Appunti sull’Aritmetica dei Calcolatori

Giovanni Stea

a.a. 2019/20

Ultima modifica: 30/10/2019

Sommario

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.
- 21/10/15: Corretti errori nel testo e negli esercizi.
- 28/10/15: Modifiche cosmetiche varie.
- 27/10/16: Modifiche cosmetiche ulteriori.
- 15/11/17: Corretti alcuni errori nel testo (detti a lezione), modifiche cosmetiche varie.
- 25/10/18: Tolti i riferimenti alle *alee* negli esercizi (le alee non fanno più parte del programma), corretti alcuni errori (detti a lezione), modifiche all'impaginazione.
- 05/11/18: Migliorate alcune figure.
- 11/10/19: Restyling e semplificazione di alcuni argomenti.
- 30/10/19: modifiche cosmetiche e correzione di alcuni errori (detti a lezione).

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 β **simboli**, detti **cifre**, a ciascuno dei quali è associato **un numero naturale compreso tra** 0 e $\beta - 1$.
3) Una **legge di rappresentazione** che fa corrispondere ad ogni **sequenza di cifre** un numero naturale.

Dato il numero $A \in \mathbb{N}$, lo **rappresento** in base β 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 **differente** 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.

<u>Esempio</u>: 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 (talvolta verrà scritto in lettere: "dieci").
- la sua **rappresentazione** in una qualche base (ad esempio base dieci) 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

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

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

$$(1A2F)_{16} = (F)_{16} \cdot \left(16^0\right)_{10} + (2)_{16} \cdot \left(16^1\right)_{10} + (A)_{16} \cdot \left(16^2\right)_{10} + (1)_{16} \cdot \left(16^3\right)_{10}$$
$$= (15)_{10} \cdot \left(16^0\right)_{10} + (2)_{10} \cdot \left(16^1\right)_{10} + (10)_{10} \cdot \left(16^2\right)_{10} + (1)_{10} \cdot \left(16^3\right)_{10}$$
$$= 15 \cdot 1 + 2 \cdot 16 + 10 \cdot 256 + 1 \cdot 4096 = 6703$$

Esempio: sistema numerico binario

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

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

1.1 Teorema della divisione con resto

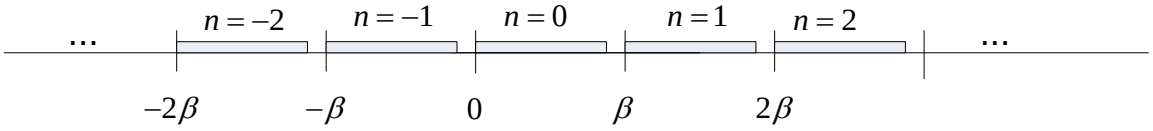
Data una base di rappresentazione, posso **sempre** rappresentare un numero in quella base? La sua rappresentazione, data una base, è **unica**? Come faccio a **trovarla**? La risposta alle tre domande passa per il seguente teorema:

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 $r \in \mathbb{N}$, $0 \leq r < \beta$, 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 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 $\beta - 1$**. Inoltre, $x \in \mathbb{N} \Rightarrow q \in \mathbb{N}$. Se la divisione è tra naturali, anche il quoziente è un numero naturale.

□

Diamo a q il nome di **quoziente**, e ad r il nome di **resto** della divisione di x per β. D'ora in avanti li indichiamo come:

Parte intera inferiore della frazione x/β → $q = \left\lfloor \frac{x}{\beta} \right\rfloor$, $r = |x|_\beta$ ← x modulo β

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) $|x + k \cdot \alpha|_\alpha = |x|_\alpha$, $k \in \mathbb{Z}$. 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 α di $x + k \cdot \alpha$.

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

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

$$|x \cdot y|_\alpha = \left|(|x|_\alpha + q_x \cdot \alpha) \cdot \left(|y|_\alpha + q_y \cdot \alpha\right)\right|_\alpha =$$
$$= \left||x|_\alpha \cdot |y|_\alpha + \left(|x|_\alpha \cdot q_y \cdot \alpha\right) + \left(|y|_\alpha \cdot q_x \cdot \alpha\right) + q_x \cdot q_y \cdot \alpha^2\right|_\alpha = ||x|_\alpha \cdot |y|_\alpha + k \cdot \alpha|_\alpha$$

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 β, 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**):

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

Mi fermo 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 il naturale A in base β. Dimostriamolo:
Sostituendo a q_i la propria definizione, si ottiene:

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

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**. Quella appena descritta è una maniera **algoritmica** di trovare le cifre della rappresentazione di un numero naturale in una qualunque base.

1.2.1 Rappresentazione su un numero finito di cifre

Se ho a disposizione n cifre in base β, potrò rappresentare $\beta \cdot \beta \cdot \dots \beta = \beta^n$ diversi numeri naturali. Visto che devo rappresentare un intervallo di numeri che include anche lo zero, il numero naturale **più grande** che posso rappresentare sarà $\beta^n - 1$. Qual è la rappresentazione di questo numero?
In base 10, il numero più grande rappresentabile su n cifre è 999…99. In generale, in base β, è quello che ottengo quando tutte le cifre hanno **valore massimo, cioè $a_i = \beta - 1$.** Infatti, si ottiene:

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

Questa è **una** rappresentazione di $\beta^n - 1$, quindi è **la** rappresentazione di $\beta^n - 1$.
Per calcolare il numero di cifre n richieste per rappresentare il numero A in base β posso osservare che $\beta^n \geq A + 1 \iff n = \lceil \log_\beta (A+1) \rceil$.

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 β. Si dimostri che, dato γ sottomultiplo di β, $|A|_\gamma = |a_0|_\gamma$. 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 β, $\beta > 2$, dimostrare che $|A|_\gamma = 0$ se e solo se $\left|\sum_{i=0}^{n-1} a_i\right|_\gamma = 0$, dove γ 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 β 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 β, generalmente pari a **due**.

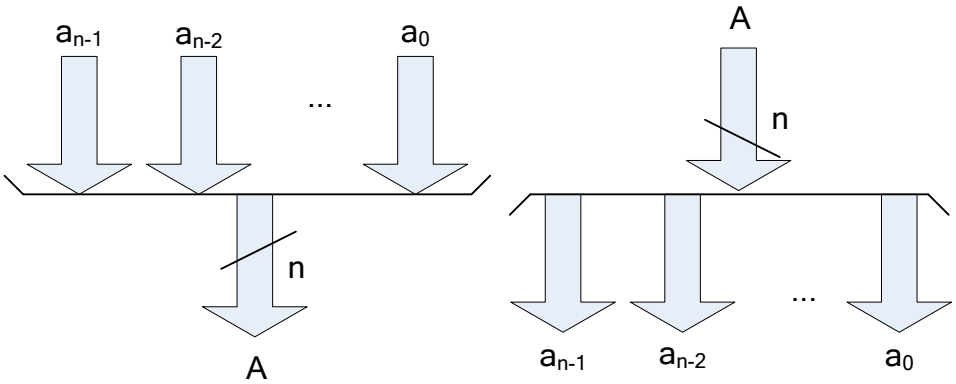
Che tipo di reti logiche saranno quelle che andrò a costruire? Saranno **reti combinatorie**, in quanto ad ogni stato di ingresso (operandi di un'operazione) corrisponderà **uno** stato di uscita (risultato dell'operazione).

Nel seguito, faremo spesso quanto segue:

1. prenderemo 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 leva sulle **proprietà della notazione posizionale**, di scomporre tali operazioni in **blocchi elementari**.
3. Dettaglieremo le reti logiche (a livello di porte elementari) che implementano i blocchi elementari **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 sequenza di n cifre in base β qualunque, scriveremo:

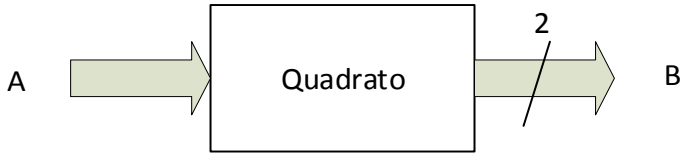


Ciascuna di queste doppie frecce rappresenta una cifra in base β generica. Con questa notazione potrò fare diagrammi come il seguente:

Esempio

Rete che prende in ingresso un numero a una cifra in base β e restituisce in uscita il suo quadrato.

È abbastanza semplice rendersi conto che, dato $A < \beta$, sarà $A^2 < \beta^2$, e quindi l'uscita sarà su due cifre, **qualunque sia la base** β.



Come sarà fatta questa rete combinatoria, in termini di porte logiche, dipenderà certamente dalla base. Il fatto che l'uscita sta su due cifre, invece, non dipende dalla base.

Come sappiamo, le reti combinatorie prendono in ingresso **variabili logiche** e restituiscono in uscita **variabili logiche**. Una variabile logica può essere messa in corrispondenza biunivoca con **una cifra in base** $\beta = 2$. Quando si lavora in base $\beta \neq 2$, è necessario **codificare le singole cifre** in termini di variabili logiche.

Per qualunque base $\beta \neq 2$, la scelta più immediata (ma non è l'unica possibile) è codificare una cifra in base β con il corrispondente numero in base 2. Ad esempio, per la base 10, la codifica BCD (Binary-coded Decimal) è quella più usata, ed è riportata nella tabella a fianco:

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

Il fatto che le cifre debbano essere codificate su variabili logiche è spesso fonte di confusione. Assumendo che la rete "quadrato" sopra descritta **operi su numeri in base 10**, avremo che gli stati di ingresso possibili sono 0, 1, 2, …, 8, 9, a cui corrispondono stati di uscita 00, 01, 04, …, 64, 81 (su due cifre in base 10). Se quindi dovessimo scrivere una tabella che lega ingressi ed uscite in base 10, scriveremmo:

A	$b_1 b_0$
0	00
1	01
2	04
…	…
8	64
9	81

BCD

BCD(A)	BCD(b_1) BCD(b_0)
0000	0000 0000
0001	0000 0001
0010	0000 0100
…	…
1000	0110 0100
1001	1000 0001

Per sintetizzare la rete "quadrato" che opera in base 10, dovremo sintetizzare una rete combinatoria che fa corrispondere alle **codifiche BCD degli ingressi** le **codifiche BCD delle uscite**, come nella tabella a destra, in cui ogni cifra in base 10 (singolarmente) è codificata secondo la tabella BCD.
Si noti che la rete **opera in base 10**. Infatti, risponde 1000 0001 (81) quando l'ingresso è 1001 (9), e questa è la risposta corretta solo se si interpretano le variabili logiche di uscita come codifiche di cifre in base 10. Se la rete calcolasse il quadrato in base 2, all'ingresso 1001 dovrebbe far corrispondere l'uscita 1010001 (81 in base 2), invece che 1000 0001.
Nel seguito, ci preoccuperemo molto poco delle codifiche, e lavoreremo sulle **cifre**. Infatti, la maggior parte delle operazioni aritmetiche si possono esprimere in termini di **relazioni tra cifre degli operandi**, indipendentemente dalla base in cui si lavora. Parleremo di codifica soltanto quando dovremo **sintetizzare le reti combinatorie**, ma in questo caso lavoreremo quasi sempre in base 2, dove una cifra è anche una variabile logica.

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 β su n cifre, $0 \leq A < \beta^n$, definisco **complemento di A (in base β 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 β 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 loro complemento è **diverso**.

Voglio sintetizzare una rete che, date in ingresso le **cifre della rappresentazione di *A*** in una qualunque base, dia in uscita quelle di $\overline{A}$. Come faccio a trovare **le cifre** di $\overline{A}$ 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 $0 \leq \overline{A} < \beta^n$, quindi sono certo che $\overline{A}$ è rappresentabile su n cifre. Dalla definizione ricavo che:

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

$$\uparrow$$

$$\beta^n - 1$$

Ma:

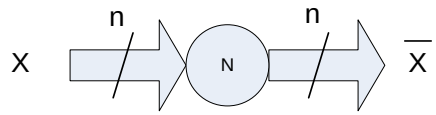
- $\beta - 1 - a_i$ è una cifra in base β, in quanto compresa tra 0 e $\beta - 1$.

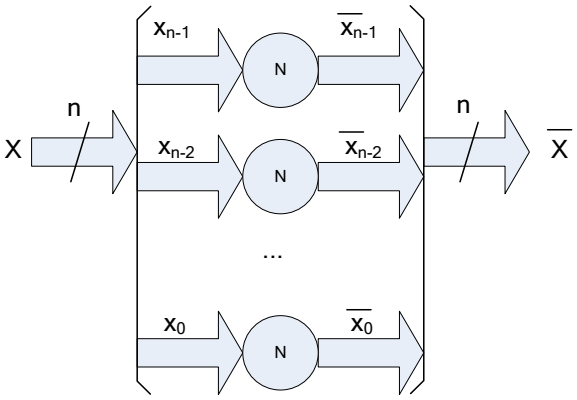
- $\beta - 1 - a_i$, dalla definizione, è il **complemento del numero naturale a_i su una cifra in base β**.

Quindi, le cifre in base β della rappresentazione di $\overline{A}$ sono $\overline{a_i} = \beta - 1 - a_i$. Ogni cifra di $\overline{A}$ si ottiene complementando la corrispondente cifra di A. $\overline{A} \equiv (\overline{a_{n-1}}\ \overline{a_{n-2}}\ \dots\ \overline{a_1}\ \overline{a_0})_\beta$. Questa è una proprietà della notazione posizionale, valida in qualunque base.

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 β su n cifre.





Basta quindi saper sintetizzare una rete che fa il complemento di **una singola cifra**. Come sarà fatta tale rete dipende dalla base. In base $\beta = 2$ è estremamente semplice. Infatti, è la rete elementare che:

- se la cifra è 1, ha uscita 0;
- se la cifra è 0, ha uscita 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.

2.1.2 Esercizio (da fare a casa)

1) Descrivere e sintetizzare un circuito di complemento per numeri naturali ad n cifre in base 10, con codifica BCD.

2) Si supponga di codificare una cifra a in base 10 come la cifra $a + 3$ in base 2 (codifica "**eccesso 3**"). Descrivere e sintetizzare un circuito di complemento per numeri naturali ad n cifre in base 10 così codificati.

Soluzione

2.2 Moltiplicazione e divisione per una potenza della base

Devo eseguire una moltiplicazione (divisione) per β^k.

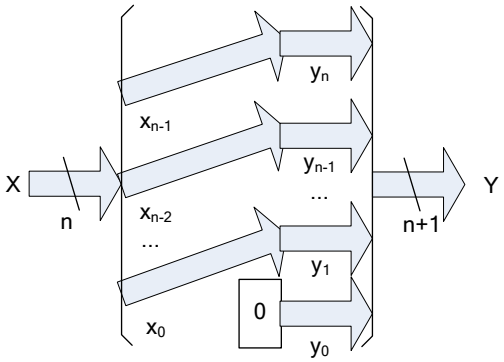
Domande:

- Quando, in base 10, devo calcolare 25×1000, 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.

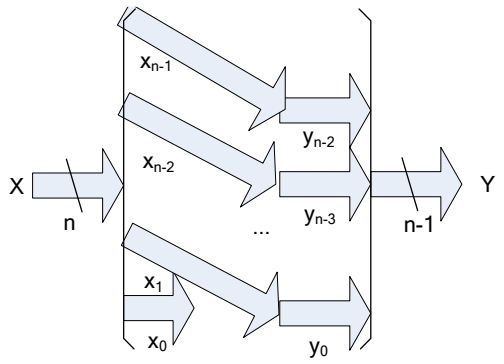
Tutto questo perché sto moltiplicando e dividendo per una **potenza della base** in cui lavoro (la base 10, appunto). Questa è un'altra **proprietà della notazione posizionale,** valida in qualunque base. Vediamo di dare una dimostrazione formale delle proprietà che abbiamo appena intuito.

In qualunque base:

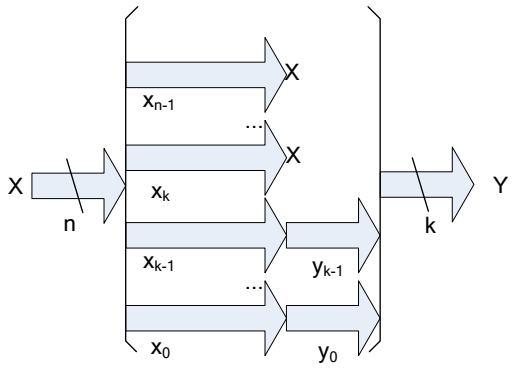
- **Moltiplicare** un numero su n cifre per β^k significa costruire un nuovo numero di $n + k$ cifre, di cui le k meno significative valgono **zero**. Nella figura abbiamo la rete che esegue la moltiplicazione per β^1:



- Il **quoziente** della divisione di un numero su n cifre per β^k è un numero costituito dalle $n - k$ cifre più significative del numero di partenza. Nella figura abbiamo la rete che calcola il quoziente della divisione per β^1:



- Il **resto** della divisione di un numero su n cifre per β^k è un numero costituito dalle k cifre meno significative del numero di partenza.

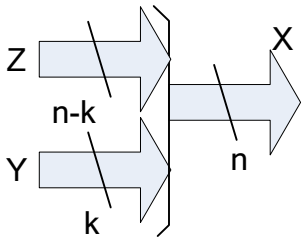


Le reti che implementano queste operazioni, dette reti di **shift** (sinistro per la moltiplicazione, destro per la divisione) sono di **complessità nulla**, così come è di complessità nulla il procedimento mentale per ottenere il risultato.

D'ora in avanti, sarà considerato **errore** usare porte logiche per moltiplicare e dividere un numero per β^k. Quanto scritto sopra ha una conseguenza non ovvia:

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.



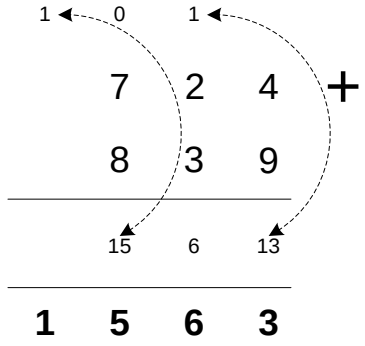
2.3 Estensione di campo

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 in qualunque base con i numeri **naturali** (**attenzione**: per gli interi sarà diverso).

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 $X^{EST} \equiv (0\ x_{n-1}x_{n-2}\dots x_0)_\beta$. 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, 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**.

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.

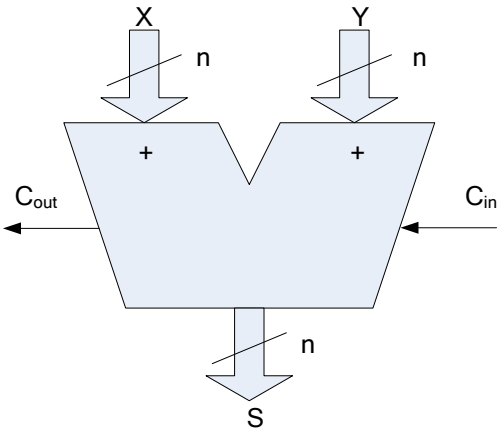
Dati X, Y in base β su n cifre, quindi $0 \leq X, Y \leq \beta^n - 1$, e dato C_{in}, $0 \leq C_{in} \leq 1$, voglio calcolare il numero $Z = X + Y + C_{in}$. Il termine C_{in}, che a prima vista sembra essere una complicazione inutile, gioca invece un ruolo fondamentale in quanto consente di rendere 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**, ma potrebbe **non essere rappresentabile su n cifre**. La $n+1^{ma}$ cifra è il riporto dell'ultima somma, e quindi può essere soltanto 0 o 1.

