



**Macchine per L'Elaborazione dell'Informazione -
8. Macchine Aritmetiche e Rappresentazioni**

Your Name
Your Title

Your Organization (Line #1)
Your Organization (Line #2)

Lecture Outline

- Rappresentazione degli Interi
- Rappresentazione dei FP
- Macchine aritmentiche semplici
- Addizionatori
- Moltiplicatori
- Divisori

Rappresentazione dei numeri Interi

- Funzioni definite sugli interi :
 - Funzione Parte Intera di un numero Reale
 - $x \in \mathbb{R}$ $y = [x]$ con $y \in \mathbb{N}$
 - $[7.9]=7$ $[7]=[7]$ $[3.01]=3$ $[-7.8]=-8$ $[-7]=-7$
 - Se x non è intero $\Rightarrow [-|x|] = -([|x|]+1)$
 - Funzione Parte Intera per eccessi:
 - $\{[x]\} =$
 - $[x]$ se $x \in \mathbb{N}$
 - $[x]+1$ se x non è in \mathbb{N}
 - Funzione Quoziente (della divisione di x per M):
 - $x, M \in \mathbb{N}$ e $M > 0$: $Q = [x/M] = [-|x/M|] = -([|x/M|]+1)$
 - $[7/10]=0$ $[-7/10]=-1$ $[17/10]=1$ $[-17/10]=-2$

Rappresentazione dei numeri Interi

- Funzione Quoziente (per difetto):
 - q in \mathbb{N} e posto $|x| < M$ si ha
 - $(x+qM)/M = x/M + q = q+e$ con $0 \leq |e| < 1$
 - Visto che $[q+e] = q$ se $0 \leq e < 1$ e $[q+e] = q-1$ se $-1 \leq e < 0$
 - $q \leq q+e < q+1$
 - $q-1 < q+e < q$
 - Allora:
 - x in $[0, M) \Rightarrow [(x+qM)/M] = q$ e
 - x in $(-M, 0] \Rightarrow [(x+qM)/M] = q-1$
- Funzione Quoziente (per eccesso):
 - $Q' = \{[x/M]\}$

Rappresentazione dei numeri Interi

- Funzione Resto modulo M
 - $|x|_M = x - [x/M] * M = x - Q * M$
 - Essendo Q definito per difetto nei numeri relativi
 - Il resto risulta un numero non negativo
 - $|x|_M$ in $[0, M)$
 - x in $[0, M) \Rightarrow |x|_M = |x|$
 - x in $(-M, 0] \Rightarrow |x|_M = M - |x| = M + x$
 - x in $[0, M) \Rightarrow |x + qM|_M = x + qM - qM = x = |x|_M$
 - x in $(-M, 0] \Rightarrow |x + qM|_M = x + qM - (q-1)M = M + x = |x|_M$
 - Quindi in ogni caso :
 - $|x + qM|_M = |x|_M$
 - Il resto modulo M di x e' invariante se si aggiunge o sottrae ad x un qualsiasi multiplo di M

Rappresentazione dei numeri Interi

- Il resto mod.M è il complemento ad M del resto mod.M di $-x$
 - $|-x|_M = M - |x|_M$ es: $|7|_{10} = 7$ $|-7|_{10} = 3 = 10 - 7$
- Il resto del resto di x e' il resto di x
 - $||-x|_M|_M = |-x|_M$
- Nell'intervallo x in $[0, 2M)$ posto $r = |x|_M$ si ha
 - If $x < M$ then $Q = 0$ else $Q = 1$
 - If $x < M$ then $r = x$ else $r = x - M$
- Nell'intervallo x in $(-M, M)$
 - If $x \geq 0$ then $Q = 0$ else $Q = -1$
 - If $x \geq 0$ then $r = x$ else $r = M + x$
- E' possibile definire una *aritmetica Modulo M*
 - Il risultato R di ogni operazione e' il resto mod.M di R

Rappresentazione dei numeri Interi

- Resto per eccessi
 - Prima: $r = |x|_M$ se diverso da 0 e approssimazione di Q per difetto
 - Analogamente, se $Q' = \{[x/M]\}$ si può definire r' *resto negativo*.
 - Coincide con r se $r=0$
 - Altrimenti:
 - $Q' = Q + 1$
 - $R' = r - M$
 - Es: $x/M = 7/10 \Rightarrow Q' = 1 \quad r' = -3 \quad Q = 0 \quad r = 7$
 - $x/M = (-7)/10 \Rightarrow Q' = 0 \quad r' = -7 \quad Q = -1 \quad r = 3$

Rappresentazione dei numeri Interi

- Resto minimo
 - Il minore in valore assoluto tra r e r'
- Divisione tra interi
 - Trovare x ed y :
 - $x=y*Q+r$
 - Se $y < 0$ se $Q=[x/y]$ il resto è negativo
 - Se $Q=\{[x/y]\}$ il resto è positivo
 - Le coppie (Q,r) :
 - | x | y | $[x/y]$ | $\{[x/y]\}$ | Q | Q' | Q'' (sui reali) |
|-----|-----|---------|-------------|------|-------|-------------------|
| 13 | 7 | 1,6 | 2, -1 | 1,6 | 2,-1 | 1,6 |
| -13 | 7 | -2,1 | -1,-6 | -2,1 | -1,-6 | -1,-6 |
| 13 | -7 | -2,1 | -1,6 | -1,6 | -2,-1 | -1,6 |
| -13 | -7 | -1,-6 | 2,1 | 2,1 | 1,-6 | 1,-6 |

Rappresentazione dei numeri Interi

- Rappresentazione dei numeri interi:
 - b : base
 - $X_{n-1}, X_{n-2}, \dots, X_0$ X_i in $[0, b)$ le CIFRE
 - $x = \sum_{i=0, i \leq n-1} X_i * b^i$
- Proprietà delle rappresentazioni degli interi positivi
 - Estensione:
 - Semplicemente aggiungendo 0 a sinistra
 - Rappresentazione di b^k :
 - Numero con 1 al posto k e 0 altrove
 - Rappresentazione di $x * b^k$ e $\lfloor x / b^k \rfloor$
 - Moltiplicazione: shl di k posizioni
 - Divisione: shr di k posizioni

Rappresentazione dei numeri Interi

- Proprietà delle rappresentazioni degli interi positivi
 - Rappresentazione del complemento a b^n :
 - Detto $c = b^n - x$ si pone:
 - $X_0 = X_1 = \dots = X_{i-1} = 0$
 - $C_j = 0$ per $j < i$
 - $C_i = b - X_i$
 - $C_j = b - X_j - 1$ per $j > i$
 - Ovvero: si lasciano inalterati gli 0 in coda, si effettua il complemento a b della prima cifra da destra $\neq 0$ e il complemento a $b-1$ delle altre cifre

Rappresentazione dei numeri Interi

- Proprietà delle rappresentazioni degli interi positivi
 - Rappresentazione di $b^k - 1$:
 - $x = b^k - 1 \iff X_j = b - 1$ per $j < k$; $X_j = 0$ per $j \geq k$
 - Rappresentazione del complemento a $b^n - 1$:
 - Detto $c = b^n - x - 1$
 - For $i = 0$ to $n - 1$ $C'_i = b - X_i - 1$
 - (in binario si complementano tutte le cifre)

Numeri Relativi: rappr. Segno e Modulo

- x in $[M1, M2]$ con $M1 < 0$, $M2 > 0$
- $M1$ e $M2$ sono uguali in valore assoluto o differiscono al più di una unità
- Una cifra codifica il segno.

Classi dei resti modulo M; rappr. Per complementi

- $\text{congr}_M(x,y)$ se hanno lo stesso resto in modulo M
- $\text{congr}_M(x,y) \Leftrightarrow \exists q: x = y + q \cdot M$
- E' una classe di equivalenza
 - Riflessiva
 - Simmetrica
 - Transitiva
- $|x \pm y|_M = | |x|_M \pm |y|_M |_M$
- $|x \cdot y|_M = | |x|_M \cdot |y|_M |_M$

Complementi alla base

- x in $[-b^n/2, b^n/2)$; X in $[0, b^n)$
- $X =$
 - $|x|$ se $x \geq 0$
 - $b^n - |x| = b^n + x$ per $x < 0$
- x in $[-b^n/2, b^n/2 - 1]$
- Condizione di overflow
 - $X < 0$ and $X < b^n/2$ or $x > 0$ and $X \geq b^n/2$
- Il segno si può trarre dalla prima cifra della rappresentazione avendosi:
 - $x \geq 0 \iff x < b^n/2 \iff X_{n-1} < b/2$
- Estensione della rappresentazione:
 - Estensione con $(b-1)$ in testa

Complementi alla base

- Rappresentazione di $-b^k$:
 - $x = -b^k \iff X_i =$
 - $b-1$ per $i \geq k$
 - 0 per $i < k$
- Prodotto per b : shift con estensione (arithmetic shift)
 - $\text{ashl0}(X)$
 - $\text{Ashl0}(X) \iff \text{shl}(X,0)$
- Divisione per b
 - $\text{ashr0}(X)$
 - $\text{ashr0}(X) \iff \text{if } X_{n-1} < b/2 \text{ then } \text{shr}(X,0) \text{ else } \text{shr}(X,b-1)$

Complementi diminuiti

- x in $[-(b^n/2-1), b^n/2-1]$; X in $[0, b^n)$
- $X =$
 - $|x|$ se $x \geq 0$
 - $b^n - |x| - 1 = b^n + x$ per $x < 0$
- 2 Zeri
- Condizione di overflow
 - $|x| > b^n/2$
- Il segno si può trarre dalla prima cifra
- Estensione della rappresentazione:
 - Estensione con $(b-1)$ in testa

Complementi diminuiti

- Rappresentazione di $-b^k$:
 - $x = -b^k \iff X_i =$
 - $b-2$ per $i=k$
 - $b-1$ per $i \neq k$
- Prodotto per b : shift con estensione (arithmetic shift)
 - $ashl1(X)$
 - $Ashl1(X) \iff$ if $X_{n-1} < b/2$ then $shl(X,0)$ else $shl(X,b-1)$
- Divisione per b
 - $ashr1(X)$
 - $ashr1(X) \iff$ arrotondamento di x/b

Rappresentazione per eccessi (biased)

- x in $[-(b^n/2), b^n/2)$; $ro(x)$ in $[0, b^n)$
- $Ro = x + b^n/2$

Rappresentazione virgola mobile

- Un numero non intero può essere rappresentato in infiniti modi quando utilizziamo la notazione esponenziale:
- – Es.
 - $34.5 = 0.345 \cdot 10^2 = 0.0345 \cdot 10^3 = 345 \cdot 10^{-1}$
- Questo formato prende il nome di floating-point (virgola mobile)
- Essendo infinite le rappresentazioni è necessario sceglierne una di riferimento (rappresentazione normalizzata)

Rappresentazione virgola mobile

- Nei numeri decimali possiamo ad es. considerare come normalizzata la rappresentazione in cui la parte intera è formata da una sola cifra;
- – es. $3.45 \cdot 10^1$
- Possiamo quindi distinguere in numero
 - le cifre significative (significando o mantissa)
 - l'esponente da dare alla base

Rappresentazione virgola mobile

Nel nostro esempio:

$$3.45 \cdot 10^1$$

mantissa



esponente

Rappresentazione virgola mobile

- In generale un numero rappresentato in f.p. assume la forma:

- $$d_0 \cdot d_{-1} d_{-2} d_{-3} \dots d_{-(p-1)} \cdot b^e$$

- il cui significato è

- $$(d_0 + d_{-1} b^{-1} + d_{-2} b^{-2} + d_{-3} b^{-3} + \dots d_{-(p-1)} b^{-(p-1)}) \cdot b^e$$

- p è la precisione della rappresentazione

- Nel caso della base 2 possiamo rappresentare soltanto le cifre dopo la virgola risparmiando un bit

Rappresentazione virgola mobile

- Un numero binario rappresentato in f.p. assume la forma normalizzata:

- $$1.d_{-1}d_{-2}d_{-3}\dots d_{-(p-1)} \cdot b^e$$

- si rappresenteranno esplicitamente:
 - i bit dopo la virgola;
 - l'esponente;
 - il segno del numero.

