

# **Appunti sull'Aritmetica dei Calcolatori**

**Giovanni Stea**

**a.a. 2016/17**

**Ultima modifica: 27/10/2016**

## Sommario

1	Rappresentazione dei numeri naturali.....	5
1.1	Teorema della divisione con resto .....	6
1.1.1	Proprietà dell'operatore modulo .....	7
1.2	Correttezza ed unicità della rappresentazione dei numeri in una data base.....	8
1.2.1	Rappresentazione su un numero finito di cifre .....	9
1.2.2	Esercizio (da fare a casa) .....	10
2	Elaborazione di numeri naturali tramite reti combinatorie .....	11
2.1	Complemento .....	13
2.1.1	Circuito logico per l'operazione di complemento .....	14
2.2	Moltiplicazione e divisione per una potenza della base.....	16
2.2.1	Moltiplicazione per $\beta^k$ .....	17
2.2.2	Divisione (quoziente) per $\beta^k$ .....	18
2.2.3	Modulo $\beta^k$ .....	18
2.3	Estensione di campo .....	19
2.4	Addizione .....	20
2.4.1	Full Adder in base 2 .....	22
2.4.2	Tempi di risposta e circuito di lookahead .....	24
2.4.3	Incrementatore .....	25
2.4.4	Esercizio (da fare a casa) .....	26
2.4.5	Esercizio (da fare a casa) .....	26
2.4.6	Esercizio (da fare a casa) .....	27
2.5	Sottrazione .....	27
2.5.1	Comparazione di numeri naturali.....	29
2.5.2	Sommatore/sottrattore in base 2.....	30
2.6	Moltiplicazione .....	30
2.6.1	Moltiplicatore con addizionatore $n \times 1$ in base 2 .....	33
2.6.2	Algoritmi per il calcolo iterativo del prodotto .....	34
2.6.3	Esercizio.....	35
2.7	Divisione .....	37
2.7.1	Divisore elementare in base 2 .....	41
2.7.2	Esercizio (da fare a casa) .....	42
2.7.3	Esercizio svolto .....	43
3	Rappresentazione dei numeri interi .....	47

3.1	Possibili leggi di rappresentazione dei numeri interi .....	50
3.2	Proprietà del complemento alla radice.....	52
3.2.1	Esercizio (da fare a casa) .....	54
4	Operazioni su interi in complemento alla radice .....	55
4.1	Valore assoluto.....	55
4.1.1	Circuito di conversione da CR a MS .....	57
4.2	Cambiamento di segno.....	57
4.3	Estensione di campo .....	59
4.3.1	Esercizio (da fare a casa) .....	60
4.4	Riduzione di campo .....	60
4.4.1	Esercizio (da fare a casa) .....	62
4.5	Moltiplicazione/Divisione per potenza della base .....	62
4.5.1	Shift Logico ed Aritmetico .....	62
4.6	Somma .....	63
4.7	Sottrazione .....	66
4.7.1	Comparazione di numeri interi .....	66
4.8	Moltiplicazione e divisione.....	67
4.8.1	Circuito di conversione da MS a CR .....	67
4.8.2	Moltiplicazione .....	68
4.8.3	Esercizio (da fare a casa) .....	69
4.8.4	Divisione.....	69
4.8.5	Esercizio (da fare a casa) .....	72
4.9	Conversione di base tra interi .....	73
4.9.1	Esercizio (da fare a casa) .....	74
5	Soluzioni degli esercizi per casa .....	75
5.1	Soluzione dell'esercizio 1.2.2 .....	75
5.2	Soluzione dell'esercizio 2.4.4 .....	76
5.3	Soluzione dell'esercizio 2.4.5 .....	78
5.4	Soluzione dell'esercizio 2.4.6 .....	79
5.5	Soluzione dell'esercizio 2.7.2 .....	80
5.6	Soluzione dell'esercizio 3.2.1 .....	81
5.7	Soluzione dell'esercizio 4.3.1 .....	82
5.8	Soluzione dell'esercizio 4.4.1 .....	83
5.9	Soluzione dell'esercizio 4.8.3 .....	84

5.10	Soluzione dell'esercizio 4.8.5 .....	85
5.11	Soluzione dell'esercizio 4.9.1 .....	85
6	Altri esercizi svolti .....	88
6.1	Esercizio – Febbraio 2006 (numeri naturali e interi) .....	88
6.1.1	Soluzione.....	88
6.2	Esercizio – Gennaio 2008 (numeri naturali) .....	89
6.2.1	Soluzione.....	90
6.3	Esercizio – Febbraio 2005 (numeri naturali) .....	91
6.3.1	Soluzione.....	91
6.4	Esercizio – Giugno 2004 (numeri interi) .....	92
6.4.1	Soluzione.....	93
6.5	Esercizio – Settembre 2012 (numeri interi) .....	93
6.5.1	Soluzione.....	94

## Version history

- 12/10/12: Modifiche di formule a pag. 6, 8
- 17/12/12: Modifiche “cosmetiche” su tutta la parte degli interi (riguardanti esclusivamente cose dette a lezione, o comunque dette *meglio* a lezione).
- 17/12/12: Aggiunta di esercizi svolti alla fine.
- 21/10/13: Aggiunti esercizi svolti nel testo ed in fondo.
- 06/12/13: Aggiunto l'esercizio 2.7.3 svolto a lezione. Modifiche “cosmetiche” su cose dette a lezione.
- 07/02/14: Aggiunti esercizi svolti (4.8.3, 4.8.5).
- 21/10/15: Corretti errori nel testo e negli esercizi.
- 28/10/15: Modifiche cosmetiche varie.
- 27/10/16: Modifiche cosmetiche ulteriori.

# 1 Rappresentazione dei numeri naturali

Si parte da un concetto intuitivo, che è quello di numero naturale. I naturali sono i numeri con cui si conta. Esistono definizioni più formali di cosa sia un numero naturale, ma in questo ambito non ci interessano.

Un **sistema numerico di rappresentazione** di tipo posizionale si compone di:

- 1) Un numero  $\beta \geq 2$ , detto **base di rappresentazione**. Nel caso del sistema decimale, è  $\beta = \text{dieci}$ .
- 2) Un insieme di  $\beta$  **simboli**, detti **cifre**, a ciascuno dei quali è associato **un numero naturale compreso tra 0 e  $\beta - 1$** .
- 3) Una **legge** che fa corrispondere ad ogni **sequenza di cifre** un numero naturale.

Dato il numero  $A \in N$ , lo **rappresento** in base  $\beta$  con una sequenza di cifre  $(a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ , con  $0 \leq a_i \leq \beta - 1$ ,  $0 \leq i \leq n-1$ . Dirò in tal caso che  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ . Nel caso di sistema numerico di tipo **posizionale**, la legge che fa corrispondere il numero naturale con la sua rappresentazione è

$$A = \sum_{i=0}^{n-1} a_i \cdot \beta^i.$$

Notazione **posizionale** significa che, nella rappresentazione di un numero naturale, una cifra contribuisce a determinare il numero in modo **diverso** a seconda della propria posizione. Infatti, a seconda della propria posizione sarà moltiplicata per una differente potenza della base.

La notazione posizionale non è l'unico modo possibile di rappresentare i numeri. Fino al 1200 in Europa sono stati usati i **numeri romani**, nei quali ogni simbolo ha un valore indipendente dalla propria posizione (sistema **additivo**). Fu peraltro un pisano, Leonardo Fibonacci, ad introdurre in Europa la notazione posizionale, avendola appresa dagli Arabi.

## Esempio: sistema numerico decimale

$\beta = \text{dieci}$ . Le cifre sono  $\{0,1,2,3,4,5,6,7,8,9\}$ .  $(2042)_{10} = 2 \cdot 10^0 + 4 \cdot 10^1 + 0 \cdot 10^2 + 2 \cdot 10^3$

A ben guardare, quando dico “il numero 54” sto in realtà menzionando contemporaneamente:

il numero naturale, quale concetto intuitivo (spesso verrà scritta in lettere: “dieci”).

- la sua **rappresentazione** in una qualche base (spesso 10) in notazione posizionale.

Dovremmo tener distinte le due cose, a rigor di logica. In pratica è difficile, e quindi non riusciremo a farlo sempre.

### **Esempio: sistema numerico esadecimale**

$\beta = \text{sedici}$ . Le cifre sono  $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ .

Abbiamo che  $(A)_{16} = (10)_{10}, \dots, (F)_{16} = (15)_{10}$

$$\begin{aligned}(1A2F)_{16} &= (F)_{16} \cdot (16^0)_{10} + (2)_{16} \cdot (16)_{10} + (A)_{16} \cdot (16^2)_{10} + (1)_{16} \cdot (16^3)_{10} \\ &= (15)_{10} \cdot (16^0)_{10} + (2)_{10} \cdot (16)_{10} + (10)_{10} \cdot (16^2)_{10} + (1)_{10} \cdot (16^3)_{10} \\ &= 15 \cdot 1 + 2 \cdot 16 + 10 \cdot 256 + 1 \cdot 4096 = 9231\end{aligned}$$

### **Esempio: sistema numerico binario**

$\beta = \text{due}$ . Le cifre sono  $\{0,1\}$ .

Questo è il sistema usato nei calcolatori per effettuare operazioni aritmetiche.

## **1.1 Teorema della divisione con resto**

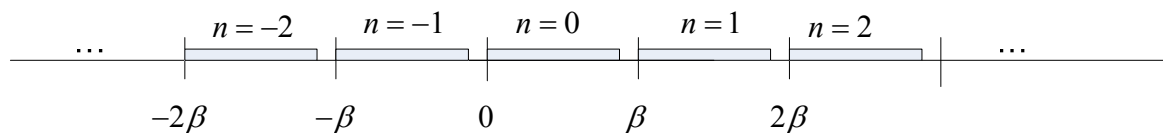
Domande. Data una base di rappresentazione, posso **sempre** rappresentare un numero in quella base? La sua rappresentazione, data una base, è **unica**? Come faccio a **trovarla**? **Quante cifre** mi servono? La risposta a tutte queste domande passa per il seguente:

### **Teorema della Divisione con Resto**

Dato  $x \in \mathbb{Z}$ ,  $\beta \in \mathbb{N}$ ,  $\beta > 0$ , **esiste ed è unica la coppia di numeri**  $q, r$ , con  $q \in \mathbb{Z}$  e  $r \in \mathbb{N}$ ,  $\boxed{0 \leq r < \beta}$ , tale che  $x = q \cdot \beta + r$ .

### **Dimostrazione**

**Esistenza:** Dimostriamo che una coppia di numeri con queste caratteristiche esiste sempre. Si prende l'asse dei numeri interi e lo si divide in blocchi  $[n \cdot \beta, (n+1) \cdot \beta[$ ,  $n \in \mathbb{Z}$ .



L'unione di questi intervalli ricopre tutto quanto l'asse dei numeri interi.  $\bigcup_{n \in \mathbb{Z}} [n \cdot \beta, (n+1) \cdot \beta[ \equiv \mathbb{Z}$ .

Pertanto, il numero  $x$  fa parte di un intervallo, sia il  $q$ -simo. Allora  $q \cdot \beta \leq x < (q+1) \cdot \beta$ . Definisco allora  $r = x - q \cdot \beta$ , ed ho garanzia che  $0 \leq r < \beta$ . Ho quindi dimostrato che una tale coppia esiste sempre.

**Unicità:** Supponiamo per assurdo che esistano **due** coppie  $(q_1, r_1)$  e  $(q_2, r_2)$  **diverse** tali che  $x = q_1 \cdot \beta + r_1 = q_2 \cdot \beta + r_2$ , con  $q_i \in \mathbb{Z}$  e  $0 \leq r_i < \beta$ . Allora  $(q_1 - q_2) \cdot \beta = r_2 - r_1$ . Però per ipotesi  $0 \leq r_1 < \beta$  e  $0 \leq r_2 < \beta$ , con il che  $-\beta < (r_2 - r_1) < \beta$ .

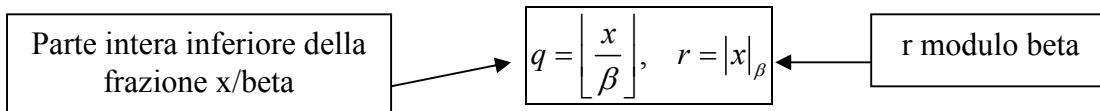
Da questo si ricava che  $-\beta < (q_1 - q_2) \cdot \beta < \beta$ , cioè che  $-1 < (q_1 - q_2) < 1$ . Visto che  $q_1, q_2 \in \mathbb{Z}$ , ne consegue che  $q_1 = q_2$ .

Visto che  $q_1 = q_2$ , allora anche  $r_1 = r_2$ , contro l'ipotesi. Questo dimostra che la divisione col resto ha un **unico risultato**.

Attenzione: l'unicità è garantita dal fatto che  **$r$  è un numero naturale, compreso tra 0 e  $\beta-1$** .



In forza di quanto appena dimostrato, diamo a  $q$  il nome di **quoziente**, e ad  $r$  il nome di **resto** della divisione di  $x$  per  $\beta$ . D'ora in avanti li indichiamo come:



### 1.1.1 Proprietà dell'operatore modulo

L'operatore modulo ha una serie di proprietà, che vanno sapute perché saranno usate nel seguito.

Dato  $\alpha \in \mathbb{N}, \alpha > 0$ :

1)  $\boxed{|x + k \cdot \alpha|_\alpha = |x|_\alpha}$ ,  $k \in \mathbb{Z}$ . Infatti  $x = \left\lfloor \frac{x}{\alpha} \right\rfloor \cdot \alpha + |x|_\alpha$ , e quindi  $x + k \cdot \alpha = \left( \left\lfloor \frac{x}{\alpha} \right\rfloor + k \right) \cdot \alpha + |x|_\alpha$ . Ma

$\left\lfloor \frac{x}{\alpha} \right\rfloor + k$  è sempre un numero intero, e  $|x|_\alpha$  è sempre un numero compreso tra 0 e  $\alpha - 1$ . Quindi,

per l'unicità del teorema della divisione con resto, quelli sono quoziente e resto della divisione per  $\alpha$  di  $x + k \cdot \alpha$ .

2)  $\boxed{|x|_\alpha = |x|_{\alpha+A}}$ ,  $A \in \mathbb{N}$ . Anche qui:  $0 \leq |x|_\alpha < \alpha \leq \alpha + A$ . Quindi, la divisione di  $|x|_\alpha$  per  $\alpha + A$  ha quoziente nullo e resto  $|x|_\alpha$ .

3)  $\boxed{|x + y|_\alpha = ||x|_\alpha + |y|_\alpha|_\alpha}$ . Infatti,  $|x + y|_\alpha = ||x|_\alpha + |y|_\alpha + (q_x + q_y) \cdot \alpha|_\alpha$ . Ma applicando la prima proprietà dimostrata si ottiene la tesi.

4)  $\boxed{|x \cdot y|_\alpha = ||x|_\alpha \cdot |y|_\alpha|_\alpha}$ . Infatti,

$$\begin{aligned}
 |x \cdot y|_\alpha &= \left| (|x|_\alpha + q_x \cdot \alpha) \cdot (|y|_\alpha + q_y \cdot \alpha) \right|_\alpha = |x|_\alpha \cdot |y|_\alpha + (|x|_\alpha \cdot q_y \cdot \alpha) + (|y|_\alpha \cdot q_x \cdot \alpha) + q_x \cdot q_y \cdot \alpha^2 \Big|_\alpha \\
 &= |x|_\alpha \cdot |y|_\alpha + k \cdot \alpha \Big|_\alpha
 \end{aligned}$$

Ed applicando ancora una volta la prima proprietà si dimostra anche questa.

## 1.2 Correttezza ed unicità della rappresentazione dei numeri in una data base

Il teorema della divisione con resto mi consente di trovare la rappresentazione di un numero naturale in una qualunque base. Data una base  $\beta$ , devo trovare  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ ,  $n$  cifre tali che

$$A = \sum_{i=0}^{n-1} a_i \cdot \beta^i.$$

Le trovo applicando iterativamente il teorema del resto (**algoritmo delle divisioni successive, o MOD & DIV**):

$$\begin{aligned} A &= q_1 \cdot \beta + a_0 \\ q_1 &= q_2 \cdot \beta + a_1 \\ &\dots \\ q_{n-1} &= 0 \cdot \beta + a_{n-1} \end{aligned}$$

Mi fermo, ovviamente, quando l'ultimo quoziente è nullo. A quel punto, la  $n$ -upla di resti, letta **dal basso verso l'alto**, costituisce l'insieme di cifre che rappresentano l'intero  $A$  in base  $\beta$ . Vediamo di dimostrarlo:

Sostituendo a  $q_i$  la propria definizione, si ottiene:

$$A = a_0 + \beta \cdot q_1 = a_0 + \beta \cdot (a_1 + \beta \cdot (a_2 + \beta \cdot (\dots)))$$

E quindi  $A = \sum_{i=0}^{n-1} a_i \cdot \beta^i$ .

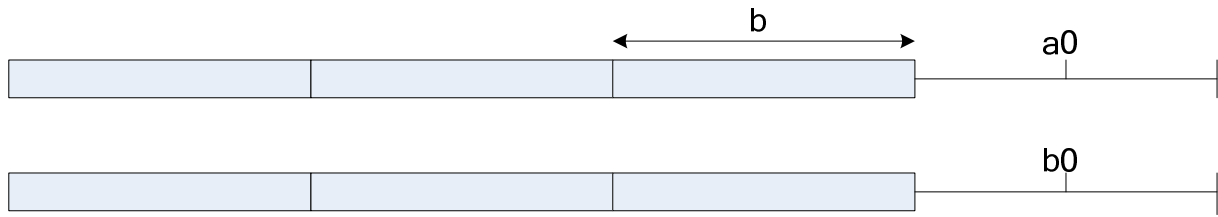
Inoltre, il teorema della divisione con resto garantisce anche che la  $n$ -upla di cifre che ho trovato è **unica**. Infatti, supponiamo per assurdo che ne esistano due  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ ,  $A \equiv (b_{n-1}b_{n-2}\dots b_1b_0)_\beta$ . Supponendo quindi che esistano  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ ,  $A \equiv (b_{n-1}b_{n-2}\dots b_1b_0)_\beta$ ,

tali che  $\sum_{i=0}^{n-1} a_i \cdot \beta^i = \sum_{i=0}^{n-1} b_i \cdot \beta^i$ , si ottiene che:

$$a_0 + \beta \cdot \left( \sum_{i=0}^{n-2} a_{i+1} \cdot \beta^i \right) = b_0 + \beta \cdot \left( \sum_{i=0}^{n-2} b_{i+1} \cdot \beta^i \right)$$

Ma  $0 \leq a_0 \leq \beta - 1$ ,  $0 \leq b_0 \leq \beta - 1$ , e quindi per l'unicità si ottiene necessariamente che  $a_0 = b_0$ .





Quindi, andando avanti:

$$\cancel{a_0} + \beta \cdot \left( \sum_{i=0}^{n-2} a_{i+1} \cdot \beta^i \right) = \cancel{b_0} + \beta \cdot \left( \sum_{i=0}^{n-2} b_{i+1} \cdot \beta^i \right)$$

$$a_1 + \beta \cdot \sum_{i=1}^{n-2} a_{i+1} \cdot \beta^{i-1} = b_1 + \beta \cdot \sum_{i=1}^{n-2} b_{i+1} \cdot \beta^{i-1}$$

Ma allora anche  $a_1 = b_1$ , e così via per tutti gli altri. Pertanto la rappresentazione è **unica**.

Peraltro, quella trovata prima è una maniera **algoritmica** di trovare le cifre della rappresentazione di un numero naturale in una qualunque base.

### Quante cifre servono?

$$q_i = \left\lfloor \frac{A}{\beta^i} \right\rfloor. \text{ Infatti, } q_1 = \left\lfloor \frac{A}{\beta} \right\rfloor, \left\lfloor \frac{A}{\beta^2} \right\rfloor = \left\lfloor \frac{a_0 + a_1 \cdot \beta + \beta^2 \cdot q_2}{\beta^2} \right\rfloor = \left\lfloor \frac{a_0 + a_1 \cdot \beta}{\beta^2} \right\rfloor + q_2 = q_2 \text{ etc.}$$

Quindi, se  $\beta \geq 2$ ,  $\exists n : q_n = 0$ . Il che conferma che l'algoritmo termina sempre. Basta sempre un numero finito di cifre. Prendiamo il più piccolo  $n$  che verifica quella proprietà, cioè quello per cui  $q_{n-1} \neq 0$ , a questo punto,  $\beta^{n-1} \leq A < \beta^n$ . Ho quindi bisogno di  $n$  cifre per rappresentare  $A$  in base  $\beta$ . Per calcolare  $n$  posso osservare che:

$\beta^n \geq A+1 \Leftrightarrow n = \lceil \log_\beta (A+1) \rceil$ , oppure, in alternativa,  $n = \lfloor \log_\beta A \rfloor + 1$ . Le due espressioni sono identiche (sia che il  $\log A$  sia intero, sia che non sia intero).

### 1.2.1 Rappresentazione su un numero finito di cifre

Ovviamente, se ho a disposizione una quantità finita di cifre in una data base, potrò rappresentare una quantità finita di numeri naturali. In particolare, data una base  $\beta$  ed  $n$  cifre, non potrò che rappresentare quei numeri per i quali l'algoritmo di cui sopra termina entro  $n$  passi, cioè quei numeri tali per cui  $A < \beta^n$ . Quindi, il numero naturale **più grande** che posso rappresentare è  $\beta^n - 1$ . Qual è la rappresentazione di questo numero?

In base 10, il numero più grande che posso rappresentare su  $n$  cifre è 999...99. In generale, in base  $\beta$ , è quello che ottengo quando tutte le cifre hanno **valore massimo**, cioè  $a_i = \beta - 1$ . Infatti, si ottiene:

$$A = \sum_{i=0}^{n-1} (\beta - 1) \cdot \beta^i = \sum_{i=0}^{n-1} \beta^{i+1} - \sum_{i=0}^{n-1} \beta^i = \sum_{i=1}^n \beta^i - \sum_{i=0}^{n-1} \beta^i = \beta^n - 1$$

Questo è un modo di rappresentare  $\beta^n - 1$ . Ma visto che rappresentazione è unica, è la rappresentazione di  $\beta^n - 1$ .

### 1.2.2 Esercizio (da fare a casa)

1) Sia  $A = (a_{n-1} \dots a_0)_\beta$  un numero naturale rappresentato su  $n$  cifre in base  $\beta$ . Si dimostri che, dato  $\gamma$  sottomultiplo di  $\beta$ ,  $|A|_\gamma = |a_0|_\gamma$ . Si osservi che, per  $\beta = 10$ , si ricava il (noto) criterio di divisibilità per due e per cinque dei numeri naturali.

2) Sia  $A = (a_{n-1} \dots a_0)_4$  un numero naturale rappresentato su  $n$  cifre in base quattro. Dimostrare che  $|A|_3 = 0$  se e solo se  $\left| \sum_{i=0}^{n-1} a_i \right|_3 = 0$ , ovvero che  $A$  è multiplo di tre se e solo se lo è la somma delle sue cifre.

3) Estendere la precedente dimostrazione al caso di numero  $A$  in base  $\beta > 2$  generica: sia  $A = (a_{n-1} \dots a_0)_\beta$  un numero naturale rappresentato su  $n$  cifre in base  $\beta$ ,  $\beta > 2$ , dimostrare che  $|A|_\gamma = 0$  se e solo se  $\left| \sum_{i=0}^{n-1} a_i \right|_\gamma = 0$ , dove  $\gamma$  è un qualunque sottomultiplo di  $\beta - 1$ .

Si osservi che, per  $\beta = 10$ , si ricava il (noto) criterio di divisibilità per tre dei numeri naturali.

4) Dato  $A$  in base  $\beta$  su  $n$  cifre dimostrare che  $|A|_{(\beta+1)} = 0$  se e solo se  $\left| \sum_{i=0}^{n-1} (-1)^i \cdot a_i \right|_{(\beta+1)} = 0$ . Si osservi che, per  $\beta = 10$ , si ricava il (probabilmente poco noto) criterio di divisibilità per undici dei numeri naturali.

**Nota:** per risolvere i punti 2, 3, 4, può far comodo avvalersi della (nota) formula dello sviluppo di binomio di Newton, qui richiamata per facilitare lo studente:

$$(a + b)^n = \sum_{k=0}^n \left[ \binom{n}{k} \cdot a^{n-k} \cdot b^k \right]$$

### Soluzione

## 2 Elaborazione di numeri naturali tramite reti combinatorie

Tutto quanto quello che è stato detto finora prescinde dalla rappresentazione dell'informazione all'interno di un calcolatore.

Il nostro obiettivo è costruire **reti logiche** che elaborino numeri **naturali** rappresentati in una data base  $\beta$ . Reti, cioè, che **producano in uscita le cifre in base  $\beta$  del risultato**, date le **cifre in base  $\beta$  degli operandi come ingresso**.

Le reti logiche, ovviamente, operano su **variabili logiche**. Quindi, per raggiungere l'obiettivo, sono necessarie due cose.

- 1) stabilire **la codifica** di una cifra in base  $\beta$  in termini di variabili logiche.
- 2) Descrivere e sintetizzare la rete logica che svolge l'operazione

Che tipo di reti logiche saranno quelle che andrò a costruire? Saranno **reti combinatorie**, in quanto ad ogni stato di ingresso (operandi di un'operazione) corrisponderà **uno** stato di uscita (risultato dell'operazione).

Per quanto riguarda la codifica, vediamo di farsi qualche idea:

- se  $\beta = 2$ , la codifica è banale. Una variabile logica codifica una cifra in base 2.
- Se  $\beta > 2$ , ci vorranno **un certo numero di variabili logiche** per codificare una cifra in base  $\beta$

**Esempio:** codifica BCD per le cifre in base 10

**stringa di 4 bit.**

$J$	$x_3$	$x_2$	$x_1$	$x_0$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Posso costruire una rete combinatoria che esegua la **somma** di due numeri in base 10 codificati BCD su 1 cifra, e mi produca il risultato su 2 cifre (infatti, il risultato massimo è  $9+9=18$ , che non sta su una cifra). Tale rete combinatoria avrà  $4+4=8$  variabili logiche di ingresso e  $4+4=8$  variabili logiche di uscita, e la chiamerò **sommatore ad 1 cifra per numeri naturali in base 10 con codifica BCD**. Questa **non** è una rete che opera somme su numeri in base 2.

Infatti, se do in ingresso al sommatore i numeri naturali 1 e 9, codificati BCD. L'uscita, su 2 cifre, dovrà essere corrispondente al numero 10. Quindi:

- stato di ingresso: 0001 1001 (codifica BCD dei due addendi)
- stato di uscita: 0001 0000 (codifica BCD delle due cifre del risultato)

Questo circuito **non ha eseguito la somma in base 2 degli ingressi**. Se lo avesse fatto, avrebbe dovuto produrre

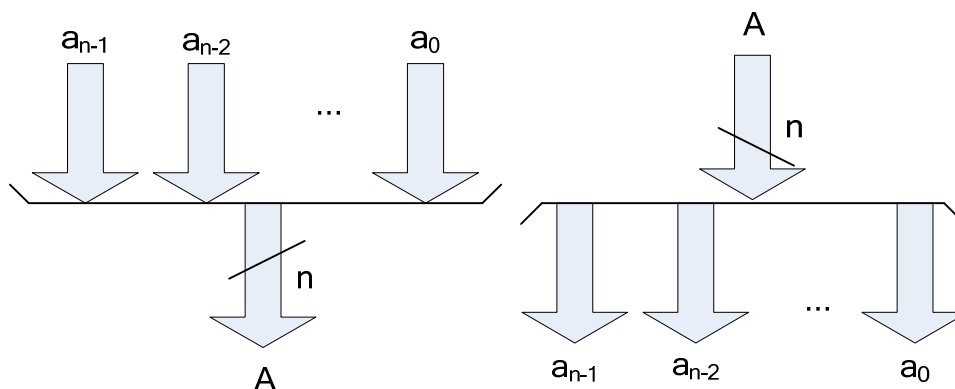
- un'uscita su 5 cifre binarie, dato che può valere al massimo 18
- un'uscita pari a 01010, che è effettivamente la somma degli ingressi, considerati come numeri in base 2.

Nel seguito, faremo quanto segue:

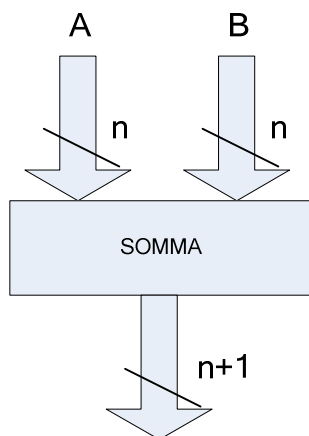
1. prendiamo in esame le operazioni aritmetiche di base (somme, sottrazioni, etc.). Ne daremo una descrizione **indipendente dalla base**, valida quindi per qualunque base.
2. Cercheremo, facendo conto sulle **proprietà della notazione posizionale**, di rendere tali operazioni **più semplici**.
3. Dettaglieremo le reti logiche (a livello di porte elementari) che le eseguono **in base 2**, che è la base in cui lavorano i calcolatori.

Per poterlo fare, è necessario dotarsi di una notazione **indipendente dalla base**.

Per rappresentare graficamente il fatto che un numero  $A$  è individuato da una serie di  $n$  cifre in base  $\beta$ , scriveremo:

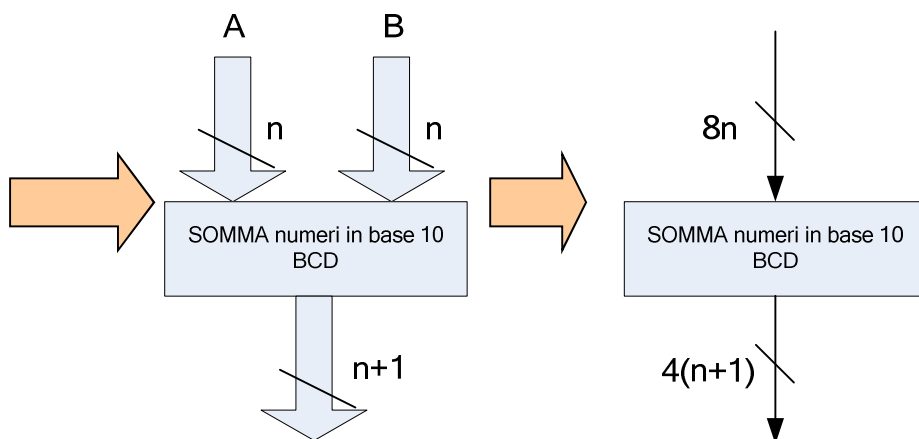


Ciascuna di queste doppie frecce rappresenta il numero di variabili logiche necessario a **codificare** una cifra in base  $\beta$ . Per esempio, per  $\beta = 10$  avremmo quattro variabili logiche per cifra. Nel caso particolare  $\beta = 2$  useremo invece **frecce singole**, perché una cifra in base 2 può essere codificata da una variabile logica.



Ad esempio, vogliamo poter costruire reti tipo questa, che prendono in ingresso due numeri in base  $\beta$  e producono la somma di questi due numeri.

$J$	$x_3$	$x_2$	$x_1$	$x_0$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



## 2.1 Complemento

È un'operazione particolarmente ricorrente, anche se a prima vista non sembra utile. Dato un numero  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$ , rappresentato in base  $\beta$  su  $n$  cifre,  $0 \leq A < \beta^n$ , definisco **complemento di A (in base  $\beta$  su  $n$  cifre) il numero:**

$$\overline{A} \triangleq \beta^n - 1 - A$$

Cioè,  $\overline{A}$  è quel numero che sommato ad  $A$  dà il **massimo numero** rappresentabile in base  $\beta$  su  $n$  cifre.

### Esempi:

$$- \overline{(1034)}_{10} = (8965)_{10}$$

$$- \overline{(1034)}_5 = (3410)_5$$

**Attenzione:** il complemento richiede che si specifichi il **numero di cifre**. Infatti,

$$\overline{(001034)}_{10} = (998965)_{10}.$$

I numeri di partenza sono gli stessi, ma rappresentati su un numero diverso di cifre. Il complemento è **diverso**.

Devo sintetizzare reti che calcolano **cifre in uscita** a partire da **altre cifre in ingresso**. Come faccio a trovare le cifre di  $\bar{A}$  conoscendo quelle di  $A$ ?

**La prima cosa da chiedersi** quando si definisce un'operazione è **su quante cifre sta il risultato**.

Nel nostro caso, se  $0 \leq A < \beta^n$ , allora anche  $0 \leq \bar{A} < \beta^n$ , quindi sono certo che  $\bar{A}$  è rappresentabile su  $n$  cifre. Dalla definizione ricavo che:

$$\bar{A} = \sum_{i=0}^{n-1} (\beta-1) \beta^i - \sum_{i=0}^{n-1} a_i \beta^i = \sum_{i=0}^{n-1} (\beta-1-a_i) \beta^i$$

$\uparrow$   
 $\beta^n - 1$

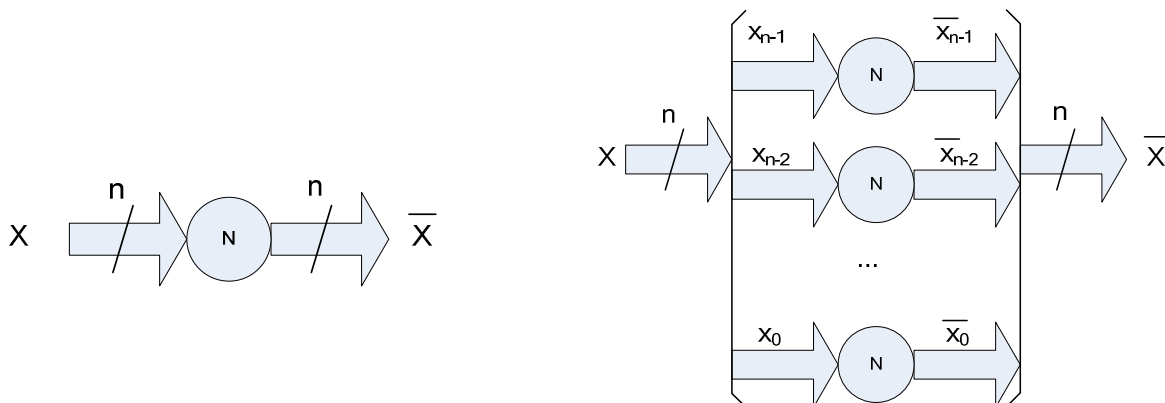
Ma:

- $\beta-1-a_i$  è una cifra in base  $\beta$ , in quanto compresa tra 0 e  $\beta-1$ .
- $\beta-1-a_i$ , dalla definizione, è il **complemento del numero naturale  $a_i$  su una cifra in base  $\beta$** .