L'algoritmo sopra richiamato e le proprietà appena esposte sono **indipendentemente dalla base**.

Quindi: **la somma di due numeri naturali espressi in base β 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 β, l'$(n+1)^{ma}$ delle quali, detta *riporto uscente*, può essere soltanto <u>zero</u> o <u>uno</u>**.

Posso quindi disegnare ingressi ed uscite di un circuito che calcola la somma di due numeri in base β su n cifre, detto **sommatore in base β a n cifre**. Il circuito è fatto così:



I riporti sono frecce **singole** (sono var. logiche, non cifre in base β)

Il riporto uscente va interpretato 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.

Richiamo sull'Assembler: L'istruzione macchina ADD setta il CF quando il risultato non è un numero naturale rappresentabile sul numero di cifre degli operandi.

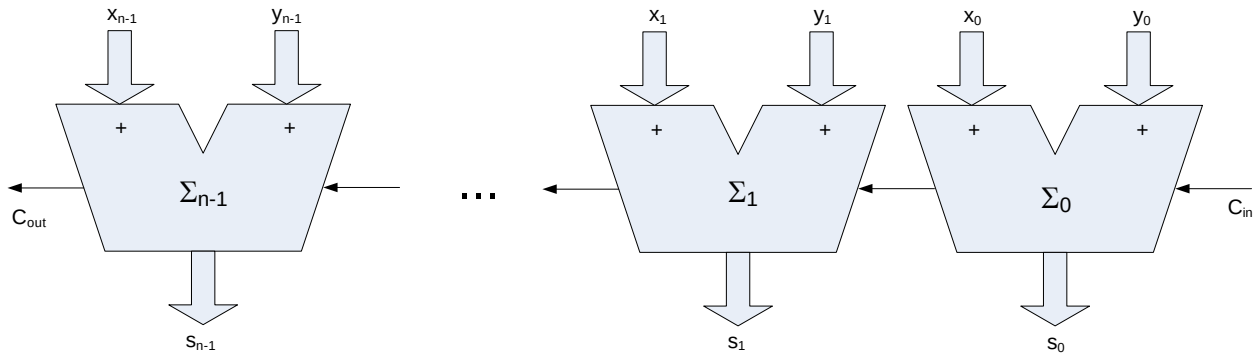
Attenzione al **dimensionamento** del sommatore: il numero di cifre di **entrambi gli ingressi e dell'uscita è lo stesso**, e non può essere diversamente (altrimenti è **errore**). Infatti, nella ALU di un processore, questi circuiti lavorano su operandi (contenuti in registri e celle di memoria) che stanno su un numero prefissato di bit. Le varie versioni dell'istruzione ADD prevedono addendi della stessa lunghezza (8, 16, 32 bit) e sostituiscono uno degli addendi con il risultato.
Quindi:

- se gli addendi sono su un numero di cifre differente (ad esempio, n ed m cifre, con $n > m$, dovete **estendere quello ad m cifre** (inserendo $n - m$ zeri in testa);
- se gli addendi sono **su n cifre** e volete che la somma S sia **sempre** rappresentabile, dovete usare un sommatore ad $n + 1$ **cifre**, ed estendere gli ingressi su $n + 1$ cifre.

2.4.1 Full Adder in base 2

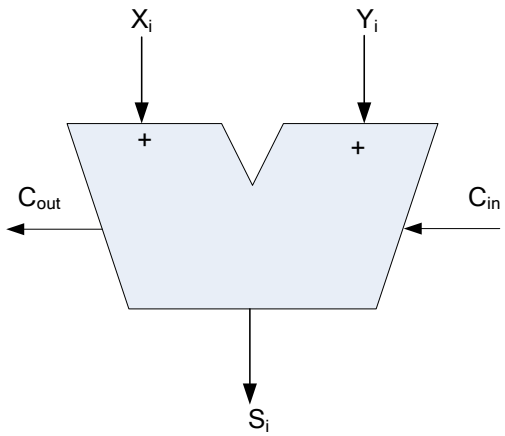
Come faccio a sintetizzare il circuito sommatore disegnato sopra? Posso far leva sul fatto che **la somma** in base β **su n cifre** può essere scomposta in **somme in base β su una sola cifra**, purché:

- esegua le somme andando dalla cifra meno significativa alla più significativa;
- sia in grado di **propagare il riporto** tra uno stadio di somma ed il successivo.



Il montaggio che si ottiene si chiama **ripple carry** (propagazione del riporto). I suoi componenti (sommatori ad una cifra) prendono il nome di **full adder**.

Come effettivamente si realizza un full adder dipende dalla base. 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

s_i

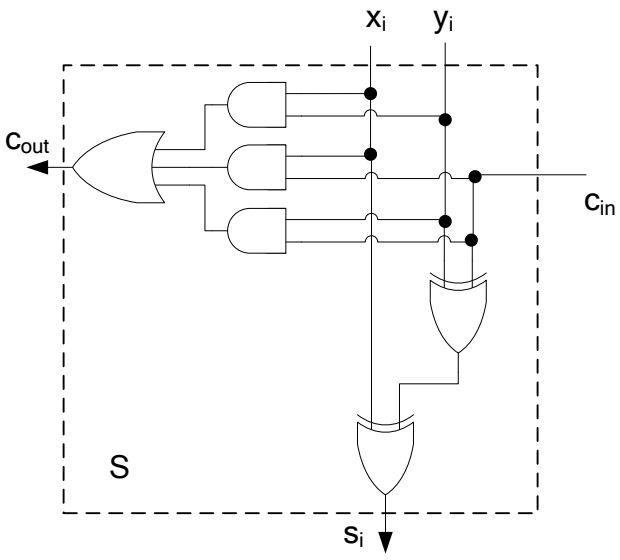
c_{in} \ x_iy_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

c_{out}

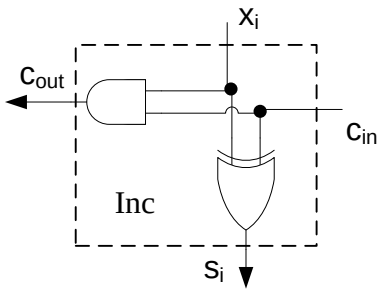
c_{in} \ x_iy_i	00	01	11	10
0	0	0	1 (A)	0
1	0	1 (B)	1	1 (C)

- 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.



2.4.2 Incrementatore



Un incrementatore è un circuito combinatorio che somma Cin (che vale 0 o 1) ad un numero dato. Possiamo pensare che tale incremento sia un caso particolare di **somma con riporto tra due addendi ad n cifre**, in cui **uno dei due addendi è nullo**. Pertanto, lo possiamo realizzare scomponendo il tutto in n **moduli full adder**. Però questi 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`, anche se generano lo stesso risultato, sono **istruzioni diverse**. In particolare, la seconda potrebbe essere più veloce della prima (era così quando i calcolatori erano più lenti).

2.4.3 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 uscente.

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

- individuare e classificare gli implicanti principali
- trovare tutte le liste di copertura irridondanti
- scegliere la lista di costo minimo secondo il criterio a diodi

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

2.4.4 Esercizio (da fare a casa)

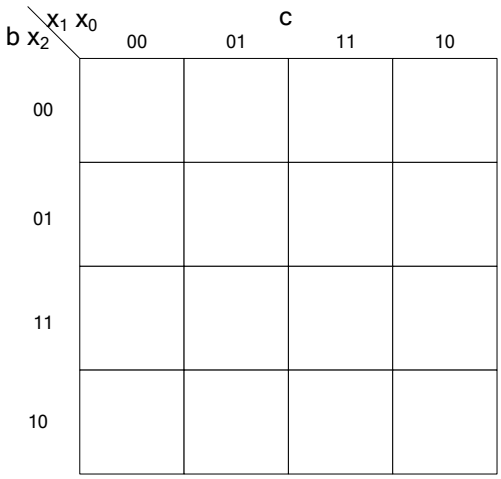
Sia data una rete combinatoria che: i) riceve in ingresso tre variabili x_2, x_1, x_0 che esprimono un numero naturale X ad una cifra in base 5 (in codifica 421) ed una *variabile di comando b*, e ii) produce in uscita tre variabili y_2, y_1, y_0 che esprimono un numero naturale Y ad una cifra in base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

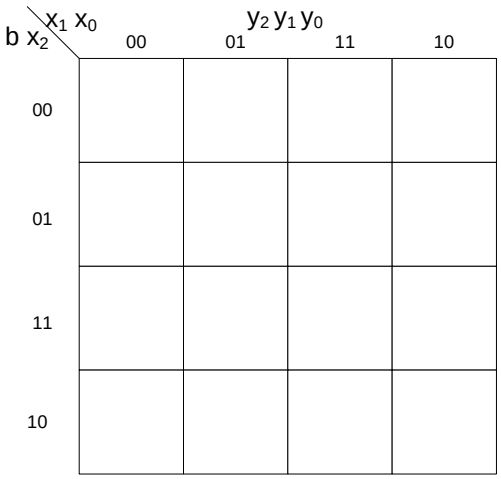
Il numero naturale Y è legato al numero naturale X dalla relazione

$$Y = \begin{cases} |2X|_5 & b = 0 \\ |X+1|_5 & b = 1 \end{cases}$$

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

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





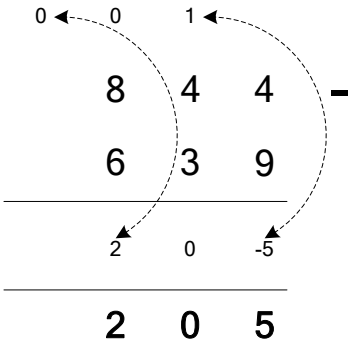
2) Sintetizzare la sottorete che genera y_0 a costo minimo sia a porte NAND sia a porte NOR

3) 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**.



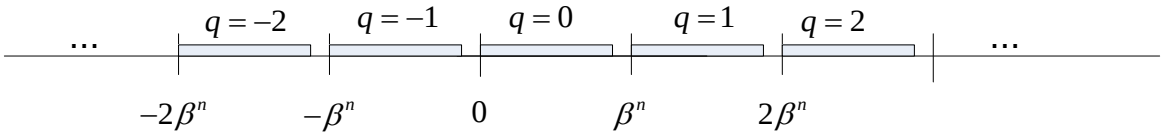
$\beta = 10$. L'algoritmo consiste in:

- 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 β 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**.

Per quanto riguarda la rappresentabilità, $-\beta^n \leq X - Y - b_{in} \leq \beta^n - 1$. 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). Come già fatto per la somma, scrivo Z come quoziente e resto di una divisione per β^n: $Z = X - Y - b_{in}$ diviso per β^n dà quoziente al minimo **-1**.



Pertanto, definisco:

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

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

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

Quindi: **la differenza *D* tra due numeri naturali in base β su *n* cifre, meno un eventuale prestito entrante, produce un numero che, se naturale, è sempre rappresentabile su *n* cifre in base β. 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, procedo come segue:

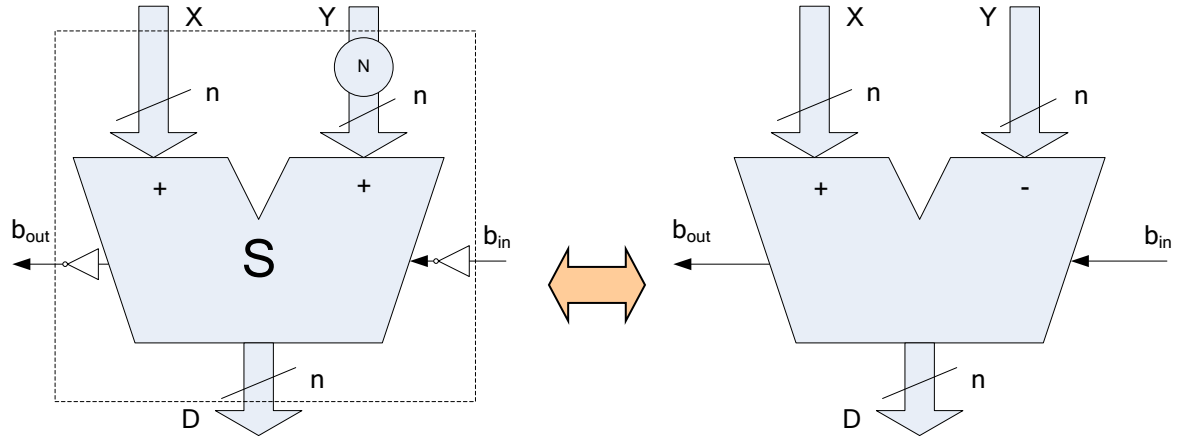
osservo che $Y + \overline{Y} = \beta^n - 1$ (definizione di **complemento**). Dal che derivo che $-Y = \overline{Y} - \beta^n + 1$. Sostituendo quest'ultima nell'espressione riquadrata sopra, si ottiene:

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

Come si interpreta questa equazione? Dicendo che:

- **la differenza tra *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 tirar fuori un circuito di decremento semplificato, etc.

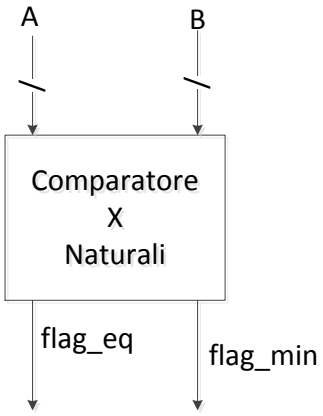
2.5.1 Comparazione di numeri naturali

I sottrattori vengono usati spesso come **comparatori**. Dati due numeri *naturali* A e B, se voglio sapere se $A < B$, basta che li sottragga e guardi il prestito uscente. Se $b_{out} = 1$, allora A è più piccolo. Si noti che questo tipo di comparazione vale **soltanto per i naturali** (per gli interi sarà diverso).

Inoltre, se voglio testare se due numeri sono **uguali**, posso comunque prendere l'uscita del sottrattore e testare se D è uguale alla codifica del numero (000…000). Assumendo che la cifra 0 in base β sia codificata con n variabili logiche pari a 0 (il che è vero nella maggior parte dei casi), questo significa passare l'uscita del sottrattore ad una **porta NOR** ad un opportuno numero di ingressi. In quest'ultimo caso, però, si fa prima a fare lo XOR bit a bit delle codifiche di ciascuna cifra, portando tutte le uscite in ingresso ad una NOR (la rappresentazione di un numero è comunque unica).

D'ora in avanti, posso usare la rete **comparatore per naturali**, che avrà due uscite:

- *flag_eq*, che vale 1 se i due numeri da comparare sono uguali,
- *flag_min*, che vale 1 se $A < B$.



2.6 Moltiplicazione

Dati:

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

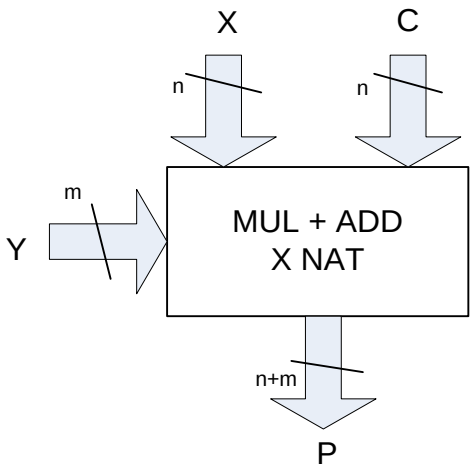
Voglio calcolare:

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

Ormai è chiaro 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 prodotti più semplici. Dalle nostre ipotesi 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 $\boldsymbol{n+m}$ **cifre**. Ciò detto, possiamo disegnare lo schema della rete che svolge la moltiplicazione, che si chiama **moltiplicatore con addizionatore per naturali**. Vediamo come si scompone in reti più semplici. Usiamo la consueta **tecnica di scomposizione del problema**, partendo dall'algoritmo che abbiamo imparato alle elementari.



		2	4	5	×
		1	3	1	
		2	4	5	+
	7	3	5		+
2	4	5			
3	**2**	**0**	**9**	**5**	

- 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.

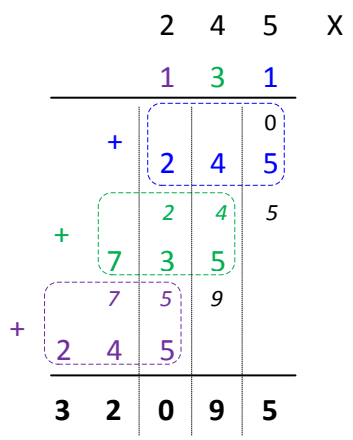
L'algoritmo e le proprietà sopra menzionate valgono indipendentemente dalla base. Dovremmo quindi essere in grado di:

- Moltiplicare **un numero ad n cifre per un numero ad una cifra**
- Sommare m addendi (traslandoli opportunamente), con riporto, per ottenere il risultato finale.

In realtà la somma di m addendi tutti insieme è un'operazione complessa da realizzare dal punto di vista circuitale. Conviene realizzare la somma in un altro modo, sfruttando il fatto che:

1) la somma è associativa.
2) **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 il valore della i-sima cifra del prodotto.

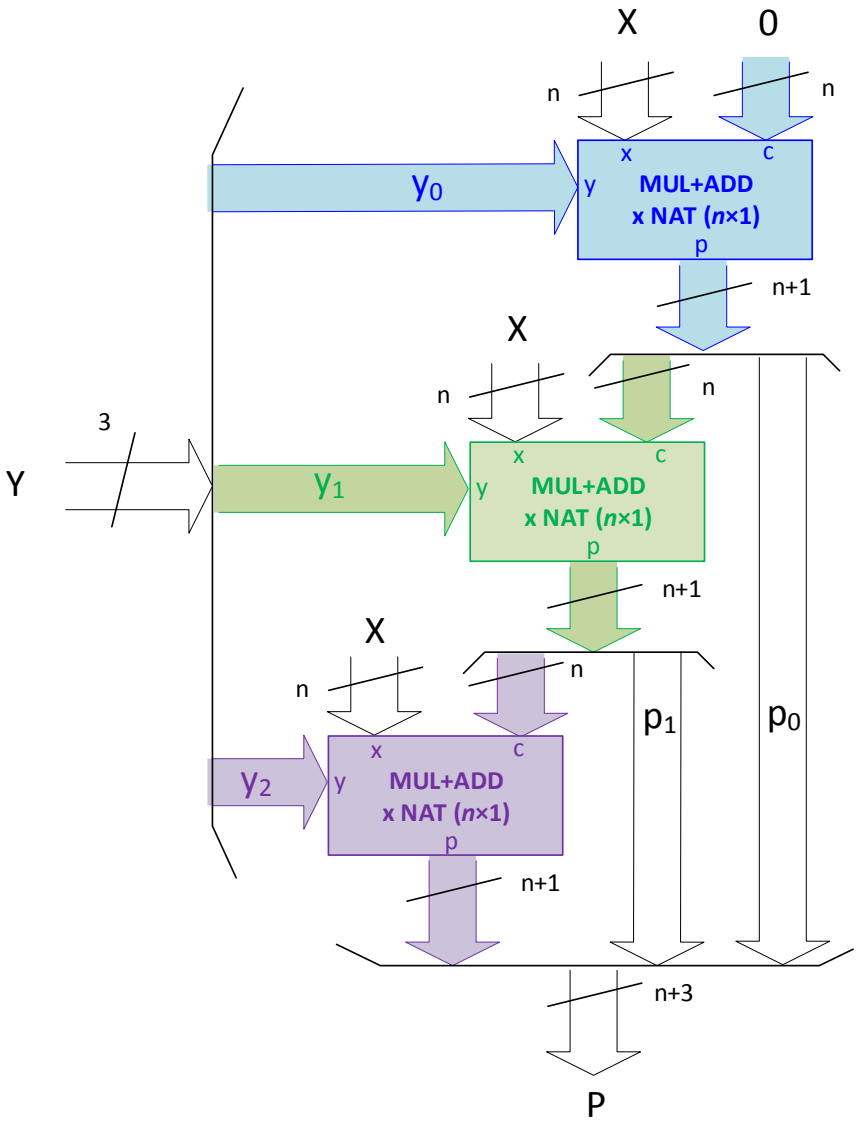
In questo modo, posso ottenere lo stesso risultato sommando soltanto due addendi alla volta. Vediamo come:



Il passo elementare dell'algoritmo consiste nel moltiplicare il numero X per **una cifra di Y,** sommando un termine C (che inizialmente è nullo). Il risultato che si ottiene viene suddiviso:

- la cifra meno significativa diventa la cifra i-sima del prodotto.
- le altre cifre diventano **il nuovo termine da sommare** per il prossimo passo.

