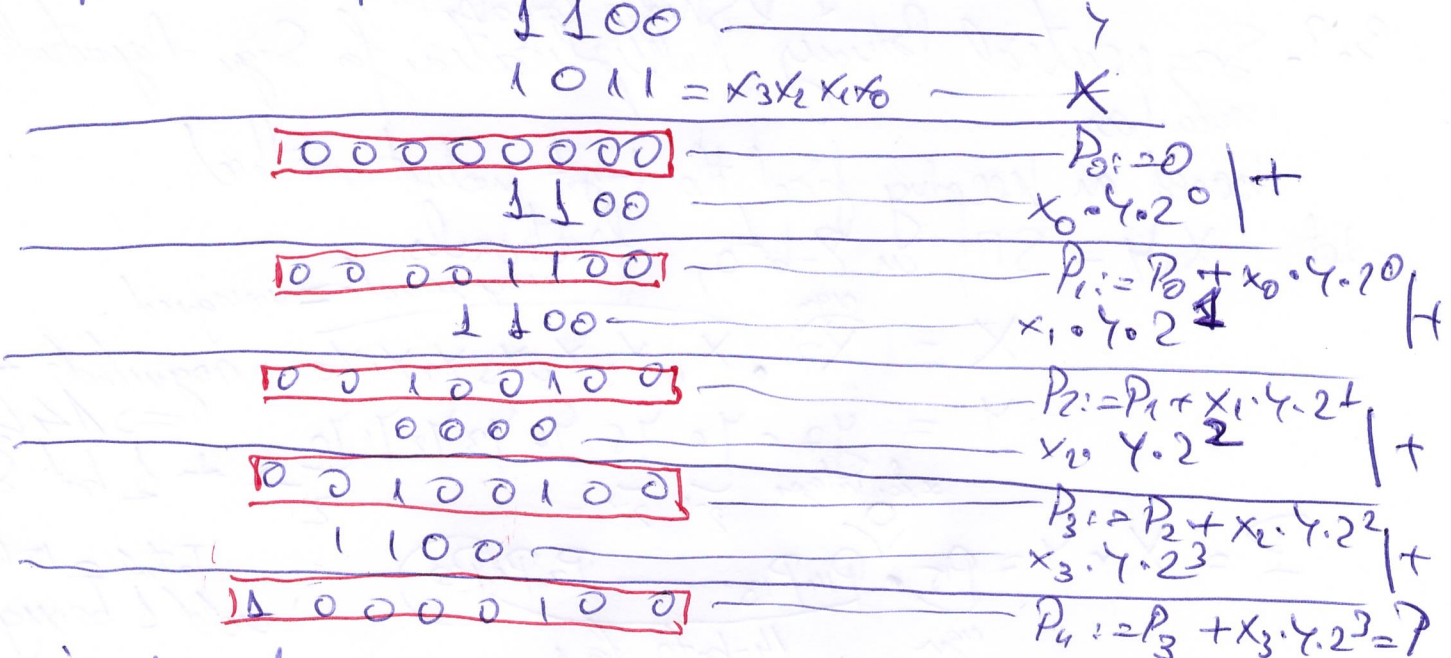
3.1. HW investment:
- 2 4-bit registers for operands
- multi-operand adder (CSA)
- multi-level gating