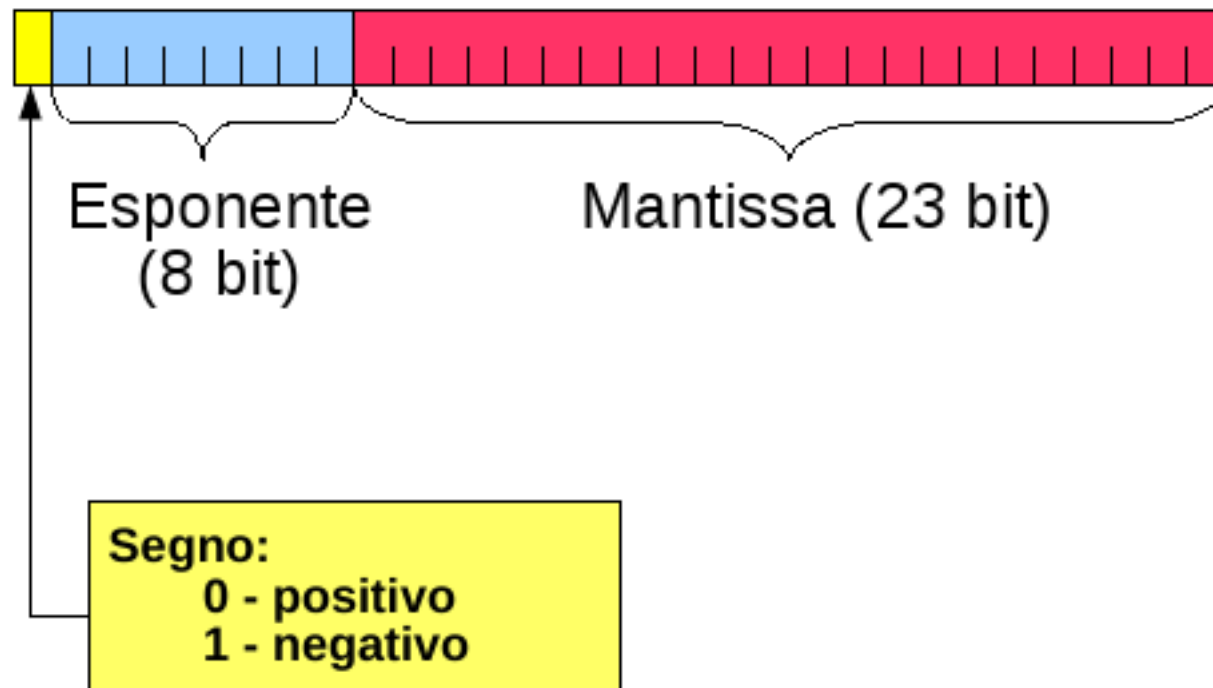
Rappresentazione virgola mobile

- Si può facilmente osservare che in tale rappresentazione:
 - i numeri non sono equispaziati fra di loro (il peso del LSB della mantissa dipende dall'esponente)
 - la finitezza della rappresentazione introduce la possibilità sia di overflow (valore assoluto troppo grande) che di underflow (valore assoluto troppo piccolo)

IEEE-754

- E' lo standard internazionale adottato per rappresentare i numeri reali
- Prevede la rappresentazione di
 - numeri in f.p. Normalizzati
 - alcuni numeri denormalizzati
 - infinito (positivo e negativo)
 - NaN (not a number) per rappresentare
 - risultati indeterminati delle operazioni (come 0/0 ecc.)

IEEE-754



IEEE-754

0 10000100 000101000000000000000000000000

$$\text{Es. } 34.5_{10} = 100010.1_2 = 1.000101_2 \cdot 2^5 = 1.000101_2 \cdot 10_2^{101}$$

Segno: 0 (positivo)

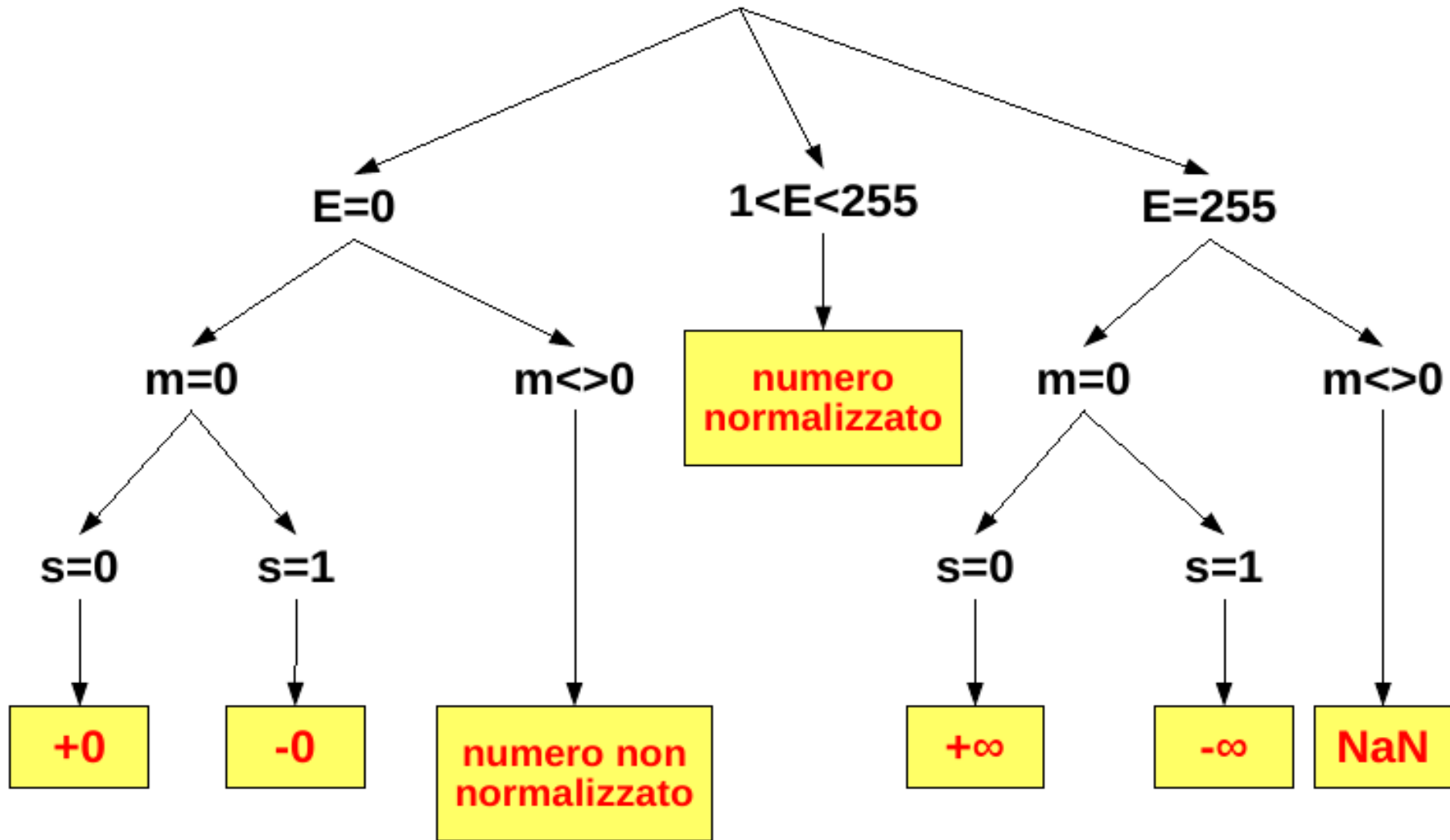
Mantissa: 000101 (la parte intera pari a 1 si sottintende)

Esponente: $5+127 = 132 = 10000100_2$
(rappresentazione in **eccesso 127**)

IEEE-574

- Non tutti i valori possibili di E e m sono utilizzati
- Se $E=0$ il numero può essere nullo o non normalizzato
- Se $E=255$ (tutti 1) si rappresenta un infinito (positivo o negativo) o un non numero (NaN)
- Negli altri casi ($0 < E < 255$) si rappresenta un numero normalizzato
- I numeri non normalizzati si utilizzano per riempire lo spazio tra lo 0 e il più piccolo numero normalizzato

IEEE-754

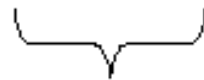


La rappresentazione dell'esponente in eccesso 127 (**biased**) consente una maggior facilità di progettazione dei circuiti della ALU: il confronto avviene, **a parte il segno**, confrontando semplicemente il resto del numero **lessicograficamente**.

