Macchine Aritmetiche

Addizionatori

Corso di Architettura dei Calcolatori

Seconda Università degli Studi di Napoli
Prof. Salvatore Venticinque

(MEI cap. 5 e 7)

Macchine aritmetiche

- Componenti elementari per il calcolo
- Caratterizzati da strutture modulari
- Parametrizzati secondo:
  - Tipo di aritmetica utilizzata: in modulo, intera, frazionaria, estesa, relativa;
  - Base di numerazione: binaria, decimale
  - Numero di “cifre” degli operandi

Problemi

- Problemi di progettazione:
  - Necessità di struttura modulare per ridurre i costi
  - ->problemi di architettura e interconnessione
  - Esistenza di macchine “tipiche”
  - -> si ottengono macchine a partire da altre macchine “più potenti”
  - Scelta degli algoritmi migliori per realizzare le funzioni di cui si necessita

Cosa vedremo

- Addizionatori
  - Full adder, half adder, addizionatore b^n, addizionatore veloce, addizionatore carry lookahead, addizionatore in complementi diminuiti, sottrattori
- Moltiplicatori
  - Moltiplicatore seriale, moltiplicatore parallelo, moltiplicatore binario veloce
- Divisori
  - Divisore con e senza restoring

Addizionatori modulo M

- Un addizionatore modulo M (full adder) è una macchina che esegue le operazioni:
  - $S=|X+Y+r|_M$
  - $R=[(X+Y+r)/M]$

  Dove:
  - X e Y sono gli operandi
  - S è la somma
  - R è il riporto uscente, r è il riporto entrante
  - M è il modulo
- X, Y, S sono segnali che rappresentano numeri in aritmetica modulo M, R e r sono bit
- Definiamo anche la somma Z=X+Y+r

Rappresentazione

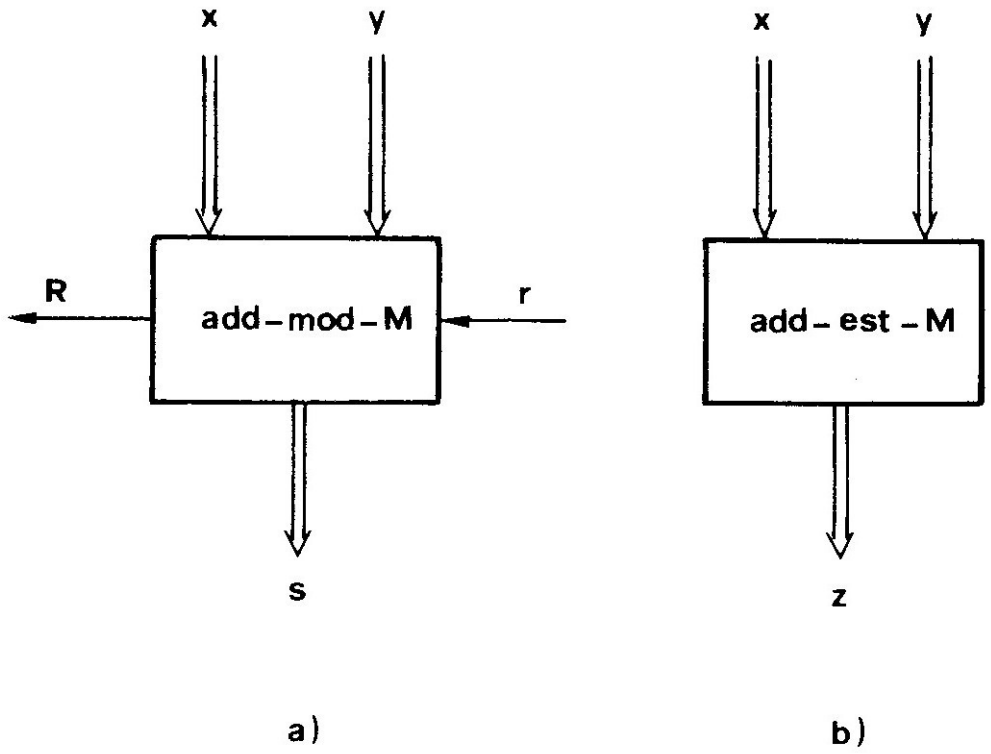


Figura 2.1 – Addizionatori: a) in modulo; b) esteso.