Quindi, le cifre in base  $\beta$  della rappresentazione di  $\bar{A}$  sono  $\bar{a}_i = \beta-1-a_i$ . Ogni cifra di  $\bar{A}$  si ottiene complementando la corrispondente cifra di  $A$ .  $\bar{A} \equiv (\bar{a}_{n-1} \bar{a}_{n-2} \dots \bar{a}_1 \bar{a}_0)_\beta$

### 2.1.1 Circuito logico per l'operazione di complemento

Supponiamo di voler sintetizzare un circuito che esegua il complemento di un numero in base  $\beta$  su  $n$  cifre.



Basta quindi che sappia fare una rete che fa il complemento di **una singola cifra**. Vediamo come sono fatte le reti nelle varie basi.

In base  $\beta=2$ , tale rete è estremamente semplice. Infatti, è la rete elementare che:

- se la cifra è 1 ritorna 0
- se la cifra è 0 ritorna 1

Cioè è la porta elementare **NOT**. In base 2 il complemento di un numero su  $n$  cifre si fa con una barriera di  $n$  invertitori. Ciò accade perché le cifre in base 2 (0, 1) sono codificate in termini di una singola variabile logica.

Vediamo adesso di ragionare in **base 10**. Scegliamo **una codifica**, e vediamo come si sintetizza la rete che realizza il complemento di una singola cifra in quella codifica. Scegliamo l'unica che conosciamo, cioè la **BCD** (detta anche **8421**, dal peso associato a ciascuna cifra).

$J$	$X$				$\overline{X}$				$J$
	$x_3$	$x_2$	$x_1$	$x_0$	$z_3$	$z_2$	$z_1$	$z_0$	
0	0	0	0	0	1	0	0	1	9
1	0	0	0	1	1	0	0	0	8
2	0	0	1	0	0	1	1	1	7
3	0	0	1	1	0	1	1	0	6
4	0	1	0	0	0	1	0	1	5
5	0	1	0	1	0	1	0	0	4
6	0	1	1	0	0	0	1	1	3
7	0	1	1	1	0	0	1	0	2
8	1	0	0	0	0	0	0	1	1
9	1	0	0	1	0	0	0	0	0

A regola andrebbe disegnato anche il resto della tabella di verità. Tanto sappiamo che le uscite non sono specificate per  $J > 9$

Vediamo di affrontarne la sintesi. Per fare le cose bene andrebbe usata 4 volte la mappa di Karnaugh (una per ciascuna uscita). Però mi interessa arrivare ad una sintesi qualunque, non fare quella a costo minimo, e quindi posso procedere a occhio, a rischio di fare una sintesi non ottima:

- si vede al volo che  $z_0 = \overline{x_0}$ ,  $z_1 = x_1$
- si vede al volo che  $z_3 = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1}$
- si vede al volo che  $z_2 = x_1 \oplus x_2$  (posso scriverlo perché ci sono degli stati non specificati)

Quindi, per fare il complemento di una cifra in base 10 – codificata BCD – si deve usare un po' di logica. Non troppa, ma un po' sì.

Ai fini della realizzazione del **complemento** in modo efficiente, si possono pensare altre codifiche per la base 10, dette **autocomplementanti**. Una codifica si dice autocomplementante quando “**il complemento della rappresentazione è uguale alla rappresentazione del complemento**”, quando cioè posso ottenere il complemento di una cifra semplicemente facendo il complemento delle **variabili logiche** che la codificano. La codifica BCD non è autocomplementante.

### Esempio: codifica “eccesso 3”

Si rappresenta una cifra  $X$  in base 10 come la cifra  $X+3$  in base 2.

$J$	$X$			
	$x_3$	$x_2$	$x_1$	$x_0$
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1

5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

**Esempio:** codifica “2421”

Ogni cifra decimale viene codificata con una quaterna di bit. A ciascuna di queste si assegna peso 2, 4, 2, 1, a seconda della posizione. Con questo metodo esistono, per alcune cifre, diverse scelte equivalenti. Ad esempio, posso decidere che 3 è rappresentato come 1001 o come 0011. Ad esempio 6 può essere rappresentato come 1100 o come 0110. Però se scelgo che 3 è rappresentato come 1001, vorrà dire che sceglierò il suo complemento, cioè 6, come 0110.

$J$	$X$			
	$x_3$	$x_2$	$x_1$	$x_0$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Per tali due codifiche posso utilizzare circuiti di complemento fatti da una **barriera di invertitori**, più semplici dei precedenti.

**Osservazione (importante):**

- Come faccio il complemento di una cifra in base  $\beta$  dipende dalla **base** e dalla **codifica**.
- Il fatto che il complemento di un numero su  $n$  cifre possa essere eseguito complementando le singole cifre non dipende né dalla base né dalla codifica. È in realtà una **proprietà della notazione posizionale**. La ho infatti descritta in termini algebrici, senza indicare una base precisa.

Nel seguito di questa parte del corso descriveremo proprietà della notazione posizionale che consentono, come quella vista adesso, di **scindere problemi complessi in sottoproblemi più semplici**, in maniera **indipendente dalla base e dalla codifica**.

## 2.2 Moltiplicazione e divisione per una potenza della base

Devo eseguire una moltiplicazione (divisione) per  $\beta^k$ .

**Domande:**



- Quando, in base 10, devo calcolare  $25 \times 1000$ , faccio forse i conti? No, aggiungo tre zeri in fondo al numero.
- Quando devo calcolare  $2562/100$ , resto e quoziente, faccio forse i conti? No: il resto è 62, ed il quoziente è 25.

Tutto questo perché sto moltiplicando e dividendo per una **potenza della base** in cui lavoro (la base 10, appunto). Questa è un'altra **proprietà della notazione posizionale**, valida in qualunque base. Vediamo di dare una dimostrazione formale delle proprietà che abbiamo appena intuito.

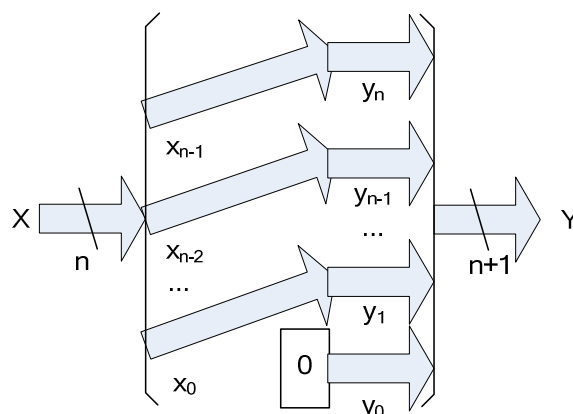
### 2.2.1 Moltiplicazione per $\beta^k$

Dato  $X \equiv (x_{n-1}x_{n-2}\dots x_0)_\beta$ , definisco  $Y = \beta \cdot X$ , e voglio trovare le cifre che rappresentano  $Y$ . Si fa inizialmente per  $k = 1$ , e poi si generalizza.

- **La prima domanda da farsi** quando si considera un'operazione (qualunque) è: **su quante cifre sta il risultato?**  $X \leq \beta^n - 1 \Rightarrow Y \leq (\beta^n - 1) \cdot \beta < \beta^{n+1} - 1$ , cioè  $Y$  è rappresentabile sicuramente su  $n+1$  cifre.
- $X = \sum_{i=0}^{n-1} x_i \beta^i \Rightarrow Y = \sum_{i=0}^n y_i \beta^i = \sum_{i=0}^{n-1} x_i \beta^{i+1} = \sum_{i=1}^n x_{i-1} \beta^i$ . Ma, dato che  $Y \equiv (y_n y_{n-1} \dots y_0)_\beta$ , visto che ho appena dimostrato che sta su  $n+1$  cifre, dalla precedente uguaglianza ottengo:  
 $Y \equiv (y_n y_{n-1} \dots y_0)_\beta \equiv (x_{n-1} x_{n-2} \dots x_0 0)_\beta$ . Quindi, si ottiene:

$$y_i = \begin{cases} x_{i-1} & 1 \leq i \leq n \\ 0 & i = 0 \end{cases}$$

Questa è una soluzione della precedente uguaglianza. Visto che la rappresentazione di un numero in una data base è **unica**, allora questa è l'unica soluzione. Quindi:  $Y \equiv (x_{n-1} x_{n-2} \dots x_0 0)_\beta$ . La rete che implementa questa cosa è a **complessità nulla (shift sinistro)**. Così come il procedimento mentale per ottenere i risultati è di complessità nulla, possiamo sintetizzare una rete di complessità nulla che esegue questo procedimento.



Ovviamente, il tutto si generalizza alla moltiplicazione per  $\beta^k$ . Il numero risultante ha  $n+k$ , cifre, le ultime  $k$  delle quali sono nulle, e le prime  $n$  delle quali sono le cifre di  $X$ .

$$y_i = \begin{cases} x_{i-k} & k \leq i \leq n+k-1 \\ 0 & 0 \leq i \leq k-1 \end{cases}$$

### 2.2.2 Divisione (quoziante) per $\beta^k$

Anche in questo caso facciamo i conti inizialmente per  $k=1$ , e poi generalizziamo. Dato  $X \equiv (x_{n-1}x_{n-2}\dots x_0)_\beta$ , definisco  $Y = \lfloor X/\beta \rfloor$ , e voglio trovare le cifre di  $Y$ .

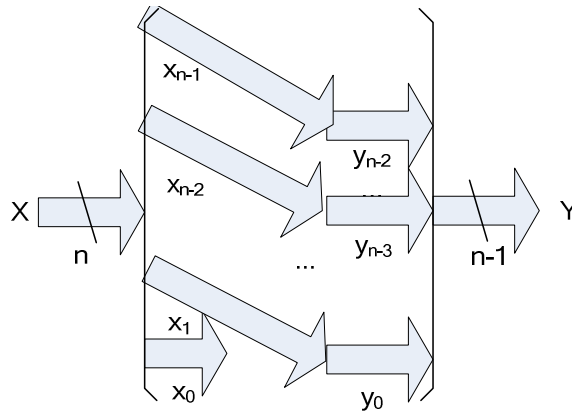
- Su quante cifre sta il risultato?  $X \leq \beta^n - 1 \Rightarrow Y \leq \lfloor \beta^{n-1} - 1/\beta \rfloor < \beta^{n-1}$ , cioè  $Y$  è rappresentabile sicuramente su  $n-1$  cifre

$$X = \sum_{i=0}^{n-1} x_i \beta^i \Rightarrow Y = \left\lfloor \frac{\sum_{i=0}^{n-1} x_i \beta^i}{\beta} \right\rfloor = \left\lfloor \frac{x_0 + \sum_{i=1}^{n-1} x_i \beta^i}{\beta} \right\rfloor = \left\lfloor \frac{x_0}{\beta} + \sum_{i=0}^{n-2} x_{i+1} \beta^i \right\rfloor = \sum_{i=0}^{n-2} x_{i+1} \beta^i. \text{ L'ultimo pas-}$$

saggio deriva dal fatto che  $x_0/\beta < 1$  e la sommatoria è un numero intero. Quindi, si ottiene:

$$y_i = x_{i+1}, \quad 0 \leq i \leq n-2$$

Anche in questo caso l'operazione richiede una rete di complessità nulla:



Ed anche in questo caso, il tutto si generalizza alla divisione per  $\beta^k$ . Il numero risultante ha  $n-k$ , cifre, corrispondenti alle  $n-k$  cifre più significative di  $X$ .

$$y_i = x_{i+k}, \quad 0 \leq i \leq n-1-k$$

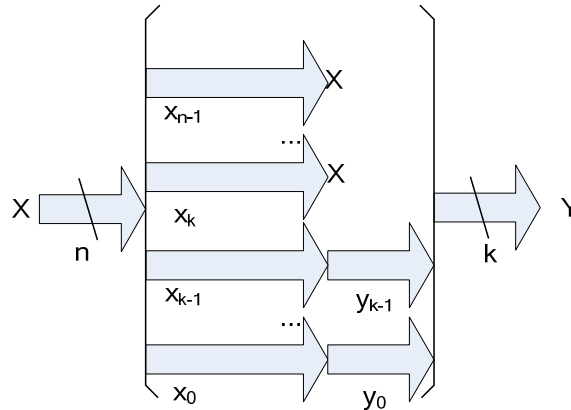
### 2.2.3 Modulo $\beta^k$

Dato  $X \equiv (x_{n-1}x_{n-2}\dots x_0)_\beta$ , definisco  $Y = |X|_{\beta^k}$ , e voglio trovare le cifre di  $Y$ .

- $X \leq \beta^n - 1 \Rightarrow Y \leq \left| \beta^n - 1 \right|_{\beta^k} \leq \beta^k - 1$ , cioè  $Y$  è rappresentabile sicuramente su  $k$  cifre (per la stessa definizione di modulo, cioè resto della divisione).
- $X = \sum_{i=0}^{n-1} x_i \beta^i \Rightarrow Y = \left| \sum_{i=0}^{n-1} x_i \beta^i \right|_{\beta^k} = \left| \sum_{i=0}^{k-1} x_i \beta^i + \sum_{i=k}^{n-1} x_i \beta^i \right|_{\beta^k} = \left| \sum_{i=0}^{k-1} x_i \beta^i + \beta^k \cdot \sum_{i=k}^{n-1} x_i \beta^{i-k} \right|_{\beta^k} = \sum_{i=0}^{k-1} x_i \beta^i$ .

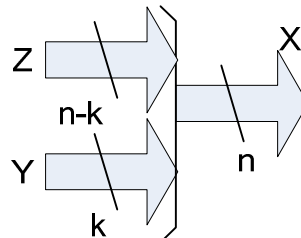
Quindi, si ottiene:  $y_i = x_i, 0 \leq i \leq k-1$

Anche in questo caso l'operazione richiede una rete di complessità nulla:



Quindi, **dati due numeri  $Y$  e  $Z$ , rispettivamente a  $k$  ed  $n-k$  cifre**, l'operazione di **concatenamento**  $X = Z \cdot \beta^k + Y$ , che produce un numero su  $n$  cifre, è di **complessità nulla**.

Allo stesso modo, ha complessità nulla l'operazione inversa di **scomposizione** di un numero su  $n$  cifre in due blocchi di  $k$  ed  $n-k$  cifre. Useremo spesso queste due operazioni.



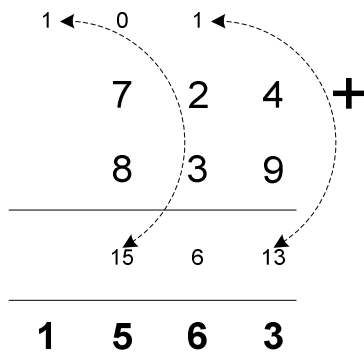
## 2.3 Estensione di campo

L'estensione di campo è l'operazione con la quale si intende rappresentare un numero naturale usando un numero di cifre maggiore. Quando si vuol scrivere il numero 32 su 4 cifre, si mettono **due zeri in testa**. La stessa cosa si fa con i numeri **naturali** (attenzione: con gli interi sarà diverso), in qualunque base.

Dato  $X \equiv (x_{n-1}x_{n-2}\dots x_0)_\beta$ , definisco  $X^{EST}$  come il numero che vale quanto  $X$  ma è rappresentato su  $n+1$  cifre. L'unica possibilità è che  $X^{EST} \equiv (0x_{n-1}x_{n-2}\dots x_0)_\beta$ . O meglio, questa è **una** possibilità, ma visto che la rappresentazione è **unica**, è anche la sola.

## 2.4 Addizione

Riprendere la somma in base 10. Algoritmo imparato alle **elementari**.



$\beta = 10$ . L'algoritmo consiste in:

- sommare le cifre di pari posizione singolarmente, partendo dalla meno significativa e andando verso sinistra
- se la somma di due cifre non è rappresentabile con una sola cifra, usare il **riporto** per la coppia di cifre successive
- Il riporto vale sempre 0 o 1. Per la prima coppia di cifre (quelle meno significative), possiamo assumerlo **nullo**.

Dimostriamo adesso che l'utilizzo di questo algoritmo **non dipende dalla base di rappresentazione**, ma soltanto dal fatto che usiamo una **notazione posizionale**. Può pertanto essere usato per sommare numeri in base qualunque.

Dati  $X, Y$  in base  $\beta$  su  $n$  cifre, quindi  $0 \leq X, Y \leq \beta^n - 1$ , e dato  $C_{in}$ ,  $0 \leq C_{in} \leq 1$ , voglio calcolare il numero  $Z = X + Y + C_{in}$ . Il termine  $C_{in}$ , che a prima vista non sembra essere di una qualche utilità, gioca invece un ruolo fondamentale, in quanto consente di rendere l'operazione **modulare**.

**Su quante cifre** sta il risultato?  $0 \leq X + Y + C_{in} \leq 2\beta^n - 1 \leq \beta^{n+1} - 1$ , visto che  $\beta \geq 2$ . Quindi,  $Z$  è rappresentabile **sempre su  $n+1$  cifre**. Vediamo qualche dettaglio in più su queste  $n+1$  cifre.

Posso scrivere  $Z$  come quoziente e resto di una divisione per  $\beta^n$ :  $Z = C_{out} \cdot \beta^n + S = X + Y + C_{in}$ .

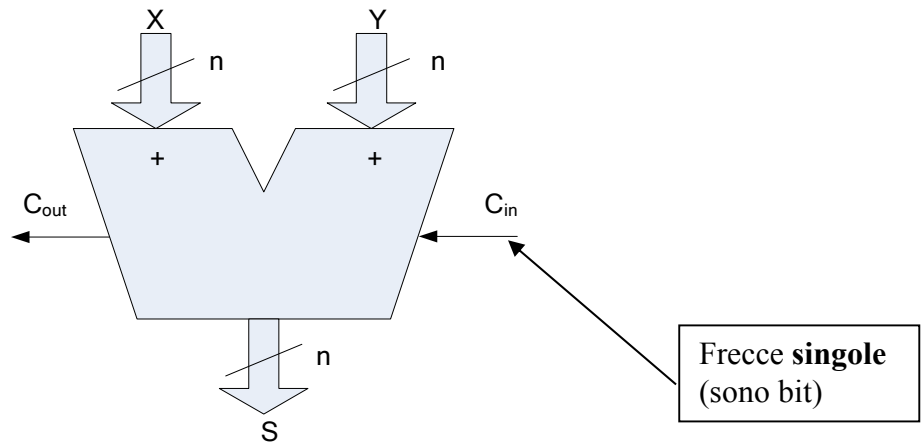
Per il teorema del resto, abbiamo:

$$C_{out} = \left\lfloor \frac{X + Y + C_{in}}{\beta^n} \right\rfloor, S = |X + Y + C_{in}|_{\beta^n}$$

Ma, visto che  $Z \leq 2\beta^n - 1$ , per l'unicità della rappresentazione deve essere  $C_{out} < 2$ . Quindi, visto che è un numero naturale, abbiamo che  $C_{out} \in \{0, 1\}$ , e questo è vero **indipendentemente dalla base**.

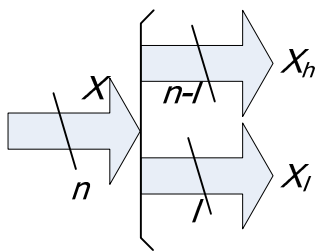
Quindi: **la somma di due numeri naturali espressi in base  $\beta$  su  $n$  cifre, più un eventuale riporto entrante che vale zero o uno, produce un numero naturale che è sempre rappresentabile con  $n+1$  cifre in base  $\beta$ , l' $(n+1)^{\text{esima}}$  delle quali, detta *riporto uscente*, può essere soltanto zero o uno.**

Posso quindi pensare di costruire un circuito che fa la somma di due numeri in base  $\beta$  su  $n$  cifre.



Questo circuito è una rete combinatoria. Ovviamente, se  $n$  è elevato, o se  $\beta > 2$ , sarà molto complessa da sintetizzare. Vediamo di **scomporre il problema della somma in sottoproblemi più semplici**.

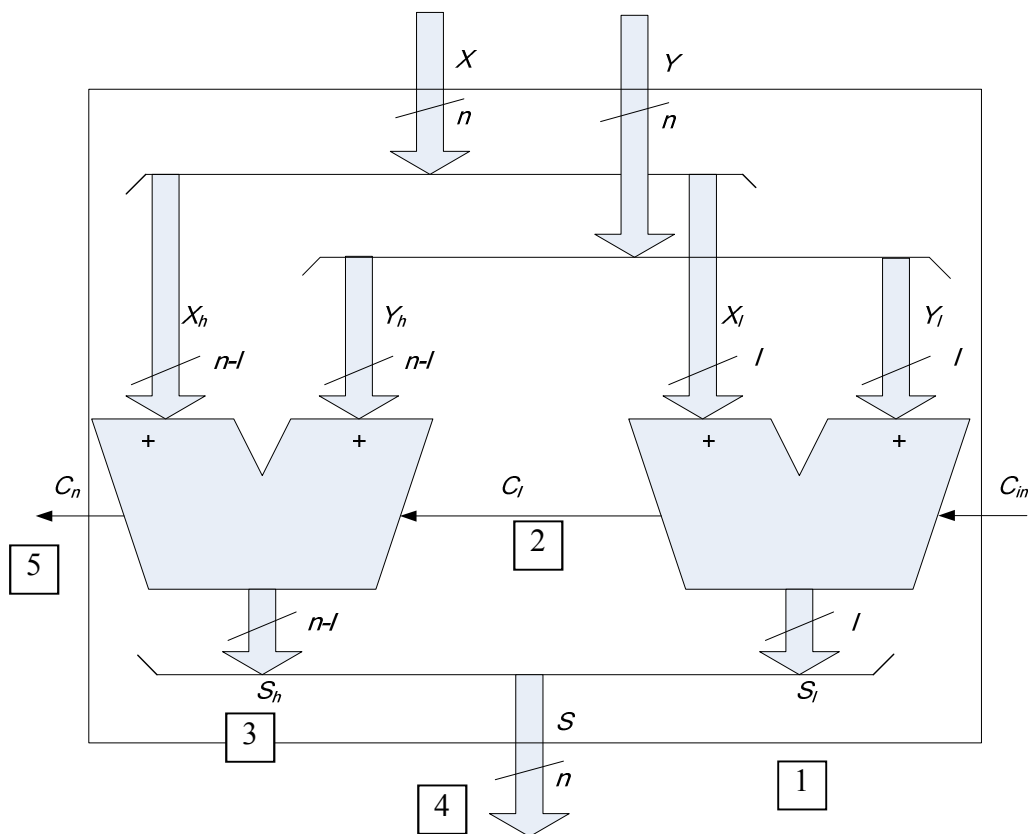
Scompongo le rappresentazioni di  $X$  ed  $Y$  in due parti distinte (operazione **a costo nullo**):



$$X = X_h \cdot \beta^l + X_l$$

$$Y = Y_h \cdot \beta^l + Y_l$$

Con  $X_l, Y_l$  su  $l$  cifre, e  $X_h, Y_h$  su  $n-l$  cifre,  $1 \leq l \leq n$ .

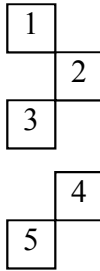


Supponiamo di **saper sommare separatamente** i blocchi di  $l$  ed  $n-l$  cifre.

$$C_h \cdot \beta^{n-l} + S_h = X_h + Y_h + C_l$$

$$C_l \cdot \beta^l + S_l = X_l + Y_l + C_{in}$$

Posso allora scrivere la somma finale come:

$  \begin{aligned}  X + Y + C_{in} &= (X_h + Y_h) \cdot \beta^l + X_l + Y_l + C_{in} \\  &= (X_h + Y_h) \cdot \beta^l + C_l \cdot \beta^l + S_l \\  &= (X_h + Y_h + C_l) \cdot \beta^l + S_l \\  &= (C_h \cdot \beta^{n-l} + S_h) \cdot \beta^l + S_l \\  &= C_h \cdot \beta^n + (S_h \cdot \beta^l + S_l) \\  &= C_{out} \cdot \beta^n + S  \end{aligned}  $	
---	---

Posso quindi **scomporre la somma** in base  $\beta$  **su  $n$  cifre** in **somme in base  $\beta$**  su un **minor numero di cifre**, purché:

- esegua le somme partendo dai gruppi di cifre meno significativi
- sia in grado di **propagare il riporto** tra un gruppo ed il successivo.

Il circuito disegnato sopra si chiama, appunto, **ripple carry** (propagazione del riporto)

Portando questo ragionamento alle sue estreme conseguenze, posso realizzare un sommatore in base  $\beta$  su  $n$  cifre utilizzando  **$n$  sommatore in base  $\beta$  ad una cifra (*full adder*)** e propagando il riporto in uscita dallo stadio  $j$ -simo in ingresso allo stadio  $j+1$ .

La presenza di un riporto uscente allo stadio  $n$ -simo va interpretata come segue:

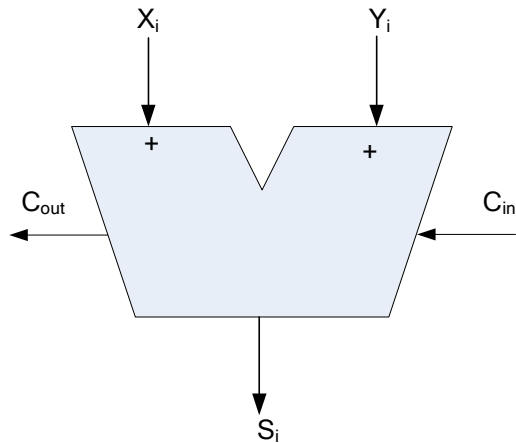
- **se il riporto è zero**, la somma è **rappresentabile su  $n$  cifre**, cioè sul numero di cifre degli operandi
- **se il riporto è uno**, la somma **non è rappresentabile su  $n$  cifre**, ma ce ne vuole una in più.

L'istruzione macchina ADD, infatti, setta il CF quando il risultato non è un numero naturale rappresentabile su  $n$  cifre.

## 2.4.1 Full Adder in base 2

Fin qui abbiamo descritto **proprietà della notazione posizionale**, valide in qualunque base, che ci consentono di scomporre una somma in somme più semplici, addirittura somme ad una cifra. Come effettivamente si realizza un full adder dipende dalla base e dalla codifica. Il **full adder in base 2** è un circuito che fa somme di una cifra in base 2, con riporto.

È una rete combinatoria con 3 ingressi e 2 uscite, e quindi la sappiamo sintetizzare.



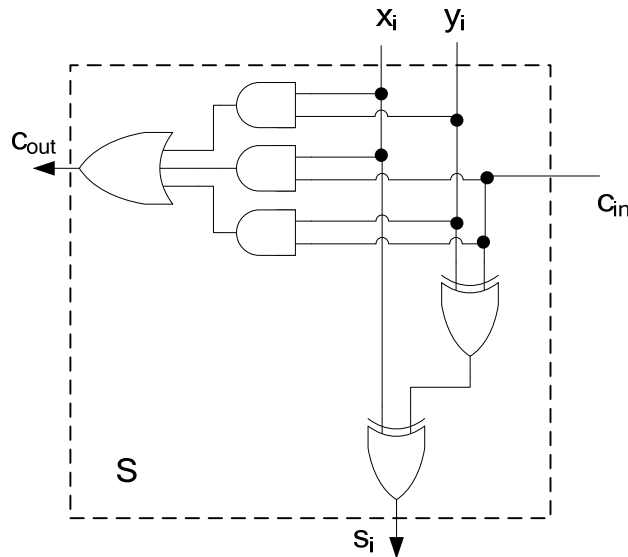
$X_i$	$Y_i$	$C_{in}$	$S_i$	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$C_{in}$	$S_i$			
	$X_i Y_i$ 00	01	11	10
0	0	1	0	1
1	1	0	1	0

$C_{in}$	$C_{out}$			
	$X_i Y_i$ 00	01	11	10
0	0	0	1	0
1	0	1	1	1

- Per quanto riguarda la produzione del **riporto uscente**, non ci sono problemi: si può fare in forma SP con 3 porte AND a 2 ingressi ed una porta OR a tre ingressi.
- Per quanto riguarda la produzione della **somma  $s_i$** , osservo che essa è ad uno se e solo se il numero di 1 in ingresso è dispari.

Per quest'ultimo, esiste una semplice realizzazione, fatta tramite porte XOR. Mettere più porte XOR in cascata (eventualmente ad albero) consente di fare circuiti che **riconoscono un numero dispari di 1**, che cioè danno 1 quando lo stato di ingresso ha un numero dispari di 1.

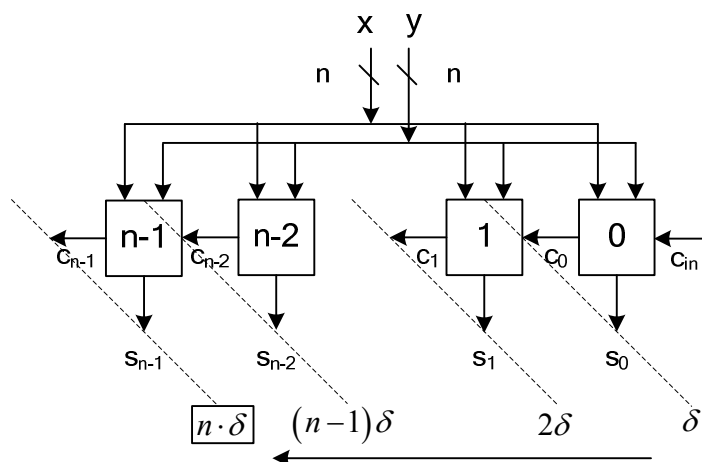


Il **full adder** è una rete a 2 livelli di logica. Detto  $\tau$  il tempo di attraversamento di un singolo livello di logica, abbiamo che ciascun full adder ha un tempo di risposta  $\delta = 2\tau$ .

## 2.4.2 Tempi di risposta e circuito di lookahead

Il problema che ci poniamo adesso è il seguente: se lavoro in base 2, posso realizzare una somma di un qualsiasi numero di cifre usando una catena di full adder, che sommano una cifra per volta. Quale sarà il **tempo di risposta** di questa rete?

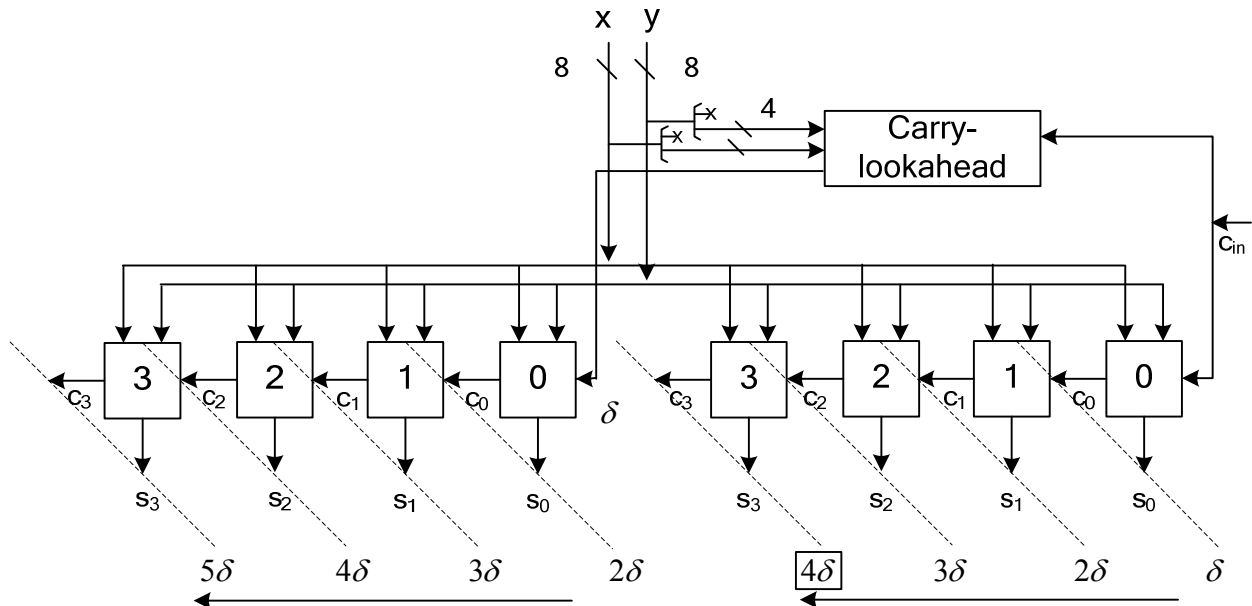
Ciascun full adder richiede un tempo  $\delta = 2\tau$  per produrre la propria **coppia di risultati**. In particolare, anche il **riporto uscente** sarà prodotto dopo un tempo  $\delta$ . Ma il riporto uscente serve **da ingresso** al full adder successivo. Quindi:



In totale, l'ultimo bit del risultato e l'ultimo riporto uscente vengono prodotti ad un istante  $n \cdot \delta$ . È lo stesso problema che si sperimenta quando si fanno le somme **a mano**. Se dobbiamo sommare numeri a 50 cifre, la 50ma somma può essere fatta **solo** quando ho il riporto della 49-ma. L'algoritmo di somma è scomponibile, ma purtroppo **non parallelizzabile**.