– es: il primo numero è più piccolo del secondo

• $12.34E-03 = 0\ 01111000\ 10010100010110110110110$

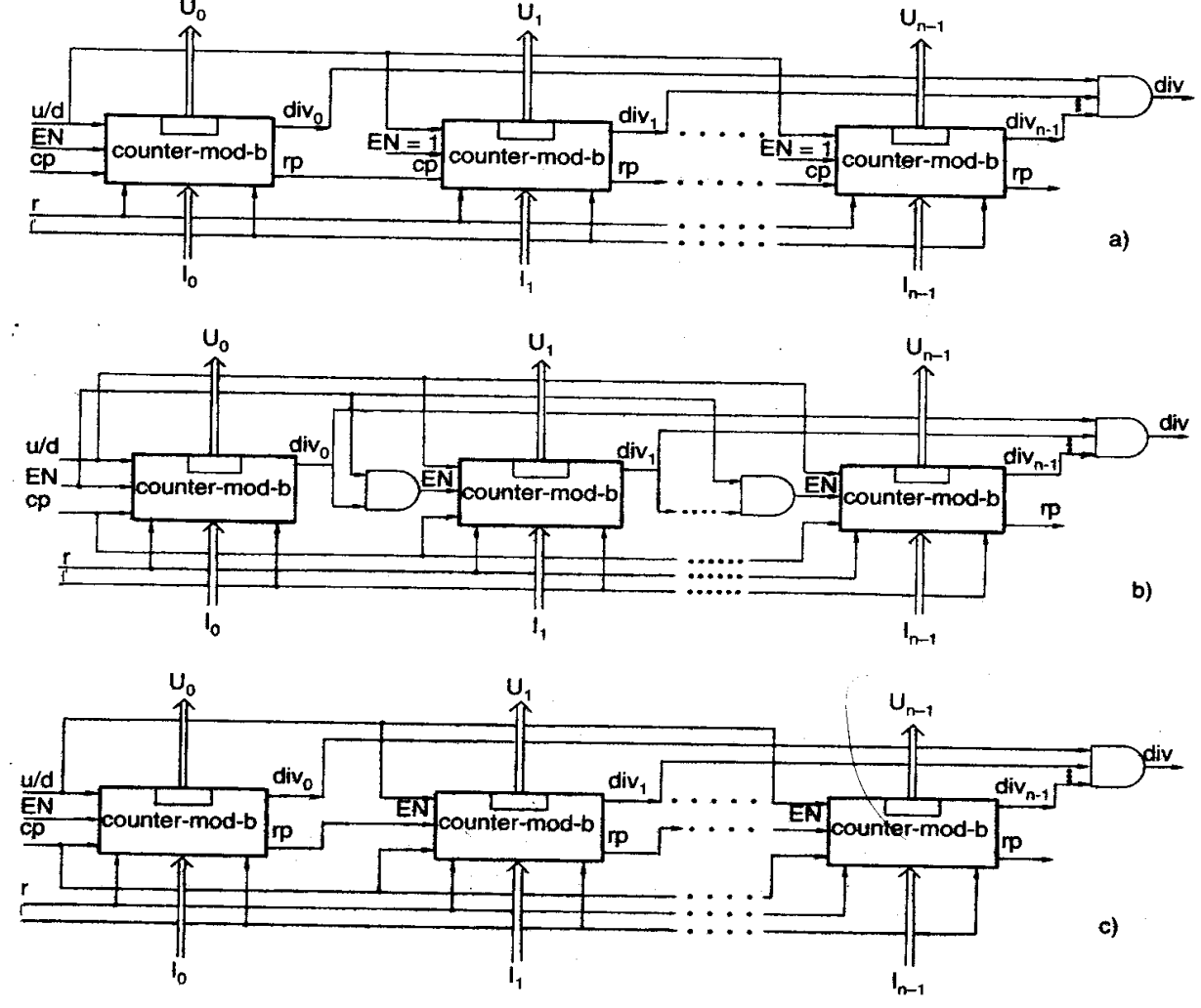
• $32.87E-02 = 0\ 01111101\ 01010000100101101011110$



=

Primo bit diverso

Contatori mod. b^n



Addizionatori in Modulo

- X, Y: Addendi; S: Somma; R: resto modulo M
- $S = |X + Y + r|_M$
- $R = [(X + Y + r) / M]$
- Se: $Z = X + Y + r$
 - $Z = MR + S$
- Quindi:
 - If $Z < M$ then $S = Z$ else $S = Z - M$
 - If $Z < M$ then $R = 0$ else $R = 1$
 - Oppure
 - $R = (Z \geq M)$
- Un addizionatore con r identicamente nullo si chiama *semiaddizionatore (half_adder_mod_M)*
 - $half_adder_mod_M(X, Y, S, R) \Leftrightarrow adder_mod_M(X, Y, S, 0, R)$
 - $R = 1$ su un addizionatore intero equivale ad un overflow

Addizionatori in Modulo

- Un addizionatore in modulo si può realizzare utilizzando sue semiaddizionatori:
 - $\text{half_adder_mod_M}(X, Y, S', R')$;
 - $\text{half_adder_mod_M}(S', r, S, R'')$
 - $R = R'$ or R''
 - Infatti:
 - $S = |S' + r|_M = ||X + Y|_M + r|_M = |X + Y + r|_M$

Addizionatori di interi positivi

- $\text{adder_mod_b}^n(X, Y, S, t, T)$
 - X, Y addendi
 - S somma
 - r riporto entrante
 - T riporto uscente
 - Rappresentazione in base b

```
For  $i=0$  to  $n-1$  do begin  
  if  $i=0$  then  $r_i=t$  else
```

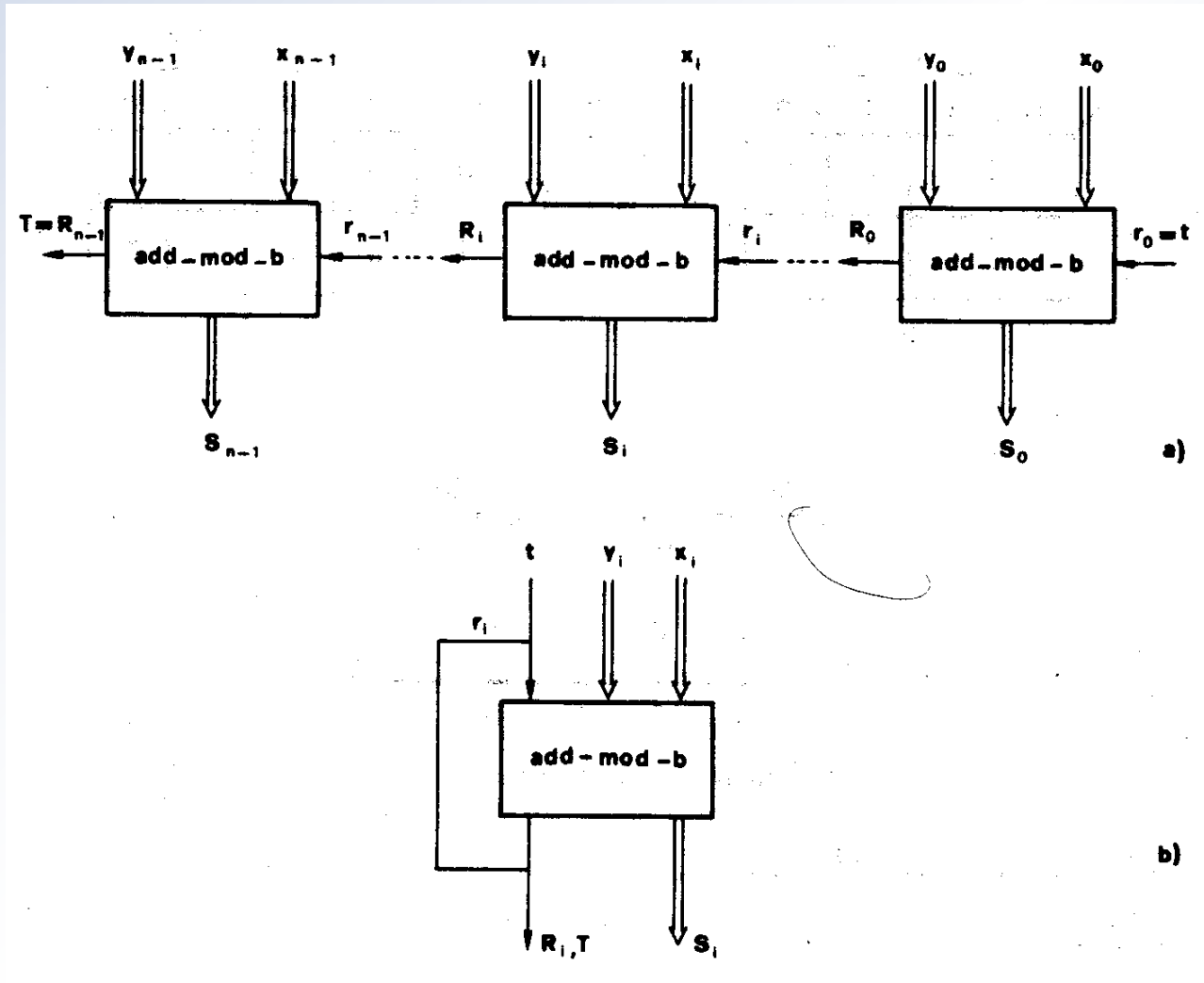
```
 $r_i=R_{i-1}$ ;
```

```
   $\text{add\_mod\_b}(X_i, Y_i, S_i, r_i, R_i)$ 
```

```
end;
```

```
 $T=R_{n-1}$ 
```

Addizionatori di interi positivi



Sottrattori in modulo

- $\text{sub_mod_M}(X, Y, S, r, R)$
 - X diminuendo
 - Y sottraendo
 - S: risultato
 - r riporto entrante (borrow)
 - R: riporto uscente (borrow)
 - $S = |X - Y - r|_M$
 - $R = |[(X - Y - r) / M]|_M$
 - $\text{half_sub_mod_M}(X, Y, S, R) \Leftrightarrow \text{sub_mod_M}(X, Y, S, 0, R)$
 - *Posto $D = X - Y - r = S - RM$*
 - *If $D \geq 0$ then $S = D$ else $S = M + D$*
 - *If $D \geq 0$ then $R = 0$ else $R = 1$*
 - *Oppure*
 - $R = (D < 0)$

Sottrattori in modulo

- Il semisottrattore $|X-Y|_M$ è in relazione con il semisottrattore $|X-Y|_M$
- Considerata la macchina $\text{half_sub_mod}(Y,X,S',R')$
- Escludendo il caso banale $X=Y$
 - $S' = |Y-X|_M = M - |X-Y|_M = M-S$
 - $R' = (Y-X < 0) = (X-Y > 0) = \text{not}(R)$
- $\text{half_sub_mod_M}(X,Y,S,R) \iff \text{half_sub_mod_M}(Y,X,S',R')$

