CURS S2

CHAPTER 1: COMPUTER ARITHMETICS

1.1 Introduction

→ e conv. să fol. aceeași reprezentare (C2) în toate operațiile.

A * B

A: multiplicand (de înmulțit)
B: multiplier (înmulțitorul)

=> mă interesează biții lui B:

$B = (b_{m-1}\ b_{m-2} \dots b_i \dots b_1\ b_0)_{C2}$

$b_i \in B = \{0,1\}$, $i = \overline{0,m-1}$

Robertson →

$$B_{C2} = -b_{m-1} * 2^{m-1} + \sum_{i=0}^{m-2} b_i * 2^i$$

◎ la pasul $i=0$: $b_0 * A * 2^0$
$i=1$ + $b_1 * A * 2^1$
+ . . .
$i=m-2$ + $b_{m-2} * A * 2^{m-2}$
$i=m-1$ − $b_{m-1} * A * 2^{m-1}$
P

La Robertson, fiecare pas e: o operație aritmetică, o shiftare, clock

(1 000 000 1 => 8+2 = 10 clock)

Booth's Algorithm → te uiți la bitul curent, dar și la cel din dr.

b_i	b_{i-1}	OP
0	0	0
0	1	+A
1	0	-A
1	1	0

! b_{-1} se consideră 0 (când $i=0$)

$(b_{i-1} - b_i) * 2^i * A$

$i=0$: $(b_{-1} - b_0) * 2^0 * A$
$i=1$: + $(b_0 - b_1) * 2^1 * A$
⋮
$i=j-1$: + $(b_{j-2} - b_{j-1}) * 2^{j-1} * A$
$i=j$: + $(b_{j-1} - b_j) * 2^j * A$
$i=j+1$: + $(b_j - b_{j+1}) * 2^{j+1} * A$
⋮
$i=m-2$: + $(b_{m-3} - b_{m-2}) * 2^{m-2} * A$
$i=m-1$: + $(b_{m-2} - b_{m-1}) * 2^{m-1} * A$

$-b_j * 2^j + b_j * 2^{j+1} = b_j * 2^j (-1+2) = b_j * 2^j$

$$\Rightarrow \boxed{P = \left(\sum_{i=0}^{m-2} b_i * 2^i - b_{m-1} * 2^{m-1} \right) * A}$$

- Performanța

1 = adunare
$\bar{1}$ = scădere

B = 0 1 0 1 0 1 0 1 | 0 R: 4
B' = 1 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$ B: 8

=> Alg. lui Booth devine mai puțin eficient când am 1 și 0 izolat.

De aceea, în situațiile în care e mai eficient Booth, acesta se comportă ca Booth și invers.

B = 1 1 1 1 0 0 0 0 | 0 R = 4
B' = 0 0 0 $\bar{1}$ 0 0 0 0 | B = 1

Trebuie să mă uit și în stg. să văd dacă e 0 izolat / 1 izolat...

F' = statutul curent al șirului în care mă aflu

	b_{i+1}	b_i	F	B''	F'
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	0	0
5	1	0	1	$\bar{1}$	1
6	1	1	0	$\bar{1}$	1
7	1	1	1	0	1

0 - șir de zerouri

! referința mea este i

1. Începe un nou șir de zerouri și F' = 0 pt. că sunt într-un șir de zerouri

2. $-A * 2^i + A * 2^{i+1} = A * 2^i (-1+2) = A * 2^i$ (adun un A => B''=1)

3. sunt în int. unui șir de 1 => F' = 1
nu fac nimic în int. lui => B'' = 0

4. Analog cu 3

5. $+A * 2^i - A * 2^{i+1} = -A * 2^i$ => B'' e -1, adică $\bar{1}$
nu s-a sch. situația => F' = 1

7. sunt în șir de 1, nu fac nimic

=> (cu ajut. calculului)

B =	0	1	0	1	0	1	0	1	0
B' =	1	$\bar{1}$	1	$\bar{1}$	1	$\bar{1}$	1	$\bar{1}$	
B'' =	0	1	0	1	0	1	0	1	
F =	0	0	0	0	0	0	0	0	0

MB = 4
R = 4
B = 8

B =	1	1	1	1	1	0	0	0	0	0
B' =		0	0	0	$\bar{1}$	0	0	0	0
B'' =		0	0	0	$\bar{1}$	0	0	0	0
F =	1	1	1	1	1 0	0	0	0	0

R = 4
B = 1
Modified B = 1

ex: X = − 105
Y = − 73

P = 315
735
+7665

105 − 64 = 41; 41 − 32 = 9
73 − 64 = 9

X = 1110 1001 SM
= 1001 0111 C2

Y = 1100 1001 SM
= 1011 0111 C2

M = 1001 0111
−M = 0110 1000

$A'[7] = OVR \cdot \overline{A[7]} + \overline{OVR} \cdot A[7]$

→ avem nevoie de un counter pt. implementare tare overflow

COUNT	OVR	A	Q[8]	Q	F	M
000	0	0000 0000	1	1001 0111	0	1011 0111
	+	0100 1001		−M	1	
		0100 1001				
		0010 0100	1	1100 1011		
001	0	0001 0010	0	1110 0101	1	
010	0	0000 1001	0	0111 0010	1	
011		+0100 1001			1	
	0	0100 1101				
		0010 0110	0	0011 1001	1	
100	0	0001 0011	1	0001 1100	1	
	+	1011 0111		+M	0	
	0	1100 1010	1			
		1110 0101	1	1000 1110		
101	0	1111 0010	1	1100 0111	0	
	+	0100 1001		−M	1	
110	0	0011 1011				
		0001 1101	1	1111 0001		

→ rezultatul final
verific.:

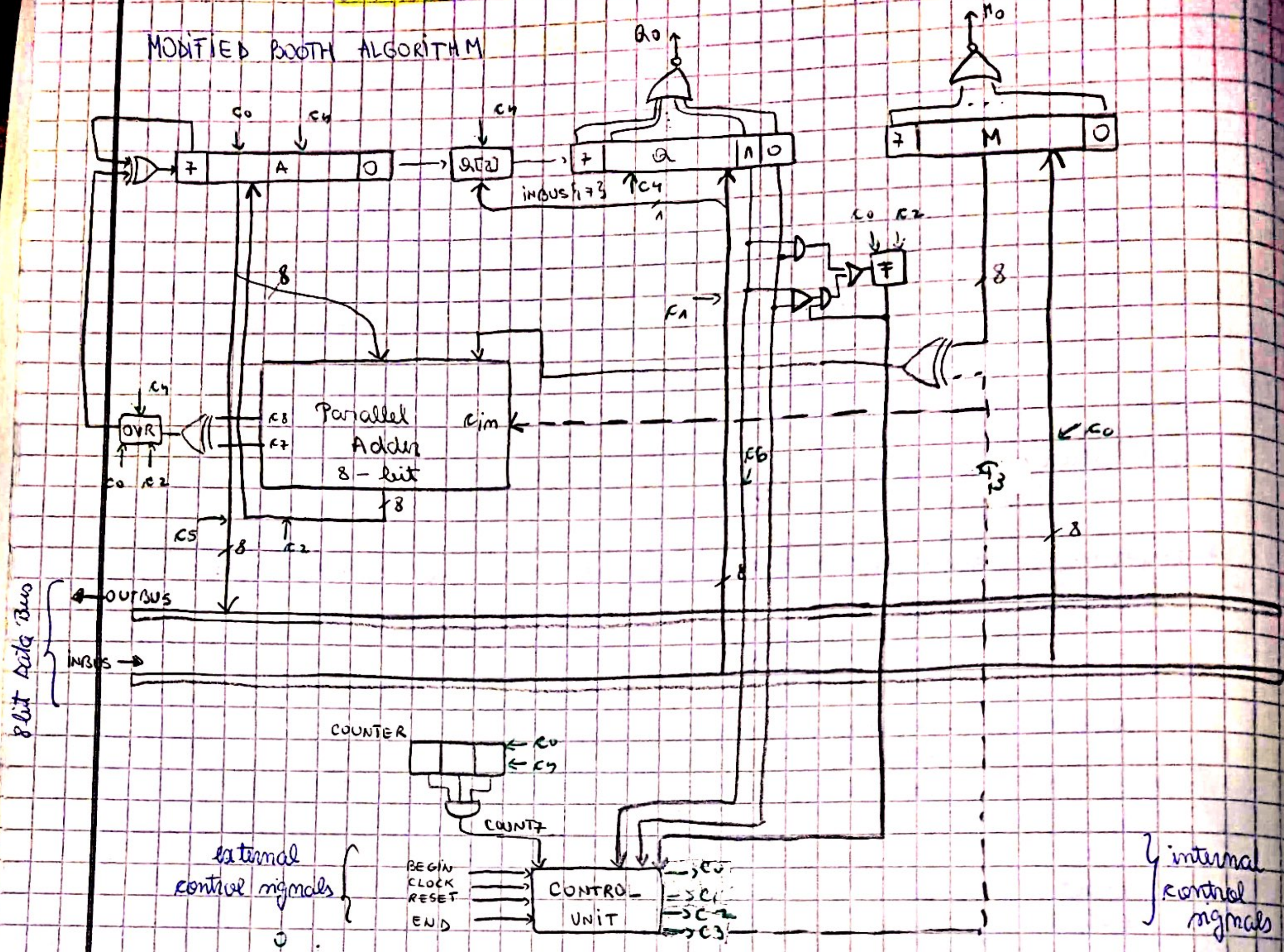
241 +
256
1024
2048
4096
+7665

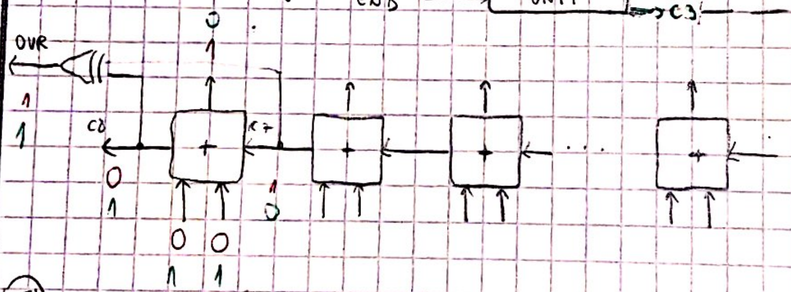
17 + 32 = 49
64
128
241

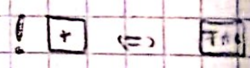
05.03.2019

CURS S3

MODIFIED BOOTH ALGORITHM



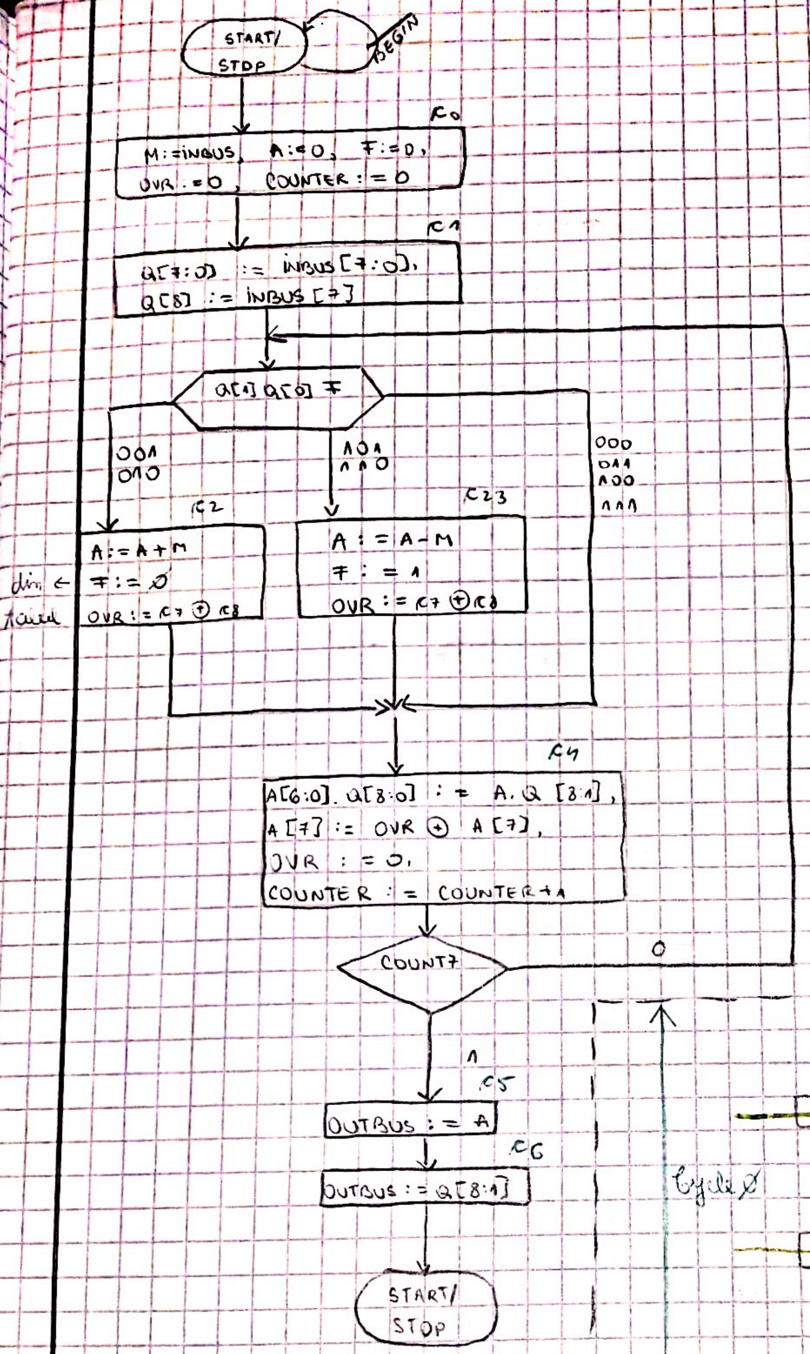




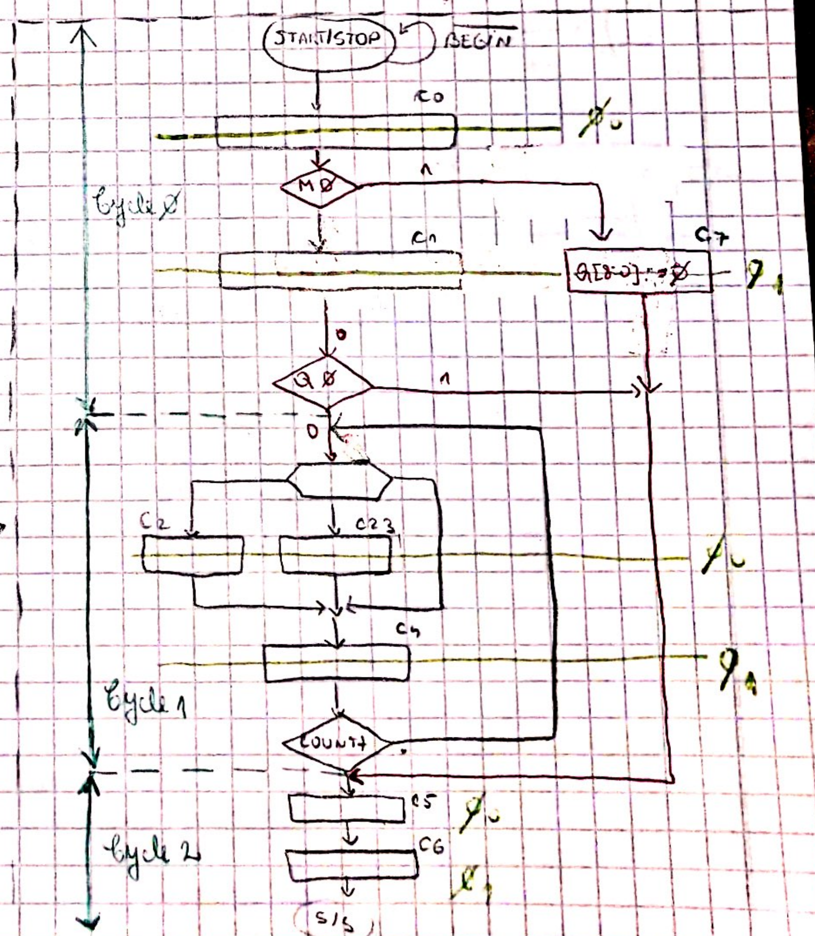
F \ Q_1Q_0	00	01	11	10
0			1	
1		1	1	1

$$F' = Q_1Q_0 + F(Q_0 + Q_1)$$

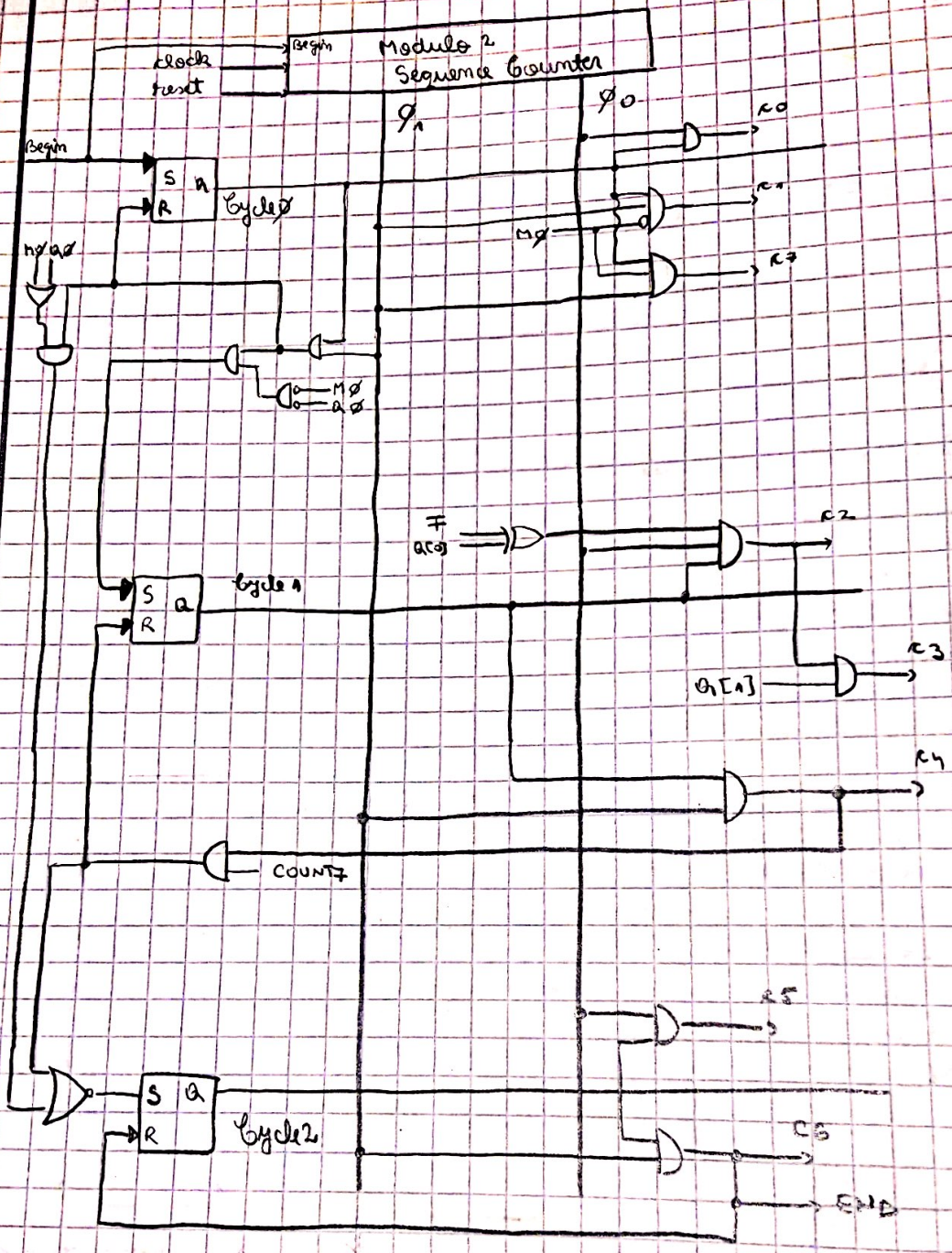
Q[1]	Q[0]	F	F'	OP
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	$\bar{1}$
1	1	0	1	$\bar{1}$
1	1	1	1	0