Si può fare qualcosa? Certo che sì. Possiamo **investire risorse** per aggiungere un circuito che, presi in ingresso i  $j$  bit meno significativi di entrambi i numeri produce il **riporto  $j$ -simo in uscita**. Questo circuito **combinatorio**, pur complesso, lo posso sintetizzare **a due livelli di logica**. In questo modo, la parte alta del circuito può **iniziare prima il proprio lavoro**, in quanto dispone del riporto entrante all'istante  $\delta = 2 \cdot \tau$  invece che  $j \cdot \delta$ ).



È ovvio che:

- dovrò sintetizzare un circuito **complesso** (che ha, in questo caso, 9 ingressi ed 1 uscita), ed in teoria **non strettamente necessario** (in quanto il carry  $j$ -simo sarebbe comunque prodotto).
- questo circuito mi consente di **anticipare** la generazione del carry per gli stadi successivi.

Ad esempio, in una somma con operandi a  $4n$  bit,

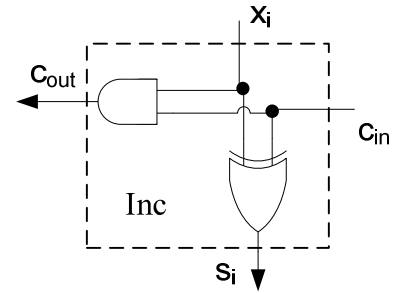
- senza lookahead il tempo di risposta è:  $4 \cdot n \cdot \delta$
- usando  $n-1$  carry lookahead a 4 bit il tempo diventa:  $(4 + (n-1)) \cdot \delta = (n+3) \cdot \delta < 4 \cdot n \cdot \delta$

Quanto mi conviene fare grande il lookahead? Ci sono ovvi limiti di fattibilità (un lookahead a 32 bit non è troppo più semplice da sintetizzare che un sommatore a 32 bit). Farlo più grande della **metà** del numero di bit non serve, perché la parte più grande del circuito fa il passo.

### 2.4.3 Incrementatore

Un incrementatore è un circuito combinatorio che somma **addendi è nullo**. Pertanto, lo possiamo realizzare scomponendo il tutto in che tale incremento sia un caso particolare di **somma con  $n$  moduli full adder**. Però questi **riporto tra due addendi ad  $n$  cifre**, in cui **uno dei due** moduli hanno un ingresso ( $y_i$ ) che è

sempre nullo. Quindi posso semplificarli.



Questo è un circuito **più semplice del full adder** (ad un solo livello di logica).

**Richiamo sull'assembler:** `ADD $1, %AL` ed `INC %AL`, pur se ottengono lo stesso risultato, sono **istruzioni diverse**. In particolare, la seconda potrebbe essere più veloce della prima (era così quando i calcolatori erano più lenti).

#### 2.4.4 Esercizio (da fare a casa)

Descrivere un incrementatore in base 7 in codifica 4-2-1. Chiamare  $z_2, z_1, z_0$  le variabili che supportano la cifra in uscita e  $c_{in}$  e  $c_{out}$  i riporti entranti ed uscenti.

Tracciare la mappa di Karnaugh della variabile  $z_2$  (**si noti** di  $z_2$ ) e:

- individuare e classificare gli impicanti principali
- trovare tutte le liste di copertura irridondanti
- scegliere la lista di costo minimo secondo il criterio a diodi
- controllare se la sintesi così ottenuta è soggetta ad alee, ed eventualmente classificarle e rimuoverle

Effettuare infine la sintesi a porte NOR di  $z_2$  (**si noti:** di  $z_2$ ).

**NB:** Al fine di rendere standard il layout delle mappe di Karnaugh, semplificando così la correzione dell'esercizio, si utilizzi  $c_{in}$  come la variabile di ingresso di ordine maggiore.

#### Soluzione

#### 2.4.5 Esercizio (da fare a casa)

- 1) *Descrivere* il circuito lookahead di un sommatore per addendi a 2 bit.
- 2) Detto  $\tau$  il tempo di attraversamento di un livello di logica (supposto costante ed identico per tutte le porte), calcolare il tempo di risposta di un sommatore per addendi a  $N=2n$  bit, che utilizzi il circuito lookahead di cui al punto 1.
- 3) *sintetizzare* il circuito a costo minimo in forma SP.

## Soluzione

### 2.4.6 Esercizio (da fare a casa)

Sia data una rete combinatoria che: i) riceve in ingresso tre variabili  $x_2, x_1, x_0$  che esprimono un numero naturale  $X$  ad una cifra in base 5 (in codifica 421) ed una *variabile di comando*  $b$ , e ii) produce in uscita tre variabili  $y_2, y_1, y_0$  che esprimono un numero naturale  $Y$  ad una cifra in base 5 ed una variabile  $c$  secondo la seguente legge.

Il numero naturale  $Y$  è legato al numero naturale  $X$  dalla relazione

$$Y = \begin{cases} \lfloor 2X \rfloor_5 & b = 0 \\ \lfloor X + 1 \rfloor_5 & b = 1 \end{cases}$$

La variabile  $c$  vale 1 se il risultato dell'operazione scritta tra  $\lfloor \cdot \rfloor$  non è rappresentabile su una cifra in base 5 e 0 altrimenti.

1) Descrivere la rete nella sua completezza riempiendo le seguenti mappe

$x_1 x_0$		$c$			
		00	01	11	10
$b x_2$	00				
	01				
	11				
	10				

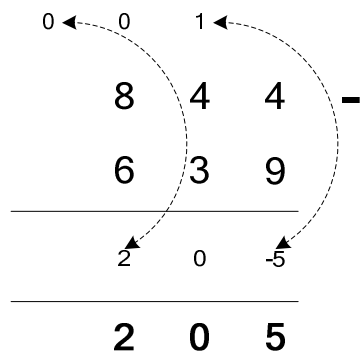
$x_1 x_0$		$y_2 y_1 y_0$			
		00	01	11	10
$b x_2$	00				
	01				
	11				
	10				

- Sintetizzare la sottorete che genera  $y_0$  a costo minimo sia a porte NAND sia a porte NOR
- Calcolare il costo delle due realizzazioni (sia a porte che a diodi), e specificare quale delle due sia di costo minore.

## Soluzione

### 2.5 Sottrazione

Riprendere la differenza in base 10. Algoritmo imparato alle **elementari**.



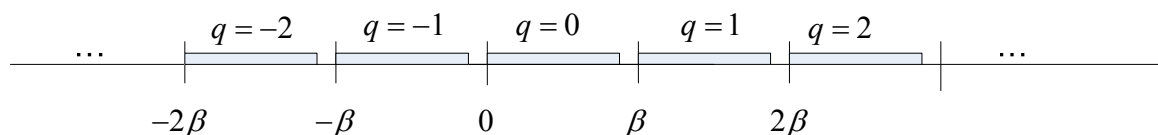
$\beta = 10$ . L'algoritmo consiste in:

- sottrarre le cifre di pari posizione singolarmente, partendo dalle meno significative
- se la differenza di due cifre non è rappresentabile con una sola cifra, usare il **prestito (borrow)** per la coppia di cifre successive
- Il prestito **vale sempre 0 o 1**. Per la prima coppia di cifre (quelle meno significative), possiamo assumerlo **nullo**.

Questo algoritmo **non dipende dalla base di rappresentazione**, ma soltanto dal fatto che usiamo una **notazione posizionale**. Può pertanto essere usato per sottrarre numeri in base qualunque.

Dati  $X, Y$  naturali in base  $\beta$  su  $n$  cifre, quindi  $0 \leq X, Y \leq \beta^n - 1$ , e dato  $b_{in}$ ,  $0 \leq b_{in} \leq 1$ , voglio calcolare il numero naturale  $Z = X - Y - b_{in}$ , **ammesso che esista**.

Sappiamo che  $-\beta^n \leq X - Y - b_{in} \leq \beta^n - 1$ . Quindi,  $Z$  può **anche non essere un numero naturale**, cosa che sappiamo bene dall'aritmetica (i naturali non sono un insieme chiuso rispetto alla sottrazione). Comunque,  $Z = X - Y - b_{in}$  diviso per  $\beta^n$  dà quoziente **al minimo -1**.



Pertanto definisco:

$$-b_{out} = \left\lfloor \frac{X - Y - b_{in}}{\beta^n} \right\rfloor, \quad D = |X - Y - b_{in}|_{\beta^n}$$

ed ottengo che  $b_{out} \in \{0, 1\}$ , ancora una volta **indipendentemente dalla base**. Quindi, posso scrivere

$$\boxed{Z = -b_{out} \cdot \beta^n + D = X - Y - b_{in}}$$

Quindi: **la differenza di due numeri naturali in base  $\beta$  su  $n$  cifre, meno un eventuale prestito entrante, produce un numero che, se naturale, è sempre rappresentabile su  $n$  cifre in base  $\beta$ . Può inoltre produrre un numero non naturale, nel qual caso c'è un prestito uscente. In ogni caso il prestito uscente può valere soltanto zero o uno.**

Per calcolare la differenza di due numeri, faccio quanto segue:

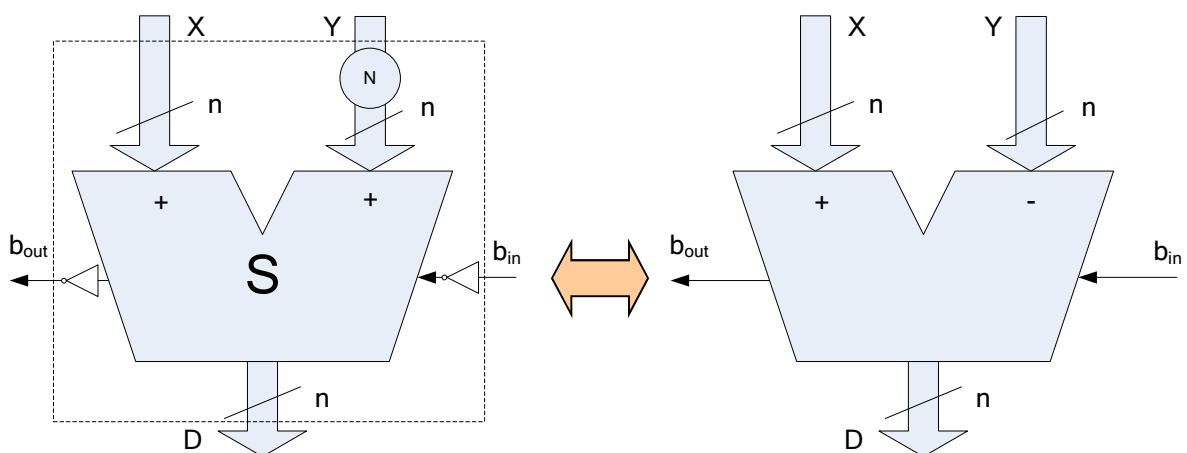
osservo che  $Y + \bar{Y} = \beta^n - 1$  (definizione di **complemento**). Dal che derivo che  $-Y = \bar{Y} - \beta^n + 1$ . Sostituendo quest'ultima nell'espressione riquadrata sopra, si ottiene:

$$(1 - b_{out}) \cdot \beta^n + D = X + \bar{Y} + (1 - b_{in})$$

Come si interpreta questa equazione? Dicendo che:

- la **differenza fra X ed Y**, meno un eventuale **prestito entrante**, qualora essa sia un numero naturale, può essere ottenuta se **sommo X ed Y complementato**, più un eventuale **riporto entrante**, ottenuto complementando il prestito entrante.
- Se il **riporto uscente** di detta somma è pari ad 1, la differenza è un **numero naturale pari a D**, ed il **prestito uscente**, ottenuto complementando il riporto uscente della somma, è zero.
- Se il **riporto uscente** di detta somma è pari a 0, la differenza **non è un numero naturale**, ed il **prestito uscente**, ottenuto complementando il riporto uscente della somma, è uno.

Quindi, posso realizzare la differenza di due numeri con un circuito fatto così:



Con tutto quel che ne consegue, incluso:

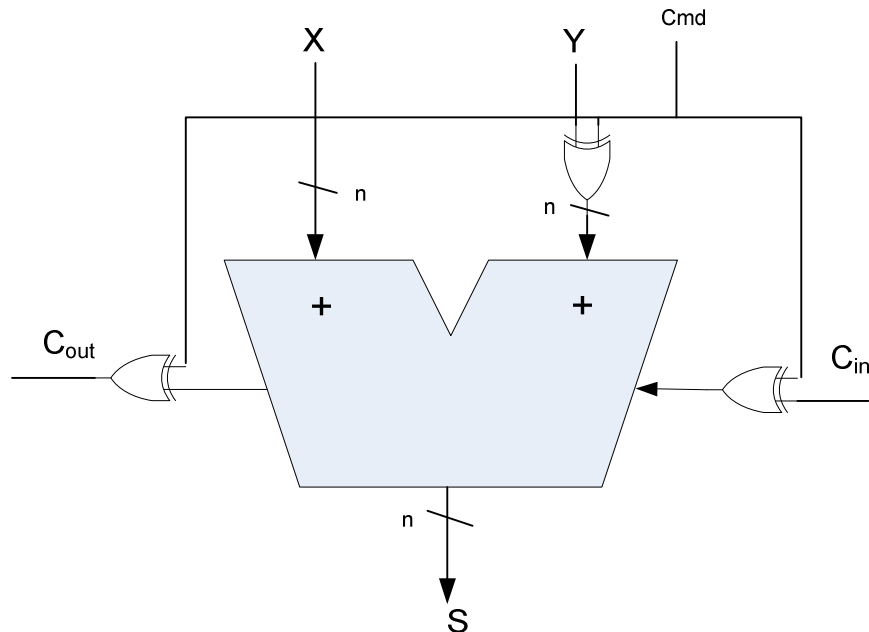
- la possibilità di scomporre il tutto in blocchi più semplici (fino ad una cifra), in quanto sia il complemento che la somma possono essere scomposti fino ad una cifra,
- la possibilità di usare full adder con lookahead,
- la possibilità di tirar fuori un circuito di decremento semplificato, etc.

## 2.5.1 Comparazione di numeri naturali

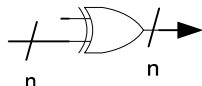
I sottrattori vengono usati spesso come **comparatori**. Dati due numeri naturali A e B, se voglio sapere se  $A < B$ , basta che sottragga  $A - B$  e guardi il prestito uscente. Se  $b_{out} = 1$ , allora A è più piccolo. Inoltre, se voglio testare se due numeri sono **uguali**, posso comunque prendere l'uscita del sottrattore e passarla ad una porta NOR ad un opportuno numero di ingressi. In quest'ultimo caso, però, conviene fare lo XOR cifra per cifra e poi passare tutto ad un NOR.

## 2.5.2 Sommatore/sottrattore in base 2

La stretta parentela tra sommatore e sottrattore può essere sfruttata per realizzare **un'unità multifunzionale** che realizzi entrambe le operazioni. Un'unità, cioè, che a seconda del valore di una **variabile di comando** esegue la somma o la sottrazione (con riporto/prestito) tra due numeri.



Per il circuito sopra disegnato, se **cmd=0** le porte XOR sono elementi neutri. Se **cmd=1** fungono invece da invertitori. Pertanto, se **cmd=1** questo circuito esegue una sottrazione.



NB: il circuito indica  $n$  porte XOR a 2 ingressi: a ciascuna viene dato in ingresso il comando cmd ed uno dei bit di  $Y$ , e le uscite vengono raccolte in un unico fascio.

Attenzione ai **dimensionamenti**: un sommatore (o un sottrattore) ha **lo stesso numero di cifre su entrambi gli ingressi e sull'uscita**. Non è un limite, in quanto l'uscita può sempre essere estesa.

## 2.6 Moltiplicazione

Anche in questo caso ripartiamo dall'algoritmo imparato alle elementari, detto di **shift e somma**.

	2	4	5	<b>X</b>
		3	1	
	2	4	5	
7	3	5		
<b>7</b>	<b>5</b>	<b>9</b>	<b>5</b>	

- Si moltiplica uno dei due fattori per **tutte le cifre dell'altro**, in step successivi.
- I **risultati** di ciascuno di questi prodotti parziali vengono scritti **a partire dal posto occupato dalla cifra per la quale si sta moltiplicando**
- I risultati di ciascun prodotto parziale sono **sommati (con riporto)** per ottenere il prodotto.

Possiamo inoltre osservare che:

- la cifra di posto  $i$  del prodotto,  $0 \leq i \leq n-1$ , è determinata unicamente dai prodotti parziali relativi alle cifre  $j \leq i$ . Cioè: alla fine dell' $i$ -simo prodotto parziale posso già stabilire, tramite una semplice somma, il valore della  $i$ -sima cifra del prodotto.

Come al solito, l'algoritmo e le proprietà sopra menzionate valgono indipendentemente dalla base in cui si lavora. Vediamo di impostare il problema in modo formale, e di sintetizzare la rete logica che lo risolve. Dati:

- $X, C$  numeri naturali in base  $\beta$  su  $n$  cifre, tali quindi che  $0 \leq X \leq \beta^n - 1$ ,  $0 \leq C \leq \beta^n - 1$
- $Y$  numero naturale in base  $\beta$  su  $m$  cifre, tali quindi che  $0 \leq Y \leq \beta^m - 1$

Voglio calcolare:

$$P = X \cdot Y + C$$

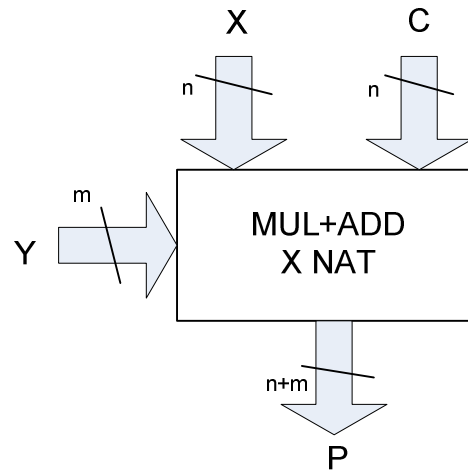
Ormai è chiaro il perché **introduco un termine in più** nell'operazione: così come una somma (sottrazione) con riporto (prestito) entrante su  $n$  cifre può facilmente essere scomposta in somme (sottrazioni) su un numero minore di cifre, allo stesso modo introdurre il termine  $C$  a sommare mi servirà per scomporre un prodotto in più prodotti più semplici.

Osserviamo che, dalle due ipotesi sopra menzionate, discende che:

$$P = X \cdot Y + C \leq (\beta^n - 1) \cdot (\beta^m - 1) + (\beta^n - 1) = \beta^m \cdot (\beta^n - 1) < \beta^{n+m} - 1$$

Il che implica che il prodotto è rappresentabile su  **$n+m$  cifre**.

Ciò detto, possiamo disegnare lo schema della rete che svolge l'operazione appena vista:



La rete sopra disegnata si chiama **moltiplicatore con addizionatore per naturali**. Vediamo come si realizza in termini di reti più semplici. Usiamo la consueta **tecnica di scomposizione del problema**.

- 1) Si scompone  $Y$  (attenzione, soltanto  $Y$ , non  $X$ ) in due blocchi:  $Y = Y_h \cdot \beta^l + Y_l$ , con  $Y_h, Y_l$  rispettivamente quoziente e resto della divisione per  $\beta^l$ .
- 2) Posso allora scrivere:

$$P = (X \cdot Y_l + C) + X \cdot Y_h \cdot \beta^l = P_l + X \cdot Y_h \cdot \beta^l$$

Il primo termine è un **prodotto su  $n$  per  $l$  cifre**, che posso svolgere con un circuito analogo a quello disegnato sopra, ma **più semplice** (con un numero inferiore di ingressi).

- 3) Il risultato di questo primo prodotto sarà un numero che posso **scomporre (a costo nullo) in quoziente e resto della divisione per  $\beta^l$** .

$$P = |P_l|_{\beta^l} + \left\lfloor \frac{P_l}{\beta^l} \right\rfloor \cdot \beta^l + X \cdot Y_h \cdot \beta^l = |P_l|_{\beta^l} + \left( \left\lfloor \frac{P_l}{\beta^l} \right\rfloor + X \cdot Y_h \right) \cdot \beta^l$$

- 4) Ma adesso, il secondo termine è un numero su  $n$  cifre, e quindi l'ultimo termine tra parentesi è quello che produce in uscita un **moltiplicatore con addizionatore a  $n$  per  $(m-l)$  cifre**.
- 5) Posso **sommare** i due prodotti parziali così ottenuti per formare il prodotto finale. Attenzione, però, che la **somma** è fatta su **intervalli di cifre disgiunti**. L'ultimo addendo, infatti, ha  $\beta^l$  a moltiplicare, e quindi ha le ultime  $l$  cifre nulle. Il primo addendo, invece è su  $l$  cifre.

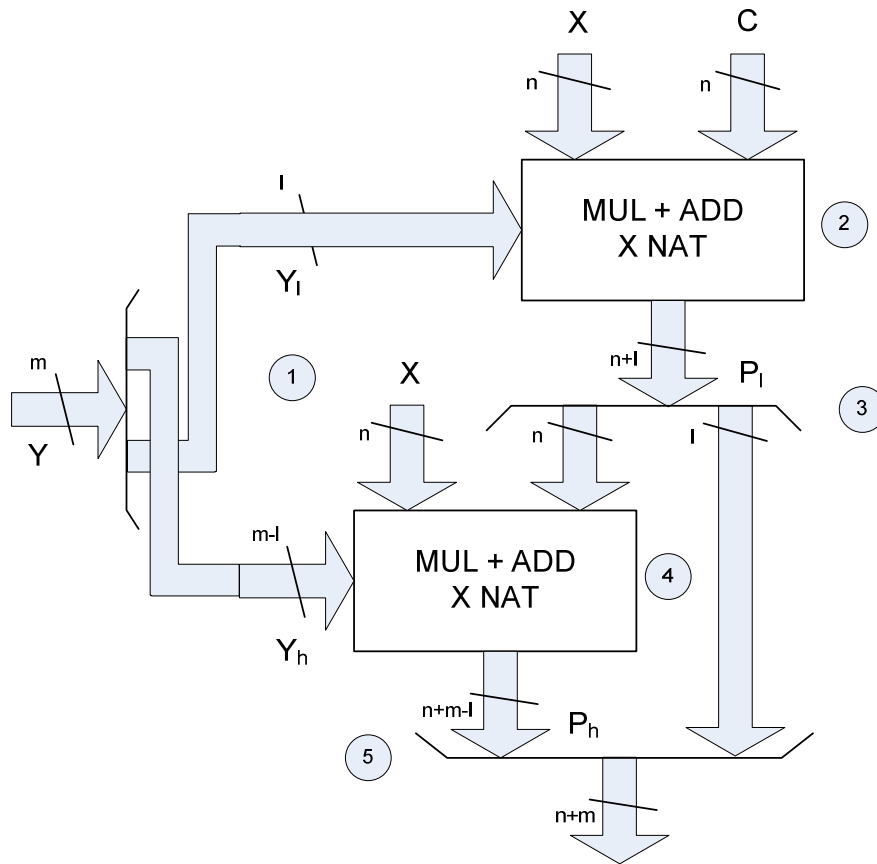
Quindi, non si tratta di somma ma di **concatenamento** dei due prodotti parziali.

In sostanza, ho calcolato  $P$ , prodotto di naturali su  $n$  ed  $m$  cifre, utilizzando

- un moltiplicatore con addizionatore a  **$n$  per  $l$  cifre**
- un moltiplicatore con addizionatore a  **$n$  per  $m-l$  cifre**
- concatenamenti e scomposizioni, quindi operazioni di costo nullo



Ovviamente, quello che faccio per via algebrica lo posso fare anche per via circuitale:

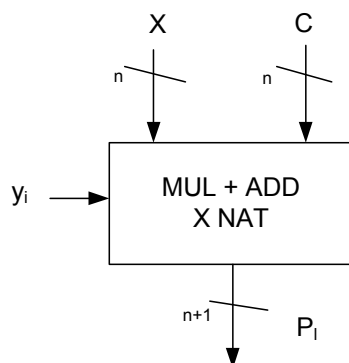


Posso iterare questa scomposizione **tante volte quante sono le cifre di  $Y$** . Quindi, posso realizzare la moltiplicazione utilizzando soltanto **moltiplicatori (con addizionatore) ad  $n$  per una cifra**.

### 2.6.1 Moltiplicatore con addizionatore $n \times 1$ in base 2

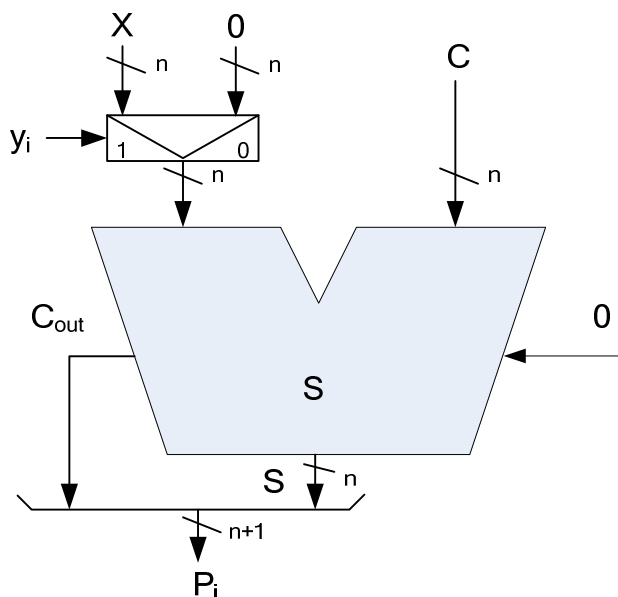
Vediamo come si sintetizza il moltiplicatore con addizionatore ad  $n$  per una cifra in base 2.

Il risultato che deve uscire da qui è:



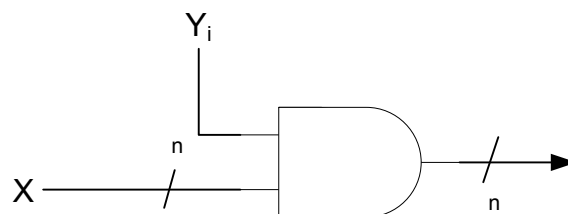
$$P_i = y_i \cdot X + C = \begin{cases} C & y_i = 0 \\ X + C & y_i = 1 \end{cases}$$

Quindi la sintesi di questo circuito è particolarmente semplice: è un circuito che, sulla base del valore di  $y_i$ , **deve sommare a  $C$  o  $X$  oppure zero**.



E' una rete che fa passare o 0 o un ingresso, a seconda che  $Y_i$  sia 0 o 1. Quindi è banalmente una porta AND

Il multiplexer a sinistra, in realtà, rappresenta  **$n$  multiplexer 2 to 1** in parallelo, ciascuno dei quali è relativo alla coppia corrispondente di bit. Vediamo più in dettaglio come è fatto ciascuno di questi multiplexer.



È una semplice barriera di porte **AND**.

## 2.6.2 Algoritmi per il calcolo iterativo del prodotto

Il prodotto di due numeri si presta, come abbiamo già intravisto, ad essere calcolato in maniera iterativa.

- Si possono scrivere **sottoprogrammi assembler** che eseguono questo compito
- Si possono dare **implementazioni microprogrammate** di reti che svolgono questo compito

Esistono due diversi algoritmi che calcolano il prodotto in modo iterativo:

### Algoritmo di shift e somma

Voglio calcolare:

$$P = X \cdot Y + C$$

- $X, C$  numeri naturali in base  $\beta$  su  $n$  cifre, tali quindi che  $0 \leq X, C \leq \beta^n - 1$
- $Y$  numero naturale in base  $\beta$  su  $m$  cifre, tale quindi che  $0 \leq Y \leq \beta^m - 1$

Pongo:

$$\begin{aligned} P_0 &= C \\ P_{i+1} &= y_i \cdot \beta^i \cdot X + P_i \end{aligned}$$

Questa è la formalizzazione dell'algoritmo che usiamo di solito per fare la moltiplicazione "a mano"

In questo modo, si dimostra banalmente che si ottiene  $P_m = P$ . Infatti

$$P_m = y_{m-1} \cdot \beta^{m-1} \cdot X + \dots + y_0 \cdot \beta^0 \cdot X + C$$

$$= X \cdot \sum_{i=0}^{m-1} y_i \cdot \beta^i + C = X \cdot Y + C$$

- un sottoprogramma assembler che realizzi questa moltiplicazione è ragionevolmente semplice
- una **rete sequenziale** che realizzi questa operazione è piuttosto complessa.

### Algoritmo di somma e shift

Sempre nelle stesse ipotesi, definisco

$$P'_i \triangleq \beta^{m-i} \cdot P_i$$

Quindi,  $P'_0 = C \cdot \beta^m$ , e  $P'_m = \beta^{m-m} \cdot P_m = P_m$ , cioè l'algoritmo dà lo stesso risultato dopo  $m$  passi.

Moltiplicando entrambi i membri della relazione di ricorrenza dell'algoritmo precedente per  $\beta^{m-i}$ , ottengo:

$$P_{i+1} \cdot \beta^{m-i} = y_i \cdot \beta^i \cdot X \cdot \beta^{m-i} + P_i \cdot \beta^{m-i}$$

$$P_{i+1} \cdot \beta = y_i \cdot \beta^m \cdot X + P'_i$$

$$P_{i+1}' = \left\lfloor \frac{y_i \cdot \beta^m \cdot X + P'_i}{\beta} \right\rfloor$$

L'algoritmo di *somma e shift* è pertanto il seguente:

$$P'_0 = C \cdot \beta^m$$

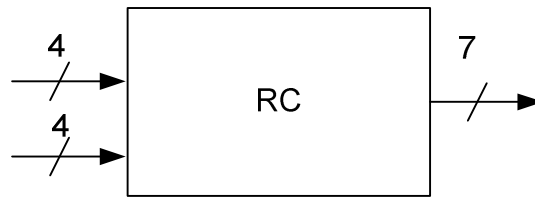
$$P_{i+1}' = \left\lfloor \frac{y_i \cdot \beta^m \cdot X + P'_i}{\beta} \right\rfloor$$

- un sottoprogramma assembler che realizzi questa moltiplicazione è ragionevolmente semplice
- una **rete sequenziale** che realizzi questa operazione è **meno** complessa della precedente. Infatti, nella prima versione (shift e somma) è richiesto ad ogni passo una moltiplicazione per  $\beta^i$ , cioè uno **shift di un numero di posizioni variabile**. Nel secondo caso, è richiesta una moltiplicazione per  $\beta^m$ , cioè uno **shift di un numero di posizioni fisso**.

### 2.6.3 Esercizio

Sintetizzare una rete combinatoria che, ricevendo in ingresso un numero naturale in base 10 a due cifre, generi in uscita il corrispondente numero binario su ? bit. Si supponga che le due cifre decimali siano codificate 8421 (BCD).

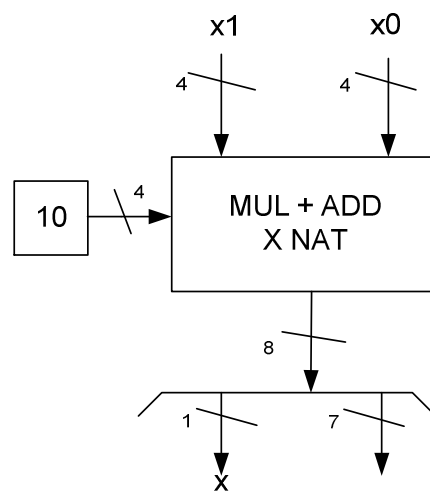
Devo realizzare un circuito fatto in questo modo



Su quante cifre sta il risultato? Visto che  $z \leq 99$ , bastano 7 bit. Le due cifre di ingresso sono  $x_1$  e  $x_0$ , e sono codificate BCD. Pertanto la loro rappresentazione (come singole cifre) è coerente con **la rappresentazione di un numero naturale in base 2 a 4 cifre**. Quindi, il risultato da calcolare è:

$$Y = 10 \cdot x_1 + x_0$$

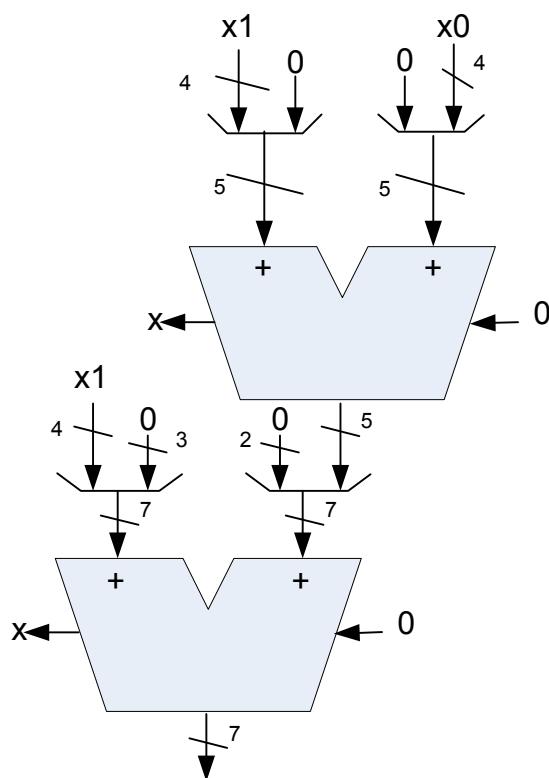
Ma un circuito che faccia questa operazione lo so sintetizzare:



Volendo, esiste un modo più furbo per fare la stessa cosa. Basta osservare che:

$$Y = 10 \cdot x_1 + x_0 = 8 \cdot x_1 + 2 \cdot x_1 + x_0$$

Allora si tratta di fare 2 somme più due shift, che sono a costo nullo.