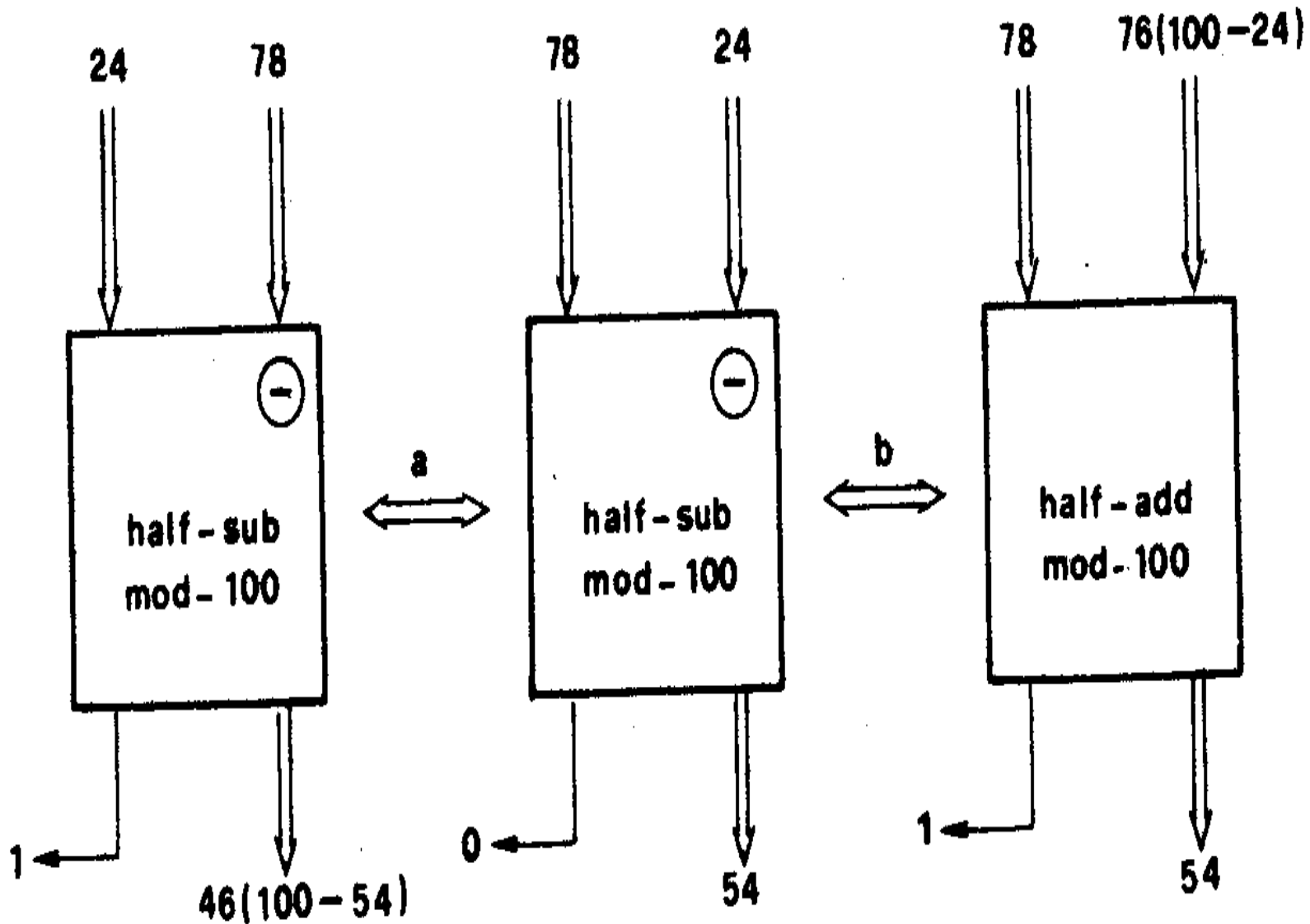
Sottrattori in modulo b^n

- Si possono costruire usando l'algoritmo della sottrazione...
MA:
- Si possono costruire anche con l'impiego di addizionatori in modulo b^n :
 - Semisottrattori:
 - Posto $Y'=M-Y$ si ha:
 - $|X+Y'|_M = |X+M-Y|_M = |X-Y|_M$
 - $(X+y' \geq M) = (X-y \geq 0)$
 - E quindi:
 - $\text{half_sub_mod_M}(X,Y,S,R) \Leftrightarrow \text{half_add_mod_M}(X,Y',S,R')$
 - $Y'=M-Y$
 - $R'=\text{not}(R)$
 - $\text{half_sub_mod_M}(X,0,S,R) \Leftrightarrow \text{half_add_mod_M}(X',y,S',R)$
 - Essendo poi: $\text{half_sub_mod_M}(X,Y,S,R) \Leftrightarrow \text{half_sub_mod_M}(Y,X,S,R)$ se $X=Y$

Sottrattori in modulo b^n

- E quindi:
 - $\text{half_sub_mod_M}(X,Y,S,R) \Leftrightarrow \text{half_add_mod_M}(X',Y,S',R)$
 - Quando $X'=M-X$, $S'=M-S$
- Riassumendo:
 - $|X-Y|_M$ si può effettuare
 - Con l'impiego del sottrattore $|Y-X|_M$ considerandone il complemento booleano del riporto uscente ed il complemento ad M della differenza
 - Con l'impiego di un addizionatore di X con il complemento ad M di Y considerandone il complemento booleano del riporto uscente
 - Con l'impiego di un addizionatore di Y con il complemento ad M di X, considerandone il complemento ad M della somma

Sottrattori in modulo b^n



Sottrattori in modulo b^n

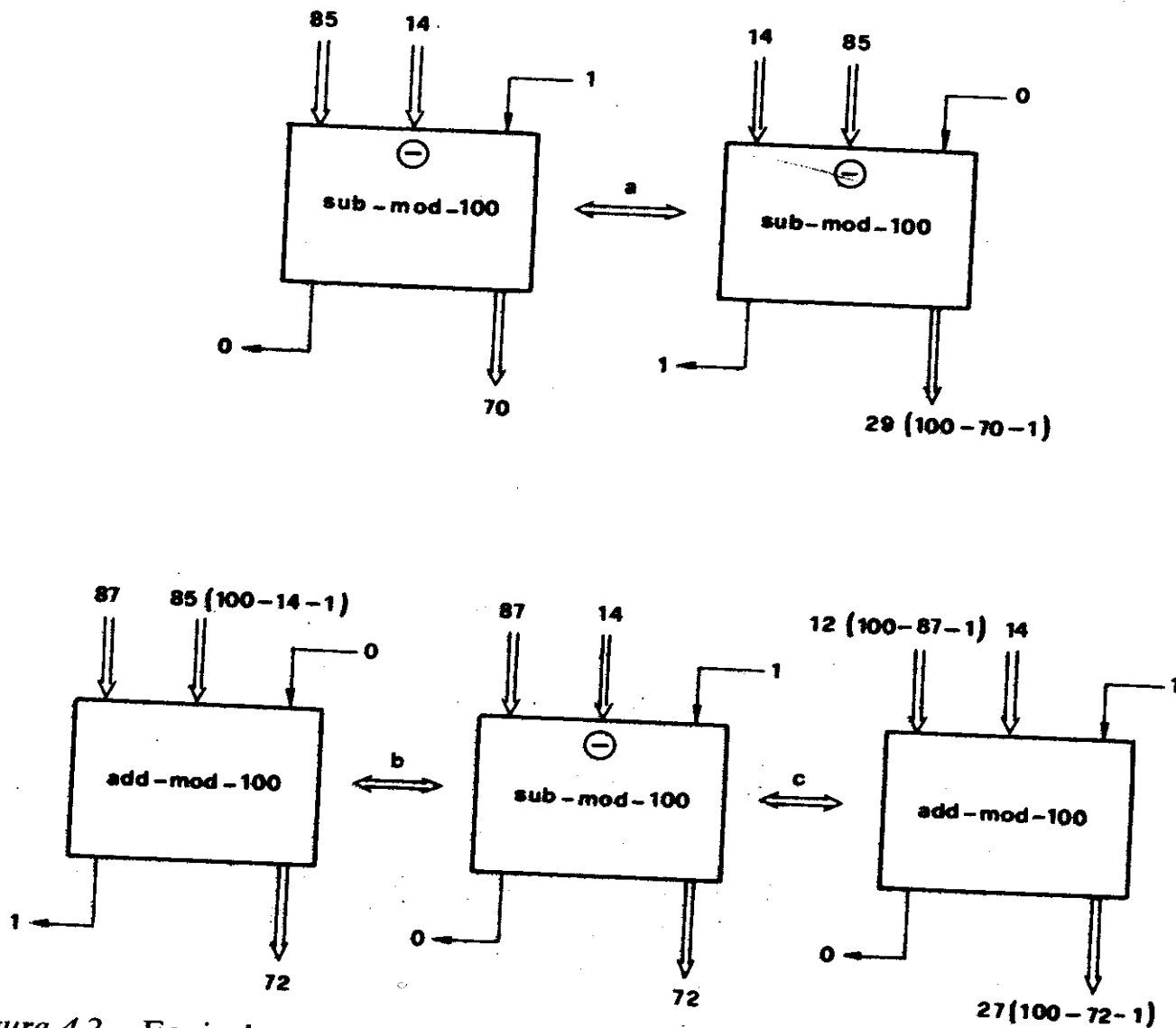


Figura 4.2 - Equivalenze tra addizione e sottrazione in modulo b^n

Addizionatori in modulo diminuito

- Trattiamo il problema più generale:
 - Realizzare un addizionatore in modulo con un altro in modulo diverso
 - Sarà possibile realizzare addizionatori in modulo $M=b^k-1$ utilizzando quelli in modulo $M'=b^k$
 - In generale, posto :
 - $M=M'-k; 0 < k < M$
 - Si vuole realizzare $\text{add_mod_}M(X, Y, S, r, R)$ tramite un $\text{add_mod_}M'(X, Y, S', r, R')$.
 - Detto $U=X+Y+r$
 - $R=(U \geq M)$;
 - $S=|U|_M$
 - $R'=(U \geq M')$
 - $S'=|U|_{M'}$

Addizionatori in modulo diminuito

- Occorre esprimere R e S in funzione di R' e S'
 - $R' = (U \geq M) = (U \geq M' - k) = (U \geq M')$ or $(M \leq U < M') = (U \geq M')$
or $(S \geq M) = R'$ or $(S' \geq M)$
 - Per S:
 - Per $R=0$ essendo U inferiore al minore dei due moduli M e M'
 - $R=0 \Rightarrow |U|_M = |U|_{M'} =; S=S'$
 - Per $R=1$ essendo $M \leq U < 2M; M' \leq U+k < 2M'$
 - $R=1 \Rightarrow |U|_M = U - M = U+k - M' = |U+k|_{M'} = |S'+k|_{M'}$
 - Quindi
 - $R = R'$ or $(S' \geq M)$
 - $S =$
 - S' per $R=0$
 - $|S'+k|_{M'}$ per $R=1$

