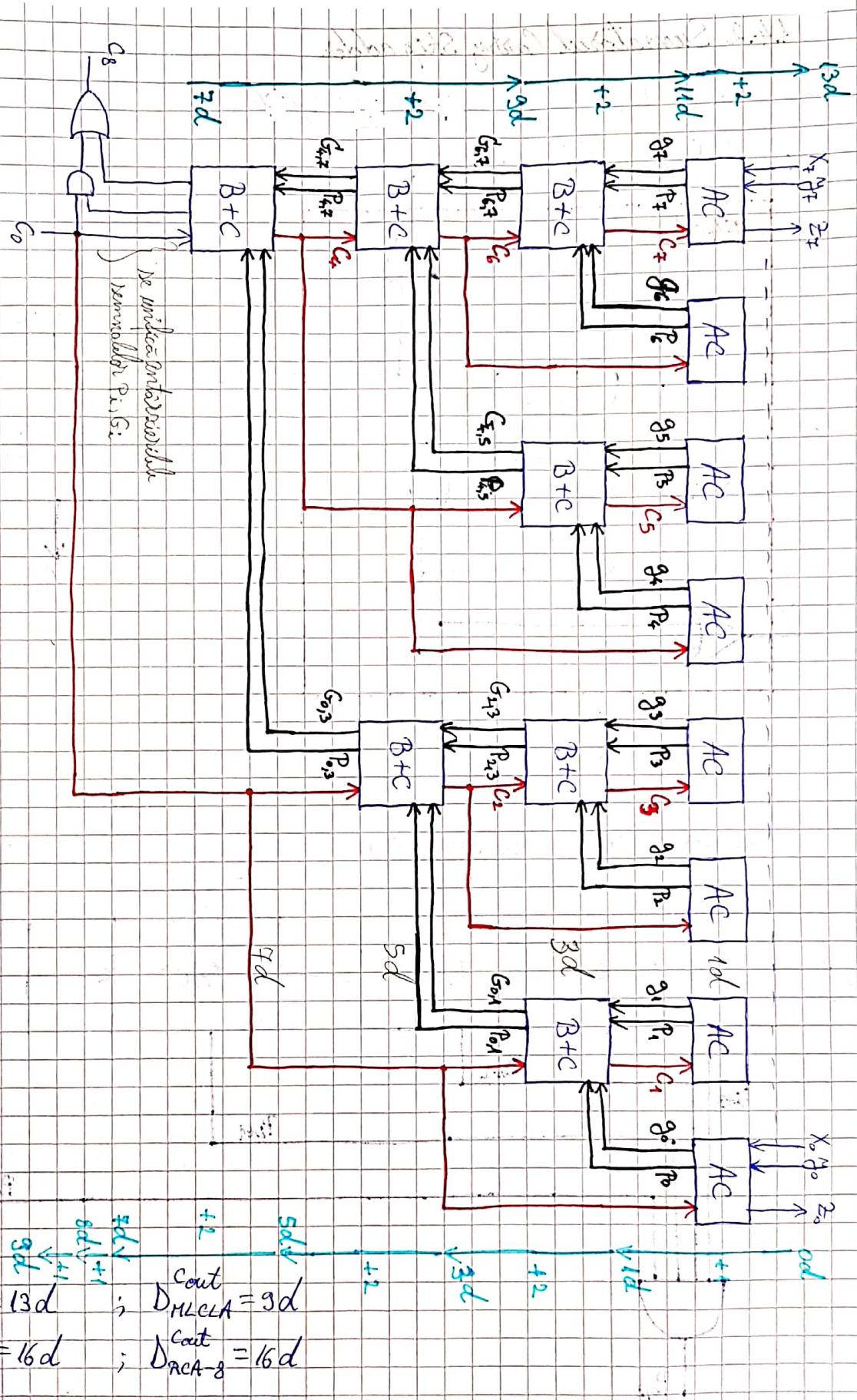
În general:

$$\begin{cases} C_{k+1} = G_{i,k} + P_{i,k} \cdot C_i, & i \le k \\ G_{i,k} = G_{j+1,k} + P_{j+1,k} \cdot G_{i,j}, & i \le j < k \\ P_{i,k} = P_{j+1,k} \cdot P_{i,j}, & i \le j < k \end{cases}$$





$$\begin{cases} P_{i,i} = p_i \\ G_{i,i} = g_i \end{cases}$$



$$D^{C_{out}}_{HLCLA} = D_{AC_{carry}} + 3\,D_{B+C_{carry}} + D_{AND} + D_{OR} = 9d$$

$$D^{z}_{HLCLA} = D_{AC_{carry}} + 5 \cdot D_{B+C_{carry}} + D_{AC_{z_i}} = 13d$$

$D^{z}_{HLCLA} = 13d$; $D^{C_{out}}_{HLCLA} = 9d$

$D^{z}_{RCA\text{-}8} = 16d$; $D^{C_{out}}_{RCA\text{-}8} = 16d$

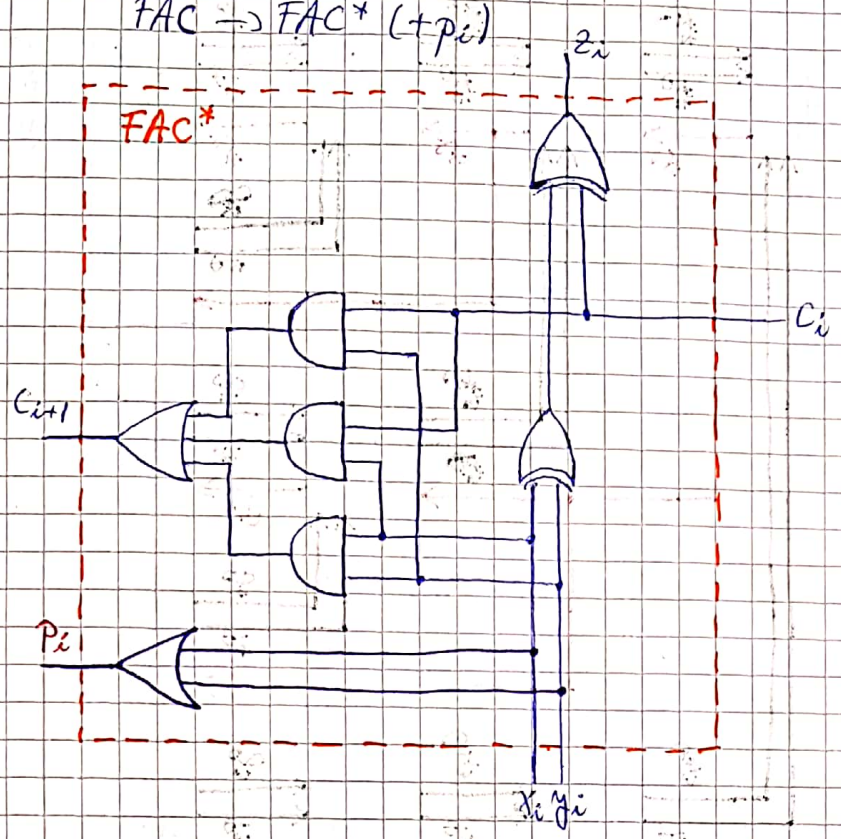
In general: $D^{z}_{HLCLA} = (4\lceil \log_2 n \rceil + 1)\,d$

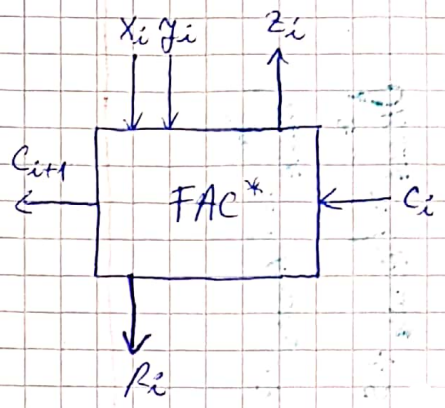
$D^{C_{out}}_{HLCLA} = (2\lceil \log_2 n \rceil + 3)\,d$

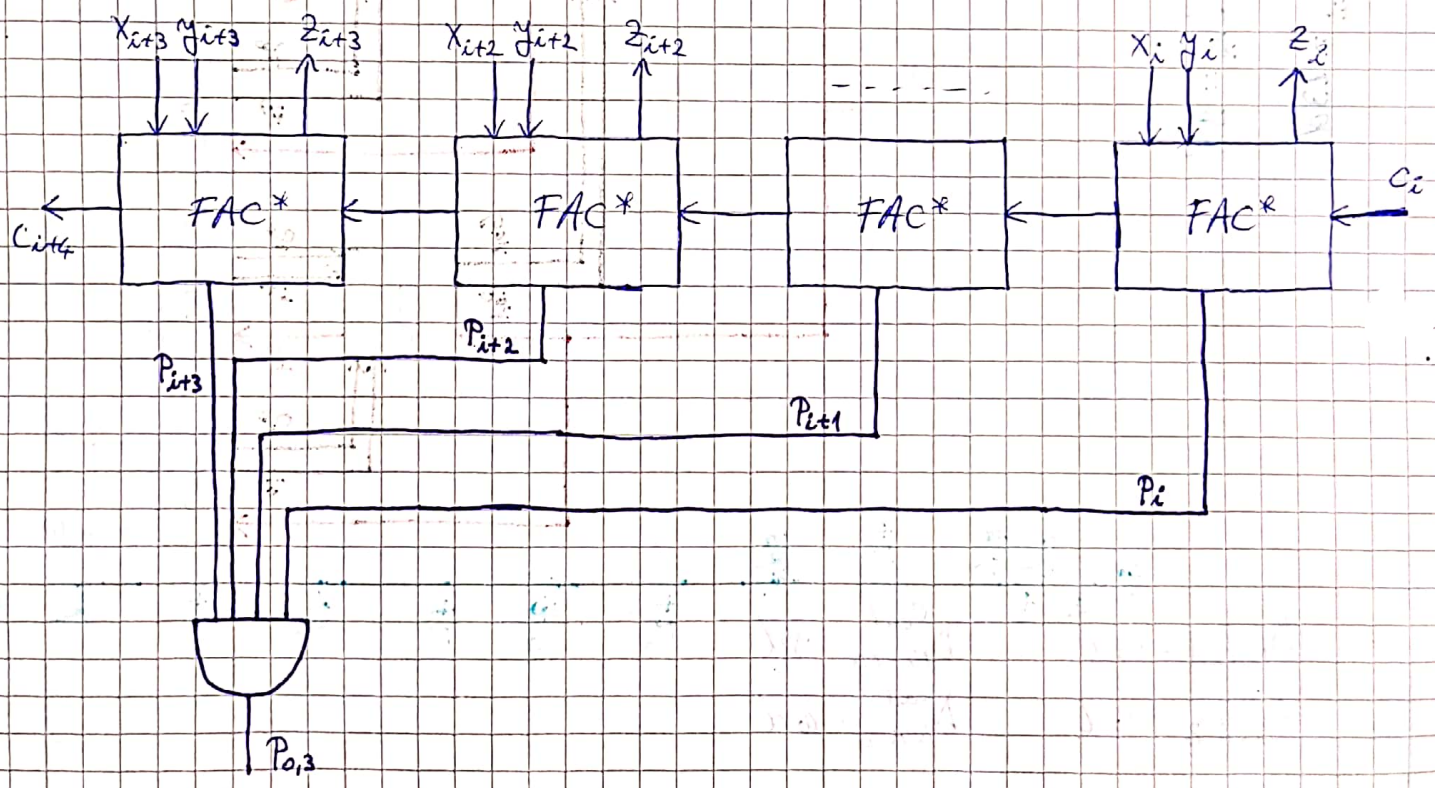
1.4.2 Sumatorul Carry Skip adder

$G_{0,3}$, $P_{0,3}$ → se generează mai repede

FAC → FAC* (+ p_i)

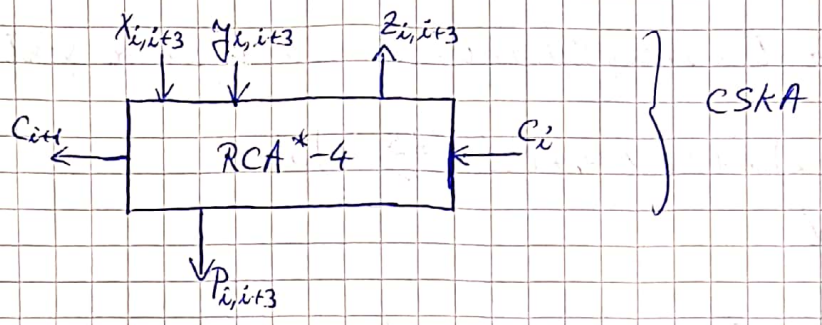






Curs 4

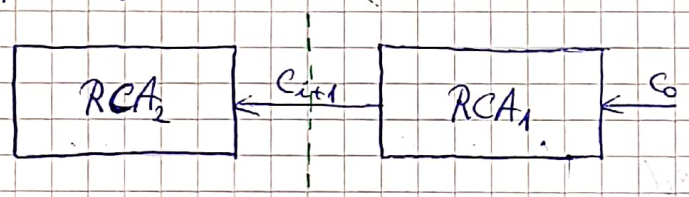
1.4.2 Sumatorul Carry Skip Adder



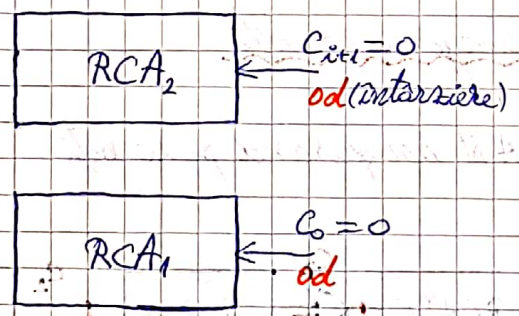
CMOS - pre-discharge

Toate liniile de carry sunt construite cu facilitatea de pre-discharge (sunt aduse inițial la val. 0)

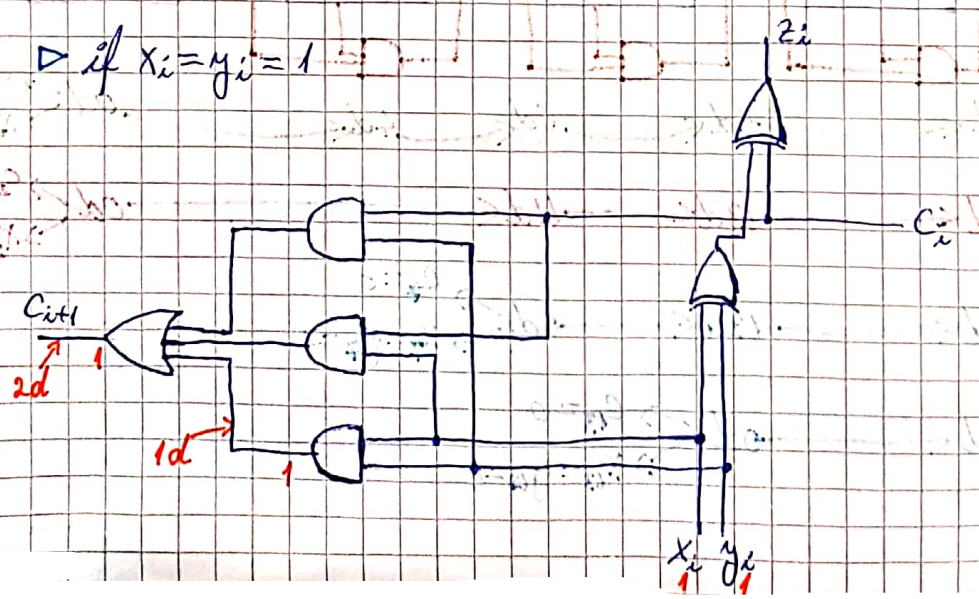
▷ if $x_i = y_i = 0 \Rightarrow c_{i+1} = 0 \Rightarrow$ sumatorul se divide



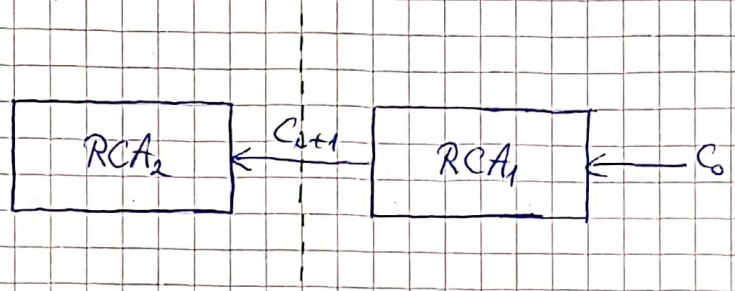
Linia c_{i+1} o să aibă val. corectă de la început, adică RCA_1 și RCA_2 lucrează paralel, nu mai lucrează serial.



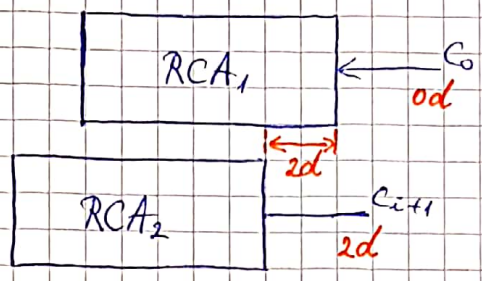
▷ if $x_i = y_i = 1$



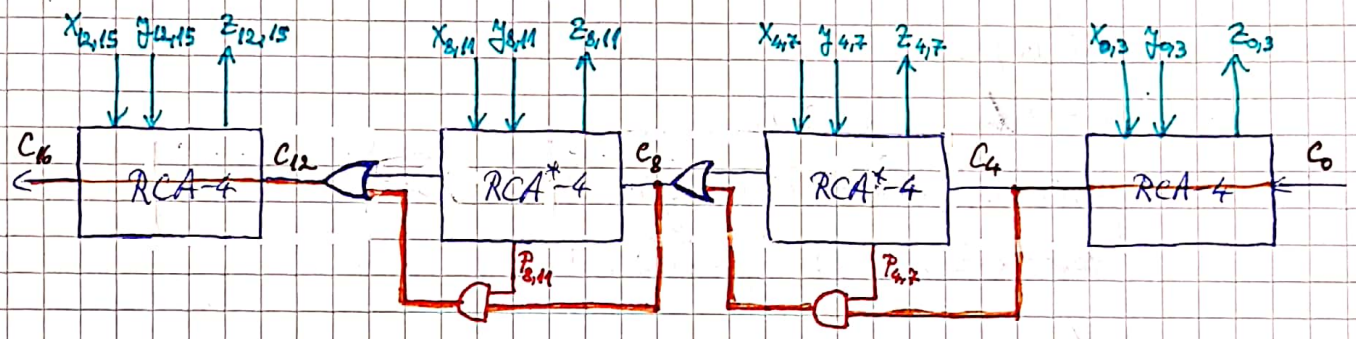
$\Rightarrow C_{i+1} = 1, 2d$



Se obține:



- Sumatorul CSKA pe 16 biți



≡ - propagarea $C_0 \to C_{16}$

- Întârzierea C_{16}:

$$D^{C_{out}}_{CSKA} = \underbrace{4 \cdot 2d}_{C_0 \to C_4} + \underbrace{2d}_{C_4 \to C_8} + \underbrace{2d}_{C_8 \to C_{12}} + \underbrace{4 \cdot 2d}_{C_{12} \to C_{16}} = 20d$$

- Întârzierea z:

$$D^{z}_{CSKA} = \underbrace{4 \cdot 2d}_{C_4} + \underbrace{2d}_{C_8} + \underbrace{2d}_{C_{12}} + \underbrace{3 \cdot 2d}_{C_{15}} + \underbrace{2d}_{z_{15}} = 20d$$

Pt. ca propagarea să nu aibă loc prin sunt trb. ca:

$P_{4,7} \cdot C_4 = 0$ $\to C_4 = 0$

$\to P_{4,7} = 0 \equiv P_4 P_5 P_6 P_7 = 0$

Deci $p_i = 0, i \in [4,7] \equiv x_i + y_i = 0 \equiv x_i = y_i = 0$

• Care e lungimea optimă pt. a avea o latență cât mai mică?

Cd: – n biți (operanzi pe ~)

– b biți cu care lucrăm (nu 4 ca mai devreme)

– c. m. semnificant și insemnificant RCA nu are logică de șunt

• $D_{CSKA}^{z_{cout}} = 2bd + 2\left(\frac{n}{b} - 2\right)d + 2bd = 4bd + \left(\frac{2n}{b} - 4\right)d =$

$= \left(4b + \frac{2n}{b} - 4\right)d$

Care e val. lui b pt. ca funcția să fie minimă?

$$\frac{dD_{CSKA}^{z_{out}}}{db} = 4 - \frac{2n}{b^2} \Rightarrow b_{optim} = \frac{\sqrt{2n}}{2}$$

• $D_{CSKA_{opt.}}^{z/cout} = 4(\sqrt{2}\,n - 1)d$

↳ pt. cazul optim

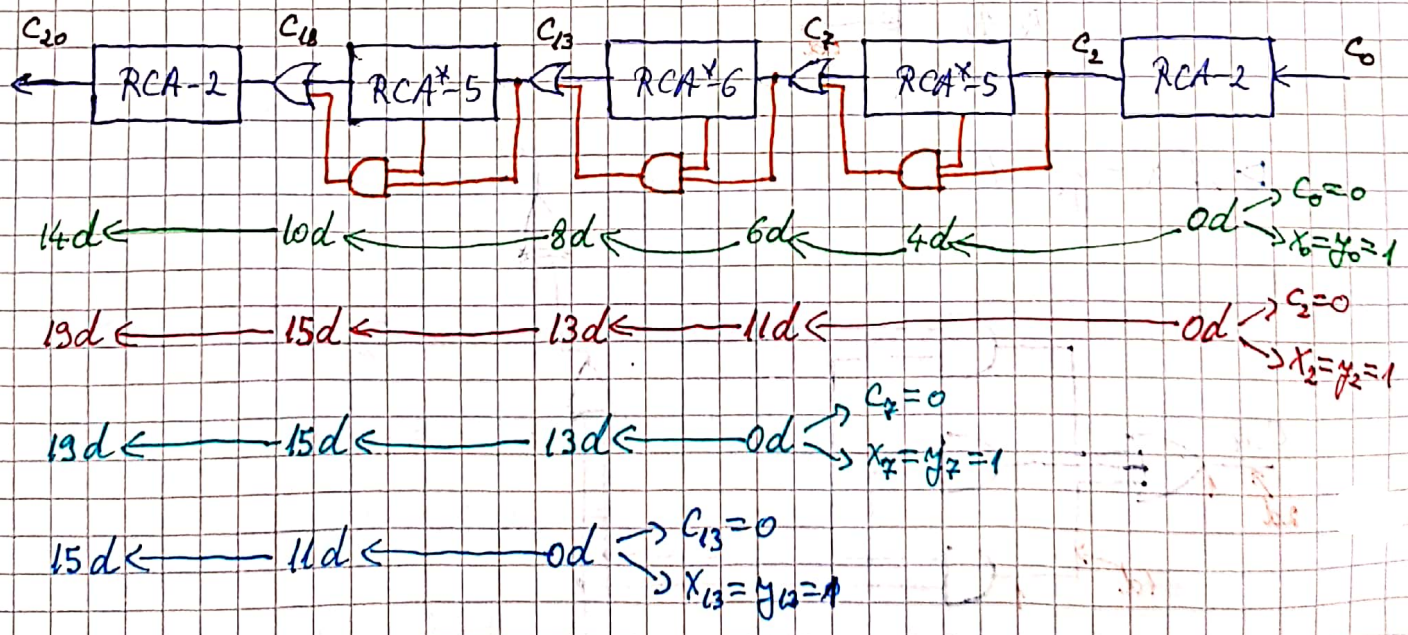
Ex: $n = 32$; $D_{CSKA_{opt}}^{z/cout} = 28d$

$D_{RCA}^{z/cout} = 64d$

* Utilizarea segmentelor RCA de lungimi variabile

Posibil exercițiu de examen

Pp. că avem de construit un RCA pt. un operand pe 20 biți



Dacă toate segmentele ar avea logică de șunt, ar forța ca c să treacă doar prin logica de șunt și rezultatul ar depinde de c_0, așa se propagă serial în unele segmente

$$z_6 = \underbrace{4d}_{c_2} + 10d = 14d$$

$$z_{12} = \underbrace{11d}_{c_7} + 12d = 23d$$

$$z_{17} = \underbrace{13d}_{c_{13}} + 10d = 23d$$

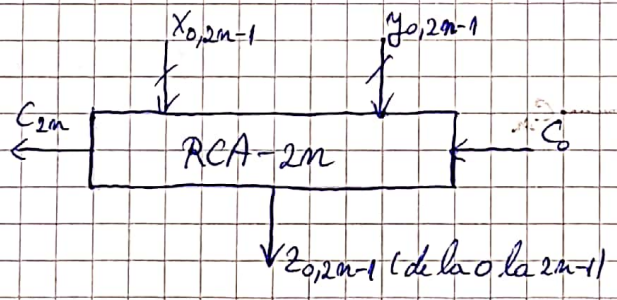
$$z_{19} = \underbrace{15d}_{c_{18}} + 4d = 19d$$

1.4.3 Carry Select Adder (CSeA)

- se bazează pe principiul Sumă condiționată de transport, adică pt. a calcula c_{i+1} și z_i avem nevoie de c_i

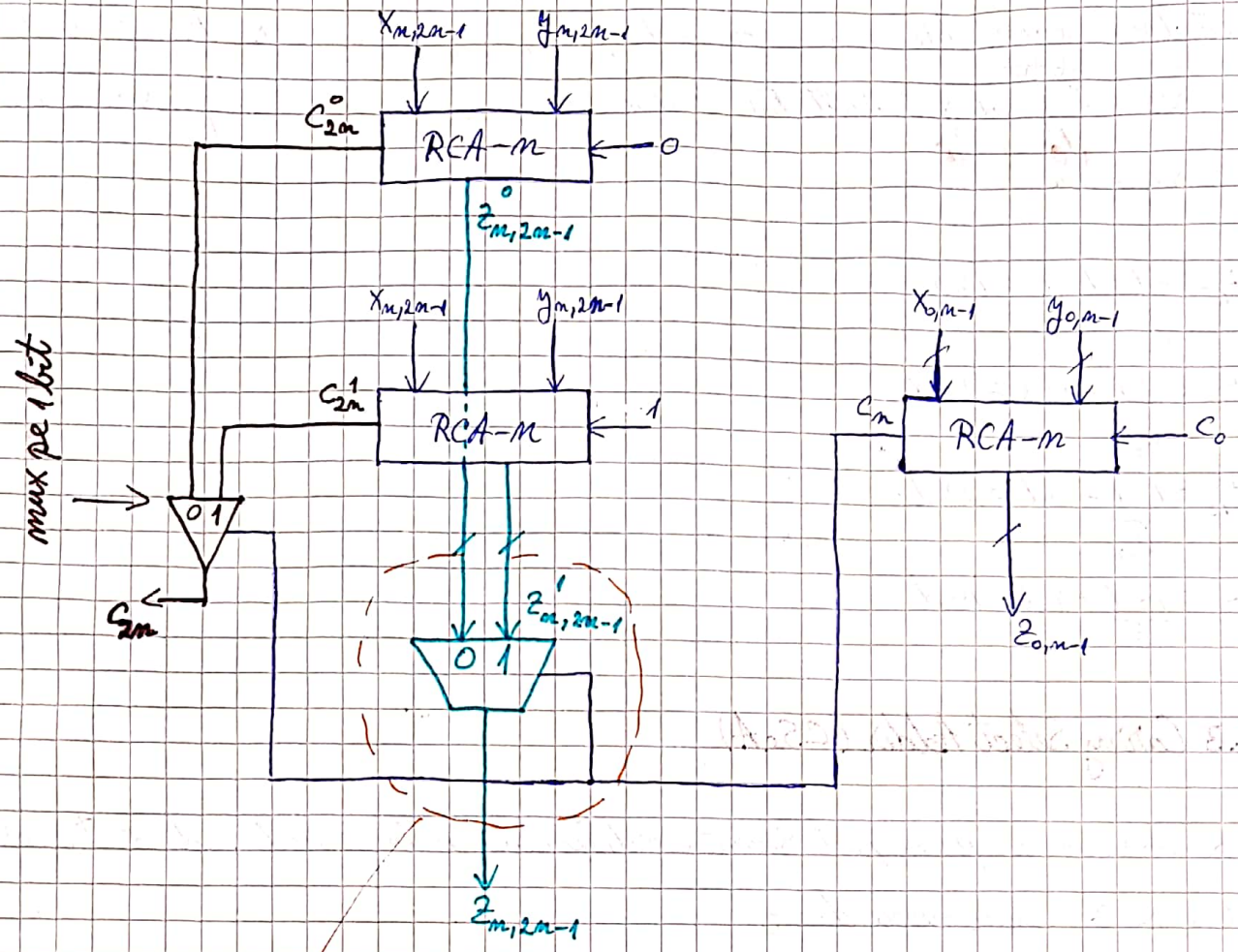
$$c_i \begin{cases} \to 0 \Rightarrow (z_i^0, c_{i+1}^0) \\ \to 1 \Rightarrow (z_i^1, c_{i+1}^1) \end{cases}$$