varianta completă



! ∅ = fază


clock
reset
Begin
Module 2
Sequence Counter
Begin
S
R
Q
Cycle0
M0
Q0
M0
Q0
C0
C1
C7
F
ACC
C2
S
R
Q
Cycle1
Q1[1]
C3
C4
COUNT7
C5
S
R
Q
Cycle2
C6
END

1.2 Speeding up: higher radix

(accelerarea înmulțirii)

- radix-4

(fac înmulțirea cu Alg. lui Booth)

b_{i+1}	b_i	b_{i-1}	OP
0	0	0	0
0	0	1	+A
0	1	0	+A
0	1	1	+2A
1	0	0	-2A
1	0	1	-A
1	1	0	-A
1	1	1	0

$A \times 0 \times 2^i + A \times 0 \times 2^{i+1} = 0 \times A \times 2^i$

$+ \quad A \times 2^i + 0 \times A \times 2^{i+1} = +A \times 2^i$

$- \quad A \times 2^i + A \times 2^{i+1} = A \times 2^i$

$0 + A \times 2^{i+1} = +2A \times 2^i$

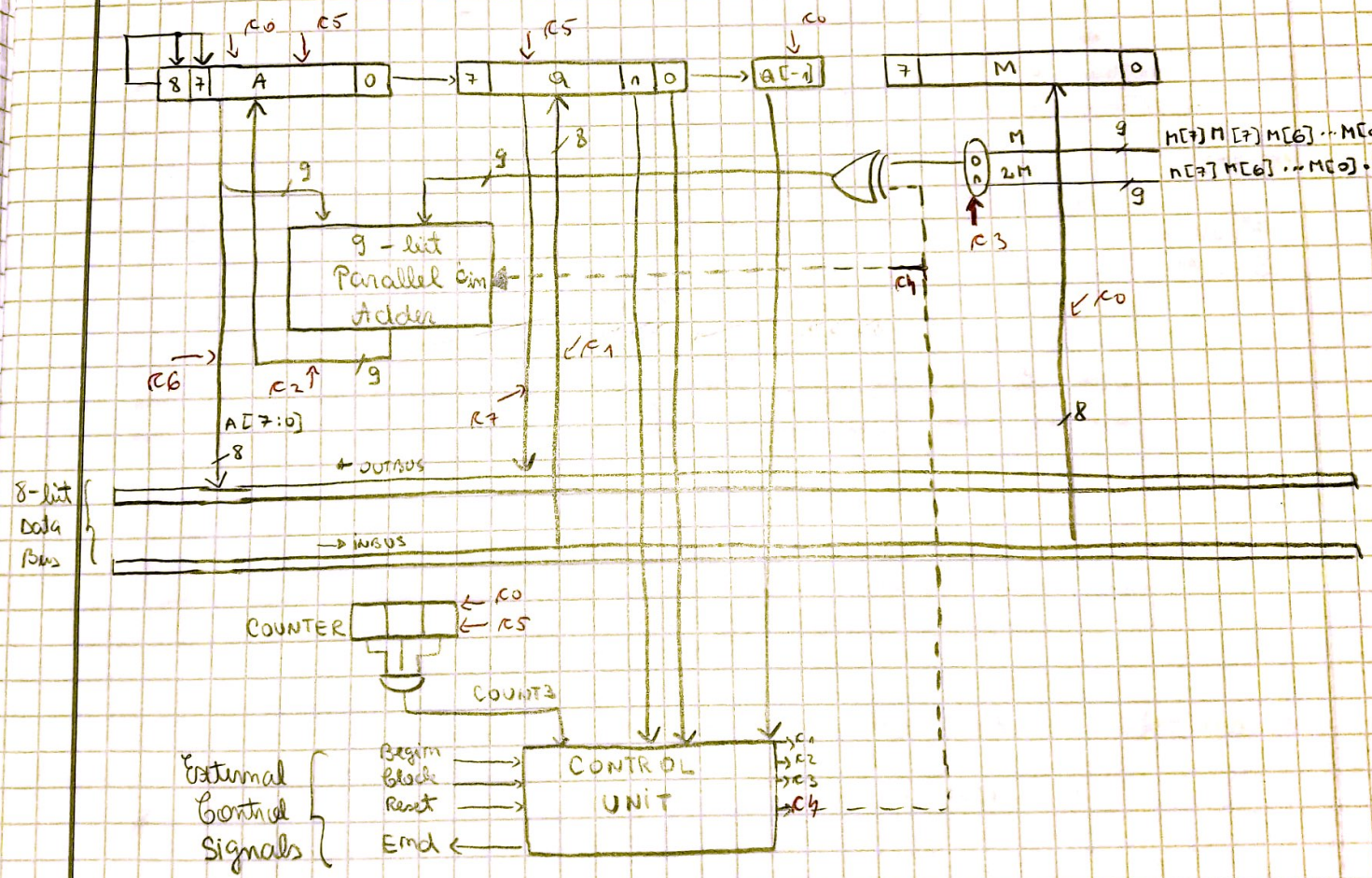
$0 - A \times 2^{i+1} = -2A \times 2^i$

$+A \times 2^i - A \times 2^{i+1} = -A \times 2^i$

$-A \times 2^i + 0$

! ponderea la pasul i este 2^i

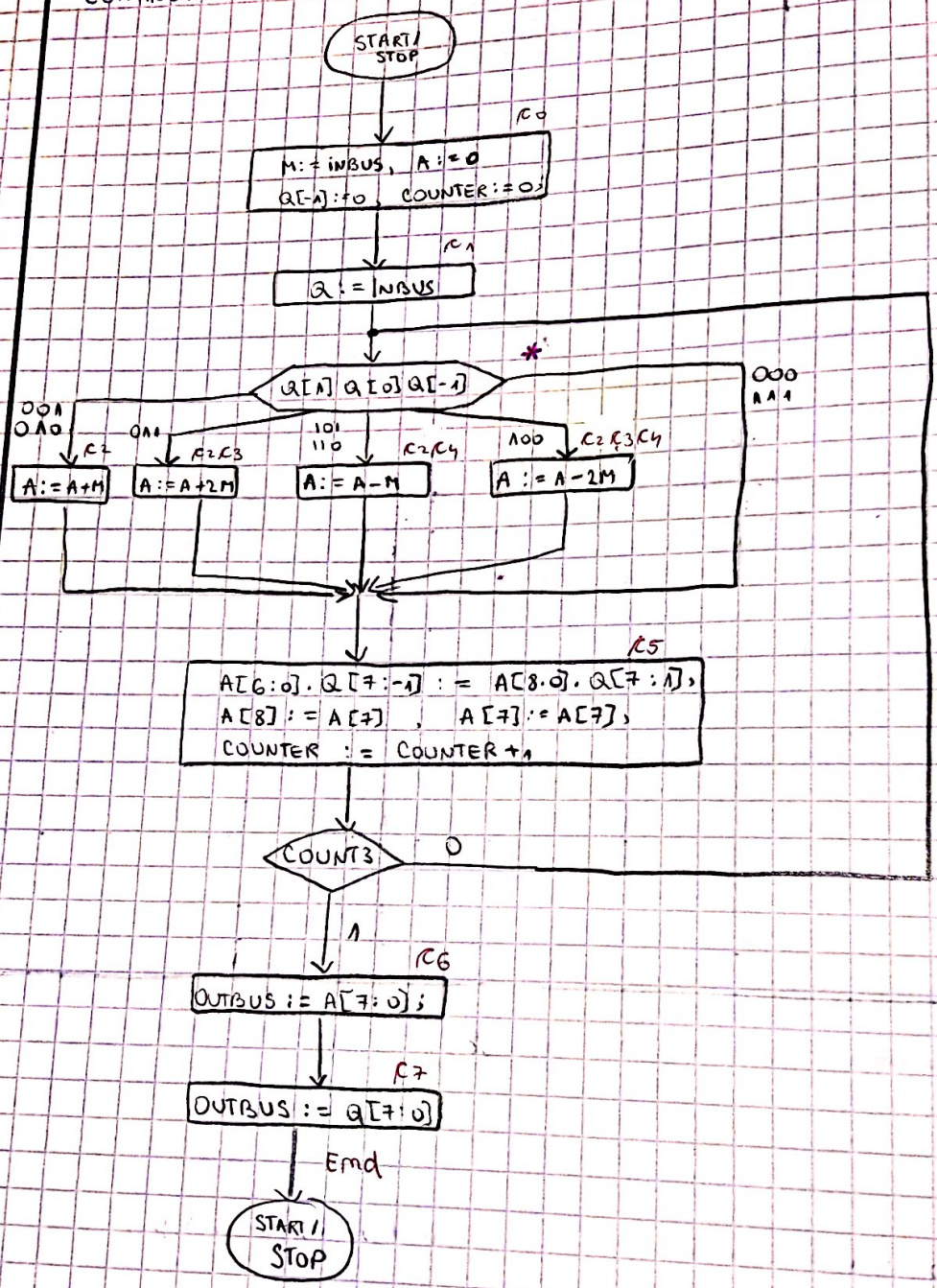
$\Rightarrow 2^i$ = weight



12.03.2018

CURS 5+4

CONTINUARE RADIX-4



← il faut la *

Q[1]	Q[0]	Q[-1]	OP
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

] +] −

EXEMPLU : X = −107 , Y = −92 ⇒ P = +9844

X = (1110 1011)SM = (1001 0101)C2

Y = (1101 1100)SM = (1010 0100)C2

COUNT	A	Q	Q[-1]	M
00	0000 0000	1010	0	1001 0101
	0000 0000	0101	0	
			+M	
01	1100 1010			
	1110 0101			
	1110 1011	0100	0	
			−2M	
10	0111 0101			
	0011 1010			
	0001 1101	1101	0	
11	0000 1110			
	0100 0111			
	0010 0011	1010	1	