multiplication of 2 unsigned on n bits
⇒ Product is on 2n

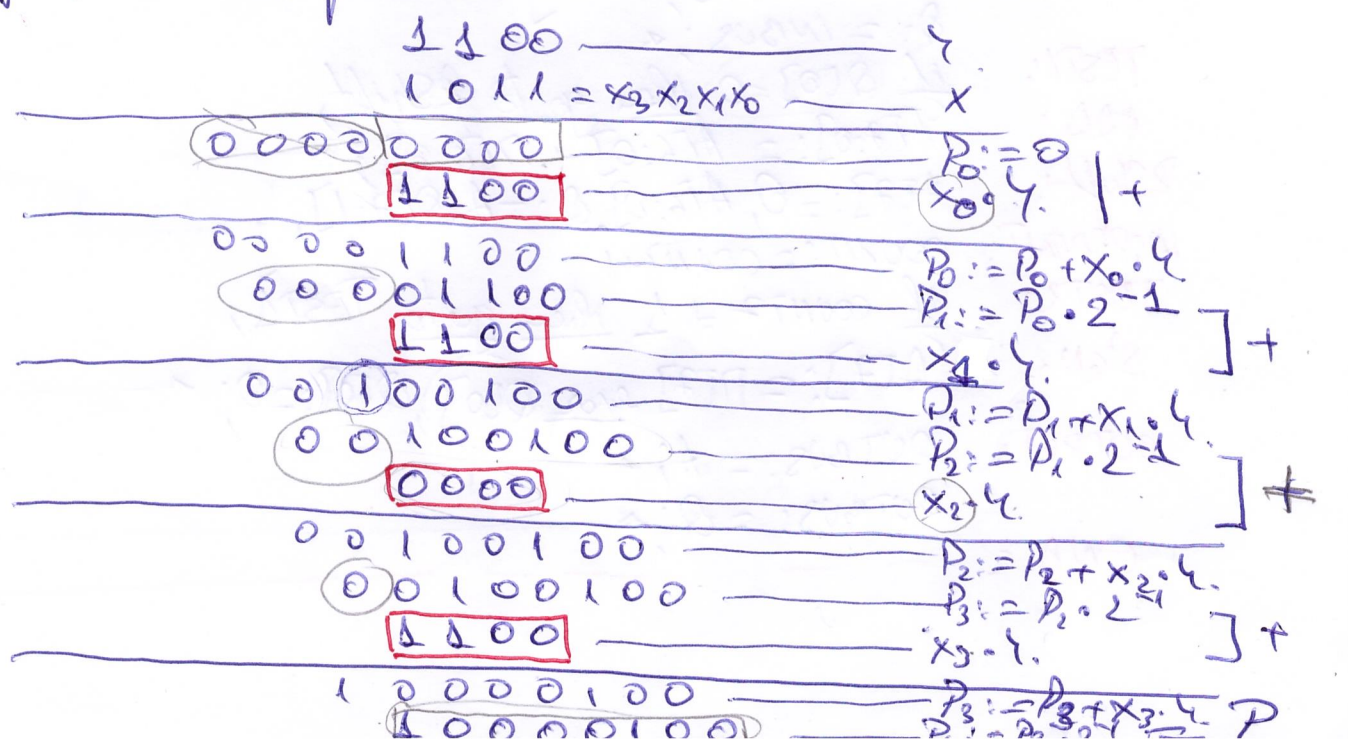
b) keep fixed the partial products.



Iteration step: $P_{i+1} := P_i + x_i \cdot Y \cdot 2^i$, $i \geq 0$, $P_0 := 0$

HW investment: → 2 4-bit registers
→ 8-bit adder
→ 8-bit register for P_i
→ LShift for $x_i \cdot Y \cdot 2^i$

c) keep fixed the 1-bit products.



Iteration step:

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

HW investment: – 2 4-bit registers
– 4-bit adder
– 8-bit register for partial prod.
+ RShift capability

3.2. Sequential Binary Multiplication for Sign Magnitude numbers.

– based on keeping fixed the 1-bit product method.

Let X, Y – SM, on 8-bits, fractionals,

magnitude = unsigned.