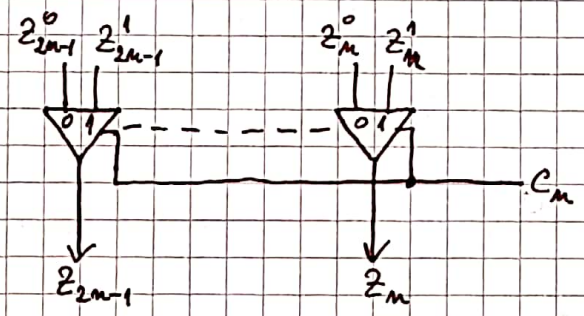
Folosim un RCA-2n

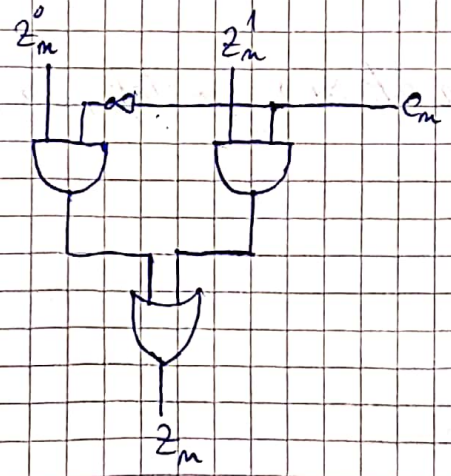


$$D_{RCA}^{z/cout} = 4nd$$

CSeA împarte sumatorul pe 2n biți în 2 jumătăți, iar pe cea mai semnificativă o deplică


$x_{n,2n-1}$
$y_{n,2n-1}$
c^0_{2n}
RCA-n
0
$z^0_{n,2n-1}$
$x_{n,2n-1}$
$y_{n,2n-1}$
c^1_{2n}
RCA-n
1
mux pe 1 bit
0 1
c_{2n}
$z^1_{n,2n-1}$
0 1
$z_{n,2n-1}$
$x_{0,n-1}$
$y_{0,n-1}$
c_n
RCA-n
c_0
$z_{0,n-1}$


z^0_{2n-1} z^1_{2n-1}
z^0_n z^1_n
0 1
0 1
c_n
z_{2n-1}
z_n


z^0_n
z^1_n
c_n
z_n

Deci: $\boxed{D_{CSeA}^{z/cout} = (2n+2)d \simeq \frac{1}{2} D_{RCA}^{z/cout}}$

- Optimizarea de arie a lui c_{2n}

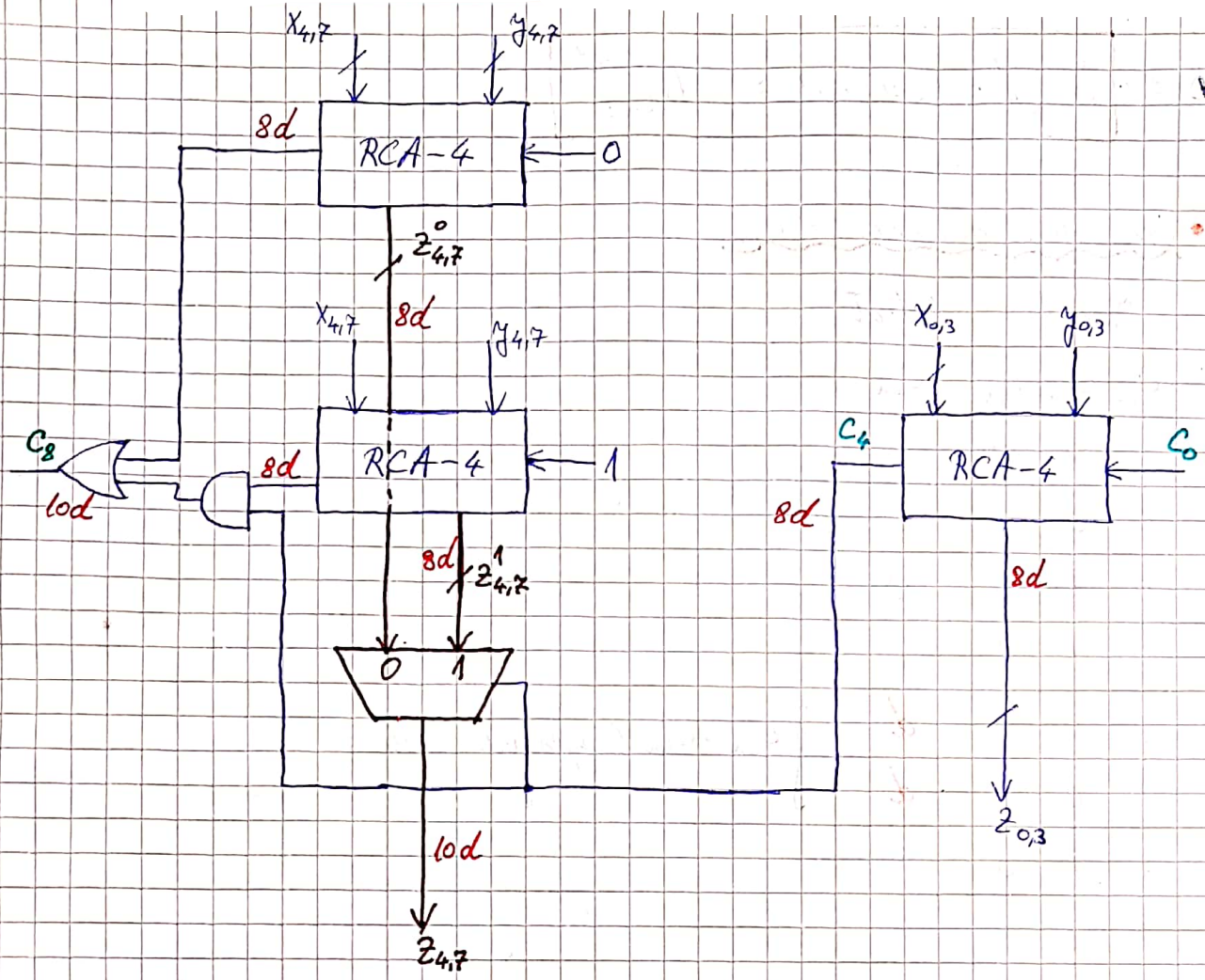
Input			Output
c_{2n}^0	c_{2n}^1	c_n	c_{2n}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	d
1	0	1	d
1	1	0	1
1	1	1	1

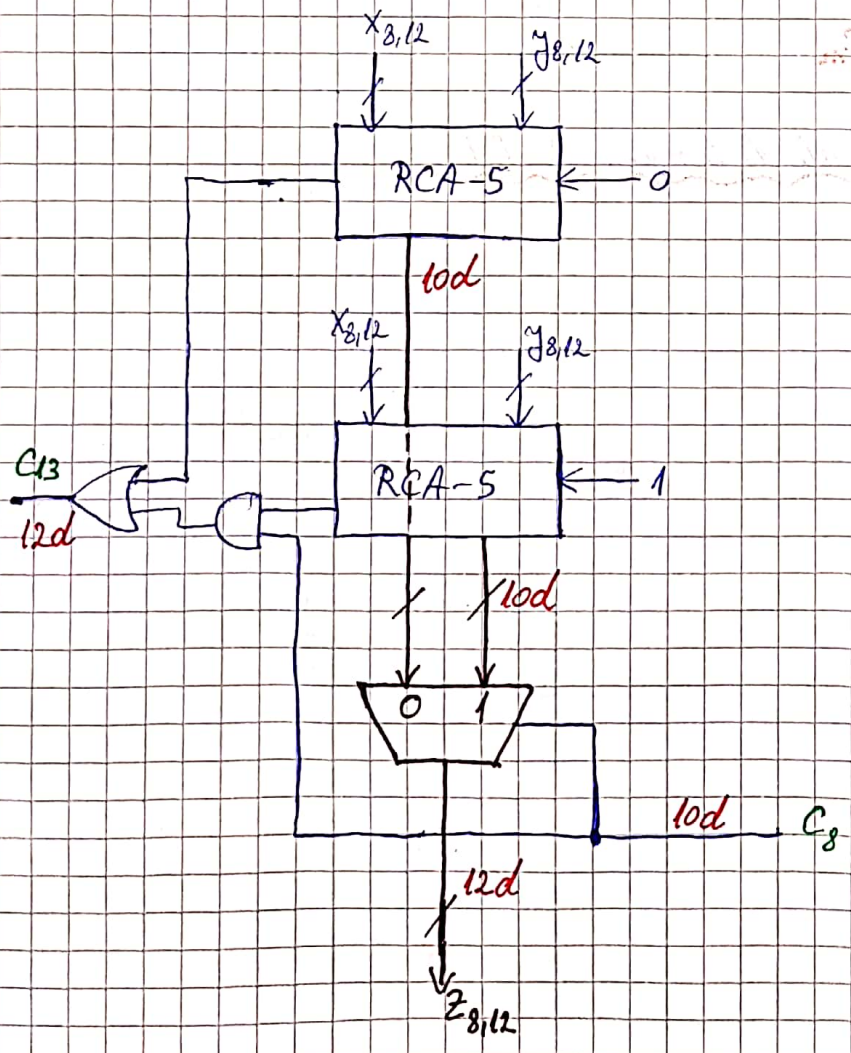
$\left.\begin{matrix} d \\ d \end{matrix}\right\}$ $c_{2n}^0 > c_{2n}^1$ (imposibil practic)

Deci $c_{2n} = c_{2n}^0 + c_{2n}^1 \cdot c_n$

- Sumatoare CSeA cu segmente de lungime variabilă

ex: pe 13 biți


$x_{4,7}$
$y_{4,7}$
8d
RCA-4
0
$z^0_{4,7}$
8d
$x_{4,7}$
$y_{4,7}$
C_8
10d
8d
RCA-4
1
8d
$z^1_{4,7}$
0
1
10d
$z_{4,7}$
$x_{0,3}$
$y_{0,3}$
C_4
RCA-4
C_0
8d
8d
$z_{0,3}$


$x_{8,12}$
$y_{8,12}$
RCA-5
0
10d
$x_{8,12}$
$y_{8,12}$
C_{13}
12d
RCA-5
1
10d
0
1
10d
C_8
12d
$z_{8,12}$

1.4.4. Conditional Sum Adder (CSuA)

- este un CSeA multinivel

1st level $\begin{cases} (z_i^0, c_{i+1}^0) \\ (z_i^1, c_{i+1}^1) \end{cases}$

ex: Cd. $X = x_3 x_2 x_1 x_0$ si $\boxed{c_0 = 0}$

$Y = y_3 y_2 y_1 y_0$

$$\left.\begin{aligned} c_{i+1} &= \underbrace{x_i y_i} + x_i c_i + y_i c_i \\ &= x_i y_i c_i + x_i y_i \bar{c}_i \end{aligned}\right\} \Rightarrow$$

$$\Rightarrow c_{i+1} = x_i y_i \bar{c}_i + x_i c_i (1 + y_i) + y_i c_i$$

$$c_{i+1} = \underbrace{x_i y_i}_{g_i} \bar{c}_i + \underbrace{(x_i + y_i)}_{p_i} c_i$$

$$c_{i+1} = g_i \bar{c}_i + p_i c_i$$

$$\bar{c}_{i+1} = (\bar{g}_i + c_i)(\bar{p}_i + \bar{c}_i) = \bar{g}_i \bar{p}_i + \bar{g}_i \bar{c}_i + c_i \bar{p}_i + c_i \bar{c}_i = \bar{g}_i \bar{c}_i + \bar{p}_i c_i$$

Sum

$c_0 = 0$: $z_0 = x_0 \oplus y_0 \oplus c_0 = x_0 \oplus y_0$

$z_1 = x_1 \oplus y_1 \oplus c_1 = (\overline{x_1 \oplus y_1}) c_1 + (x_1 \oplus y_1) \bar{c}_1$

$z_2 = x_2 \oplus y_2 \oplus c_2 = (\overline{x_2 \oplus y_2}) c_2 + (x_2 \oplus y_2) \bar{c}_2$

$z_3 = x_3 \oplus y_3 \oplus c_3 = (\overline{x_3 \oplus y_3})(g_2 \bar{c}_2 + p_2 c_2) + (x_3 \oplus y_3)(\bar{g}_2 \bar{c}_2 + \bar{p}_2 c_2) =$

$= [(\overline{x_3 \oplus y_3}) g_2 + (x_3 \oplus y_3) \bar{g}_2] \bar{c}_2 + [(\overline{x_3 \oplus y_3}) p_2 + (x_3 \oplus y_3) \bar{p}_2] c_2$

Carry

$c_1 = g_0 \bar{c}_0 + p_0 c_0 = g_0$

$c_2 = g_1 \bar{c}_1 + p_1 c_1$

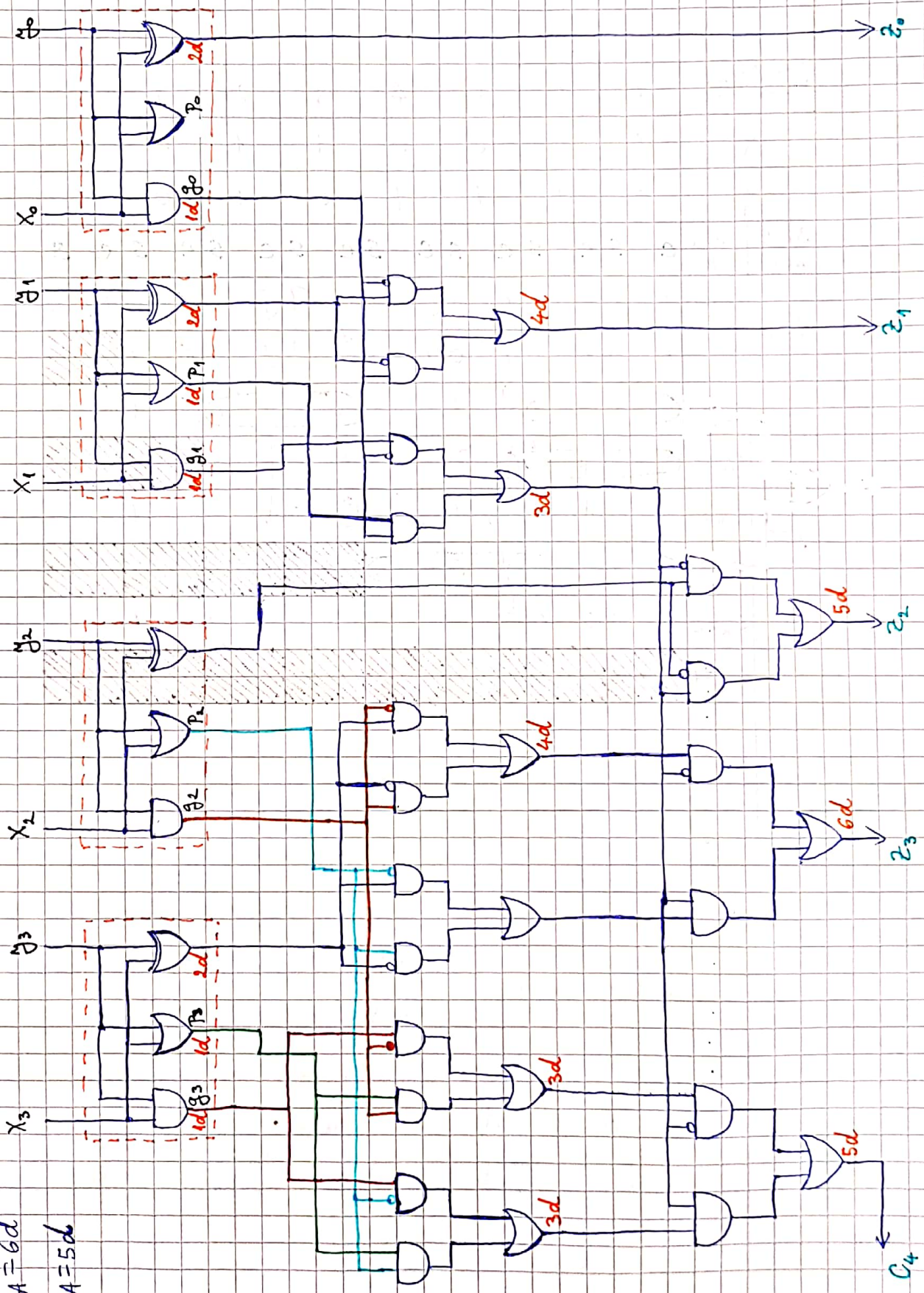
$c_3 = g_2 \bar{c}_2 + p_2 c_2$

$\bar{c}_3 = \bar{g}_2 \bar{c}_2 + \bar{p}_2 c_2$

$c_4 = g_3 \bar{c}_3 + p_3 c_3 = (g_3 \bar{g}_2 + p_3 g_2) \bar{c}_2 + (g_3 \bar{p}_2 + p_3 p_2) c_2$

Curs 5

1.4.4. Conditional Sum Adder (CSuA)



Analiza:

$D_{CSuA} = 6d$

$D_{aSuA} = 5d$

Ex: $x = 1\,0\,0\,0\,1\,1\,0\,1$

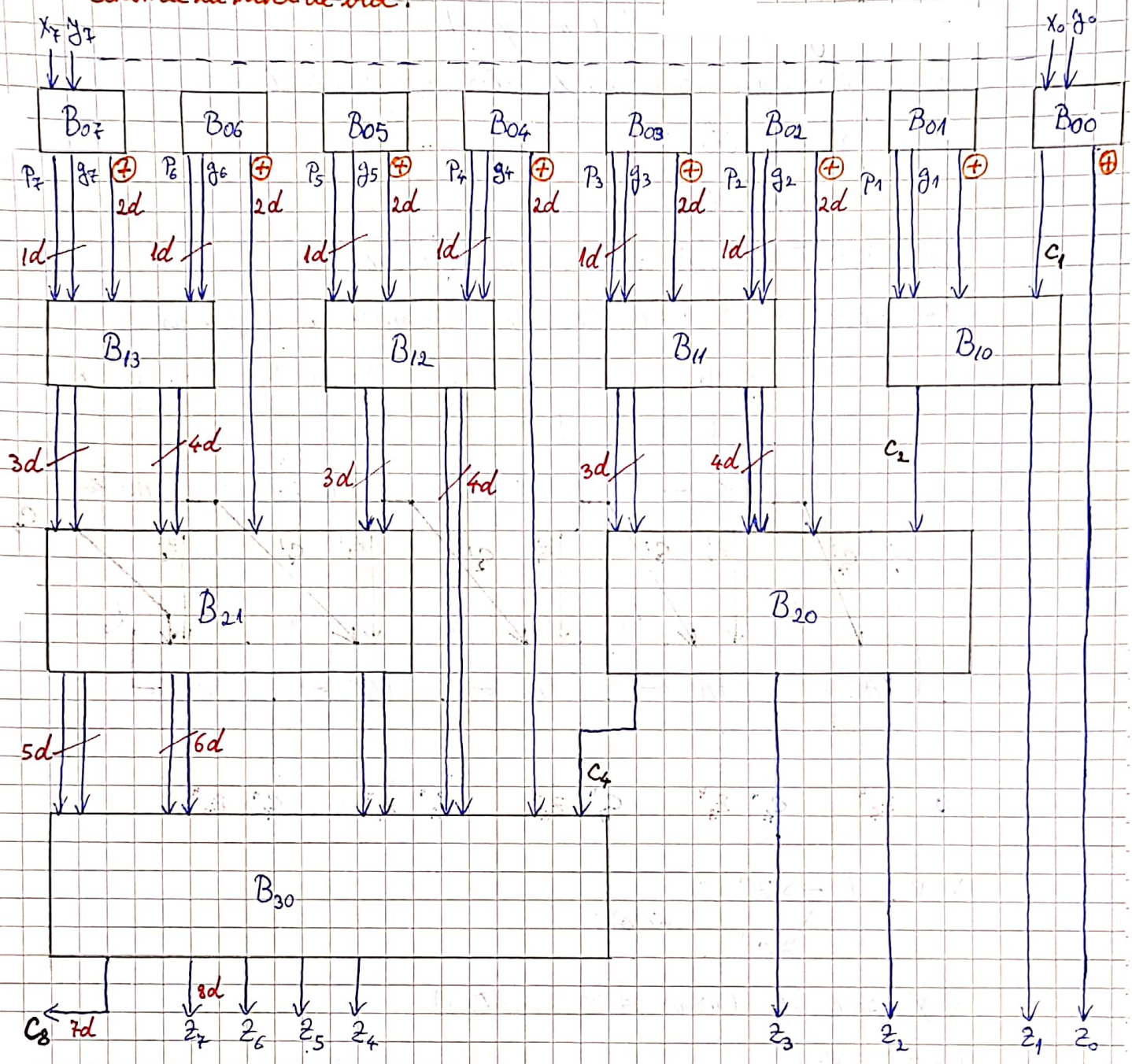
$y = 0\,1\,1\,1\,0\,1\,0\,1$

$c_0 = 0$ (întotdeauna)

posibil ex. de examen

Inputs \ Ranks		7		6		5		4		3		2		1		0	
x		1		0		0		0		1		1		0		1	
y		0		1		1		1		0		1		0		1	
Block level	Carry in	c	s	c	s	c	s	c	s	c	s	c	s	c	s	c	s
i=0	$c_{in}=0$	0	1	0	1	0	1	0	1	0	1	1	0	0	0	1	0
	$c_{in}=1$	1	0	1	0	1	0	1	0	1	0	1	1	0	1		
i=1	$c_{in}=0$	0	1		1	0	1		1	1	0		0	0	1		0
	$c_{in}=1$	1	0		0	1	0		0	1	0		1				
i=2	$c_{in}=0$	0	1		1		1		1	1	0		0		1		0
	$c_{in}=1$	1	0		0		0		0								
in=3	$c_{in}=0$	1	0		0		0		0		0		0		1		0
	$c_{in}=1$																

Schema la nivel de bloc:



$D^{2}_{CSuA} = 8d$ și $D^{c_{out}}_{CSuA} = 7d$

În general: $D^{2}_{CSuA} = 2(\lceil \log_2 n \rceil + 1)d$

$D^{c_{out}}_{CSuA} = 2(\lceil \log_2 n \rceil + 1)d$

$\lceil x \rceil$ - rotunjire în sus

▶ Compararea metodelor de adunare făcute: ex. $n = 128$

$D^{2}_{CSuA} = 16d$ $\quad D^{2}_{CSeA} = 130d$ $\quad D^{2}_{CSKA} = 4 \cdot 17d = 68d$

$D^{c_{out}}_{CSuA} = 15d$ $\quad D^{c_{out}}_{CSeA} = 130d$ $\quad D^{c_{out}}_{CSKA} = 68d$

$D^{2}_{ML-CLA} = 29d$ $\quad (4\lceil \log_2 n \rceil + 1)d$ $\quad D^{2}_{RCA} = 256d$

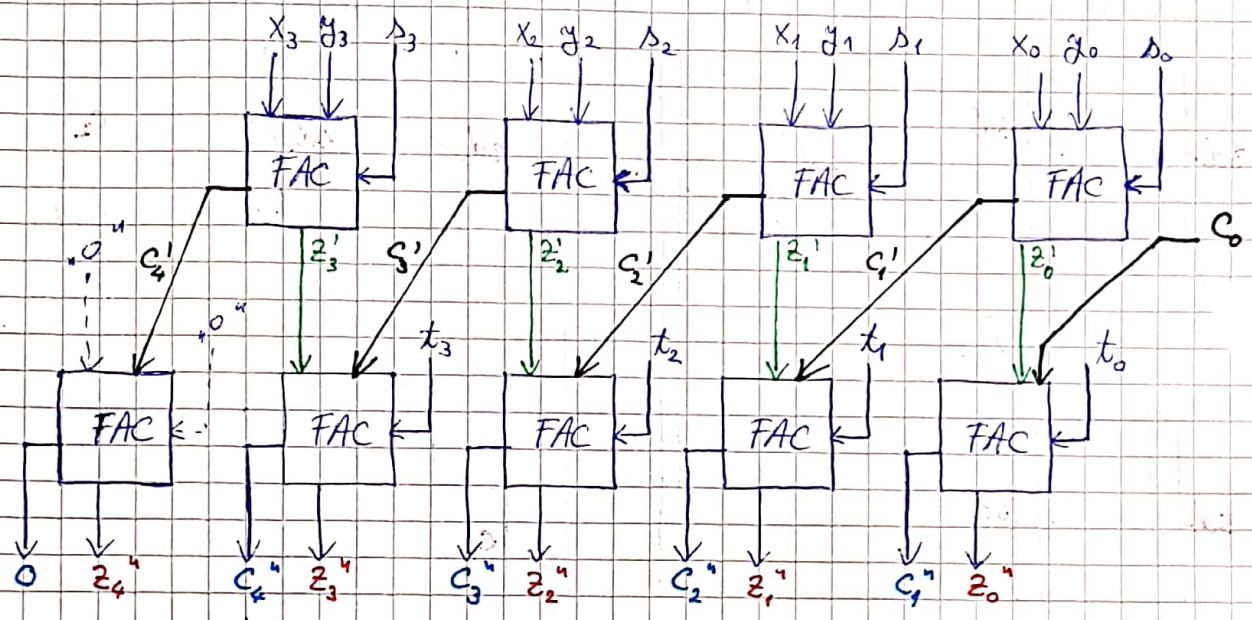
$D^{c_{out}}_{ML-CLA} = 17d$ $\quad (2\lceil \log_2 n \rceil + 3)d$ $\quad D^{c_{out}}_{RCA} = 256d$

1.4.5. Carry Save Adder (cel m. rapid)

- se calculează nu sub formă de vector, ci redundant
  - ↳ carry vector
  - ↳ sum vector

Cd. X, Y, S, T pe 4 biți, pe care vrem să le adunăm

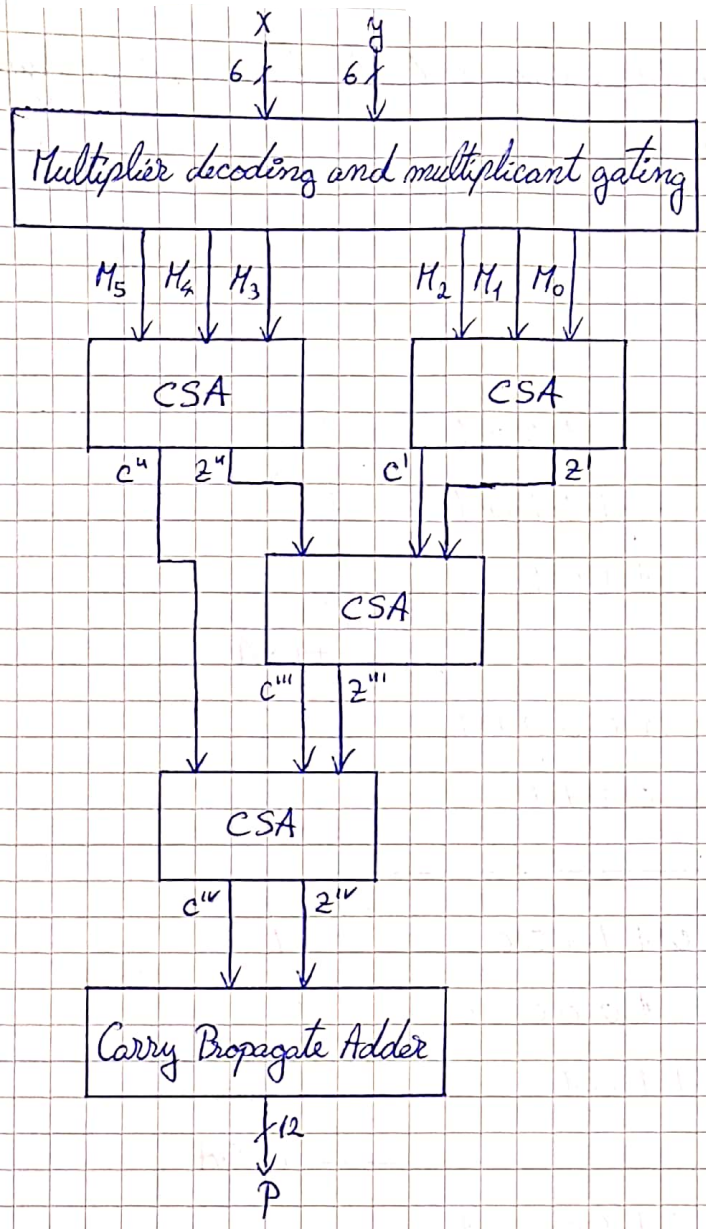
FAC adună maxim 3 biți, deci pe primul nivel adunăm 3/4 nr.