↳ = +9844 ✓

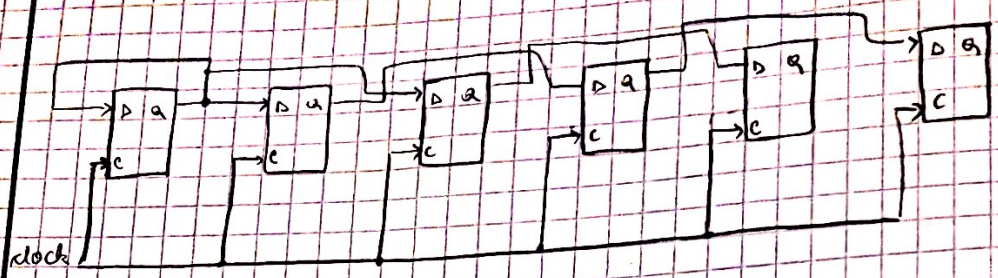
M (neg) = 1100 0101

−M = 0011 1011

2M = 1001 0100

−2M = 0110 1100

! shiftez cu 2 poziții



1.3 RADIX-8 SPEED UP

pondere 2^{i+2} pondere 2^{i+1} pondere 2^{i}

Q[2]	Q[1]	Q[0]	Q[-1]	OP
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	2
0	1	0	1	3
0	1	1	0	3
0	1	1	1	4
1	0	0	0	$\overline{4}$
1	0	0	1	$\overline{3}$
1	0	1	0	$\overline{3}$
1	0	1	1	$\overline{2}$
1	1	0	0	$\overline{2}$
1	1	0	1	$\overline{1}$
1	1	1	0	$\overline{1}$
1	1	1	1	0

$+ M * 2^i$

$- M * 2^i + M * 2^{i+1}$

$+ M * 2^{i+1} = 2M * 2^i$

$- M * 2^{i+1} + M * 2^{i+2} = 2M * 2^i$

$+ M * 2^i - M * 2^{i+1} + M * 2^{i+2} = (M - 2M + 4M) * 2^i = 3M * 2^i$

$- M * 2^i + M * 2^{i+2} = 3M * 2^i$

$M * 2^{i+2} = 4M * 2^i$

COUNT	A	Q[7:0]	Q	Q[-1]	M
00	00 0 0 0 0 0 0 0 0 +	1	1 0 1 0 0 0 1 0 0	0	1 0 0 1 0 1 0 1
	0 1 1 0 1 0 1 1 0 0			-4M	
	0 1 1 1 0 1 0 1 1 1				
	0 0 0 0 0 1 1 0 1 0 1	1	0 0 1 1 0 1 0 0 0	1	
01	0 1 0 1 0 0 0 0 0 1			-3M	
	0 1 0 1 1 1 0 1 1 1 0				
	0 0 0 0 1 0 1 1 1 0	1	1 0 1 0 0 0 1 0 1 0	1	
10	0 0 0 1 1 0 1 0 1 1 1			-2M	
	0 0 1 0 0 1 1 0 0 1				
	0 0 0 0 0 1 0 0 1 1	0	0 1 1 1 1 0 1 0 0 0	1	

pentru ex. am:

M =	11	1001	0101
-M =	00	0110	1011
2M =	11	0010	1010
-2M =	00	1101	0110
3M =	10	1011	1111
-3M =	01	0100	0001
4M =	10	0101	0100
-4M =	01	1011	1100

1.4 DIVISION ALGORITHMS

```
5771 | 135
540  | 42
=371
 270
 101
```

DIVIDENT = DIVISOR * QUOTIENT + REMAINDER

1.4.1 Restoring Division

COUNT	A	Q	M

runs 55 →

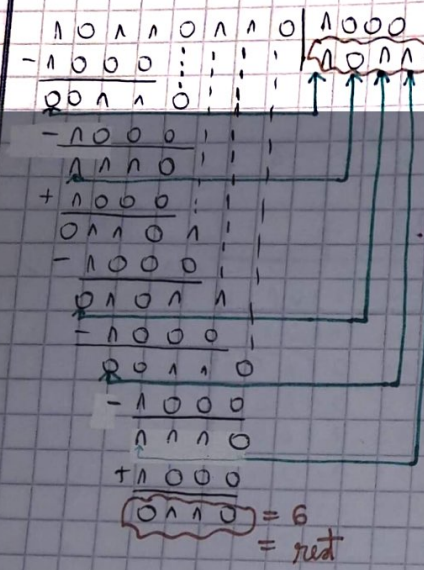
19.03 2019

CURS 55

1.4.1 Restoring Division

5771 | 135
1101 | 42

Ex. simplu:



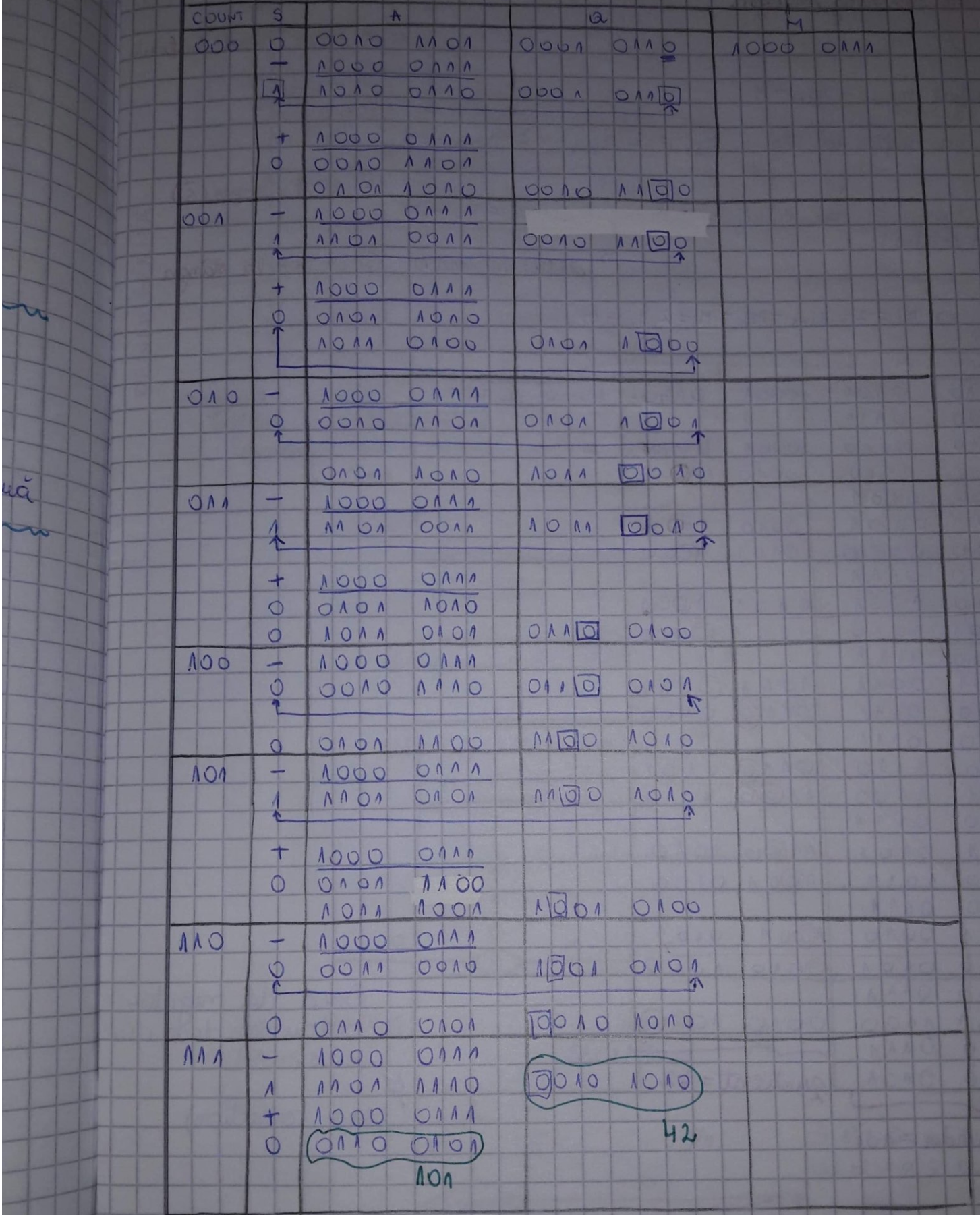
Dacă la scădere încep cu 0 → rez: 1
1 → rez: 0