$X = \underbrace{x_7}_{\text{sign}} \cdot \underbrace{x_6 x_5 x_4 x_3 x_2 x_1 x_0}_{\text{magnitude}}$

$Y = y_7 \cdot y_6 y_5 y_4 y_3 y_2 y_1 y_0$

magnitude * ⇒ 14 bit + 1 bit sign

* for integer

y_6: 2^{-1}, y_5: 2^{-2}, ..., y_0: 2^{-7}

$P = X * Y = \underbrace{p_{15}}_{\text{sign}} \cdot \underbrace{p_{14} p_{13} \dots p_2 p_1}_{\text{14-bits for magnitudes multiplicate}} \; p_0$

Total = 15 bit fct + add 1 bit sign bit

$p_0 = 0$

$p_{15} = x_7 \oplus y_7$

multiplier 2 — part-prod* × Y

declare registers: 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, } {c_0}

INPUT: M := INBUS; {c_1}

Q := INBUS;

TEST1: if Q[0] = 0 then go to RShift, {c_2}

ADD: A[7:0] := A[6:0] + M[6:0];

RShift: A[7] := 0, A[6:0], Q := A, Q[7:1], } {c_3}

INCREMENT: COUNT := COUNT + 1

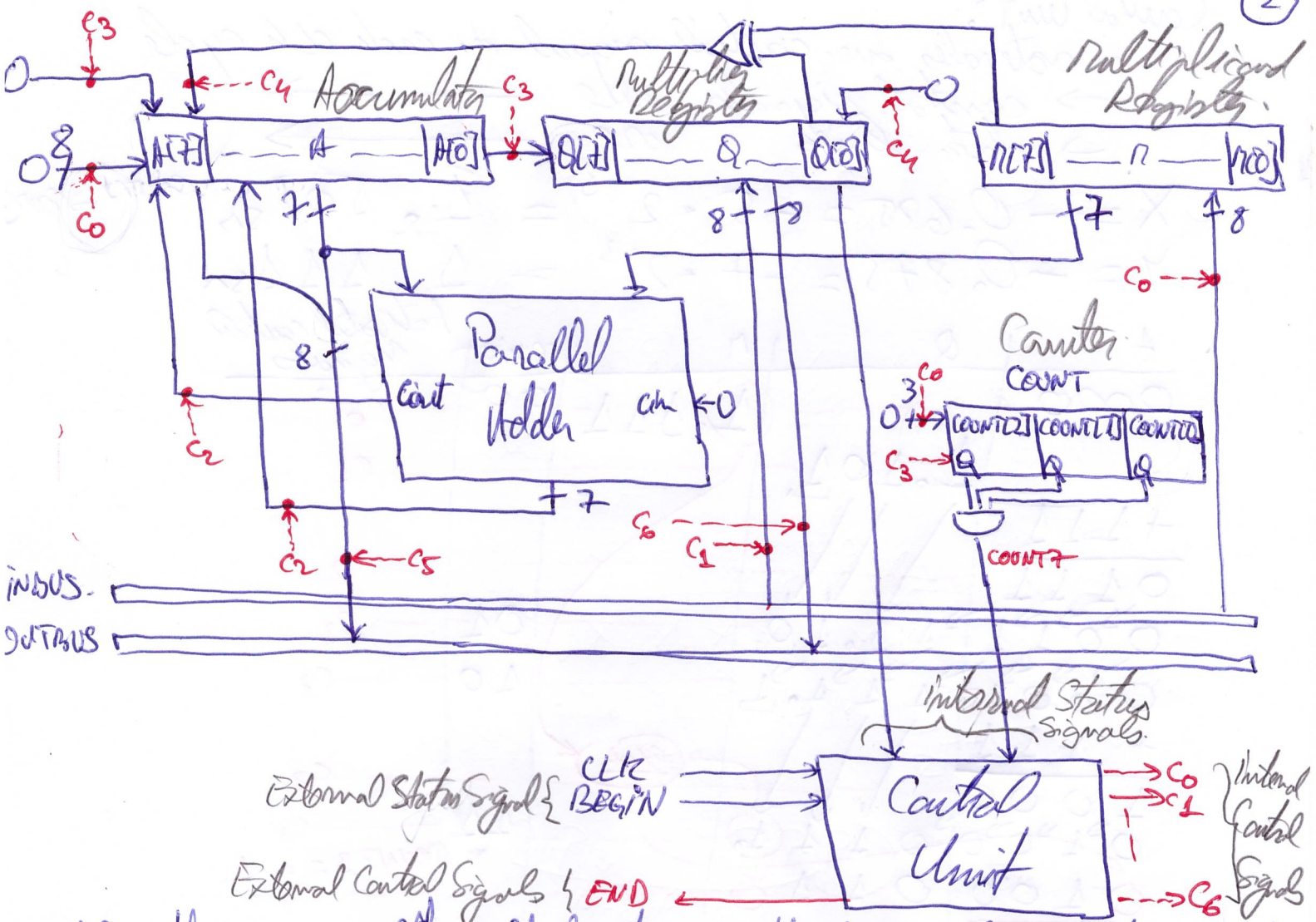
TEST2: if COUNT + 1 then goto TEST1,

SIGN: A[7] := M[7] xor Q[0], Q[0] := 0; {c_4}

OUTPUT: OUTBUS := A; {c_5}

OUTBUS := Q; {c_6}

END – {END}



Algorithm uses 3rd multiplication method: keep final 1-bit product.

iteration step
$\{ P_i := P_i + (x_i \cdot Y) \cdot 2^{-1} \rightarrow P_{i+1} := P_i \cdot 2^{-1}$

$P_i := P_i + (x_i \cdot Y) \rightarrow$ TEST & ADD

$P_{i+1} := P_i \cdot 2^{-1} \rightarrow$ RShift

if x_i (current bit of X) == 0 $\rightarrow x_i \cdot Y = 0$
skip addition of $x_i \cdot Y$ to P_i
save 1 clock cycle