Addizionatori in modulo diminuito

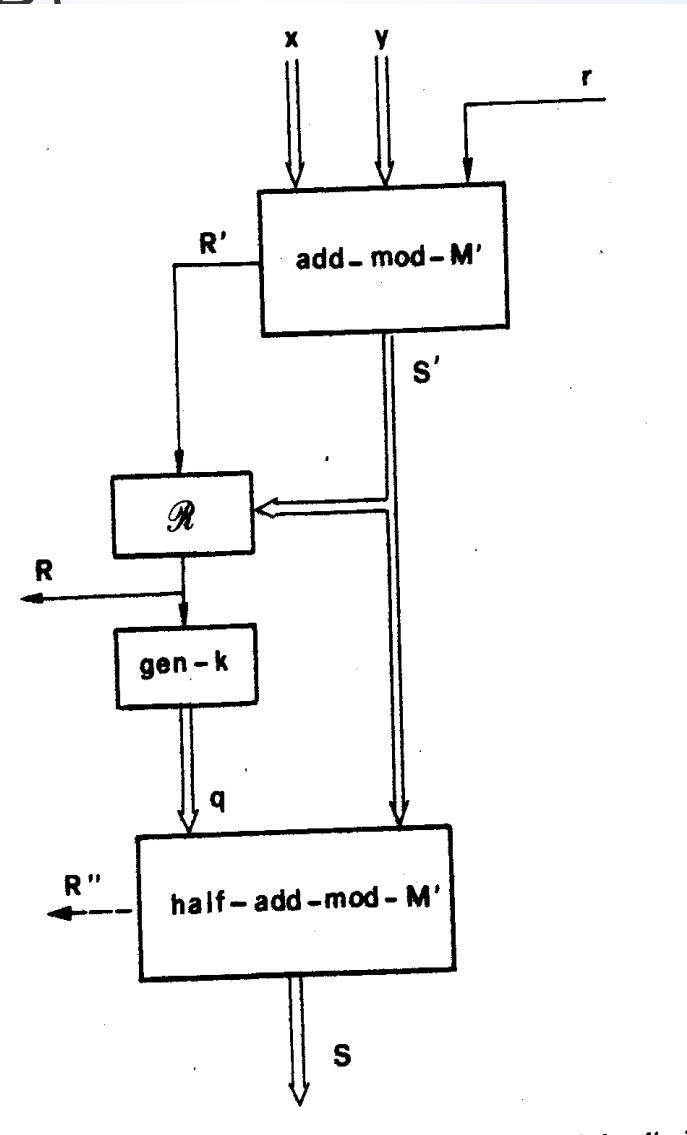
- In definitiva: $\text{add_mod_M}(X, Y, S, r, D)$.

$\text{add_mod_M}'(X, Y, S', r, R')$

$R = R'$ or $S' \geq M' - k$

If R then $q = k$ else $q = 0$

$\text{half_add_mod_M}'(S', 1, S, R'')$



Addizionatori di numeri relativi

- `add_rel_M(x,y,z,overflow)`
 - $\text{Overflow} = x+y < M1 \text{ or } x+y > M2$
 - If not overflow then $z=x+y$
- Questo in generale ma
 - Il tutto dipende dalla rappresentazione dei numeri

Addizionatori di numeri relativi

- Rappresentazione in segno e modulo

- `add_rel_M(X,Y,Z,overflow)`

- X,Y,Z rappresentazione abs

```
if (X.segno = Y.segno) then begin
    half_add_mod_M(X.abs,Y.abs,Z.abs,overflow)
    Z.segno=X.segno
end else begin
    overflow=0
    if X.abs >= Y.abs then begin
        half_sub_mod_M(X.abs,Y.abs,Z.abs,R);
        Z.segno=X.segno
    end else begin
        half_sub_mod_M(Y.abs,X.abs,Z.abs,R)
        Z.segno=Y.segno
    end
end
end
```

Addizionatori di numeri relativi

- Rappresentazione in complementi

- Dato M:

- $M1 = -M/2$ e $M2 = M/2 - 1$ per M pari

- $|M1| = |M2| = (M-1)/2$ per M dispari

- Si ha:

- $\text{arrot}(z) = |x+y|_M = ||x|_M + |y|_M|_M = |X+Y|_M$

- Quindi l'uscita Z coincide con quella di un semiaddizionatore

- `add_rel_M/2 (X,Y,Z,overflow):`

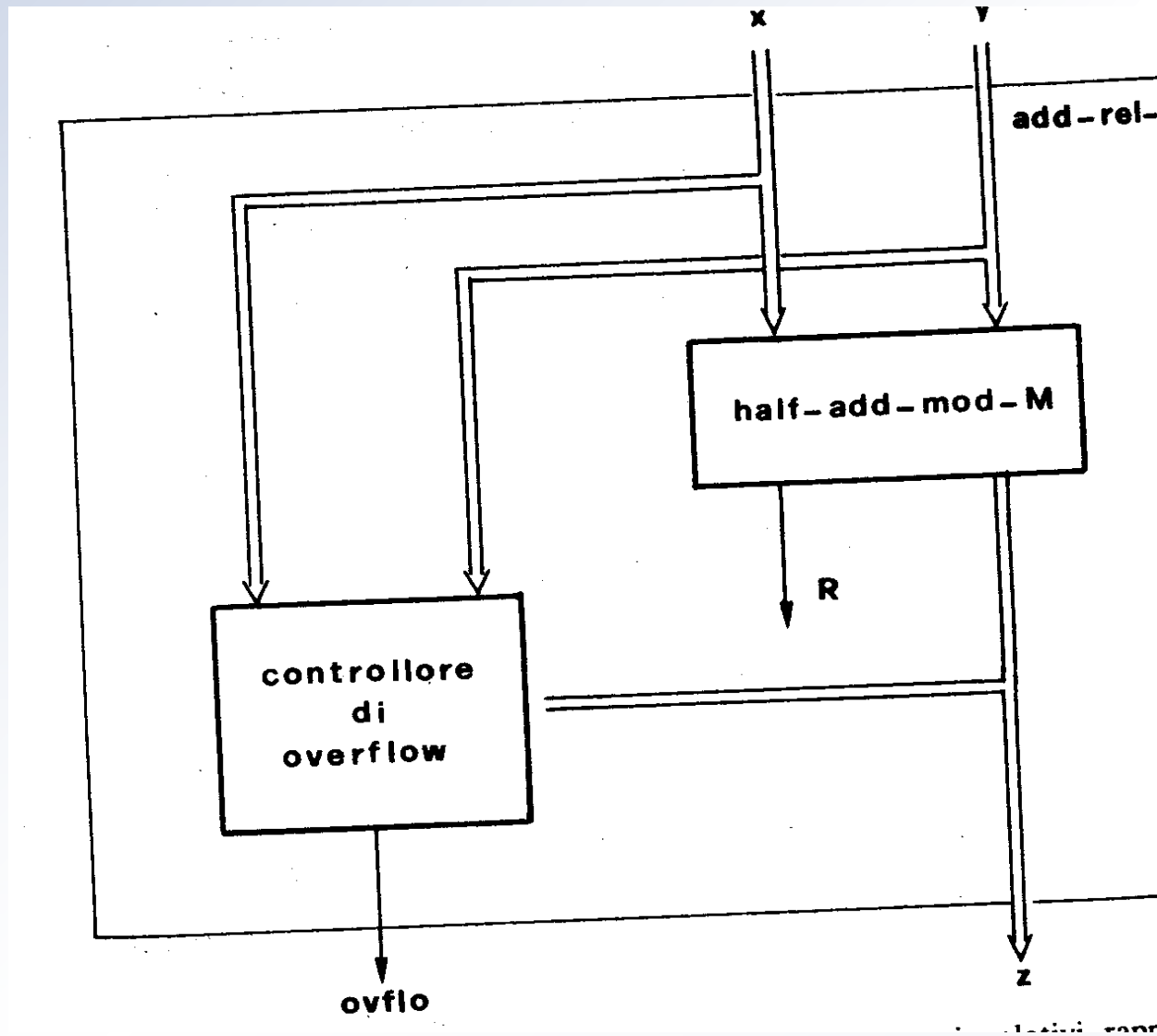
`half_add_mod_M(X,Y,Z,R)`

`overflow = (X < M/2) and (Y < M/2)`

`and (Z >= M/2) or (X >= M/2) and`

`(Y >= M/2) and (Z < M/2)`

Addizionatori di numeri relativi



Addizionatori di numeri relativi

- Rappresentazione in complementi alla base
 - $M=b^n$
 - $\text{ovflow} = (X_{n-1} < b/2) \text{ and } (Y_{n-1} < b/2) \text{ and } (Z_{n-1} \geq b/2) \text{ or } (X_{n-1} \geq b/2) \text{ and } (Y_{n-1} \geq b/2) \text{ and } (Z_{n-1} < b/2)$
 - Per $b=2$, $X \geq b/2$ diventa
 - $X_{n-1} = 1$
 - Quindi
 - $\text{Ovflow} = (\text{eq}(X_{n-1}, Y_{n-1}) \text{ and } \text{xor}(Z_{n-1}, X_{n-1}))$
 - Si ha overflow se, essendo x ed y concordi, si ottiene una rappresentazione di z di segno opposto.