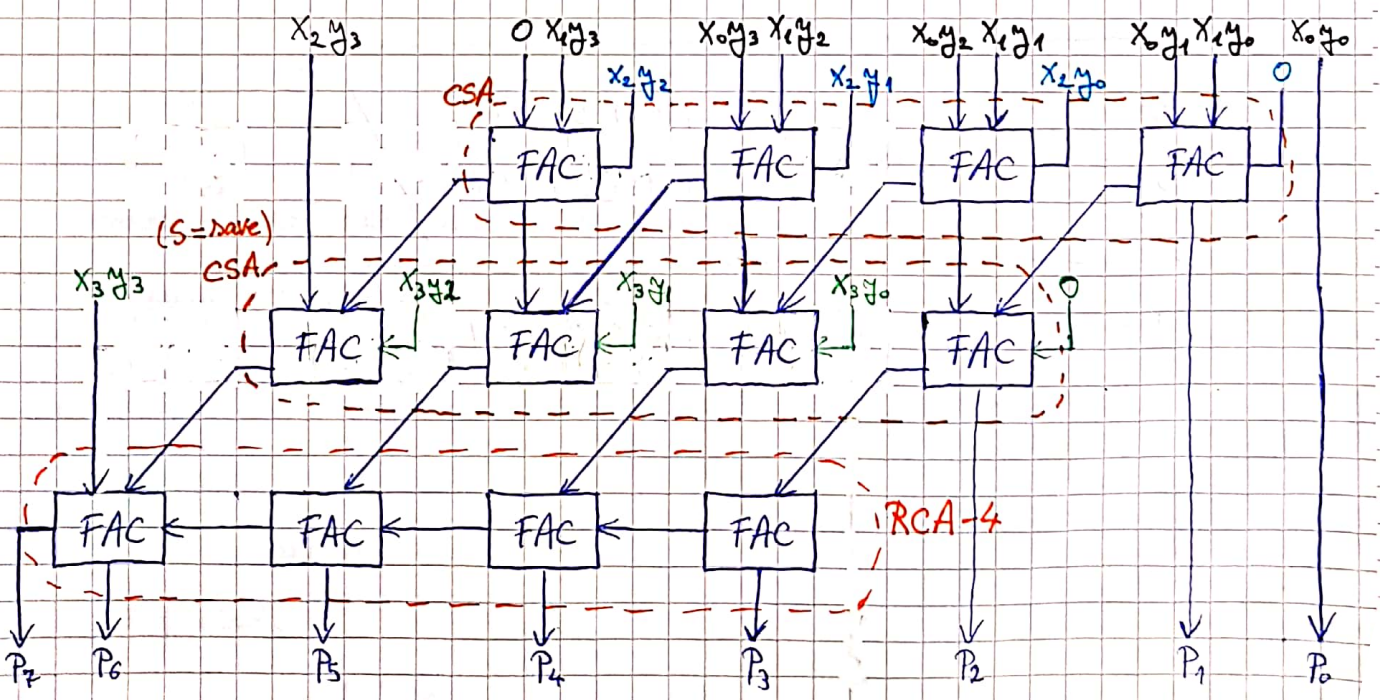
X, Y, S, T → z'', c'' } vectori

X, Y – pe 6 biți (vrem să îi înmulțim)

$M_i = x_i \cdot y \cdot 2^i$



Analizăm cazul înmulțirii a 2 nr. pe 4 biți



Pt. numere ce se introduc în FAC, suma indicilor trebuie să fie egală cu rangul biturilor de produs care ieșe

ex: $X = 15$ (1111)

$y = 13$ (1101)

$M_0 = x_0 y \cdot 2^0$ 1101

$M_1 = x_1 y \cdot 2^1$ 1101

$M_2 = x_2 y \cdot 2^2$ 1101

+CSA

z' 100011

c' 011100

$2c'$ 011100

z' 100010

$M_3 = x_3 y 2^3$ 1101

+CSA

z'' 1110011

c'' 0101000

$2c''$ 0101000

z'' 1110011

+

P 11000011

Curs 6

1.5 Reliable Computing (Fiabilitatea unui sistem de calcul)

→ availability – at. când își poate oferi serviciile pt. care a fost creat

→ reliability – continuitatea oferirii serviciilor de către un sistem

→ mantainability

1.5.1. Sumatoare cu control prin paritate

Cd. X, Y – pe n biți

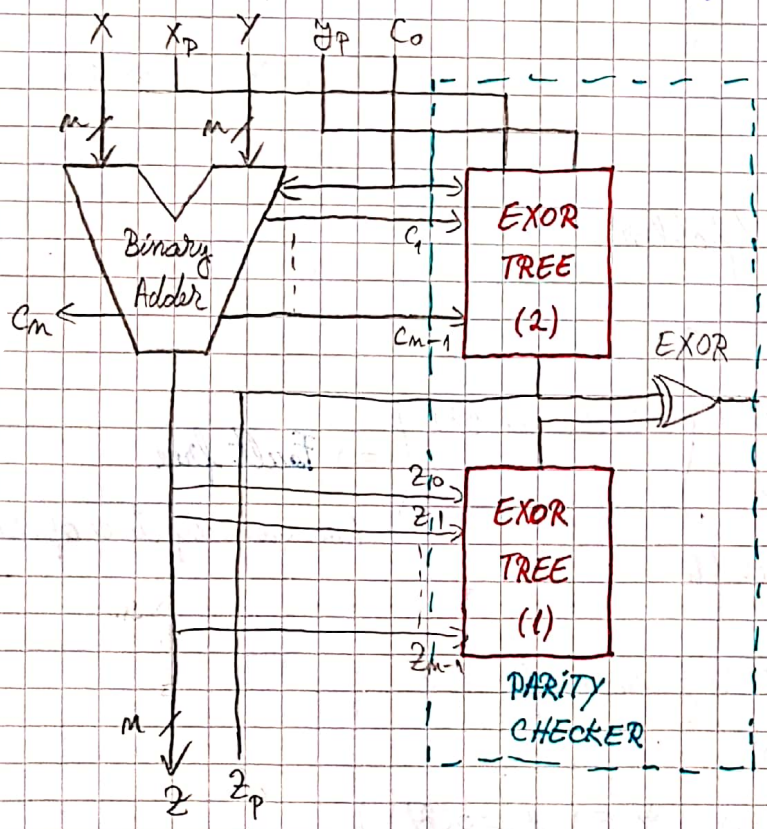
$$Z = X + Y$$

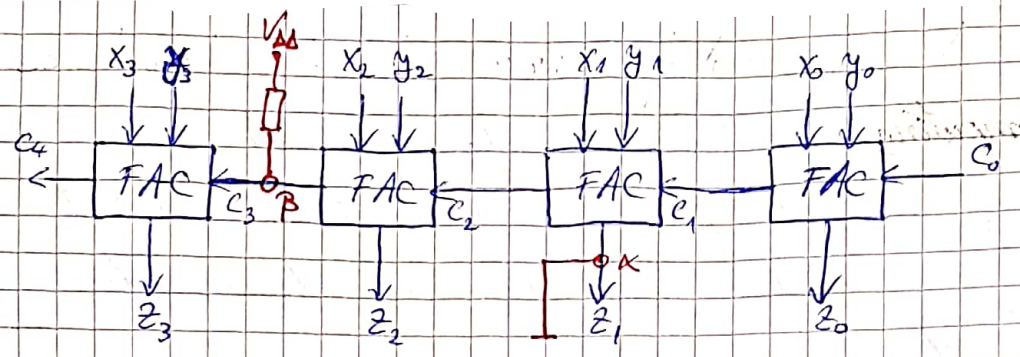
~~X_p bit~~ $X_p = X_{n-1} \oplus X_{n-2} \oplus \ldots \oplus X_0$ – bit de paritate

$$Y_p = Y_{n-1} \oplus Y_{n-2} \oplus \ldots \oplus Y_0$$

$$Z_p = Z_{n-1} \oplus Z_{n-2} \oplus \ldots \oplus Z_0 \quad (1)$$

$$\left.\begin{array}{l} Z_i = X_i \oplus Y_i \oplus C_0 \\ C_p = C_{n-1} \oplus C_{n-2} \oplus \ldots \oplus C_0 \end{array}\right\} \Rightarrow Z_p = X_p \oplus Y_p \oplus C_p \quad (2)$$





- discutăm doar despre defecte de tip logic (bit ce trebuie să fie 0 e 1 și invers)
- cd. doar defecte singulare => DEFECTE SINGLE STUCK-AT

α - scurtcirc. pe linia z_1 cu masa

β - scc. între linia de alimentare și c_3

- α afectează doar z_1 (z_1 e forțat să fie 0)
- β afectează c_3 și z_3, iar c_4 are șansa de 50% să fie afectat (c_3 e forțat să fie 1)

▷ Cazuri de funcționare a defectelor:

1) $X = 0011$ $\quad X_p = 0$

$Y = 0011$ $\quad Y_p = 0$ $\quad \Rightarrow Z_p = 0$ (2)

$C = 0011$ c_0 $\quad C_p = 0$

$Z = 0110$ $\quad Z_p = 0$ (1)

=> Fault free (nu avem defecte ce afectează circ.)

Defectul $\boxed{\alpha}$ Cd. că bitul z_p e blocat în permanență la z_0

$X = 0011$ $\quad X_p = 0$

$Y = 0011$ $\quad Y_p = 0$ $\quad \Rightarrow Z_p = 0$ (2)

$C = 0011$ $\quad C_p = 0$ $\quad$!!!

$Z = 0100$ $\quad Z_p = 1$ (1)

=> Prezența unui erori

Defectul $\boxed{B}$

$X = 0\,0\,1\,1$ $\quad X_p = 0$

$Y = 0\,0\,1\,1$ $\quad Y_p = 0$

$C = 0\,1\,1\,1\,0$ $\quad Z_p = 1$

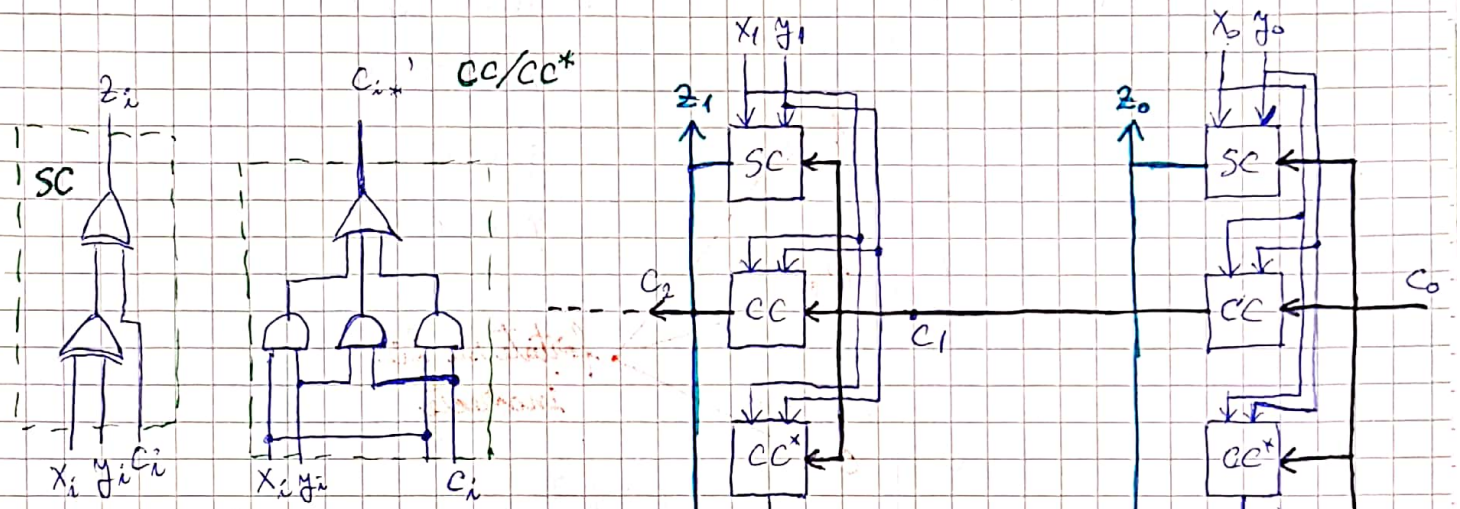
$Z = 1\,1\,1\,0$ $\quad Z_p = 1\ (1)$

$\Rightarrow Z_p = 1\ (2)$

avem egalitate pt. că $1 \oplus 1$ (așa cum se întâmplă în cazul greșit) $= 0 \oplus 0$ (așa cum e corect)

Deci doar bitul de paritate det. eroare $\Rightarrow$ soluții 1.5.2 și 1.5.3.

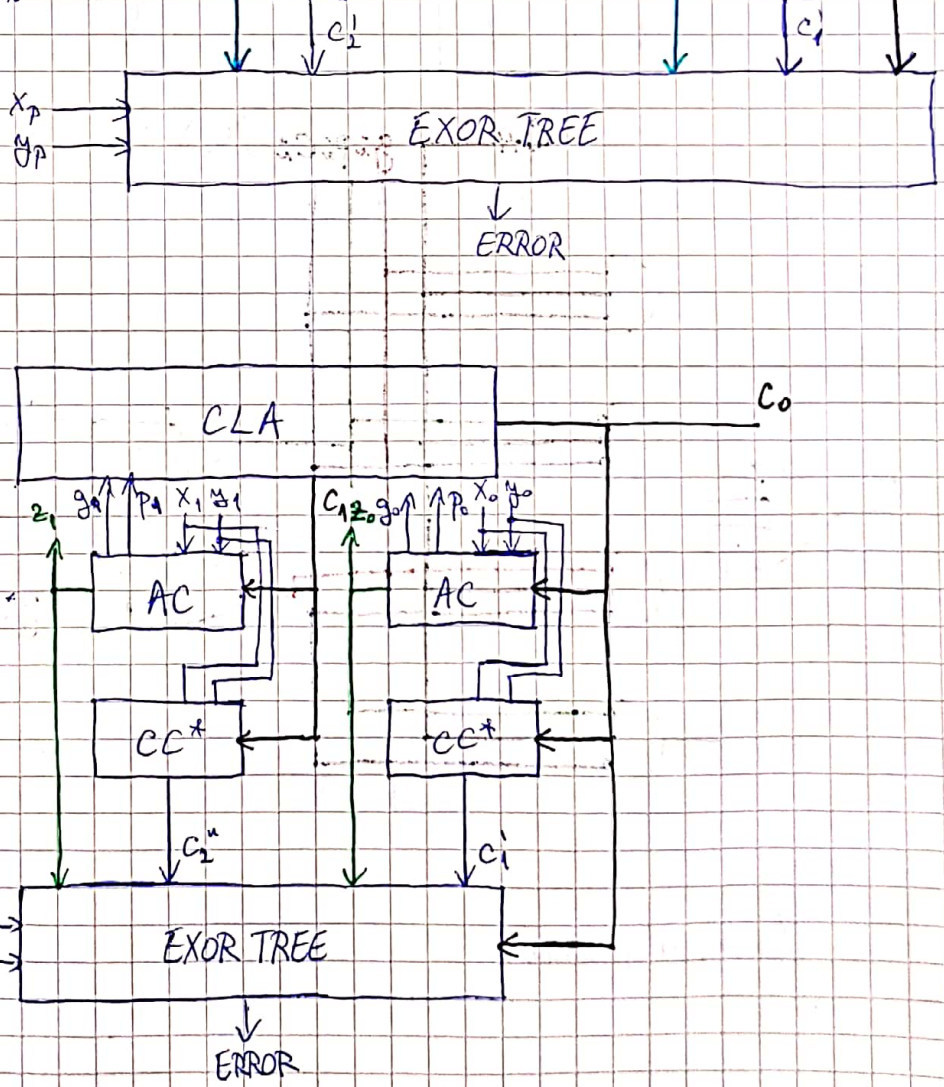
1.5.2. Carry Chain Duplication



CC – celula carry all

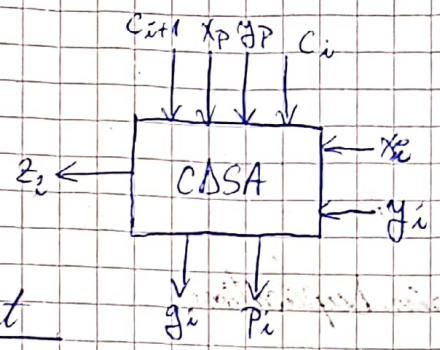
CC* —— // —— duplicated

$c_k' =$ copia lui c_k



1.5.3 Carry Depender Sum Adder ~~(CDSA)~~ (CDSA)

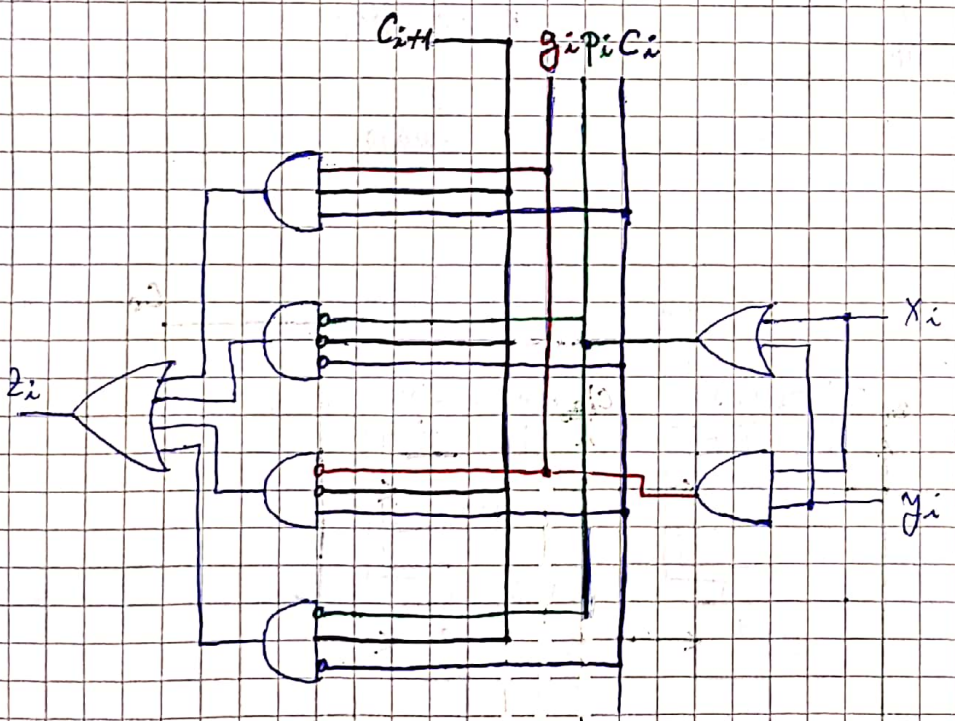
Generăm o celulă care când c_{i+1} e incorect și z_i să fie incorect:



Inputs				Output
x_i	y_i	c_i	c_{i+1}	z_i
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
⋮				
1	1	1	0	0
1	1	1	1	1

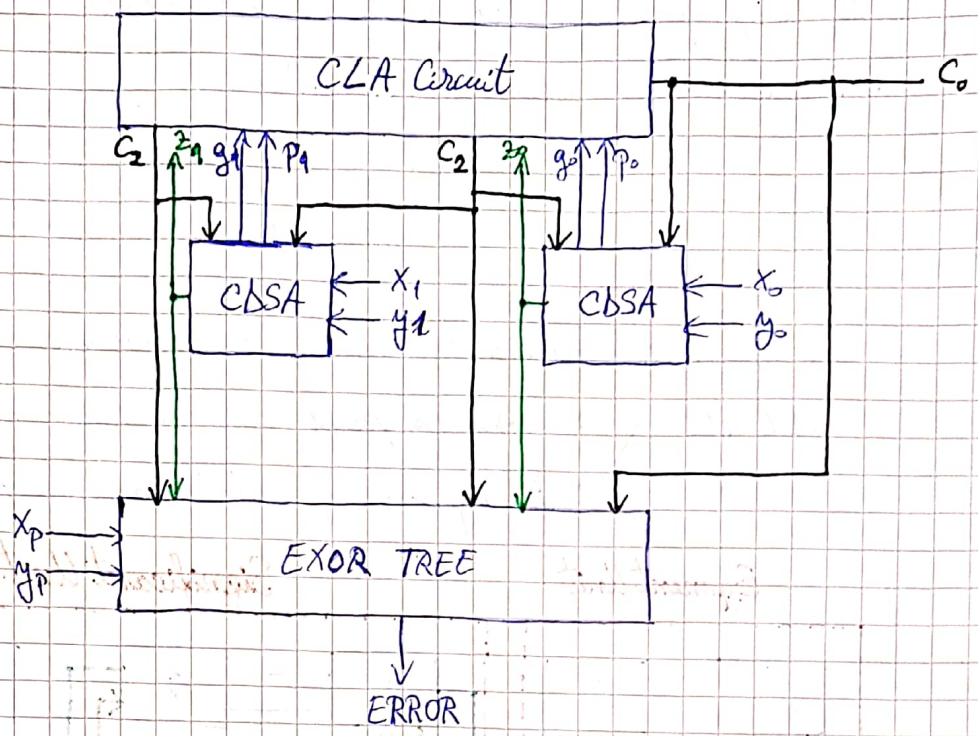
forțate să fie incorecte

$$z_i = g_i c_i c_{i+1} + p_i \overline{c_i}\,\overline{c_{i+1}} + \overline{g_i}\,\overline{c_i}\,\overline{c_{i+1}} + \overline{p_i}\,\overline{c_i}\,c_{i+1}$$



Structura celulei CDSA

La nivel general:



CAP2. Operații aritmetice cu numere de virgulă flotantă

2.1. Aritmetica nr. de virgulă flotantă

IEEE 754 → format împachetatat (hidden bit, exponent în cod exces, bit de semn)

→ despachetat (at când e folosit în calcule aritmetice) (mantisa cu hidden bit exp.)

$X = X_M \cdot 2^{X_E}$

$Y = Y_M \cdot 2^{Y_E}$

→ $X + Y = (X_M + Y_M \cdot 2^{\overbrace{Y_E - X_E}^{\text{aligment}}}) \cdot 2^{Y_E}$, $X_E \geq Y_E$

→ $X - Y = (X_M - Y_M * 2^{Y_E - X_E}) \cdot 2^{X_E}$, $X_E \geq Y_E$

→ $X * Y = X_M * Y_M * 2^{X_E + Y_E}$

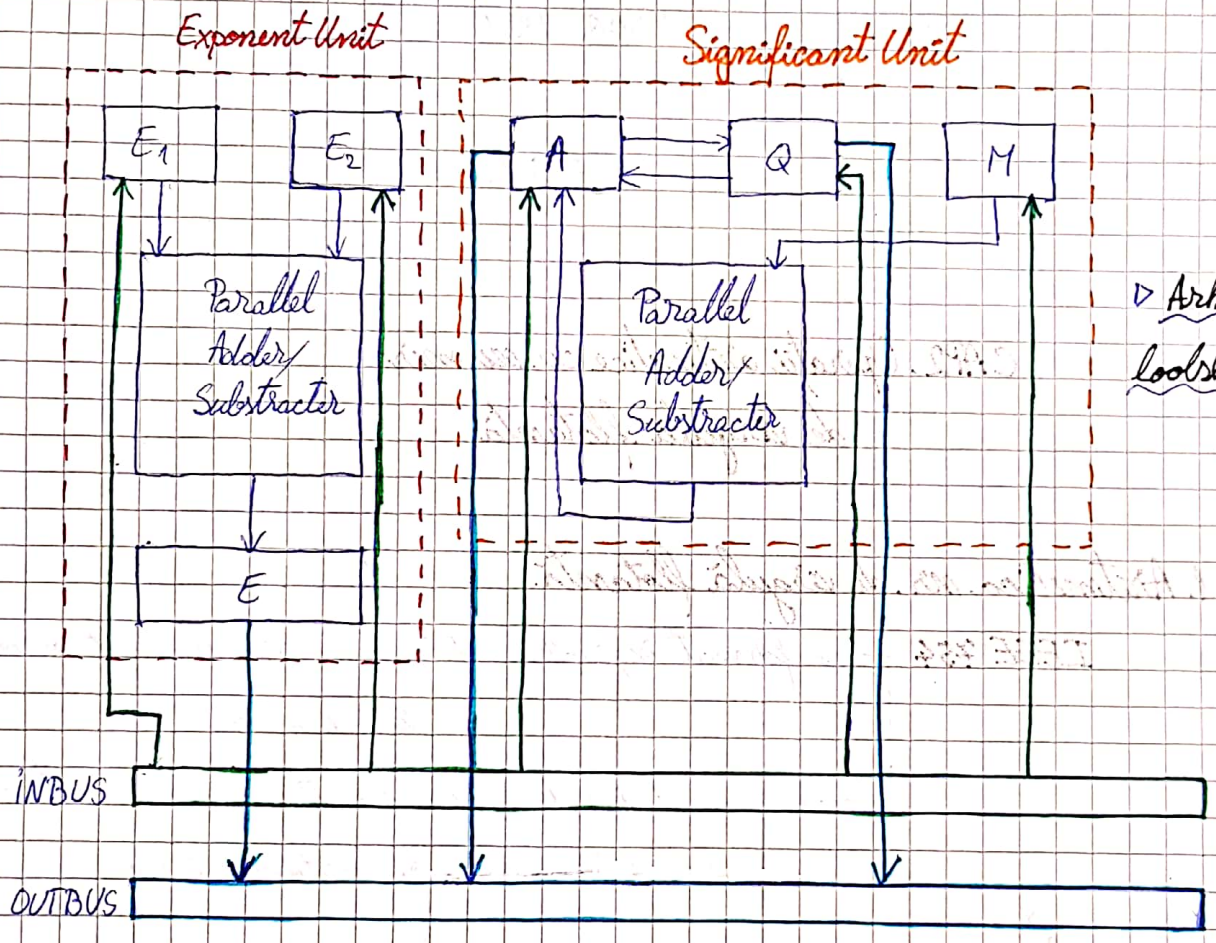
→ $X \div Y = X_M \div Y_M * 2^{X_E - Y_E}$