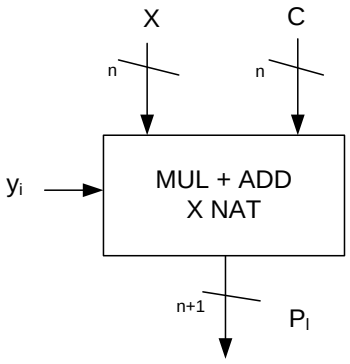
Alla fine, si ottiene il prodotto **concatenando** tutti i risultati. In questo modo si fanno solo moltiplicazioni su $n \times 1$ cifra e si sommano sempre soltanto due addendi (su $n+1$ cifre). Per via circuitale, il tutto funziona come nel diagramma della figura, dove $m = 3$ per semplicità di disegno.



Posso quindi realizzare la moltiplicazione utilizzando soltanto **moltiplicatori (con addizionatore) ad $n \times 1$ cifra**.

2.6.1 Moltiplicatore con addizionatore nx1 in base 2

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

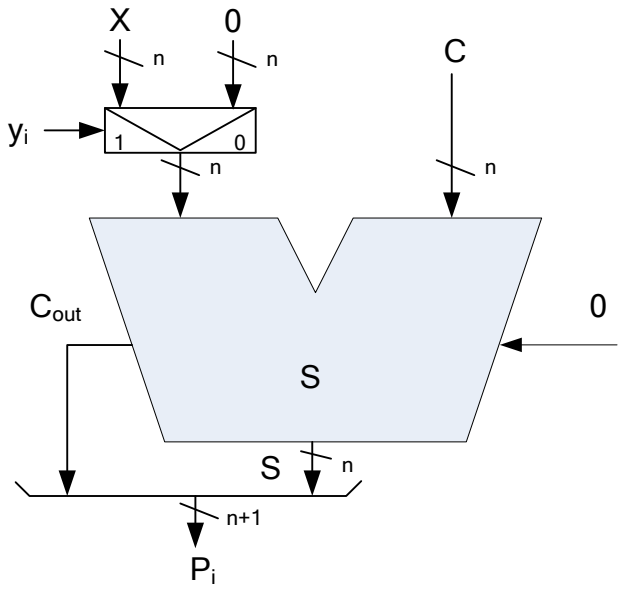


Il risultato che deve uscire da qui è:

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

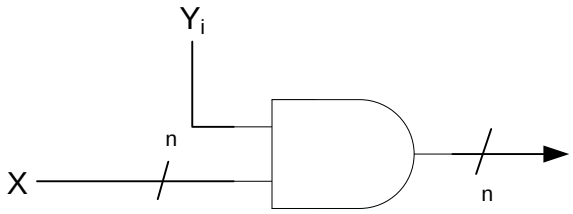
Attenzione al **dimensionamento**.

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**.



Il multiplexer a sinistra, in realtà, rappresenta n **multiplexer 2 to 1** in parallelo, ciascuno dei quali è relativo alla coppia corrispondente di bit.

Ciascuno di questi multiplexer, però può essere sostituito da una semplice **porta AND**.

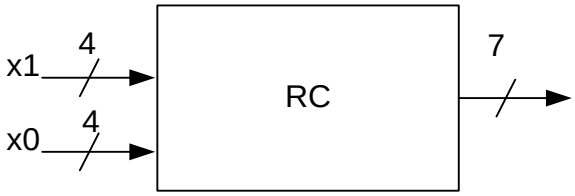


Richiamo sull'Assembler: L'istruzione Assembler **MUL** ha **un solo operando**. A seconda della **dimensione dell'operando**, seleziona l'altro operando (implicito: AL, AX, EAX), e mette il risultato in un destinatario implicito (AX, DX_AX, EDX_EAX). In pratica, abbiamo $n = m$, e quindi i circuiti che calcolano il prodotto calcolano risultati su $2n$ bit partendo da operandi a n bit.

2.6.2 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).

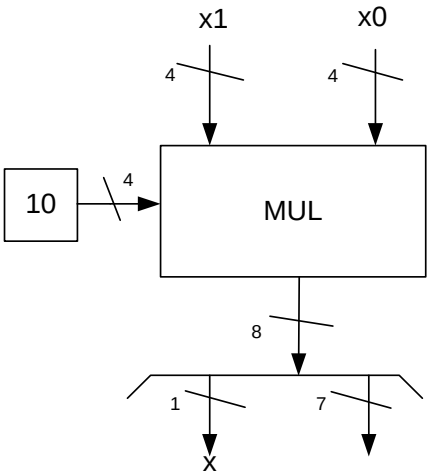
Devo realizzare un circuito fatto in questo modo



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:

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

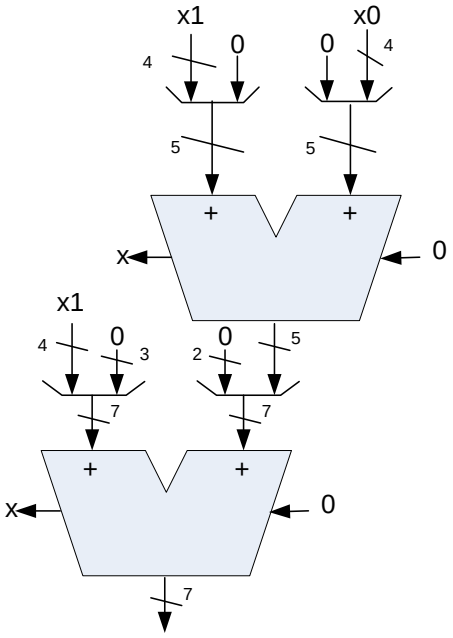
Ma un circuito che svolge questa operazione lo so sintetizzare:



Volendo, esiste un modo più furbo per ottenere lo stesso risultato. 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 bastano 2 full adder (a 7 e 5 bit).

2.7 Divisione

Dati:

- X numero naturale in base β **su $n + m$ cifre (dividendo)**, cioè tale che $0 \leq X \leq \beta^{m+n} - 1$
- Y numero naturale in base β **su m cifre (divisore)**, cioè 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. Prima di cominciare, osserviamo che una rete che fa le divisioni dovrà **necessariamente** essere dotata di un'uscita ulteriore di **non fattibilità**, che chiameremo ***no_div***. Infatti, la divisione non si può fare se $Y = 0$. Vedremo fra un attimo che l'uscita *no_div* ha anche altri utilizzi.

Assumendo che $Y > 0$, **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.**

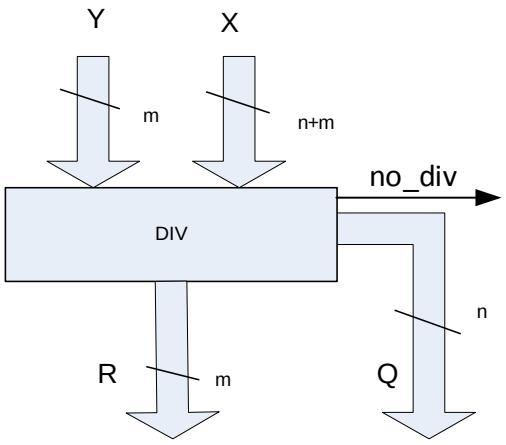
Voglio poter rappresentare il quoziente **su n cifre**. Assumere che Q stia su n cifre implica che:

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

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

Sotto quest'ipotesi non poso 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 un 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.

Il modulo divisore deve testare la **fattibilità** della divisione con le ipotesi date. Se il quoziente non sta su n cifre, **deve settare la variabile logica *no_div***, ad indicare che la divisione non è fattibile.

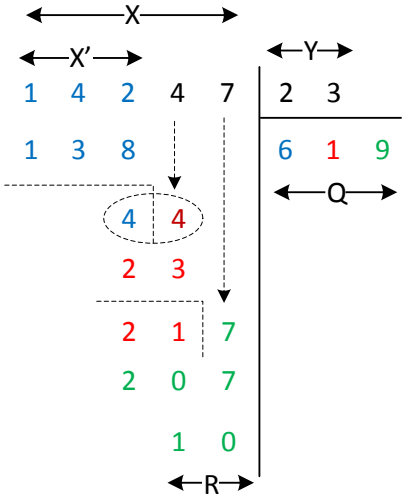


Come al solito, posso cercare di scomporre il tutto in moduli più semplici, traducendo in forma circuitale l'algoritmo che si usa per eseguire la divisione **a mano**.

Ricaviamo l'algoritmo partendo da un esempio di divisione fattibile con $m = 2, n = 3$.

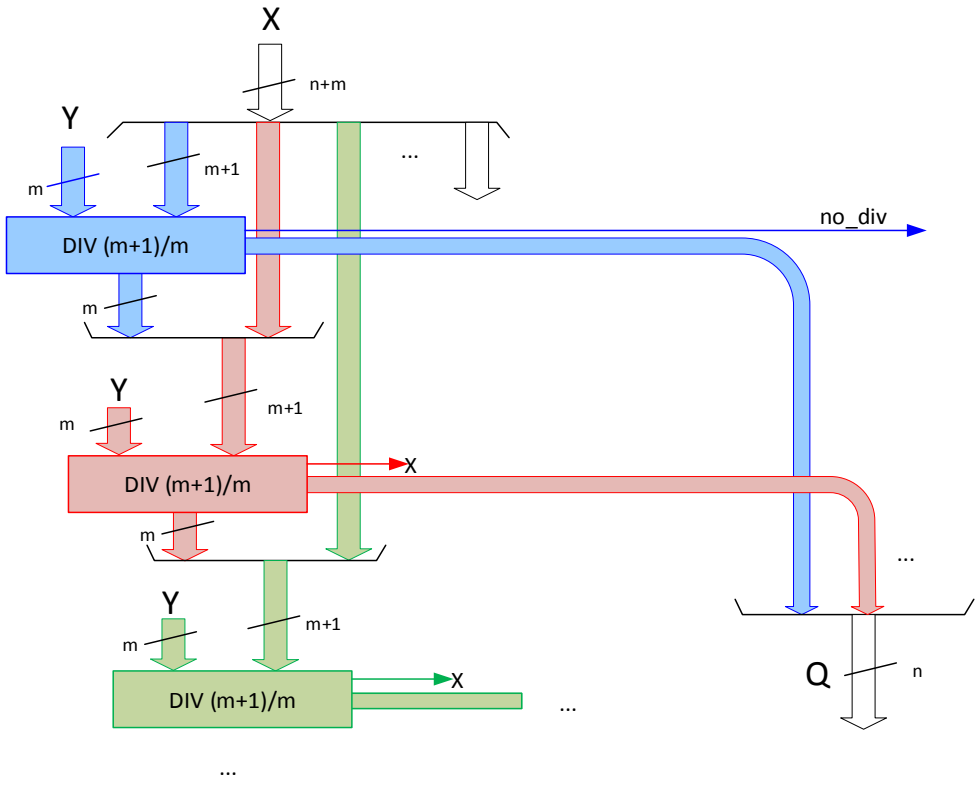
Quando faccio la divisione, invece di considerare **tutto il dividendo X** in un colpo solo:

- prendo il **minimo numero necessario** delle cifre **più significative** di X in modo tale da ottenere un **numero compreso in** $[Y, \boldsymbol{\beta} \cdot \boldsymbol{Y}[$. Quante sono queste cifre? $\boldsymbol{m}$ **possono non bastare;** $\boldsymbol{m+1}$ **cifre bastano** (purché, ovviamente, non abbia zeri in testa). Chiamo X' questo nuovo dividendo
- Calcolo un **quoziente e resto *parziali*** q_i , R della divisione di X' per Y. Sono certo che q_i **sta su una sola cifra** (perché $X' < \beta \cdot Y$ per ipotesi).
- Calcolo un **nuovo dividendo** X' concatenando il resto ottenuto con la **cifra più significativa** non ancora ancora utilizzata del dividendo X. Il nuovo dividendo X' è ancora **minore di** $\boldsymbol{\beta} \cdot \boldsymbol{Y}$, date le ipotesi.



- Vado avanti fino ad esaurire le cifre del dividendo.
- Il quoziente è ottenuto dal concatenamento dei quozienti parziali (che, **per le ipotesi che ho fatto,** stanno **tutti su una cifra**).
- Il resto è il resto dell'ultima divisione parziale.

Quindi, il nucleo di questo algoritmo iterativo è una divisione di $\boldsymbol{m+1}$ **cifre per** $\boldsymbol{m}$ **cifre**, che produce un **quoziente su una cifra ed un resto su** $\boldsymbol{m}$ **cifre.** Tutto il resto può essere ottenuto concatenando cifre (per produrre i nuovi dividendi ed il quoziente finale). In termini circuitali:



Dobbiamo inserire nel diagramma la fattibilità della divisione (l'algoritmo scritto sopra non si occupa di settare l'uscita *no_div*). Sappiamo che la condizione di fattibilità è che **il quoziente stia su n cifre, cioè che** $X < \beta^n \cdot Y$. Dall'esempio e dal circuito si vede abbastanza bene che questa condizione è **equivalente** a dire che le m **cifre più significative del dividendo rappresentano un numero più piccolo del divisore**. Infatti, posso rappresentare X e $\beta^n \cdot Y$ in termini di cifre, come:

$$\begin{array}{rccccc|ccc} X: & x_{n+m-1} & x_{n+m-2} & \cdots & x_{n+1} & x_n & x_{n-1} & \cdots & x_0 \\ \beta^n \cdot Y: & y_{m-1} & y_{m-2} & \cdots & y_1 & y_0 & 0 & \cdots & 0 \end{array}$$

Dal diagramma di sopra si vede piuttosto bene che $X < \beta^n \cdot Y$ se e solo se le cifre di X a sinistra della linea verticale rappresentano un numero minore di Y.

Pertanto l'uscita *no_div* del circuito finale è data dall'uscita *no_div* dal **primo modulo** della scomposizione. Se la prima *no_div* è zero, lo saranno anche le altre.

Richiamo 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.

2.7.1 Divisore elementare in base 2

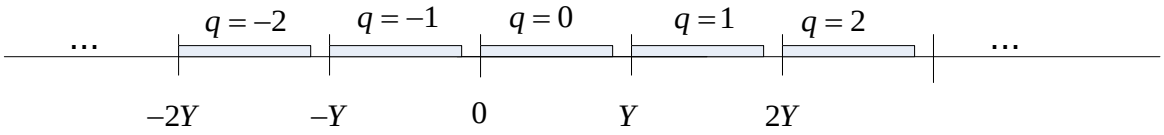
Vediamo di sintetizzare l'unità in base 2 che esegue una divisione di un dividendo a $m + 1$ cifre per un divisore ad m cifre, **sotto l'ipotesi che $X < 2Y$.**

Tale unità produce un

- **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**.

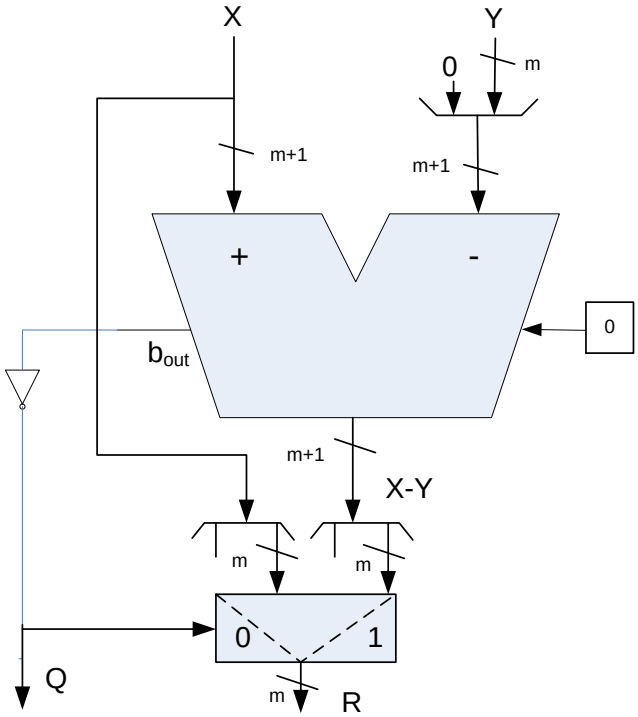
$$Q = \begin{cases} 0 & X < Y \\ 1 & X \geq Y \end{cases}, \qquad R = \begin{cases} X & X < Y \\ X - Y & X \geq Y \end{cases}$$



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

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

Ma per stabilire se X sia minore di Y, basta che li **sottragga** e guardi un eventuale **prestito uscente**.



Il rilevatore di fattibilità (la parte di rete che genera l'uscita *no_div)* è l'uscita *flag_min* di un *comparatore* tra X e $2Y$ (non disegnata per semplicità).

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

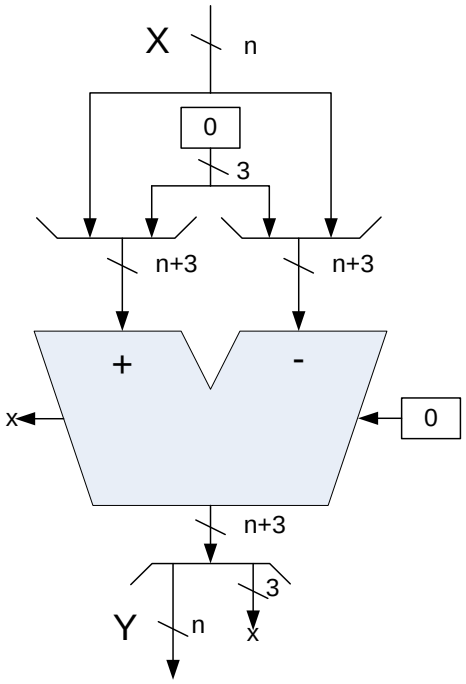
"Sia dato X numero naturale rappresentato su n cifre in base 2. Senza far uso di moltiplicatori e divisori, progettare una rete combinatoria che riceve in ingresso X e produce in uscita la rappresentazione del numero naturale $Y = \lfloor 7/8 \cdot X \rfloor$ in base 2 su ? cifre."