! in each clock cycle the current bit of X, x_i, is stored in Q[0] !

Partial products P_i, $i \geq 0$
- at the beginning of method P_0 (8 bit) $\rightarrow$ A
- in each iteration, P_{i+1} advances into Q at RShift
  $\rightarrow$ concurrently X from Q looses the lsb

Parallel Adder:
- on 7 bits; adds the magnitudes of P_i (A) and Y (7), on 7 bit
- Cout is stored in A[7] $\Rightarrow$ no OVERFLOW

Counter COUNT: counts the 7 iterations (for the 7 bits of magnitude)
- incremented at RShift

COUNT7 is active when COUNT == 7 (1 1 1)

Control unit:
- actuals are control signal in each clk cycle
→ control signals are ——— →
→ data lines are ——————→

$X = -0.625 = -5 * 2^{-3} = \underbrace{1}_{\text{sign}}.\underbrace{101}_{5}{}_{sm}$ CONT2 → CONT3

$Y = -0.875 = -7 * 2^{-3} = 1.111_{sm}$

Activated control signals

A	Q	Q	COUNT	
0000		1.111	00	C_0
	1.101			C_1
+1111				
0111				C_2
0011	1.10		01	C_3
0001	11.1		10	C_3
+1111				
1000				C_2
0100	011		11	C_3
0100	011			C_4
0100				C_5
	0110			C_6

* COUNT3 == 1

after

$P = 0.1000110 = \underbrace{1000}_{32}\underbrace{11}_{3} * 2^{-6} = 35 * 2^{-6}$

$X * Y = -5 * 2^{-3} * -7 * 2^{-3} = +35 * 2^{-6}$

$X = -6 = 1110_{sm}$

$Y = +7 = 0111_{sm}$

! For integer multiplication, before getting the sign perform one last RShift.

A	Q	Q	COUNT
0000		0111	00
	1110		
0000	0111		01
+1111			
0111			10
0011	1011		
+1111			
1010		after	11
0101	0101		
1 0 1 0	1010		