Aceste op. fundamentale folosesc 2 subunități:

- pt. calculul: → exponent: (întreg) +/−
  → mantisă: (nr. de regulă fix.) +/−/×/÷

aligment

∃ 2 tipuri de atribuiri care efectuează operațiile aritmetice cu nr. de virgulă flotantă: → loosly coupled (unit. de exp. și mantisă prin magistrale partajate)
→ tighthy coupled (unitățile comunică prin legături directe)



▷ Arhitectura loosly-coupled

INBUS – magistrala de intrare

OUTBUS – magistrala de ieșire

M = păstrează deînmulțitul

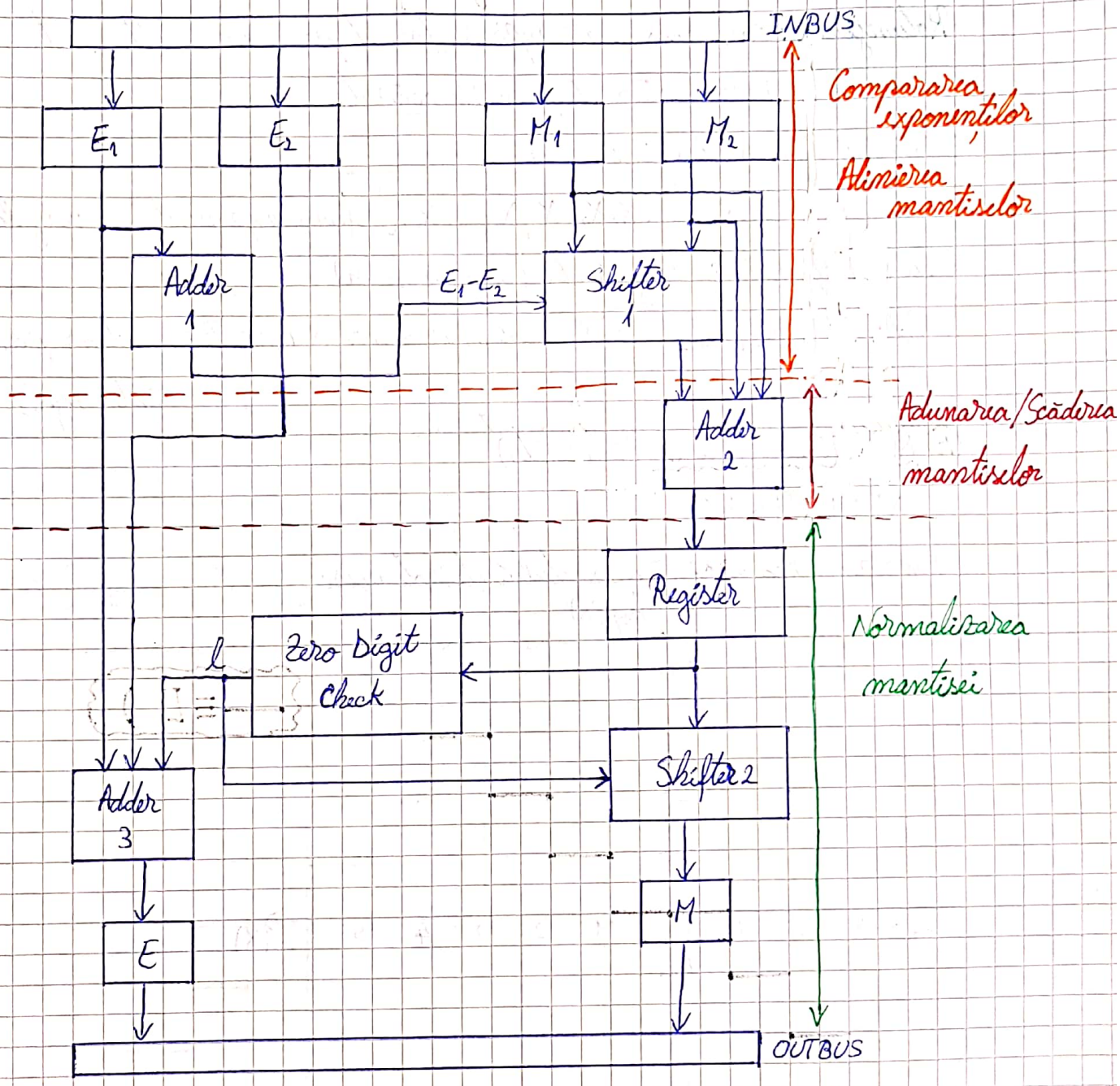
Q = păstrează câtul la împărțire / jumătatea nesemnif. a înmulțirii

A = restul împărțirii / jumătatea mai semnif. a înmulțirii

Curs 7

▷ Arhitectura tightly coupled:

IBM FORMAT (operează nr. pe 32 sau 64 biți): baza 16



Dacă $E_1-E_2=k$ cu $k>0$ se va shifta la dreapta M_2 >> $4k$ biți

Dacă $E_1-E_2=k$ cu $k<0$ se va shifta la M_1

- Adder 3: pe 64 biți: 56+7+1
- Shifter 1: accelerează alinierea mantisei
- Zero Digit Check: accelerează normalizarea; det. nr. de grupuri de câte 4 biți de „0" în pozițiile cele mai semnificative ale mantisei. Acest nr. va fi scăzut din exponentul rezultatului.

• Shifter 2: accelerează normalizarea, shiftează la dreapta mantisa cu 4* nr. de grupuri de 0

2.2. Rotunjirea numerelor în virgulă flotantă

Rotunjirea: conversia unei reprezentări de precizie înaltă într-o reprezentare de precizie scăzută

Folosim standardul IEEE 754

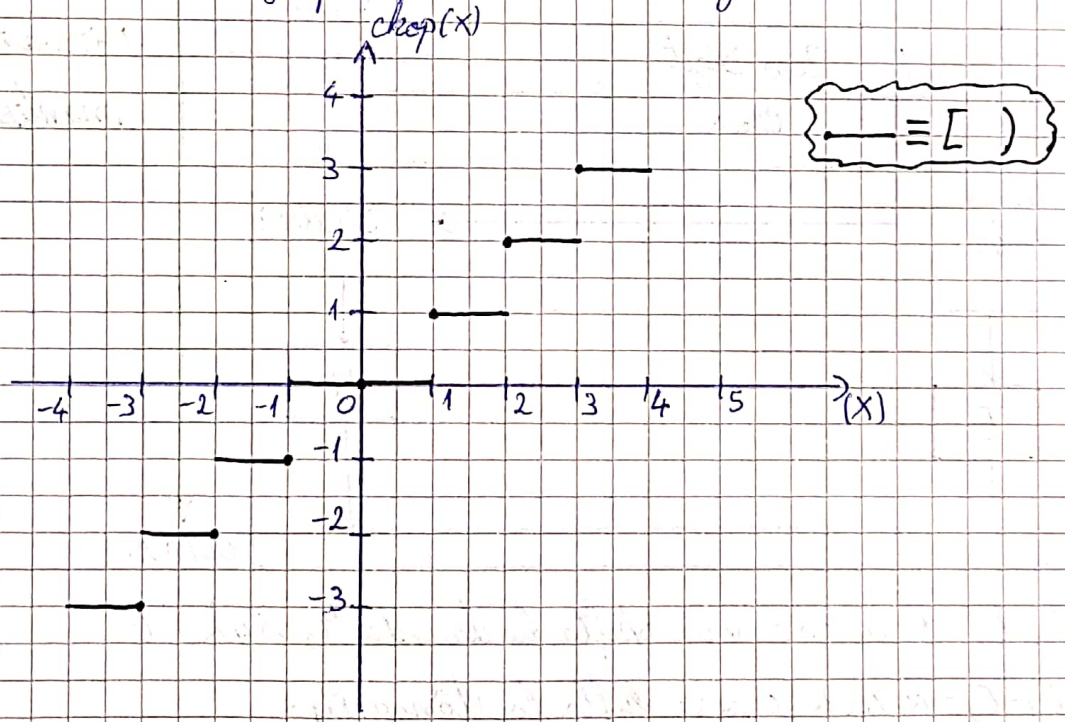
Cd.: $X = x_{n-1}x_{n-2}\dots x_1x_0.x_{-1}x_{-2}\dots x_{-m}$ fără exponent, cu virgulă fixă cu oricâți biți în stg. și dr. (aici n și m)

Cd: $X^* = x^*_{n-1}x^*_{n-2}\dots x^*_1x^*_0$ rotunjirea nr. X (păstrăm doar partea întreagă)

Ⓐ Rotunjirea către 0 (inwards)

$X^*: |X^*| \le |X|$

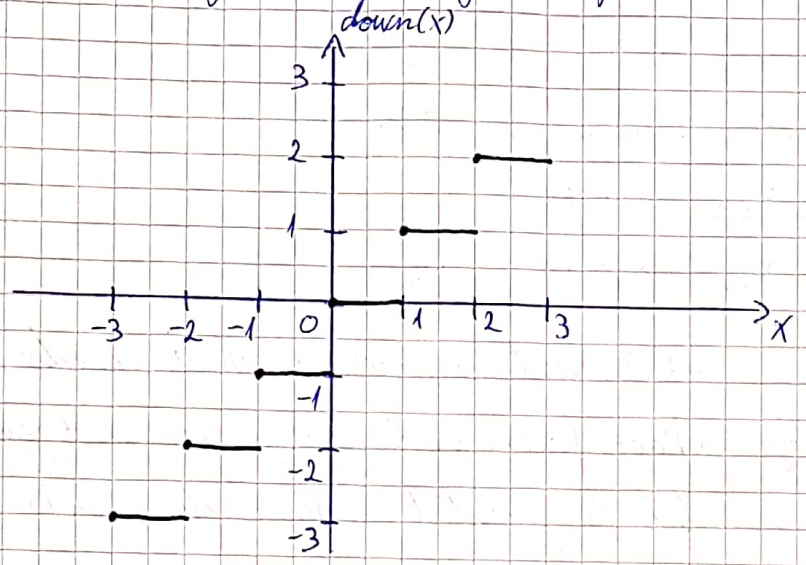
Reprezentarea grafică a modului de rotunjire



Dacă X e reprezentarea în semn-mărime, rotunjirea către 0 reprez. trunchierea părții fracționare.

Ⓑ Rotunjirea către $-\infty$ (downwards)

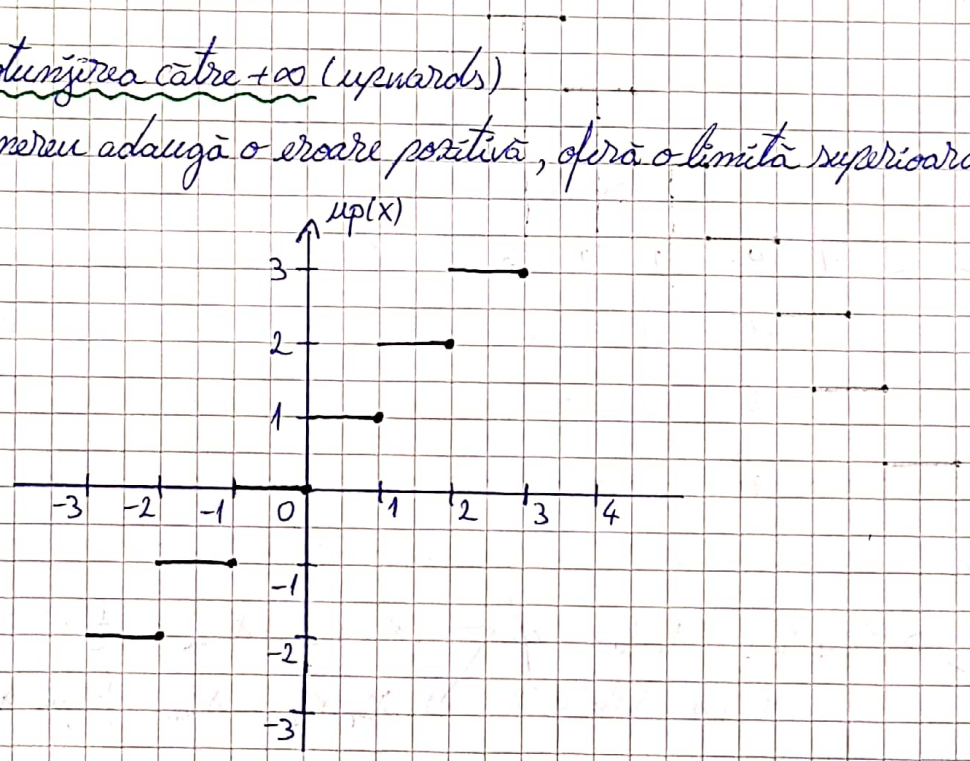
– mereu adaugă o eroare negativă, oferă o limită inferioară a rezultatului



Dacă x e reprezentarea în complement de 2, rotunjirea înseamnă trunchierea părții fracționare.

Ⓒ Rotunjirea către $+\infty$ (upwards)

– mereu adaugă o eroare pozitivă, oferă o limită superioară a rezultatului



Ⓓ Rotunjirea la cel mai apropiat întreg (rot. intermediară)

Cd. că $x \geq 0$: $x^* \to x_{n-1}x_{n-2}\ldots x_1x_0$ if. $x_{-1}x_{-2}\ldots x_{-n} < \frac{1}{2}$

$\to x_{n-1}x_{n-2}\ldots x_1x_0 + 1$ if. $x_{-1}x_{-2}\ldots x_{-n} \geq \frac{1}{2}$

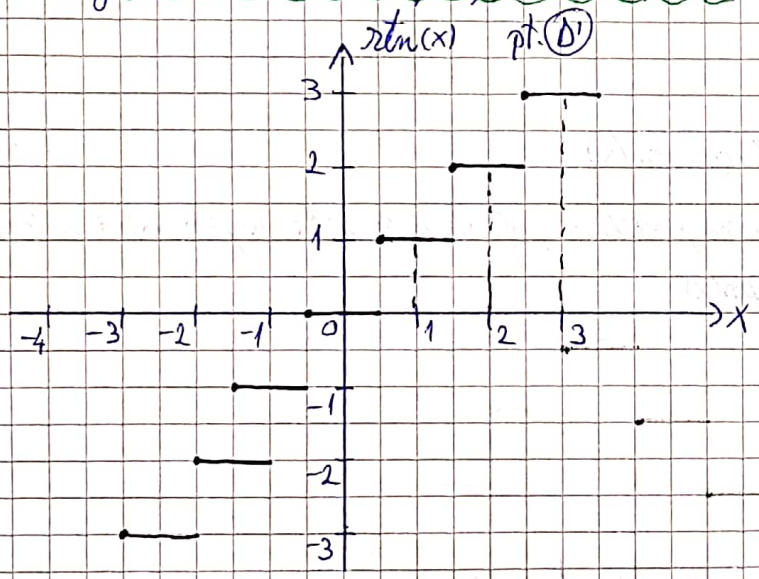
La Ⓑ și Ⓒ erorile se acumulează în aceeași direcție (înspre $-\infty$ și resp. $+\infty$)

La Ⓓ acumularea erorilor: luăm de ex. $x_{-3} = x_{-4} = \ldots = x_{-n} = 0$.

	inputs		outputs	
	X_{-1}	X_{-2}	X^*	$\varepsilon = X^* - X$
$.00 = 0,0$	0	0	$X_{n-1}X_{n-2}\cdots X_1X_0$	0
$.01 = 0,25$	0	1	$X_{n-1}X_{n-2}\cdots X_1X_0$	$-0,25$
$.10 = 0,5$	1	0	$X_{n-1}X_{n-2}\cdots X_1X_0+1$	$+0,5$
$.11 = 0,75$	1	1	$X_{n-1}X_{n-2}\cdots X_1X_0+1$	$+0,25$

Dacă cd. că toate cele 4 cazuri sunt echiprobabile, eroarea medie este:
$\frac{0-0,25+0,5+0,25}{4} = \frac{1}{8}$ => Dacă facem 8 adunări consecutive ∃ șansa să apară o eroare de 1 unitate la rezultat => Apare rotunjirea D

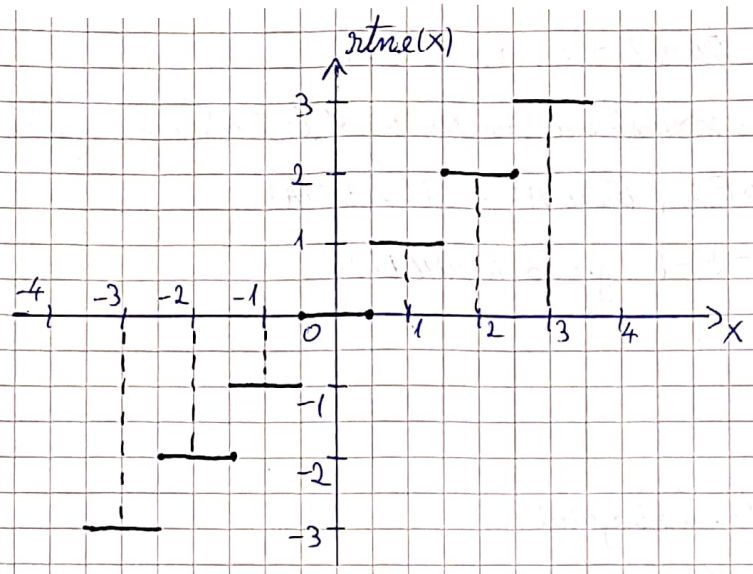
(D) Rotunjirea la cel mai apropiat nr. par



Cd. $X \geq 0$:

$X^* \to X_{n-1}X_{n-2}\cdots X_1X_0$ if $\to X_{-1}X_{-2}\cdots X_{-m} < \frac{1}{2}$
$\to X_{-1}X_{-2}\cdots X_{-m} = \frac{1}{2}$ și $X_0 = 0$

$X^* \to X_{n-1}X_{n-2}\cdots X_1X_0+1$ if $\to X_{-1}X_{-2}\cdots X_{-m} > \frac{1}{2}$
$\to$ —— // —— $= \frac{1}{2}$ și $X_0 = 1$



2.3. Adunarea și scăderea nr. de virgulă flotantă cu precizie infinită (nu pierdem niciun bit din rezultat)

$X_E > Y_E$: la adunarea X_M și Y_M se shiftează Y_M cu k biți

Pseudo limbaj:

Adder / Substractor 1:

- declare register : $E_1[(e-1):0]$, $E_2[(e-1):0]$, $E[(e-1):0]$,
  (declară mediul de stocare)
  $A[(m+1):0]$, $M[(m+1):0]$; A_cout ; Error ;
  ↳ mantisa primului operant
  (declară magistrale)
- declare bus : $INBUS[(m+e+1):0]$, $OUTBUS[(m+e+1):0]$;

BEGIN : Error := 0 , A_cout := 0,

INPUT : $E_1 := INBUS(X_E)$; $A := INBUS(X_M)$;

$E_2 := INBUS(Y_E)$, $M := INBUS(Y_M)$;

COMPARE : $E := E_1 - E_2$;

ALIGN : <u>if</u> $E<0$ <u>then</u> A := rightshift(A), E := E+1,

<u>go to</u> ALIGN;

<u>if</u> $E>0$ <u>then</u> M := rightshift(M), E := E-1,

<u>go to</u> ALIGN;

ADD/SUBTR : $A := A \pm M$, $E := \underline{max}(E_1, E_2)$;

OVERFLOW : if A_cout = 1 <u>then</u> <u>begin</u>

if $E = E_{max}$ <u>then</u> <u>go to</u> Error,

A := rightshift(A), E := E+1 , <u>go to</u> <u>END</u>;

end

ZERO: if A=0 then E=0, go to END

NORMALIZE: if is_normalized(A) then go to END,

UNDERFLOW: if E > Emin then A := leftshift(A),

E := E-1, go to NORMALIZE;

ERROR: Error := 1;

END:

→ Elemente ale pseudo-limbajului

1) A_cout.A (rezultat final)
   → concatenare

2) IOBUS = INBUS + OUTBUS

3) := → atribuire

rightshift → shift cu 1 la dr. } fct. cablate

max

4) Sequential operation: → non-conflicting (,) → se execută în acelaşi ciclu de clock
   → conflicting (;) → nu se execută simultan

5) Control flow → unconditional jumps
   → conditional jumps

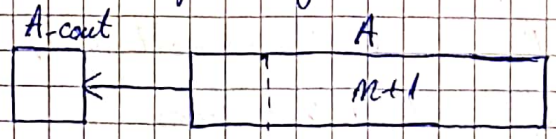
6) A := A+M, M := 0;
   → ex. în care se ştearsă M şi A în acelaşi clock

• operanzii: m+e+2
   → 1 bit semn
   → e biţi exponent
   → m+1 biţi mantisă

A, M - mantisele operanzilor

A_cout - rangul mai semnif. al registr. A.

• regiştrii: A, M;



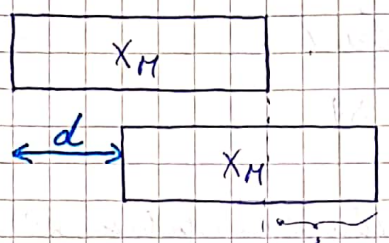
E_1, E_2 - au facilităţi de stocare a rez.; E - facilit. de increm./decrem.

ERROR

Curs 8

2.4. Rotunjirea operației de adunare/scădere a nr. de virgulă flotantă.

$X_M + Y_M$: Y_M alignment $|Y_E - X_E| \overset{not}{=} d$



↳ ce se întâmplă cu acești biți?

Y_M aligned → precizie infinită : păstrează toți biți $w < 2^{-m+1}$
→ precizie finită : păstrează 3 biți : sticky bits
→ weights (w)

$X_M = 1.X_{m-2}X_{m-3}\ldots X_i \ldots X_1X_0$

→ g = bitul de gardă, folosit pt. păstrarea preciziei, are ponderea 2^{-m}, e ultimul bit shiftat în afara capacității de stocare a lui y

→ r = rol în rotunjirea rezultatului +/−, are ponderea 2^{-m-1} (bitul de rotunjire)

→ s = e un sau logic între toți ceilalți biți ce ies din capacitatea de stocare, are ponderea 2^{-m-2} (sticky bit)

$Z_M = X_M + Y_{M\,aligned}$

Adică $Z_M = Z_m Z_{m-1}.Z_{m-2}Z_{m-3}\ldots Z_1Z_0 \,|\, g\, r\, s$

↳ reprez. un carry out din poz. cea mai semnif. a semnificantului

Ne interesează normalizarea lui Z

$Z_M = Z_m Z_{m-1}.Z_{m-2}Z_{m-3}\ldots Z_1Z_0 \,|\, g\, r\, s$

Forma normalizată:

$Z_{M_n} = 1.Z_{m-2_n}Z_{m-3_n}\ldots Z_{1_n}Z_{0_n} \,|\, R\, S$

↳ normalizat.

▷ Cazurile de normalizare:

a) $z_m = 1 \Rightarrow$ trebuie shiftat z_M cu un bit la dreapta

$z_{M_n} = 1.z_{m-1}z_{m-2}\ldots z_2 z_1 \,\vdots\, z_0$ (g or r or s)

b) $z_m = 0, z_{m-1} = 1 \Rightarrow z_m$ e deja normalizat

$z_{M_n} = 1.z_{m-2}z_{m-3}\ldots z_1 z_0 \,\vdots\, g$ (r or s)

c) $z_m = 0, z_{m-1} = 0, z_{m-2} = 1 \Rightarrow z_m$ se shiftează cu 1 bit la stânga

$z_{M_n} = 1.z_{m-3}z_{m-4}\ldots z_0 \,\vdots\, r\ s$

d) $z_m = 0, z_{m-1} = 0, z_{m-2} = 0, z_{m-3} = 1 \Rightarrow z_m$ se shiftează cu 2 biți la stg.

$z_{M_n} = 1.z_{m-4}z_{m-5}\ldots g\,0 \,\vdots\, 0\ 0$

At. când pt. normalizare e necesară o shiftare cu 2/mai mulți biți la stg. a lui z_M, z_M e completat cu g urmat de „0"-uri, iar val. lui R și S sunt și ele egale cu „0"

Pe baza rezultatului normalizat, următorul pas este rotunjirea semnificantului.

$z_{M_n} = 1._{m-1_n}z_{m-2_n}\ldots z_{1_n}z_{0_n} \,\vdots\, R\ S$

Aflăm z_M^* = forma rotunjită a lui z_{M_n}

Modul de rotunjire	$z_{M_n} \geq 0$	$z_{M_n} < 0$
Către 0	$z_M^* = z_{M_n}$ Se elimină R, S	$z_M^* = z_{M_n}$ Se elimină R, S
Către $-\infty$	$z_M^* = z_{M_n}$	if (R or S) $z_M^* = z_{M_n} - 1$ else $z_M^* = z_{M_n}$
Către $+\infty$	if (R or S) then $z_M^* = z_{M_n} + 1$	$z_M^* = z_{M_n}$
Către cel mai apropiat par	if R and (S or z_{0_n}) then $z_M^* = z_{M_n} + 1$	if (R and (S or z_{0_n})) then $z_M^* = z_{M_n} - 1$

Dacă se adună 1 la 11...1 $\Rightarrow$ apare bit de carry out $\Rightarrow$ trebuie o post-normalizare (>> cu 1 bit și crește exponentul cu 1)

2.5. Algoritmul de adunare/scădere a nr. de virgulă flotantă cu rotunjire

▷ Rotunjirea cu precizie finită

– eroarea de rotunjire nu e corelată cu diferența exponenților

Diferențe	Eroare
+6	11/512
+5	-10/512
+1 (pt. adunare)	f. mare
+1 (pt. scădere)	0

Cd. un format simplificat al IEEE 754:

→ 1 bit semn

→ 3 biți exponent ; $bias = 2^{3-1} - 1 = 3$

→ 4 biți significant (cu 1 bit ascuns)

ex: $X = 0,5625 = 0.1001 \cdot 2^0 = \underbrace{1,001 \cdot 2^{-1}}_{\text{forma normalizată}}$

$Y = -3,75 = -11,11 = -1,111 \cdot 2^1$

X:

0	0	1	0	0	0	1

$exp = -1 + bias$

1	1	0	0	1	1	1

Step 1: Unpack

- adăugarea bitului 1 supraunitar
- se verifică cazurile speciale

X

0	0	1	0	1.	0	0	1

Y

1	1	0	0	1.	1	1	1

Step 2: • Se calculează diferența exponenților

$d = X_E - Y_E = -2 < 0$ -> ar trebui X shiftat și aliniat, dar doar Y se poate shifta => swap(X, Y)

• Se alege exponentul rezultatului ca fiind $\max(X_E, Y_E) \Rightarrow Z_E = 4 (100_2)$

X	1	1	0	0	1	1	1	1

Y	0	0	1	0	1	0	0	1

Step 3: if sign(X) ≠ sign(Y) => facem scădere

- ca să nu avem un alt dispozitiv de scădere, facem complementul de 2 unuia dintre cei 2
- cd. că doar Y se poate complementa și la final ~~trebuie~~ pt. alegerea semnului trebuie să ținem cont de toate operațiile făcute

$Y_{M_{C_2}} = 0.1\underbrace{1\ 1}$

Step 4: Dacă Y_M a fost C_2 în pasul 3, se introduc biți de 1 în poz. cele mai semnificative

- se face alinierea lui Y_M (se face shift>>2)

$Y_{M_{C_2\,aligned}} = 1.101 \,|\, \overset{g}{1}\ \overset{r}{1}$

- se păstrează biții sticky
- dacă nu complementăm în pasul 3, aveam:

$Y_{M_{C_2\,aligned}} = 0.001 \,|\, 1\ 1$

Step 5: $X_M = 1.111$

$Y_{M_{C_2\,al}} = 1.101 \,|\, \overset{g}{1}\ \overset{r}{1}\ \overset{s}{0}$

$Z_M = 11.100 \,|\, 1\ 1\ 0$

- Dacă sign X = sign Y și ∃ carry out, păstrăm carry out și shiftăm cu 1 la dreapta

- Dacă $sign X \neq sign(Y)$
  - → dacă $\nexists$ carry out ⇒ Complement C_2 pt. că Z_M e negativ
  - → dacă $\exists$ carry out ⇒ Reg. pozitiv și ignorăm C_{out}

  Aici $Z_M = 1.100|110$

Step 6: Facem prenormalizarea (ca în 2.4)

Pt. normalizare verificăm cazurile excepție:

- if $Z_E = Z_{E_{maxim}}$ (6 pt. 3 biți, 7 e rezervat pt $\pm\infty$)

  și Z_M are nevoie de 1-bit Right-Shift ⇒ overflow
- if $Z_E = Z_{E_{min}}$ (1 pt. că 0 e păstrat pt. val. 0 și pt. nr. denormalizate)

  și Z_M are nevoie de 1 sau m. mulți biți Left Shift ⇒ underflow

$Z_{M_n} = 1.100$

Step 7: R, S

Step 8: Rotunjirea (ca în 2.4)

În cazul în care adunarea cu 1 pe LSB generează un C_{out}, se va impune o postnormalizare, adică 1-bit Right-Shift și trebuie verificat cazul de overflow

- Cd. ca mod de rotunjire „round to nearest even"

  Cond. de adunare a unui bit de 1: R <u>and</u> (S <u>or</u> Z_{0_n})

  Aici 1 and (1 or 0) = 1 ⇒ adunăm 1 la LSB în Z_0:

$$\begin{array}{r} Z_{M_n} = 1.100+ \\ 0.001 \\ \hline Z_M^* = 1.101 \end{array}$$

Step 9: Alegem semnul rezultatului

- Dacă $sign(X) = sign(Y)$ ⇒ $sign(Z) = sign(X)$
- Dacă $sign(X) \neq sign(Y)$

Swap (Step2)	Complement (Step5)	sign(X)	sign(Y)	sign(Z)
YES	–	+ (*)	– (*)	–
YES	–	–	+	+
NO	YES	+	–	–
NO	YES	–	+	+
NO	NO	+	–	+
NO	NO	–	+	–

(*) – semnale inițiale, de dinainte de swap

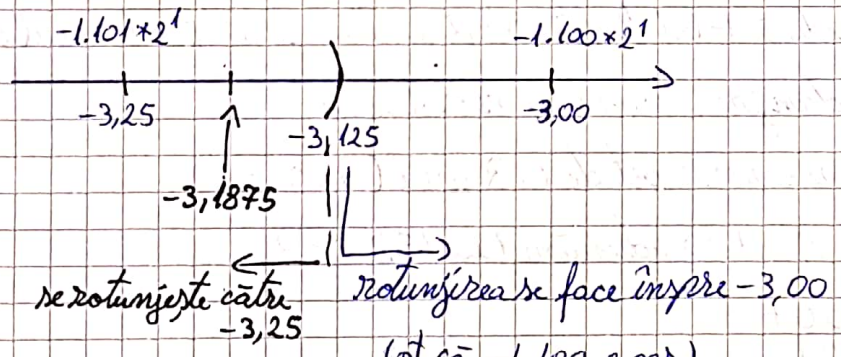
Step 10: Se calculează val. rezultatului, în format împachetat

Z:	1	1	0	0	1	0	1

Verificare: $X = 0,5625$ $\quad X + Y = -3,1875$ (rez. exact)

$Y = -3,75$

$Z = (-1)^1 * 2^{4-3} * (1.101) = -2 * 1.101 = -11.01 = -3.25$ (rez. aprox.)