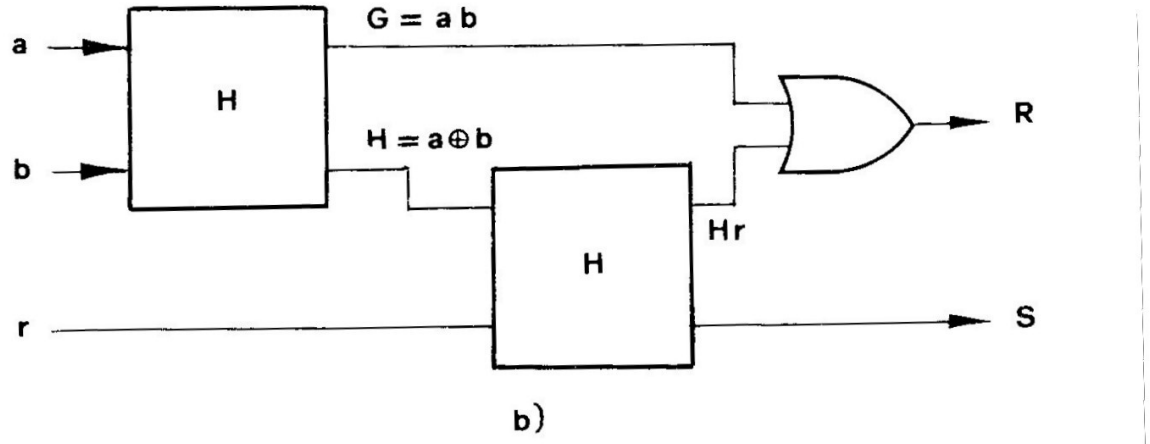
Interpretazione

- Se X, Y e S sono in aritmetica di interi modulo M, R è un segnale di overflow
- Se X, Y e S sono in aritmetica estesa la somma è sempre definita ed è data da Z=MR+S
- Se X, Y e S sono in aritmetica di frazionari non varia nulla purchè si ponga:
  - X'=X/M
  - Y'=Y/M
  - S'=S/M
  - Z'=Z/M

Realizzazione di un full adder

Si può realizzare un full adder a partire da due half adder che eseguano:

- $H=|X+Y|_M$, $G=[(X+Y)/M]$
- $S=|H+r|_M$,$R' =[(H+r)/M]$
- R=G or R'



b)

Figura 11.3 – Schemi logici di addizionatori completi realizzati con semiaddizionatori.

Equazioni

- Per l'half adder:

  $H = a \text{ xor } b = a\bar{b} + \bar{a}b$

  $G = ab$

- Per il full adder:

  $S = a \text{ xor } b \text{ xor } r = \bar{a}\bar{b}r + \bar{a}b\bar{r} + a\bar{b}\bar{r} + abr$

  $R = \bar{a}br + a\bar{b}r + ab\bar{r} + abr = ab + br + ar =$

  $= ab + r(a+b)$

- Full adder con half adder:

  $S = a \text{ xor } b \text{ xor } r = H \text{ xor } r$

  $R = ab + r(a \text{ xor } b) = G + rH$

Schema full adder

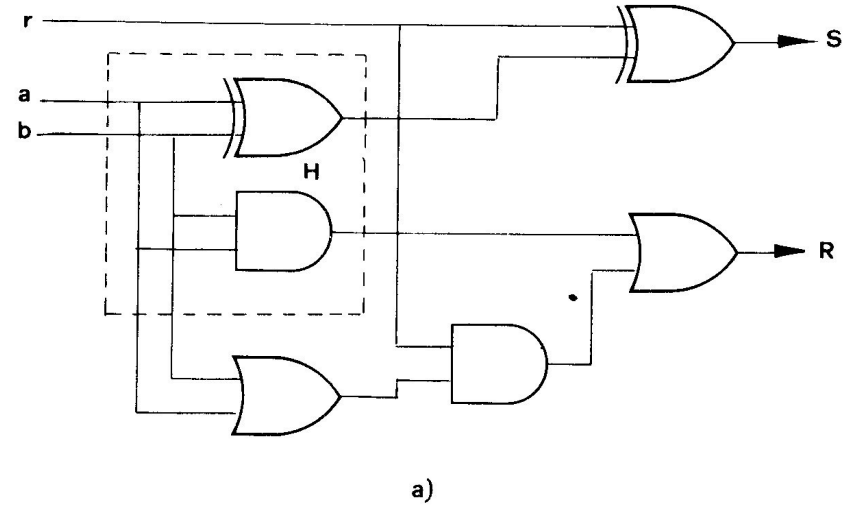


Figura 11.3 – Schemi logici di addizionatori completi realizzati con semiaddizionatori.

Realizzazione adder binari

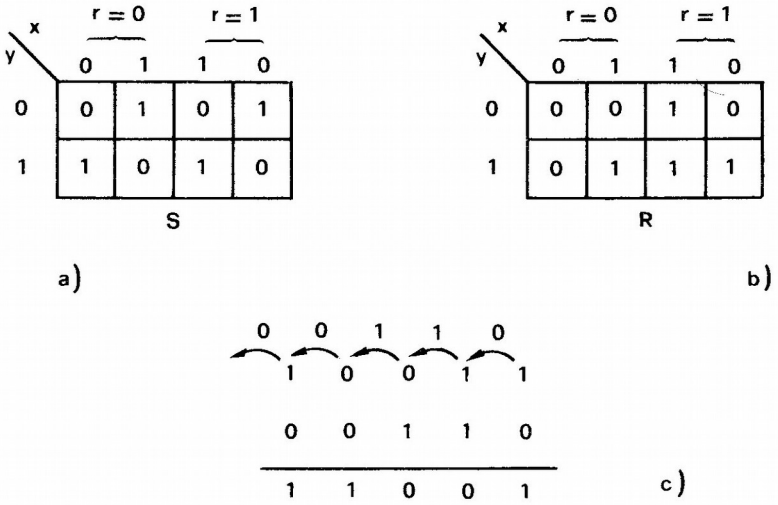


Figura 3.2 – Addizione binaria.

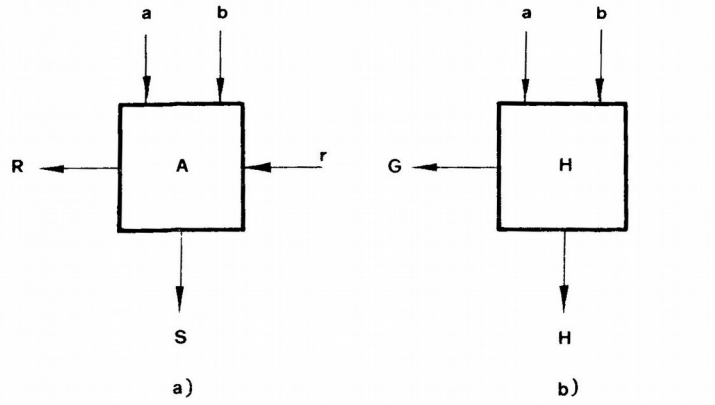


Figura 11.1 – Addizionatori binari elementari: a) addizionatore completo; b) semiaddizionatore.