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 $Y = \lfloor 7/8 \cdot X \rfloor$, posso scrivere: $Y = \lfloor (8X - X)/8 \rfloor$. Nell'ordine:

- moltiplicare X per una **potenza della base** ($8 = 2^3$) è un'operazione di 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 di costo nullo.

Quindi:



Attenzione:

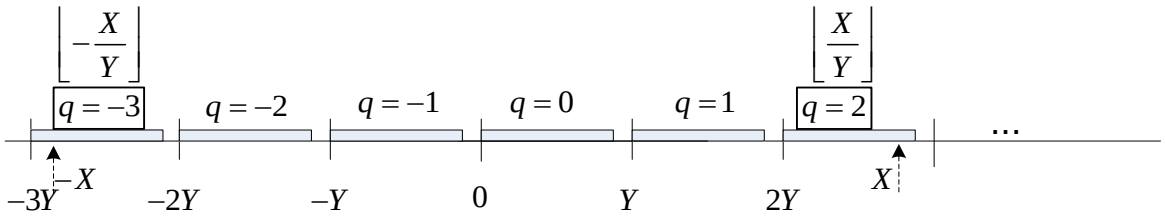
Qualcuno potrebbe aver pensato la seguente cosa: "**prima** calcolo $\lfloor X/8 \rfloor$ e **poi** lo sottraggo da X, ottenendo il risultato corretto". Oltretutto, così facendo, **la differenza è su n cifre**, invece che su $n + 3$. Il risultato che si ottiene, però, **è sbagliato.** Infatti, in questo modo sto calcolando

$$X - \left\lfloor \frac{X}{8} \right\rfloor \neq \left\lfloor \frac{7X}{8} \right\rfloor$$

I due numeri sono uguali **soltanto quando *X* è multiplo di 8.** Infatti:

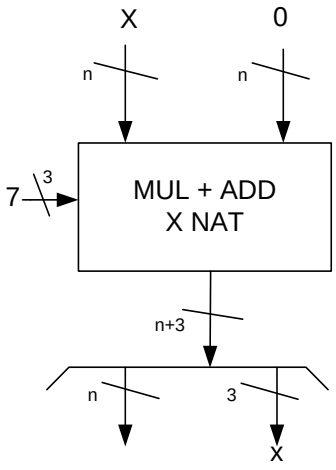
- $X = 16$: il primo dà 14, il secondo dà 14;
- $X = 13$: il primo dà 12, **il secondo dà 11.**

Quindi, **attenzione** quando si usano gli operatori $\lfloor \cdot \rfloor$, perché si corre il rischio di sbagliare. In particolare, è bene osservare che $\left\lfloor -\frac{X}{Y} \right\rfloor \neq -\left\lfloor \frac{X}{Y} \right\rfloor$, in quanto la divisione per un naturale ha resto positivo (quindi **approssima verso sinistra**).



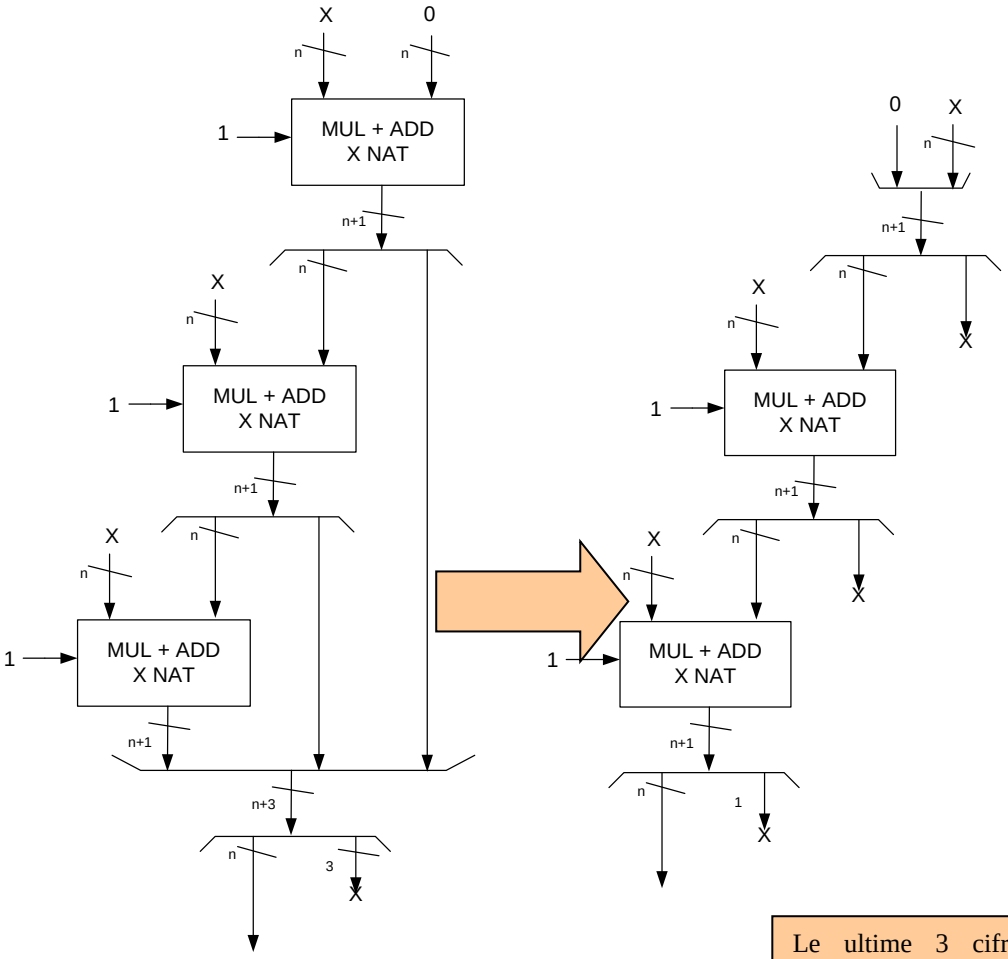
Come si fa ad evitare errori del genere? La tecnica standard è **testare i circuiti** con degli ingressi, così come si fa con i programmi. Si scelgono i valori degli ingressi in modo oculato, cercando di coprire eventuali casi particolari, e si **simula** il circuito con la carta e la penna, calcolando le uscite. Se le uscite sono corrette, abbiamo **maggior sicurezza** (ma mai la certezza, tranne che dopo testing esaustivo di tutti i possibili ingressi) che il circuito sia corretto. Se almeno in un caso l'uscita non torna, è necessario rivedere la sintesi.

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 scrivere:



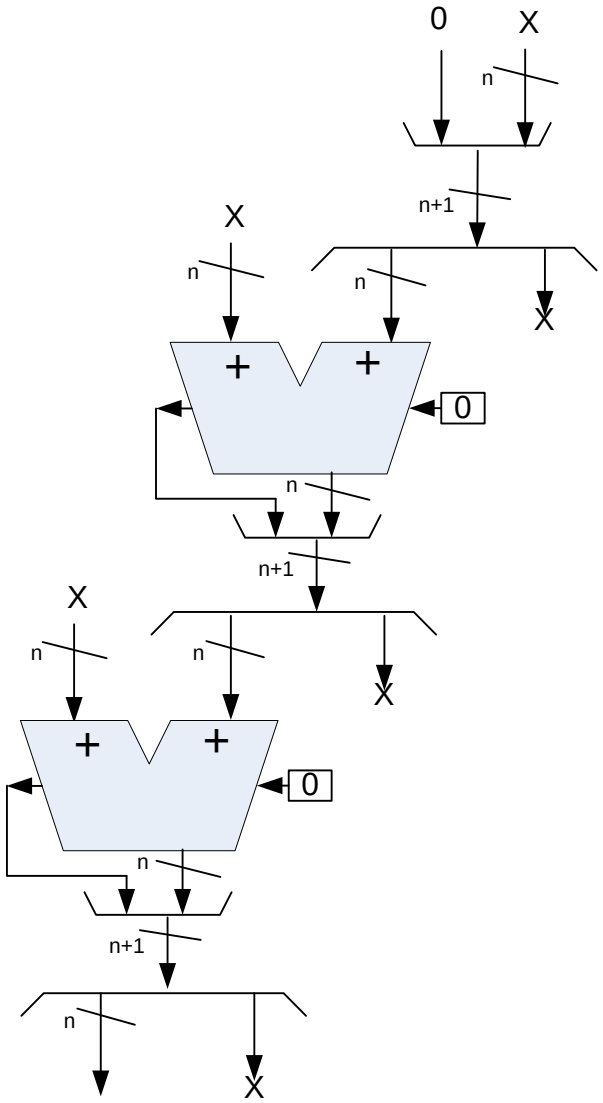
Si può sostituire alla scatola disegnata sopra la sua implementazione in termini di **full adder**, e, con le opportune semplificazioni, sintetizzare un terzo circuito che fa la stessa cosa.

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



Le ultime 3 cifre vengono buttate via dalla divisione per 8

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.



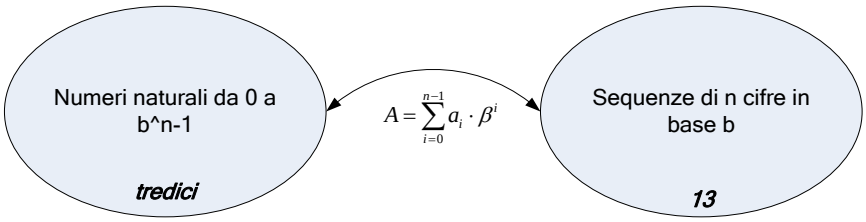
In questo modo, con **due** full adder ad n bit ce la facciamo. Nell'altro caso, ci vuole **un solo full adder**, ma ad $n + 3$ bit.

3 Rappresentazione dei numeri interi

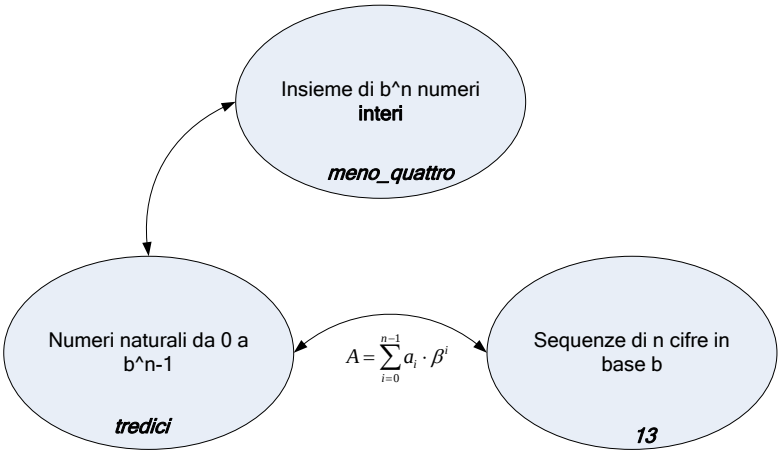
Poniamoci adesso il problema della rappresentazione dei numeri **interi**, quelli che siamo abituati a rappresentare con **un simbolo, detto segno, ed un numero naturale**, **detto modulo o valore assoluto.** Servono perché l'insieme dei numeri naturali non è chiuso rispetto alla sottrazione.
In realtà, molti anni fa si decise di non rappresentare i numeri come modulo e segno (1 bit) all'interno dei calcolatori, perché questo avrebbe reso le reti che operano sulle cifre (leggermente) più complesse. Per questo è stata adottata una rappresentazione **non intuitiva**, detta rappresentazione in **complemento alla radice**, che ormai non può che essere mantenuta per compatibilità.

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

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



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



Procedo come segue:

- stabilisco innanzitutto di lavorare su **campi finiti**, cioè avendo a disposizione un numero **limitato n** di cifre in base **β, noto a priori**.
- so rappresentare i numeri naturali in base **β** 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 $\boldsymbol{\beta}$ 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)

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

E quindi posso scrivere:

$$a \overset{L}{\leftrightarrow} A \equiv (a_{n-1}a_{n-2}\dots a_1 a_0)_\beta$$

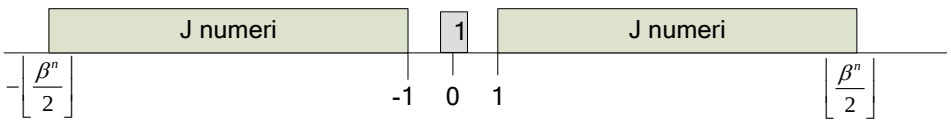
A dire che quella a destra è la **rappresentazione del numero intero $\boldsymbol{a}$ su $\boldsymbol{n}$ cifre in base $\boldsymbol{\beta}$.**

Perché usare tutto questo marchingegno? Per non usare un simbolo in più? Perché scegliendo opportunamente la legge $L(\ \ )$ posso ottenere dei vantaggi implementativi. In particolare, esistono leggi $L(\ \ )$ che consentono di utilizzare i circuiti già visti per le operazioni sui naturali anche per operare su rappresentazioni di interi. Posso sfruttare questo per sintetizzare reti che producono risultati **corretti** sia interpretando le cifre come rappresentazioni di numeri naturali, sia interpretandole come rappresentazioni di numeri interi.

Descriviamo la relazione biunivoca $L(\ \ )$, dandone dominio, codominio e legge di corrispondenza:
L'insieme di numeri interi che voglio rappresentare (il dominio di $L(\ \ )$) dovrà essere:

- contiguo (privo di buchi). Dovrà quindi essere **un intervallo** (altrimenti non potrei calcolare il successivo di qualche numero).
- 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 β^n possibilità, logica vorrebbe che dividessi l'intervallo dei numeri 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 β^n numeri **soltanto se β è dispari**.

Noi lavoreremo sempre con β pari a 2, 8, 10, 16. Se β è pari, dovremo accettare di rappresentare un numero positivo o negativo in più. La scelta che si opera in questo caso sarà di rappresentare l'intervallo di interi:

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

Cioè di **rappresentare un numero negativo in più**.

3.1 Possibili leggi di rappresentazione dei numeri interi

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

$$L: \quad A = a + \frac{\beta^n}{2}$$

$\beta^n/2$ 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
$rapp.$	00000000	00000001	01111111	10000000	10000001	11111110	11111111

In questo caso, lo zero sarà rappresentato come il numero naturale 128, cioè 10000000. 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**[1].

Complemento alla radice

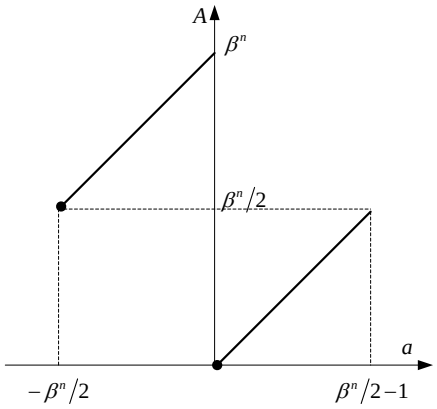
[1] In questo caso il fattore di polarizzazione può essere diverso da $\beta^n/2$ (di poco, in genere: al massimo $\beta^n/2 \pm 1$)

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

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

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

Questo disegno ci aiuta a ricordare come è fatta la rappresentazione in complemento alla radice, e va tenuto **sempre in mente**.



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 $a < b \Leftrightarrow A < B$. È monotona all'interno di numeri **con lo stesso segno**.

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)**.

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

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

Questo tipo di rappresentazione non ricade nella trattazione scritta prima. Infatti, non 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 proprietà, che saranno usate nel seguito.

$$a \leftrightarrow A \equiv (a_{n-1}a_{n-2}\dots a_1 a_0)_\beta$$

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

$$a \ge 0 \quad \Leftrightarrow \quad 0 \le A < \frac{\beta^n}{2}$$

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

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 $\beta^n/2 \equiv (\beta/2 \;\; 00\ldots0)_\beta$. Quindi:

$$\beta^n/2 - 1 \equiv \big((\beta/2 - 1)(\beta - 1)(\beta - 1)\ldots(\beta - 1)\big)_\beta$$

Esempio:

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

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

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

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

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

$$a \ge 0 \quad \Leftrightarrow \quad a_{n-1} = 0$$
$$a < 0 \quad \Leftrightarrow \quad a_{n-1} = 1$$

<u>Legge inversa</u>

Si ottiene banalmente per sostituzione:

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

Nella formula di destra posso:

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

$$L^{-1}: \quad a = \begin{cases} A & a_{n-1} < \dfrac{\beta}{2} \\ -\big(\overline{A} + 1\big) & a_{n-1} \ge \dfrac{\beta}{2} \end{cases}$$

<u>Esempi</u>:

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

- $A \equiv (852)_{10}$. Visto che la cifra più significativa è maggiore di $\beta/2 - 1 = 4$, il numero rappresentato è **negativo**. Quindi, per trovarlo devo calcolare $\overline{A} \equiv (147)_{10}$, sommargli uno e cambiare il segno: $a = -148$
- $A \equiv (500)_{10}$. $a = -(499 + 1) = -500$

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

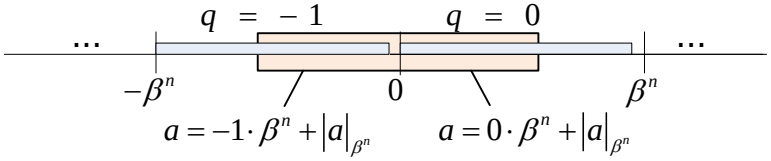
- $A \equiv (1011)_2$. Visto che la cifra più significativa è 1, il numero è negativo. Quindi: $a = -(0100 + 1)_2 = -(101)_2 = -5$
- $A \equiv (1111)_2$. Visto che la cifra più significativa è 1, il numero è negativo. Quindi: $a = -(0000 + 1)_2 = -(1)_2 = -1$

Forma alternativa per L

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

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

Infatti, se a è un numero positivo, allora è anche minore di β^n, e quindi $A = a = |a|_{\beta^n}$. Se invece è un numero negativo, allora è compreso in $[-\beta^n/2\,, 0[$, e quindi se lo divido per β^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 β in complemento alla radice, cioè se appartiene a $[-\beta^n/2\,, \beta^n/2 - 1]$. È 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 β (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 β).

Esempio:

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

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** a questo 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) β 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

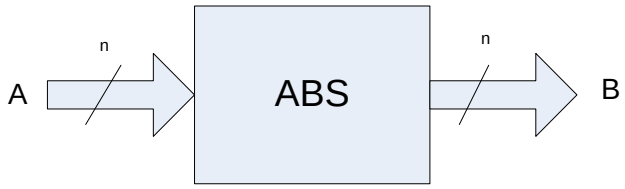
4 Operazioni su interi in complemento alla radice

Il motivo per cui abbiamo definito una legge di rappresentazione degli interi come numeri naturali è che vogliamo progettare circuiti che **lavorano sulle rappresentazioni**: saranno circuiti che hanno in ingresso **cifre in base β** e producono in uscita **cifre in base β**. Interpretando quelle cifre (in ingresso e in uscita) secondo la legge di rappresentazione che abbiamo in testa (quella in *complemento alla radice*), quei circuiti eseguiranno operazioni significative (e.g., somma, sottrazione, etc.) su numeri interi. È importante sottolineare che **i circuiti vedono solo cifre**. Il legame tra quelle cifre ed i numeri interi che queste rappresentano **sta soltanto nella nostra mente**, e non nei circuiti stessi.