→ la 0 restaurez la 1, deci restaurarea continuă

• Despre dimensiune: $(2^m-1)(2^m-1) = 2^{2m} - 2^{m+1} + 1 < 2^{2m}$

$2^{m+1} > 1 \quad \forall m \geq 1$

divizor → n biți
quotient → n biți
remainder → n biți
dividend → (2n-1) biți


COUNT
S
A
Q
M
42
101

1.4.2 Non-Restoring Division

Restul partial $r_{i-1} = r_{i-1} - M$

La fiecare pas shiftez la stg: $r_i = 2 r_{i-1} = 2(r_{i-1} - M)$

dacă $q_i = 0$ (cifra câtului) $\Rightarrow$ nu intra $\Rightarrow$ $r_i = 2 r_{i-1} - M = r_i + M$
↳ shiftez și scad M

$q_i = 1 \Rightarrow r_i = 2(r_{i-1} - M) - M = \overline{r_i} - M$

① Adun / Scad în fct de S:
$S = 1 \Rightarrow +$
$S = 0 \Rightarrow -$

② $Q[0] = \text{invers}(\bar{S})$

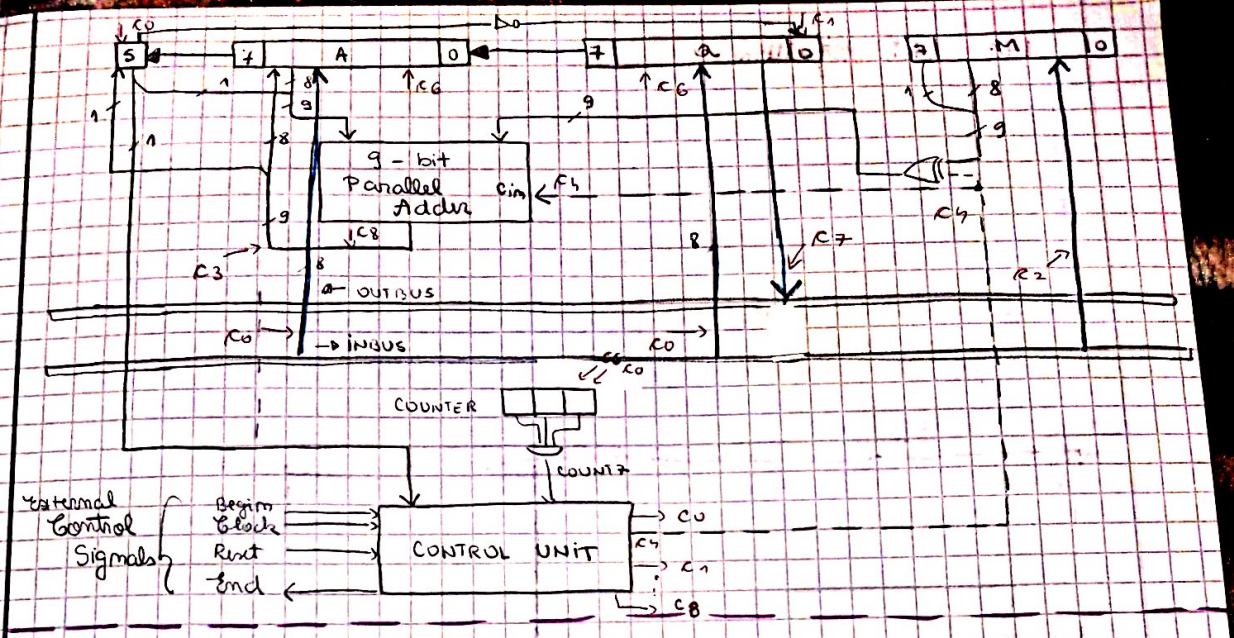
③ Shiftez la stânga

Cnt	S	A	Q	M
000	0	0010 1101	0001 0110	1000 0111
	−	1000 0111		
	1	1010 0110	0001 0110	
	1	0100 1100	0010 1100	
001	+	1000 0111		
	1	1100 0011	0010 1100	
	1	1011 0001	0101 1000	
010	+	1000 0111		
	0	0011 1000	0101 1000	
	0	0001 1100	1011 0000	
011	−	1000 0111		
	1	1101 0011	1011 0000	
	1	1010 0111	0110 0000	
100	+	1000 0111		
	0	0010 1110	0110 0000	
	0	0101 1100	1100 0000	
101	−	1000 0111		
	1	1101 0011	1100 0000	
	1	1010 0111	1000 0001	
110	+	1000 0111		
	0	0010 1110	1000 0001	
	0	0101 1100	0000 0010	
111	−	1000 0111		
	1	1101 0011	0000 0010	
cor	+	1000 0111		
	0	0101 1010	quotient = 42	

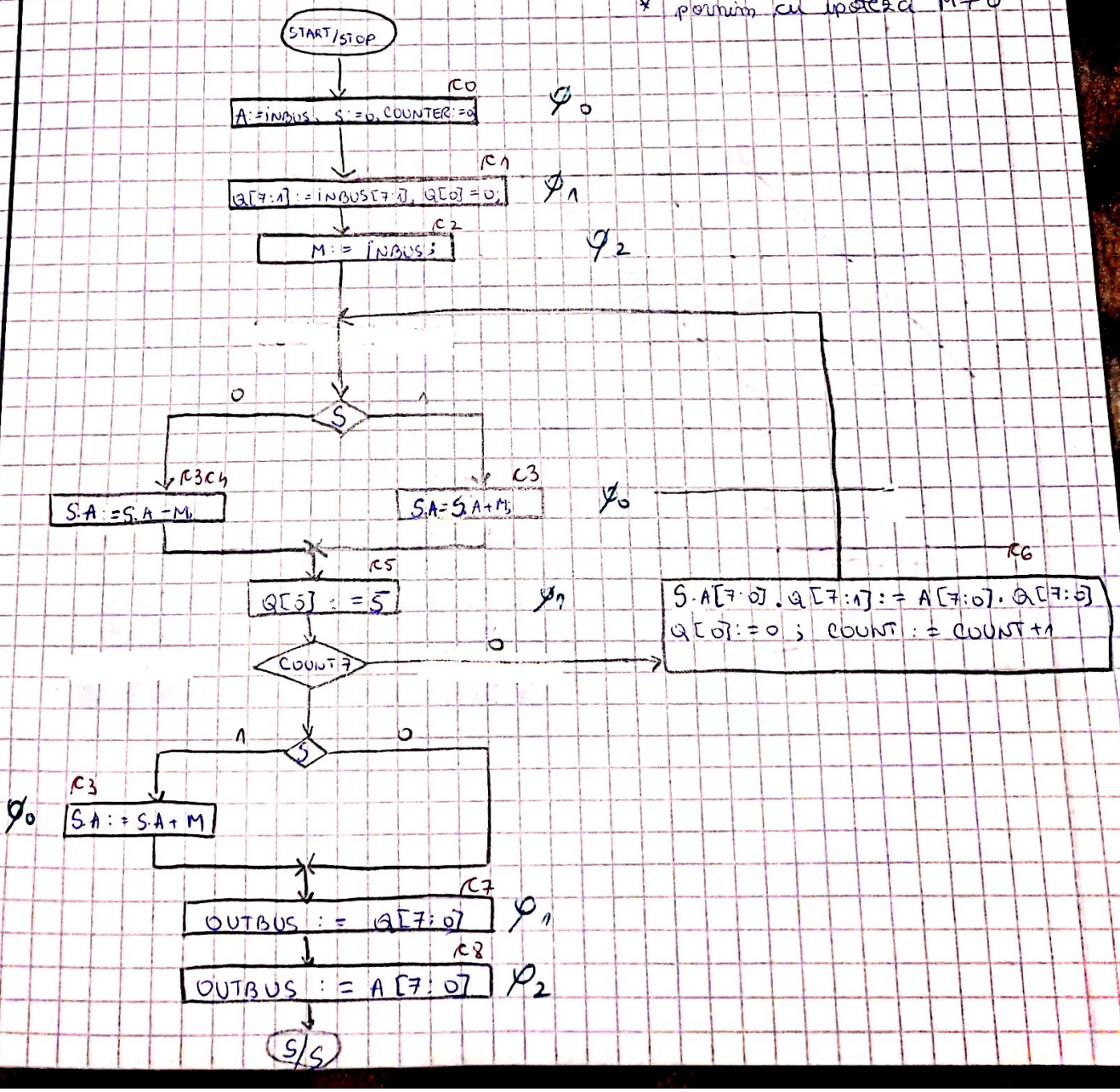
remainder = 1011

Dacă dă negativ il restaurăm doar aici
↳ (cor = correction)

ORDINOGRAMA



* pornim cu ipoteza M ≠ 0



1.4.3 SRT Algorithm

0 → {2̄, 1̄, 0, 1, 2} → radix 4

1

2 {8̄, 7̄, 6̄, 5̄, 4̄, 3̄, 2̄, 1̄, 0, 1, 2, 3, 4, 5, 6, 7, 8} → radix 16

1/2

{4̄, 3̄, 2̄, 1̄, 0, 1, 2, 3, 4} → radix 8

{1̄, 0, 1} → radix 2

↳ $10\bar{1}0\bar{1} = -1\cdot 2^0 + 0\cdot 2^1 - 1\cdot 2^2 - 1\cdot 2^3 + 0\cdot 2^3 + 1\cdot 2^4$