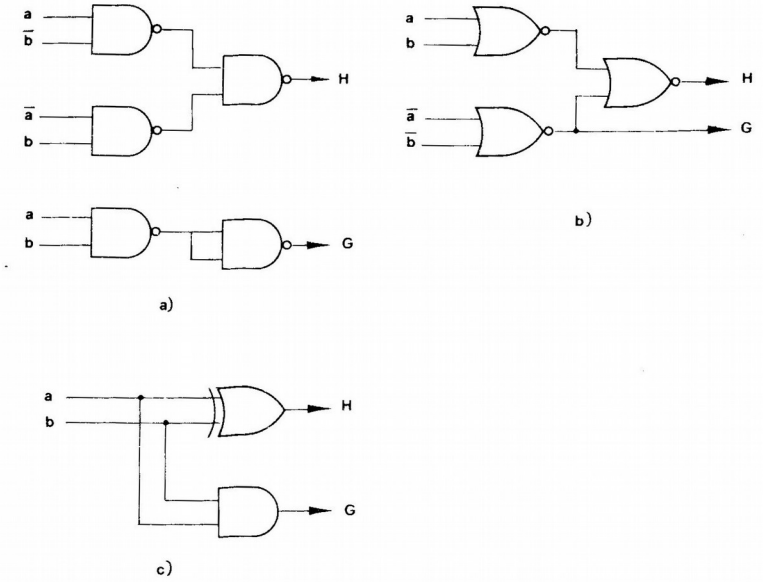
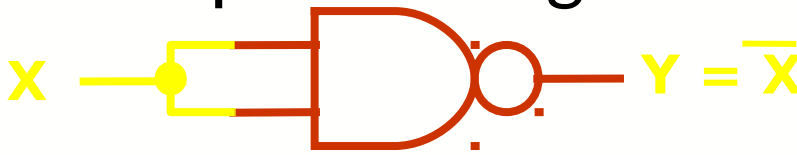


Figura 11.2 – Schemi logici di semiaddizionatori: a) a *nand*; b) a *nor*; c) in logica mista.

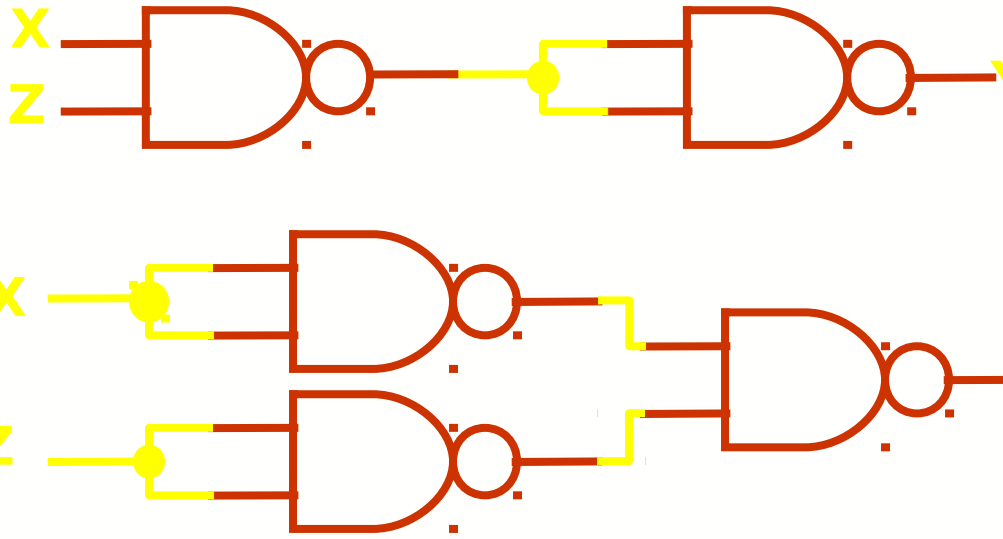
Proprietà della porta NAND (NOR)

- Utilizzando solamente porte NAND (NOR) è possibile realizzare qualunque rete logica

- NOT



- AND



- OR

Addizionatore di interi positivi

- Si ipotizzino:
  - Operandi rappresentati in base b^n
  - L'esistenza di un full adder in base b
- E' possibile costruire un addizionatore in base b^n applicando l'algoritmo classico per l'addizione