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 $A \equiv (a_{n-1}, \ldots, a_0)_\beta$ produce in uscita $B \equiv (b_{n-1}, \ldots, b_0)_\beta$, 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**).

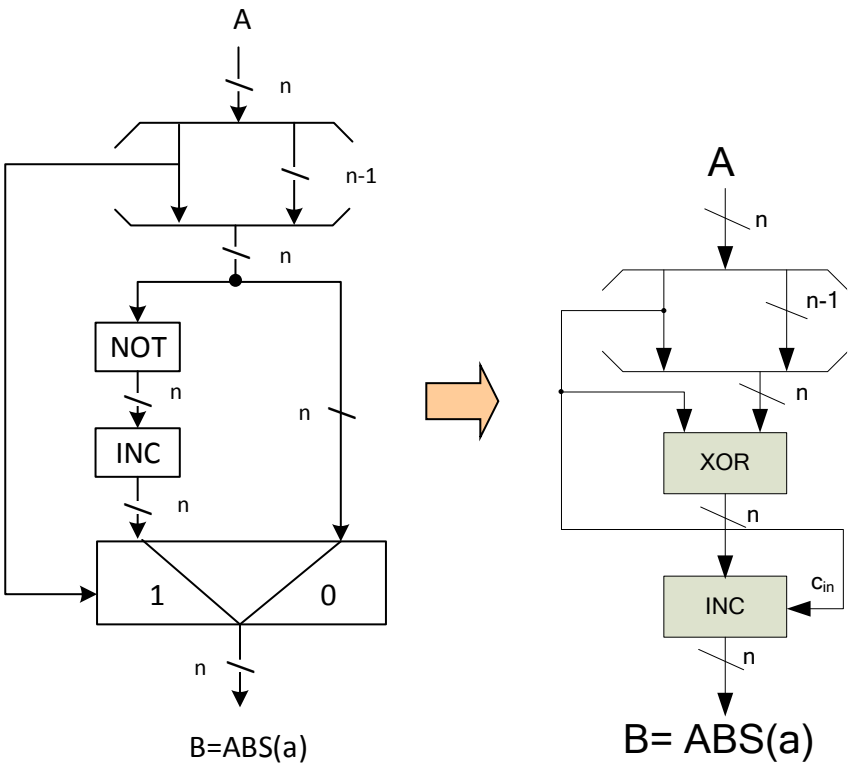
$$B = ABS(a) = \begin{cases} A & a_{n-1} < \beta/2 \\ \overline{A} + 1 & a_{n-1} \geq \beta/2 \end{cases}$$

Esempi:

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

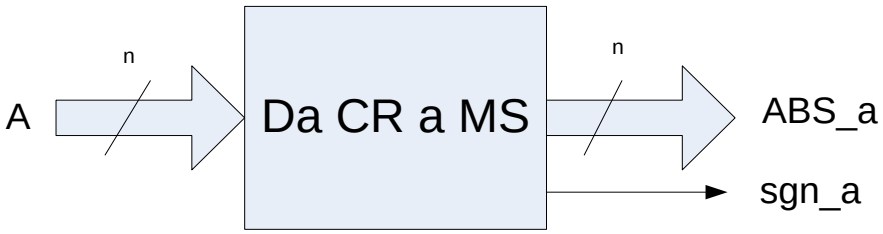
$\beta = 2, \quad n = 4$: $\quad A \equiv (1011)_2 \Rightarrow B = ABS(a) = (0100)_2 + (1)_2 = 5$

In base 2, la legge è più semplice: infatti, si semplificano le condizioni, che diventano a_{n-1}=0, a_{n-1}=1. Inoltre, il complemento del numero A si fa con una barriera di invertitori.



Posso però semplificare ulteriormente, usando una barriera di n XOR, in cui la porta j-sima calcola $a_{n-1} \oplus a_j$, per realizzare o meno il complemento del numero[2]. Posso poi usare un incrementatore che, sempre guidato dalla cifra più significativa di A, incrementa il numero o lo lascia invariato.

4.1.1 Circuito di conversione da CR a MS

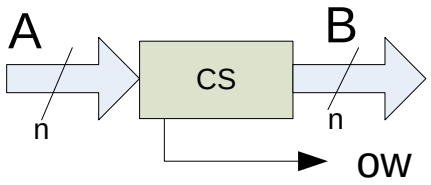


Dal precedente schema si ricava immediatamente un circuito che converte la rappresentazione A in complemento alla radice nella rappresentazione in modulo e segno dello stesso numero.
L'operazione è sempre fattibile nelle ipotesi scritte sopra, visto che l'intervallo di rappresentabilità di un numero su n cifre in MS **contiene** interamente quello su n cifre in complemento alla radice.

[2] La porta XOR $n-1$ calcola $a_{n-1} \oplus a_{n-1} = 0$, e quindi può essere omessa, semplificando leggermente il circuito.

4.2 Calcolo dell'opposto

Dato $A \leftrightarrow a$, voglio trovare $B \leftrightarrow b$ tale che $b = -a$. È un'operazione **non sempre possibile** se decido di rappresentare a e b sullo stesso numero di cifre. 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 nel range. 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**.

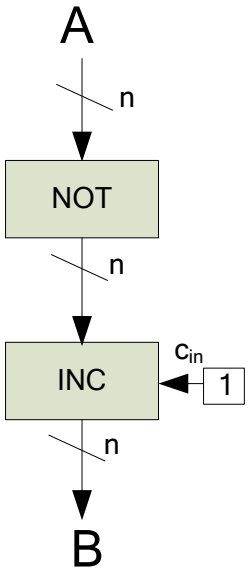
In questo caso abbiamo:

$$B \triangleq |-a|_{\beta^n}$$
$$= \left||-1|_{\beta^n} \cdot |a|_{\beta^n}\right|_{\beta^n} = |(\beta^n - 1) \cdot A|_{\beta^n}$$
$$= |\beta^n \cdot A - A|_{\beta^n} = |-A|_{\beta^n}$$
$$= \left|-\beta^n + 1 + \overline{A}\right|_{\beta^n} = \left|1 + \overline{A}\right|_{\beta^n}$$

NB: se a è l'estremo inferiore, la **rappresentazione di -a non esiste**.

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

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. Questo conferma che, nell'espressione di sopra, **il modulo serve**.

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

- controllare se $A \equiv (10\ldots00)$, 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 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.

$$L_{n+1}{}': \quad A^{EST} = \begin{cases} a & 0 \le a < \dfrac{\beta^n}{2} \\ \beta^{n+1} + a & -\dfrac{\beta^n}{2} \le a < 0 \end{cases}$$

ed inoltre

$$L_n^{-1}: \quad a = \begin{cases} A & a_{n-1} < \dfrac{\beta}{2} \\ -\left(\overline{A} + 1\right) = -\beta^n + A & a_{n-1} \ge \dfrac{\beta}{2} \end{cases}$$

Quindi:

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

Quindi, ricavo immediatamente che:

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

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

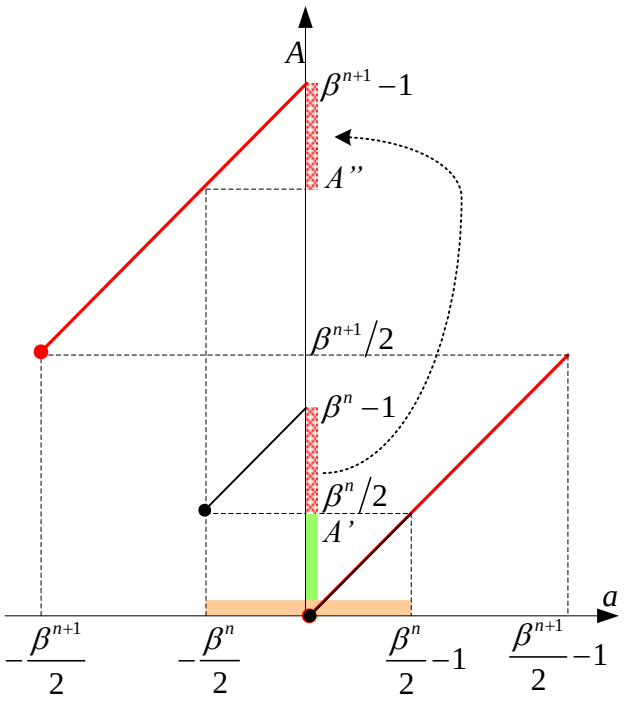
È utile ripetere lo stesso ragionamento **per via grafica**. Partendo dalla figura della rappresentazione in complemento alla radice su n cifre, vogliamo ricavare quella per $n+1$ cifre e vedere come variano gli intervalli di ordinate. Per poter mantenere la legge di rappresentazione su $n+1$ cifre:

- l'intervallo di ordinate che rappresenta i numeri negativi deve essere traslato verso l'alto di una quantità $\left(\beta^{n+1} - 1\right) - \left(\beta^n - 1\right) = \beta^n \cdot (\beta - 1)$. Ciò significa aggiungere una cifra in più (a sinistra) rispetto alla rappresentazione originale, di peso $\beta - 1$.

- L'intervallo di ordinate che rappresenta gli interi positivi deve invece rimanere invariato, e quindi l'unica possibilità è aggiungere una cifra più significativa di peso nullo.

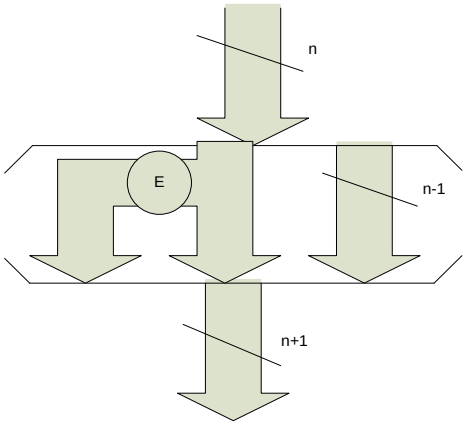
Quindi, le rappresentazioni **estese** dei numeri interi appartenenti all'intervallo $\left[-\frac{\beta^n}{2}, \frac{\beta^n}{2}-1\right]$ saranno fatte così:

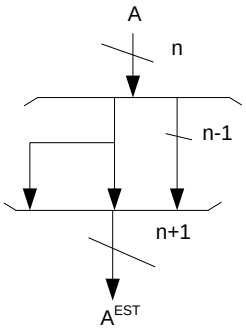
- Se rappresentazioni di numeri *positivi*, avranno MSD=0, cifra successiva $< \frac{\beta}{2}$;
- Se rappresentazioni di numeri *negativi*, avranno MSD=$\beta - 1$, cifra successiva $\geq \frac{\beta}{2}$.



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, CWDE, CDQ**, che interpretano una sequenza di bit come la rappresentazione di un numero **intero** (su 8, 16, 32 bit), e producono la rappresentazione dello stesso numero **estesa su 16, 32, 64** bit. Per contro, non esiste nessuna istru-

zione 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 BCD. Sintetizzare il circuito *sia* a porte NOR *che* a porte NAND a costo minimo, svolgendo il procedimento in maniera completa (cioè classificando gli implicanti, trovando tutte le liste di copertura irridondanti e scegliendo una di quelle a costo minimo).
Soluzione

4.4 Riduzione di campo

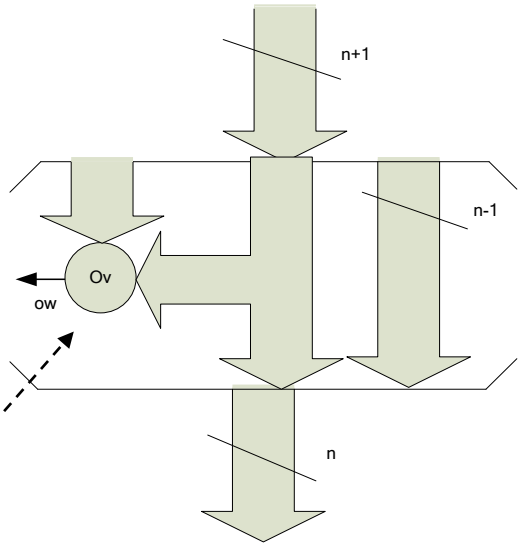
Affrontiamo adesso il problema inverso. Dato A, $A \equiv (a_n a_{n-1} a_{n-2} \dots a_1 a_0)_\beta$, su $\boldsymbol{n+1}$ cifre, tale che $a \leftrightarrow A$, voglio trovare $A^{RID} \equiv (a_{n-1}' a_{n-2}' \dots a_1' a_0')_\beta$, su n cifre, tale che $\boldsymbol{a \leftrightarrow A^{RID}}$.
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** $\boldsymbol{A}$. Non è difficile: quando a è nell'intervallo nel quale *a* è riducibile, A appartiene ad uno dei due intervalli marcati sull'asse delle ordinate. Ma, per quanto detto prima, le rappresentazioni di numeri **estesi** saranno fatte così:

- L'intervallo di ordinate $[0;\ A']$, che contiene rappresentazioni di numeri *positivi*, ha MSD=0, cifra successiva $< \frac{\beta}{2}$;
- L'intervallo di ordinate $[A'';\ \beta^{n+1} - 1]$, che contiene rappresentazioni di numeri *negativi*, ha MSD=$\beta - 1$, cifra successiva $\geq \frac{\beta}{2}$.

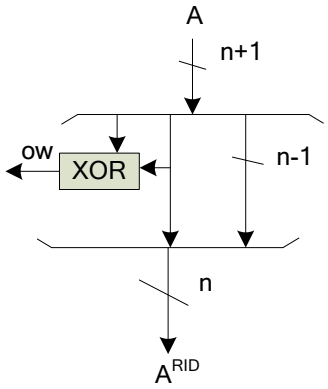
Quindi, la condizione per la riducibilità, testabile direttamente sulle cifre della rappresentazione, è:

$$ow = 0 \leftrightarrow \left(a_n = 0 \wedge a_{n-1} < \frac{\beta}{2}\right) \vee \left(a_n = \beta - 1 \wedge a_{n-1} \geq \frac{\beta}{2}\right)$$

Nel caso in cui il numero sia riducibile, la sua rappresentazione su n cifre è data **dalle** $\boldsymbol{n}$ **cifre meno significative di** $\boldsymbol{A}$ (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 l'overflow è lo XOR delle due cifre più significative.

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 in base 2 da $n + k$ a n cifre, posso

a) utilizzare k XOR a due ingressi, e poi ottenere l'overflow come OR delle uscite di questi;
b) 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.

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 $B = \beta \cdot A$. Infatti,

$$L: \quad B = \begin{cases} b = \beta \cdot a & 0 \leq a < \frac{\beta^n}{2} \\ \beta^{n+1} + b = \beta^{n+1} + \beta \cdot a = \beta \cdot (\beta^n + a) & -\frac{\beta^n}{2} \leq a < 0 \end{cases}$$

, da cui la tesi.

Analogamente, dato $A \equiv (a_n a_{n-1} a_{n-2} \dots a_0)_\beta$ su $\underline{\boldsymbol{n+1}}$ cifre, tale che $a \leftrightarrow A$, voglio trovare B su n cifre tale che $b \leftrightarrow B$ e $b = \lfloor a/\beta \rfloor$. La soluzione è $B = \lfloor A/\beta \rfloor$. Infatti:

$$B = \left|\left|\frac{a}{\beta}\right|\right|_{\beta^n} = \left|\left|\frac{\left\lfloor a/\beta^{n+1}\right\rfloor \cdot \beta^{n+1} + |a|_{\beta^{n+1}}}{\beta}\right|\right|_{\beta^n} = \left|\left|\left\lfloor a/\beta^{n+1}\right\rfloor \cdot \beta^n + \frac{|a|_{\beta^{n+1}}}{\beta}\right|\right|_{\beta^n}$$

$$= \left|\left|\frac{A}{\beta}\right|\right|_{\beta^n} = \left\lfloor \frac{A}{\beta} \right\rfloor$$

Scrivo a come quoziente e resto della divisione per β^{n+1}

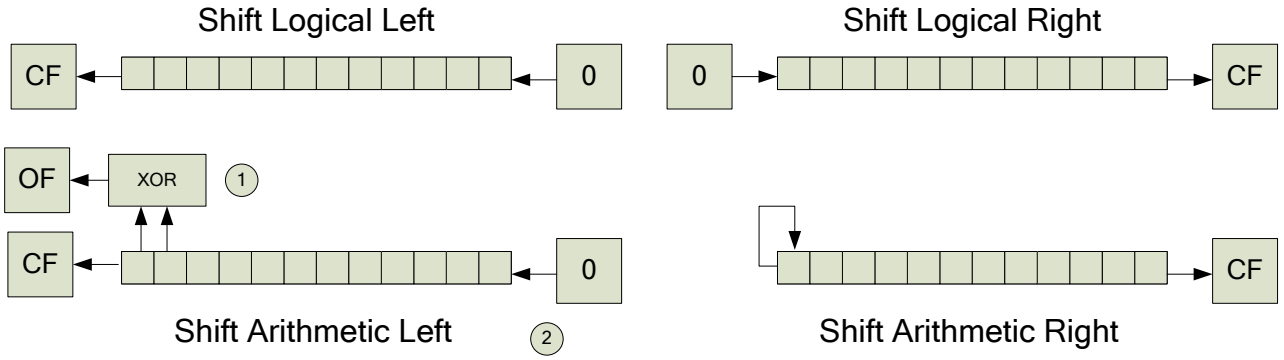
Anche per gli **interi**, moltiplicare e dividere per una potenza della base è un'operazione di costo nullo, e si fa nello stesso modo che con i naturali.

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 rappresentazione. 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 $\boldsymbol{n-1}$ 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.

Ciò significa che, a guardare bene, basta una sola istruzione di shift sinistro per interi e naturali (sta poi al programmatore discriminare quale flag guardare in base al contenuto del registro, come avviene per le somme/sottrazioni).

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 $\boldsymbol{n}$ cifre più significative della rappresentazione estesa, buttando via la meno significativa.

Quindi le due istruzioni di shift **destro** sono intrinsecamente **differenti**, e per questo devono essere tenute distinte. Questo è il motivo per cui (per simmetria) ne esistono anche due di shift sinistro.

4.6 Somma

Dati A, B in base β 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**.