- 1) calcolo  $2x_1 + x_0$ . Devo dimensionare correttamente l'uscita, che vale al massimo  $2 \cdot 9 + 9 = 27$ . Mi servono 5 bit in uscita.
- 2) Dimensiono l'ingresso su 5 bit di conseguenza
- 3) Calcolo adesso  $8x_1 + (2x_1 + x_0)$ , che sta su 7 bit perché è minore di 99. Devo dimensionare gli ingressi opportunamente.

Si tenga presente che il n. di bit in ingresso ad un addizionatore deve essere **identico su entrambi gli ingressi**. Altrimenti è errore (grave).

In questo modo servono 2 full adder (a 7 e 5 bit).

## 2.7 Divisione

Dati:

- $X$  numero naturale in base  $\beta$  **su  $n+m$  cifre (dividendo)**, tale che  $0 \leq X \leq \beta^{m+n} - 1$
- $Y$  numero naturale in base  $\beta$  **su  $m$  cifre (divisore)**, tale che  $0 \leq Y \leq \beta^m - 1$

Voglio calcolare i due numeri  $Q$  ed  $R$  tali che:

$$X = Q \cdot Y + R.$$

$Q$  ed  $R$  sono il quoziente ed il resto, e sono unici per il teorema della divisione con resto. **Su quante cifre dovranno essere rappresentati  $Q$  ed  $R$ ?**

- Il **resto**, dovendo essere minore del divisore, sta sicuramente **su  $m$  cifre**.
- Per il **quoziente** non posso dire molto. Infatti, se  $Y=1$ , allora  $Q=X$ , e quindi alla peggio **sta su  $n+m$  cifre**.

**Voglio** rappresentare il quoziente **su  $n$  cifre**. Assumere che  $Q$  stia su  $n$  cifre implica che:

$$X = Q \cdot Y + R \leq (\beta^n - 1) \cdot Y + (Y - 1) = \beta^n \cdot Y - 1$$

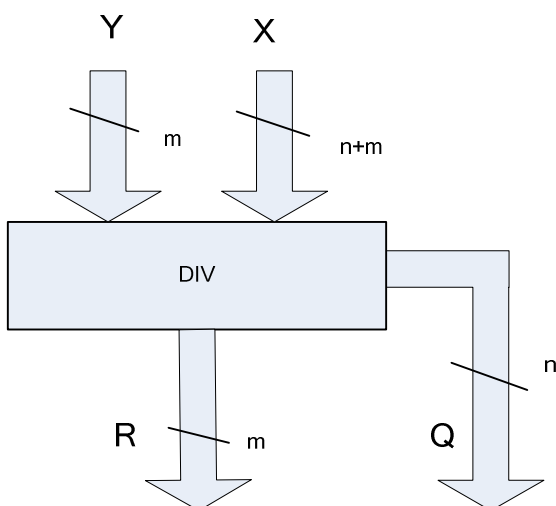
Quindi, l'**ipotesi aggiuntiva** che mi garantisce che il **quoziente stia su  $n$  cifre** è:  $X < \beta^n \cdot Y$ .

Sotto quest'ipotesi non posso fare **tutte le divisioni**, ma **soltanto alcune**. Quest'ipotesi è **restrittiva**? Dipende. Se il numero delle cifre  $n, m$  non è un dato del problema (cioè non è fissato a priori), dati  $X$  ed  $Y$  posso **sempre** trovare  $n$  tale che quella disuguaglianza sia vera (il che vuol dire che posso sempre fare la divisione, purché sia in grado di **estendere** la rappresentazione del dividendo ed abbia un numero sufficiente di cifre per il quoziente). Il problema sussiste se il **numero di cifre  $n, m$  è un dato del problema**. Questo accade, ovviamente, quando si lavora su **campi finiti**, cioè sempre all'interno di un calcolatore.

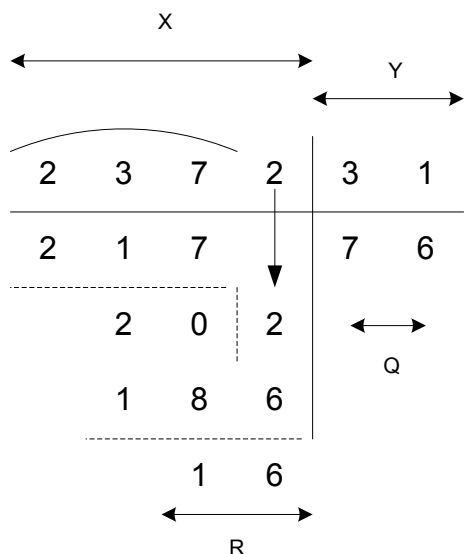
**Piccola divagazione sull'Assembler:**

La DIV ammette **dividendo su  $2n$  bit** e **divisore su  $n$  bit**, con  $n=8,16,32$ , e richiede che il **quoziente** stia su  **$n$  bit** (altrimenti genera un'interruzione). Nello schema di sopra, è quello che si otterrebbe ponendo  $n=m$ . Il dividendo è selezionato implicitamente sulla base della lunghezza del divisore. In questo caso, è **cura del programmatore** assicurarsi che  $X < \beta^n \cdot Y$ , eventualmente **estendendo la rappresentazione** del dividendo (e del divisore) su un numero maggiore (doppio) di bit. Così facendo  $X$  ed  $Y$  rimangono identici. Poter disporre di  $n=32$  (divisione con dividendo a 64 bit e divisore a 32 bit) significa poter garantire che quella disuguaglianza può essere resa vera, eventualmente estendendo le rappresentazioni, per **qualunque dividendo su 32 bit e qualunque divisore**. Qualora il dividendo **non stia** su 32 bit, non è detto che la divisione si possa sempre fare, perché non si possono estendere ulteriormente gli operandi.

Il modulo divisore deve testare la **fattibilità** della divisione con le ipotesi date. Se il quoziente non sta su  $n$  cifre, **deve settare una variabile logica  $no\_div$**  che dice che la divisione non è fattibile. Il rivelatore di fattibilità si fa con un **comparatore ad  $n+m$  cifre (sottrattore)**, che ha in ingresso  $X, \beta^n \cdot Y$  e ha come uscita  $no\_div = \overline{b_{out}}$ . D'ora in avanti non lo disegniamo per semplicità.



Posso realizzare il tutto con un circuito a 2 livelli di logica (che non sono, però, in grado di sintetizzare perché troppo complesso), o posso cercare di scomporre il tutto in moduli più semplici. La scomposizione in moduli più semplici si ottiene traducendo in forma circuitale l'algoritmo che si usa per eseguire la divisione **a mano**.



Quando faccio le divisioni, invece di considerare **tutto il dividendo** in un colpo solo:

- prendo il **minimo numero necessario** delle cifre **più significative** del dividendo in modo tale da ottenere un **numero compreso in  $[Y, \beta \cdot Y]$** . Quante sono queste cifre?  **$m$  possono non bastare;  $m+1$  cifre bastano** (purché, ovviamente, non abbia zeri in testa).
- Calcolo un **quoziente e resto parziali** della divisione così ottenuta. Sono certo che il quoziente **sta su una sola cifra** (per l'ipotesi che ho fatto).
- Calcolo un **nuovo dividendo** concatenando il resto parziale così ottenuto con la **cifra più significativa** non ancora utilizzata del dividendo. Ottengo nuovamente un dividendo parziale **certamente minore di  $\beta \cdot Y$** , date le ipotesi.
- Vado avanti fino ad esaurire le cifre del dividendo.
- Il quoziente è ottenuto dal concatenamento dei quozienti parziali (tutti su una cifra)
- Il resto è il resto dell'ultima divisione parziale.

Vediamo di scrivere la scomposizione della divisione in modo formale.

$$X = X_h \cdot \beta^l + X_l$$

$$Q = Q_h \cdot \beta^l + Q_l$$

Con  $X_l, Q_l$  su  $l$  cifre;  $X_h$  su  $n+m-l$  cifre;  $Q_h$  su  $n-l$  cifre, sotto l'ipotesi  $X < \beta^n \cdot Y$ .

Da  $X = Q \cdot Y + R$  ottengo:

$$X_h \cdot \beta^l + X_l = Q_h \cdot \beta^l \cdot Y + Q_l \cdot Y + R$$

Dalla precedente discende che:

$$\left\lfloor \frac{X_h \cdot \beta^l + X_l}{\beta^l} \right\rfloor = \left\lfloor \frac{Q_h \cdot \beta^l \cdot Y + Q_l \cdot Y + R}{\beta^l} \right\rfloor$$

Rimuovendo i multipli di  $\beta^l$  da entrambi i lati si ottiene:

$$X_h = Q_h \cdot Y + \left\lfloor \frac{Q_l \cdot Y + R}{\beta^l} \right\rfloor = Q_h \cdot Y + R'$$

Posso quindi ottenere  $Q_h$  dalla divisione di  $X_h$  per  $Y$ . Ottengo anche un resto  $R'$ . Per convincersi che  $R'$  sia un resto, basta osservare che  $Q_l \leq \beta^l - 1$  e  $R \leq Y - 1$ , quindi

$$Q_l \cdot Y + R \leq \beta^l \cdot Y - Y + Y - 1 < \beta^l \cdot Y. \text{ Ma allora } \left\lfloor \frac{Q_l \cdot Y + R}{\beta^l} \right\rfloor < Y.$$

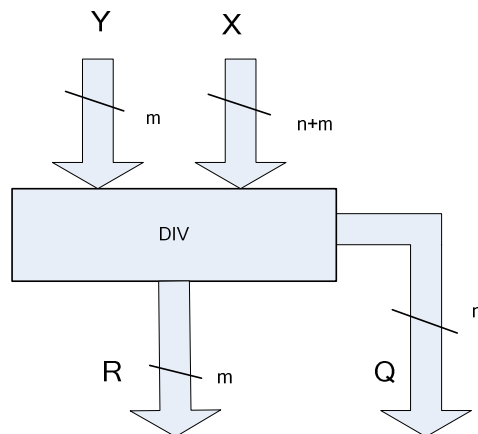
Per andare avanti, sostituisco ad  $X_h$  l'espressione trovata

$$\begin{aligned} (Q_h \cdot Y + R') \cdot \beta^l + X_l &= Q_h \cdot \beta^l \cdot Y + Q_l \cdot Y + R \\ R' \cdot \beta^l + X_l &= Q_l \cdot Y + R \end{aligned}$$

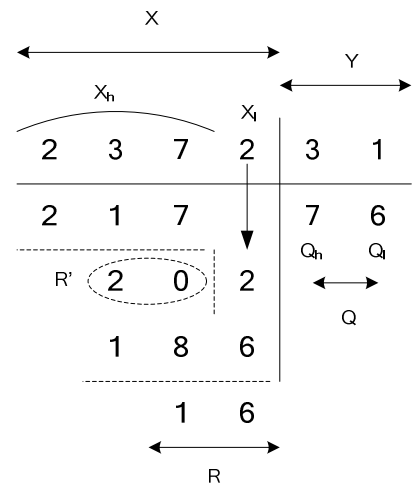
Quest'ultima operazione consiste nel

- prendere il resto della precedente divisione, shiftarlo a sinistra di  $l$  cifre e concatenarlo con il resto del dividendo
- eseguire una nuova divisione, dalla quale ottengo:
  - o le  $l$  cifre meno significative del quoziente
  - o il resto della divisione.

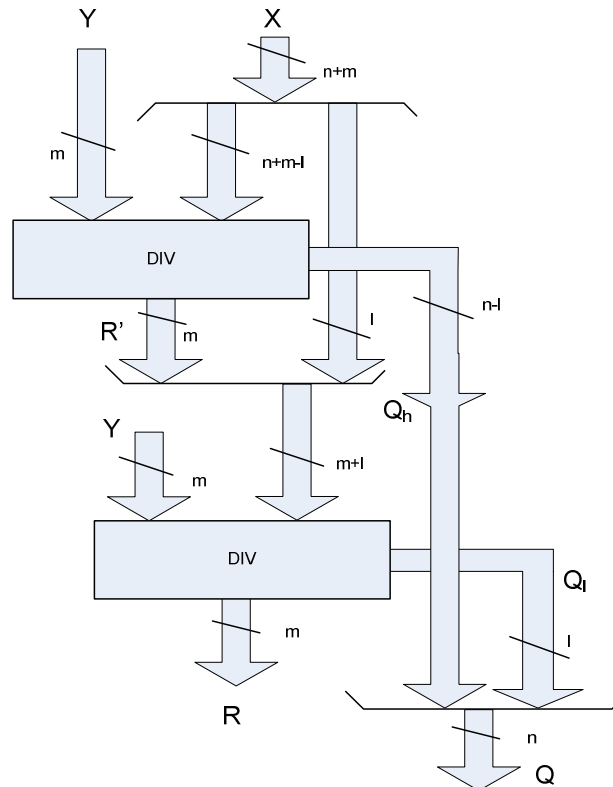
In termini circuitali:



Che voglio scomporre in moduli più semplici, secondo il procedimento appena visto.







Spingendo al massimo la scomposizione, per eseguire una divisione di un dividendo su  $n+m$  cifre per un divisore su  $m$  cifre, basta:

- saper eseguire una divisione “base” di un dividendo su  $1+m$  cifre per un divisore su  $m$  cifre
- scomporre e concatenare numeri.

**Ricapitolando:** per eseguire la divisione di un numero su  $m+n$  cifre per un numero su  $m$  cifre, uso  $n$  divisori elementari in cascata. Ciascuno di questi divisori divide un numero su  $m+1$  cifre per un numero su  $m$  cifre. Sotto l’ipotesi che  $X < \beta^n \cdot Y$ , ciascuno di questi divisori produce un quoziente che sta su una sola cifra, e quindi il quoziente finale lo ottengo come **giustapposizione** dei quozienti parziali. Tale ipotesi ( $X < \beta^n \cdot Y$ ) è **equivalente** a dire che le  **$m$  cifre più significative del dividendo rappresentano un numero più piccolo del divisore**.

### 2.7.1 Divisore elementare in base 2

Vediamo di sintetizzare l’unità in base 2 che esegue una divisione di un dividendo a  $m+1$  cifre per un divisore ad  $m$  cifre, sotto l’ipotesi che  $X < 2Y$ .

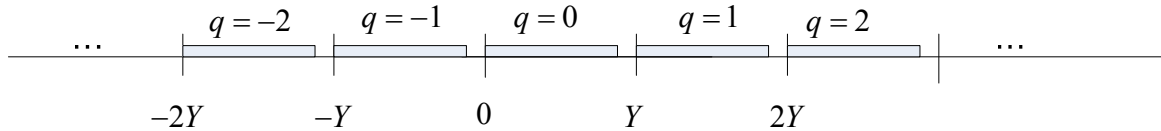
Tale unità produce

- **quoziente su una cifra**
- **resto su  $m$  cifre**

Quindi, il quoziente può valere 0 o 1. Vale 0 se il **divisore** è **maggiore del dividendo**, ed 1 altrimenti.

Il resto, invece, è uguale al **dividendo** se questo è **minore del divisore**. Altrimenti è uguale al **dividendo meno il divisore**.

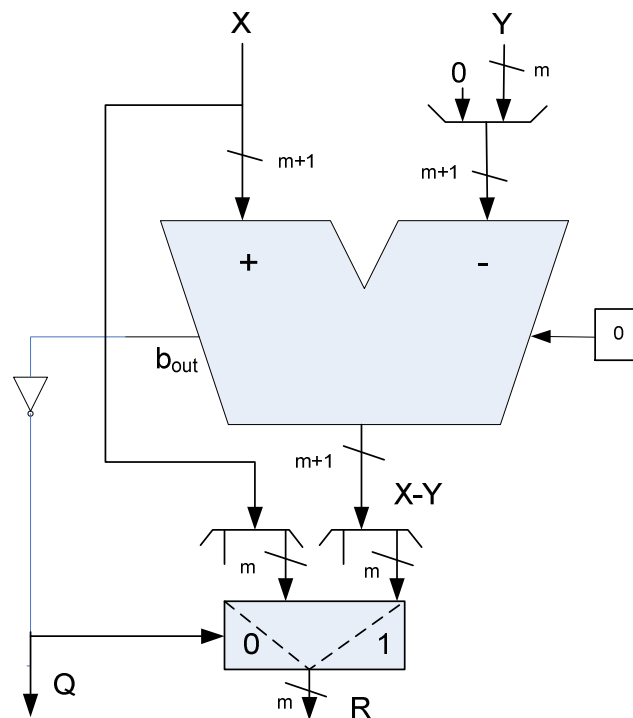
$$Q = \begin{cases} 0 & X < Y \\ 1 & X \geq Y \end{cases}, R = \begin{cases} X & X < Y \\ X - Y & X \geq Y \end{cases}$$



Quindi, tutto quello che mi serve di saper fare è:

- stabilire se il dividendo sia o meno minore del divisore
- eventualmente, fare una sottrazione.

Ma per stabilire se  $X$  sia minore di  $Y$ , basta che li **sottragga** e controlli l'eventuale **prestito uscente**.



Da dove si vede, in questo circuito, che è necessaria l'ipotesi che  $X < 2Y$ ? È l'ipotesi che mi garantisce la **riducibilità del numero  $X$  (o della differenza) su  $m$  cifre**. Infatti

- se  $X < Y$ , allora  $X$  sta su  $m$  cifre come  $Y$  ed è il resto;
- se  $X \geq Y$ , allora  $X-Y$  è il resto se e solo se  $X < 2Y$

## 2.7.2 Esercizio (da fare a casa)

Sintetizzare una rete combinatoria con quattro uscite  $z_2, z_3, z_5, z_{10}$  (ed un opportuno numero di ingressi da dettagliare), che prende in ingresso un numero naturale  $N$  a 5 cifre in base 10, codificato BCD. L'uscita  $z_k$  deve valere 1 solo quando  $N$  è divisibile per  $k$ .

Per i criteri di divisibilità si faccia riferimento alle dimostrazioni oggetto di un precedente esercizio.

### Soluzione

#### 2.7.3 Esercizio svolto

Sia dato  $X$  numero naturale rappresentato su  $n$  cifre in base 2. Senza far uso di moltiplicatori e divisori, progettare una rete combinatoria che riceve in ingresso  $X$  e produce in uscita la rappresentazione del numero naturale  $Y = \lfloor 7/8 \cdot X \rfloor$  in base 2 su ? cifre.

Su quante cifre potrò rappresentare il risultato? **Sicuramente su  $n$** , in quanto è minore di  $X$ . **Ne posso usare meno?** Basta trovare un controesempio:  $X=7$ ,  $n=3$ . Allora  $Y=6$ , che richiede comunque  $n=3$  cifre. **Ne devo usare  $n$ .**

Per ottenere il risultato, basta osservare che, se devo calcolare  $Y = 7/8 \cdot X$ , o meglio ancora  $Y = \lfloor 7/8 \cdot X \rfloor$ , posso scrivere:  $Y = \lfloor (8X - X)/8 \rfloor$ . Nell'ordine:

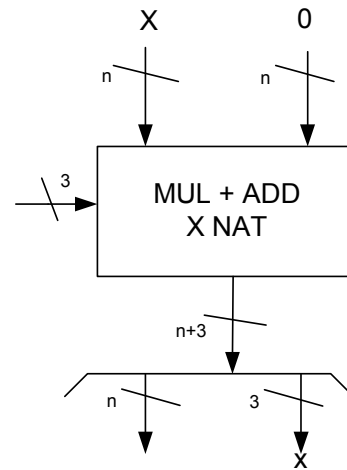
- moltiplicare  $X$  per una **potenza della base** ( $8=2^3$ ) è un'operazione che so fare, ed è a costo nullo.
- **sottrarre** due numeri naturali è un'operazione che so fare
- calcolare il **quoziente** della divisione di un naturale per una **potenza della base** è un'operazione che so fare, ed è a costo nullo.

Quindi:

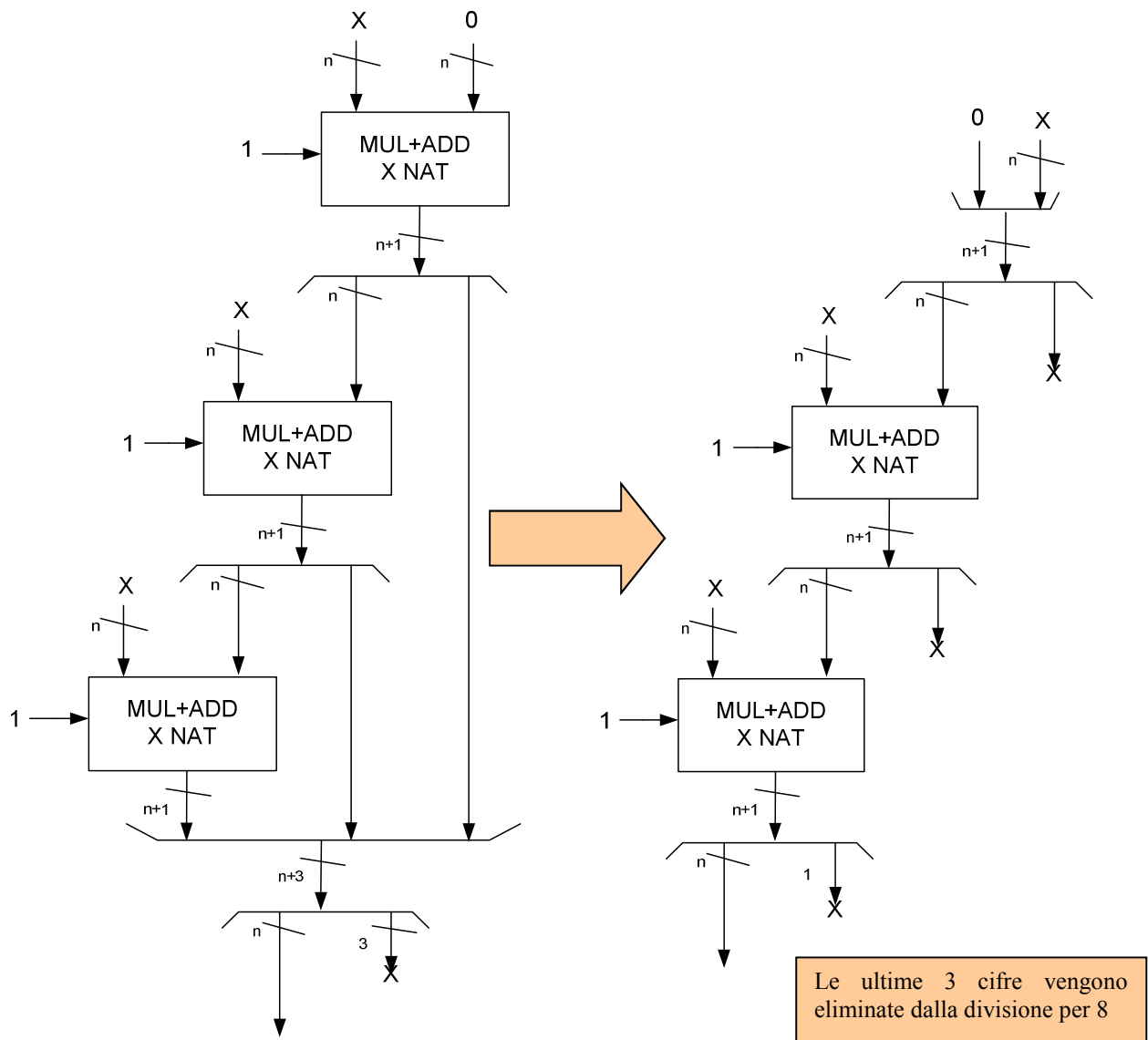


Vediamo ora di risolvere il medesimo esercizio partendo dalla realizzazione con **moltiplicatore**, anche se il testo lo vietava. Dobbiamo usare un moltiplicatore ad  $n$  per 3 cifre in base 2, e disegnare il circuito accanto.

Si può sostituire a questo circuito la sua implementazione in termini di **full adder**, e, con le opportune semplificazioni, sintetizzare un circuito che fa la stessa cosa di quello precedente.



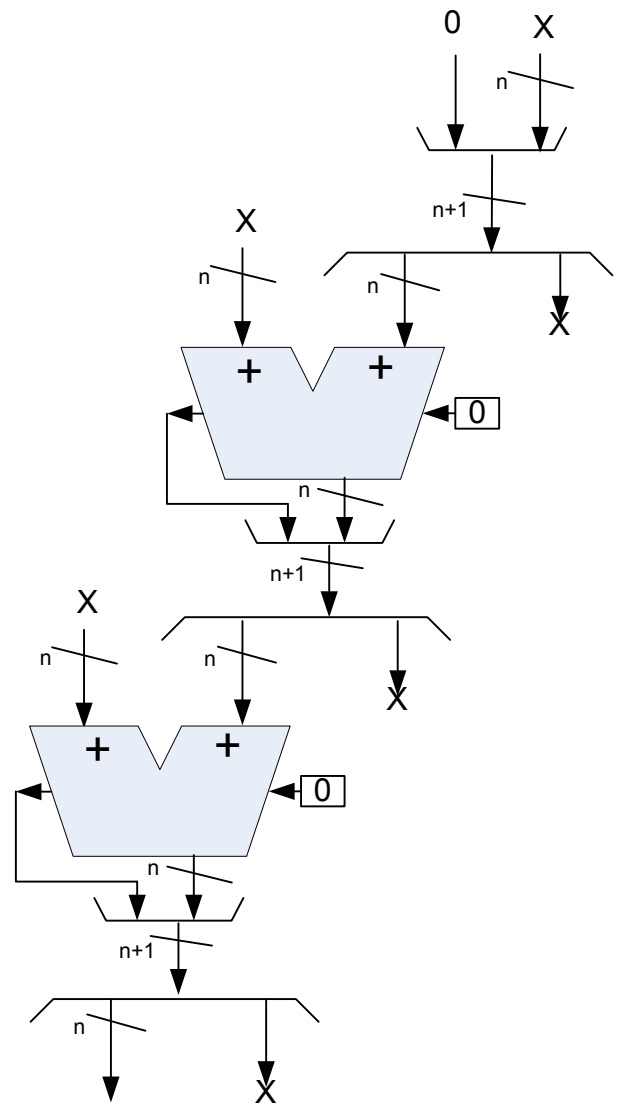
1) si scompone il moltiplicatore su 3 cifre, osservando che la rappresentazione di 7 è 111.



2) Si osserva che ciascun moltiplicatore per 1 può essere sostituito con un full adder, visto che la cifra a moltiplicare è sempre uno, e quindi le porte AND sono corto circuiti.

In questo modo, riusciamo a fare la sintesi con **due** full adder ad  $n$  bit. Nell'altro caso, ci vuole **un solo full adder**, ma ad  $n+3$  bit. Quindi, questa implementazione costa di più della precedente se  $n$  è maggiore di 3.

Si osservi che questa implementazione calcola il risultato come:  $Y = \lfloor (X + 2X + 4X)/8 \rfloor$ .

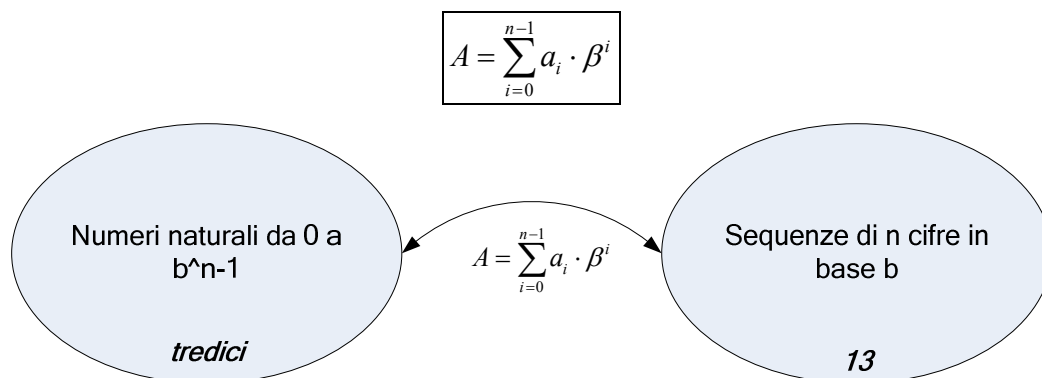


### 3 Rappresentazione dei numeri interi

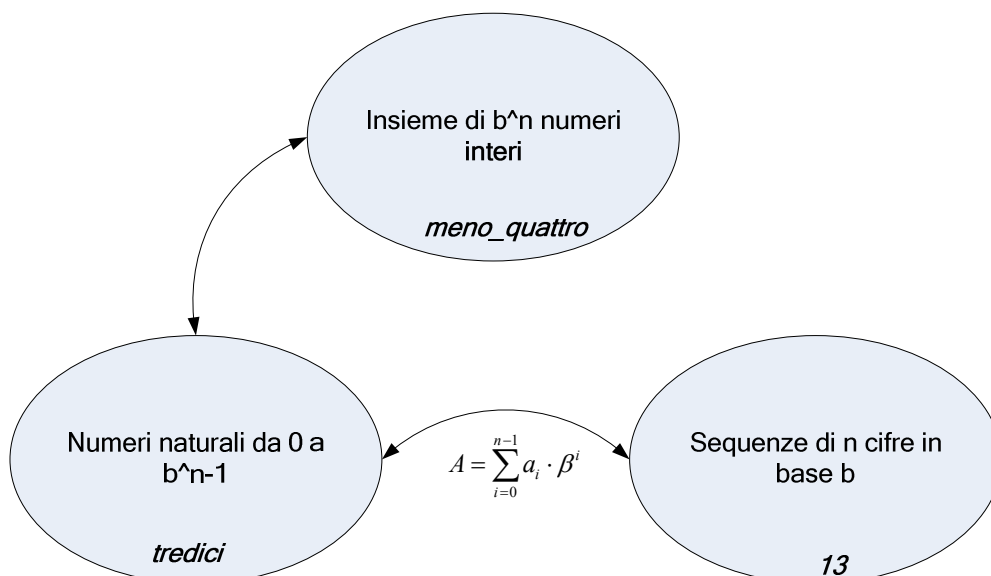
Poniamoci adesso il problema della rappresentazione dei numeri **interi**, quelli che siamo abituati a scrivere con **un simbolo, detto segno, ed un numero naturale, detto modulo**. Servono perché l'insieme dei numeri naturali non è chiuso rispetto alla sottrazione.

In realtà, all'interno dei calcolatori non conviene usare questo stile di rappresentazione: non conviene rappresentare i numeri come modulo e segno (1 bit), perché questo renderebbe le reti che operano sulle cifre più complesse. Bisognerà adottare una rappresentazione **non intuitiva**, detta rappresentazione in **complemento alla radice**.

Supponiamo di avere a disposizione **sequenze di  $n$  cifre in base  $\beta$** , quindi la possibilità di rappresentare  $\beta^n$  **combinazioni diverse**. Conosco una legge (**biunivoca**) che mi permette di associare a ciascuna di queste combinazioni un **numero naturale**, cioè:



Preso un insieme di  $\beta^n$  numeri **interi**, posso **sempre trovare una legge biunivoca** che gli fa corrispondere un insieme di  $\beta^n$  numeri **naturali**.



Procedo come segue:

- stabilisco innanzitutto di lavorare su **campi finiti**, cioè avendo a disposizione un numero **limitato** di cifre in base  $\beta$ , **noto a priori**.
- so rappresentare i numeri naturali in base  $\beta$  su  $n$  cifre
- posso associare ad ogni elemento di un insieme di numeri **interi** un elemento di un insieme di numeri **naturali**
- Quindi posso associare un insieme di  $n$  cifre in base  $\beta$  ad un numero intero. Questa sarà la **rappresentazione del numero naturale** che gli faccio corrispondere. Parlo in senso lato di **rappresentazione di un numero intero**.

Definisco una legge  $L(\ )$  da  $\mathbb{Z} \rightarrow \mathbb{N}$ , tale per cui, detto:

- $A$  un numero naturale (li scrivo **maiuscoli** d'ora in avanti)
- $a$  un numero intero (li scrivo **minuscoli** d'ora in avanti)

$$A = L(a), \quad a = L^{-1}(A)$$

Posso scrivere:

$$a \xleftrightarrow{L} A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$$

A dire che quella a destra è la **rappresentazione del numero intero  $a$  su  $n$  cifre in base  $\beta$** .

### Esempio

$\beta = 3, n = 2$ . Ho a disposizione un totale di 9 combinazioni di cifre.

$\begin{array}{c} \swarrow L() \\ \nwarrow L^{-1}() \end{array}$	$L()$	-4	-3	-2	-1	0	+1	+2	+3	+4	$a$
		0	1	2	3	4	5	6	7	8	$A$
		00	01	02	10	11	12	20	21	22	$(a_1a_0)_3$

Stabilisco una legge (che posso scegliere come mi fa più comodo) che fa corrispondere ad ogni numero intero dell'insieme che ho scelto un numero naturale, del quale so dare rappresentazione in base  $\beta$  su  $n$  cifre. Posso dire che  $+1 \leftrightarrow (12)_3$  secondo la legge che ho scelto.