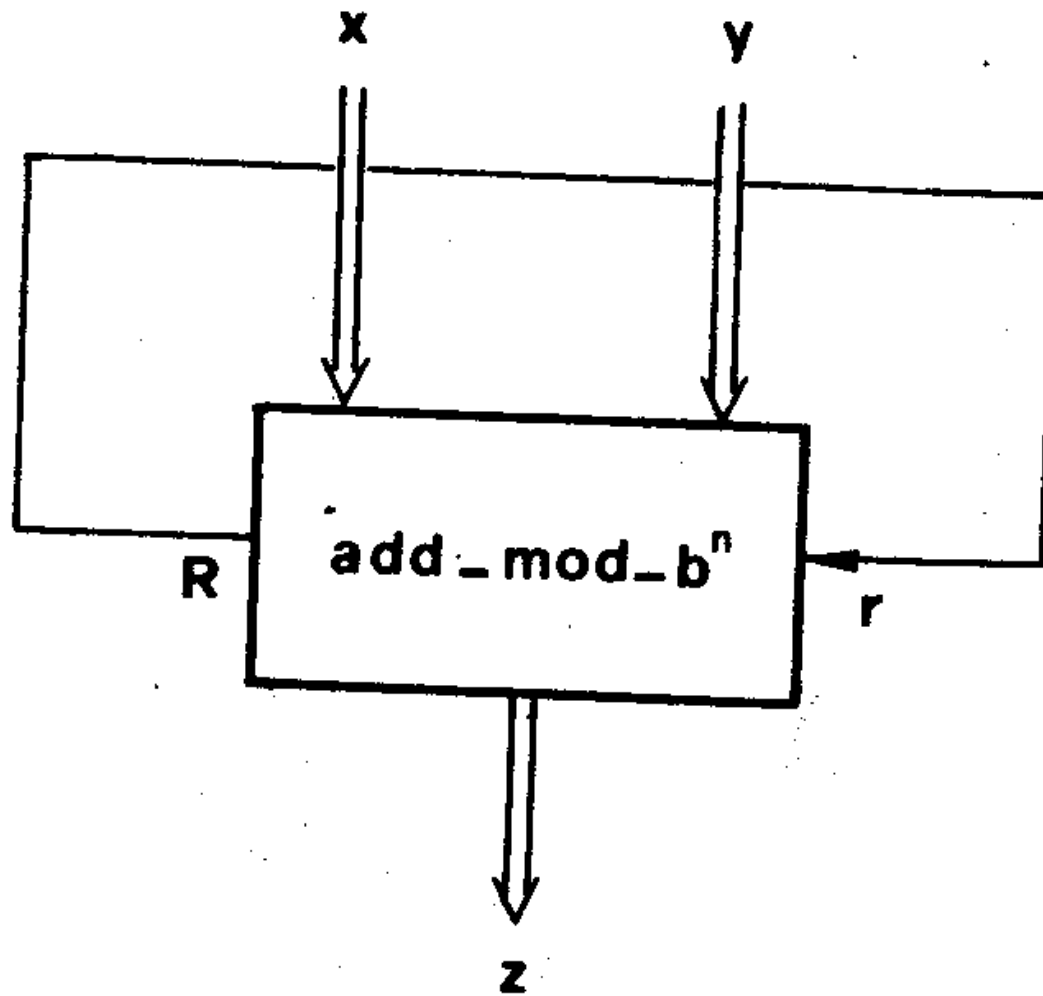
Addizionatori di numeri relativi

- Rappresentazione in complementi diminuiti
 - $M=b^n-1$
 - Si può usare un semiaddizionatore modulo b^n-1 implementato tramite un addizionatore modulo b^n
 - $R= R'$ or $(S'=b^n-1)$
 - $S=$
 - S' per $R=0$
 - $|S' + 1|_b$ per $R=1$
 - La correzione si effettuerebbe, oltre che per $R'=1$ nel solo caso $S'=b^n-1$ producendo il valore $S= |b^n-1+1|_b = 0$
 - Se per $S'=b^n-1$ non si effettuasse alcuna correzione si avrebbe
 - $S= b^n-1$ poichè 0 e b^n-1 sono due rappresentazioni di $s=0$

Addizionatori di numeri relativi

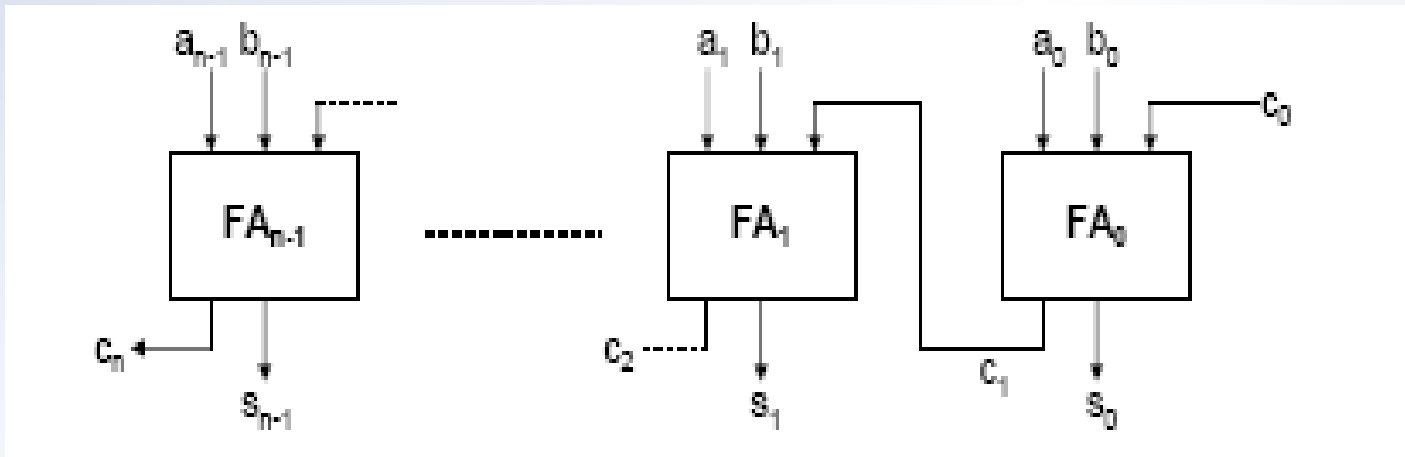
- Rappresentazione in complementi diminuiti
 - Quindi si può evitare la correzione $S'=b^n-1$ e porre:
 - $R=R'$
 - $S=$
 - S' se $R'=0$
 - $|S'+1|_b$ se $R'=1$
 - Quindi
 - $\text{half_add_mod_}b^n(X,Y,S',R')$;
 - $\text{half_add_mod_}b^n(S',R',Z,R'')$; [R'' ignorato]
 - I due semiaddizionatori possono essere fusi in un unico adder sfruttandone l'ingresso riporto per inserirvi direttamente la correzione.
 - Ne deriva uno schema in cui sono direttamente collegati il riporto entrante ed il riporto uscente
 - (end-around-carry)

Addizionatori di numeri relativi



Riassunto

- Addizione Senza Segno (ripple carry)



Addizionatori Veloci

- Somma e riporto allo stato i :
 - $s_i = x_i'y_i'c_i + x_i'y_i'c_i' + x_iy_i'c_i' + x_iy_i c_i$
 - $c_{i+1} = x_iy_i + x_i c_i + y_i c_i$
- *Fattorizzando la seconda espressione si ha:*
 - $c_{i+1} = G_i + P_i c_i$ con $G_i = x_iy_i$ e $P_i = x_i + y_i$
- *Le funzioni G_i e P_i Sono dette funzioni di generazione e propagazione*
 - *Possono essere calcolate in parallelo.*
- *Quando la funzione di generazione dello stadio i (cioè G_i) è uguale a 1, allora il riporto c_{i+1} è uguale a 1 indipendentemente dal riporto in ingresso c_i*

Addizionatori Veloci

- L'espressione per c_i è:
 - $c_i = G_{i-1} + P_{i-1}c_{i-1}$
- Sostituendo nell'espressione di c_{i+1} si ha:
 - $c_{i+1} = G_i + P_i(G_{i-1} + P_{i-1}c_{i-1}) = G_i + P_iG_{i-1} + P_iP_{i-1}c_{i-1}$
- Continuando con l'espansione fino a c_0 si ottiene:
 - $c_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-1} + \dots + P_iP_{i-1}\dots P_1G_0 + P_iP_{i-1}\dots P_1c_0$
- Il riporto può quindi essere ottenuto
 - Mediante una forma a due livelli attraversando
 - sue sole porte logiche

Addizionatori Veloci

- Il ritardo totale per ottenere tutte le somme ed il riporto più a sinistra c_{i+1} è dato dalla somma di:
 - Un ritardo di porta per il calcolo delle funzioni di generazione e di propagazione
 - 2 ritardi di porta logica per calcolare il riporto i -esimo
 - 1 ritardo di porta logica per negare il valore del riporto
 - 2 ritardi di porta logica per calcolare la somma i -esima
- Totale:
 - 6 ritardi di porta logica
- Il ritardo non dipende dalla lunghezza degli operandi
 - Problema: Realizzazione circuitale per operandi lunghi (ad esempio 32 bit) fa uso di porte con un *fan-in* molto elevato

Addizionatore Veloce

- Divisione del sommatore completo in blocchi
- Un sommatore a 4 bit, ad esempio, richiede porte a
- 5 ingressi, come mostra l'equazione seguente:
 - $c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$
- Il riporto c_4 è generato con ritardo di 2 porte
- Collegando in cascata 8 di tali sommatore si ottiene un sommatore a 32 bit con ritardo pari a:
 - 1 ritardo di porta logica per le funzioni G_i e P_i
 - 2 ritardi di porta logica per i c_4, c_8, \dots, c_{28} e c_{31}
 - 3 ritardi per la somma s_{31}
 - Totale: $1 + 2 \cdot 8 + 3 = 20$ ritardi di porta logica

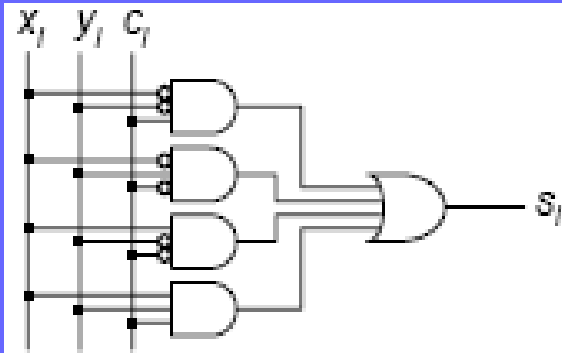
Addizione Veloce

- I sommatore che sfruttano il meccanismo della generazione dei riporti in anticipo sono detti *Carry-Look-Ahead Adders* o *CLA*
- Il meccanismo presentato può essere applicato ricorsivamente ai blocchi di 4 bit dell'esempio precedente
- In questo modo si ottengono sommatore ancora più veloci pur mantenendo limitato il *fan-in* delle porte in ingresso

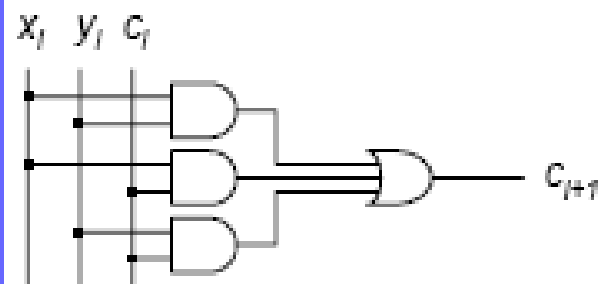
Alcune implementazioni

- Full Adder

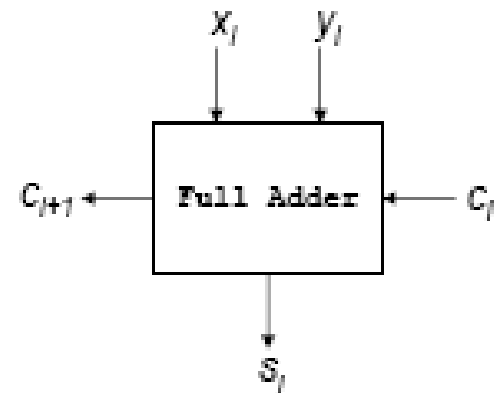
$$S_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$$