▷ Construirea unui dispozitiv combinațional care să efectueze prenormalizarea (pașii 6, 7)

Step 5: $Z_M = Z_4 Z_3 . Z_2 Z_1 Z_0 \mid g\ r\ s$

$Z_M = 1 . Z_{2m} Z_{1m} Z_{0m} \mid R\ S$

a) Z_M normalizat: $Z_M = 0$, $Z_3 = 1$

$Z_{Mm} = 1 . Z_2 Z_1 Z_0 \mid g\ (r \underline{or} s)$

b) Z_M: 1-bit Left Shift: $Z_4=0$, $Z_3=0$, $Z_2=1$

$Z_{M_m} = 1.Z_1Z_0\ g\ r\ s$

c) Z_M: 2-bit Left Shift: $Z_4=0=Z_3=Z_2$; $Z_1=1$

$Z_{M_m} = 1.Z_0\ g\ 0\ 0\ 0$

d) Z_M: 3-bit Left Shift

$Z_{M_m} = 1.g\ 0\ 0\ 0\ 0$

e) Z_M: 1-bit Right Shift

$Z_{M_m} = 1.Z_3Z_2Z_1\ Z_0\ (g \underline{or}\ r \underline{or}\ s)$

Asociem variabile booleene fiecărui caz:

a) $\rightarrow l/r_0$; b) $\rightarrow l_1$; c) $\rightarrow l_2$;

d) $\rightarrow l_3$; e) $\rightarrow r_1$

$Z_{2m} = Z_2 \cdot l/r_0 + Z_1 \cdot l_1 + Z_0 l_2 + g\, l_3 + Z_3 r_1$

$Z_{1m} = Z_1 \cdot l/r_0 + Z_0 \cdot l_1 + g \cdot l_2 + Z_2 \cdot r_1$

$Z_{0m} = Z_0 \cdot l/r_0 + g \cdot l_1 + Z_1 \cdot r_1$

$R = g \cdot l/r_0 + r \cdot l_1 + Z_0 \cdot r_1$

$S = (r \underline{or}\ s) \cdot l/r_0 + s \cdot l_1 + (g \underline{or}\ r \underline{or}\ s) \cdot r_1$

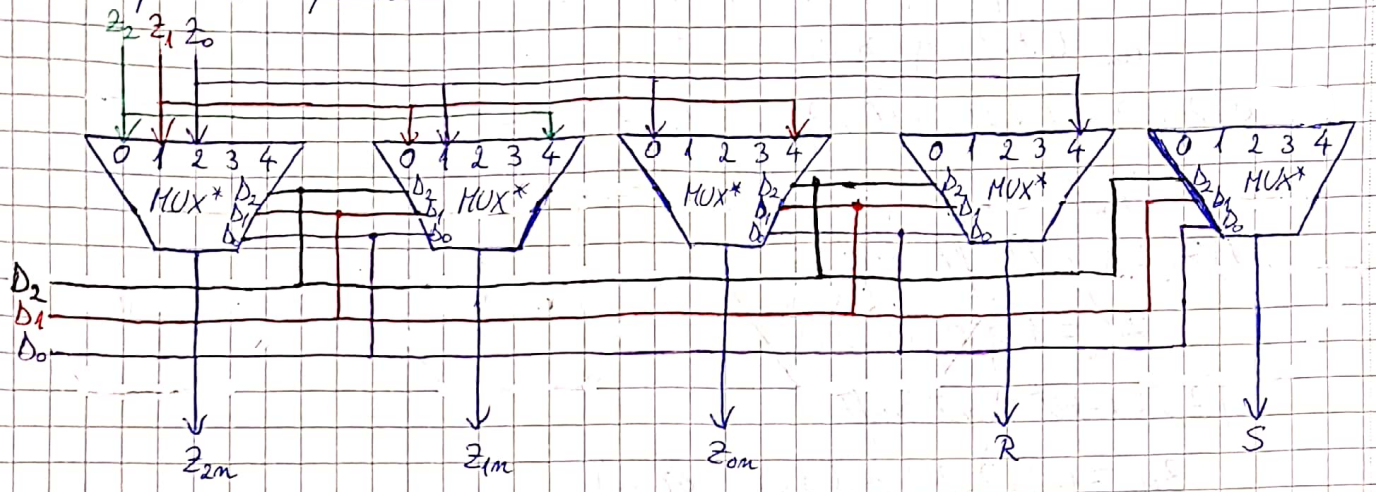
Curs 9 2.5.

Inputs					Outputs		
r_1	l_3	l_2	l_1	l/r_0	0	D_1	D_0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	1	0	0	0	0	1	1
1	0	0	0	0	1	0	0

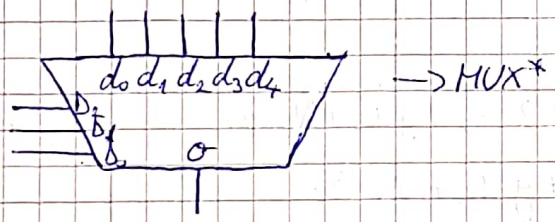
- fiecare intrare poate fi activă doar ea la un moment dat

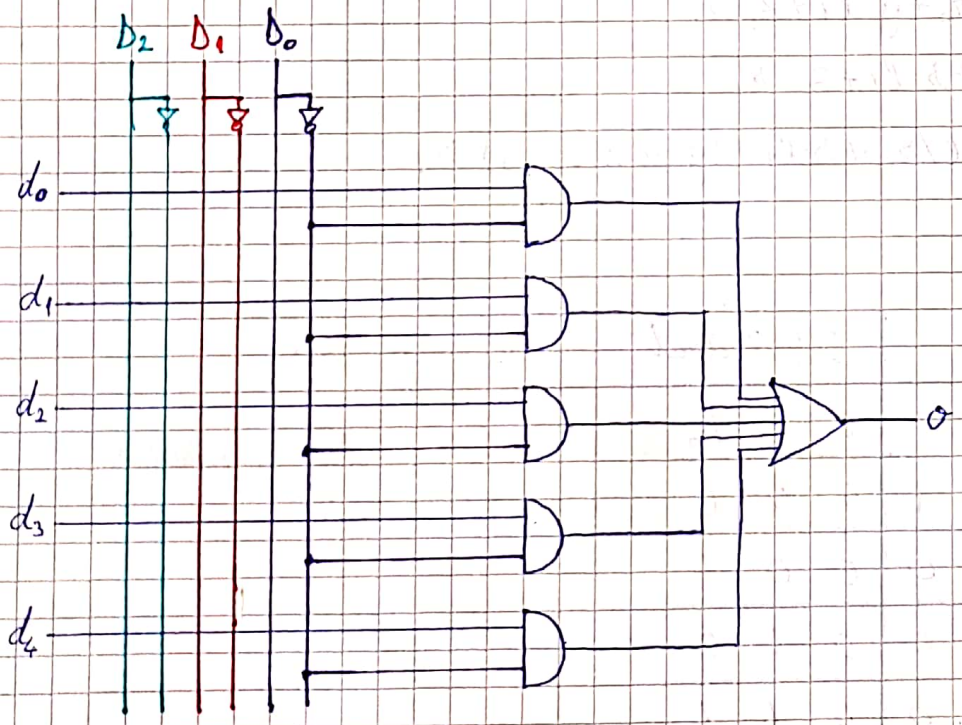
MUX* - degenerat fiindcă nu folosim 8 configurații de intrare ci doar 5

→ Dispozitiv de prenormalizare



→ Structura internă a unui MUX*





CAP III. Analiza funcțională și sinteza dispozitivelor de înmulțire binară

3.1. Metode de înmulțire binară

X – deînmulțitul

Y – înmulțitorul

a) paper & pencil

Vorbim doar despre numere fără semn.

$$
\begin{array}{l}
1100 \ldots\ldots\ldots\ldots Y \\
1011 \ldots\ldots\ldots\ldots X \\
\hline
\end{array}
$$

$$
\text{produse de 1 bit}\left\{
\begin{array}{ll}
1100 & X_0 \cdot 4 \cdot 2^0 \\
1100 & X_1 \cdot 4 \cdot 2^1 \\
0000 & X_2 \cdot 4 \cdot 2^2 \\
1100 & X_3 \cdot 4 \cdot 2^3
\end{array}\right\} +
$$

$$
\begin{array}{l}
\hline
10000100 \ldots\ldots\ldots\ldots P = \sum_{i=0}^{3} X_i \cdot Y \cdot 2^i
\end{array}
$$

HW implementation – combinațională

- folosim un Carry Save Adder pt. că avem nevoie de un sumator multioperant (multi-operand adder 8 bit)
- nu e nevoie de shifter
- avem nevoie de:
  - 2 registre pt. stocarea X și Y
  - multiplicant gating (aici trecerea unui semnal prin niște porți și ca să îi permită să devină 0 at. când ai 111+1)
  - multioperand adder 8 bit

b) Keep partial products fixed

$$
\begin{array}{rl}
1100 & X \\
1011 & Y \\
\hline
\boxed{00000000} & P_0 = 0 \\
1100 & X_0 = Y \cdot 2^0 \quad / + \\
\hline
\boxed{00001100} & P_1 = P_0 + X_0 Y 2^0 \\
1100 & X_1 \cdot Y \cdot 2^1 \quad / + \\
\hline
\boxed{00100100} & P_2 = P_1 + X_1 Y \cdot 2^1 \\
0000 & X_2 \cdot Y \cdot 2^2 \quad / + \\
\hline
\boxed{00100100} & P_3 = P_2 + X_2 \cdot Y \cdot 2^2 \\
1100 & X_3 \cdot Y \cdot 2^3 \quad / + \\
\hline
10000100 & P_4 = P_3 + X_3 \cdot Y \cdot 2^3 = P
\end{array}
$$

→ pasul de iterație : $P_{i+1} = P_i + X_i * Y * 2^i$; $i \geq 0$, $P_0 = 0$

HW implementation :

- un registru pe 8 biți pt. produsele parțiale
- Leftshift (MUX)
- Adder pe 8 biți
- 2 registre pe 4 biți pt. X și Y

c) Keep 1-bit products fixed

Iteration Step : $P_i = P_i + X_i \cdot Y$

$P_{i+1} = P_i * 2^{-1}$ cu $i \geq 0$, $P_0 := 0$

1100	Y
$1011 = X_3X_2X_1X_0$	X
00000000	$P_0 = 0$
$\boxed{1100}$	$X_0 \cdot Y \cdot 2^0$ / +
00001100	$P_0 = P_0 + X_0 \cdot Y$
00001100	$P_1 = P_0 * 2^{-1}$
$\boxed{1100}$	$X_i \cdot Y$ / +
001001100	$P_1 = P_1 + X_1 \cdot Y$
001001100	$P_2 = P_1 * 2^{-1}$
$\boxed{0000}$	$X_2 \cdot Y$ / +
001001100	$P_2 = P_2 + X_2 \cdot Y$
001001100	$P_3 = P_2 * 2^{-1}$
$\boxed{1100}$	$X_3 \cdot Y$ / +
10000100	$P_3 = P_3 + X_3 \cdot Y$
10000100	$P_4 = P_3 * 2^{-1} = P$

Avem nevoie de:

- 8 bit register pt. produsele parțiale cu Right Shift
- 4 bit Adder
- 2 register pe 4 biți pt. X și Y

- Dintre a), b) și c), c)-ul este cea mai rapidă
- Metoda c) e preferată pt. înmulțire secvențială, iar b) pt. combinațională

3.2. Dispozitiv secvențial pt. înmulțirea nr. în SM

Cd. X, Y – nr. fracționare reprez. în SM.

$X = x_7 \mid x_6 x_5 x_4 x_3 x_2 x_1 x_0$

$Y = \underbrace{y_7}_{\text{semn}} \mid \underbrace{y_6 y_5 y_4 y_3 y_2 y_1 y_0}_{\text{magnitudine}}$

$P = \underbrace{p_{15}}_{\text{bit de semn}} \mid p_{14} p_{13} \ldots p_2 p_1 p_0$

- $p_0 = 0$ și e necesar pt. a avea răsp. pe 16 biți
- $p_{15} = x_7 \oplus y_7$
- Algoritmul (numit Multiplier 2):

declare registre: A[7:0], Q[7:0], M[7:0], COUNT[2:0];

declare bus: INBUS[7:0], OUTBUS[7:0];

BEGIN: A:=0, COUNT:=0;
INPUT: M:=INBUS; } se execută când semnalul de control $\{c_0\}$ e activ

Q:=INBUS; ← activat de semnalul de control $\{c_1\}$

TEST1: if Q[0]=0 then go to RSHIFT;

ADD: A[7:0]:=A[6:0]+M[6:0]; ----------> $\{c_2\}$

RSHIFT: A[7]:=0, A[6:0].Q:=A.Q[7,1];
INCREMENT: COUNT:=COUNT+1; } ----------> $\{c_3\}$

TEST2: if COUNT7 ≠ 1 then go to TEST1;

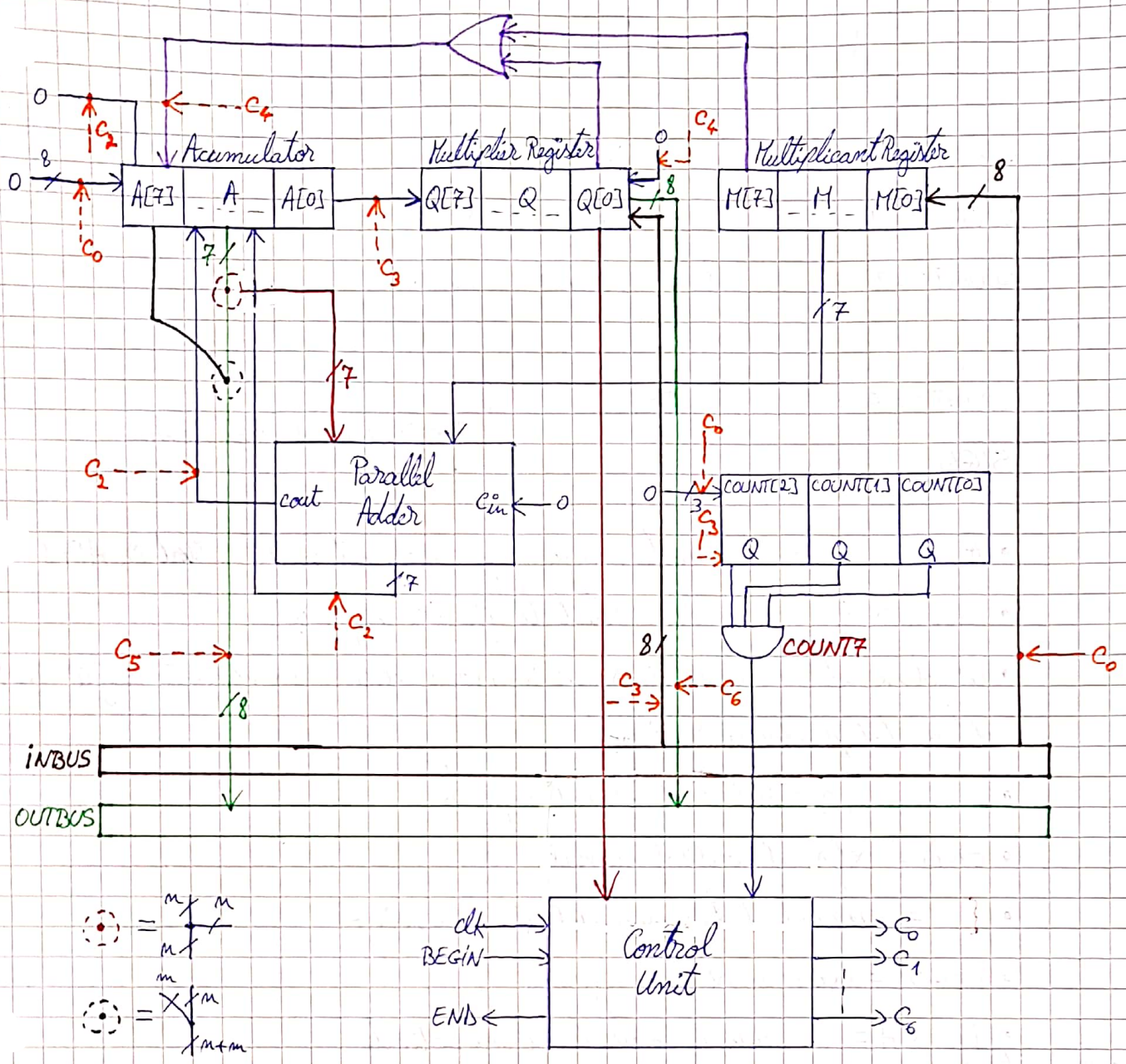
SIGN: A[7]:=M[7] exor Q[0], Q[0]:=0; ----------> $\{c_4\}$

OUTPUT: OUTBUS:=A; ----------> $\{c_5\}$

OUTBUS:=Q; ----------> $\{c_6\}$

END: ----------> {END}

→ Implementare hardware:



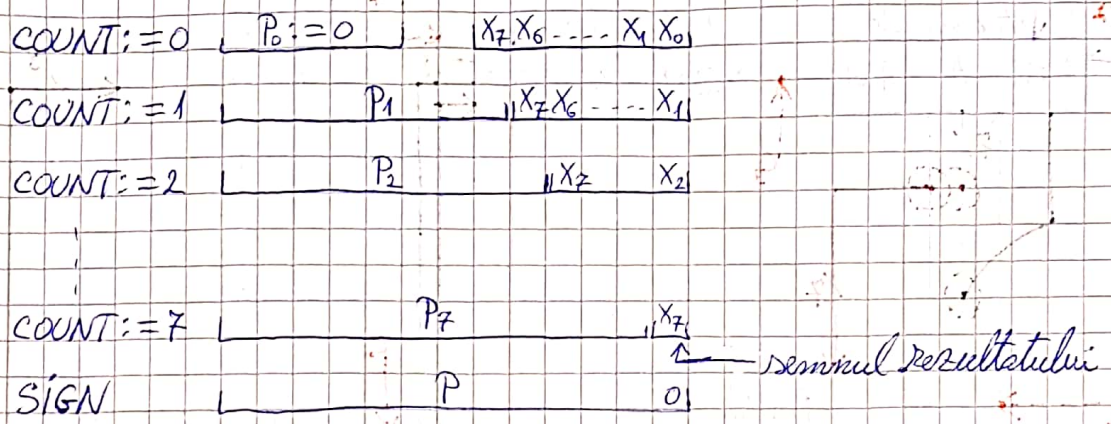
m – semnal de control

Obs:

- Acest algoritm se bazează pe 3.1.c (M păstrează produsele de 1 bit; produsul final este salvat în A concatenat cu Q)
- Produsele parțiale sunt stocate în A concatenat cu c.m. semnificative poziții din Q)

→ Mod de funcționare

$$\begin{cases} P_i := P_i + X_i \cdot Y - \text{TEST1, ADD} \\ P_{i+1} = P_i * 2^{-1} - \text{RSHIFT} \end{cases}$$



- A este pe 8 biți pt. a evita overflow și carry out va fi salvat în A[7]
- Numărătorul e incrementat la fiecare shiftare
- Când se setează bitul de semn X_7, vom pune 0 la eticheta SIGN și se obține tot produsul în A.Q
- $C_0, C_1, \ldots, C_6$ = internal control signals
- END = external control signal
- { = internal status signal (Q[0], COUNT7)
- BEGIN = external status signal
- I.C.S.A = Semnale de control interne activate

referitor la ex. ce urmează
- e COUNT3 pt. că sunt nr. pe 4 biți
- era COUNT7 pt. că sunt nr. pe 8 biți

Ex: $X = -0{,}625 = -5 * 2^{-3} = 1.101_{SM}$

$Y = -0{,}875 = -7 * 2^{-3} = 1.111_{SM}$

$P = 0.1000110 = 1000011 * 2^{-6} = 35 * 2^{-6}$

A	Q(X)	M(Y)	COUNT	i.C.S.A
0000		1111	00	C_0
	1101			C_1
+111				
0111				C_2
0011	1110		01	C_3
0001	1111		10	C_3
+111				
1000				C_2
0100	0111		11	C_3
			*COUNT3==1	
0100	0110			C_4
0100				C_5
	0110			C_6

Ex : $X = -5 = 1101.SM$

$Y = +6 = 0110.SM$

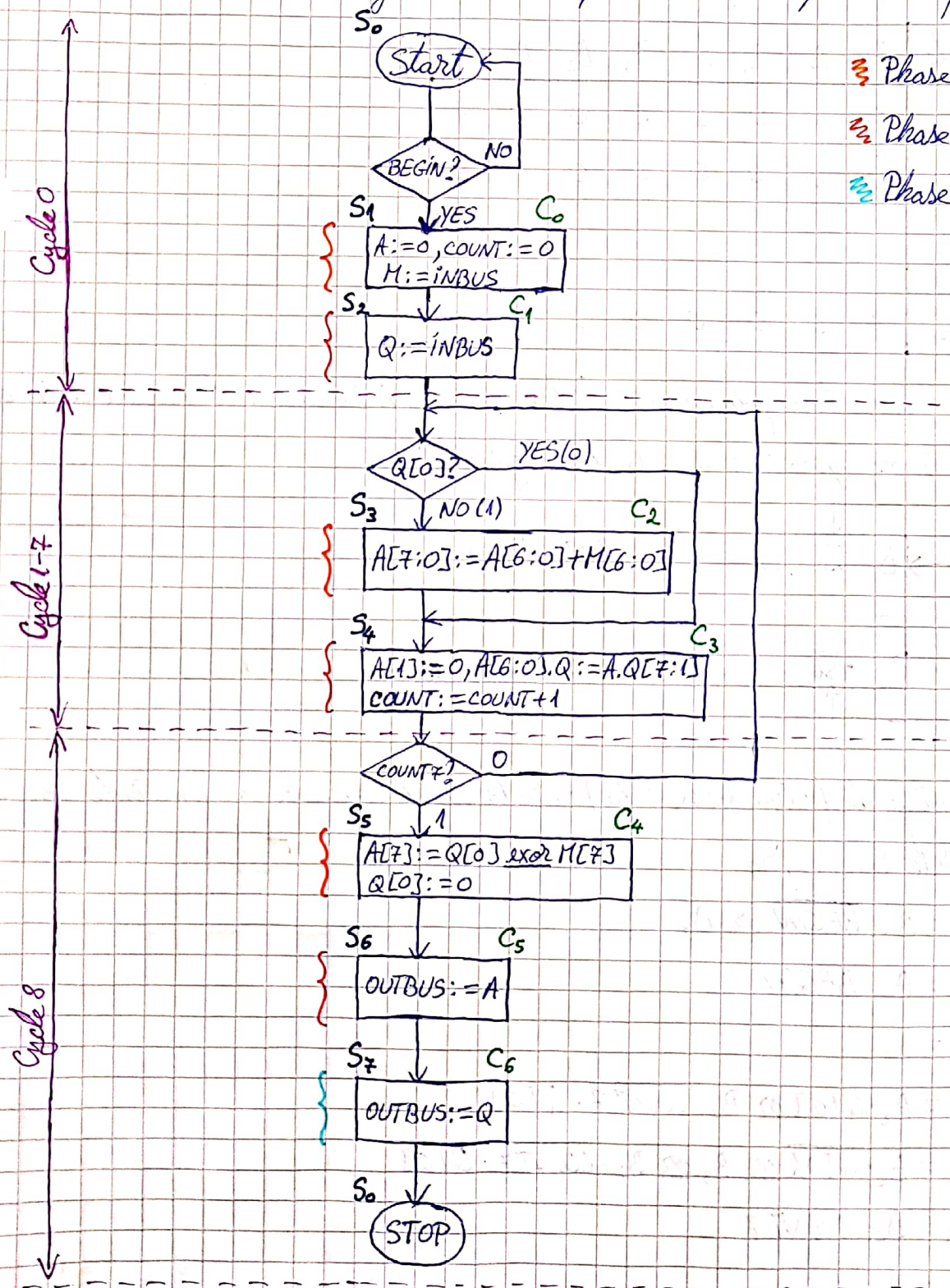
$P = 10011110 = -11110 = -30$

A	Q	M	COUNT
0000		0110	00
	1101		
+110			
0110			
0011	0 10		01
0001	1011		10
+110			
0111			
0011	1101		11
1001	1110		

Curs 10

3.3. Elemente de sinteza unităților de control

- construirea unui FSM care să genereze semnalele de control C_0, C_1 etc. în ordinea corresp. secvențială
- o metodă studiată anterior pt. realizarea FSM este STATE TABLE, dar pe lângă aceasta vom mai studia alte 2 metode
- avem nevoie de o diagramă de stări pe care acum o reprez. sub formă de organigramă



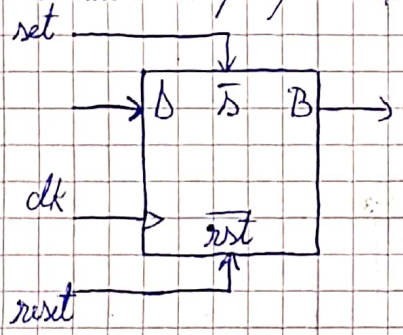
(A) One-hot encoding

- va asocia fiecărei stări câte un elem. de stocare
- dezavantaje: folosește toate elementele de stocare
- avantaje: ecuațiile de excitare și ieșire se pot deriva în mod direct din flow-chart.