È chiaro che la domanda: “di cosa è la rappresentazione la sequenza di cifre  $(12)_3$ , del numero naturale 5 o del numero intero +1?” è una domanda **priva di senso**. È come chiedere: “il contenuto di AL è un numero naturale o un carattere ASCII?” La risposta è che sono bit, e cosa ho codificato



in termini di quei bit è un pensiero contenuto nella mia testa, non nei bit stessi. Allo stesso modo, il fatto che quelle due cifre codifichino un intero o un naturale sta nella mia testa. Se ho stabilito che sto lavorando sui numeri naturali, sarà 5. Se ho stabilito che sto lavorando su numeri interi – secondo quella legge  $L()$  - sarà +1. Quindi, ciò che discrimina l'un caso dall'altro, consentendomi di dare la risposta corretta, è l'**interpretazione** che la mia testa assegna ad una sequenza di cifre.

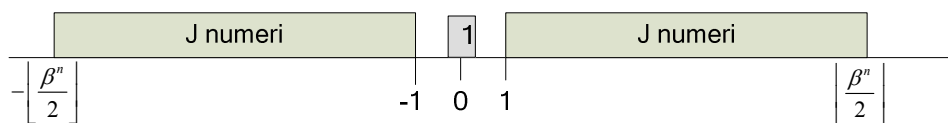
**Nota:** Perché sto facendo tutti questi conti? Per non usare un simbolo in più? Più che altro, perché se scelgo la legge di corrispondenza in maniera **furba**, posso ottenere dei vantaggi implementativi. Conosco infatti la maniera di realizzare reti che svolgono operazioni sui **numeri naturali**. Vedremo poi che, a seconda di come scelgo la corrispondenza tra interi e naturali (cioè la legge  $L()$  ), posso **riciclare** gran parte di ciò che so fare per lavorare con gli interi. Posso cioè realizzare dei circuiti che producono risultati corretti sia interpretando le cifre come rappresentazioni di naturali, sia interpretandole come rappresentazioni di interi.

Mettiamo qualche paletto in più sulla legge  $L$ :

L'insieme di numeri interi che voglio rappresentare (il dominio di  $L$ , cioè) dovrà esser necessariamente:

- contiguo (privo di buchi). Dovrà quindi essere **un intervallo**.
- Dovrà essere **il più simmetrico possibile** rispetto allo zero.

Questo perché avere un intervallo non simmetrico limita la possibilità di eseguire operazioni (ad esempio, calcolare l'opposto di un numero). Ciò pone dei **vincoli** su come scelgo il dominio della legge  $L()$  . Dato che, operando su  $n$  cifre, ho a disposizione  $\beta^n$  possibilità, logica vorrebbe che lo zero dividesse l'intervallo in due parti **identiche**.



Però  $2J+1$  numeri sono un numero *dispari di numeri*, che può essere messo in corrispondenza biunivoca con un insieme di  $\beta^n$  numeri **soltanto se  $\beta$  è dispari**, il che accade raramente (noi lavoriamo sempre con  $\beta$  pari a 2, 8, 10, 16).

Nel caso in cui  $\beta$  sia pari, dovremo accettare di rappresentare un numero positivo o negativo in più. La scelta che opereremo sarà di rappresentare sempre l'intervallo di interi

$$\left[ -\left\lfloor \frac{\beta^n}{2} \right\rfloor, \left\lceil \frac{\beta^n}{2} \right\rceil - 1 \right]$$

che, nel caso di base pari, corrisponde ad **avere un numero negativo in più**.

D'ora in avanti, per semplicità di scrittura, faremo **sempre l'ipotesi che  $\beta$  sia pari**. Ciò consente di non portarsi dietro  $\lfloor \rfloor, \lceil \rceil$  quando si fanno divisioni per due. La teoria esposta è facilmente generalizzabile al caso di base dispari, dove però la notazione è più complicata.

### 3.1 Possibili leggi di rappresentazione dei numeri interi

Ho appena definito il **dominio** della funzione  $L(\ )$ . Il **codominio** lo abbiamo individuato implicitamente, ed è l'intervallo di numeri naturali rappresentabili su  $n$  cifre in base  $\beta$ , cioè:

$$L: \left[ -\frac{\beta^n}{2}, \frac{\beta^n}{2} - 1 \right] \rightarrow [0, \beta^n - 1]$$

Resta sempre aperta la scelta di come definire la legge  $L(\ )$ . In teoria ho un numero molto elevato di possibilità. Cerco quindi di sfruttare questa libertà per fare le cose nel modo più semplice possibile. Vediamo alcune leggi usate nella pratica.

#### Traslazione

$L: A = a + \frac{\beta^n}{2}$ .  $\beta^n/2$  è detto **fattore di polarizzazione**.

Es:  $\beta = 2, n = 8$

$a$	-128	-127	-1	0	1	126	127
$A$	0	1	127	128	129	254	255
<i>rapp.</i>	00000000	00000001	01111111	10000000	10000001	11111110	11111111

In questo caso, lo zero sarà rappresentato come il numero naturale 128, cioè 10000000. Anche nell'esempio visto prima avevamo usato la stessa legge (pur senza averla definita). Ha il pregio di essere **monotona**:  $a < b \Leftrightarrow A < B$ , con il che dal confronto delle rappresentazioni si ricava un confronto tra i numeri. Viene usata: a) all'interno dei convertitori A/D e D/A, dove si chiama *binario bipolare*, e b) nel rappresentare **l'esponente dei numeri reali**<sup>1</sup>.

---

<sup>1</sup> In quest'ultimo caso si possono avere rappresentazioni in traslazione con fattori di polarizzazione diversi da  $\beta^n/2$  (di poco, in genere: al massimo  $\beta^n/2 \pm 1$ ).

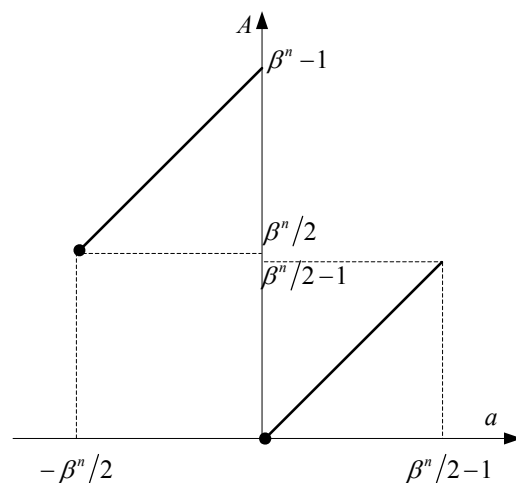
### Complemento alla radice

$$L: A = \begin{cases} a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^n + a & -\frac{\beta^n}{2} \leq a < 0 \end{cases}$$

Es:  $\beta = 2, n = 8$

$a$	-128	-127	-1	0	1	126	127
$A$	128	129	255	0	1	126	127
<i>rapp.</i>	10000000	10000001	11111111	00000000	00000001	01111110	01111111

Questo disegno ci aiuta a ricordare come è fatta la rappresentazione in complemento alla radice.



Questa è la legge usata all'interno dei calcolatori, ed è quella su cui ci focalizzeremo d'ora in avanti. È una legge **non monotona**, come si vede bene dal disegno, e quindi non posso dire che  $a < b \Leftrightarrow A < B$ .

### Modulo e segno

Tramite questa legge si fa corrispondere ad un numero intero non già un numero naturale, ma una **coppia** costituita da **un numero naturale (modulo)** e da **una variabile logica (segno)**.

$$(s, M) \leftrightarrow a$$

$$s = \begin{cases} 0 & a \geq 0 \\ 1 & a < 0 \end{cases}, M = \text{abs}(a).$$

Questo tipo di rappresentazione non ricade nella trattazione scritta prima. Infatti, non si mette in corrispondenza un intervallo di interi con un intervallo di naturali.

### 3.2 Proprietà del complemento alla radice

Il complemento alla radice gode di alcune interessanti proprietà, che saranno usate massicciamente nel seguito.

**Determinazione del segno:** posso determinare il segno di un numero semplicemente **guardandone la rappresentazione**. Dalla figura in alto si vede bene che:

$$a \geq 0 \Leftrightarrow 0 \leq A < \frac{\beta^n}{2}$$

$$a < 0 \Leftrightarrow \frac{\beta^n}{2} \leq A < \beta^n$$

Infatti, il più grande numero  $a$  rappresentabile è  $\beta^n/2 - 1$ .

Qual è la rappresentazione di questo numero su  $n$  cifre? È abbastanza facile scoprirlo se si pensa che è il numero **precedente al numero naturale**  $\beta^n/2$ . La rappresentazione di  $\beta^n/2$  è facile, perché  $\beta^n/2 = \beta^{n-1} \cdot \beta/2$ , e quindi  $\beta^n/2 \equiv (\beta/2 \ 00 \dots 0)_\beta$ . Per trovare la rappresentazione di  $\beta^n/2 - 1$  basta fare una banale sottrazione:

$$\begin{array}{ccccccc} & 1 & & 1 & & \dots & & 1 \\ \beta/2 & 0 & & \dots & & 0 & & 0 & - \\ 0 & 0 & & \dots & & 0 & & 1 & = \\ \hline \beta/2 - 1 & \beta - 1 & & \dots & & \beta - 1 & & \beta - 1 & \end{array}$$

Quindi:

$$\boxed{\beta^n/2 - 1 \equiv ((\beta/2 - 1)(\beta - 1)(\beta - 1) \dots (\beta - 1))_\beta}$$

Esempio:

$$\beta = 10, n = 4 \quad \beta^n/2 - 1 \equiv (4999)_{10} = 10^4/2 - 1$$

$$\beta = 2, n = 8 \quad \beta^n/2 - 1 \equiv (01111111)_2 = 2^8/2 - 1$$

Per capire se la rappresentazione  $A$  è un numero naturale **maggiore o minore di**  $\beta^n/2$  basta **guardare la cifra più significativa**. Infatti:

$$a_{n-1} \leq \frac{\beta}{2} - 1 \Leftrightarrow 0 \leq A < \frac{\beta^n}{2}$$

$$a_{n-1} \geq \frac{\beta}{2} \Leftrightarrow \frac{\beta^n}{2} \leq A < \beta^n$$

In **base 2**, il tutto è ovviamente più semplice:

$$\boxed{\begin{array}{ll} a \geq 0 & \Leftrightarrow a_{n-1} = 0 \\ a < 0 & \Leftrightarrow a_{n-1} = 1 \end{array}}$$

### Legge inversa

Si ottiene banalmente per sostituzione:

$$L: A = \begin{cases} a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^n + a & -\frac{\beta^n}{2} \leq a < 0 \end{cases} \Leftrightarrow L^{-1}: a = \begin{cases} A & 0 \leq A < \frac{\beta^n}{2} \\ A - \beta^n & \frac{\beta^n}{2} \leq A < \beta^n \end{cases}$$

Nella formula di destra posso:

- semplificare le condizioni a destra, che possono essere scritte guardando la cifra + significativa di  $A$
- sostituire  $A - \beta^n$  con qualcosa di più semplice, facendo leva sulla definizione di **complemento**

$$L^{-1}: a = \begin{cases} A & a_{n-1} < \frac{\beta}{2} \\ -(\bar{A} + 1) & a_{n-1} \geq \frac{\beta}{2} \end{cases}$$

Esempio:

$$\beta = 10, \quad n = 3$$

$A \equiv (852)_{10}$ . Visto che la cifra più significativa è maggiore di  $\beta/2 - 1 = 4$ , il numero rappresentato è **negativo**. Quindi, per trovarlo devo calcolare  $\bar{A} \equiv (147)_{10}$ , sommargli uno e cambiare il segno:  $a = -148$

$$A \equiv (500)_{10}. \quad a = -(499 + 1) = -500$$

$$\beta = 2, \quad n = 4$$

$A \equiv (1011)_2$ . Visto che la cifra più significativa è 1, il numero è negativo. Quindi:

$$a = -(0100 + 1)_2 = -(101)_2 = -5$$

$A \equiv (1111)_2$ . Visto che la cifra più significativa è 1, il numero è negativo. Quindi:

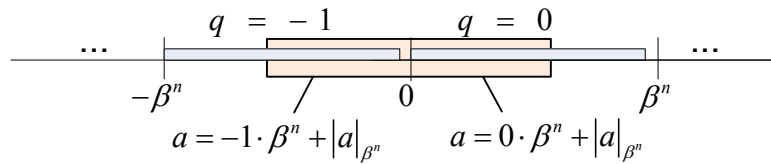
$$a = -(0000 + 1)_2 = -(1)_2 = -1$$

### Forma alternativa per L

Posso scrivere la legge  $L$  in un altro modo:

$$L: A = \begin{cases} a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^n + a & -\frac{\beta^n}{2} \leq a < 0 \end{cases} \Leftrightarrow A = |a|_{\beta^n} \quad \text{se} \quad -\frac{\beta^n}{2} \leq a < \frac{\beta^n}{2}$$

Infatti, se  $a$  è un numero positivo, allora è anche minore di  $\beta^n$ , e quindi  $a = |a|_{\beta^n}$ . Se invece è un numero negativo, allora è compreso in  $[-\beta^n/2, 0[$ , e quindi se lo divido per  $\beta^n$  ottengo quoziente -1. Cioè,  $a = -\beta^n + |a|_{\beta^n}$ , ma allora si ottiene ugualmente (ramo inferiore)  $A = |a|_{\beta^n}$ .



**Attenzione:** quanto appena scritto è vero **solo se**  $a$  è rappresentabile su  $n$  cifre in base  $\beta$  in complemento alla radice, cioè se appartiene a  $[-\beta^n/2, \beta^n/2 - 1]$ . E' facile scordarselo, e questa cosa è fonte di **errori gravi**. In particolare, se  $a$  non appartiene all'intervallo scritto sopra,  $|a|_{\beta^n}$  esiste ed è un numero su  $n$  cifre in base  $\beta$  (ovviamente), ma quel numero non ha niente a che vedere con la rappresentazione di  $a$  in complemento alla radice (che invece non esiste su  $n$  cifre in base  $\beta$ ).

### Esempio:

$$\beta = 10, \quad n = 3, \quad a = -953. \quad |a|_{\beta^n} = |-953|_{1000} = 47.$$

Ma 47 **non** è la rappresentazione di  $a$  in complemento alla radice su tre cifre, perché su tre cifre posso rappresentare soltanto i numeri nell'intervallo  $[-500, +499]$ , ed  $a$  **non appartiene** all'intervallo. Il numero  $A=47$  è la rappresentazione (in complemento alla radice, su 3 cifre in base 10) dell'intero  $a=+47$ , non dell'intero  $a=-953$ .

### 3.2.1 Esercizio (da fare a casa)

Sia  $X$  la rappresentazione in complemento alla radice su  $n$  cifre in una base generica (pari)  $\beta$  del numero intero  $x$ . Sia  $Y$  la rappresentazione *in traslazione* dello stesso numero.

- 1) esprimere la relazione algebrica che consente di trovare  $Y$  in funzione di  $X$ ;
- 2) sintetizzare a costo minimo il circuito che produce  $Y$  avendo  $X$  in ingresso nel caso  $\beta = 6$  (con codifica 421);
- 3) sintetizzare a costo minimo il circuito che produce  $Y$  avendo  $X$  in ingresso nel caso  $\beta = 16$  (con codifica 8421).

### Soluzione

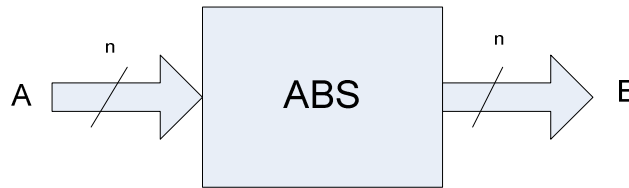
## 4 Operazioni su interi in complemento alla radice

Il motivo per cui abbiamo definito una legge di rappresentazione degli interi come numeri naturali è che vogliamo sintetizzare dei circuiti che **lavorano sulle rappresentazioni**, che cioè sono in grado di fornire il risultato corretto semplicemente lavorando sulle rappresentazioni dei numeri interi.

### 4.1 Valore assoluto

Vogliamo trovare il numero **naturale**  $B = ABS(a)$ . Visto che  $a \in [-\beta^n/2, \beta^n/2 - 1]$ , otteniamo che  $B \in [0, \beta^n/2]$ , cioè che  $B$  è **un numero naturale rappresentabile su  $n$  cifre** (attenzione:  $n-1$  non bastano, neanche se  $\beta = 2$ ).

Vogliamo quindi disegnare un circuito che, prendendo in ingresso  $A \equiv (a_{n-1}, \dots, a_0)_\beta$  produce in uscita  $B \equiv (b_{n-1}, \dots, b_0)_\beta$ , con  $a \leftrightarrow A$  e  $B = ABS(a)$ .



$$ABS(a) = \begin{cases} a & a \geq 0 \\ -a & a < 0 \end{cases}.$$

Conosco un modo semplice per stabilire il segno di  $a$  **guardandone la rappresentazione  $A$** , ed un modo semplice per calcolare  $-a$  **usando il complemento della sua rappresentazione** (far riferimento alla **legge inversa**).

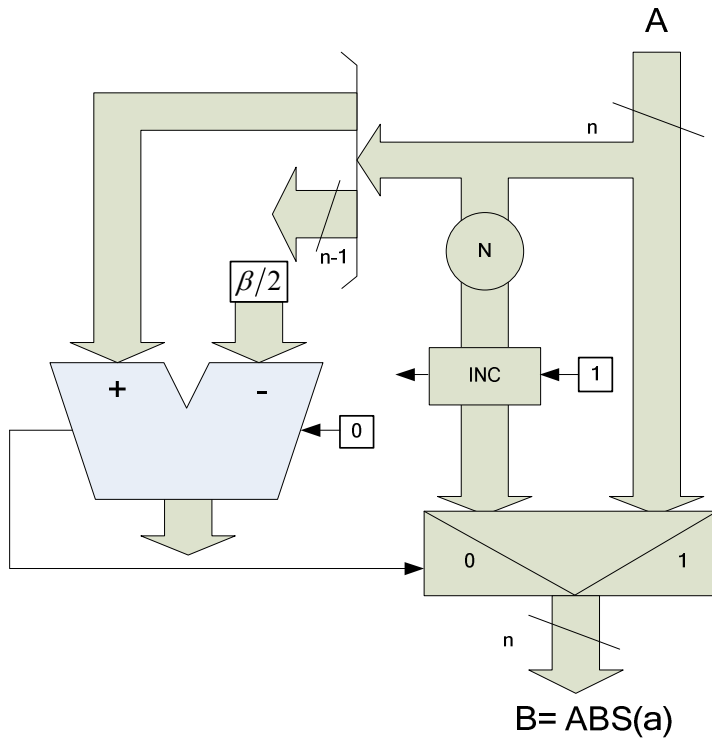
$$B = ABS(a) = \begin{cases} A & a_{n-1} < \beta/2 \\ \bar{A} + 1 & a_{n-1} \geq \beta/2 \end{cases}$$

Esempi:

$$\beta = 10, \quad n = 3: \quad A \equiv (852)_{10} \Rightarrow B = ABS(a) = 147 + 1 = 148$$

$$\beta = 2, \quad n = 4: \quad A \equiv (1011)_2 \Rightarrow B = ABS(a) = (100)_2 + (1)_2 = 5$$

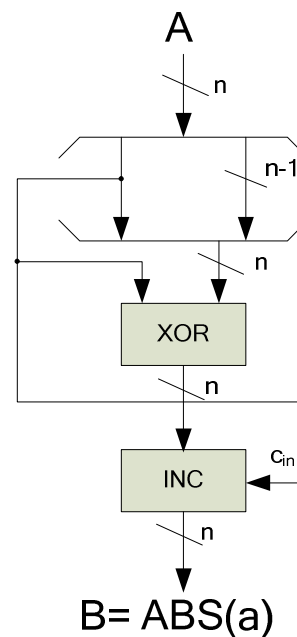
Dal punto di vista circuitale, il tutto si fa così



Per sapere se devo far passare un ingresso o l'altro del multiplexer devo confrontare la cifra più significativa di  $A$  con  $\beta/2$ . Un confronto lo posso fare con una **sottrazione**, perché  $a_{n-1}$  e  $\beta/2$  sono naturali in base  $\beta$  su una cifra, guardando poi il valore del **prestito uscente**.

In base 2, il circuito di sopra si può semplificare nella sua parte a sinistra.

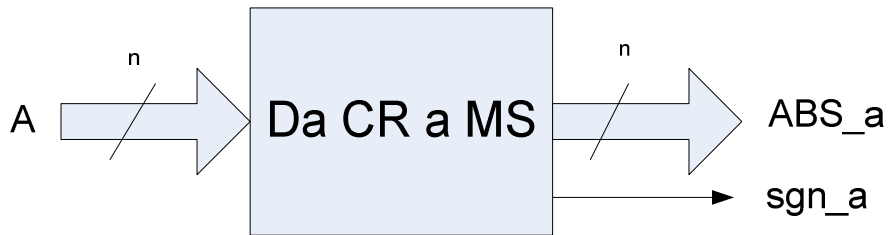
Inoltre, posso semplificarlo ulteriormente. Infatti, posso usare una barriera di XOR guidate dalla cifra più significativa di  $A$  per realizzare o meno il complemento del numero. Posso poi usare un incrementatore che, sempre guidato dalla cifra più significativa di  $A$ , incrementa o lascia invariato il numero.



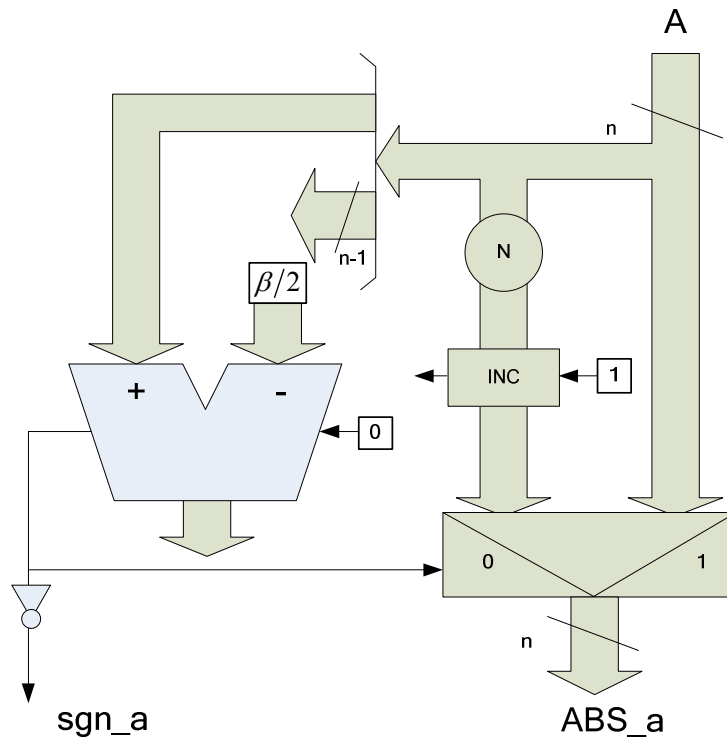
In realtà il circuito a destra può essere ulteriormente ottimizzato, contando che **l'ultima XOR dà sempre zero**.



#### 4.1.1 Circuito di conversione da CR a MS



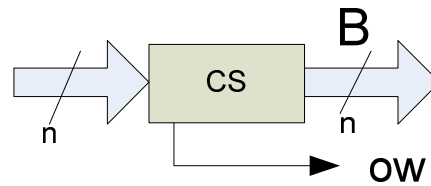
Dal precedente schema si ricava immediatamente un circuito che converte la rappresentazione  $A$  in complemento alla radice nella rappresentazione in modulo e segno dello stesso numero.



L'operazione è sempre fattibile nelle ipotesi in cui mi sono messo, visto che l'intervallo di rappresentabilità di un numero su  $n$  cifre in MS **contiene** interamente quello su  $n$  cifre in complemento alla radice.

#### 4.2 Cambiamento di segno

Dato  $A \leftrightarrow a$ , voglio trovare  $B \leftrightarrow b$  tale che  $b = -a$ , con  $B$  su  $n$  cifre. È un'operazione **non sempre possibile**. Infatti, visto che  $a \in [-\beta^n/2, \beta^n/2 - 1]$ , abbiamo a disposizione un intero negativo in più, che non ha un opposto nell'intervallo. Il circuito che dobbiamo disegnare quindi è fatto così:



Con *ow* segnale di *overflow*, che deve essere messo ad 1 se l'operazione non è possibile, a 0 altrimenti. Per adesso, **assumiamo che**  $a \neq -\beta^n/2$  inizialmente, poi **verificheremo l'ipotesi**.

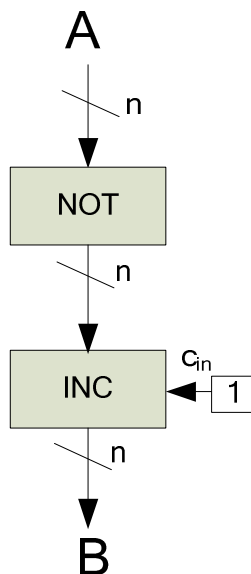
In questo caso abbiamo:

$$\begin{aligned}
 B^{(*)} &= |-a|_{\beta^n} \\
 &= \left| -1 \right|_{\beta^n} \cdot |a|_{\beta^n} \Big|_{\beta^n} = \left| (\beta^n - 1) \cdot A \right|_{\beta^n} \\
 &= \left| \beta^n \cdot A - A \right|_{\beta^n} = |-A|_{\beta^n} \\
 &= \left| -\beta^n + 1 + \bar{A} \right|_{\beta^n} = \left| 1 + \bar{A} \right|_{\beta^n}
 \end{aligned}$$

(\*) se  $a$  è l'estremo inferiore, la rappresentazione di  $-a$  non esiste.

Uso il complemento:  
 $\bar{A} = \beta^n - 1 - A$

Il circuito in base 2 è il seguente



Fare i seguenti esempi:

- $A=10000000$

In questo caso il valore in uscita non è corretto (si ottiene ancora  $B=10000000$ ). Del resto,  $A \leftrightarrow -\beta^n/2$ , e quindi l'opposto non è rappresentabile.

- $A=00000000$

In questo caso si avrebbe un riporto in uscita dall'INC.

Come si fa a vedere se c'è o meno **overflow**? Ci sono due modi:

- controllare se  $A \equiv (10...00)$ , che richiede una AND a  $n$  ingressi.
- guardare se A e B hanno entrambi **segno negativo**, cioè se  $a_{n-1} = b_{n-1} = 1$ , perché questo è il solo caso che non torna. Si può usare una **AND a due ingressi**.

**Richiamo sull'assembler**: esiste un'istruzione **NEG**, che interpreta una sequenza di bit come la rappresentazione di un numero **intero**, e produce la rappresentazione dell'opposto se questo esiste. Altrimenti setta il **flag di overflow OF**.

### 4.3 Estensione di campo

L'estensione di campo è l'operazione con la quale si intende rappresentare un numero usando un numero di cifre maggiore. Dato il naturale  $A \equiv (a_{n-1}a_{n-2}\dots a_1a_0)_\beta$  che rappresenta l'intero  $a \leftrightarrow A$  in complemento alla radice su  $n$  cifre, voglio trovare  $A^{EST} \equiv (a_n'a_{n-1}'a_{n-2}'\dots a_1'a_0')_\beta$ , che rappresenta lo stesso intero in complemento alla radice su  $n+1$  cifre. Ovviamente, **il problema ha sempre soluzione** perché l'intervallo di rappresentabilità su  $n+1$  cifre contiene quello su  $n$  cifre.

$$L_{n+1}: A^{EST} = \begin{cases} a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^{n+1} + a & -\frac{\beta^n}{2} \leq a < 0 \end{cases}, \text{ ed inoltre } L_n^{-1}: a = \begin{cases} A & a_{n-1} < \frac{\beta}{2} \\ -(\bar{A}+1) = -\beta^n + A & a_{n-1} \geq \frac{\beta}{2} \end{cases}$$

Quindi:

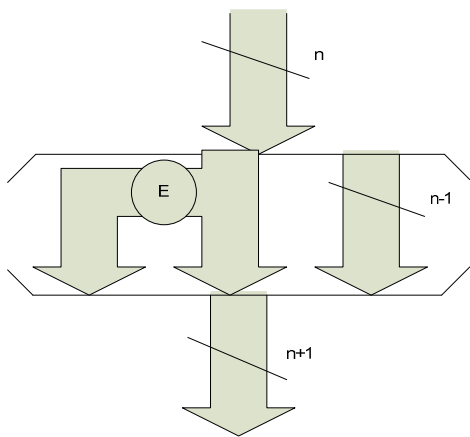
$$A^{EST} = \begin{cases} A & a_{n-1} < \frac{\beta}{2} \\ (\beta-1) \cdot \beta^n + A & a_{n-1} \geq \frac{\beta}{2} \end{cases} \quad \Rightarrow \quad A^{EST} = \begin{cases} 0 \cdot \beta^n + A & a_{n-1} < \frac{\beta}{2} \\ (\beta-1) \cdot \beta^n + A & a_{n-1} \geq \frac{\beta}{2} \end{cases}$$

Quindi, ricavo immediatamente che:

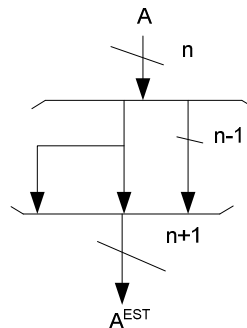
$$a_i' = a_i \quad 0 \leq i \leq n-1, \quad a_n' = \begin{cases} 0 & a_{n-1} < \beta/2 \\ \beta-1 & a_{n-1} \geq \beta/2 \end{cases}$$

Cioè, le  $n$  cifre meno significative sono identiche, la  $n+1$ -sima è **diversa** a seconda che il numero rappresentato sia positivo o negativo.

Quindi l'estensione di campo dei numeri **interi**, a differenza di quella dei **naturali**, richiede **in generale** della logica, quella necessaria a discriminare il segno del numero intero rappresentato.



In base 2 il tutto è molto più semplice:  $a_n' = a_{n-1}$



**Richiamo sull'assembler:** esistono istruzioni **CBW**, **CWD**, **CDQ**, che interpretano una sequenza di bit come la rappresentazione di un numero **intero** (su 8, 16, 32 bit), e producono la rappresenta-

zione dello stesso numero **estesa su 16, 32, 64 bit**. Per contro, non esiste nessuna istruzione dedicata allo stesso scopo per i naturali: per i naturali l'estensione si fa aggiungendo degli zeri in testa.

#### 4.3.1 Esercizio (da fare a casa)

Descrivere il circuito di estensione di campo per numeri interi rappresentati in complemento alla radice in base 10 in codifica 8421 (BCD). Sintetizzare il circuito *sia* a porte NOR *che* a porte NAND a costo minimo, svolgendo il procedimento in maniera completa (cioè classificando gli implicant, trovando tutte le liste di copertura irridondanti e scegliendo una di quelle a costo minimo). Individuare, classificare, ed eliminare eventuali alee in ciascuna sintesi.

#### Soluzione

### 4.4 Riduzione di campo

Affrontiamo adesso il problema inverso. Dato  $A$ ,  $A \equiv (a_n a_{n-1} a_{n-2} \dots a_1 a_0)_\beta$ , su  **$n+1$**  cifre, tale che  $a \leftrightarrow A$ , voglio trovare  $A^{RID} \equiv (a_{n-1}' a_{n-2}' \dots a_1' a_0')$ , su  $n$  cifre, tale che  $a \leftrightarrow A^{RID}$ .

Il problema ha soluzione **soltanto se**  $a \in [-\beta^n/2, \beta^n/2 - 1] \subset [-\beta^{n+1}/2, \beta^{n+1}/2 - 1]$ .

Voglio determinare  $A^{RID}$  **a partire dalle cifre di  $A$** . Quando  $a$  è nell'intervallo nel quale  $a$  è riducibile,  $A$  appartiene ad uno dei due intervalli marcati sull'asse delle ordinate. Uno va da 0 a  $A'$ , che mi devo calcolare; l'altro va da  $A''$  (che mi devo calcolare) a  $\beta^{n+1} - 1$ .

Vediamo quali sono le cifre di  $A$  quando  $a$  assume i **valori estremi** per cui l'operazione si può fare.

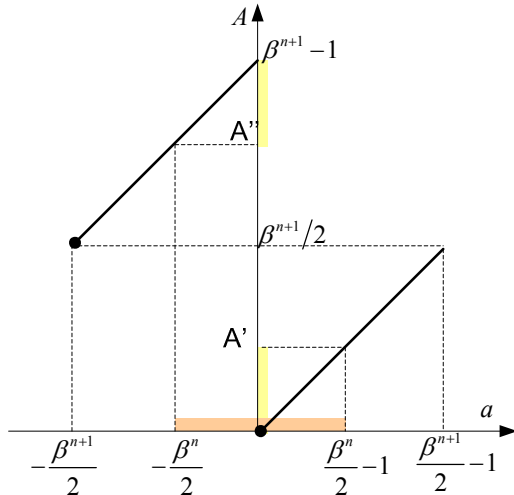
$$- a = \beta^n/2 - 1 \leftrightarrow A' \equiv (0(\beta/2 - 1)(\beta - 1)(\beta - 1) \dots (\beta - 1)(\beta - 1))_\beta,$$

Quindi, quando  $a$  è **positivo**, la **condizione per la riducibilità è che**:

$$a_n = 0 \wedge a_{n-1} < \beta/2$$

$$- a = -\beta^n/2 \leftrightarrow A'' = \beta^{n+1} - \beta^n/2 = \beta^n(\beta - 1) + \beta^n/2, \text{ quindi } A'' \equiv ((\beta - 1)(\beta/2)00 \dots 00)_\beta$$

Sommo e sottraggo  $\beta^n$

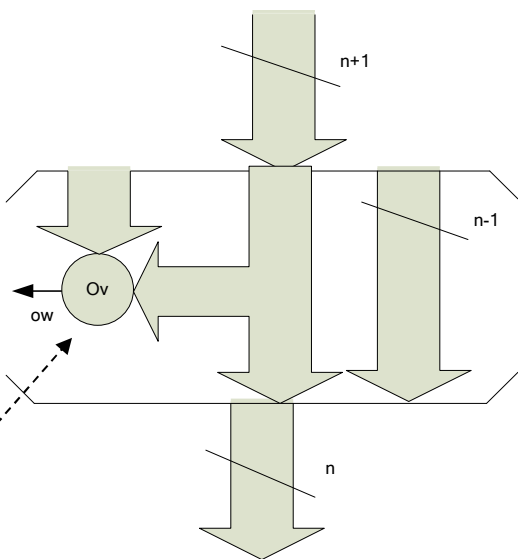


Ma in complemento alla radice, per interi con **lo stesso segno** (i.e., solo positivi o solo negativi), la relazione tra numero e rappresentazione è **monotona crescente**: a numero intero (negativo) più grande corrisponde numero naturale più grande. Quindi, la **condizione per la riducibilità** è che:

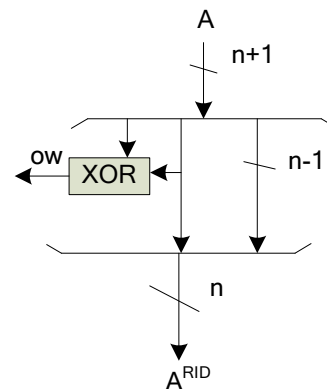
$$a_n = \beta - 1 \wedge a_{n-1} \geq \beta/2$$

Allora, la condizione per la riducibilità di un numero **intero** è l'**OR delle due condizioni**.