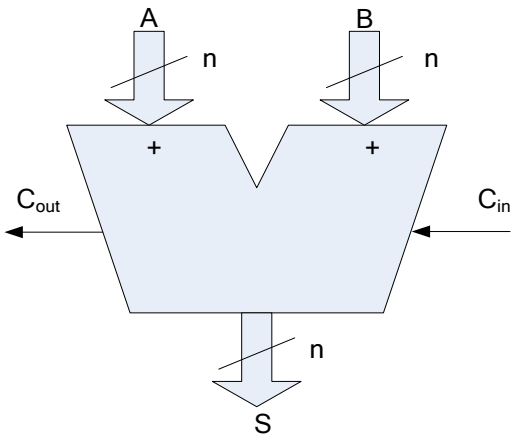
Quando *s* è rappresentabile su *n* cifre, abbiamo:

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

Il che significa che posso ricavare **la rappresentazione della somma come somma delle rappresentazioni** modulo β^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 β^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, $C_{out} = 1$

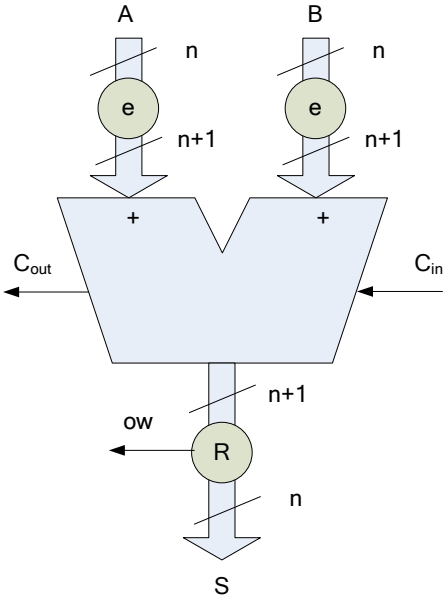
- $A = \beta^n/2 - 1, B = \beta^n/2 - 1 \Rightarrow S = \beta^n - 2, C_{out} = 0$

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

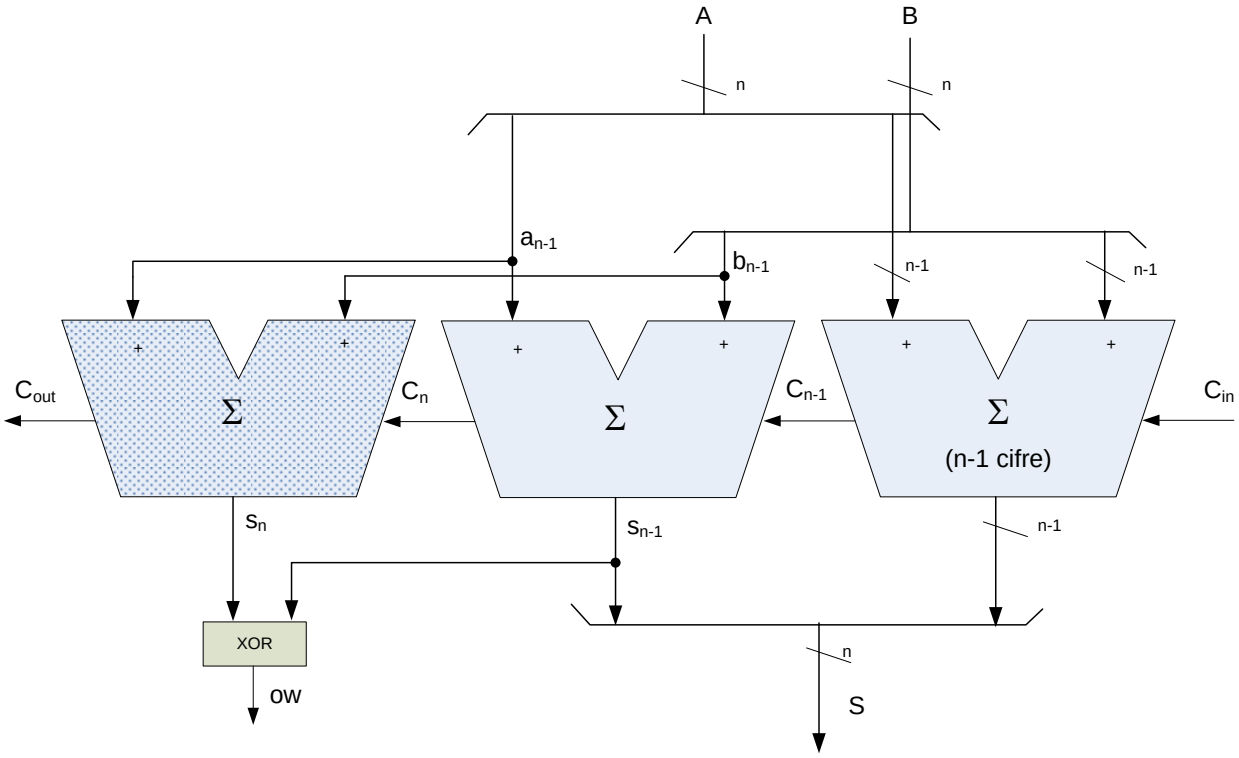
Quindi, quando la somma è tra **naturali**, il riporto uscente mi dice se è rappresentabile. Quando la somma è tra rappresentazioni di **interi**, **il riporto uscente non offre nessuna informazione riguardo alla rappresentabilità del risultato**.

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:



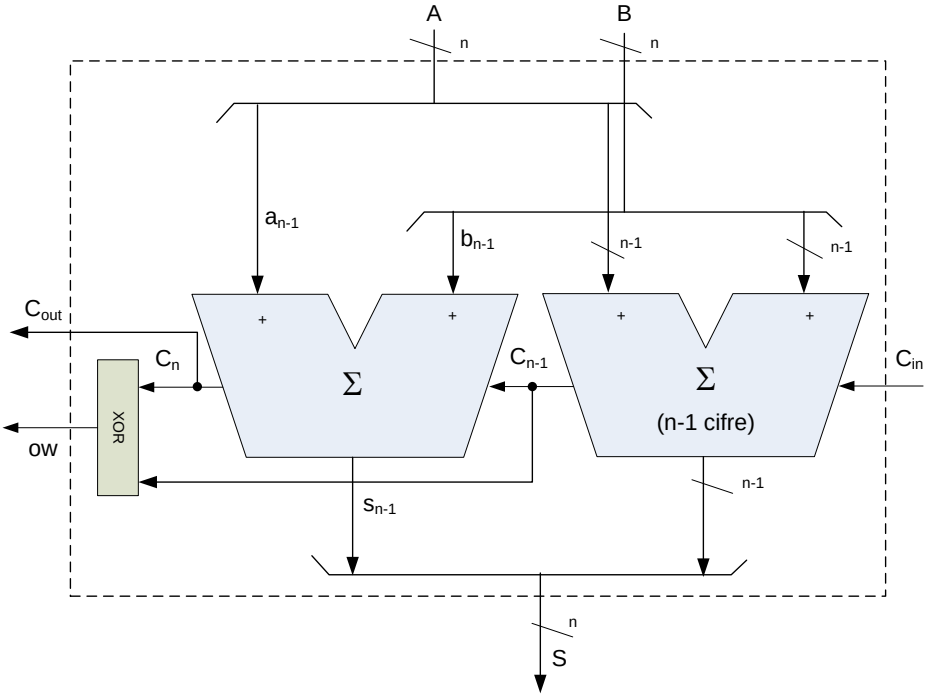
In base β generica, quindi, non posso che usare **uno stadio di sommatore in più** rispetto al numero delle cifre di cui sono composti gli addendi. In base 2, il detettore di riducibilità è uno XOR delle due cifre più significative.



Ricordando le proprietà algebriche dello XOR, posso scrivere:

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

Quindi in base 2 l'overflow può essere calcolato dai **riporti uscenti** degli ultimi **due full adder**, e non c'è bisogno di aggiungere un altro stadio di sommatore.



In sostanza con **la stessa circuiteria** con la quale faccio somme tra naturali, posso fare **somme tra interi**. Basta aggiungere un'uscita di overflow, che può essere prodotta quasi gratis. Quindi:

Non esiste il "sommatore per naturali" ed il "sommatore per interi". Esiste il sommatore, che somma sia naturali che (rappresentazioni in CR di) interi, e genera uscite corrette in entrambi i casi (purché la somma sia rappresentabile). Per testare la rappresentabilità, si guardano uscite diverse: C_{out} se gli operandi sono naturali, *ow* se sono interi.

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

La sottrazione si tratta in modo simile alla somma. La differenza è sempre rappresentabile su $n+1$ cifre, e non sempre su n. Dati A e B in base β 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:

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

(*) se d è rappresentabile su n cifre.

Ma l'espressione che sta a destra è quella in uscita da un **sottrattore per numeri naturali**, che abbia in ingresso le rappresentazioni $\boldsymbol{A}$ e $\boldsymbol{B}$. 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:

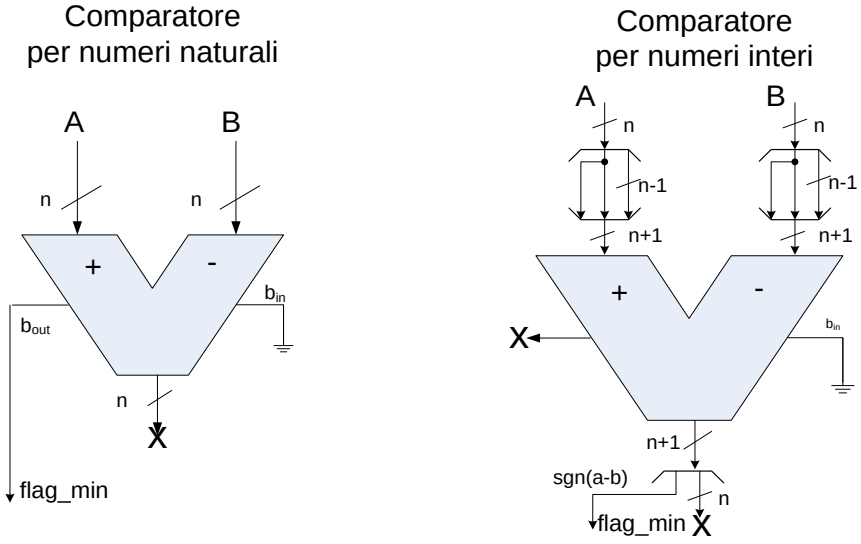
$$D^{EST} = |d|_{\beta^{n+1}} = |a-b|_{\beta^{n+1}} = \left||a|_{\beta^{n+1}} - |b|_{\beta^{n+1}}\right|_{\beta^{n+1}} = |A^{EST} - B^{EST}|_{\beta^{n+1}}$$
$$= \left|A^{EST} + \overline{B^{EST}} + 1\right|_{\beta^{n+1}}$$

Pertanto, in una base generica β, posso generare *ow* testando la riducibilità del risultato della differenza su $n+1$ cifre. Per la base due si può facilmente verificare che *ow* è lo XOR degli ultimi due prestiti (e quindi non c'è bisogno di estendere la differenza su $n+1$ cifre).

4.7.1 Comparazione di numeri interi

La comparazione di uguaglianza tra i numeri interi si fa come tra i numeri naturali (la rappresentazione è unica). La comparazione di minoranza tra numeri interi si fa sempre con un sottrattore, ma **non si deve guardare il prestito uscente**. Si deve guardare il **segno** della differenza, per svolgere la quale è necessario **estendere gli operandi** (il risultato, infatti, potrebbe non essere rappresentabile su n cifre). Questo vale **in qualunque base**, compresa la base 2. Il circuito per la comparazione di interi in base 2 è disegnato sotto. Quello per una base β generica (fare per esercizio) si ricava osservando che:

- L'estensione richiede della logica;
- Il calcolo del segno della differenza richiede logica.



4.8 Moltiplicazione e divisione

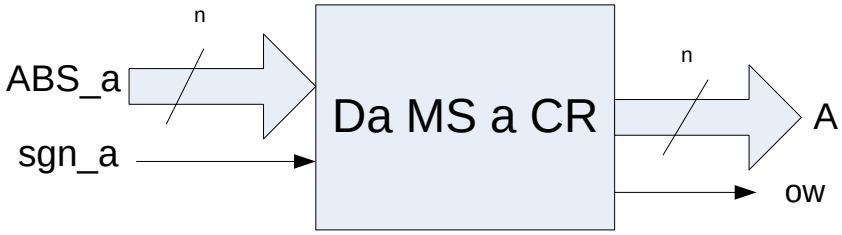
Per la moltiplicazione e divisione conviene riferirsi ai valori assoluti degli operandi, ed aggiustare i segni successivamente. Ciò consente di riutilizzare la circuiteria per le moltiplicazioni/divisioni tra numeri **naturali**.

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 in ingresso il valore assoluto (su *n* cifre) ed il segno della rappresentazione di un numero intero, e produce in uscita 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 fattibile, a 0 altrimenti.

Se l'operazione è fattibile, è

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

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

Per quanto riguarda l'overflow, abbiamo:

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

Quest'ultima operazione si sintetizza 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 che:

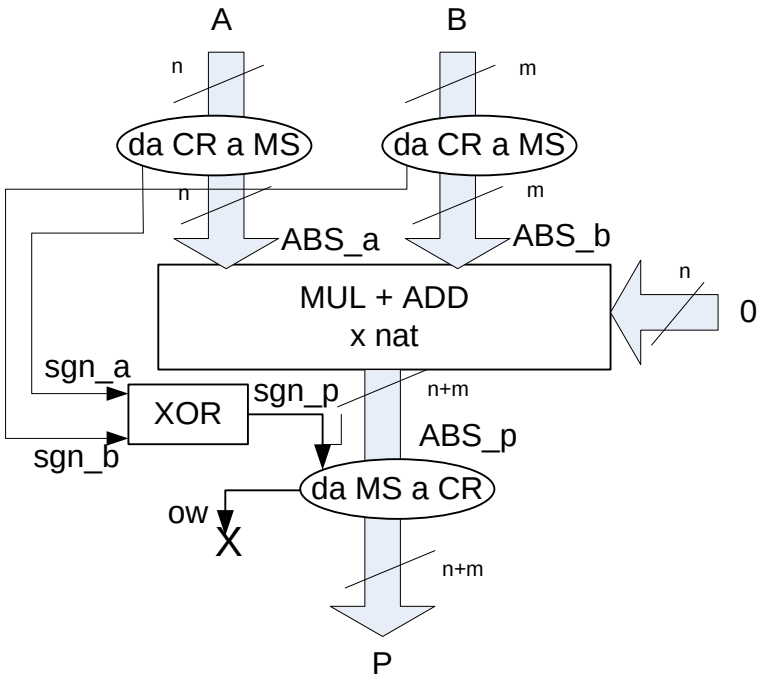
$sgn(x)$ è una funzione matematica che vale $+1$ se $x \geq 0$, e -1 altrimenti. Quindi,
$x = sgn(x) \cdot ABS(x)$

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

Quindi:

$$ABS(p) = ABS(a) \cdot ABS(b)$$
$$\text{sgn}(p) = \text{sgn}(a) \cdot \text{sgn}(b)$$

Quindi posso calcolare $ABS(a) \cdot ABS(b)$, che è il risultato di una moltiplicazione tra **naturali**, e poi rappresentare **questo risultato o il suo opposto** a seconda del fatto 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. Si noti infine che nel moltiplicatore per interi **non esiste un ingresso di somma** (che invece c'è nel il moltiplicatore con addizionatore per naturali).

4.8.3 Esercizio (da fare a casa)

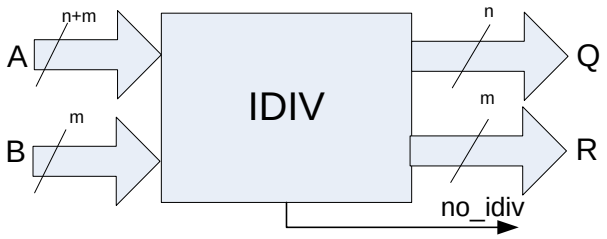
1) Descrivere e sintetizzare (come rete SP a costo minimo) un moltiplicatore per interi ad una cifra in base 3.

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 esistere o 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** (b potrebbe essere negativo)**.** Deve essere adottato un vincolo che **generalizzi il precedente**. Si potrebbe pensare di adottare il seguente vincolo: $ABS(r) < ABS(b)$, che di fatto racchiude il precedente nel caso di numeri naturali. Anche questo vincolo, però, **non basta** a rendere il risultato della divisione **univoco**. Infatti, ecco un controesempio: $a = +17$, $b = -5$. Posso avere $q = -3$, $r = +2$, oppure $q = -4$, $r = -3$. 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:

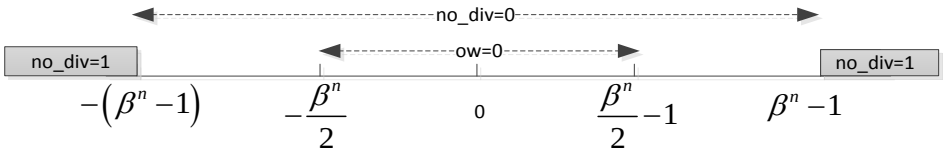
$$ABS(a) = [q \cdot \text{sgn}(b) \cdot \text{sgn}(a)] \cdot ABS(b) + ABS(r)$$

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

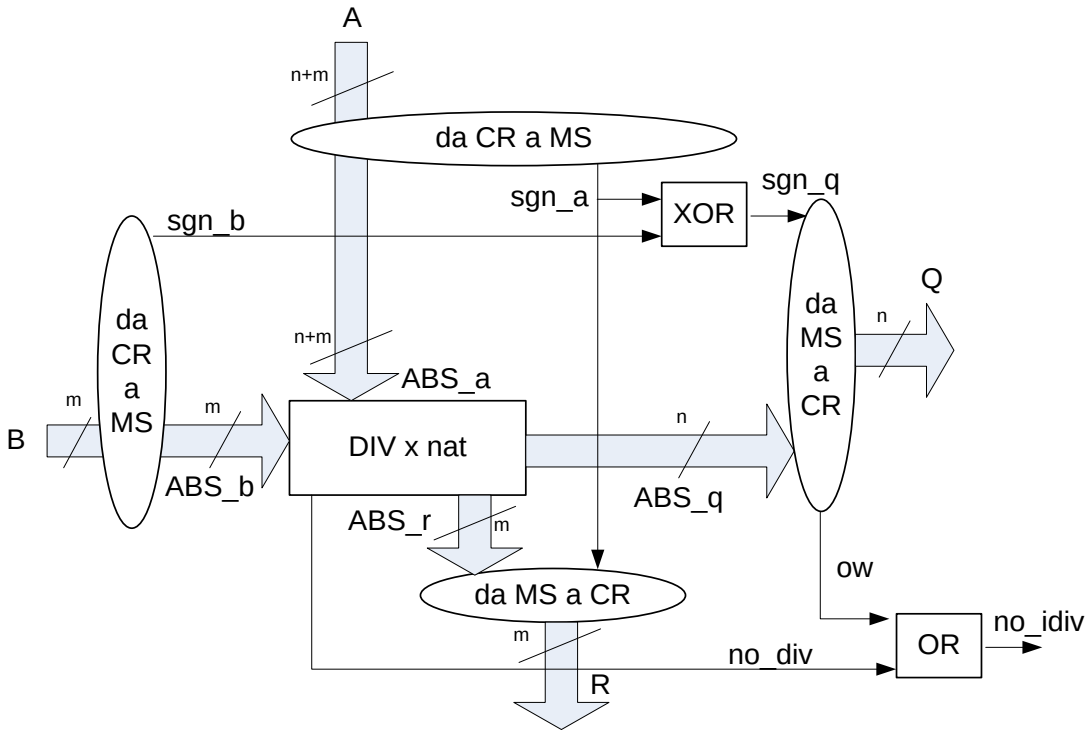
Posso calcolare $ABS(r)$ e $ABS(q)$ utilizzando un modulo **divisore tra naturali**, purché $ABS(q)$ sia **un numero (naturale) rappresentabile su n cifre**. Per testare se questo è vero, dobbiamo controllare se:

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

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). Se $ABS(q)$ non è un naturale rappresentabile su n cifre, quindi, il quoziente q **non è un numero intero rappresentabile su n cifre**. Quindi, la **fattibilità della divisione (naturale) tra i valori assoluti è condizione <u>necessaria</u> per la fattibilità della divisione intera**.