- State variables: $B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0$
- State encoding:

State	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
S_0	0	0	0	0	0	0	0	1
S_1	0	0	0	0	0	0	1	0
⋮								
S_7	1	0	0	0	0	0	0	0

- pt. implementare folosim un bistabil de tip D în care not. ieșirea cu B ca să nu ne suprapunem cu Q din algoritm.



$B(t+1) = D$

- Ecuațiile de feedback (excitație): arată când urmează să fim într-una din stările $S_0, \dots, S_7$

$D_0 = B_0 \cdot \overline{BEGIN}$ or B_7

$D_1 = B_0 \cdot BEGIN$

$D_2 = B_1$

$D_3 = B_2 \cdot Q[0]$ or $B_4 \cdot \overline{COUNT7} \cdot Q[0]$

$D_4 = B_2 \cdot \overline{Q[0]}$ or B_3 or $B_4 \cdot \overline{COUNT7} \cdot \overline{Q[0]}$

$D_5 = B_4 \cdot COUNT7$

$D_6 = B_5$

$D_7 = B_6$

- Ecuațiile de ieșire:

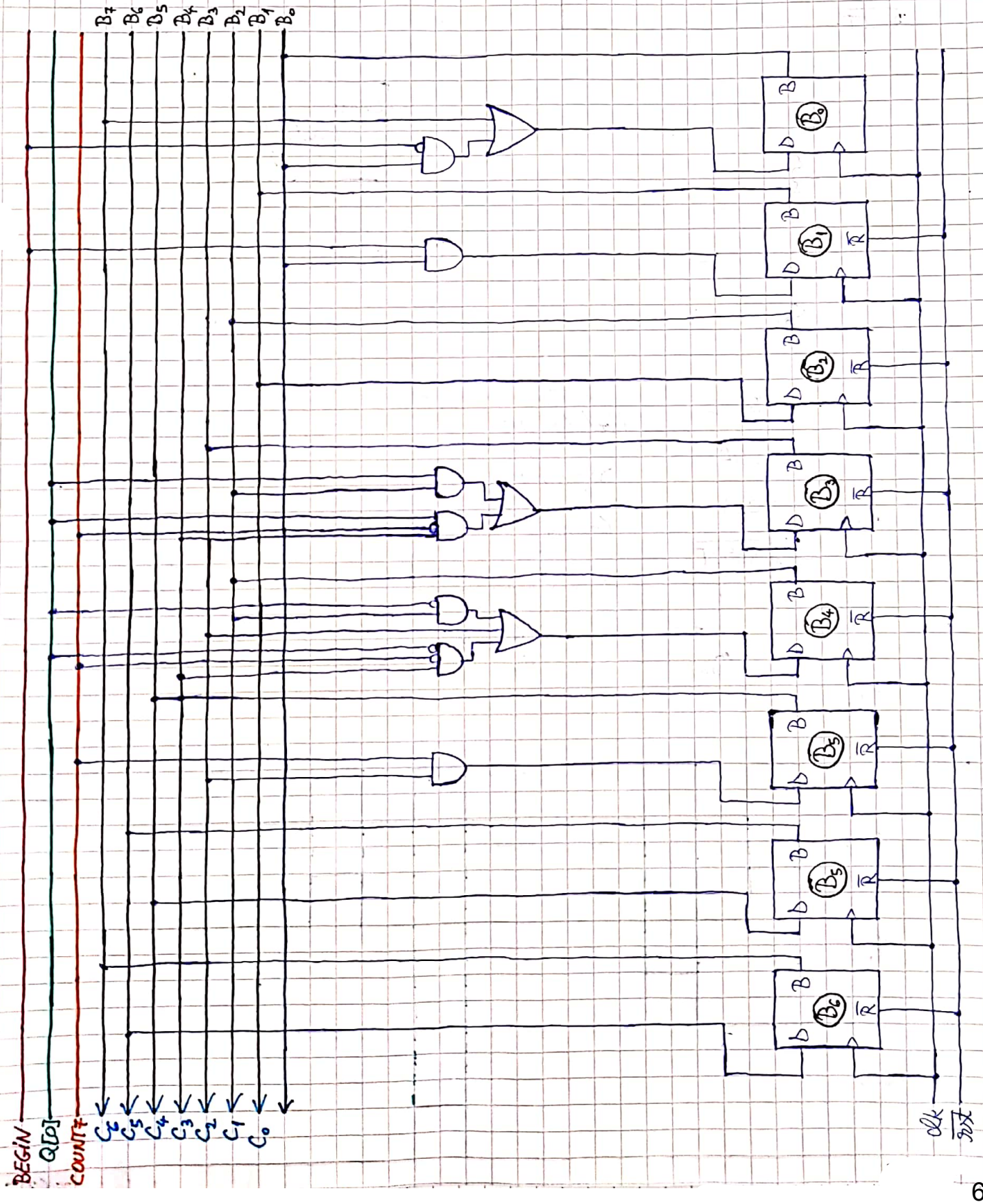
$ENB = B_0$ $\qquad$ $C_3 = B_4$

$C_0 = B_1$ $\qquad$ $C_4 = B_5$

$C_1 = B_2$ $\qquad$ $C_5 = B_6$

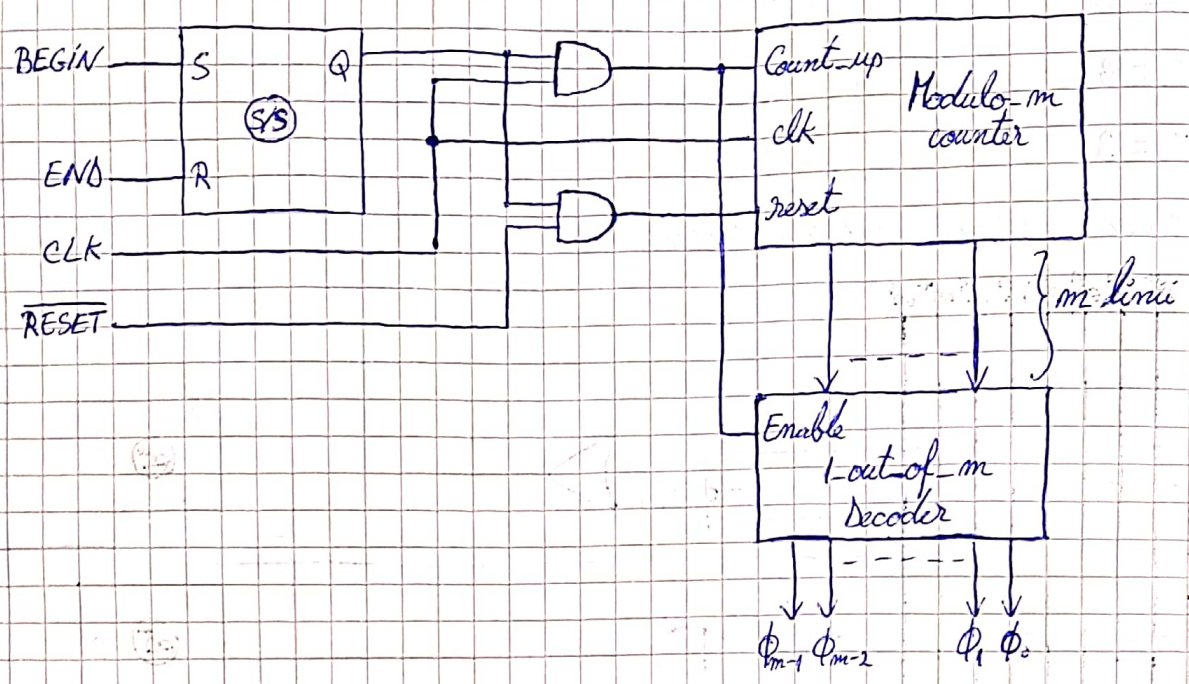
$C_2 = B_3$ $\qquad$ $C_6 = B_7$

- Arhitectura (implementarea fizică):

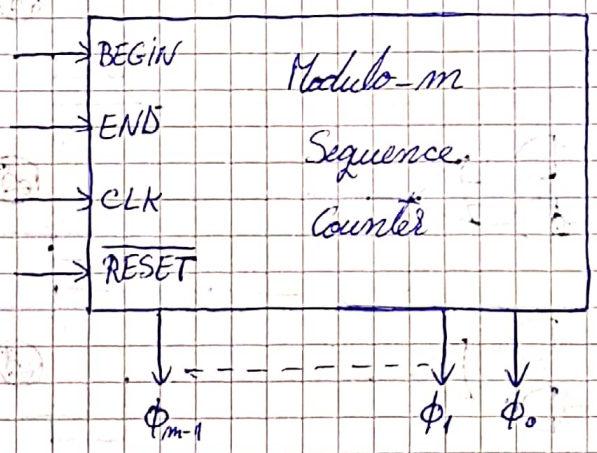


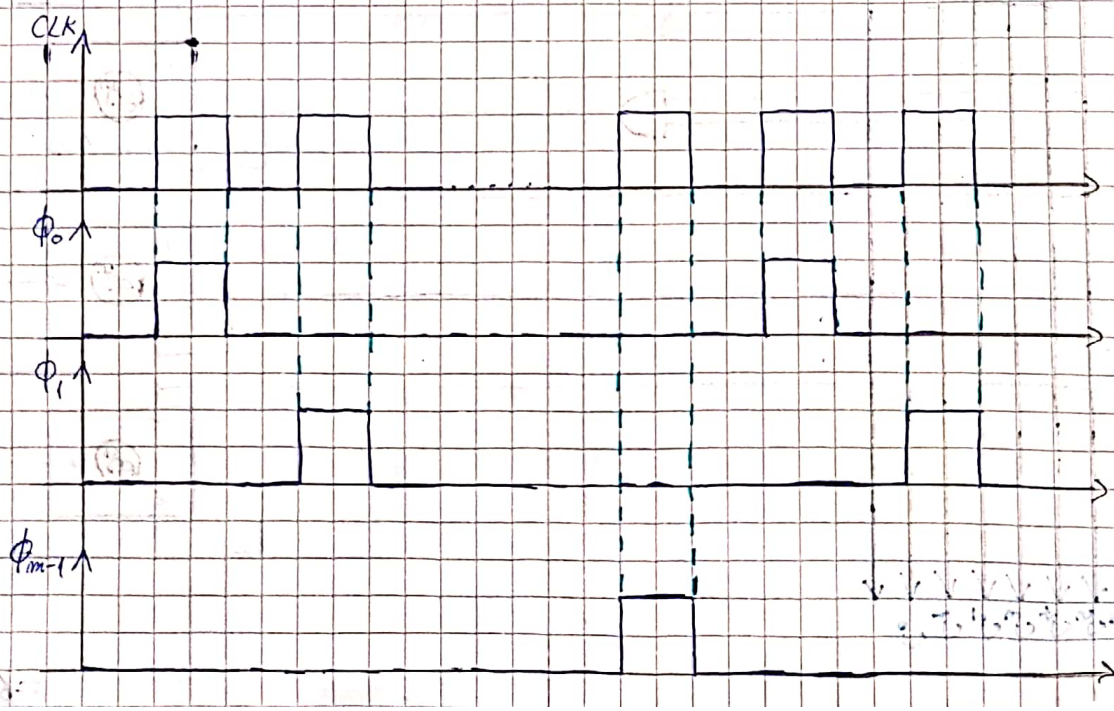
(B) Sequence Counter Method

- folosim un latch SR



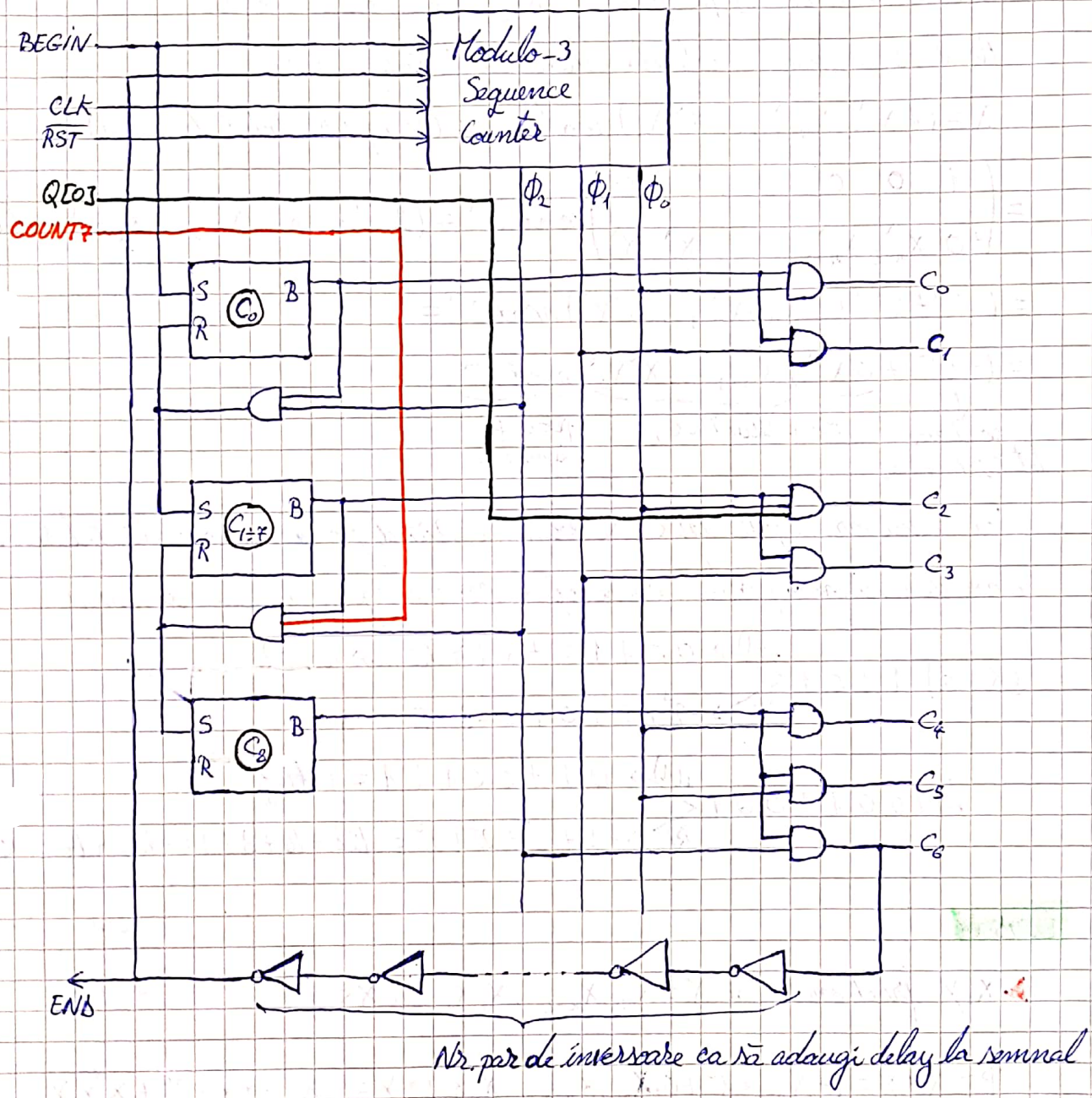
Sau





- pt. fiecare ciclu al unității de control se prevede un latch SR distinct.
- când u.c. începe execuția unui ciclu, resp. latch SR se activează și atunci când se părăsește un ciclu, latch-ul se dezactivează
- val. lui m este dată de nr. max de faze din ciclurile algoritmului

Implementarea arhitecturii:



3.4. Dispozitiv secvențial de înmulțire binară pt. nr. în C2 după procedura lui Robertson.

Opțiuni de înmulțire a nr. în C2:

- $C_2 \to SM$; * în SM; $SM \to C2$ (greoaie)
- interpretarea lui Robertson: (R.I.)

Se cd. nr. X în C_2 pe n biți: $X_{C_2} = 1X'_{n-2}X'_{n-3}\ldots X'_1X'_0$

Acest nr. e negativ datorită bitului de 1. Putem scrie:

$X = (1X'_{n-2}X'_{n-3}\ldots X'_1X'_0) \bmod 2^n =$ (se ignoră bitul de semn)

$$= \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ +0 & X'_{n-2} & X'_{n-3} & \ldots & X'_1 & X'_0 \end{pmatrix} \bmod 2^n = (2^{n-1} + 0X'_{n-2}X'_{n-3}\ldots X'_1X'_0) \bmod 2^n =$$

$$= (-2^n + 2^{n-1} + 0X'_{n-2}X'_{n-3}\ldots X'_1X'_0) \bmod 2^n =$$

$$= (\underbrace{-2^{n-1}}_{\text{ponderea bitului de semn}} + \underbrace{0X'_{n-2}X_{n-3}\ldots X'_1X'_0}_{\text{nr. pozitiv în } C_2}) \underbrace{\bmod 2^n}_{\text{poate fi ignorat}}$$

Val. unui nr. negativ în C2 se obține scăzând ponderea asociată bitului de semn din val. nr. pozitiv care se obține prin ștergerea bitului de semn.

ex: a) 1101 $\xrightarrow{*(-1)}$ 0011 = +3 (rută normală)

$\xrightarrow{RI}$ $-2^3 + 0101 = -8 + 5 = -3$

b) 10011001 $\xrightarrow{*(-1)}$ 01100111 = +103

$\xrightarrow{RI}$ $-2^7 + 2^4 + 2^3 + 2^0 = -128 + 16 + 8 + 1 = -120 + 17 = -103$

Curs 11

▶ X, Y - fracționare și $X = X_{n-1}X_{n-2}\ldots X_1X_0$ cu $X<0$

$$P = X*Y = (-X_{n-1}*2^0 + 0.\underbrace{X_{n-2}X_{n-3}\ldots X_1X_0}_{X^* \text{ pozitiv, în } C_2, \text{ în SM}})*Y = \underbrace{-X_{n-1}*Y*2^0}_{\text{correction}} + X^**Y$$

Relații pt. R.I. în C2:

Dacă X e în C2 cu reprezentarea: $X = X_{n-1}X_{n-2}\ldots X_1X_0$

→ val. lui X ca întreg este: $-X_{n-1}*2^{n-1} + 0X_{n-2}X_{n-3}\ldots X_1X_0$

→ val. lui X ca fracționar: $-X_{n-1}*2^0 + 0.X_{n-2}X_{n-3}\ldots X_1X_0$

• 11 → 3 ← $0{,}75 = 3 * 2^{-2}$ unde 2 = nr. de biți

▶ $Y<0$

Știm că pt. nr. în SM: $\begin{cases} P_i := P_i + X_i \cdot Y \text{ - add} \\ P_{i+1} := P_i * 2^{-1} \text{ - rightshift} \end{cases}$

Încercăm să adaptăm algoritmul de mai sus pt. C2

- produsele parțiale au același semn cu deînmulțitul
- bitul de semn e prelucrat ca ∀ alt bit (avantaj C2)

Folosim algoritmul numit Multiplier 3

Q - înmulțitorul X

M - deînmulțitul Y

A - acumulator (registru în care se adună)

declare register: A[7:0], Q[7:0], M[7:0], COUNT[2:0], F; ← flag

declare bus: INBUS[7:0], OUTBUS[7:0];

BEGIN: A:=0, COUNT:=0, F:=0; * ---------- {C_0}

INPUT: M:=INBUS;

Q:=INBUS; * ---------- {C_1}

TEST1: if Q[0]=0 then go to RSHIFT,

ADD: A:=A+M, F:=[F or (Q[0] & M[7])]; * ---------- {C_2}

RSHIFT: A[7]:=F, A[6:0].Q:=A.Q[7:1]; * ---------- {C_3}

INCREMENT: COUNT:=COUNT+1;

TEST2: if COUNT7 ≠ 1 then go to TEST1,

↳ nu am efectuat toate cele 7 iterații

TEST3: if Q[0]=0 then go to OUTPUT;

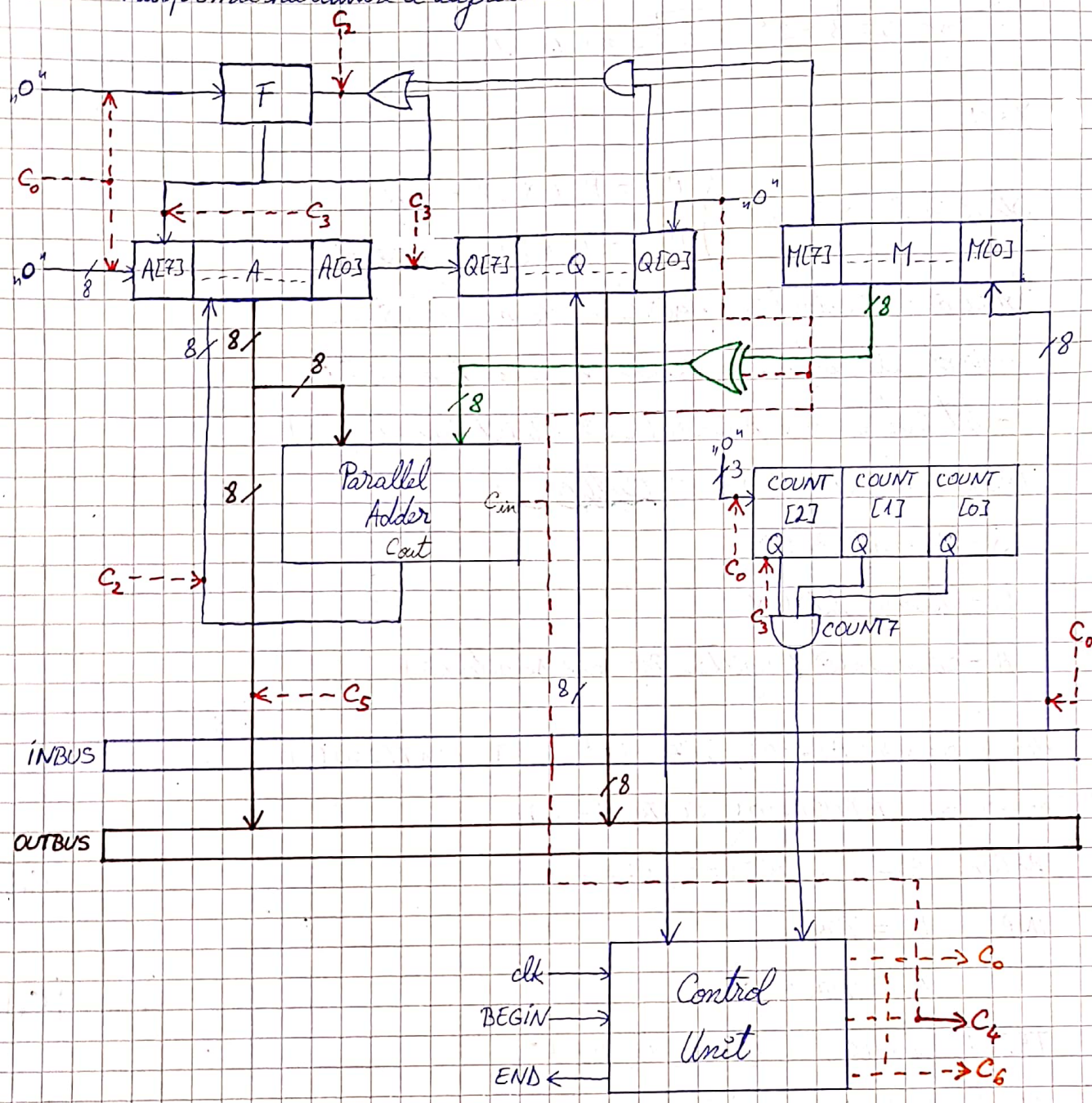
CORRECTION: A:=A-M, Q[0]:=0; * ---------- {C_2, C_4}

OUTPUT: OUTBUS:=A; * ---------- {C_5}

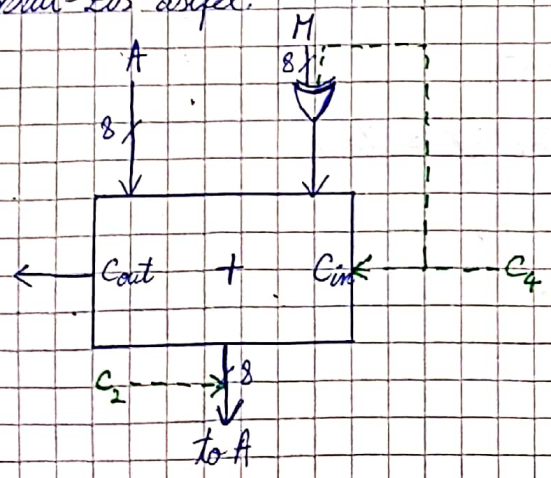
OUTBUS:=Q; * ---------- {C_6}

END: ---------- ➤ {END}

Platforma hardware a algoritmului:



Sumatorul paralel, spre deosebire de cel din SM, e pe 8 biți și e structurat propriu-zis astfel:



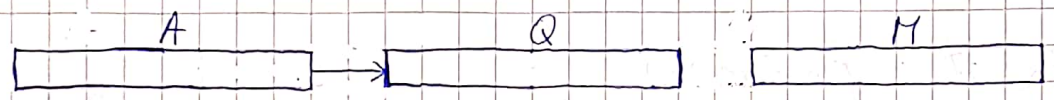
• Rezolvarea situației de overflow aici: flagul F

Dacă $M > 0 \Rightarrow F = 0$

<u>Ex</u>: $X = X_7 X_6 X_5 X_4 X_3$

$Y = 1 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0$

$$\begin{cases} P_i := P_i + X_i * Y \\ P_{i+1} := P_i * 2^{-1} \end{cases} \quad ; P_0 := 0$$



$P_0 := 0$ $\quad$ 0 0 0 0 0 0 0 0 $\quad$ $X_7 X_6 X_5 X_4 X_3$ 1 0 0 $\quad$ 1 $Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0$

$P_1 := 0$ $\quad$ 1 0 0 0 0 0 0 0 0 $\quad$ $X_7 X_6 X_5 X_4 X_3$ 1 0

1 0 0 0 0 0 0 0 0 $= -2^8 = -256$ NU 0 $\Rightarrow$

→ asta dacă punem în A[7] pe M[7]

$\Rightarrow$ Nu putem pune A[7] := M[7] direct, și cât timp Q[0] = 0, A[7] = 0

Deci:

0 0 0 0 0 0 0 0 0 $\quad$ $X_7 X_6 X_5 X_4 X_3$ 1 0

$P_2 := 0$ $\quad$ 0 0 0 0 0 0 0 0 0 $\quad$ $X_7 X_6 X_5 X_4 X_3$ 1

$P_3 := Y \cdot 2^2$ $\quad$ 1 1 $Y_6 Y_5 Y_4 Y_3 Y_2 Y_1$ $\quad$ Y_0 0 0 $\quad$ $X_7 X_6 X_5 X_4 X_3$

→ semnul lui M și de acum F rămâne 1 pt. totdeauna

→ pt. nr. întregi, dar dacă erau fracționare, aveam 2^3.