CODIFICARE = 11

1 → 1 0

0 → 0 0

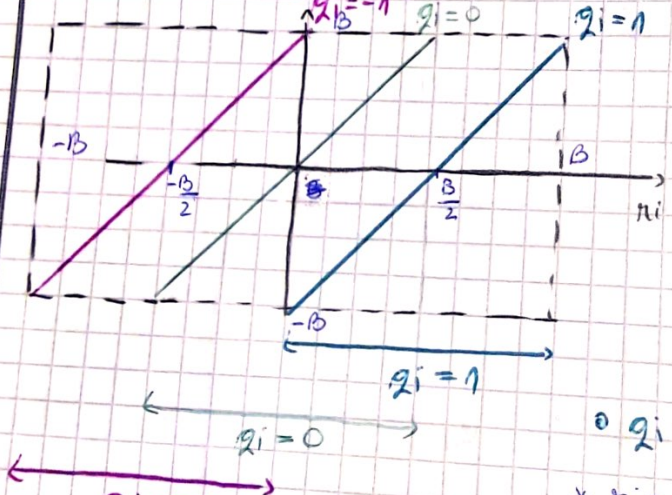
1̄ → 0 1

$$\begin{array}{r} 1\,00000 - \\ 00101 \\ \hline 01\,01\,11 \\ \underbrace{\qquad\quad}_{11} \end{array}$$

$r_{i+1} \leftarrow 2r_i - q_i B$, $q_i \in \{1̄, 0, 1\}$

$|r_i| < B$

Aici: A → P, Q → A, M → B^{i+1}



• $q_i = -1$

* $r_{i+1} = B \Rightarrow r_i = 0$

* $r_{i+1} = 0 \Rightarrow r_i = -B/2$

* $r_{i+1} = -B \Rightarrow r_i = -B$

• $q_i = 1$

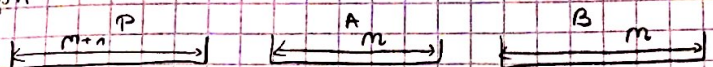
* $r_i = B \Rightarrow r_{i+1} = 2B - B = B \Rightarrow q_i = 1$

* $r_{i+1} = 2r_i - B = 0 \Rightarrow r_i = \frac{B}{2}$

* $r_{i+1} = -B = 2r_i - B \Rightarrow r_i = -B$

SRT RADIX 2

Step 1



if B has K leading 0's, Shift P.A.B with K positions left.

Step 2

for i = 0 to m-1

if top 3 bits of register P are equal then { $q_i = 0$; shift P.A. left

if top 3 bits of register P are not equal & P > 0 then { $q_i = 1$, subtract B from P, shift P.A. left

if top 3 bits of register P are not equal & P < 0 then $q_i = \bar{1}$, { add P to B, shift P.A. left

Step 3

if P < 0 add { B to P & { subtract 1 from A

Step 4 shift P right K positions

20.03.

LABORATOR S5
(implem. în Verilog)

26.03.2019

CURS S6

exemplu de SRT: 229 : 12

COUNT	P		A		B	
					0000	1100
000	00000	0000	1110	0101	1100	0000
	000000	1110	0101	0000		
	$q_0=0$					
	00001	1100	1010	000(0)		
001	$q_1=0$					
	00011	1001	0100	00(0)0		
010	$q_2=0$					
	00111	0010	1000	0(0)00		
011	$q_3=1$					
	01110	0101	0000	(0)001		
	−1100	0000				
	00010	0101				
100	$q_4=0$					
	00100	1010	000(0)	0010		
101	$q_5=1$					
	01001	0100	00(0)0	0101		
	−1100	0000				
	11101	0100				
110	$q_6=0$					
	11010	1000	0(0)00	1010		
111	$q_7=\bar{1}$					
	10101	0000	(0)001	010$\bar{1}$		
	+1100	0000				
	*00001	0000				
shift	00000	0001	0001	0011		

← am shiftat la stg cu 4 poz. ca să scap de 0.

remainder = 1

quotient = 19

37
64
128

229 | 12
12 | 19
109
108
= = 1

când restul e negativ (primul bit e 1) fac corecție, adică adun înB.

aici nu e cazul, pt. că nr. e pozitiv* și doar shiftez înapoi cu 4 poz. doar reg. P

* $0 \rightarrow \begin{bmatrix}0\\0\end{bmatrix}$, $1 \rightarrow \begin{bmatrix}1\\0\end{bmatrix}$, $\bar{1} \rightarrow \begin{bmatrix}0\\1\end{bmatrix}$