Dalla figura si vede bene che **non è una condizione sufficiente**. Quando *no_div*=0, infatti, non abbiamo garanzia che q sia un numero intero rappresentabile su n cifre.



A questo punto, posso trovare Q ed R abbastanza facilmente, visto che ho i valori assoluti di q e 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

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

e quindi si può avere overflow.

Quando è, allora, che la **divisione intera** è fattibile? Quando:

a) è fattibile quella naturale
b) ìl 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**.

<u>Richiamo sull'Assembler</u>: 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

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 $\delta \triangleq \beta/\gamma$, $\delta \in \mathbb{N}\backslash\{0\}$ per ipotesi. L'ultimo passaggio è dovuto al fatto che il secondo addendo è certamente multiplo di γ, in quanto δ 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 γ_i numero naturale. Quindi, abbiamo:

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

$$= \left|a_0 + \sum_{i=1}^{n-1} a_i + 3 \cdot \cancel{\sum_{i=1}^{n-1} a_i \cdot \gamma_i}\right|_3 = \left|\sum_{i=0}^{n-1} a_i\right|_3$$

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

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

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

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

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

4) Il risultato si dimostra come segue:

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

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

$$= \left|a_0 + \sum_{i=1}^{n-1} a_i \cdot \left[(-1)^i + \sum_{k=1}^{i} \binom{i}{k} \cdot (-1)^{i-k} \cdot (\beta+1)^k\right]\right|_{(\beta+1)}$$

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 γ_i:

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

$$= \left| a_0 + \sum_{i=1}^{n-1} (-1)^i \cdot a_i + (\beta+1) \cdot \sum_{i=1}^{n-1} a_i \cdot \gamma_i \right|_{(\beta+1)}$$

$$= \left| a_0 + \sum_{i=1}^{n-1} (-1)^i \cdot a_i \right|_{(\beta+1)} = \left| \sum_{i=0}^{n-1} (-1)^i \cdot a_i \right|_{(\beta+1)}$$

5.2 Soluzione dell'esercizio 2.1.2

Dalla teoria sappiamo che il complemento di un numero ad n cifre può essere svolto complementando le singole cifre, qualunque sia la base. In base 10, il circuito elementare di complemento avrà in ingresso una cifra in base 10 (codificata su 4 variabili logiche) e presenterà in uscita una cifra in base 10 (ancora su 4 variabili logiche). La tabella di verità è riportata di sotto

J	x_3	x_2	x_1	x_0	z_3	z_2	z_1	z_0	$\bar{J}$
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
	others				-	-	-	-	

Una sintesi (non necessariamente ottimizzata) delle quattro uscite è $z_0 = \overline{x_0}$, $z_1 = x_1$, $z_2 = x_1 \oplus x_2$, $z_3 = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1}$.

2) La codifica "eccesso 3" è rappresentata in tabella:

J	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

Ripetendo il ragionamento svolto al punto precedente, è facile concludere che il circuito di complemento di una singola cifra in base 10 è una barriera di 4 invertitori. Si dice che la codifica "eccesso 3" è autocomplementante.

5.3 Soluzione dell'esercizio 2.4.3

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:

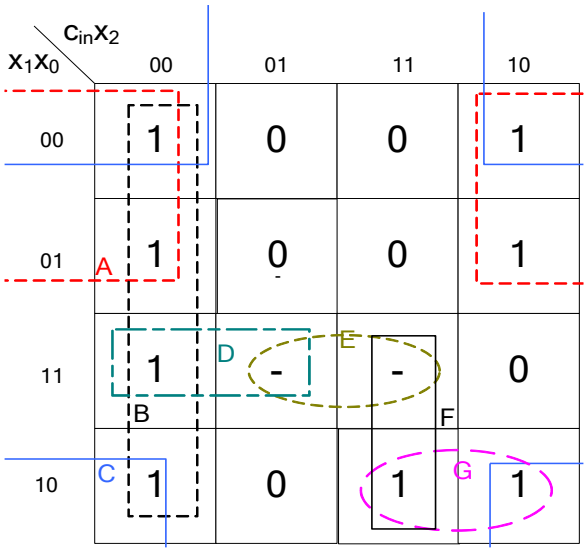
c_{in}	x_2	x_1	x_0	c_{out}	z_2	z_1	z_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	-	-	-	-
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	1	1
1	0	1	1	0	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	1	0	1	0	0	0
1	1	1	1	-	-	-	-

La mappa di Karnaugh per z_2 è la seguente:

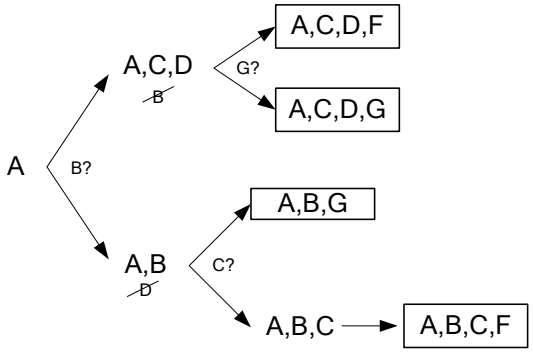
x_1x_0 \ $c_{in}x_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	-	-	1
10	0	1	0	0

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

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



L'unico implicante 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:

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

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

5.4 Soluzione dell'esercizio 2.4.4

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

c

b x_2 \ x_1 x_0	00	01	11	10
00	000	010	001	100
01	011	----	----	----
11	000	----	----	----
10	001	010	100	011

$y_2\, y_1\, y_0$

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

$$c = x_2 + \overline{b} \cdot x_1 \cdot x_0$$
$$y_2 = b \cdot x_1 \cdot x_0 + \overline{b} \cdot x_1 \cdot \overline{x_0}$$
$$y_1 = \overline{x_1} \cdot x_0 + \overline{b} \cdot x_2 + b \cdot x_1 \cdot \overline{x_0}$$
$$y_0 = b \cdot \overline{x_2} \cdot \overline{x_0} + \overline{b} \cdot x_2 + \overline{b} \cdot x_1 \cdot x_0$$

$$c = \overline{\overline{x_2} \cdot \left(\overline{b \cdot x_1 \cdot x_0}\right)}$$
$$y_2 = \overline{\left(\overline{b \cdot x_1 \cdot x_0}\right) \cdot \left(\overline{\overline{b} \cdot x_1 \cdot \overline{x_0}}\right)}$$
$$y_1 = \overline{\left(\overline{\overline{x_1} \cdot x_0}\right) \cdot \left(\overline{\overline{b} \cdot x_2}\right) \cdot \left(\overline{b \cdot x_1 \cdot \overline{x_0}}\right)}$$
$$y_0 = \overline{\left(\overline{b \cdot \overline{x_2} \cdot \overline{x_0}}\right) \cdot \left(\overline{\overline{b} \cdot x_2}\right) \cdot \left(\overline{\overline{b} \cdot x_1 \cdot x_0}\right)}$$

3) Per la sintesi PS abbiamo:

$$\overline{c} = \overline{x_2} \cdot \overline{x_1} + x_1 \cdot \overline{x_0} + b \cdot \overline{x_2}$$
$$\overline{y_2} = \overline{x_1} + \overline{b} \cdot x_0 + b \cdot \overline{x_0}$$
$$\overline{y_1} = \overline{b} \cdot x_1 + x_1 \cdot x_0 + b \cdot x_2 + \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0}$$
$$\overline{y_0} = b \cdot x_2 + \overline{x_1} \cdot x_0 + b \cdot x_0 + \overline{b} \cdot \overline{x_2} \cdot \overline{x_0}$$

$$c = (x_2 + x_1) \cdot (\overline{x_1} + x_0) \cdot \left(\overline{b} + x_2\right)$$
$$y_2 = (x_1) \cdot (b + \overline{x_0}) \cdot \left(\overline{b} + x_0\right)$$
$$y_1 = (b + \overline{x_1}) \cdot (\overline{x_1} + \overline{x_0}) \cdot \left(\overline{b} + \overline{x_2}\right) \cdot (x_2 + x_1 + x_0)$$
$$y_0 = \left(\overline{b} + \overline{x_2}\right) \cdot (x_1 + \overline{x_0}) \cdot \left(\overline{b} + \overline{x_0}\right) \cdot (b + x_2 + x_0)$$

Da cui si ricava quella a porte NOR.

$$c = \overline{(\overline{x_2 + x_1}) + \left(\overline{\overline{x_1} + x_0}\right) + \left(\overline{\overline{b} + x_2}\right)}$$
$$y_2 = \overline{(\overline{x_1}) + \left(\overline{b + \overline{x_0}}\right) + \left(\overline{\overline{b} + x_0}\right)}$$
$$y_1 = \overline{\left(\overline{b + \overline{x_1}}\right) + \left(\overline{\overline{x_1} + \overline{x_0}}\right) + \left(\overline{\overline{b} + \overline{x_2}}\right) + \left(\overline{x_2 + x_1 + x_0}\right)}$$
$$y_0 = \overline{\left(\overline{\overline{b} + \overline{x_2}}\right) + \left(\overline{x_1 + \overline{x_0}}\right) + \left(\overline{\overline{b} + \overline{x_0}}\right) + \left(\overline{b + x_2 + x_0}\right)}$$

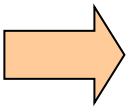
1) la realizzazione a porte NAND di y_0 costa 4 a porte e 11 a diodi, mentre quella a porte NOR costa 5 a porte e 13 a diodi. Pertanto, la prima ha costo minore.

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 $x_3, \ldots, x_0$ 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	-	-	-



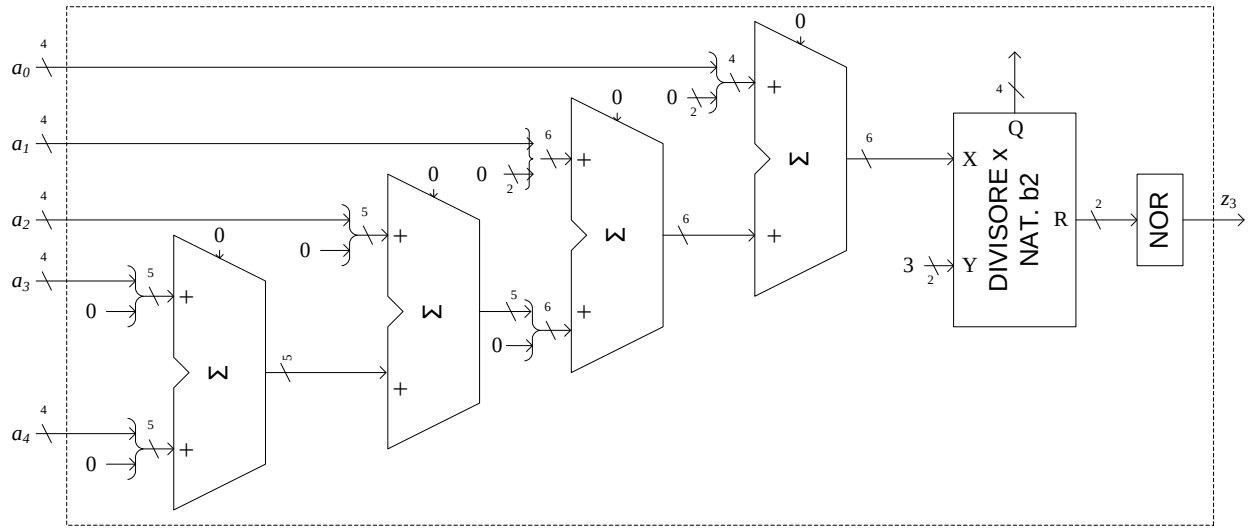
x_1x_0 \ x_3x_2	00	01	11	10
00	111	100	---	100
01	000	010	---	000
11	000	000	---	---
10	100	100	---	---

Da cui si ricava immediatamente:

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

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:

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

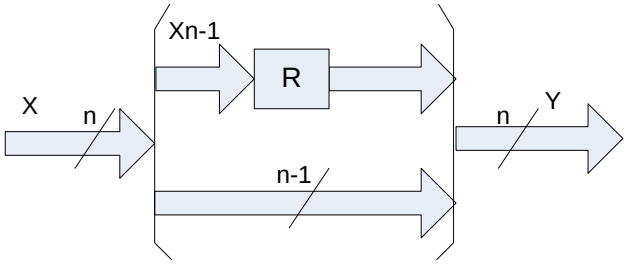
mentre quella tra Y e x è $Y = x + \beta^n/2$. Mettendo insieme le due si ottiene:

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

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

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

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 $d_2 \ldots d_0$ i tre bit che rappresentano la cifra X_{n-1}, e $z_2 \ldots z_0$ 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_0 \ d_2d_1	00	01	11	10
0	011	101	---	001
1	100	000	---	010

$z_2z_1z_0$

La cui sintesi a costo minimo è la seguente:

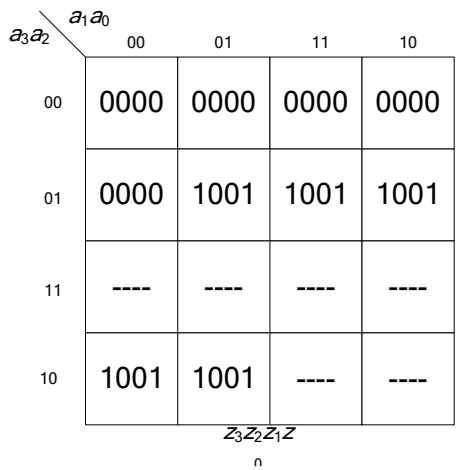
$$z_2 = d_1 \cdot \overline{d_0} + \overline{d_2} \cdot \overline{d_1} \cdot d_0$$
$$z_1 = d_2 \cdot d_0 + \overline{d_2} \cdot \overline{d_1} \cdot \overline{d_0}$$
$$z_0 = \overline{d_0}$$

3) Siano $d_3 \ldots d_0$ i quattro bit che rappresentano la cifra X_{n-1}, e $z_3 \ldots z_0$ 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_1d_0 \ d_3d_2	00	01	11	10
00	1000	1100	0100	0000
01	1001	1101	0101	0001
11	1011	1111	0111	0011
10	1010	1110	0110	0010

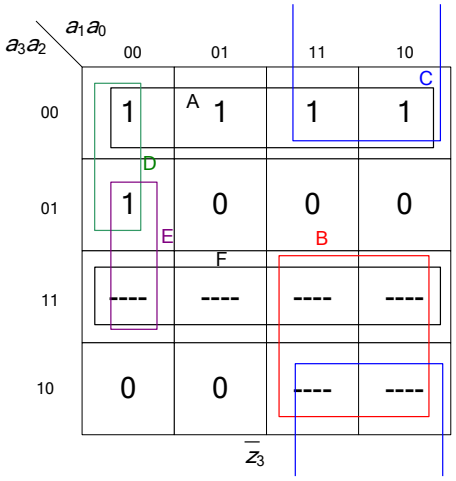
$z_3z_2z_1z_0$

5.7 Soluzione dell'esercizio 4.3.1



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

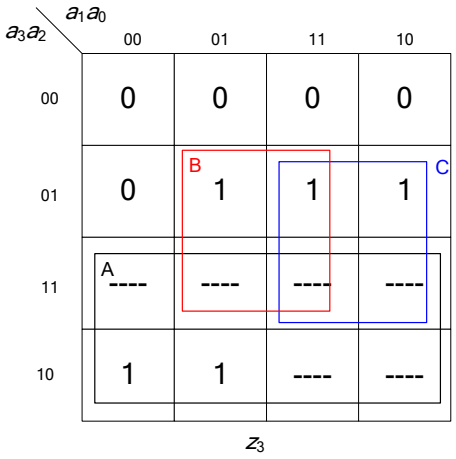


A è l'unico IP essenziale, B, C, F sono assolutamente eliminabili e D, E sono semplicemente eliminabili. Pertanto le sintesi a costo minimo sono due, A, D ed A, E. La prima è:

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

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

Sintesi a porte NAND



I tre implicanti sono tutti essenziali. Esiste una sola sintesi, che è la seguente:

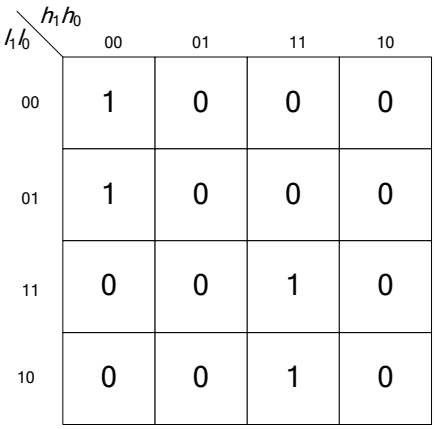
$$z_3 = z_0 = a_3 + a_2 \cdot a_0 + a_2 \cdot a_1$$

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

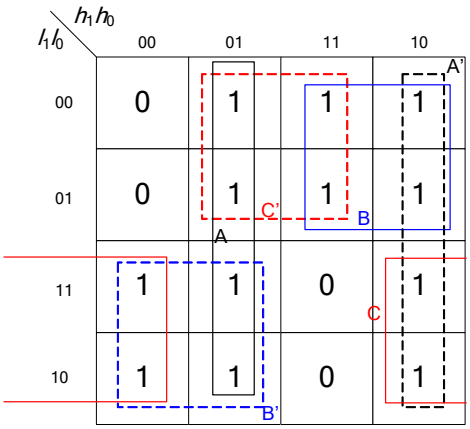
$$= \overline{\overline{a_3} \cdot \overline{(a_2 \cdot a_0)} \cdot \overline{(a_2 \cdot a_1)}}$$

5.8 Soluzione dell'esercizio 4.4.1

Dette h_1h_0 e l_1l_0 le variabili che codificano le cifre di peso $n-1$ ed $n-2$, la mappa di Karnaugh è la seguente:



z



$\overline{z}$

Gli implicanti principali per $\overline{z}$ sono elencati a destra, e sono tutti semplicemente eliminabili. Si osserva che non esistono liste di copertura con due implicanti, 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:

$$\overline{z} = \overline{h_1} \cdot h_0 + h_1 \cdot \overline{l_1} + \overline{h_0} \cdot l_1$$
$$z = \overline{(\overline{h_1} \cdot h_0) + (h_1 \cdot \overline{l_1}) + (\overline{h_0} \cdot l_1)}$$
$$= \overline{\overline{(h_1 + \overline{h_0})} + \overline{(\overline{h_1} + l_1)} + \overline{(h_0 + \overline{l_1})}}$$

$$\overline{z} = h_1 \cdot \overline{h_0} + \overline{h_1} \cdot l_1 + h_0 \cdot \overline{l_1}$$
$$z = \overline{(h_1 \cdot \overline{h_0}) + (\overline{h_1} \cdot l_1) + (h_0 \cdot \overline{l_1})}$$
$$= \overline{\overline{(\overline{h_1} + h_0)} + \overline{(h_1 + \overline{l_1})} + \overline{(\overline{h_0} + l_1)}}$$

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):

x_1x_0 \ y_1y_0	0 (00)	+1 (01)	- (11)	-1 (10)
0 (00)	00 00 0 0	00 00 0 0	-- --	00 00 0 0
+1 (01)	00 00 0 0	00 01 0 1	-- --	10 10 2 2
-- (11)	-- --	-- --	-- --	-- --
-1 (10)	00 00 0 0	10 10 2 2	-- --	00 01 0 1

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 implicanti usati sono tutti essenziali.