• Particularitățile algoritmului în funcție de semnele lui X și Y:

X_7	Y_7	Correction	Arithmetic RSHIFT
0	0	NO	NO
0	1	NO	YES
1	0	YES	NO
1	1	YES	YES

→ Shiftare aritmetică: se păstrează bitul de semn

ex: 1 1 0 0
→ 1 1 1 0

→ Shiftare logică: se adaugă 0

ex: 1 1 0 0
→ 0 1 1 0

Ex: pt. nr. fracționare

$X = -7 * 2^{-3} = 1.111_{SM} = 1.001_{C_2}$

$Y = -4 * 2^{-3} = 1.100_{SM} = 1.100_{C_2}$

Internal Control Signals Activated

F	A	Q	M	COUNT	ICSA
0	0000		1100	00	C_0
		1001	and		C_1
	+ 1100 / 1100	or			C_2
1	1110	0100		01	C_3
1	1111	0010		10	C_3
1	1111	1001		11	C_3
				* COUNT3==1	
	− 1100 / 0011	0			C_2, C_4
	0011				C_5
		1000			C_6

Deci răsp. obținut este: $0.0111000 = 111 * 2^{-4} = 7 * 2^{-4}$

Verif: $-7 * 2^{-3} * (-4) * 2^{-3} = -7 * 2^{-3} * (-1) * 2^{-1} = +7 * 2^{-4}$

Ex: pt. nr. întregi cu overflow

$X = -5 = -1011_{C_2}$

$Y = +6 = 0110_{C_2}$

F	A	Q	M	COUNT
0	0000		0110	00
		1011		
	+0110			
	0110			
	0011	0101		01
	+0110			
	1001			
	0100	1010		10
	0010	0101		11
	-0110			
	1100			
	1110	0010		

shiftare aritmetică

$P = 11100010 = -127+64+32+2 = -30$

Curs 12

3.5. Dispozitiv secvențial pt. înmulțirea binară a nr. în C_2 după procedura lui Booth

$$\begin{cases} P_i := P_i + X_i Y \\ P_{i+1} := P_i * 2^{-1} \end{cases}$$ clk-uri diferite

- Dacă la P_i adunăm 0, irosim un clk degeaba, așa că am putea direct shifta la dreapta
- Cu cât nr. de biți de 1 e mai mare, cu atât latența este mai mare pt. înmulțire
- Booth: ia în considerare câte 2 biți

Ex: $X = 0.\overset{2^{-1}}{1}\overset{2^{-2}}{1}1111\,10$

Y

$$P = X * Y = (2^{-1} + 2^{-2} + \ldots + 2^{-6}) * Y = Y * \frac{2^0 - 2^{-6}}{2-1} = +2^0 * Y - 2^{-6} * Y$$

Obs: În loc de 6 operații de adunare (Robertson), putem efectua 2 operații: 1 adunare și 1 scădere

$X = \boxed{0\ 1}\ 1\ 1\ 1\ 1\ \boxed{1\ 0}$ → cere op. de scădere

↳ cere op. de adunare

Perechile (0,0) și (1,1) nu cer nicio operație

La fiecare moment verificăm perechile (X_i, X_{i-1}) și avem:

→ 0 1 : +Y

→ 1 0 : −Y

→ 0 0, 1 1 } nicio operație

Dar pt. bitul X_0 nu avem pereche, deci se adaugă un bit X_{-1} cu valoarea 0 și weight $\frac{1}{2}$ of X_0 (adică mereu avem 10 la final)

Procedeul Booth folosește recodificarea Booth a înmulțitorului: se scanează de la dr. la stg. și se înlocuiește fiecare pereche $X_i X_{i-1}$ cu următorul digit:

$X_i X_{i-1}$

→ 0 0 → 0

→ 0 1 → 1

→ 1 0 → $\bar{1}$ (minus 1)

→ 1 1 → 0

Recodificarea Booth: sistem de numerație redundant (fiecare val. se poate reprez. în minim 2 feluri) ce folosește biți cu semn (sign digit)

ex: $X = -1 * 2^{-3}$

Rank / Operand	X_3	X_2	X_1	X_0	X_{-1}
	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
X_{SM}	1	0	0	1	
X_{C_2}	1	1	1	1	0
X_B	0	0	0	$\bar{1}$	

← Booth

Deci $X_B = \bar{1} * 2^{-3} = -1 * 2^{-3}$

$X_{C_2} == X_B = Y_{C_2} * X_{C_2} == Y_{C_2} * X_B$

Bitul de semn (de pe P_{i+1}) se face prin shiftare aritmetică a lui P_i (se copiază c. m. semnif. bit a lui)

Algoritmul Multiplier 4:

declare register: A[7:0], Q[7:-1], M[7:0], COUNT[2:0];

declare bus: INBUS[7:0], OUTBUS[7:0]

BEGIN: A:=0, COUNT:=0;

INPUT: M:=INBUS;

Q[7:0]:=INBUS[7:0], Q[-1]:=0;

TEST1: if Q[0]Q[-1]=01 then A:=A+M go to TEST2;

if Q[0]Q[-1]=10 then A:=A-M;

TEST2: if COUNT7=1 then go to OUTPUT;

RSHIFT: A[7]:=A[7], A[6:0],Q:=A.Q[7:0]

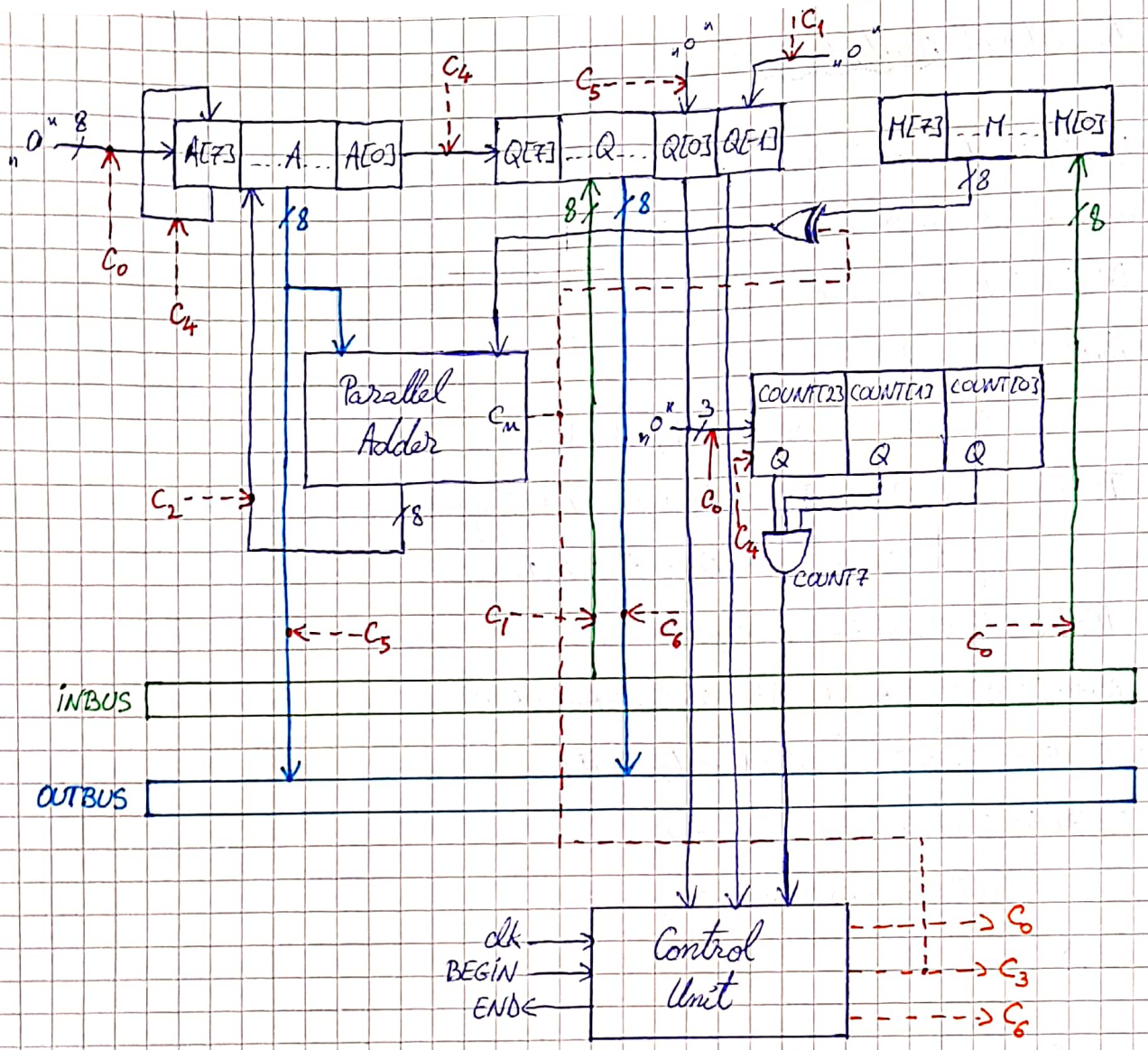
INCREMENT: COUNT:=COUNT+1 go to TEST1

OUTBUS: OUTBUS:=A, Q[0]:=0;

OUTBUS[7:0]:=Q[7:0];

END

- Q[0] se pune pe 0 pt. că aşa se extind nr. fracţionare
- algoritmul este pt. nr. fracţionare
- pt. nr. întregi rezultatul trebuie shiftat la dr. şi bitul de semn recirculat



Ex: $X = -5 * 2^{-3} = 1.011_{C_2}$; $Y = -7 * 2^{-3} = 1.001_{C_2}$

A	Q	M	CONT	I.C.S.A.
0000	Q[-1] 1011 0	1001	00	C_0
				C_1
-1001 / 0111 / 0011 / 0001	11011 / 11101		01 / 10	C_2, C_3 / C_4 / C_4
+1001 / 1010 / 1101 / -1001 / 0100	01110		*COUT3==1 11	C_2 / C_4 / C_2, C_3
0100	0			C_5
	0110			C_6

Deci $P = 0.10000110 - 100011 * 2^{-6} = (32+3) * 2^{-6} = 35 * 2^{-6}$

Ex: $X = +7 = 0111_{C_2}$; $Y = -3 = 1101_{C_2}$

A	Q	M	COUNT
0000		1101	00
	0111 0		
-1101			
0011			
0001	1011 1		01
0000	1101 1		10
0000	0111 0		11 + COUNT3 = 1
+1101			
1101			
1110	10110		
$\boxed{1110}$	$\boxed{1011}$		

Deci $P = 1.1101011 = -128+64+32+11 = -21$

3.6. Structuri combinaționale sistolice pt. înmulțirea binară

Structuri sistolice = vectori / struct. matriceale

Cd.: X, Y - unsigned integer pe 4 biți

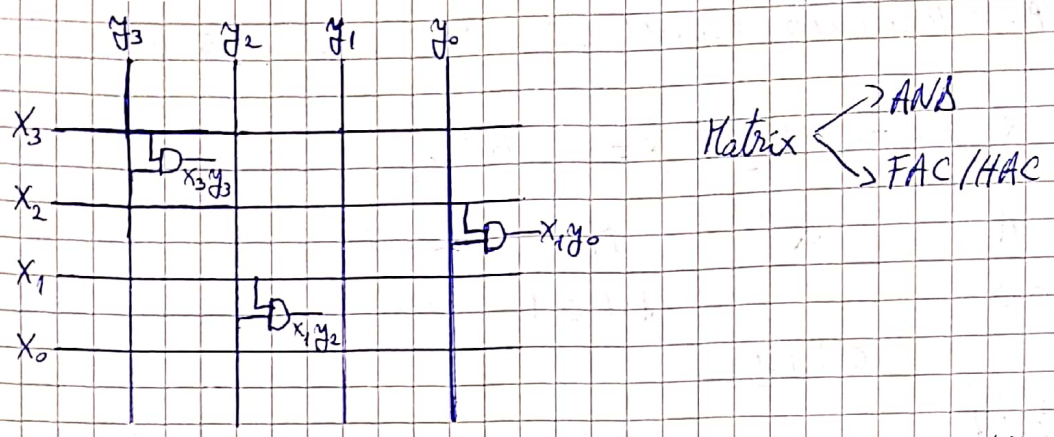
$$X = X_3X_2X_1X_0 = \sum_{i=0}^{3} X_i \cdot 2^i$$

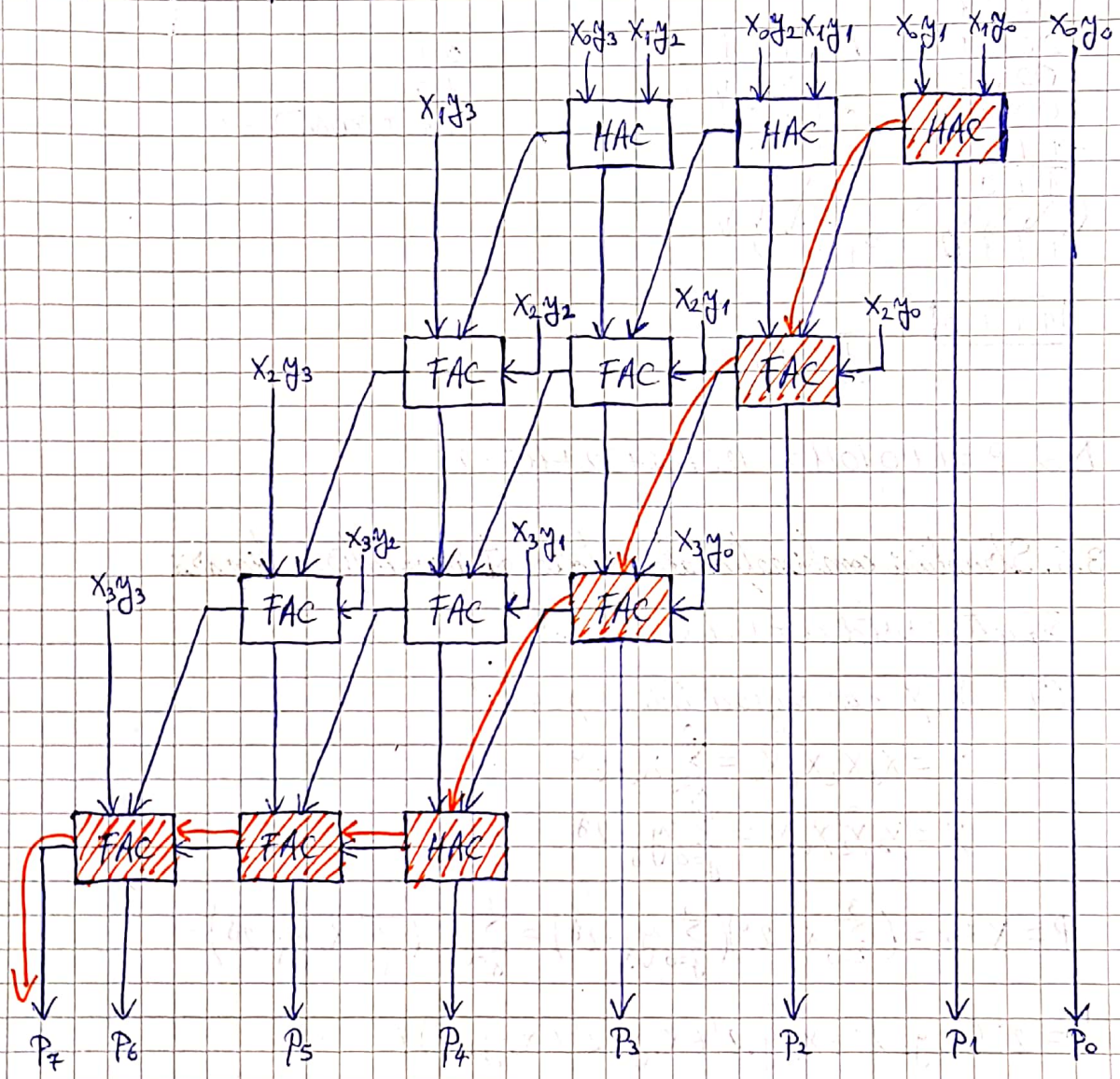
$$Y = Y_3Y_2Y_1Y_0 = \sum_{j=0}^{3} y_j \cdot 2^j$$

$$P = X * Y = \left(\sum_{i=0}^{3} X_i \cdot 2^i\right)\left(\sum_{j=0}^{3} y_j \cdot 2^j\right) = \sum_{j=0}^{3} 2^i \left(\sum_{j=0}^{3} X_i y_j \cdot 2^j\right) =$$

$$= 2^0(X_0y_0 \cdot 2^0 + X_0y_1 2^1 + X_0y_2 \cdot 2^2 + X_0y_3 2^3) + \ldots +$$
$$+ 2^3(X_3y_0 2^0 + X_3y_1 2^1 + X_3y_2 2^2 + X_3y_3 2^3)$$

$$\begin{aligned} P = {} & 2^5 \cdot X_3y_3 + 2^5(X_3y_2 + X_2y_3) + \\ & 2^4(X_3y_1 + X_2y_2 + X_1y_3) + \\ & 2^3(X_3y_0 + X_2y_1 + X_1y_2 + X_0y_3) + \\ & 2^2 \quad (X_2y_0 + X_1y_1 + X_0y_2) + \\ & 2^1 \quad (X_1y_0 + X_0y_1) + \\ & 2^0 \quad X_0y_0 \end{aligned}$$





~ - cea mai lungă cale prin circuit

$$D = \underset{\text{matrix}}{D_{AND}} + \underset{\text{matrix}}{D_{FAC}} = 1d + 2 * (\underbrace{1d}_{HAC} + \underbrace{2(n-2)d}_{FAC})$$

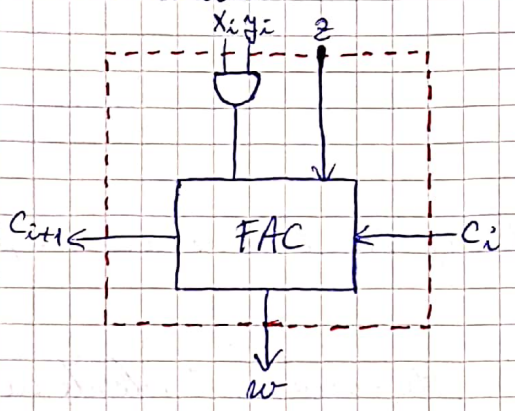
$$D = (4n-5)d$$

aria → $$A = \underset{\text{matrix}}{A_{AND}} + \underset{\text{matrix}}{A_{FAC}} = n^2 \cdot \underset{\text{gate}}{A_{AND}} + n(n-1)\underset{\text{matrix}}{A_{FAC}}$$

$A = O(n^2)$ – A are complexitatea $O(n^2)$

Curs 13

– suprapunerea matricei de porți și cu cea de FAC-uri este dificilă, așa că găsim o alternativă:

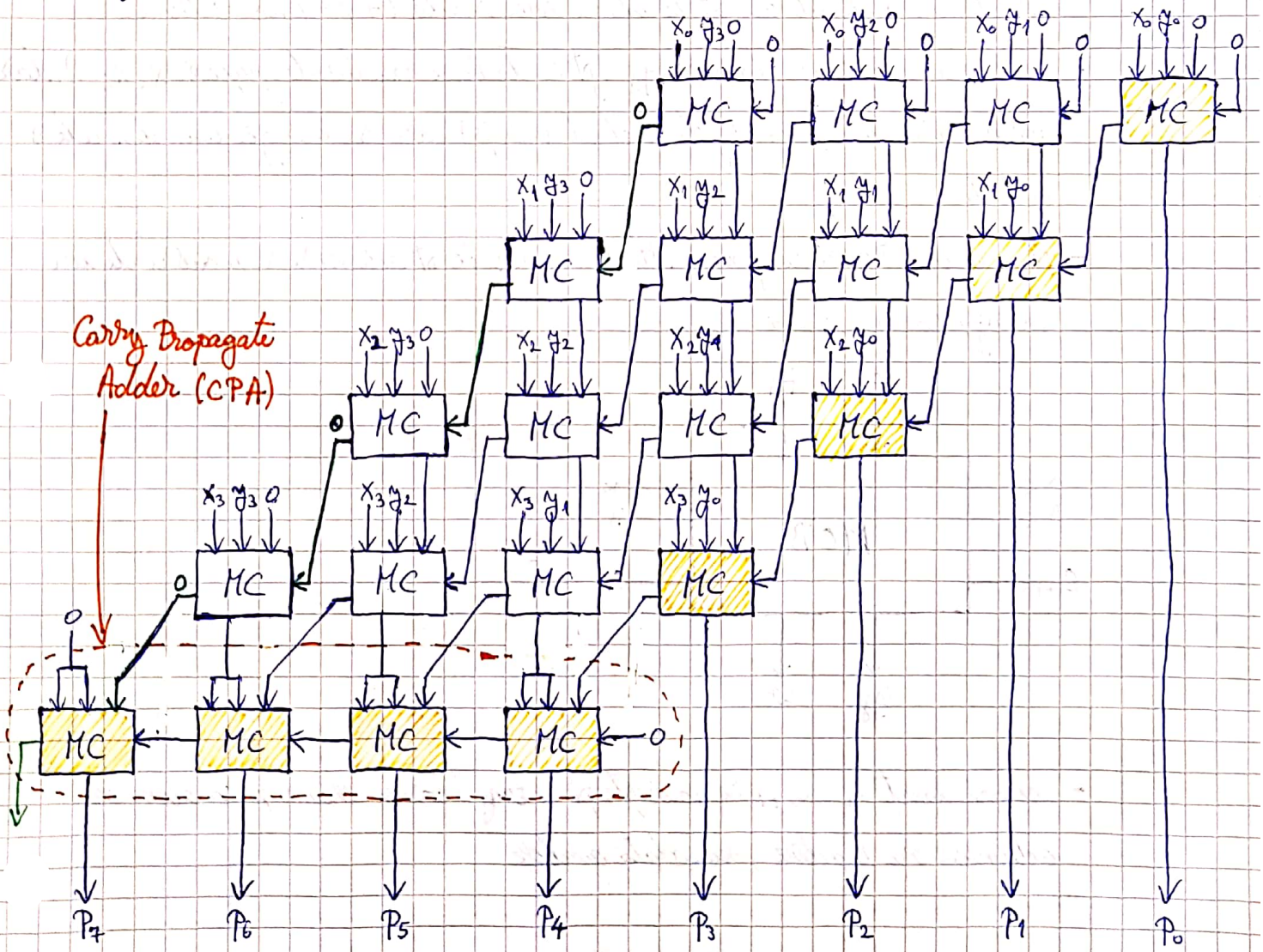


z – ține locul produsului parțial curent

w – produsul parțial următor

Refacem arhitectura folosind MC-uri (folosim metoda de înmulțire cu produsele parțiale fixe)

$$\begin{cases} P_{i+1} := P_i + X_i \cdot Y \cdot 2^i \\ P_0 := 0 \end{cases}$$



- linia z_5 este mereu 0 pt. că de la început se adună numai 2 biți de 0 + încă un bit care poate fi orice
- Latența se calculează prin calea cea mai scurtă

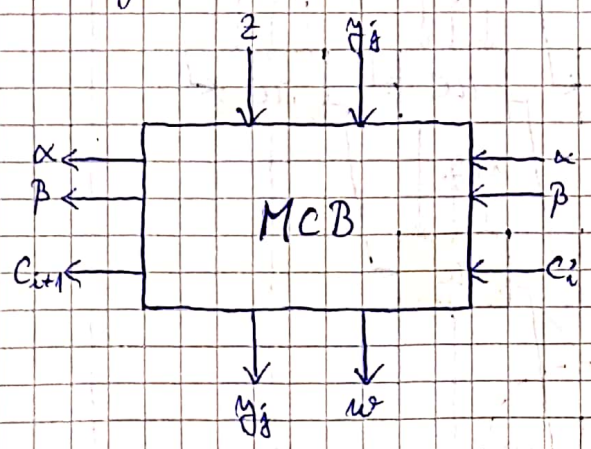
$$D = \underbrace{1d}_{ANS} + 2n \cdot 2d = (4n+1)d$$

$$A = n(n+1) \cdot A_{MC} \Rightarrow A = O(n^2)$$

- Fiecare nivel adună câte un produs parțial (prima linie de coloane e pt. primul produs, etc.)
- Pe ultima linie, prin acea intrare dublă, MC-urile sunt folosite ca FAC-uri

Arhitectura pt procedura de înmulțire a lui Booth

- avem nevoie de o celulă MCB care trebuie să poată păstra produsul parțial curent nemodificat ori să le adune sau scadă (+/-/no arithm)
- MCB are ca intrare doar pe y_j (pt. că trebuie analizate grupuri $x_i x_{i-1}$), dar avem nevoie de 2 semnale de control (α, β) care să aleagă una din cele 3 operații
- pt. a evita CPA de la final, MCB are nevoie pt. adunare și scădere de un carry / borrow chain



- același lanț $c_i - c_{i+1}$ se folosește pt. carry / borrow chain pt. că avem ori adunare ori scădere, niciodată ambele

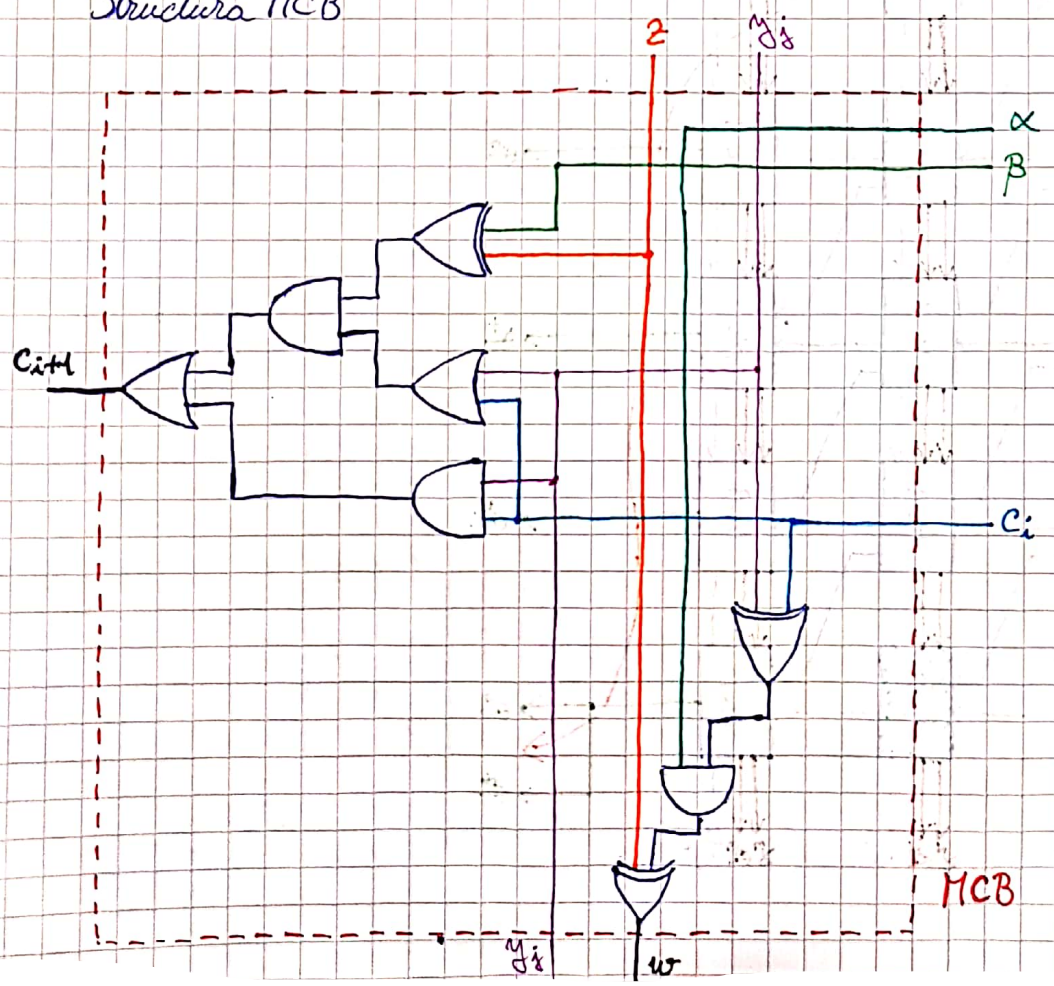
Operation	Outputs	
	w	c_{i+1}
Add	$w = z \oplus y_j \oplus c_i$	$c_{i+1} = z y_j + z c_i + y_j \cdot c_i$
Substract	$w = z \oplus y_i \oplus c_i$	$c_{i+1} = \bar{z}(y_j + c_i) + y_j \cdot c_i$
No arith.	$w = z$	irrelevant