Algoritmo

- Se:
  - X e Y sono gli operandi
  - S è la somma
  - T è il riporto uscente, t è il riporto entrante
  - M è il modulo, $M= b^n$
  - Col pedice i indico le cifre i-esime e i segnali relativi all'i-esimo adder modulo b

allora l'algoritmo è:

for (i=0 to n-1) {

if i=0 then r_i=t else $r_i = R_{i-1}$

addiziona con l'adder modulo b X_i,Y_i,r_i e ottieni S_i e R_i

}

T=R_i

Funzionamento e schema

- Si sfrutta l'addizione "di cifra"
- Si può usare uno schema seriale o parallelo

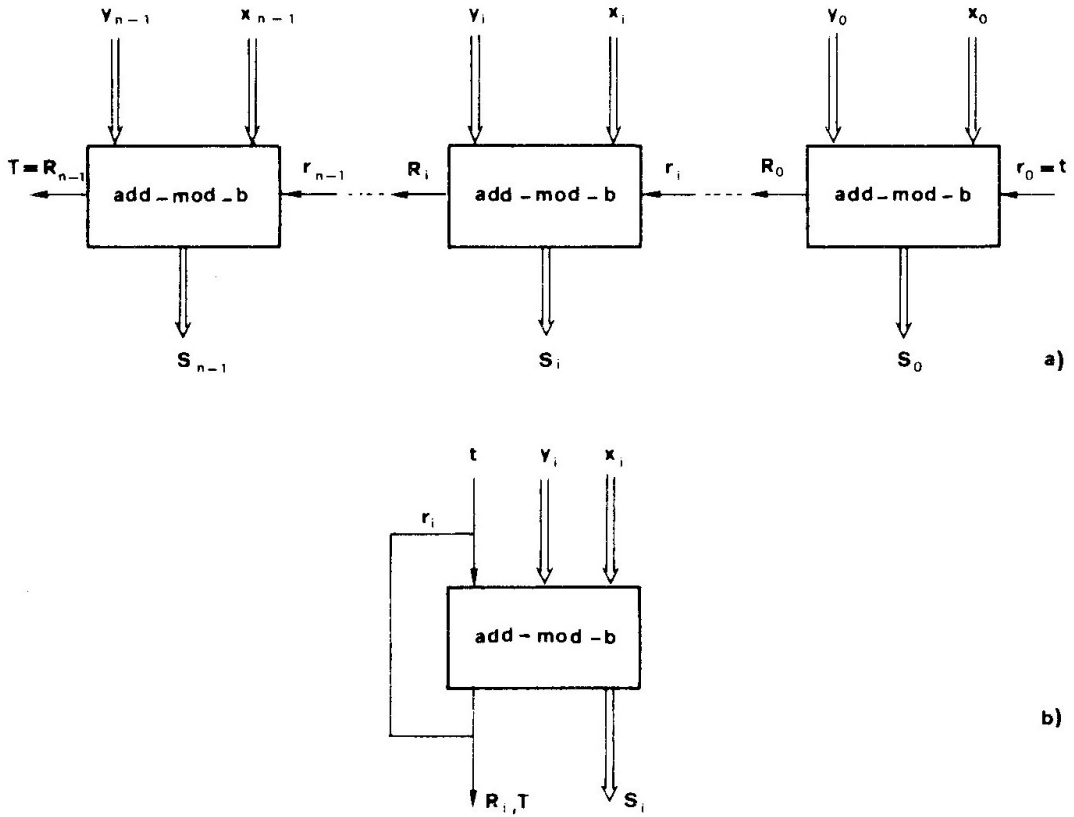


Figura 3.1 – Architetture di un addizionatore in modulo: a) parallela; b) seriale.