Nel caso in cui il numero sia riducibile, la sua rappresentazione su  $n$  cifre è data **dalle  $n$  cifre meno significative di  $A$**  (il che è ovvio, se si pensa che  $A = (A^{EST})^{RID}$ , e che nell'estensione le  $n$  cifre meno significative rimangono identiche). Quindi:  $A^{RID} = |A|_{\beta^n}$ .



Il circuito che riconosce la **non riducibilità** si chiama **circuito di overflow**.



In base 2, la condizione per la **riducibilità** di un numero (**importante**) è che le **due cifre più significative siano uguali**. Il che significa che posso, guardando l'uscita di una XOR a 2 ingressi, stabilire che un numero non è riducibile.

$$\text{Testa se: } (a_n = 0 \wedge a_{n-1} < \beta/2) \vee (a_n = \beta - 1 \wedge a_{n-1} \geq \beta/2)$$

Se devo ridurre un numero da  $n+k$  a  $n$  cifre, posso

- utilizzare  $k$  XOR a due ingressi, e poi ottenere l'overflow come OR delle uscite di questi
- utilizzare **una AND ed una OR a  $k+1$  ingressi**, e controllare che **le loro uscite siano identiche** usando una XOR a due ingressi (in genere si fa così).

#### 4.4.1 Esercizio (da fare a casa)

Descrivere un detettore di riducibilità per numeri interi a  $n$  cifre in base 4. Si assuma che l'uscita del detettore valga 1 se il numero intero è riducibile. Sintetizzare il circuito a porte NOR. Individuare le liste di copertura di costo minimo. Individuare, classificare eliminare eventuali alee per tutte le liste di copertura a costo minimo trovate.

#### Soluzione

### 4.5 Moltiplicazione/Divisione per potenza della base

Dato  $A \equiv (a_{n-1}a_{n-2}\dots a_0)_\beta$  su  $n$  cifre, con  $a \leftrightarrow A$ , voglio trovare  $B$  su  $n+1$  cifre tale che  $b \leftrightarrow B$  e  $b = \beta \cdot a$ . In realtà dovremmo prima chiederci se  $b$  è rappresentabile su  $n+1$  cifre, ma la risposta è ovvia. La soluzione è  $\boxed{B = \beta \cdot A}$ . Infatti,

$$L: B = \begin{cases} b = \beta \cdot a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^{n+1} + b = \beta^{n+1} + \beta \cdot a = \beta \cdot (\beta^n + a) & -\frac{\beta^n}{2} \leq a < 0 \end{cases}, \text{ da cui la tesi.}$$

Analogamente, dato  $a \leftrightarrow A \equiv (a_n a_{n-1} a_{n-2} \dots a_0)_\beta$  su  $n+1$  cifre, voglio calcolare  $B \leftrightarrow b$ , con  $b = \lfloor a / \beta \rfloor$ . La soluzione è, ovviamente,  $\boxed{B = \lfloor A / \beta \rfloor}$ . Infatti

$$\begin{aligned} B = \left\lfloor \frac{a}{\beta} \right\rfloor_{\beta^n} &= \left\lfloor \frac{\lfloor a / \beta^{n+1} \rfloor \cdot \beta^{n+1} + |a|_{\beta^{n+1}}}{\beta} \right\rfloor_{\beta^n} = \left\lfloor \lfloor a / \beta^{n+1} \rfloor \cdot \beta^n + \frac{|a|_{\beta^{n+1}}}{\beta} \right\rfloor_{\beta^n} \\ &= \left\lfloor \frac{A}{\beta} \right\rfloor_{\beta^n} = \left\lfloor \frac{A}{\beta} \right\rfloor \end{aligned}$$

Scivo  $a$  come quoziente e resto della divisione per  $\beta^{n+1}$

Anche per gli **interi**, moltiplicare e dividere per una potenza della base è un'operazione di costo nullo, e si fa nello stesso modo che con i naturali.

#### 4.5.1 Shift Logico ed Aritmetico

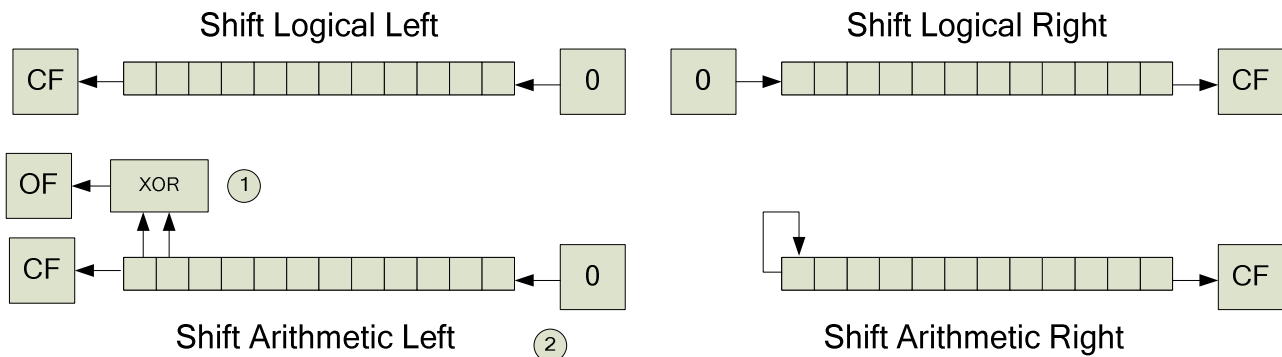
In Assembler esistono **due tipi di istruzione di shift** (per ciascuna direzione)

- shift **logical** left (right) – corrisponde alla moltiplicazione (divisione) per  $2^k$  di un **naturale**
- shift **arithmetic** left (right) – corrisponde alla moltiplicazione (divisione) per  $2^k$  di un **intero**

La cosa sembra strana, a prima vista, dato che abbiamo appena dimostrato che le operazioni di moltiplicazione e divisione per una potenza della base sono identiche per i naturali e per gli interi.

Lo sono quando posso, rispettivamente, **aumentare e diminuire** di una unità le cifre della rappre-

sentazione. Quando, invece, lavoro sul contenuto dei registri in assembler, il numero di bit della rappresentazione è **fissato** (e pari alla dimensione del registro), e quindi devo adottare qualche cautela in più.



Infatti, per moltiplicare per due un **intero** devo compiere le seguenti azioni:

- 1) avendo a disposizione solo  $n$  bit, devo stabilire se il nuovo numero sta su  $n$  bit. Questo è equivalente a stabilire se il **vecchio** numero sta su  $n-1$  bit, il che si può fare usando un **detettore di riducibilità**, la cui uscita è associata all'overflow.
- 2) Una volta salvata in OF la memoria della fattibilità dell'operazione, la moltiplicazione si fa come con i naturali, cioè shiftando ogni cifra a sinistra

Pertanto, i due shift sinistri potrebbero essere svolti con la stessa operazione macchina, lasciando al programmatore il compito di guardare il flag rilevante alla luce dell'interpretazione del contenuto del registro.

Per **dividere** un intero per due, devo compiere le seguenti azioni.

- 1) **estenderne** la rappresentazione ad  $n+1$  cifre, il che si fa **ripetendo la cifra più significativa**
- 2) prendere le  $n$  cifre più significative della rappresentazione estesa, buttando via la meno significativa.

Pertanto, i due shift destri compiono operazioni intrinsecamente differenti. Questo è il motivo per cui esistono due istruzioni di shift destro differenti (e, per simmetria, due di shift sinistro differenti) per naturali ed interi.

## 4.6 Somma

Dati  $A, B$  in base  $\beta$  su  $n$  cifre, tali che  $a \leftrightarrow A$  e  $b \leftrightarrow B$ , voglio calcolare  $S$  su  $n$  cifre tale che  $s \leftrightarrow S$  ed  $s = a + b$ . Il problema è che  $-\beta^n \leq s \leq \beta^n - 2$ , e quindi la somma può **non essere rappresentabile su  $n$  cifre**. Però lo è sicuramente su  **$n+1$  cifre**, in quanto  $\beta^{n+1}/2 \geq \beta^n$ . Pertanto il circuito sommatore dovrà essere dotato di un'uscita di **overflow**.

Quando  $s$  è rappresentabile su  $n$  cifre, abbiamo:

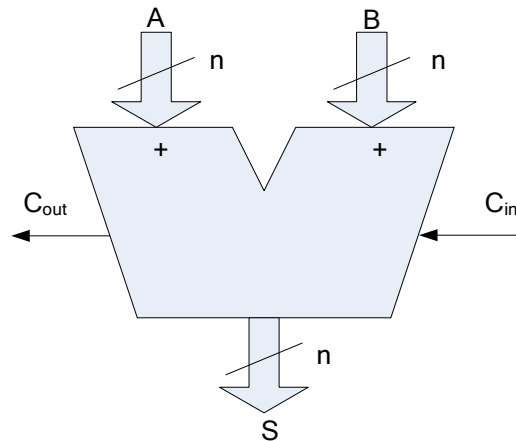
$$S = |s|_{\beta^n} = |a + b|_{\beta^n} = \left| |a|_{\beta^n} + |b|_{\beta^n} \right|_{\beta^n} = |A + B|_{\beta^n}$$

Il che significa che posso ricavare **la rappresentazione della somma come somma delle rappresentazioni** modulo  $\beta^n$ . Questa è una proprietà estremamente importante, che da sola motiva l'utilizzo della **rappresentazione in complemento alla radice** dei numeri interi. Del resto, le altre tre operazioni fondamentali (sottrazione, moltiplicazione, divisione) si fanno usando il sommatore come circuito di base.

Ricordiamo che, presi due naturali  $A$  e  $B$ , il

**circuito sommatore** produce in uscita

- $|A + B|_{\beta^n}$
- $C_{out}$ , che mi dice se la somma tra **naturali** è rappresentabile, in quanto minore di  $\beta^n$



Vediamo come si sintetizza la parte di circuito che produce l'overflow. Osserviamo che il **riporto uscente** non è di nessun aiuto. Consideriamo infatti i seguenti esempi:

- $A = \beta^n - 1, B = 1 \Rightarrow S = 0, C_{out} = 1$

Ma se  $A = \beta^n - 1$ , allora  $a \leftrightarrow -(\overline{A} + 1) = -1$ . Quindi  $s = a + b = 0$ , che è un numero intero perfettamente rappresentabile. Ciononostante,  $C_{out} = 1$

- $A = \beta^n / 2 - 1, B = \beta^n / 2 - 1 \Rightarrow S = \beta^n - 2, C_{out} = 0$

Ma se  $S = \beta^n - 2$ , allora  $s \leftrightarrow -(\overline{S} + 1) = -2$ , che non può essere la somma di due numeri **positivi**. Ciononostante,  $C_{out} = 0$

Quindi, quando la somma è tra **naturali**, il riporto uscente mi dice se è rappresentabile. Quando la somma è tra rappresentazioni di **interi**, il **riporto uscente non offre nessuna informazione riguardo alla rappresentabilità del risultato**.

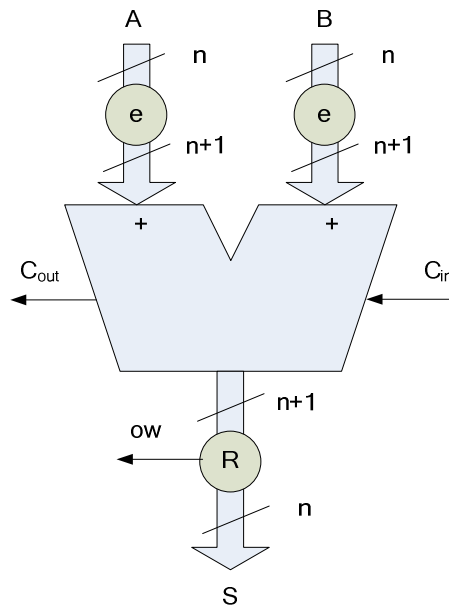
Abbiamo già osservato come la rappresentabilità della somma sia garantita se ho a disposizione **n+1 cifre**. Allora possiamo procedere come segue:

- 1) faccio la somma su  $n+1$  cifre, **estendendo** gli addendi. Calcolo cioè

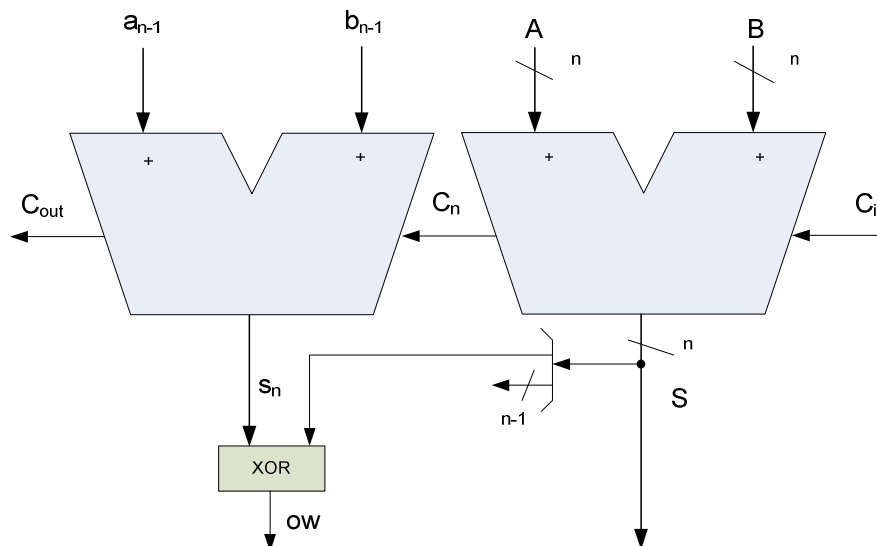
$$S^{EST} = |s|_{\beta^{n+1}} = |a + b|_{\beta^{n+1}} = \left| |a|_{\beta^{n+1}} + |b|_{\beta^{n+1}} \right|_{\beta^{n+1}} = |A^{EST} + B^{EST}|_{\beta^{n+1}}$$

- 2) controllo la **riducibilità della somma** con un apposito detettore di riducibilità:





In base 2, il tutto si semplifica come segue



Se realizzo il sommatore in modo modulare (usando full adder) in teoria mi ci vorrebbe **un modulo in più** per produrre l'overflow. Vediamo se se ne può fare a meno:

$$ow = s_n \oplus s_{n-1} = (a_{n-1} \oplus b_{n-1} \oplus c_n) \oplus (a_{n-1} \oplus b_{n-1} \oplus c_{n-1}) = 0 \oplus c_n \oplus c_{n-1} = c_n \oplus c_{n-1}$$

XOR è  
commutativo  
e associativo

Quindi basta prendere i **riporti uscenti** degli ultimi **due full adder**, senza aggiungere un altro stadio.

In sostanza con **la stessa circuiteria** con la quale faccio somme tra naturali, posso fare **somme tra interi**. Basta aggiungere uno XOR come rilevatore di overflow.

**Richiamo sull'Assembler:** in Assembler esiste una sola istruzione **ADD**, che somma numeri naturali secondo l'algoritmo visto a suo tempo. Il risultato di tale somma è corretto anche se quei numeri sono la rappresentazione di numeri interi. La rappresentabilità del risultato si stabilisce:

- guardando se CF=0, nel caso in cui i numeri sommati siano naturali
- guardando se OF=0, nel caso in cui i numeri sommati siano rappresentazioni di interi

Visto che l'unico a saperlo è il programmatore, la **ADD** setta **sia CF che OF**, e sarà poi il programmatore a testare la condizione giusta, coerentemente con quella che sa essere l'interpretazione corretta. Tipicamente, l'istruzione che segue una **ADD** sarà una **JC/JNC** nel caso di somma di naturali, oppure una **JO/JNO** nel caso di somma di interi.

## 4.7 Sottrazione

Per la sottrazione il lavoro da fare è simile a quello della somma. Il risultato non è sempre rappresentabile su  $n$  cifre, ma lo è sempre su  $n+1$  cifre. Dati  $A$  e  $B$  in base  $\beta$  su  $n$  cifre,  $a \leftrightarrow A$  e  $b \leftrightarrow B$ , voglio calcolare  $D$  su  $n$  cifre, con  $d \leftrightarrow D$  e  $d = a - b$ . Quando  $d$  è rappresentabile su  $n$  cifre, abbiamo:

$$D = \overset{(*)}{|d|_{\beta^n}} = |a - b|_{\beta^n} = \left| |a|_{\beta^n} - |b|_{\beta^n} \right|_{\beta^n} = |A - B|_{\beta^n} = |A + \overline{B} + 1|_{\beta^n} \quad (*) \text{ se } d \text{ è rappresentabile su } n \text{ cifre.}$$

Ma l'espressione che sta a destra è quella in uscita da un **sottrattore per numeri naturali**, che abbia in ingresso le rappresentazioni  $A$  e  $B$ . Quindi posso usare un sottrattore **sia** per sottrarre numeri naturali, sia per sottrarre rappresentazioni di numeri interi. Il risultato (se è rappresentabile) sarà comunque corretto in entrambi i casi. Inoltre, avrò con certezza che:

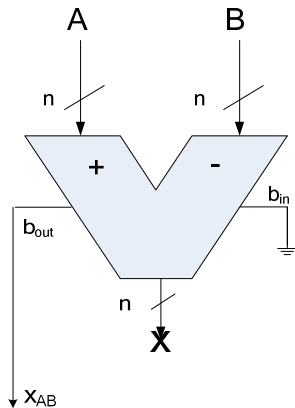
$$D^{EST} = |d|_{\beta^{n+1}} = |a - b|_{\beta^{n+1}} = \left| |a|_{\beta^{n+1}} - |b|_{\beta^{n+1}} \right|_{\beta^{n+1}} = |A^{EST} - B^{EST}|_{\beta^{n+1}} = |A^{EST} + \overline{B^{EST}} + 1|_{\beta^{n+1}}$$

Pertanto posso rilevare l'overflow verificando che il risultato della differenza su  $n+1$  cifre sia riducibile. In base due, quindi, si può facilmente verificare che il rilevatore di overflow è lo XOR degli ultimi due prestiti.

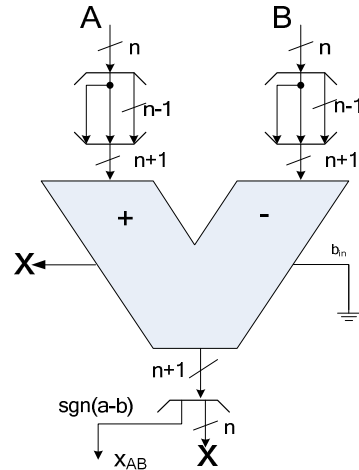
### 4.7.1 Comparazione di numeri interi

La comparazione tra numeri interi si fa sempre con un sottrattore, ma **non si deve guardare il riporto uscente**. Si deve guardare il **segno** della differenza, per svolgere la quale è necessario **estendere gli operandi** (potrebbe, infatti, non essere fattibile su  $n$  cifre).

Comparatore  
per numeri naturali



Comparatore  
per numeri interi



## 4.8 Moltiplicazione e divisione

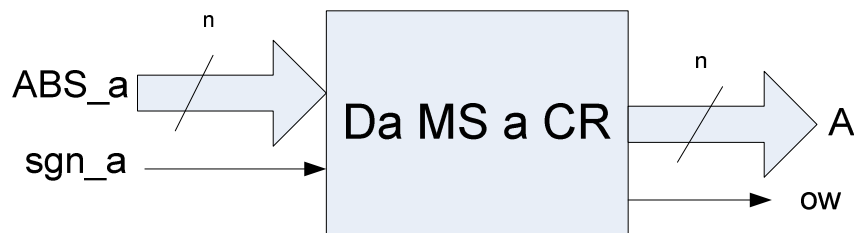
Per la moltiplicazione e divisione conviene riferirsi ai valori assoluti degli operandi, ed aggiustare i segni successivamente. Ciò consente di riutilizzare la circuiteria per le moltiplicazioni/divisioni tra numeri **naturali**.

Dovremo quindi far uso di reti che trasformano:

- da complemento alla radice a modulo e segno (già vista a suo tempo) e
- da modulo e segno a complemento alla radice.

### 4.8.1 Circuito di conversione da MS a CR

Voglio progettare un circuito che prende il modulo (su  $n$  cifre) ed il segno di un numero e ritorna la sua rappresentazione in complemento alla radice **su  $n$  cifre**.



L'operazione **non è sempre possibile**. Infatti, abbiamo:

$-(\beta^n - 1) \leq a \leq +(\beta^n - 1)$  per il numero intero rappresentato in ingresso, mentre abbiamo

$-\frac{\beta^n}{2} \leq a \leq \frac{\beta^n}{2} - 1$  per l'uscita.

È pertanto necessaria un'uscita in più, che mi dice se l'operazione è **fattibile o meno**. Tale uscita la chiamo *ow*, segnale di *overflow*, che deve essere messo ad 1 se l'operazione non è possibile, a 0 altrimenti.

Se l'operazione è fattibile, è

$$A = |a|_{\beta^n} = \begin{cases} |ABS\_a|_{\beta^n} & \text{sgn\_}a = 0 \\ |-ABS\_a|_{\beta^n} & \text{sgn\_}a = 1 \end{cases} = \begin{cases} ABS\_a & \text{sgn\_}a = 0 \\ \overline{ABS\_a + 1} & \text{sgn\_}a = 1 \end{cases}$$

Che si fa con un multiplexer ed un circuito per il calcolo dell'opposto, quindi quello visto prima.

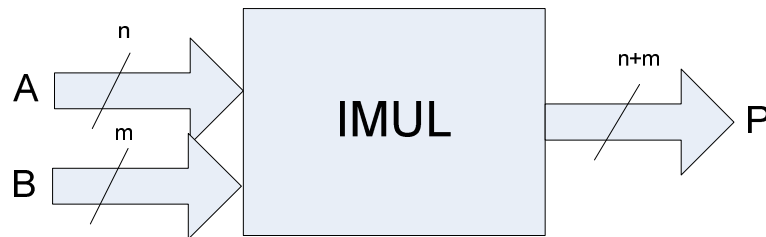
Per quanto riguarda l'overflow, abbiamo:

$$ow = 1 \Leftrightarrow (ABS\_a > \beta^n/2) \text{ or } (ABS\_a = \beta^n/2 \text{ and } \text{sgn\_}a = 0)$$

Quest'ultima cosa si fa con un comparatore e poco più (farla per esercizio).

## 4.8.2 Moltiplicazione

Dati  $A$  su  $n$  cifre e  $B$  su  $m$  cifre,  $a \leftrightarrow A$  e  $b \leftrightarrow B$ , voglio calcolare  $P$  su  $n+m$  cifre tale che  $p \leftrightarrow P$  e  $p = a \cdot b$ . Si vede velocemente che  $-\beta^{n+m}/4 \leq p \leq \beta^{n+m}/4$ , il che vuol dire che **non ci sono problemi di rappresentabilità per il risultato**.

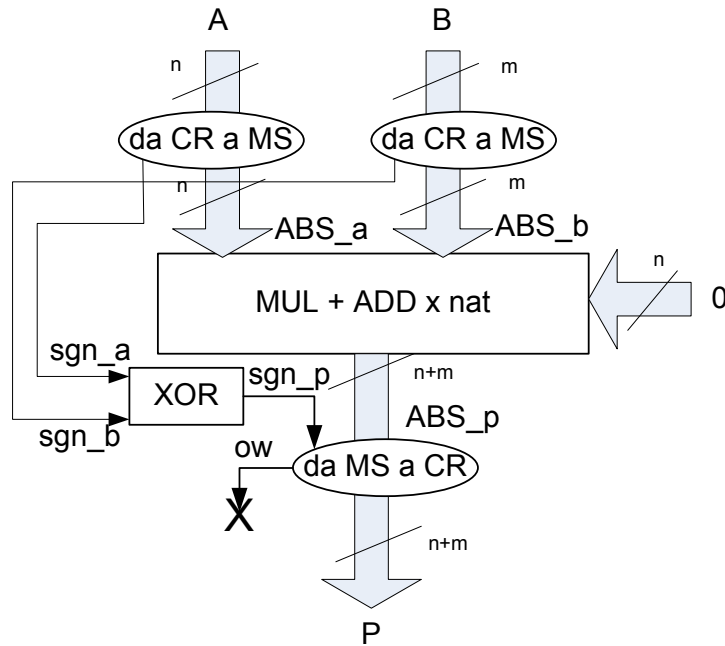


Osserviamo intanto che:

$$p = \begin{cases} ABS(a) \cdot ABS(b) & a, b \text{ concordi} \\ -ABS(a) \cdot ABS(b) & a, b \text{ discordi} \end{cases} \quad \begin{aligned} ABS(p) &= ABS(a) \cdot ABS(b) \\ \text{sgn}(p) &= \text{sgn}(a) \cdot \text{sgn}(b) \end{aligned}$$

$\text{sgn}(x)$  è una funzione matematica che vale +1 se  $x \geq 0$ , e -1 altrimenti. Quindi,  
 $x = \text{sgn}(x) \cdot ABS(x)$

Quindi posso calcolare  $ABS(a) \cdot ABS(b)$ , che è il risultato di una moltiplicazione tra **naturali**, e poi rappresentare quel risultato o il suo opposto a seconda che  $a$  e  $b$  siano concordi o discordi.



Si osservi che nella rete finale il segnale di overflow può essere trascurato, perché abbiamo già accertato che non ci sono problemi di rappresentabilità del risultato.

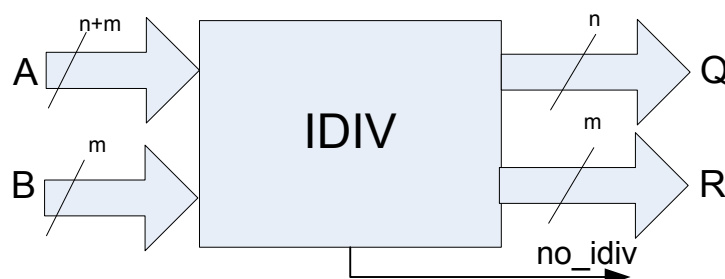
#### 4.8.3 Esercizio (da fare a casa)

- 1) Descrivere e sintetizzare (come rete SP a costo minimo) un moltiplicatore per interi ad una cifra in base 3. Individuare, classificare e rimuovere eventuali alee sulle singole uscite.
- 2) Descrivere e sintetizzare (come rete SP a costo minimo) la rete che prende in ingresso l'uscita della rete precedente e produce, su ? bit, il corrispondente numero in base due.

[Soluzione](#)

#### 4.8.4 Divisione

Dati  $A$  su  $n+m$  cifre e  $B$  su  $m$  cifre,  $a \leftrightarrow A$ ,  $b \leftrightarrow B$ , voglio calcolare  $Q$  ed  $R$ , rispettivamente su  $n$  ed  $m$  cifre, tali che  $q \leftrightarrow Q$  e  $r \leftrightarrow R$ ,  $a = q \cdot b + r$ . Si deve tener conto del fatto che  $q$  **può non essere rappresentabile su  $n$  cifre, e quindi  $Q$  può non esistere.**



Quando abbiamo enunciato il teorema della divisione con resto, abbiamo considerato soltanto il caso di **divisore naturale**. In tal caso, il teorema garantisce che il risultato è unico se  $0 \leq r \leq b-1$ .

Nel caso di divisione tra **interi**, il vincolo sopra scritto **non ha senso**. Deve essere adottato un vincolo che **generalizzi il precedente**. Si potrebbe pensare di adottare il seguente vincolo:

$\boxed{ABS(r) < ABS(b)}$ , che di fatto racchiude il precedente se i numeri sono naturali. Anche questo vincolo, però, **non basta** a rendere il risultato della divisione **univoco**. Infatti, ecco un controesempio:  $a = +17$ ,  $b = -5$ . Posso avere  $q = -3$ ,  $r = -2$ , e  $q = -4$ ,  $r = -3$ . In entrambi i casi,  $ABS(r) < ABS(b)$ .

Quindi devo aggiungere **un'altra condizione**. La condizione che si sceglie è che **il resto abbia il segno del dividendo**. In questo caso sono in grado di discriminare sempre tra i due casi. Le ipotesi che rendono unico il risultato sono quindi:

$$\begin{cases} ABS(r) < ABS(b) \\ \text{sgn}(r) = \text{sgn}(a) \end{cases}$$

Si osservi che quest'ipotesi vuol dire che nella divisione tra interi **il quoziente è approssimato per troncamento**, quindi  $q \cdot b$  è sempre più vicino all'origine rispetto ad  $a$  (la divisione che abbiamo usato finora, invece, approssima **a sinistra**).

Sotto queste ipotesi, posso riscrivere la divisione come:

$\text{sgn}(a) \cdot ABS(a) = q \cdot \text{sgn}(b) \cdot ABS(b) + \text{sgn}(r) \cdot ABS(r)$ , con  $\text{sgn}(a) = \text{sgn}(r)$  per l'ipotesi che ho appena fatto. Moltiplicando entrambi i membri per  $\text{sgn}(a)$ , si ottiene:

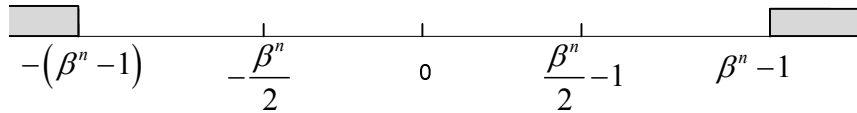
$$\boxed{ABS(a) = [q \cdot \text{sgn}(b) \cdot \text{sgn}(a)] \cdot ABS(b) + ABS(r)}$$

Che è **una divisione tra naturali**. Infatti, se il dividendo ed il divisore sono naturali, lo sarà anche il quoziente. Peraltro,  $q$  sarebbe negativo solo nel caso di segni discordi tra  $a$  e  $b$ , nel qual caso l'espressione tra parentesi quadre è comunque positiva. Chiamo  $q \cdot \text{sgn}(b) \cdot \text{sgn}(a) = ABS(q)$ .

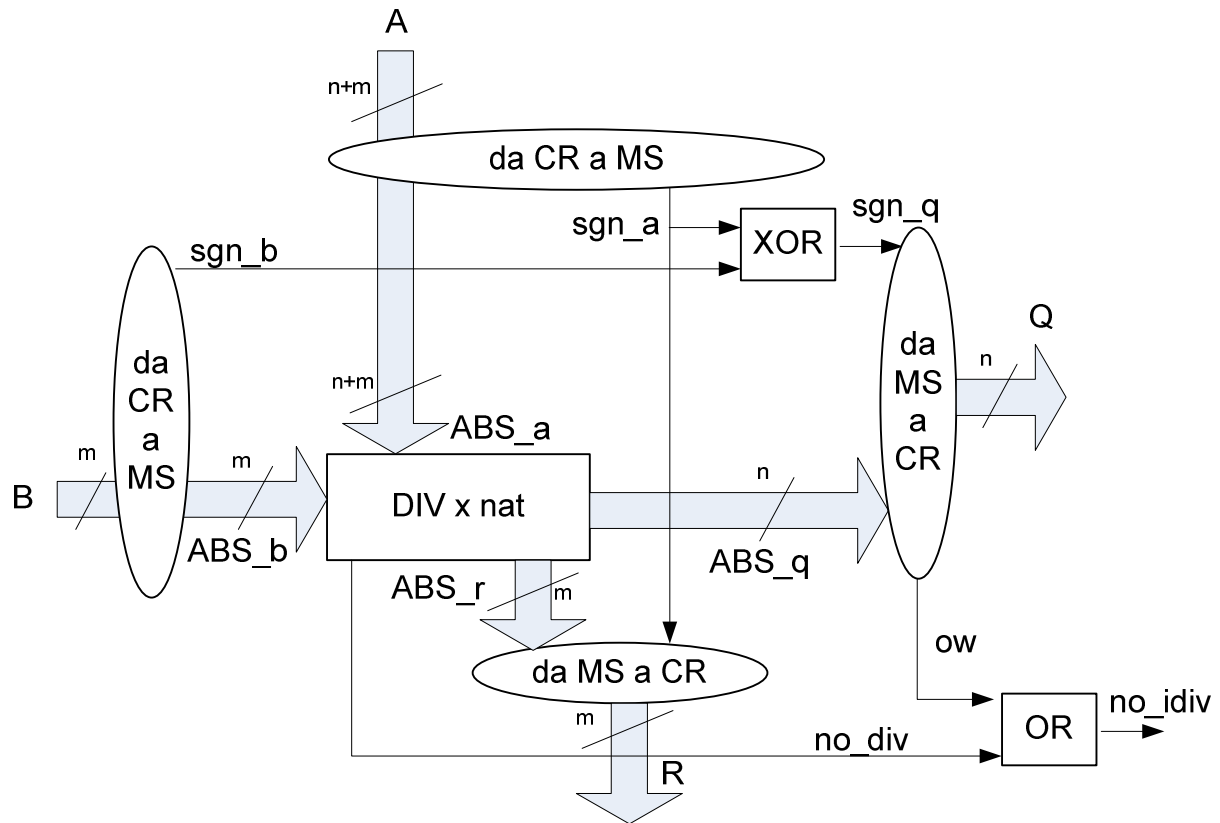
Posso calcolare  $ABS(r)$  e  $ABS(q)$  utilizzando un modulo **divisore tra naturali**, purché  $ABS(q)$  sia **un numero (naturale) rappresentabile su  $n$  cifre**. Per testare se questo è vero, dobbiamo controllare se

$$ABS(a) < \beta^n \cdot ABS(b) \quad (1)$$

il che si fa con un sottrattore ad  $n+m$  cifre (lo stesso che useremmo per testare la fattibilità di una divisione naturale). Se questa disuguaglianza è **falsa**, il quoziente della divisione naturale di sopra è **un numero naturale che sta su più di  $n$  cifre**, quindi  $ABS(q) \geq \beta^n$ . In questo caso, il quoziente  $q$  della **divisione intera** è o  $q \geq \beta^n$  (se  $q$  è positivo), oppure  $q \leq -\beta^n$  (se negativo). In ogni caso, il quoziente  $q$  **non è un numero intero rappresentabile su  $n$  cifre** se  $ABS(q)$  non è un naturale rappresentabile su  $n$  cifre.