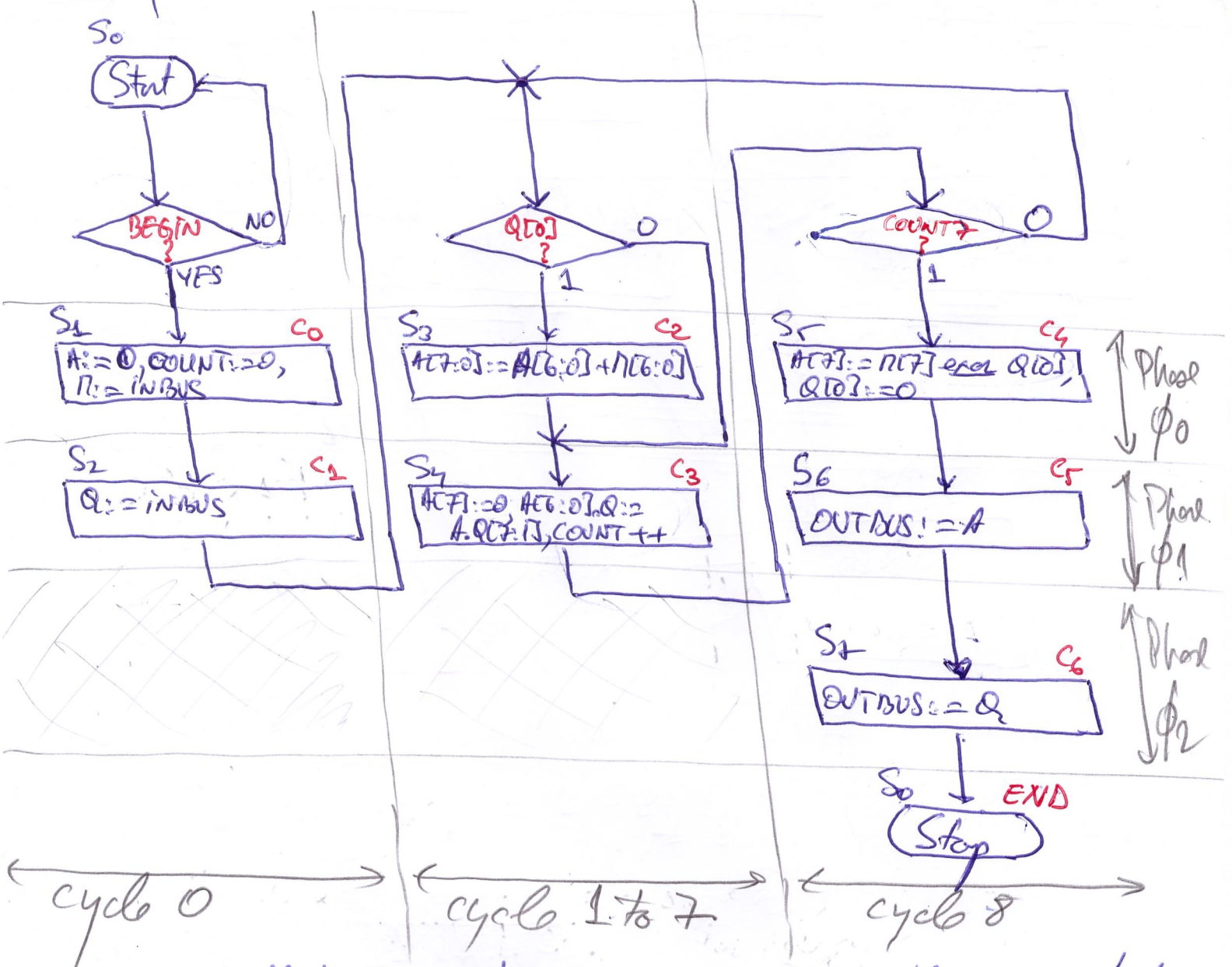
$P = 1010\,1010 = \underbrace{1010}_{32+8}\underbrace{10}_{2} = -(32+8+2) = -42$

$X * Y = -6 * 7 = -42$

3.3. Elements of Control Units Synthesis

- C.U. sequence the control signals in correct order

C.U. flowchart:



- non-conflicting operations are grouped in the same state
  - separate by ;
  - all, activate by the same control signal.
- state are grouped ↔ cycles
  ↔ phases.

(A) One Hot

- use one storage element for each state
  - at any given moment only 1 storage element is active
    → the "Hot" element.