Addizionatore binario parallelo

- Addizionatore modulo b^n con b=2
- Struttura interna a n stadi formati da un full adder binario

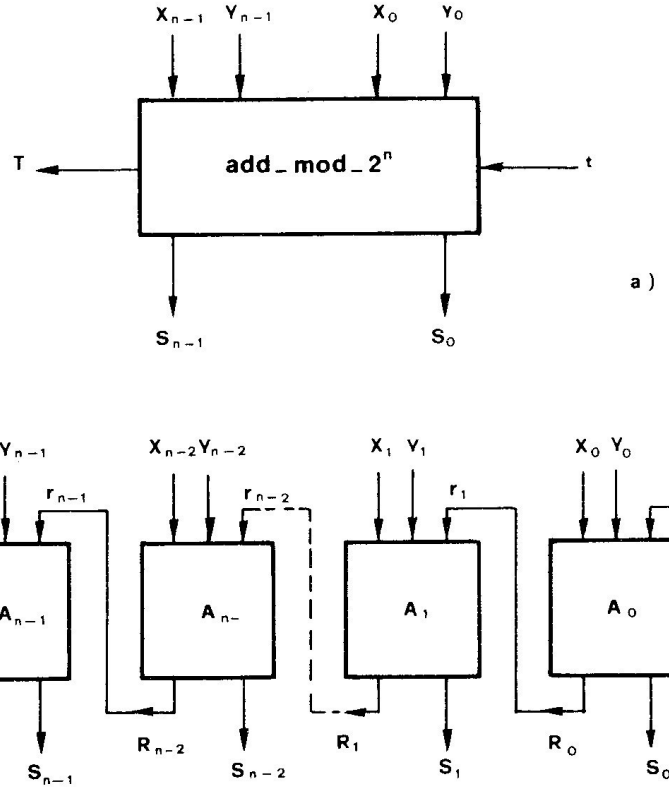


Figura 13.1 – Addizionatore binario parallelo: a) schema terminale; b) struttura interna.

Considerazioni sul riporto

- G è detto *riporto generato* perché esprime se il riporto R nasce nello stesso addizionatore a prescindere da r
- P = H = a xor b viene detto *propagazione del riporto* ovvero la condizione per cui R=r
- La forma R=G+rP evidenzia i contributi
- Si può definire Z=$\underline{ab}$ come *condizione di riporto ucciso*
- Si può definire un segnale ausiliario di *non-riporto uscente* N = $\underline{R}$ = Z + nP

Problema dei ritardi

- Il tempo di risposta Δt dipende dal tempo ε dei singoli stadi e dal tempo occorrente alla propagazione dei riporti t_{pr}
- t_{pr} dipende dai valori degli addendi X e Y poiché in ogni stadio il riporto viene:
  - o generato ($G_i = X_i Y_i = 1$),
  - o ucciso ($Z_i = \underline{X_i}\,\underline{Y_i} = 1$),
  - o propagato ($P_i = X_i$ xor $Y_i = 1$)

  e nei primi due casi il riporto è noto dopo un tempo ε dall'applicazione degli ingressi mentre nel terzo è noto dopo un tempo ε dalla produzione del riporto dello stadio precedente

Ritardo massimo e minimo

- Il ritardo massimo (pari a nε) si ha quando $\Pi_{0<i<n}P_i=1$ ovvero si ha un riporto entrante propagato per tutti gli stadi
- Il ritardo minimo (pari a ε) si ha quando $\Pi_{0<i<n}G_i+Z_i=1$ ovvero quando in tutti gli stadi il riporto è generato o ucciso
- L'uscita dell'addizionatore visto è quindi valida dopo un tempo nε perché bisogna sempre considerare per il tempo di risposta Δt il caso peggiore

Segnale di fine addizione

- Per migliorare le prestazioni e ridurre il tempo di risposta Δt al minimo possibile serve un segnale che indichi quando effettivamente finisce l'addizione
- Sfruttando i segnali R_i e N_i di un full adder dotato di segnali di non riporto entrante e uscente, per ogni stadio vale:

$$R_i = G_i + r_i P_i = G_i + R_{i-1} P_i$$

$$Ni = Z_i + n_i P_i = Z_i + N_{i-1} P_i$$

- Quindi la funzione di completamento vale in ogni stadio

$$F_i = R_i + N_i = G_i + Z_i + (R_{i-1} + N_{i-1}) P_i$$

- ed è vera o dopo ε o dopo ε dall'arrivo di un segnale dallo stadio precedente (quindi il prima possibile) e $F = \Pi F_i$ è la funzione cercata