Quindi, la **fattibilità della divisione naturale tra valori assoluti** è condizione necessaria per la **fattibilità della divisione intera**.



A questo punto, posso trovare  $Q$  ed  $R$  abbastanza facilmente, visto che ho i valori assoluti di  $q$  ed  $r$  (come uscita dal divisore naturale) ed i loro segni (ottenuti banalmente da quelli degli operandi della divisione).

Il problema è che **nella conversione della rappresentazione di  $q$  da MS a CR posso avere overflow**. Infatti, il fatto che la divisione **naturale** sia fattibile non implica che

$$-\frac{\beta^n}{2} \leq q \leq \frac{\beta^n}{2} - 1, \quad (2)$$

e quindi si può avere overflow.

Quand'è, quindi, che la **divisione intera** è fattibile? Quando:

- è fattibile quella naturale
- il risultato di quella naturale è un valore assoluto minore di  $\beta^n/2$ , oppure uguale a  $\beta^n/2$  con un segno negativo.

Quest'ultimo, però, è esattamente ciò che ci dice il segnale di overflow in uscita al convertitore da MS a CR. Quindi, l'OR dei due mi dà il segnale **no\_idiv**.

### Esempio:

Voglio svolgere la seguente divisione tra numeri in base 10:  $(-223) \div (+28)$ . Abbiamo  $a = -223$ ,  $b = +28$ ,  $n = 1$ ,  $m = 2$ . Sappiamo che  $A = |-223|_{10^3} = 777$ ,  $B = 28$ .

- 1) svolgo la divisione tra i valori assoluti:  $223 \div 28$ . Tale **divisione naturale è fattibile**, in quanto il quoziente sta su 1 cifra, ed è pari a  $Q^* = ABS(q) = 7$  (la condizione (1) è soddisfatta). Il resto è  $R^* = ABS(r) = 27$ .

- 2) I segni degli operandi sono **discordi**, ed il segno del dividendo è **negativo**.

Il **quoziente  $q$  è negativo**. Il problema è che il numero  $q = -7$  non è rappresentabile su una cifra in base 10 in complemento alla radice. Lo sarebbe se fosse compreso in  $[-5; +4]$ , ma non è questo il caso. Lo posso peraltro vedere testando la condizione (2), che non è soddisfatta. Quindi **questa divisione intera non si può fare**.

Se, invece, avessi svolto la divisione  $-123 \div 28$ , il quoziente della divisione naturale sarebbe stato  $Q^* = ABS(q) = 4$ , ed il resto  $R^* = ABS(r) = 11$ . Quindi,  $r = -11$ , ed  $R = |-11|_{10^2} = 89$ . A questo punto,  $q = -4$ , che è rappresentabile in base 10 su una cifra. Quindi abbiamo  $Q = |-4|_{10^1} = 6$ , e la divisione intera è **fattibile**.

**Richiamo sull'Assembler:** esistono **due istruzioni distinte** per la moltiplicazione e la divisione, **MUL/IMUL**, **DIV/IDIV**, che moltiplicano/dividono rispettivamente naturali ed interi. Infatti, anche se la moltiplicazione (divisione) tra interi si fa in modo simile a quella tra naturali (si può riciclare parte della circuiteria), sono necessarie anche altre operazioni, quali calcolo del valore assoluto, etc.

### 4.8.5 Esercizio (da fare a casa)

Nella divisione tra interi, il resto ha – per definizione – il segno del dividendo. Sintetizzare un divisore per interi in base due (in complemento alla radice) che restituisce un *resto sempre positivo*, e minore del valore assoluto del divisore, partendo dal modulo IDIV spiegato a lezione ed aggiungendo eventualmente altra logica. Fare in modo che la rete generi un segnale *no\_idiv* quando il risultato non è rappresentabile.

### Soluzione



## 4.9 Conversione di base tra interi

Dato  $a \leftrightarrow A_\beta$  su  $n$  cifre in base  $\beta$ , voglio trovare  $a \leftrightarrow A_\gamma$  su  $m$  cifre in base  $\gamma$ , sempre rappresentando i numeri **in complemento alla radice**. Visto che il complemento alla radice è una legge di rappresentazione che **dipende dalla base** (infatti, abbiamo  $\beta^n$  nelle espressioni), sarà, in generale.

$$\boxed{A_\beta \neq A_\gamma}$$

Il naturale che rappresenta l'intero  $a$  dipenderà dalla base. In particolare, se  $a$  è un numero **negativo** è certo che i due naturali saranno diversi. Il problema che voglio risolvere è: data una rappresentazione, trovare l'altra. Voglio cioè trovare una legge  $f$  tale che  $A_\gamma = f(A_\beta)$ .

Il primo problema è quello della **rappresentabilità**. Sono sicuro che il numero è rappresentabile nella nuova base se  $\gamma^m \geq \beta^n$ . La condizione è sufficiente, ma non necessaria. Però abbiamo già detto che non vogliamo sapere qual è il numero  $a$ , e quindi è quanto di meglio possiamo fare. Da questa si deriva:

$$\boxed{m \geq \lceil n \cdot \log_\gamma \beta \rceil}$$

Che è anche sufficientemente intuitiva. Ad esempio:

$$\beta = 10, n = 2, \gamma = 2 \text{ otteniamo } m \geq \lceil 2 \cdot \log_2 10 \rceil = 7.$$

Sintetizzando la legge **diretta** per la base  $\gamma$  ed **inversa** per la base  $\beta$  si ottiene:

$$L_\beta^{-1}: a = \begin{cases} A_\beta & a_{n-1}^\beta < \frac{\beta}{2} \\ -\beta^n + A_\beta & a_{n-1}^\beta \geq \frac{\beta}{2} \end{cases}, \quad L_\gamma: A_\gamma = \begin{cases} a & 0 \leq a \leq \frac{\beta^n}{2} - 1 \leq \frac{\gamma^m}{2} - 1 \\ \gamma^m + a & -\frac{\gamma^m}{2} \leq -\frac{\beta^n}{2} \leq a < 0 \end{cases}$$

$$\boxed{L_\gamma: A_\gamma = \begin{cases} A_\beta & a_{n-1}^\beta < \frac{\beta}{2} \\ \gamma^m - \beta^n + A_\beta & a_{n-1}^\beta \geq \frac{\beta}{2} \end{cases}}$$

**Esempio:**

$$\beta = 2, n = 8, a \leftrightarrow (11001111)_2 = 128 + 64 + 15 = 207$$

Voglio trovare  $m$  per la rappresentazione in base 10:  $m = \lceil 8 \cdot \log_{10} 2 \rceil = 3$ . Quindi,  $\gamma = 10, m = 3$ .

Il numero rappresentato in base 2 è **negativo**, in quanto la sua cifra più significativa è 1.

La rappresentazione del numero in complemento alla radice in base 10 su tre cifre è quindi:  $10^3 - 2^8 + 207 = 951$ . Anch'essa, ovviamente, corrisponde alla rappresentazione di un numero negativo, in quanto la cifra più significativa è maggiore o uguale a 5.

Attenzione: tutto questo conto l'ho fatto senza sapere quale fosse il numero  $a$ . Ho cioè lavorato solamente sulle rappresentazioni.

A quale numero intero corrisponderà? Posso applicare la legge inversa in base 10, ottenendo:

$$a = -(\overline{951} + 1) = -(048 + 1) = -49$$

Tra l'altro,  $-49$  è un numero rappresentabile **su 2 cifre** in base 10. Questo è confermato dal fatto che la rappresentazione è **riducibile**, in quanto  $a_{m-1}^{\gamma} = \gamma - 1 = 9 \wedge a_{m-2}^{\gamma} \geq \gamma/2 = 5$ .

#### 4.9.1 Esercizio (da fare a casa)

Siano  $x \leftrightarrow (Ai)_{12}, y \leftrightarrow (46)_8$  due numeri interi in complemento alla radice, con  $i = |matr|_{12}$ . Determinare la rappresentazione di  $S = x + y$  in complemento alla radice su **due cifre in base 10**. Verificare se  $S$  è o non è sempre rappresentabile.

[Soluzione](#)

## 5 Soluzioni degli esercizi per casa

### 5.1 Soluzione dell'esercizio 1.2.2

1) Il risultato si dimostra come segue:

$$|A|_\gamma = \left| \sum_{i=0}^{n-1} a_i \cdot \beta^i \right|_\gamma = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot [\gamma \cdot \delta]^i \right|_\gamma = |a_0|_\gamma$$

dove  $\delta \triangleq \beta/\gamma$ ,  $\delta \in \mathbb{N} \setminus \{0\}$  per ipotesi. L'ultimo passaggio è dovuto al fatto che il secondo addendo è certamente multiplo di  $\gamma$ , in quanto  $\delta$  è naturale.

2) Partendo dalla rappresentazione di  $A$ , il risultato è dimostrato dalla seguente sequenza di uguaglianze:

$$|A|_3 = \left| \sum_{i=0}^{n-1} a_i \cdot 4^i \right|_3 = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot (1+3)^i \right|_3 = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[ \sum_{k=0}^i \binom{i}{k} \cdot 1^{i-k} \cdot 3^k \right] \right|_3.$$

Tutti i termini dello sviluppo del binomio di Newton, tranne quello per  $k=0$ , sono numeri naturali che contengono un fattore 3 a moltiplicare. Pertanto, posso scrivere quei termini come  $3 \cdot \gamma_i$ , per  $\gamma_i$  numero naturale. Quindi, abbiamo:

$$\begin{aligned} |A|_3 &= \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot (1+3 \cdot \gamma_i) \right|_3 \\ &= \left| a_0 + \sum_{i=1}^{n-1} a_i + 3 \cdot \sum_{i=1}^{n-1} a_i \cdot \gamma_i \right|_3 = \left| \sum_{i=0}^{n-1} a_i \right|_3 \end{aligned}$$

3) Il risultato di cui al punto precedente si generalizza banalmente:

$$\begin{aligned} |A|_\gamma &= \left| \sum_{i=0}^{n-1} a_i \cdot \beta^i \right|_\gamma = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot [1 + (\beta-1)]^i \right|_\gamma \\ &= \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[ \sum_{k=0}^i \binom{i}{k} \cdot 1^{i-k} \cdot (\beta-1)^k \right] \right|_\gamma = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot (1 + (\beta-1) \cdot \gamma_i) \right|_\gamma \\ &= \left| a_0 + \sum_{i=1}^{n-1} a_i + (\beta-1) \cdot \sum_{i=1}^{n-1} a_i \cdot \gamma_i \right|_\gamma = \left| \sum_{i=0}^{n-1} a_i \right|_\gamma \end{aligned}$$

Dove il penultimo passaggio dipende dal fatto che  $\beta-1$  è multiplo di  $\gamma$  per ipotesi.

4) Il risultato si dimostra come segue:

$$\begin{aligned}
 |A|_{\gamma} &= \left| \sum_{i=0}^{n-1} a_i \cdot \beta^i \right|_{(\beta+1)} = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot [(-1) + (\beta+1)]^i \right|_{(\beta+1)} \\
 &= \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[ \sum_{k=0}^i \binom{i}{k} \cdot (-1)^{i-k} \cdot (\beta+1)^k \right] \right|_{(\beta+1)} \\
 &= \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[ (-1)^i + \sum_{k=1}^i \binom{i}{k} \cdot (-1)^{i-k} \cdot (\beta+1)^k \right] \right|_{(\beta+1)}
 \end{aligned}$$

L'ultimo passaggio si ottiene isolando dalla sommatoria più interna il termine per  $k=0$ . Da qui si evince che il secondo addendo della somma tra parentesi quadre è multiplo intero di  $\beta+1$ , e quindi può essere scritto come  $(\beta+1) \cdot \gamma_i$  per un qualche intero  $\gamma_i$ :

$$\begin{aligned}
 |A|_{\gamma} &= \left| a_0 + \sum_{i=1}^{n-1} (-1)^i \cdot a_i + \sum_{i=1}^{n-1} a_i \cdot \gamma_i \cdot (\beta+1) \right|_{(\beta+1)} \\
 &= \left| a_0 + \sum_{i=1}^{n-1} (-1)^i \cdot a_i + (\beta+1) \cdot \sum_{i=1}^{n-1} a_i \cdot \gamma_i \right|_{(\beta+1)} \\
 &= \left| a_0 + \sum_{i=1}^{n-1} (-1)^i \cdot a_i \right|_{(\beta+1)} = \left| \sum_{i=0}^{n-1} (-1)^i \cdot a_i \right|_{(\beta+1)}
 \end{aligned}$$

## 5.2 Soluzione dell'esercizio 2.4.4

L'incrementatore in base 7 prende in ingresso una cifra in base 7, codificata su 3 variabili logiche  $x_2 \dots x_0$ , più un riporto entrante  $c_{in}$ , e produce in uscita una cifra in base 7, codificata su 3 variabili logiche  $z_2 \dots z_0$ , più un riporto entrante  $c_{out}$ . La tabella di verità è la seguente:

$c_{in}$	$x_2$	$x_1$	$x_0$	$c_{out}$	$z_2$	$z_1$	$z_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	-	-	-	-
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	1	1
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	1	0	1	0	0	0
1	1	1	1	-	-	-	-

La mappa di Karnaugh per  $z_2$  è la seguente:

$x_1 x_0 \backslash c_{in} x_2$					
		00	01	11	10
00	00	0	1	1	0
	01	0	1	1	0
11	11	0	-	-	1
	10	0	1	0	0

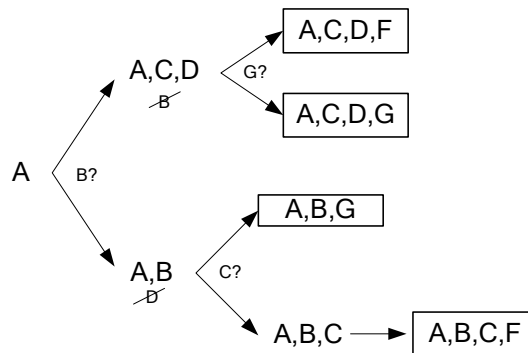
Da cui si ricava la mappa per  $\overline{z_2}$ , nella quale sono indicati anche gli implicantti principali.

$$A = \overline{x_2} \cdot \overline{x_1}, \quad B = \overline{c_{in}} \cdot \overline{x_2}, \quad C = \overline{x_2} \cdot \overline{x_0}$$

$$D = \overline{c_{in}} \cdot x_1 \cdot x_0, \quad E = x_2 \cdot x_1 \cdot x_0, \quad F = c_{in} \cdot x_2 \cdot x_1, \quad G = c_{in} \cdot x_1 \cdot \overline{x_0}$$

$\begin{matrix} c_{in}x_2 \\ x_1x_0 \end{matrix}$	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	-	-	0
10	1	0	1	1

L'unico implicants essenziale è A, E è assolutamente eliminabile e tutti gli altri sono semplicemente eliminabili. Le liste di copertura irridondanti sono quelle riquadrate nel disegno sottostante:



La lista di copertura di costo minimo rispetto al criterio a diodi è  $\{A,B,G\}$ , il cui costo è 10. La sintesi di costo minimo a porte NOR per  $z_2$  è la seguente:

$$\overline{z_2} = \overline{x_2} \cdot \overline{x_1} + \overline{c_{in}} \cdot \overline{x_2} + \overline{c_{in}} \cdot x_1 \cdot \overline{x_0}$$

$$z_2 = \overline{\left( \overline{x_2 + x_1} \right) + \left( \overline{c_{in} + x_2} \right) + \left( \overline{c_{in} + x_1 + x_0} \right)}$$

La sintesi secondo la lista di copertura di costo minimo è soggetta ad alee statiche del primo ordine sul livello 1 per le transizioni  $0010 \leftrightarrow 1010$ ,  $1000 \leftrightarrow 1010$ . Tali alee possono essere rimosse inserendo l'implicants C nella lista di copertura.

$$z_2 = \overline{\left( \overline{x_2 + x_1} \right) + \left( \overline{c_{in} + x_2} \right) + \left( \overline{c_{in} + x_1 + x_0} \right) + \left( \overline{x_2 + x_0} \right)}$$

### 5.3 Soluzione dell'esercizio 2.4.5

- 1) il circuito di lookahead a 2 bit prende in ingresso due bit di ciascun addendo, che chiamiamo  $x_1$   $x_0$  ed  $y_1$   $y_0$  ed un riporto entrante  $c_{in}$ , e produce il riporto uscente  $c_{out}$  di una somma a due bit. Pertanto, una descrizione del circuito è la seguente:

$x_1x_0 \backslash y_1y_0$		$c_{in}=0$				$x_1x_0 \backslash y_1y_0$		$c_{in}=1$			
		00	01	11	10			00	01	11	10
00	00	0	0	0	0	00	00	0	0	1	0
01	01	0	0	1	0	01	01	0	0	1	1
11	11	0	1	1	1	11	11	1	1	1	1
10	10	0	0	1	1	10	10	0	1	1	1

- 2) In un sommatore a  $N=2n$  bit con il circuito di lookahead a 2 bit, il riporto si propaga in tempo  $2\tau$  per ogni blocco di 2 full adder. All'ultimo di tali blocchi di due full adder il riporto entrante sarà pronto al tempo  $2 \cdot (n-1) \cdot \tau$ . Quindi, l'ultimo full adder potrà fornire il risultato al tempo

$$\boxed{\left[2 \cdot (n-1) + 4\right] \cdot \tau = 2 \cdot (n+1) \cdot \tau}.$$

- 3) Gli implicanti principali sono i seguenti (quelli ottenibili l'uno dall'altro per simmetria sono riportati nel medesimo colore):

$$x_1 \cdot y_1, \text{ } c_{in} \cdot x_1 \cdot y_0, \text{ } c_{in} \cdot y_1 \cdot x_0, \text{ } c_{in} \cdot x_1 \cdot x_0, \text{ } c_{in} \cdot y_1 \cdot y_0, \text{ } x_1 \cdot x_0 \cdot y_0, \text{ } y_1 \cdot y_0 \cdot x_0$$

Si vede senza difficoltà che gli IP sopra menzionati sono tutti essenziali. Quindi, la sintesi SP di costo minimo è:  $z = x_1 \cdot y_1 + c_{in} \cdot x_1 \cdot y_0 + c_{in} \cdot y_1 \cdot x_0 + c_{in} \cdot x_1 \cdot x_0 + c_{in} \cdot y_1 \cdot y_0 + x_1 \cdot x_0 \cdot y_0 + y_1 \cdot y_0 \cdot x_0$ .

## 5.4 Soluzione dell'esercizio 2.4.6

$x_1 \backslash x_0$	00	01	11	10
$b \ x_2$				
00	0	0	1	0
01	1	----	----	----
11	1	----	----	----
10	0	0	0	0

$x_1 \backslash x_0$	00	01	11	10
$y_2 \ y_1 \ y_0$				
00	000	010	001	100
01	011	----	----	----
11	000	----	----	----
10	001	010	100	011

2) Dalla mappa sopra scritta si ricava immediatamente la sintesi SP, e da questa quella a porte NAND:

$$\begin{array}{l}
 c = x_2 + \bar{b} \cdot x_1 \cdot x_0 \\
 y_2 = b \cdot x_1 \cdot x_0 + \bar{b} \cdot x_1 \cdot \bar{x}_0 \\
 y_1 = \bar{x}_1 \cdot x_0 + \bar{b} \cdot x_2 + b \cdot x_1 \cdot \bar{x}_0 \\
 y_0 = b \cdot \bar{x}_2 \cdot \bar{x}_0 + \bar{b} \cdot x_2 + \bar{b} \cdot x_1 \cdot x_0
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{l}
 c = \overline{\bar{x}_2 \cdot (b \cdot x_1 \cdot x_0)} \\
 y_2 = \overline{(b \cdot x_1 \cdot x_0) \cdot (\bar{b} \cdot x_1 \cdot \bar{x}_0)} \\
 y_1 = \overline{(\bar{x}_1 \cdot x_0) \cdot (\bar{b} \cdot x_2) \cdot (b \cdot x_1 \cdot \bar{x}_0)} \\
 y_0 = \overline{(b \cdot \bar{x}_2 \cdot \bar{x}_0) \cdot (\bar{b} \cdot x_2) \cdot (\bar{b} \cdot x_1 \cdot x_0)}
 \end{array}$$

3) Per la sintesi PS abbiamo:

$$\begin{array}{l}
 \bar{c} = \bar{x}_2 \cdot \bar{x}_1 + x_1 \cdot \bar{x}_0 + b \cdot \bar{x}_2 \\
 \bar{y}_2 = \bar{x}_1 + \bar{b} \cdot x_0 + b \cdot \bar{x}_0 \\
 \bar{y}_1 = \bar{b} \cdot x_1 + x_1 \cdot x_0 + b \cdot x_2 + \bar{x}_2 \cdot \bar{x}_1 \cdot \bar{x}_0 \\
 \bar{y}_0 = b \cdot x_2 + \bar{x}_1 \cdot x_0 + b \cdot x_0 + \bar{b} \cdot \bar{x}_2 \cdot \bar{x}_0
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{l}
 c = (x_2 + x_1) \cdot (\bar{x}_1 + x_0) \cdot (\bar{b} + x_2) \\
 y_2 = (x_1) \cdot (b + \bar{x}_0) \cdot (\bar{b} + x_0) \\
 y_1 = (b + \bar{x}_1) \cdot (\bar{x}_1 + \bar{x}_0) \cdot (\bar{b} + \bar{x}_2) \cdot (x_2 + x_1 + x_0) \\
 y_0 = (\bar{b} + \bar{x}_2) \cdot (x_1 + \bar{x}_0) \cdot (\bar{b} + x_0) \cdot (b + x_2 + x_0)
 \end{array}$$

Da cui si ricava quella a porte NOR.

$$\begin{array}{l}
 c = \overline{(\bar{x}_2 + x_1) + (\bar{x}_1 + x_0) + (\bar{b} + x_2)} \\
 y_2 = \overline{(\bar{x}_1) + (b + \bar{x}_0) + (\bar{b} + x_0)} \\
 y_1 = \overline{(\bar{b} + \bar{x}_1) + (\bar{x}_1 + \bar{x}_0) + (\bar{b} + \bar{x}_2) + (x_2 + x_1 + x_0)} \\
 y_0 = \overline{(\bar{b} + \bar{x}_2) + (x_1 + \bar{x}_0) + (\bar{b} + x_0) + (b + x_2 + x_0)}
 \end{array}$$

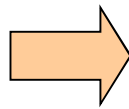
4) la realizzazione a porte NAND di  $y_0$  costa 4 a porte e 11 a diodi, mentre quella a porte NOR costa 5 a porte e 13 a diodi. Pertanto, la prima ha costo minore.

## 5.5 Soluzione dell'esercizio 2.7.2

Il numero di variabili logiche di ingresso alla rete è  $5 \cdot 4 = 20$ , essendo una cifra in base 10 BCD codificata su 4 variabili.

Verificare la divisibilità per 2, 5, 10 è estremamente semplice. Infatti, la divisibilità può essere decisa sulla base del valore della cifra  $a_0$ . Dette  $x_3, \dots, x_0$  le variabili logiche che codificano  $a_0$ , si ottiene la seguente tabella di verità

$x_3$	$x_2$	$x_1$	$x_0$	$z_2$	$z_5$	$z_{10}$
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	-	-	-
1	0	1	1	-	-	-
1	1	0	0	-	-	-
1	1	0	1	-	-	-
1	1	1	0	-	-	-
1	1	1	1	-	-	-



$x_3x_2$ $x_1x_0$	$z_2 z_5 z_{10}$			
	00	01	11	10
00	111	100	---	100
01	000	010	---	000
11	000	000	---	---
10	100	100	---	---

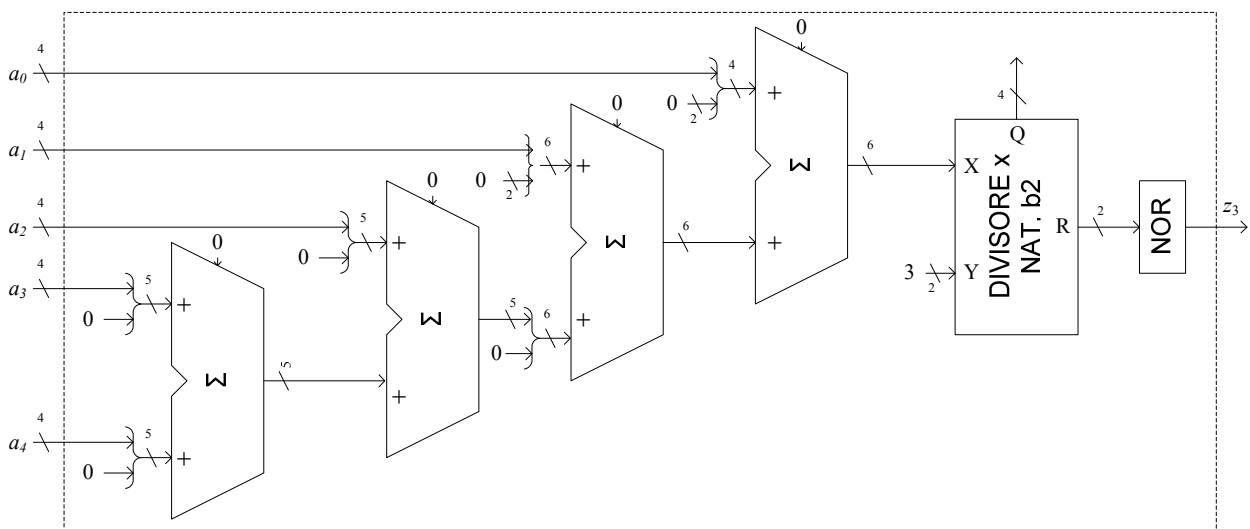
Da cui si ricava immediatamente:

$$z_2 = \overline{x_0}, \quad z_5 = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0, \quad z_{10} = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0}$$

Il criterio di divisibilità per 3 richiede invece di conoscere *la somma delle cifre* in base 10. Si noti

che, detta  $a_i$  l' $i$ -esima cifra in base 10,  $\sum_{i=0}^4 a_i \leq 45$ . Il numero naturale 45 sta su 6 bit.

La rete che sintetizza l'uscita  $z_3$  è riportata nella figura sottostante.





## 5.6 Soluzione dell'esercizio 3.2.1

1) Se  $x$  è rappresentabile su  $n$  cifre in complemento alla radice, lo è anche in traslazione e viceversa. Pertanto l'operazione di calcolare  $Y$  dato  $X$  è sempre possibile.

La relazione tra il numero intero  $x$  e la sua rappresentazione in complemento alla radice è:

$$x = \begin{cases} X & \text{se } X_{n-1} < \beta/2 \\ X - \beta^n & \text{se } X_{n-1} \geq \beta/2 \end{cases}$$

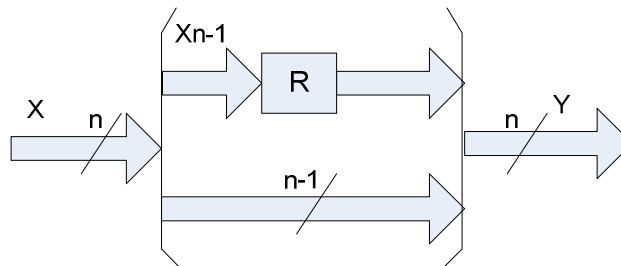
mentre quella tra  $Y$  e  $x$  è  $Y = x + \beta^n/2$ . Mettendo insieme le due si ottiene:

$$Y = \begin{cases} X + \beta^n/2 & \text{se } X_{n-1} < \beta/2 \\ X - \beta^n/2 & \text{se } X_{n-1} \geq \beta/2 \end{cases}$$

e considerando che  $\beta^n/2 = \beta^{n-1} \cdot \beta/2$ , si ottiene rapidamente

$$Y = \begin{cases} (X_{n-1} + \beta/2, X_{n-2}, \dots, X_0) & \text{se } X_{n-1} < \beta/2 \\ (X_{n-1} - \beta/2, X_{n-2}, \dots, X_0) & \text{se } X_{n-1} \geq \beta/2 \end{cases} \quad (1.3)$$

In una base generica, il circuito che realizza la (1.3) è quindi il seguente:



Dove la rete  $R$  modifica la cifra di peso più significativo, in accordo all'espressione scritta sopra.

2) Siano  $d_2 \dots d_0$  i tre bit che rappresentano la cifra  $X_{n-1}$ , e  $z_2 \dots z_0$  i tre bit che rappresentano  $Y_{n-1}$ .

Il circuito che realizza la (1.3) in base 6 si trova come sintesi minima sulla seguente mappa di Karnaugh:

$d_2 d_1$		$d_0$			
		00	01	11	10
$d_0$	0	011	101	---	001
	1	100	000	---	010

$z_2 z_1 z_0$

La cui sintesi a costo minimo è la seguente:

$$\begin{aligned} z_2 &= d_1 \cdot \overline{d_0} + \overline{d_2} \cdot \overline{d_1} \cdot d_0 \\ z_1 &= d_2 \cdot d_0 + \overline{d_2} \cdot \overline{d_1} \cdot \overline{d_0} \\ z_0 &= \overline{d_0} \end{aligned}$$

3) Siano  $d_3 \dots d_0$  i quattro bit che rappresentano la cifra  $X_{n-1}$ , e  $z_3 \dots z_0$  i quattro bit che rappresentano la cifra  $Y_{n-1}$ . La relazione richiesta è  $z_3 = \overline{d_3}$ ,  $z_i = d_i$  per  $i = 0, 1, 2$ , in quanto un numero in base 16 in codifica 8421 su  $n$  cifre ha la stessa rappresentazione di un numero in base 2 su  $4n$  cifre, ed è nota dalla teoria dei convertitori la relazione tra la rappresentazione in complemento alla radice

ed in traslazione per i numeri in base due. In ogni caso, il circuito che realizza la (1.3) in base 16 si trova come sintesi minima sulla seguente mappa di Karnaugh:

		$d_3d_2$			
		$d_1d_0$	00	01	11
$d_3d_2$	00	1000	1100	0100	0000
	01	1001	1101	0101	0001
	11	1011	1111	0111	0011
	10	1010	1110	0110	0010

$z_3z_2z_1z_0$

## 5.7 Soluzione dell'esercizio 4.3.1

		$a_1a_0$			
		$a_3a_2$	00	01	11
$a_3a_2$	00	0000	0000	0000	0000
	01	0000	1001	1001	1001
	11	----	----	----	----
	10	1001	1001	----	----

$z_3z_2z_1z_0$   
n

Dalla mappa di Karnaugh a sinistra si nota immediatamente che  $z_2 = z_1 = 0$ , e che  $z_3 = z_0$ . Pertanto la sintesi può essere svolta una volta sola.