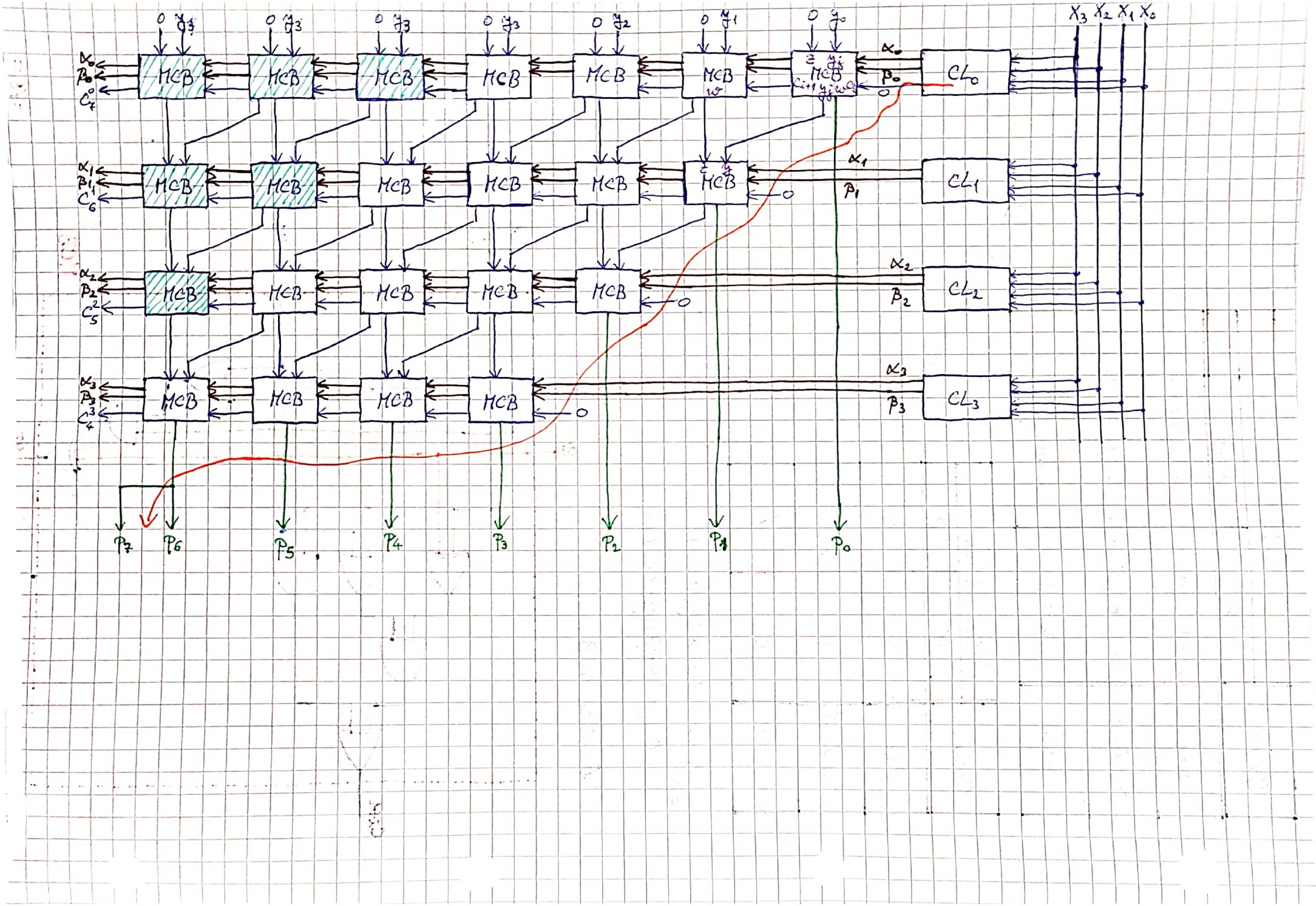
$\Rightarrow$

Operation	Control variable	
	α	β
Add	1	0
Substract	1	1
No arith	0	d

$$\Rightarrow \begin{cases} w = z \oplus \alpha (y_i \oplus c_i) \\ c_{i+1} = (z \oplus \beta)(y_i + c_i) + y_j c_i \end{cases}$$

Structura MCB




MCB
CL0
CL1
CL2
CL3
X3 X2 X1 X0
P7 P6 P5 P4 P3 P2 P1 P0

- Rolul MCB (): extindere de semn pt. că operanzii sunt cu semn
- Latența se face pe calea critică ()

Ex: $X = +5 = 0\,1\,0\,1\,0$

$Y = -6 = 1010$

00000000	P_0
11110100	$y \cdot 2^0$ / −
00000110	$P_1 = P_0 - y \cdot 2^0$
11101000	$y \cdot 2^1$
11110100	$P_2 = P_1 + y \cdot 2^1$
11010000	$y \cdot 2^2$
00100100	$P_3 = P_2 - y \cdot 2^2$
10100000	$y \cdot 2^3$
11100010	$P_4 = P_3 + y \cdot 2^3$

$P = 11100010 = -128 + 64 + 32 + 2 = -30$

Input				Booth				Outputs							
X_3	X_2	X_1	X_0	X_{B_3}	X_{B_2}	X_{B_1}	X_{B_0}	α_3	β_3	α_2	β_2	α_1	β_1	α_0	β_0
0	0	0	0	0	0	0	0	0	d	0	d	0	d	0	d
0	0	0	1	0	0	1	$\bar{1}$	0	d	0	d	1	0	1	1
0	0	1	0	0	1	$\bar{1}$	0	0	d	1	0	1	1	0	d
0	0	1	1	0	1	0	$\bar{1}$	0	d	1	0	0	d	1	1
1	1	0	0	0	$\bar{1}$	0	0	0	d	1	1	0	d	0	d
1	1	0	1	0	$\bar{1}$	1	$\bar{1}$	0	d	1	1	1	0	1	1
1	1	1	0	0	0	$\bar{1}$	0	0	d	0	d	1	1	0	d
1	1	1	1	0	0	0	$\bar{1}$	0	d	0	d	0	d	1	1

X_{B_i}	α	β
0	0	d
1	1	0
$\bar{1}$	1	1

În urma minimizărilor:

$\beta_0 = 1$ $\qquad$ $\beta_2 = X_2$

$\alpha_0 = X_0$ $\qquad$ $\alpha_2 = X_2 \oplus X_1$

$\beta_1 = X_1$ $\qquad$ $\beta_3 = X_3$

$\alpha_1 = X_1 \oplus X_0$ $\qquad$ $\alpha_3 = X_3 \oplus X_1$

Curs 14

3.6. Metoda Bough-Wooley

Cd. X, Y pe 4 biți cu semn, în C_2, întregi

$$X = -X_3 \cdot 2^3 + \sum_{i=0}^{2} X_i \cdot 2^i$$

$$Y = -Y_3 \cdot 2^3 + \sum_{j=0}^{2} y_j \cdot 2^j$$

Deci $P = X * Y = 2^6 X_3 \cdot Y_3 + 2^5(-X_3Y_2 - X_2Y_3) + 2^4(-X_3Y_1 + X_2Y_2 - X_1Y_3) +$
$+ 2^3(-X_3Y_0 + X_2Y_1 + X_1Y_2 - X_0Y_3) + 2^2(X_2Y_0 + X_1Y_1 + X_0Y_2) + 2^1(X_1Y_0 + X_0Y_1) +$
$+ 2^0 X_0 Y_0$

Pt. a folosi FAC trebuie să transformăm cumva:

$-a \cdot b = b - b - ab = b(1-a) = \bar{a}b - b = a\bar{b} - a$

- Termenul cu ponderea 2^3

$-X_3Y_0 + X_2Y_1 + X_1Y_2 - X_0Y_3 = X_3\bar{Y}_0 + X_2Y_1 + X_1Y_2 + \bar{X}_0Y_3 - X_3 - Y_3 + X_3 + Y_3 -$
$- X_3 - Y_3 = X_3\bar{Y}_0 + X_2Y_1 + X_1Y_2 + \bar{X}_0Y_3 + X_3 + Y_3 - 2(X_3 + Y_3)$

Termenul îl înmulțim cu 2^3 și se propagă la termenul cu 2^4

- Termenul cu ponderea 2^4

$-X_3Y_1 + X_2Y_2 - X_1Y_3 - X_3 - Y_3 = X_3\bar{Y}_1 + X_2Y_2 + \bar{X}_1Y_3 - 2(X_3 + Y_3)$

$\hookrightarrow 2^5$

- Termenul cu ponderea 2^5

$-X_3Y_2 - X_2Y_3 - X_3 - Y_3 = X_3\bar{Y}_2 + \bar{X}_2Y_3 - 2(X_3 + Y_3)$

$\hookrightarrow 2^6$

- Termenul cu ponderea 2^6

$X_3Y_3 - X_3 - Y_3 + 1 + 1 - 2 = X_3Y_3 + \bar{X}_3 + \bar{Y}_3 - 2$

$\hookrightarrow 2^7$

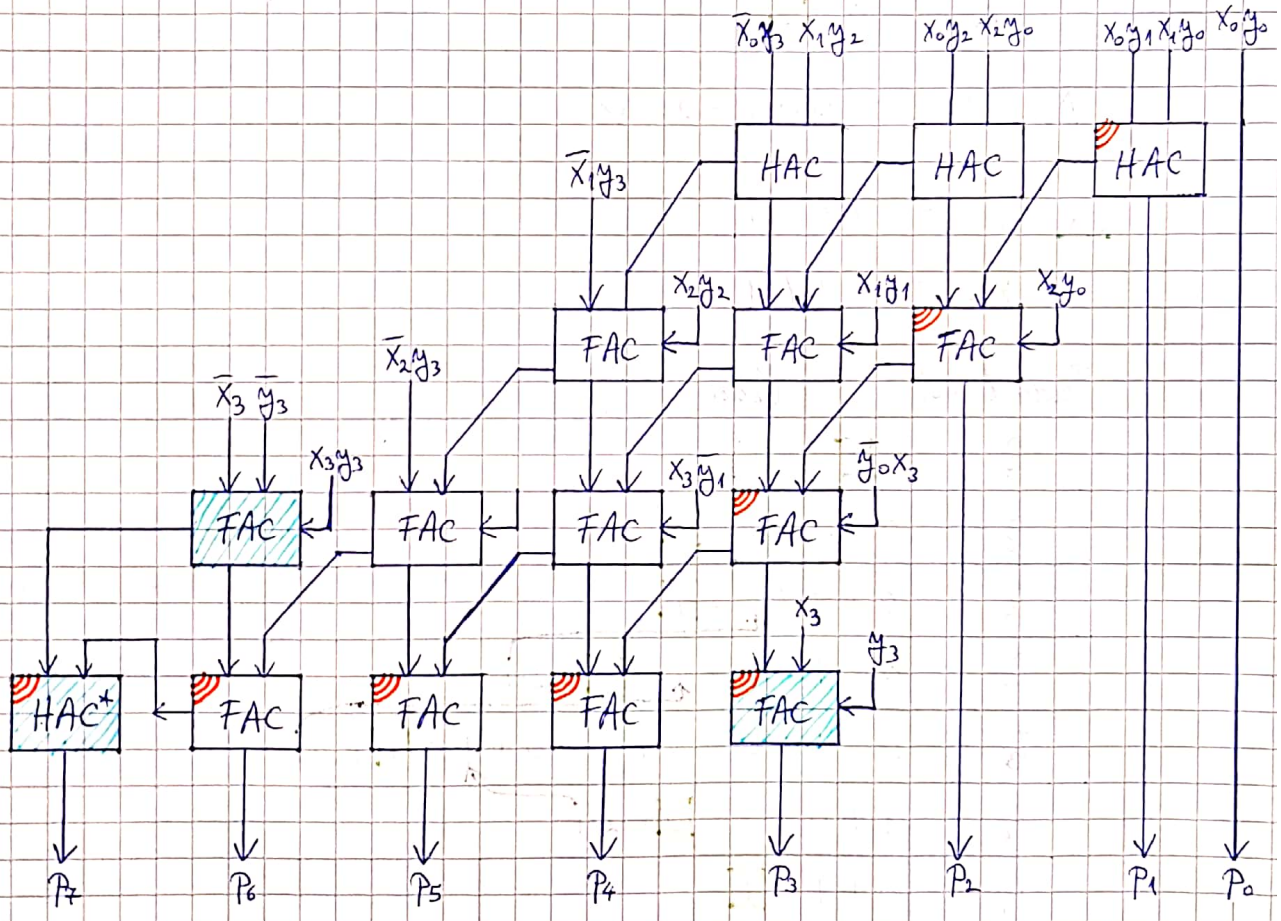
- Termenul cu ponderea 2^7

$-1 = +1 - 2$

$\hookrightarrow 2^8$ (acest bit este al 9-lea și îl asimilăm cu un carry out care, în C_2, nu contează)

- 2 nr. de 4 biți au pt. produs un rezultat pe 7 biți (dar îl reprezentăm pe 8 biți ca să nu avem probleme)
- Așadar:

$$P = 2^7 \cdot 1 + 2^6(x_3y_3 + x_3 + \bar{y}_3) + 2^5(x_3\bar{y}_2 + \bar{x}_2y_3) + 2^4(x_3\bar{y}_1 + x_2y_2 + x_1y_3) +$$
$$+ 2^3(x_3\bar{y}_0 + x_2y_1 + x_1y_2 + x_0y_3 + x_3y_3) + 2^2(x_2y_2 + x_1y_1 + x_0y_2) +$$
$$+ 2^1(x_1y_0 + x_0y_1) + 2^0 x_0 \cdot y_0$$



- Calea critică
- Elemente în plus față de structura romboidă

Ex: $X = -4 = 1100$

$Y = -6 = 1010$

Product bit	Value
P_7	$1 + 1 = \boxed{0} + 2 \cdot 1$
P_6	$1 + 0 + 0 + 1 = \boxed{0} + 2 \cdot 1$
P_5	$1 + 0 + 1 = \boxed{0} + 2 \cdot 1$
P_4	$0 + 0 + 1 = \boxed{1}$
P_3	$1 + 1 + 0 + 1 + 1 + 1 = \boxed{1} + 2^2 \cdot 1$
P_2	$0 + 0 + 0 = \boxed{0}$
P_1	$0 + 0 = \boxed{0}$
P_0	$0 = \boxed{0}$

Deci $P = 00011000 = 24$ ✓

Ex: $X = -3 = 1101$

$Y = +7 = 0111$

Product bit	Value
P_7	$\boxed{1}$
P_6	$0 + 0 + 1 = \boxed{1}$
P_5	$0 + 0 + 1 = \boxed{1}$
P_4	$0 + 1 + 0 + 1 = \boxed{0} + 2 \cdot 1$
P_3	$0 + 1 + 0 + 0 + 1 + 0 + 1 = \boxed{1} + 2 \cdot 1$
P_2	$1 + 0 + 1 = \boxed{0} + 2 \cdot 1$
P_1	$0 + 1 = \boxed{1}$
P_0	$\boxed{1}$

$\Rightarrow P = 11101011 = -128 + 64 + 32 + 11 = -32 + 11 = -21$ ✓

Cele 3 celule inserate afectează mult și aria și latența, deci încercăm să transformăm iar: (Bough–Wooley modificată)

- Termenul cu 2^3

$$x_3\bar{y}_0 + x_2y_1 + x_1y_2 + \bar{x}_0y_3 + x_3 + y_3 = x_3(1-y_0) + x_2y_1 + x_1y_2 + y_3(1-x_0) + x_3 + y_3 =$$

$$= -x_3y_0 + x_2y_1 + x_1y_2 - x_0y_3 + x_3 + y_3 + x_3 + y_3 + 1 + 1 - 2 =$$

$$= \overline{x_3y_0} + x_2y_1 + x_1y_2 + \overline{x_0y_3} + \underbrace{2(x_3+y_3-1)}_{\to 2^4} \qquad (1-ab=\overline{ab})$$

- Termenul cu 2^4

$$x_3\bar{y}_1 + x_2y_2 + \bar{x}_1y_3 + x_3 + y_3 - 1 = \overline{x_3y_1} + x_2y_2 + \overline{x_1y_3} + 2(x_3+y_3-1) - 1 - 1 + 1 =$$

$$= \overline{x_3y_1} + x_2y_2 + \overline{x_1y_3} + 1 + \underbrace{2(x_3+y_3-2)}_{\to 2^5}$$

- Termenul cu 2^5

$$x_3\bar{y}_2 + \bar{x}_2y_3 + x_3 + y_3 - 2 = \overline{x_3y_2} + \overline{x_2y_3} + 2x_3 + 2y_3 - 4 = \overline{x_3y_2} + \overline{x_2y_3} + \underbrace{2(x_3+y_3-2)}_{\to 2^6}$$

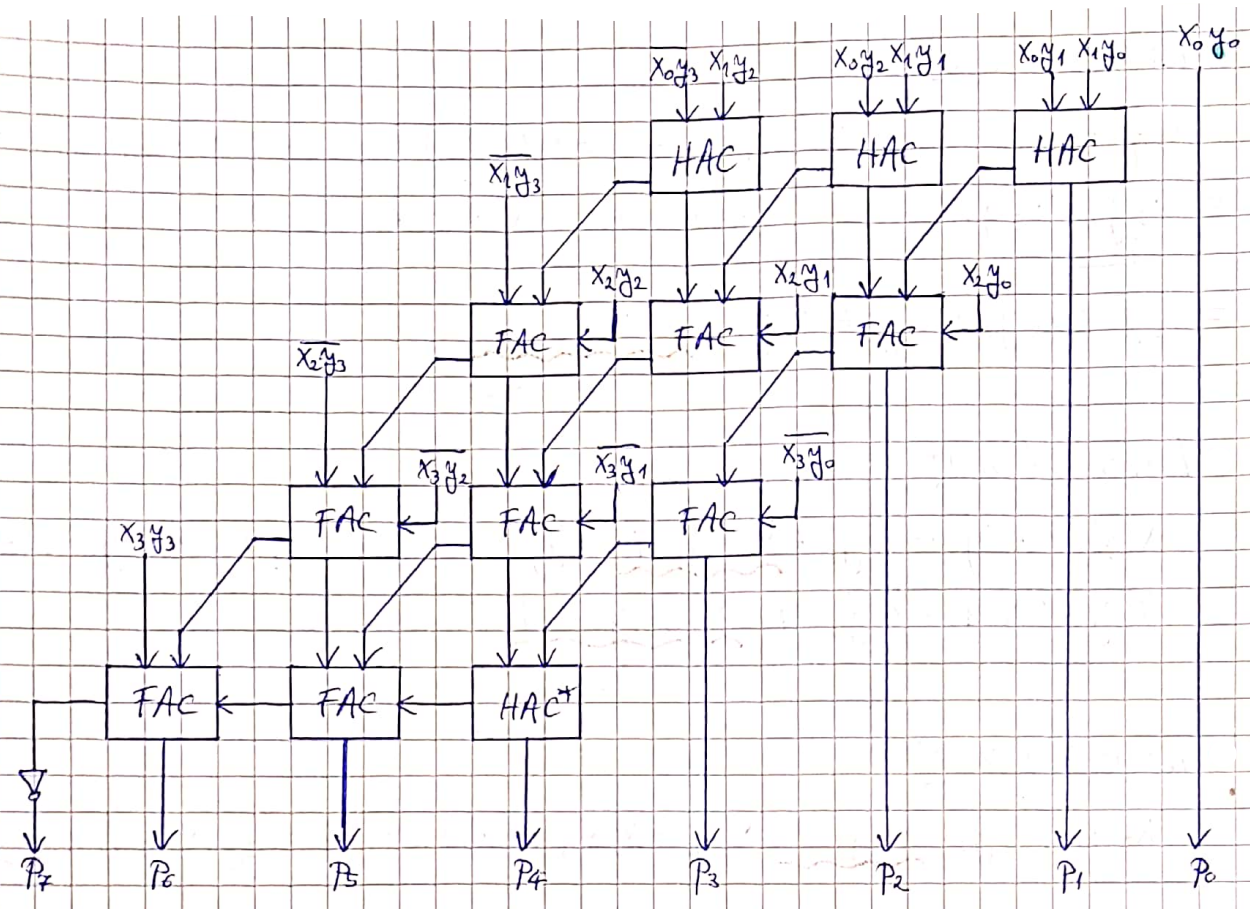
- Termenul cu 2^6

$$x_3y_3 + \bar{x}_3 + \bar{y}_3 + x_3 + y_3 - 2 = x_3y_3$$

- Termenul cu 2^7: rămâne neschimbat

Deci:

$$P = 2^7\cdot 1 + 2^6\cdot x_3y_3 + 2^5(\overline{x_3y_2} + \overline{x_2y_3}) + 2^4(\overline{x_3y_1} + x_2y_2 + \overline{x_1y_3} + 1) +$$
$$+ 2^3(\overline{x_3y_0} + x_2y_1 + x_1y_2 + \overline{x_0y_3}) + 2^2(x_2y_0 + x_1y_1 + x_0y_2) + 2^1(x_1y_0 + x_0y_1) +$$
$$+ 2^0x_0y_0$$



Ex: $X = +4 = 0\,1\,0\,0$

$Y = -5 = 1\,0\,1\,1$

Product bit	Value
P_7	$\boxed{1}$
P_6	$0 + 1 = \boxed{1}$
P_5	$1 + 0 = \boxed{1}$
P_4	$1 + 0 + 1 + 1 + 1 = \boxed{0} + 2^2 \cdot 1$
P_3	$1 + 1 + 0 + 1 = \boxed{1} + 2 \cdot 1$
P_2	$1 + 0 + 0 = \boxed{1}$
P_1	$0 + 0 = \boxed{0}$
P_0	$\boxed{0}$

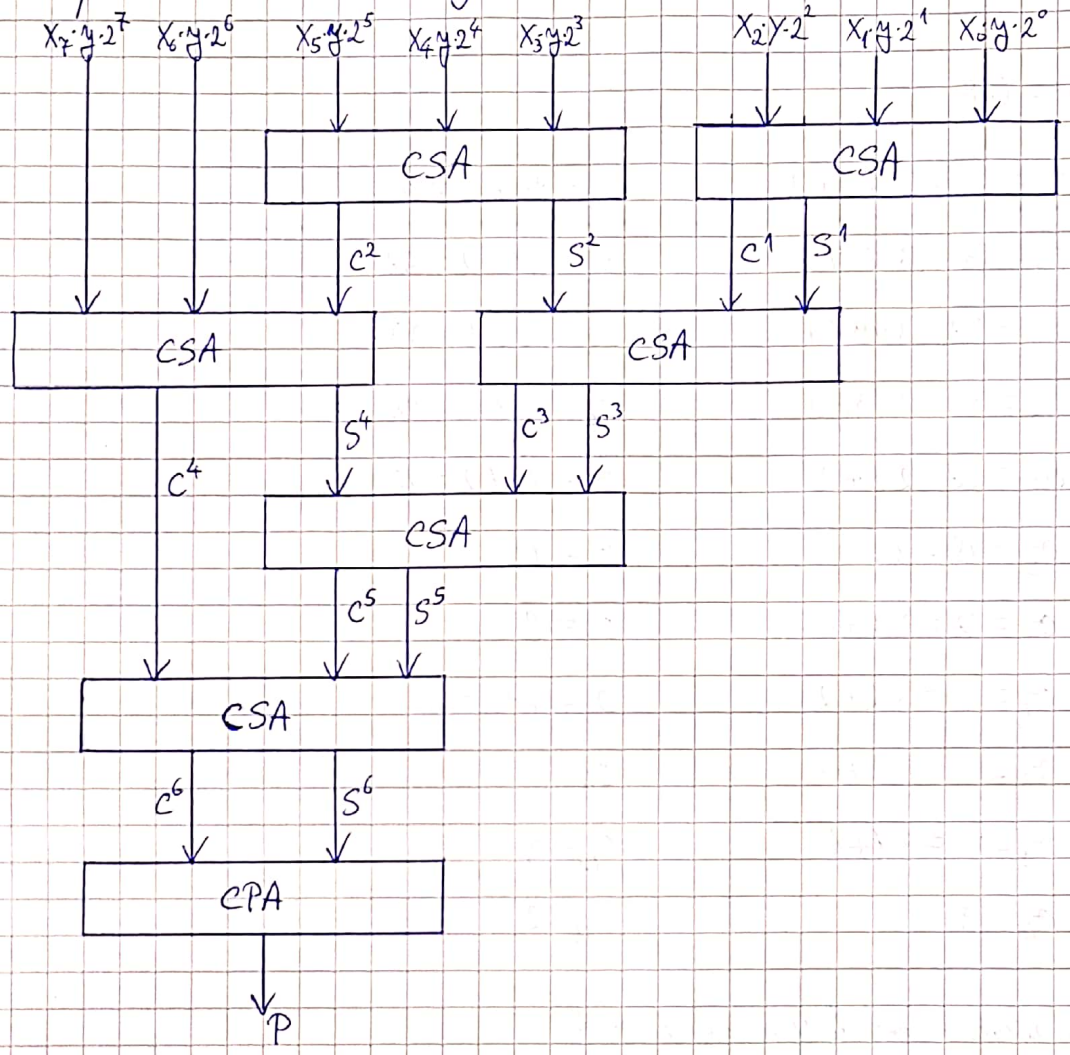
$\Rightarrow P = 1\,1\,1\,0\,1\,1\,0\,0 = -128 + 64 + 32 + 12 = -32 + 12 = -20$ ✓

3.7. Structuri combinaționale arborescente pt. realizarea operației de înmulțire

X, Y - 8 bit, integer, unsigned

$M_i = x_i \cdot y \cdot 2^i$

- se folosesc structuri Carry Save Adder



- CPA = Carry Propagate Adder (com + adder în afară de CSA)
- Nu sunt reprez. în figură lățimile (orice CSA are cu 1 bit mai mult ca cel de pe nivelul superior)
- o_i = nr. vectorilor de ieșire pt. nivelul i
- r_i = nr. vectorilor care nu pot fi adunați în nivelul i cu un CSA ($r_i \in \{0, 1, 2\}$)
- Pt. n biți:

Layer i	No. of input vectors	No. of output vectors
1	$n = 3 \cdot \left[\frac{n}{3}\right] + r_1$	$o_1 = 2 \cdot \left[\frac{n}{3}\right] + r_1$
2	$o_1 = 3 \cdot \left[\frac{o_1}{3}\right] + r_2$	$o_2 = 2 \cdot \left[\frac{o_1}{3}\right] + r_2$
⋮		
m	$o_{m-1} = 3$	$o_m = 2$

recursiv

Ex: $n = 128$

1: $n = 128 = 3 \cdot 42 + 2$ $\quad o_1 = 84 + 2 = 86$

2: $o_1 = 86 = 3 \cdot 28 + 2$ $\quad o_2 = 56 + 2 = 58$

3: $o_2 = 58 = 3 \cdot 19 + 1$ $\quad o_3 = 38 + 1 = 39$

4: $o_3 = 39 = 3 \cdot 13 + 0$ $\quad o_4 = 26 + 0 = 26$

5: $o_4 = 26 = 3 \cdot 8 + 2$ $\quad o_5 = 16 + 2 = 18$

6: $o_5 = 18 = 3 \cdot 6 + 0$ $\quad o_6 = 12$

7: $o_6 = 12 = 3 \cdot 4 + 0$ $\quad o_7 = 8$

8: $o_7 = 8 = 3 \cdot 2 + 2$ $\quad o_8 = 4 + 2 = 6$

9: $o_8 = 6 = 3 \cdot 2 + 0$ $\quad o_9 = 4$

10: $o_9 = 4 = 3 \cdot 1 + 1$ $\quad o_{10} = 3$

11: $o_{10} = 3$ $\quad o_{11} = 2$

Deci avem nevoie de 11 nivele de CSA