$r_{i+1} = 2r_i - q_i B$

$q_i = 1 \Rightarrow r_{i+1} = 2r_i - B$

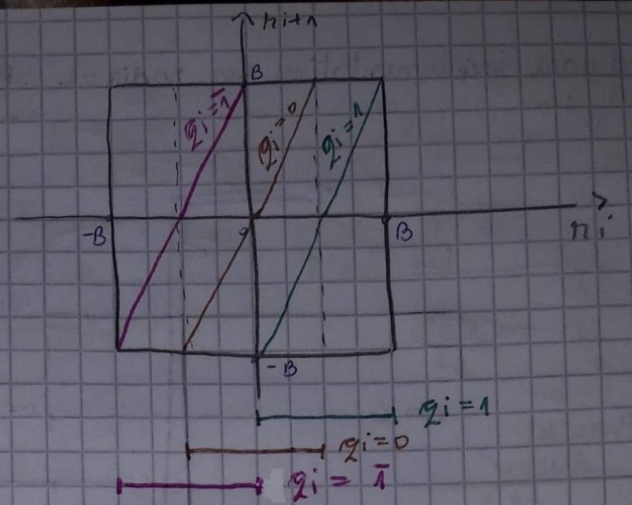
$q_i = 0 \Rightarrow r_{i+1} = 2r_i$

$q_i = \bar{1} \Rightarrow r_{i+1} = 2r_i + B$

$q_i \in \{0, 1, \bar{1}\}$

$|r_{i+1}| < B$

$|r_i| < B$

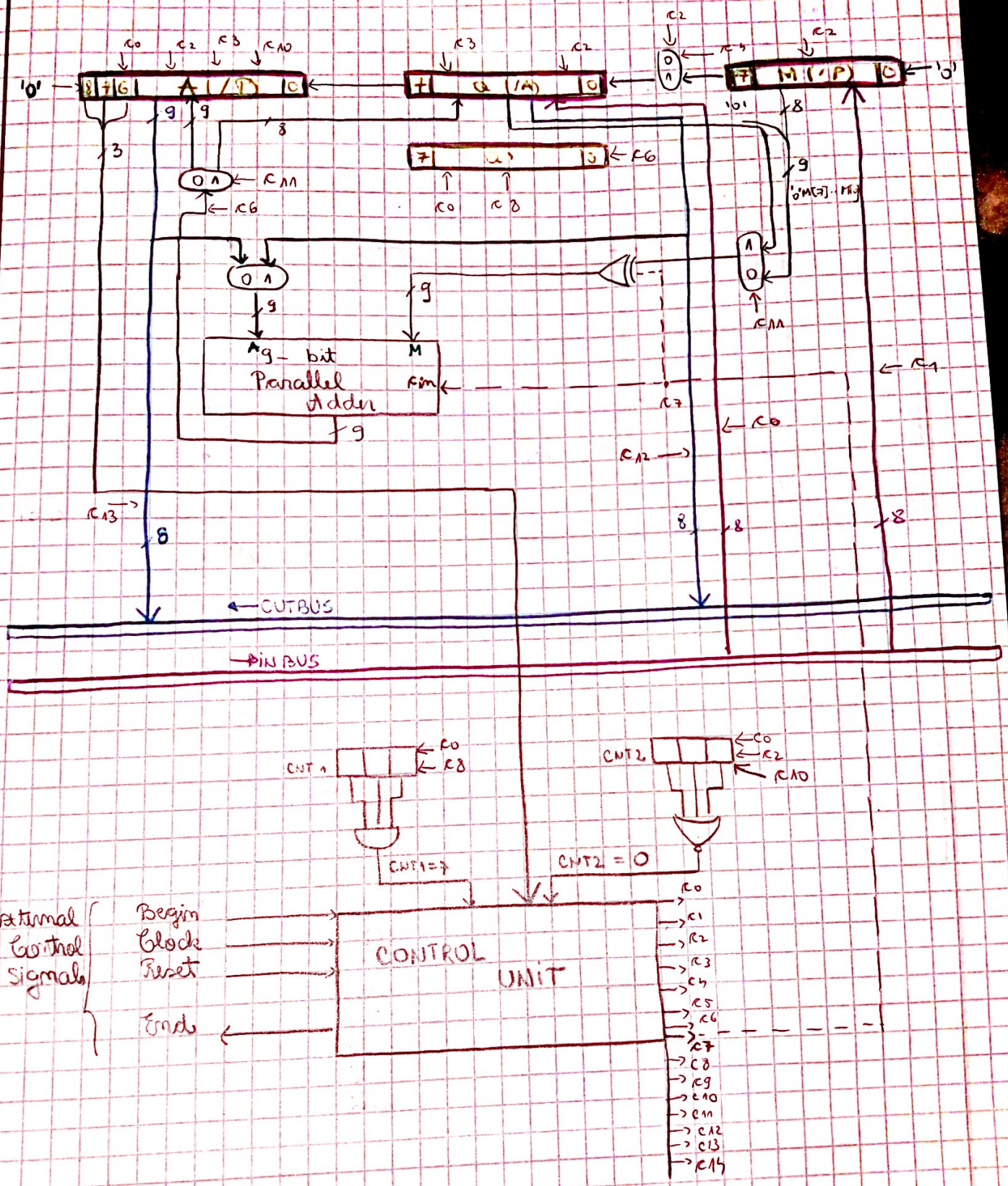


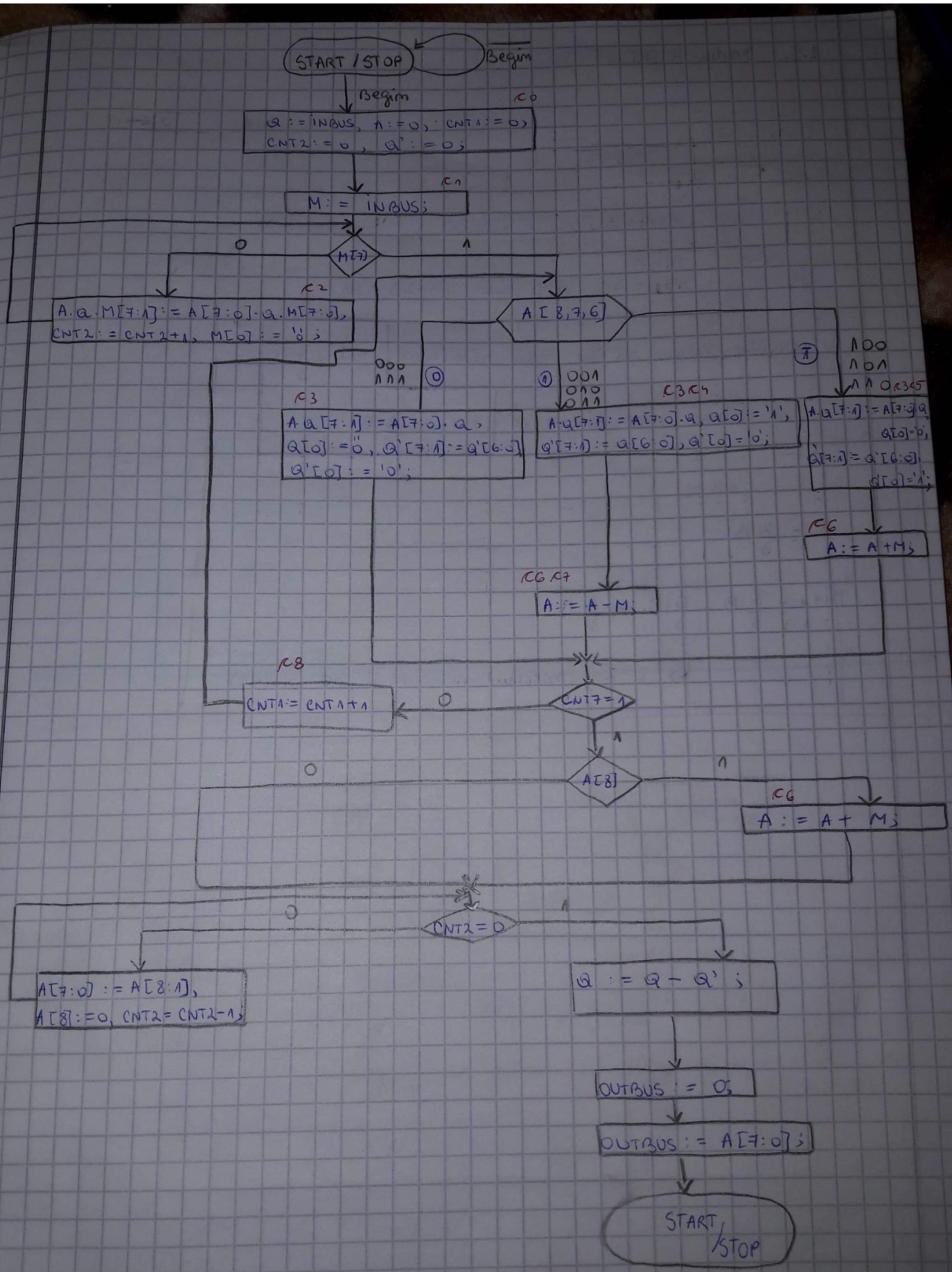
ex. 2: 233 | 9
18 | 25
=53
45
=8

COUNT	P		A		B	
000	00000	0000	1110	1001	0000	1001
	00000	1110	1001	0000	1001	0000
	$q_0 = 0$					
	00001	1101	0010	0000		
001	$q_1 = 0$					
	00011	1010	0100	0000		
010	$q_2 = 0$					
	00111	0100	1000	0000		
011	$q_3 = 1$					
	01110	1001	0000	0001		
	-1001	0000				
	00101	1001				
100	$q_4 = 1$					
	01011	0010	0000	0011		
	-1001	0000				
	00010	0010				
101	$q_5 = 0$					
	00100	0100	0000	0110		
110	$q_6 = 1$					
	01000	1000	0000	1101		
	-1001	0000				
	11111	1000				
111	$q_7 = 0$					
	11111	0000	0001	1010		
				-1		
corr	+1001	0000				
	01000	0000	0001	1001		

scad o la A pt. că am adunat în P

1.4.4 Hardware implementation for radix-2 SRT




START / STOP
Begin
Begin
C0
Q := INBUS, A := 0, CNT1 := 0,
CNT2 := 0, Q' := 0;
C1
M := INBUS;
M[7]
0
1
C2
A.Q.M[7:1] := A[7:0].Q.M[7:0],
CNT2 := CNT2+1, M[0] := '0';
A[8,7,6]
0
1
2
3
C3
A.Q[7:1] := A[7:0].Q,
Q[0] := '0', Q'[7:1] := Q'[6:0]
Q'[0] := '0';
C3C4
A.Q[7:1] := A[7:0].Q, Q[0] := '1',
Q'[7:1] := Q[6:0], Q'[0] := '0';
C3C5
A.Q[7:1] := A[7:0].Q
Q[0] := '0',
Q'[7:1] := Q'[6:0]
Q'[0] := '1';
C6
A := A + M;
C6C7
A := A - M;
C8
CNT1 := CNT1 + 1
CNT7 = 1
0
1
A[8]
0
1
C6
A := A + M;
CNT2 = 0
0
1
A[7:0] := A[8:1],
A[8] := 0, CNT2 := CNT2 - 1;
Q := Q - Q';
OUTBUS := 0;
OUTBUS := A[7:0];
START / STOP

1.5. radix-4 SRT

$\{\bar{2}, \bar{1}, 0, 1, 2\}$

$r_{i+1} = 4r_i - q_i B$

$|r_i| < B \Rightarrow |r_{i+1}| < B$ nu e suficient:

de ex: $r_i = B$

$r_{i+1} = 4B - q_i B \overset{\text{maxim}}{=\!=} 4B - 2B = 2B$

$\Rightarrow$ condiția trebuie să fie mai restrictivă: $|r_i| < \frac{2}{3} B$

și atunci mai asigur că $|r_{i+1}| < \frac{2}{3} B$

$r_i = \frac{2}{3} B$

$r_{i+1} = \frac{8}{3} B - \frac{6}{3} B = \frac{2}{3} B$

↳ (=2= maximul pe care îl pot scădea)

02. 04. 2019

CURS 57

1.5 Radix-4 SRT

$q_i \in \{2, \bar{1}, 0, 1, 2\}$

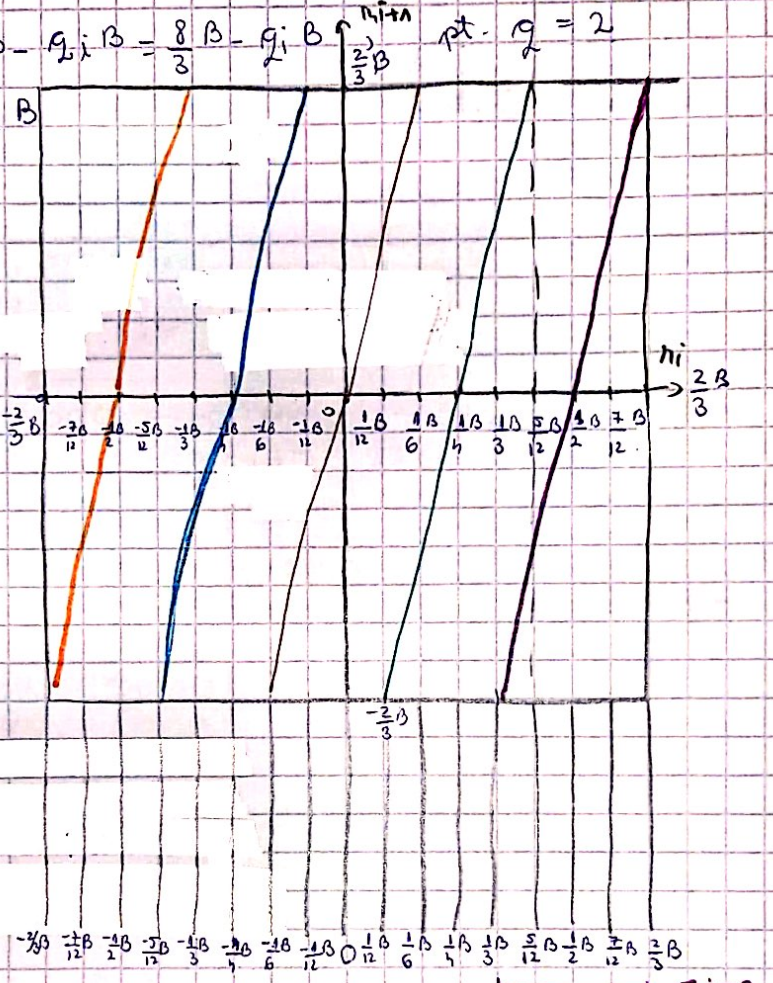
$r_{i+1} = 4 \cdot r_i - q_i B$ $\not\Rightarrow$ $|r_{i+1}| < B$

$r_{i+1} = 4B - q_i B = 2B$ pt. $q_i = 2$

$\Rightarrow$ este suficientă condiția $|r_i| < \frac{2}{3} B$ pt. $\Rightarrow r_{i+1} < B$