### Sintesi a porte NOR

		$a_1a_0$			
		$a_3a_2$	00	01	11
$a_3a_2$	00	1	A 1	1	1
	01	1	0	0	0
	11	---	---	---	---
	10	0	0	---	---

$\bar{z}_3$

A è l'unico IP essenziale, B,C, F sono assolutamente eliminabili e D,E sono semplicemente eliminabili. Pertanto le sintesi a costo minimo sono due, A,D ed A,E. La prima è anche esente da alee del 1° ordine.

$$\overline{z_3} = \overline{z_0} = \overline{a_3 \cdot a_2 + a_3 \cdot a_1 \cdot a_0}$$

$$z_3 = z_0 = \overline{(a_3 + a_2)} + \overline{(a_3 + a_1 + a_0)}$$

### Sintesi a porte NAND

$a_3 a_2$ \ $a_1 a_0$	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	---	---	---	---
10	1	1	---	---

$z_3$

I tre implicanti sono tutti essenziali. Esiste una sola sintesi, quindi esente da alee, ed è

$$\begin{aligned}
 z_3 = z_0 &= a_3 + a_2 \cdot a_0 + a_2 \cdot a_1 \\
 &= a_3 + (a_2 \cdot a_0) + (a_2 \cdot a_1) \\
 &= a_3 \cdot (a_2 \cdot a_0) \cdot (a_2 \cdot a_1)
 \end{aligned}$$

### 5.8 Soluzione dell'esercizio 4.4.1

Dette  $h_1 h_0$  e  $l_1 l_0$  le variabili che codificano le cifre di peso  $n-1$  ed  $n-2$ , la mappa di Karnaugh è la seguente:

$h_1 h_0$ \ $l_1 l_0$	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	0	0	1	0
10	0	0	1	0

$z$

$h_1 h_0$ \ $l_1 l_0$	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	0	1
10	1	1	0	1

$\bar{z}$

Gli implicanti principali per  $\bar{z}$  sono elencati a destra, e sono tutti semplicemente eliminabili. Si osserva che non esistono liste di copertura con due implicant, quindi la lista di copertura a costo minimo deve includerne almeno tre. Le liste di copertura irridondanti sono le seguenti: ABC, A'B'C', AA'CC', AA'BB', BB'CC', AA'C'B', AA'BC. Quelle di costo minimo sono ABC, A'B'C', dalle quali si ricava:

$$\begin{aligned}
 \bar{z} &= \bar{h}_1 \cdot h_0 + h_1 \cdot \bar{l}_1 + \bar{h}_0 \cdot l_1 \\
 z &= (\bar{h}_1 \cdot h_0) + (h_1 \cdot \bar{l}_1) + (\bar{h}_0 \cdot l_1) \\
 &= (\bar{h}_1 + \bar{h}_0) + (\bar{h}_1 + l_1) + (\bar{h}_0 + l_1)
 \end{aligned}$$

$$\begin{aligned}
 \bar{z} &= h_1 \cdot \bar{h}_0 + \bar{h}_1 \cdot l_1 + h_0 \cdot \bar{l}_1 \\
 z &= (h_1 \cdot \bar{h}_0) + (\bar{h}_1 \cdot l_1) + (h_0 \cdot \bar{l}_1) \\
 &= (\bar{h}_1 + h_0) + (\bar{h}_1 + l_1) + (\bar{h}_0 + l_1)
 \end{aligned}$$

In entrambe le sintesi a porte NOR si individuano alee statiche del 1° ordine sul livello 0, in corrispondenza delle seguenti transizioni:

- ABC: 010x -110x, 001x-011x, 100x-101x
- A'B'C': 100x-110x, 101x-001x, 010x-011x

In tutti e due i casi le alee possono essere eliminate inserendo tutti e tre i sottocubi mancanti. La sintesi priva di alee è quindi unica, ed include tutti i sottocubi principali.

### 5.9 Soluzione dell'esercizio 4.8.3

1) Il prodotto di due numeri interi ad una cifra si rappresenta su due cifre nella stessa base. Dette  $x_1x_0$   $y_1y_0$  le rappresentazioni dei numeri interi  $x, y$  tali che  $p = x \cdot y$ , si ottiene la seguente mappa di Karnaugh. Il riempimento della mappa risulta estremamente più semplice se si riportano, in riga e colonna, i numeri interi  $x, y$  (in rosso), e, in tabella, la rappresentazione  $P$  del risultato  $p$  (in blu):

		$y_1y_0$			
		0	+1	-	-1
$x_1x_0$	00	00 00 00	00 00 00	-- --	00 00 00
	01	00 00 00	00 01 01	-- --	10 10 22
	11	-- --	-- --	-- --	-- --
	10	00 00 00	10 10 22	-- --	00 01 01
		$z_3z_2$		$z_1z_0$	

Pertanto si ottiene:

$$z_3 = z_1 = x_1 \cdot y_0 + y_1 \cdot x_0$$

$$z_2 = 0$$

$$z_0 = y_1 \cdot x_1 + y_0 \cdot x_0$$

Gli implicant usati sono tutti essenziali, e non esistono alee di alcun tipo sulle singole uscite.

Si noti che  $P$  è riducibile.

2) Il moltiplicatore in base 3 ad una cifra produce uno dei seguenti risultati: 0, +1, -1. In base due l'intervallo di numeri  $[-1; +1]$  può essere rappresentato su due bit  $w_1w_0$ . La mappa di Karnaugh della rete è quindi la seguente (sono riportate in blu le cifre in base 3 ottenute dalla precedente mappa):

		$z_1z_0$			
		0	1	-	2
$z_3z_2$	00	00	01	--	--
	01	--	--	--	--
	11	--	--	--	--
	10	--	--	--	11
		$w_1w_0$			

Si ottiene immediatamente quanto segue:

$$w_1 = z_1$$

$$w_0 = z_0 + z_1$$

Si noti che il risultato non dipende da  $z_3, z_2$ .

Infatti, la rappresentazione  $P$  è riducibile, quindi la sua cifra più significativa (codificata da  $z_3, z_2$ ) non può che essere costante, quindi irrilevante ai fini dell'individuazione del numero stesso.



Quindi, in generale, la somma di due numeri interi in base 12 ed in base 8 su due cifre **non è rappresentabile** in base 10 su due cifre. Il che non impedisce che il risultato di **questa somma** sia rappresentabile.

Il numero  $i$  è un numero qualunque, che va da 0 a  $B$ . Più grande è  $i$ , più grande è il numero (negativo) rappresentato. Quindi, per discutere sulla rappresentabilità del risultato, dovrei osservare se la somma è rappresentabile nel caso in cui  $x$  sia il numero più piccolo (più negativo) possibile. In quel caso, ottengo  $x_{\min} = (A0)_{12}$

Che numero è la somma? Applico la regola inversa per calcolare i numeri interi  $x_{\min}$  ed  $y$  e li sommo

$$\begin{aligned} S &= -\left\{ \left[ (\overline{A0})_{12} + 1 \right] + \left[ (\overline{46})_8 + 1 \right] \right\} \\ &= -\left\{ \left[ (1B)_{12} + 1 \right] + \left[ (31)_8 + 1 \right] \right\} \\ &= -\left\{ \left[ (20)_{12} \right] + \left[ (32)_8 \right] \right\} \\ &= -\left\{ [24] + [26] \right\} = -50 \end{aligned}$$

Quindi, la somma è un numero **negativo**, ma **sempre superiore a -50**. Quindi la somma è **sempre rappresentabile** su 2 cifre in base 10.

Passiamo adesso a cercare la rappresentazione. Per farlo, fissiamo un valore per  $i$ ,  $i = 10$ , e  $10 = (A)_{12}$ . Quindi, cerchiamo la rappresentazione in base 10, che però è:

$$S = |X_{10} + Y_{10}|_{10^2}$$

Per poter trovare  $X_{10}$  posso applicare le regole della conversione di base.

Posso convertire  $(AA)_{12}$  in un numero in base 10 su un numero di cifre  $m$  tale che  $\gamma^m \geq \beta^n$ . Quindi, mi ci vogliono **tre cifre in base 10** per poter applicare la conversione in generale. È ovvio che il numero che otterrò dovrà essere **rappresentabile su 2 cifre** in base 10, e quindi dovrò necessariamente ottenere qualcosa che sia **riducibile**. Vediamo

$$X_{10} = 10^3 - 12^2 + X_{12} = 1000 - 144 + (AA)_{12} = 1000 - 144 + (130)_{10} = 986$$

Il numero 986 in base 10 è **riducibile**, perché la sua cifra più significativa è 9 e la seconda è maggiore o uguale di 5, e quindi  $X_{10} = 86$ , corrispondente a  $x = -(13 + 1) = -14$ .

Per trovare  $Y_{10}$  applico lo stesso procedimento. Mi bastano **2 cifre in base 10**, stavolta.

$$Y_{10} = 10^2 - 8^2 + Y_8 = 100 - 64 + (46)_8 = 36 + 38 = 74$$

Il che conferma che questa sia la rappresentazione di un numero negativo.

La rappresentazione della somma, quindi, è:  $S = |86 + 74|_{10^2} = 60$ .

Tale numero corrisponde a  $s = -(\overline{60}) + 1 = -40$ .

## 6 Altri esercizi svolti

### 6.1 Esercizio – Febbraio 2006 (numeri naturali e interi)

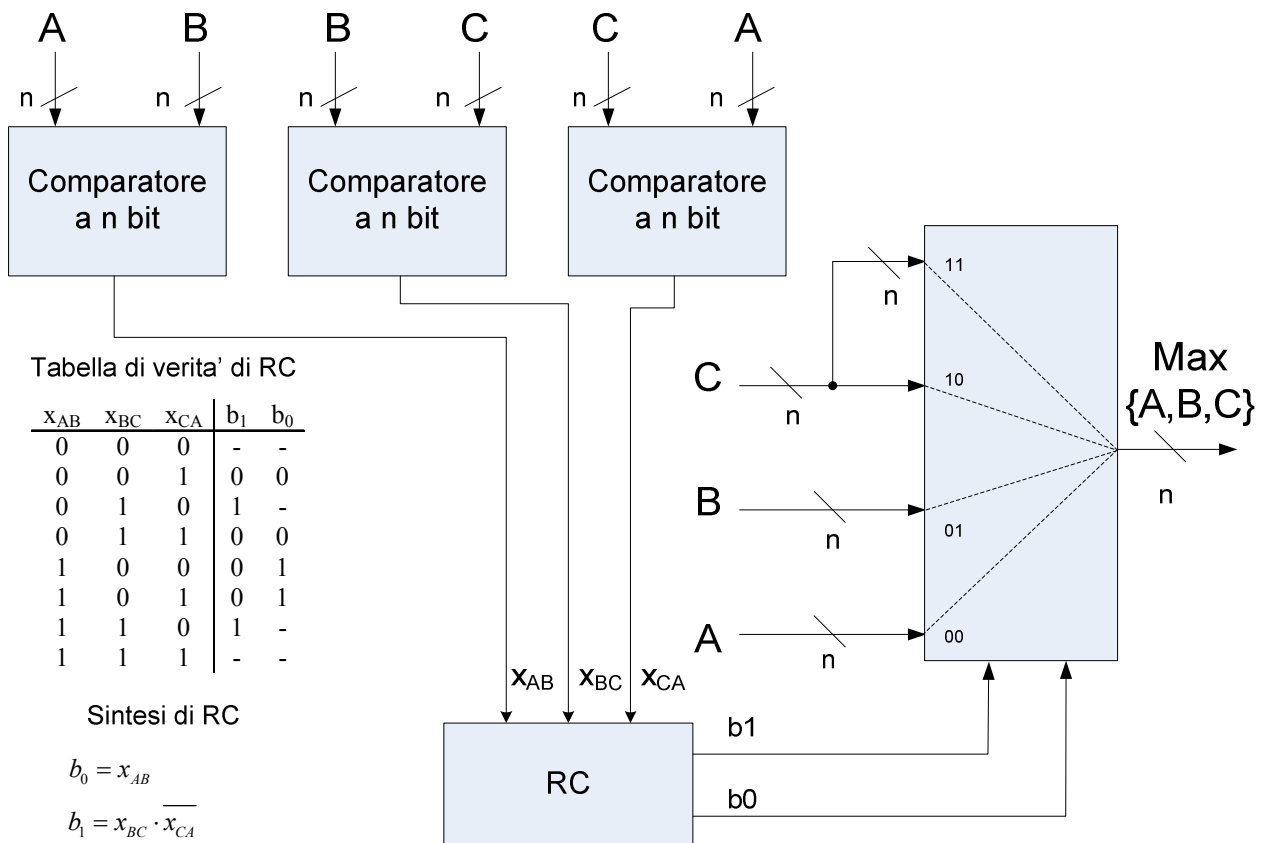
1) Disegnare la rete combinatoria che prende in ingresso la rappresentazione di tre numeri *naturali*  $a, b, c$ , ciascuno su  $n$  bit, e produce in uscita la rappresentazione del *massimo* dei tre. Descrivere e sintetizzare *tutte* le eventuali reti combinatorie non standard utilizzate.

2) Modificare, se necessario, la soluzione di cui al punto 1 in modo che produca il risultato corretto interpretando i numeri in ingresso come la rappresentazione di *interi*  $a, b, c$  codificati in *complemento alla radice*.

**NB:** nel caso si descriva la rete in Verilog, si ricordi che gli operatori di relazione  $<, >, \leq, \geq$  non sono reti standard, e quindi devono essere descritte.

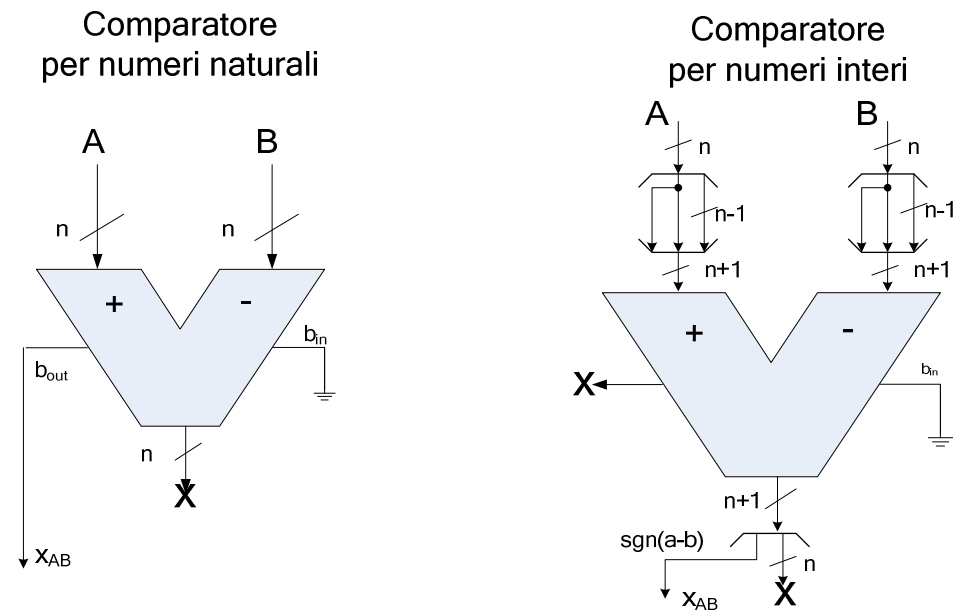
#### 6.1.1 Soluzione

Si indichi con  $A, B, C$ , la rappresentazione su  $n$  bit dei numeri  $a, b, c$ . Il bit di uscita dei tre comparatori è a 1 quando, rispettivamente,  $a < b$ ,  $b < c$ ,  $c < a$ . Confrontando le uscite dei comparatori si possono produrre, tramite la rete combinatoria RC, i due bit di comando di un multiplexer 4 to 1 che sceglie tra  $A, B, C$ .





Il comparatore può essere realizzato usando un sottrattore, in uno dei due modi descritti nella figura sottostante, a seconda che si comparino naturali (come richiesto al punto 1) o interi (come richiesto al punto 2).



Soluzioni analoghi ed ugualmente corretti si ottengono:

- usando le variabili di uscita dei tre comparatori come variabili di comando di un multiplexer 8 to 1, agli 8 ingressi del quale connettere A, B, C in modo ovvio.
- Comparando prima due numeri (e.g., A e B), e facendo uscire il massimo dei due con un multiplexer a due vie, e comparando quest'ultimo con il terzo numero (C). In questo caso servono due comparatori (invece che tre) e due multiplexer a due vie (invece che uno a quattro vie), e nessuna logica di raccordo.

## 6.2 Esercizio – Gennaio 2008 (numeri naturali)

Sintetizzare una rete combinatoria che prende in ingresso tre numeri naturali  $A$ ,  $B$  e  $C$ , ciascuno su  $n$  bit in base 2, li interpreta come le lunghezze di tre lati, e produce due uscite  $ret$  e  $area$ . L'uscita  $ret$ , su 1 bit, vale 1 se  $A$ ,  $B$  e  $C$  sono i lati di un triangolo rettangolo e zero altrimenti. L'uscita  $area$ , su  $n$  bit, contiene l'area del triangolo (a meno di approssimazioni) se  $ret$  vale 1, ed un valore non significativo altrimenti.

*Nota: Se lo si ritiene opportuno, si risolva l'esercizio supponendo di avere a disposizione una rete MAX3, che prende in ingresso i numeri  $A$ ,  $B$  e  $C$  e presenta in uscita la codifica, su 2 bit, del massimo tra i tre numeri.*

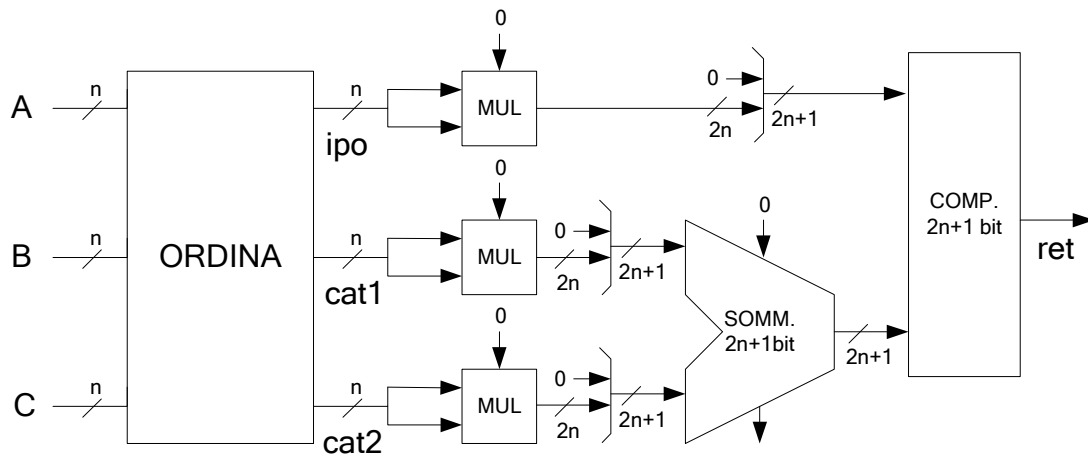
**Nota:** Si descriva esplicitamente qualunque rete non descritta a lezione.

**Domanda facoltativa:** Si sintetizzi la rete MAX3

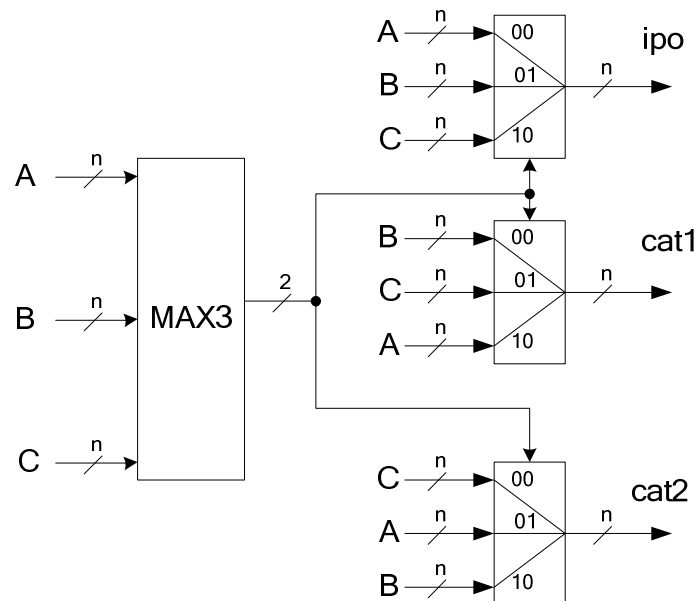
### 6.2.1 Soluzione

L'uscita *area* sta su  $2n-1$  bit.

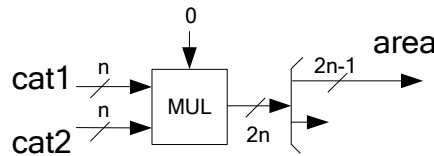
Per produrre l'uscita *ret*, bisogna stabilire se il triangolo è rettangolo o meno. Un triangolo è rettangolo quando i suoi lati verificano il teorema di Pitagora: ciò significa che deve esistere una permutazione dei tre lati  $A, B, C$ , per cui  $L_1^2 + L_2^2 = L_3^2$ . Questo è possibile solo quando  $L_3$  è il lato più lungo dei tre. Per poter testare se il triangolo è rettangolo, quindi, è necessario decidere quale dei tre lati ha lunghezza massima. Questo può essere fatto usando la rete *MAX3* dell'esercizio precedente, nel seguente modo.



Dove la rete *ORDINA* pone sull'uscita *ipo* il valore  $\max(A, B, C)$ , e sulle altre due uscite gli altri due valori. *ORDINA* è fatta come segue:



La rete COMP è un comparatore a  $2n+1$  bit, che può essere realizzato con una barriera di  $2n+1$  porte XOR seguiti da una porta NOR a  $2n+1$  ingressi, o dal NOR delle uscite di un sottrattore a  $2n+1$  bit. Per quanto riguarda la produzione dell'uscita *area*, la rete è la seguente:



Ovviamente, se il triangolo non è rettangolo, l'uscita *area* contiene un valore che non ha nulla a che vedere con l'area del triangolo stesso.

### 6.3 Esercizio – Febbraio 2005 (numeri naturali)

Sia  $A = (a_{n-1} \dots a_0)_4$  un numero naturale rappresentato su  $n = 5$  cifre in base quattro. Realizzare un circuito che prenda in ingresso le cifre di  $A$  e produca in uscita 0 se  $A$  è multiplo di tre, 1 altrimenti.

Si realizzi il circuito osservando che  $|A|_3 = 0$  se e solo se  $\left| \sum_{i=0}^{n-1} a_i \right|_3 = 0$ ,

**PARTE FACOLTATIVA:** Dimostrare il precedente risultato.

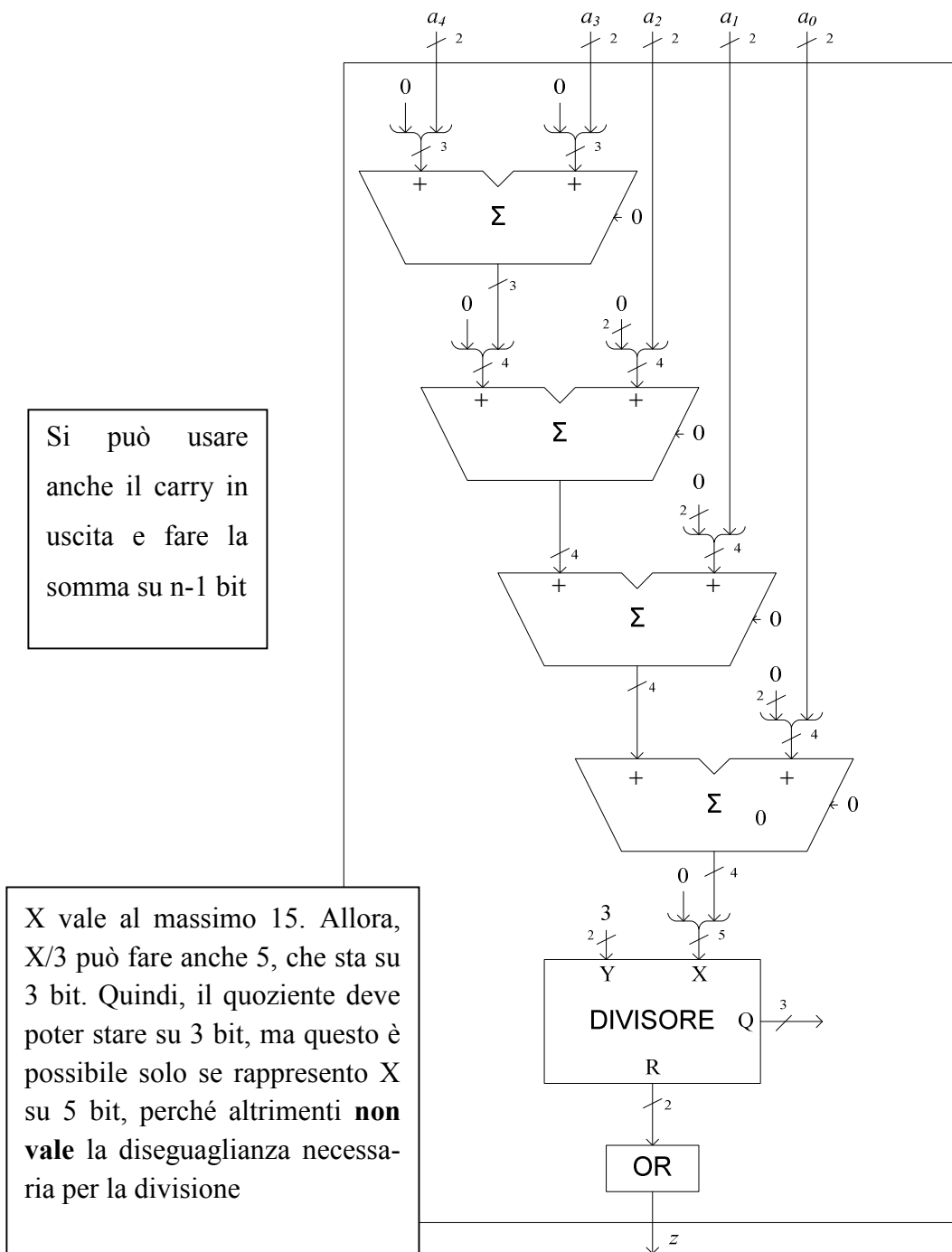
#### 6.3.1 Soluzione

Prima la parte facoltativa. Il risultato è dimostrato dalla seguente sequenza di uguaglianze:

$$\begin{aligned}
 |A|_3 &= \left| \sum_{i=0}^{n-1} a_i \cdot 4^i \right|_3 = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot (1+3)^i \right|_3 \\
 &= \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[ \sum_{k=0}^i \binom{i}{k} \cdot 1^{i-k} \cdot 3^k \right] \right|_3 = \left| a_0 + \sum_{i=1}^{n-1} a_i \cdot (1+3 \cdot \gamma_i) \right|_3 = \left| \sum_{i=0}^{n-1} a_i \right|_3 \\
 &= \left| a_0 + \sum_{i=1}^{n-1} a_i + 3 \cdot \sum_{i=1}^{n-1} a_i \cdot \gamma_i \right|_3 = \left| \sum_{i=0}^{n-1} a_i \right|_3
 \end{aligned}$$

Dove il terzultimo passaggio dipende dal fatto che tutti i termini dello sviluppo del binomio di Newton, tranne quello per  $k = 0$ , sono numeri naturali che contengono un fattore 3 a moltiplicare.

Un possibile circuito che soddisfa la specifica è il seguente:



Le cifre della rappresentazione di  $A$  sono codificate usando la rappresentazione in base due – su due bit - dei valori delle cifre stesse. Si noti che la condizione  $X < \beta^n Y$ , che esprime la garanzia di correttezza di funzionamento del divisore, è sempre rispettata, essendo, in questo caso,  $Y = 3$ ,  $\beta = 2$ ,  $n = 3$  e  $X \leq 15$ .

#### 6.4 Esercizio – Giugno 2004 (numeri interi)

Sia  $a \leftrightarrow A$  in complemento alla radice su sei cifre in base due. Progettare un circuito, **senza fare uso di moltiplicatori**, che riceve in ingresso  $A$  e produce in uscita  $B \leftrightarrow b$  in complemento alla

radice su  $n$  cifre tale che  $b = 7 - 5a$ . Esempificare il comportamento del circuito (ovvero, mostrare i livelli logici su ingressi, uscite, e collegamenti interni del circuito progettato) quando  $a = 14$ .

**Suggerimento:** Ridefinire in maniera equivalente la relazione aritmetica fra  $a$  e  $b$  in modo da rendere non necessario l'uso di moltiplicatori.

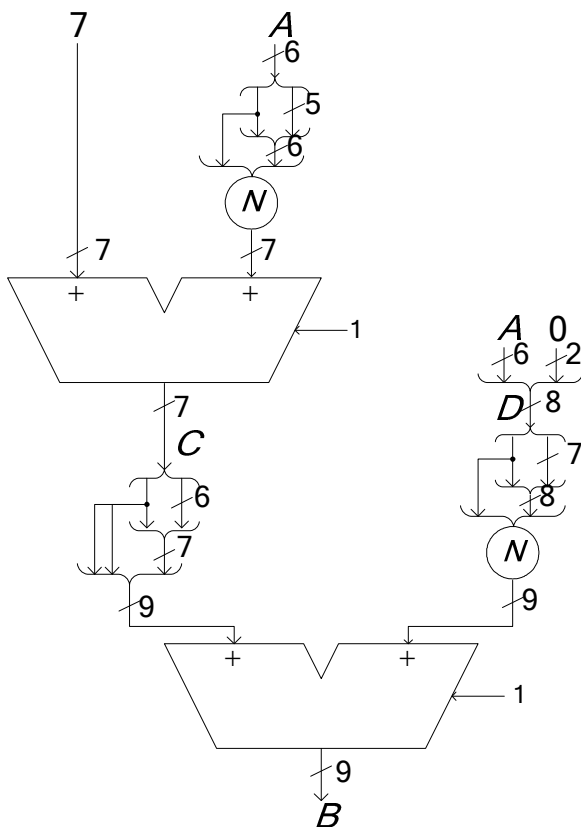
### 6.4.1 Soluzione

Poiché  $-32 \leq a < 31$ , si ha  $-148 < b \leq 167$ . Sono quindi necessarie  $n = 9$  cifre per rappresentare  $b$  in complemento alla radice.

Una possibile soluzione è riportata nello schema in figura, ottenuto ridefinendo la relazione aritmetica fra  $a$  e  $b$  come  $b = (7 - a) - 4a$ , ovvero  $b = c - d$  con  $c = 7 - a$  e  $d = 4a$ .

Calcolate allora le rappresentazioni (vedi poco sotto)  $C$  e  $D$  di  $c$  e  $d$ , si ha  $B = C + \bar{D} + 1$ .

Le rappresentazioni (su 9 cifre)  $C$  e  $D$  si ottengono da  $A$  utilizzando proprietà note della rappresentazione in complemento. In particolare, la differenza fra due numeri, da cui  $C = 7 + \bar{A} + 1$ , ed il prodotto per una potenza della radice, da cui  $D = 2^2 A$ .



Relativamente all'esempio specificato, risulterà:

$$A = 'B001110$$

$$C = 'B1111001$$

$$D = 'B00111000$$

$$B = 'B111000001$$

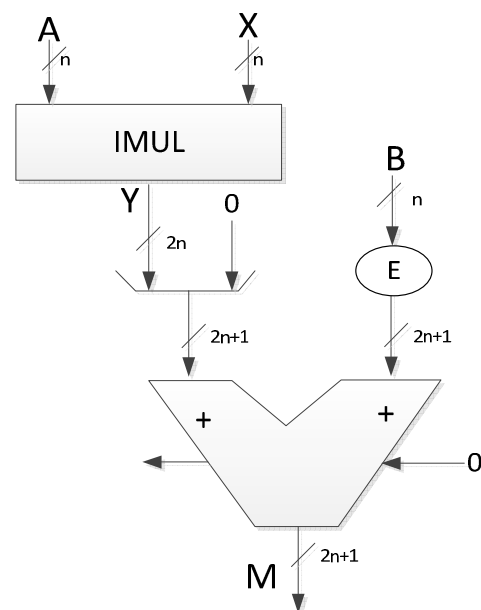
### 6.5 Esercizio – Settembre 2012 (numeri interi)

Si consideri un piano cartesiano a coordinate intere, rappresentate su  $n$  bit in complemento alla radice. Siano  $a$ ,  $b$ ,  $c$ , i tre coefficienti di una parabola e sia  $x$  un'ascissa sul piano. Si assuma che tutti i numeri sopra elencati siano rappresentabili.

- 1) Sintetizzare la rete combinatoria che prende in ingresso le rappresentazioni  $A, B, C, X$  dei 4 numeri  $a, b, c, x$  e produce in uscita su ? bit la rappresentazione  $M$  del numero  $m$ , coefficiente angolare della tangente alla parabola nel punto di ascissa  $x$ .
- 2) Assumendo  $n > 2$ , descrivere e sintetizzare la rete combinatoria che produce i due bit meno significativi di  $M, m_1 m_0$ . Si consiglia di seguire i seguenti passi:
  - a. Individuare da quali variabili logiche dipendono  $m_1 m_0$
  - b. risalire all'indietro da queste fino agli ingressi che le producono.
  - c. Scrivere la mappa di Karnaugh.

### 6.5.1 Soluzione

1) Il risultato da ottenere è il seguente:  
 $m = 2ax + b$ . Il numero minimo di bit richiesto per  $M$  è  $2n+1$ , come si può verificare meccanicamente con un calcolo semplice. La rete che produce il risultato è quella a destra (che non sente l'ingresso  $C$ ).



2) Come si vede dalla figura,  $m_0$  dipende *soltanto* dal bit meno significativo dei due ingressi del sommatore. Uno di questi due bit vale zero, quindi  $m_0 = b_0$ . Per quanto riguarda  $m_1$ , questo dipende da  $b_1$  e dal bit meno significativo di  $Y, y_0$ . Per capire quanto vale  $y_0$  in funzione degli ingressi, è necessario osservare che il bit meno significativo di un numero intero vale 0 se il numero è pari ed 1 se è dispari, indipendentemente dal segno del numero. Quindi,  $y_0$  se e solo se *entrambi* i numeri  $a$  e  $x$  sono dispari (altrimenti il loro prodotto è pari), cioè  $y_0 = a_0 \cdot x_0$ . Quindi,  $m_1 = b_1 + a_0 \cdot x_0$ . La rete combinatoria richiesta ha quindi tre ingressi,  $a_0, x_0, b_1$ , ed un'uscita,  $m_1$ , ed è la seguente:

		$m_1$			
		$a_0x_0$	00	01	11
$b_1$	0	0	0	1	0
	1	1	1	0	1

La sintesi a costo minimo è quindi  $m_1 = b_1 \cdot \overline{a_0} + b_1 \cdot \overline{x_0} + a_0 \cdot x_0 \cdot \overline{b_1}$ , oppure  $m_1 = b_1 \oplus a_0 \cdot x_0$ .