Carry lookahead

- Il metodo carry lookahead serve ad eliminare del tutto la propagazione dei riporti
- Si usa una rete che calcola i riporti indipedentemente dagli adder delle cifre usando solo X, Y e t
- Per ogni stadio il riporto è ottenuto come

  $r_i = R_{i-1} = G_{i-1} + P_{i-1}R_{i-2} =$

  $= G_{i-1} + P_{i-1}(G_{i-2} + P_{i-2}R_{i-3}) =$

  $= G_{i-1} + P_{i-1}G_{i-2} + P_{i-1}P_{i-2}G_{i-3} + \ldots + P_{i-1}P_{i-2}\ldots P_0 t$
- che ha il vantaggio di essere una rete a 2 livelli
- I termini G e P possono essere generati dagli stessi addizionatori (ad esempio realizzati con half adder)
- Il tempo di risposta complessivo è costante ed è dato dalla somma del ritardo di una rete che calcola r_i e del ritardo di un adder

Schema

- Per *n* grande la rete lookahead diventa irrealizzabile
- si usa uno schema modulare che lavora per parti

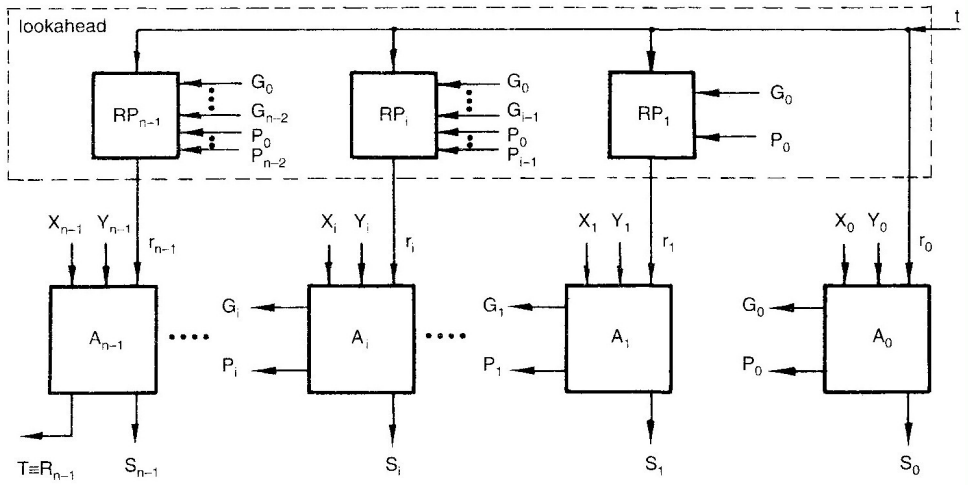


Figura 13.2 – Addizionatore binario parallelo con carry-lookahead.

addizionatori a rapida propagazione dei riporti

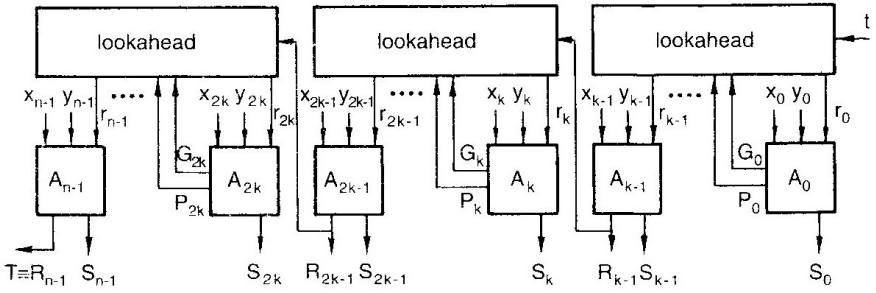


Figura 13.3 – Addizionatore binario a rapida propagazione dei riporti ($m = 3$).