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_3z_2 \ z_1z_0	0 (00)	1 (01)	- (11)	2 (10)
0 (00)	00	01	--	--
1 (01)	--	--	--	--
-- (11)	--	--	--	--
2 (10)	--	--	--	11

w_1w_0

Si ottiene immediatamente quanto segue:

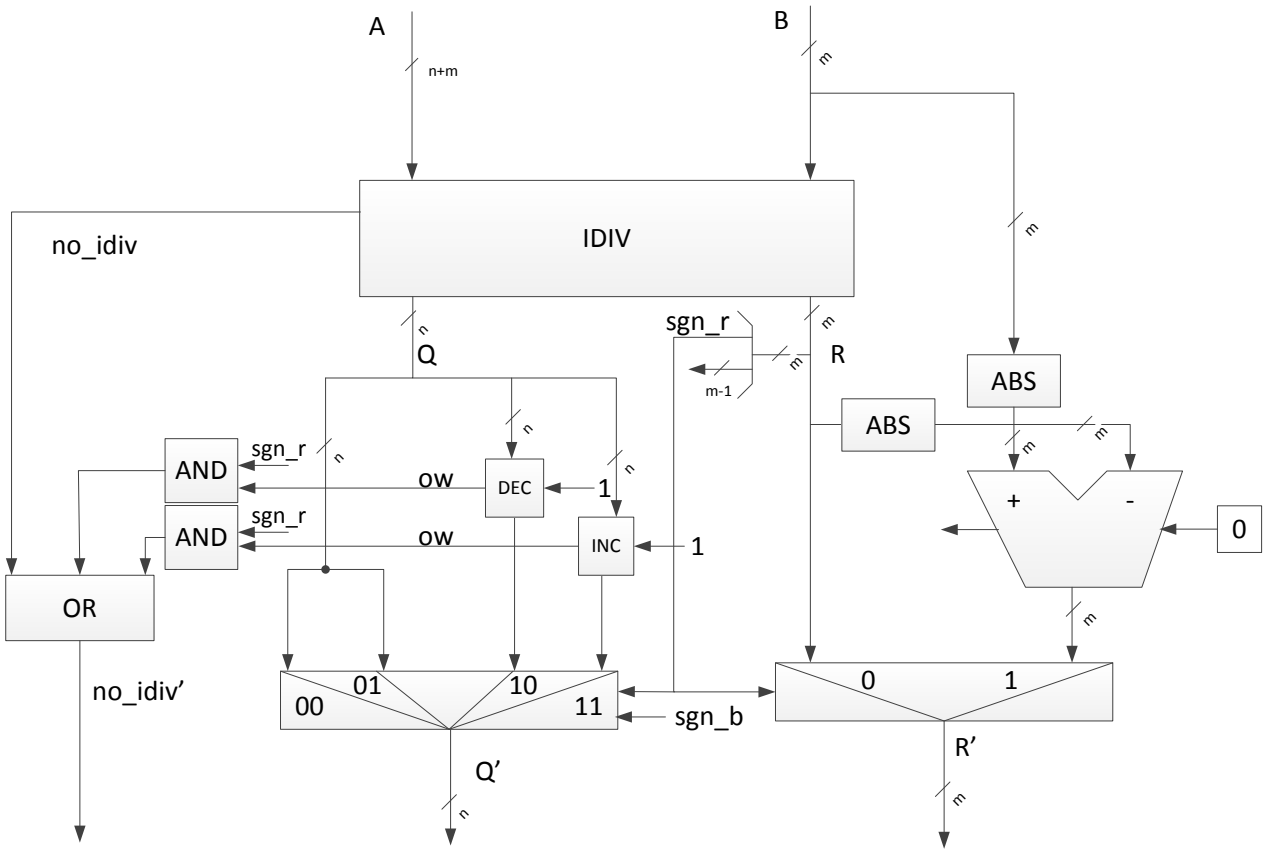
$$w_1 = z_1$$
$$w_0 = z_0 + z_1$$

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.

5.10 Soluzione dell'esercizio 4.8.5

Se il resto è positivo, il risultato richiesto è quello restituito dal modulo IDIV. Se, invece, il resto è negativo, detti q, r i risultati (quoziente e resto) la cui rappresentazione è restituita dal modulo IDIV, è necessario che la rete restituisca le rappresentazioni della coppia $q' = q - \text{sgn}(b)$, $r' = ABS(b) + r = ABS(b) - ABS(r)$, come si può facilmente vedere facendo qualche prova. In quest'ultimo caso, r' è sempre rappresentabile. q', invece, può non essere rappresentabile se l'operazione che lo produce genera overflow. L'uscita *no_idiv'* è quindi dato dall'OR dell'uscita

no_idiv del modulo IDIV e dell'overflow generato dall'operazione scritta sopra. Una possibile implementazione circuitale (non ottimizzata) è la seguente:



6 Altri esercizi svolti

6.1 Esercizio (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.

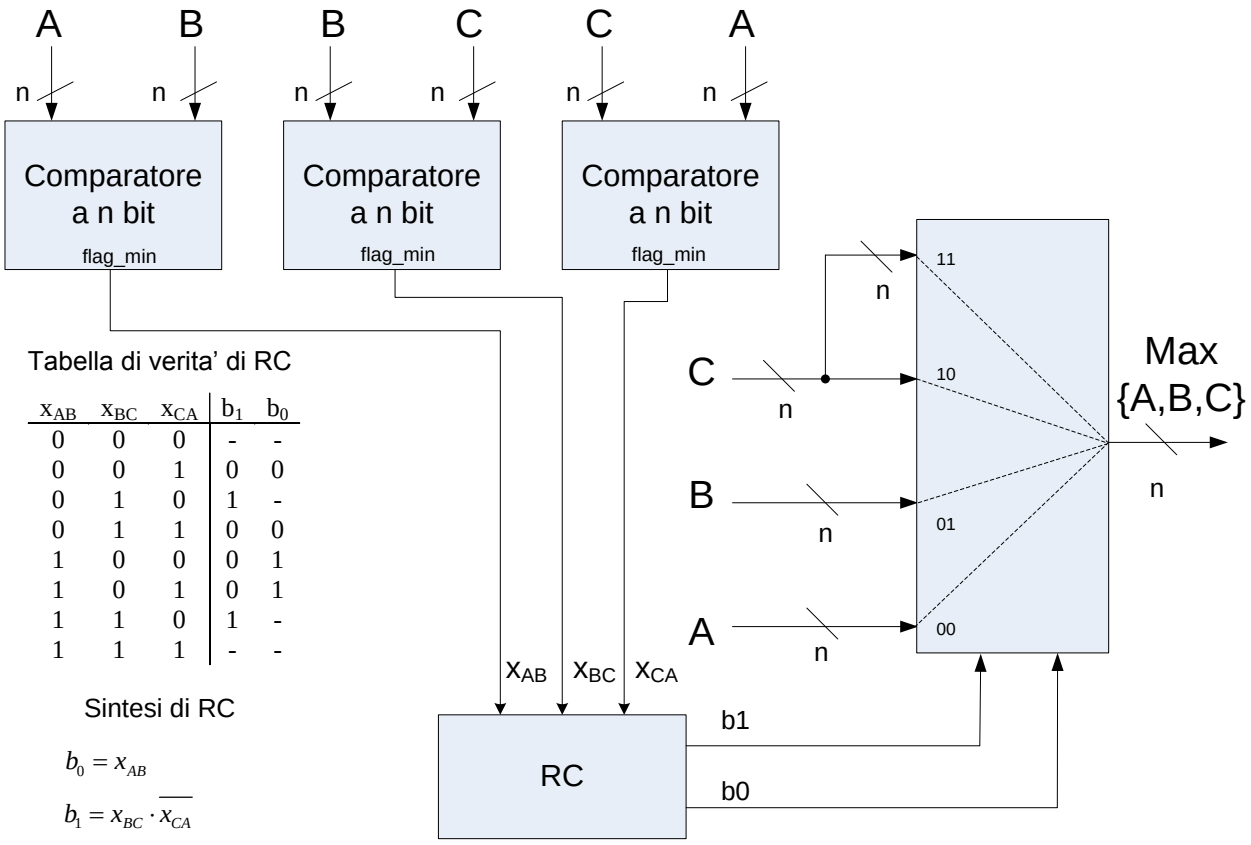


Tabella di verita' di RC

x_{AB}	x_{BC}	x_{CA}	b_1	b_0
0	0	0	-	-
0	0	1	0	0
0	1	0	1	-
0	1	1	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	-
1	1	1	-	-

Sintesi di RC

$$b_0 = x_{AB}$$

$$b_1 = x_{BC} \cdot \overline{x_{CA}}$$

Il comparatore dovrà essere diverso a seconda che si comparino naturali (come richiesto al punto 1) o interi (come richiesto al punto 2).

Soluzioni analoghe ed ugualmente corrette 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 (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 ? 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.*

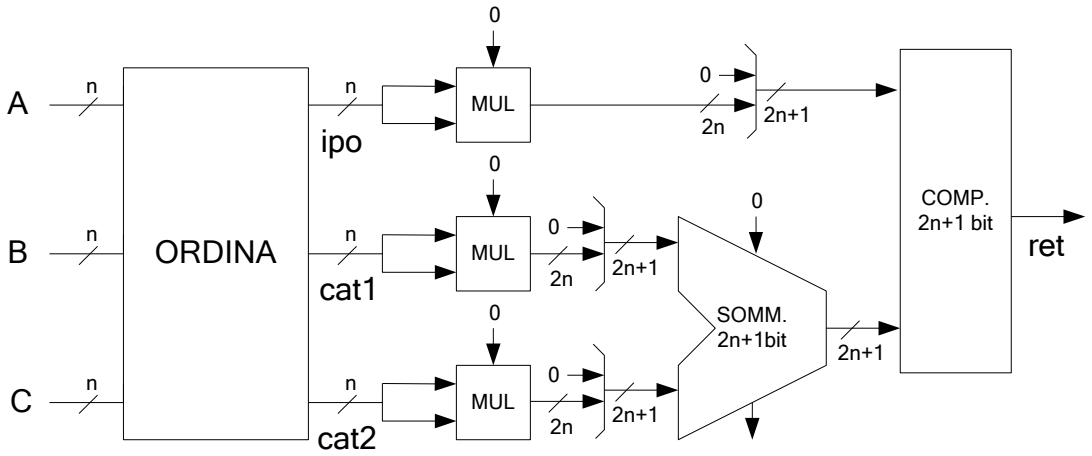
Nota: Si descriva esplicitamente qualunque rete non descritta a lezione.

Domanda facoltativa: Si sintetizzi la rete MAX3

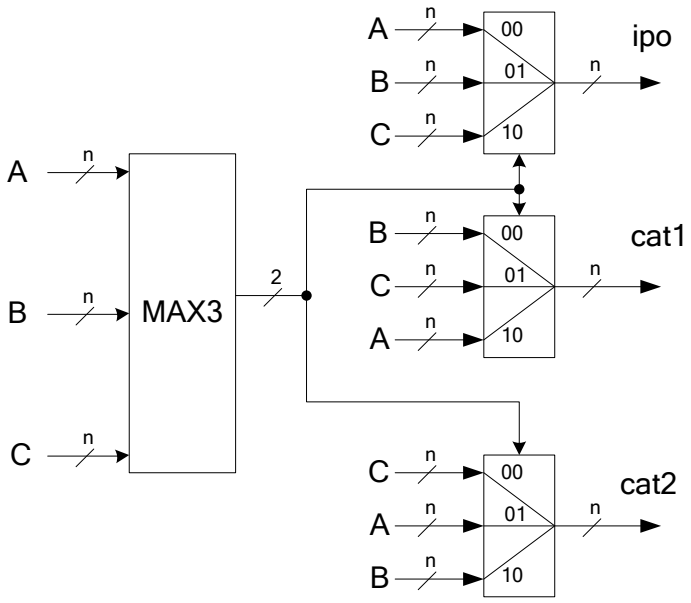
6.2.1 Soluzione

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

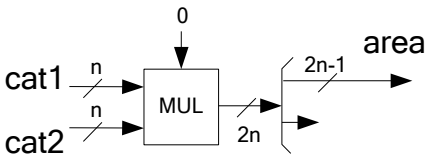
Per produrre l'uscita *ret*, bisogna stabilire se il triangolo è rettangolo o meno. Un triangolo è rettangolo quando i suoi lati verificano il teorema di Pitagora: ciò significa che deve esistere una permutazione dei tre lati A, B, C, per cui ${L_1}^2 + {L_2}^2 = {L_3}^2$. Questo è possibile solo quando 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 (numeri naturali)

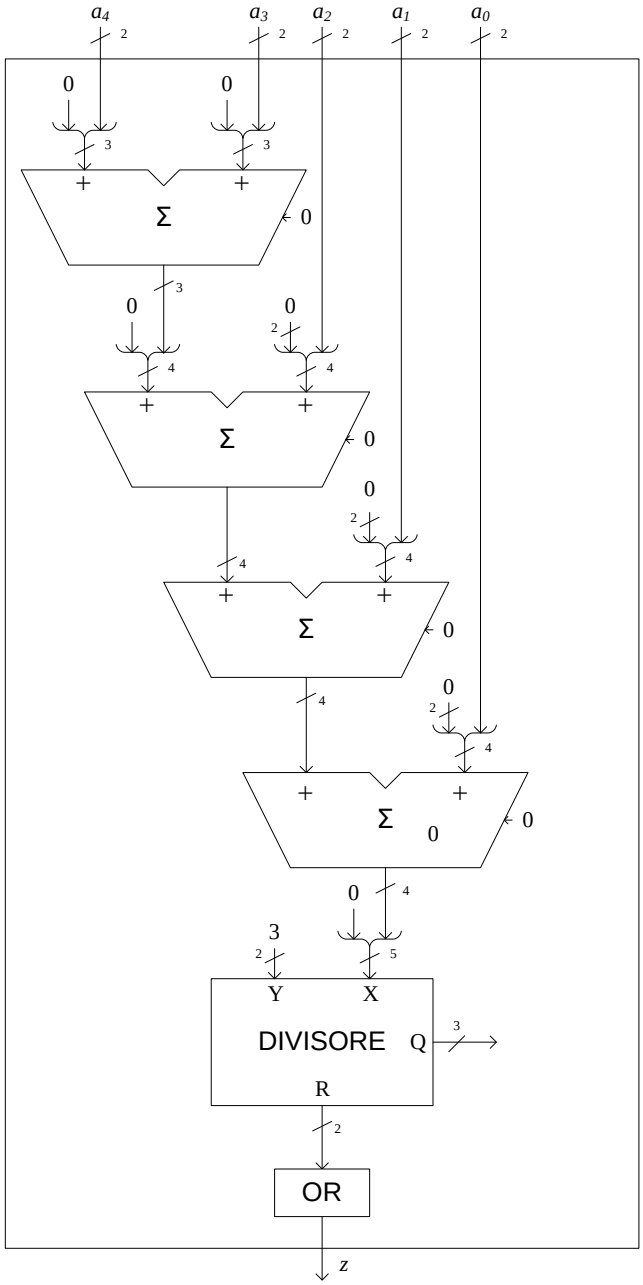
Sia $A = (a_{n-1} \dots a_0)_4$ un numero naturale rappresentato su $n = 5$ cifre in base quattro. Realizzare un circuito che prenda in ingresso le cifre di A e produca in uscita 0 se A è multiplo di tre, 1 altrimenti.

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

Un possibile circuito che soddisfa la specifica è il seguente:



Si può usare anche il carry in uscita e fare la somma su $n-1$ bit

X vale al massimo 15. Allora, $X/3$ può fare anche 5, che sta su 3 bit. Quindi, il quoziente deve poter stare su 3 bit, ma questo è possibile solo se rappresento X su 5 bit, perché altrimenti **non vale** la diseguaglianza necessaria per la divisione

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 $X \leq 15$.

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

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

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.

6.4 Esercizio (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 $b = 7 - 5a$. Esemplificare 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.

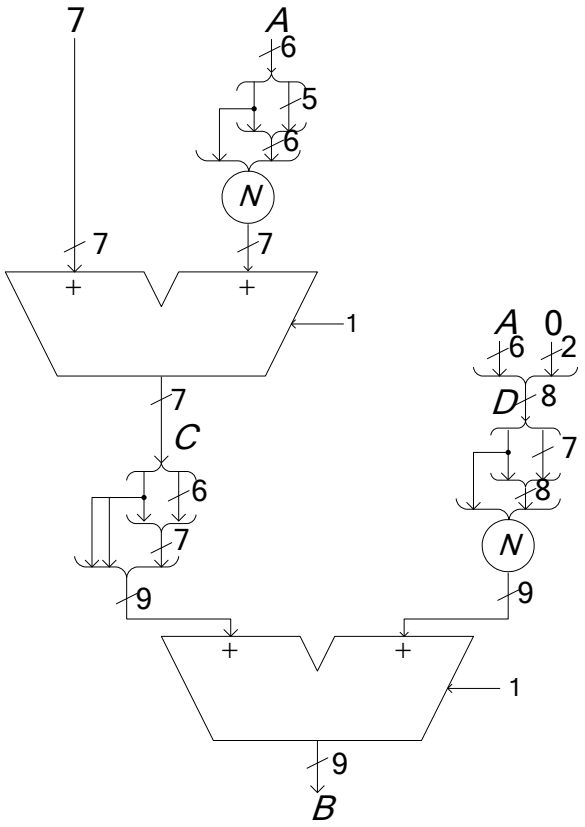
6.4.1 Soluzione

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`

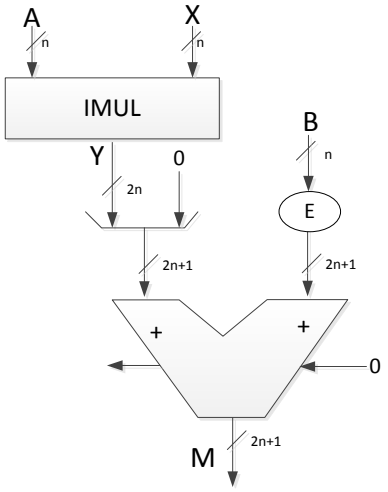
6.5 Esercizio (numeri interi)

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:

b_1 \ a_0x_0	00	01	11	10
0	0	0	1	0
1	1	1	0	1

(m_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 $m_1 = b_1 \oplus a_0 \cdot x_0$.