$r_{i+1} = 4 \cdot \frac{2}{3} B - q_i B = \frac{8}{3} B - q_i B$ pt. $q = 2$

$\Rightarrow r_{i+1} = \frac{2}{3} B$



$q_i = 2 \Rightarrow r_{i+1} = 4r_i - B$

$q_i = 1 \Rightarrow r_{i+1} = 4r_i - B$

$q_i = 0 \Rightarrow r_{i+1} = 4r_i$

$q_i = \bar{1} \Rightarrow r_{i+1} = 4r_i + B$

$q_i = \bar{2} \Rightarrow r_{i+1} = 4r_i + 2B$

ex: 239 : 12 = 19 rest 11

COUNT	P	A	B
00	00000	0000	1100
	00000	1101	0000
	q0=0		↑ b dim tabel!
	00011	1100	
01	q1=1		
	01110	0101	
	- 1100	0000	
	00010	0101	
10	q2=1		
	01001	0100	
	- 1100	0000	
	11101	0100	
11	q3=1		
	10101	0000	
	+ 1100	0000	
	00001	0000	
SHIFT	00000	0001	

Fundeea e m. negativ: 111010 → 111001 → 000110 => (111010)c2 = (-6)

$0 \to 0$ $\quad 1 \to 1$ $\quad \bar{1} \to 0$

$0 \to 00$ $\quad 1 \to 01$ $\quad 2 \to 10$

$\bar{1} \to$ $\quad \bar{2} \to$

! dacă am de ales între $q_i = 0$ sau $q_i = 1$, e mai convenabil $q_i = 0$.

ex.2: $238 : 10 = 23$ rest 8

COUNT	P	A	B
00	00000 0000 00000 1110 $q_0 = 0$ 00011 1011	1110 1110 1110 0000 1000 0001	0000 1010 1010 0000 B = 10
01	01110 $q_1=1$ 1110 − 1010 0000 00100 0110	0000 0001	
10	$q_2 = 2$ 10011 1000 − 10100 0000 11111 1000	0000 10	
11	$q_3 = 0$ 11110 0000	0002 0 − (*)	
COR	+0 1010 0000 01000 0000		
SHIFT	00000 1000 remainder = 8		

$2 \times 4^1 + 1 \times 4^2 - 1 = 16 + 8 - 1 = 23$

00 01 10 00 −
00 00 00 01
00 01 01 11 = 23

! Trebuie să fac corecție, deoarece am obț. REST NEGATIV.

↳ adun pe B.

Deoarece fac corecție, scad 1 (*)

CAP. 2

PERFORMANCE IN COMPUTER SYSTEMS

2.1 Performance Equation

1. Performance is hard to assess in computer systems
2. Producers tend to present their product in the best possible light.
3. Design decisions are based on performance assessments.

$A > B$, $B > C$ $\Rightarrow$ $A > C$

	SPEED	CAPACITY
A1:	500 mile/h	10 pasageri
A2:	700 mile/h	100 pasageri
A3:	1000 mile/h	2 pasageri

Care e cel mai bun? A2 (700 · 100 e max)

↓

câți pasageri duce în unitatea de timp

↳ THROUGHPUT / BANDWIDTH

User Response Time = timpul de procesor

= CPU time

= no. of clock cycles per program * clock cycle time

= Instruction Count * Clock cycles per instruction * Clock Cycle Time

nr. de instrucțiuni

= IC * CPI * Clock Cycle Time

$$CPI = \sum_{i=1}^{n} \frac{Ici}{IC} * CPI_i$$

CPI_i = CPI of class i

Ici = no. of instructions in class i

$$\text{Performance of } X = \frac{1}{\text{Execution Time of } X}$$

X: Computer A is n times faster the computer B

$$\Rightarrow n = \frac{\text{Performance A}}{\text{Performance B}} = \frac{\text{Execution Time A}}{\text{Execution Time B}}$$

EXAMPLE 1:

CPU time A = IC * 2.0 * 250 ps = IC * 500 ps

CPU time B = IC * 1.2 * 500 ps = IC * 600 ps

=> comp. A is $\frac{IC * 600ps}{IC * 500ps}$ = 1.2 times faster than comp. B.

exercise:

- Suppose we have 2 implementations for the same instruction architecture
- Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program;
- Computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program;

* Which computer is faster for this program and by how much?

09.04.2019

CURS 58

Example 2 — Comparing Code Segments

$$\text{CPU time}_1 = IC_1 * CPI_1 * \text{Clock Cycle Time}$$

$$= \underbrace{5}_{IC_1} \times \frac{2*1 + 1*2 + 2*3}{\underbrace{5}_{CPI_1}} \cdot \text{Clock}_2 \text{ Cycle Time} =$$

$$= 10 \text{ Clock Cycle} \cdot \text{Clock Cycle Time}$$

$$\text{CPU time}_2 = IC_2 * CPI_2 * \text{Clock Cycle Time}$$

$$= \underbrace{6}_{IC_2} \cdot \frac{4*1 + 1*2 + 1*3}{6} \cdot CCT = 9\ CC \cdot CCT$$

$$\frac{\text{Performance}_2}{\text{Performance}_1} = \frac{\text{CPU time}_1}{\text{CPU time}_2} = \frac{10}{9} = 1.1$$

Enunțul:

- A computer designer is trying to decide between 2 code segments for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

- For a particular high-level language statement, the compile writer is considering 2 code-sequences that require the following ICs:

	INSTRUCTION COUNTS FOR EACH INSTRUCTION CLASS		
Code sequence	A	B	C
1	2	1	2
2	4	1	1

Example 3

CPU time old = IC old · CPI old · CCT old

CPU time new = IC new · CPI new · CCT old

I_c new = 0.6 · I_c old

CPI new = 1.1 · CPI old

=> CPU time new = 0.6 · (Icold · 1.1 · CPI old · CCT old) =

= CPU time old

= 15 · 0.6 · 1.1 = 9.9 sec !!! unitate de masura !!!

Example 4

CPU time 1 VS CPU time 2

CPI original = 0,25 · 4,0 + 0,75 · 1,33 ≈ 2 cc

CPI1 = CPI original − 0,02 · (20−2) = 2 − 0,02·18

= 2 − 0,36 = 1,64 cc

CPI_2 = 0,25 · 2,5 + 0,75 · 1,33 = 1,625 cc

Example 5

CPU A

CMP R1, R2 ← compară conținutul celor 2 reg (scad R1 - R2)

BEQ F00

↓

branch if equal:

F00: ADD R6, R5, R6

CPU B

BEQ R1, R2, F00

F00:

CPU time A = $IC_A \cdot CPI_A \cdot CCT_A$

$CPI_A = 0,2 \cdot 2 + 0,8 \cdot 1 = 1,2$ CC → e mai bun

⇒ (CPU time A) = $IC_A \cdot \boxed{1,2} \cdot CCT_A$

CPU time B = $\underbrace{0,8 \cdot IC_A}_{IC_B} \cdot CPI_B \cdot \underbrace{1,25 \cdot CCT_A}_{CCT_B}$

$CPI_B = 0,25 \cdot 2 + 0,75$

$= 1,25$ (c.c.) clock cycles

20% ... 80%

a ... 100% ⇒ a = 25%

(CPU time B) = $IC_A \cdot \boxed{1,25} \cdot CCT_A$

Example 6

ADD $r1, $r2, # mem_loc

LDR $r1, # mem_loc

ADD $r2, $r2, $r1

STR $r2, # mem_addr_2

CPU time A = $IC_A \cdot CPI_A \cdot CCT_A$

$CPI_A = 0,43 \cdot 1 + 0,57 \cdot 2 = \boxed{1,57\ cc}$

=> CPU time A = $IC_A \cdot 1,57 \cdot CCT_A$

CPU time B = $I_{CA}(1 - 0,43 \cdot 0,25) \cdot \dfrac{(0,75 \cdot 0,43) \cdot 1 + (0,25 \cdot 0,43) \cdot 2 + (0,25 \cdot 0,43) \cdot 2 + 0,12 \cdot 2 + 0,25 \cdot 3}{1 - 0,43 \cdot 0,25} \cdot CCT_A$

(I_{CB} = $I_{CA}(1 - 0,43 \cdot 0,25)$; CPI_B = the fraction)

$= I_{CA} \cdot 1,7 \cdot CCT_A$

=> non ottimizzata e meno preferita

Esempi:

2.2 Real-World BenchMarks

Perf A bench > Perf B bench => Perf A p > Perf B p

A. Synthetic Bench Marks ×

B. Toy Programs ×

C. Kernels ×

D. Real programs mix ×

SPEC — BenchMark suites

Standard Performance Evaluation Corporation — spec.org

$$SPEC = \sqrt[n]{\prod_{i=1}^{n} SPEC_{Ai}}$$

$SPEC_{Ai}$ = Execution time on machine A of program i from the SPEC BenchMark Suite

$$\frac{SPEC\ reference}{SPEC_A} = \frac{\sqrt[n]{\prod_{i=1}^{n} SPEC\ reference_i}}{\sqrt[n]{\prod_{i=1}^{n} SPEC_{Ai}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{SPEC\ reference_i}{SPEC_{Ai}}} = \frac{Performance\ A}{Performance\ reference}$$

= SPEC Ratio A

FA - LA - LA - LA - LA

$$\frac{\text{Performance A}}{\text{Performance B}} = \frac{\text{SPEC Ratio A}}{\text{SPEC Ratio B}} = \frac{\sqrt[n]{\prod_{i=1}^{m} \text{SPEC reference}_i}}{\sqrt[m]{\prod_{i=1}^{m} \text{SPEC A}_i}} \Big/ \frac{\sqrt[n]{\prod_{i=1}^{m} \text{SPEC reference}_i}}{\sqrt[n]{\prod_{i=1}^{m} \text{SPEC B}_i}}$$

$$= \frac{\sqrt[n]{\prod_{i=1}^{m} \text{SPEC B}_i}}{\sqrt[n]{\prod_{i=1}^{m} \text{SPEC A}_i}}$$