$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$



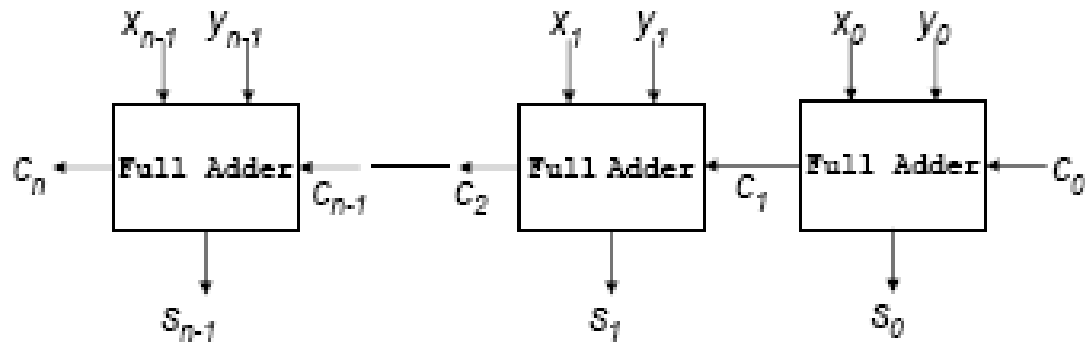
Full Adder



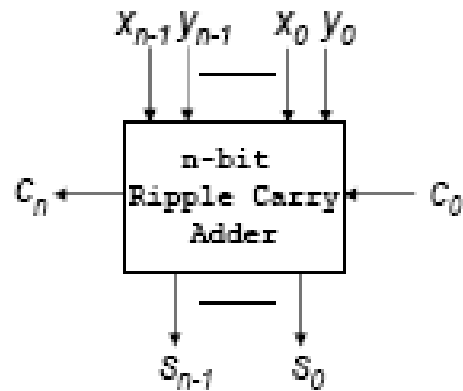
Alcune Implementazioni

- Ripple Carry

Ripple-Carry Architecture



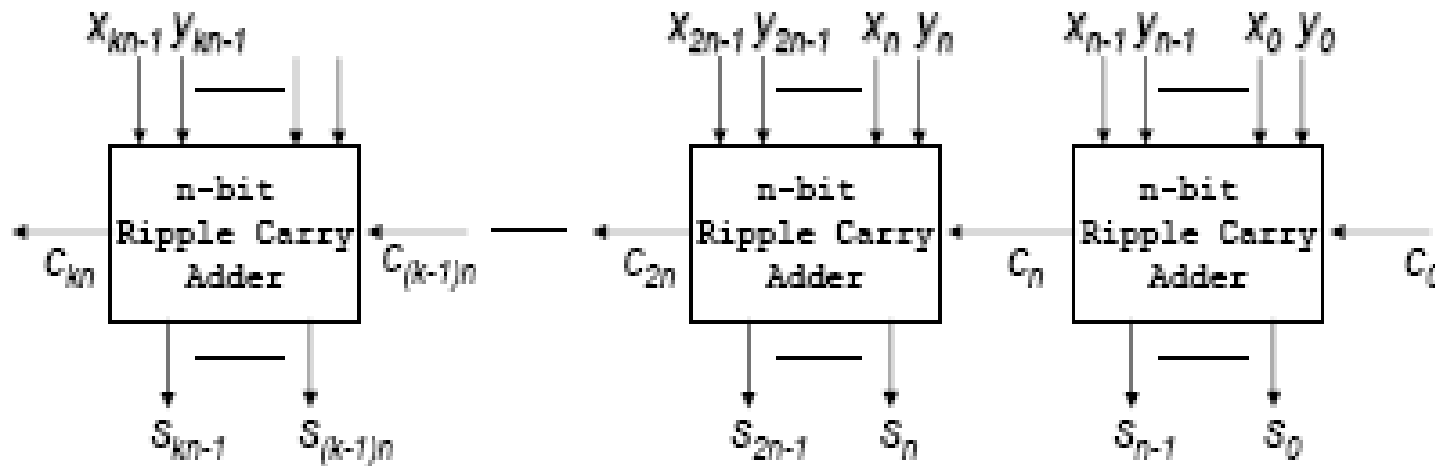
Ripple-Carry Adder



Alcune implementazioni

- Ripple Carry

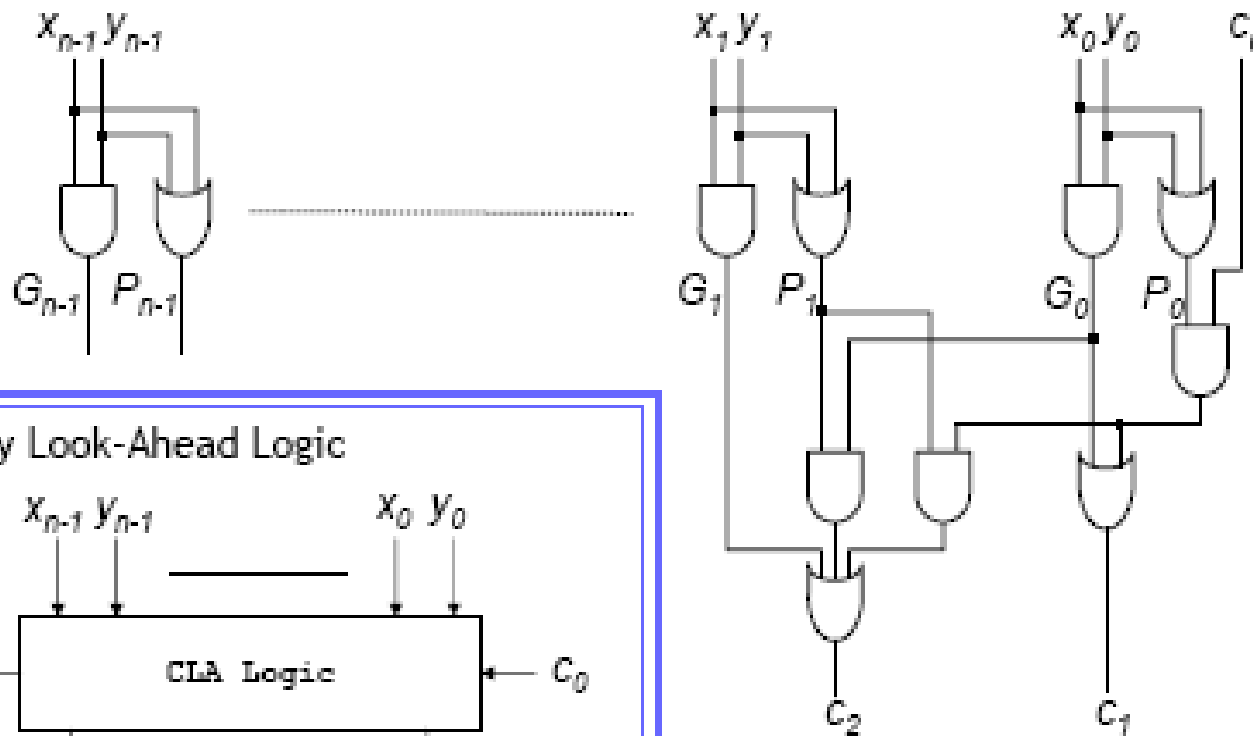
Ripple-Carry Block Architecture



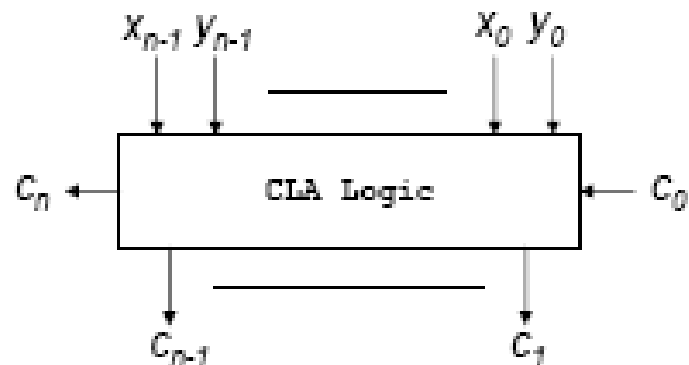
Alcune implementazioni

- Carry Look Ahead

Carry Look-Ahead Logic: Internal architecture

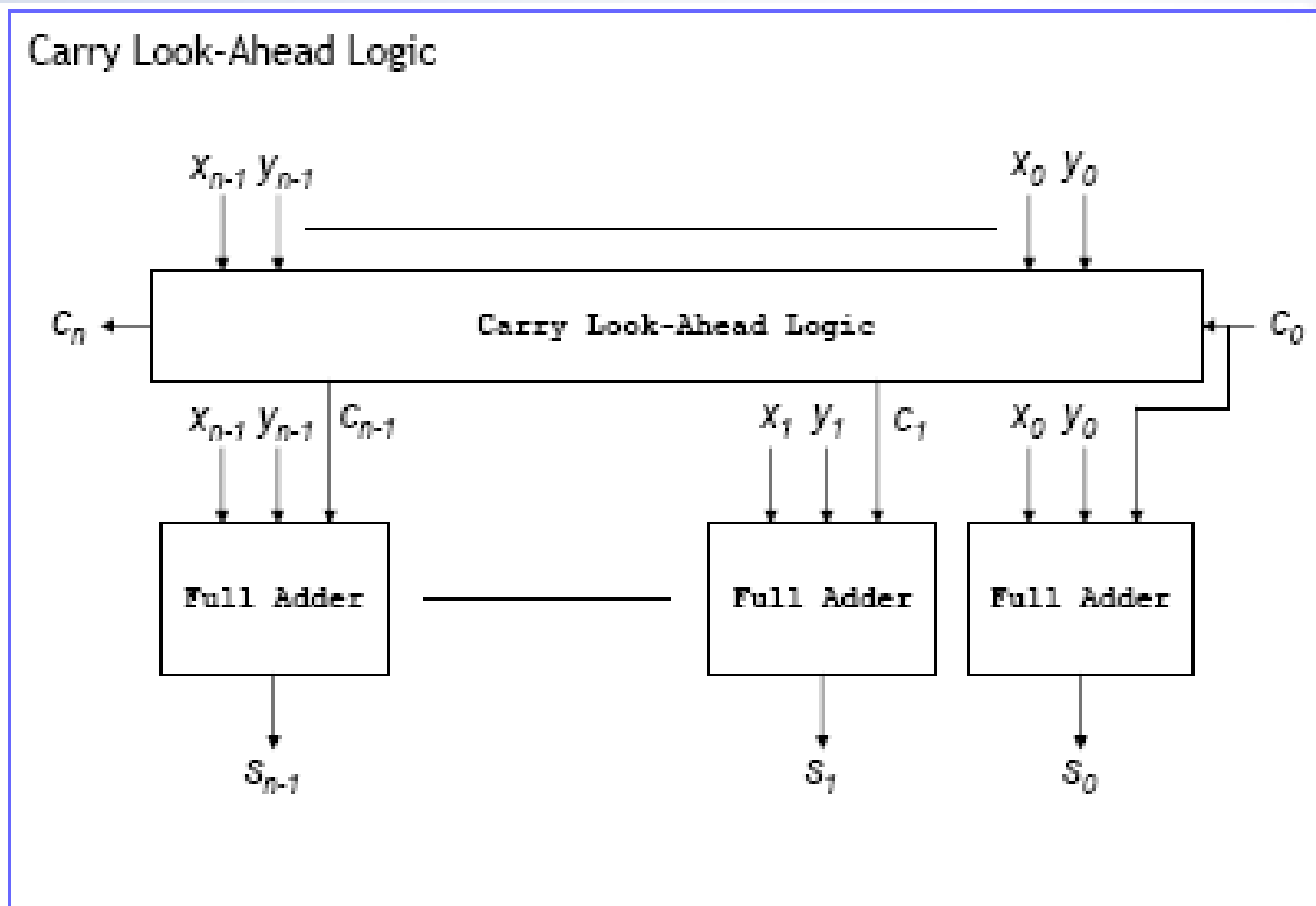


Carry Look-Ahead Logic



Alcune implementazioni

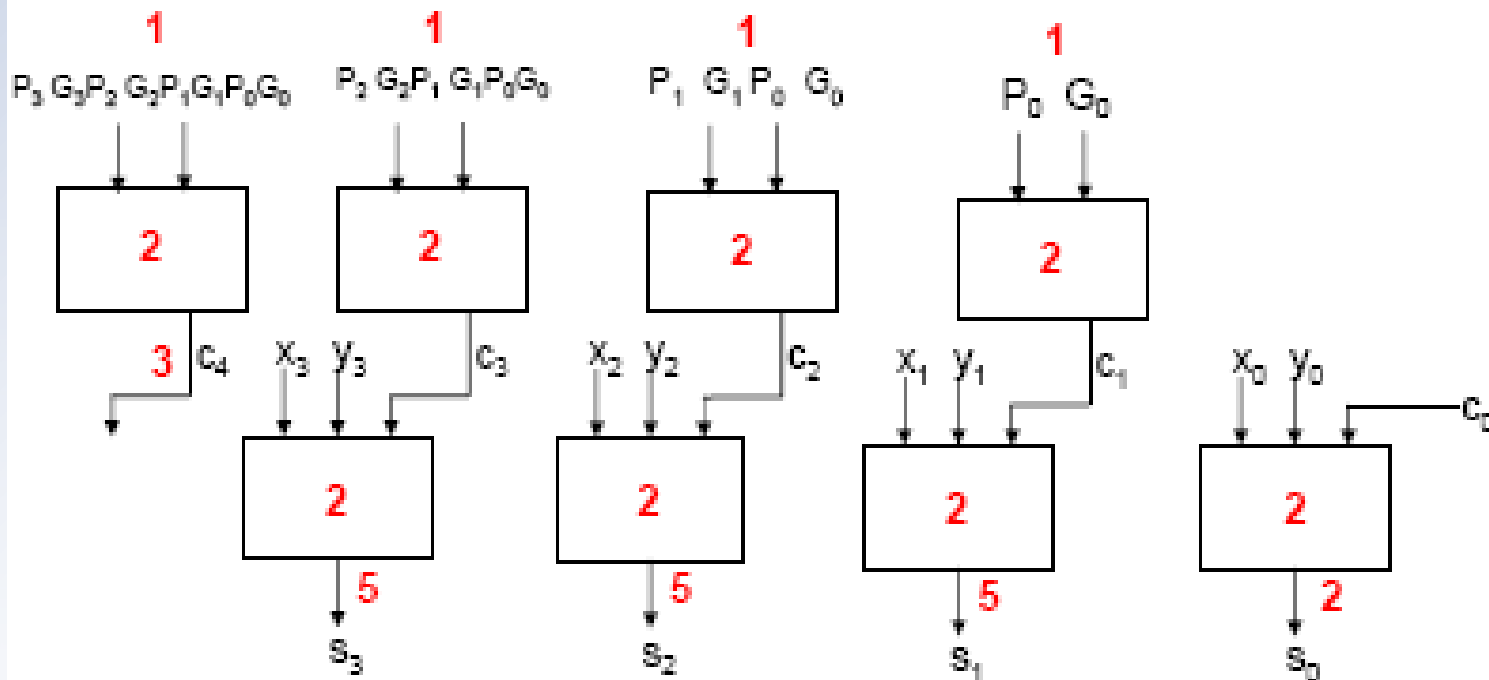
- Carry Look Ahead



CLA: Calcolo dei riporti in Parallelo

- L'espressione per il riporto c_{i+1} può essere calcolata in modo **iterativo**. Infatti $c_i = G_{i-1} + P_{i-1}c_{i-1}$
- Sostituendo nell'espressione di c_{i+1} si ha:
 - $c_{i+1} = G_i + P_i(G_{i-1} + P_{i-1}c_{i-1}) = G_i + P_iG_{i-1} + P_iP_{i-1}c_{i-1}$
- Continuando con l'**espansione** fino a c_0 si ottiene:
 - $c_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-1} +$
 - $+ \dots +$
 - $+ P_iP_{i-1}\dots P_1G_0 +$
 - $+ P_iP_{i-1}\dots P_1c_0$
- Le espressioni ottenute sono forme a due livelli (ritardo: 2 porte)
- Il riporto in uscita di ogni singolo stadio può essere calcolato in parallelo tramite:
 - le i funzioni di generazione G_i e le i funzioni di propagazione
 - P_i
 - il riporto in ingresso allo stadio 0, c_0
- I sommatore che sfruttano il meccanismo della generazione dei riporti in anticipo sono detti **Carry-Look-Ahead Adders** o **CLA**

CLA: Calcolo dei riporti in parallelo



- $c_1 = G_0 + P_0 c_0$
- $c_2 = G_1 + P_1 G_0 + P_1 P_0 c_0$
- $c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$
- $c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$

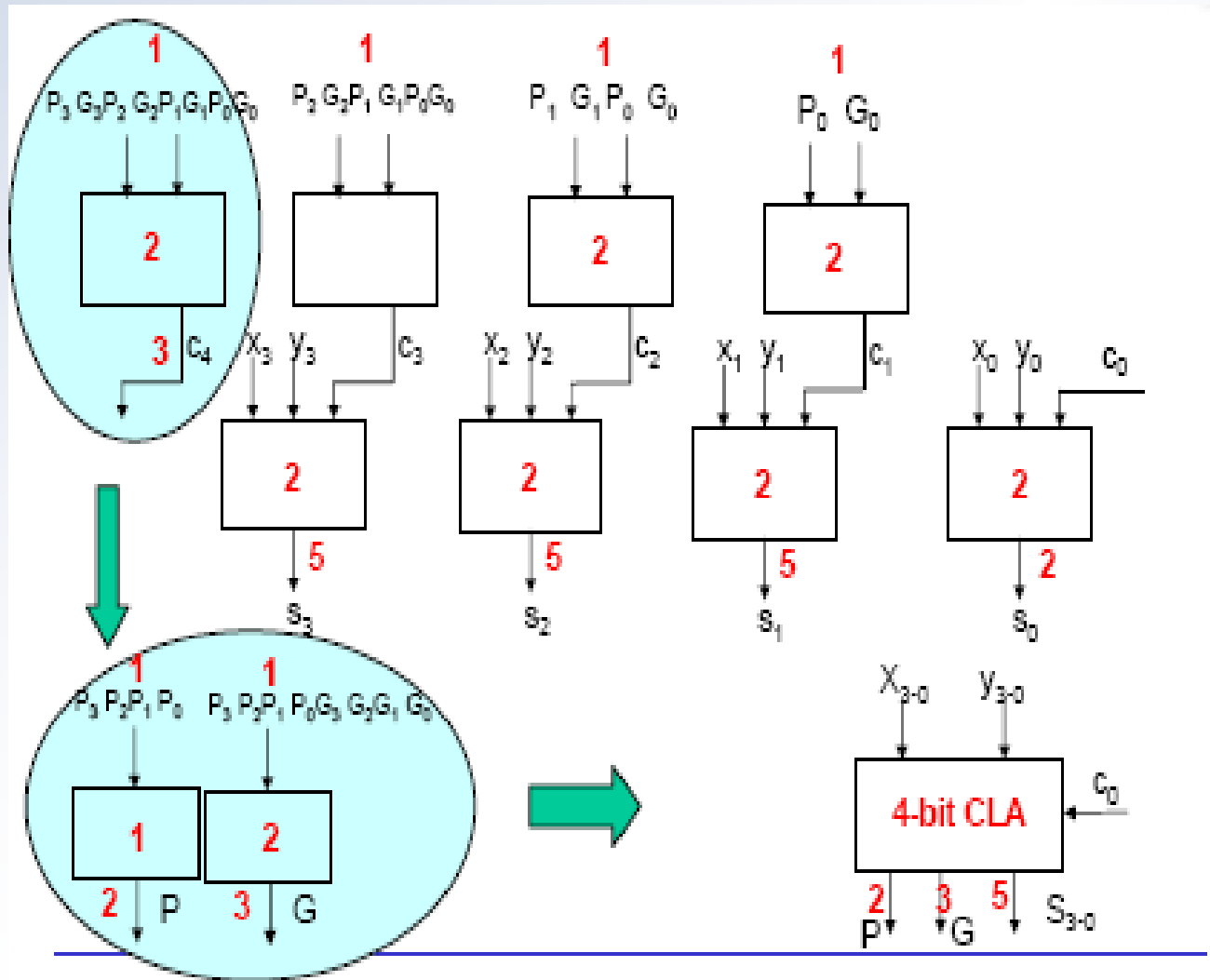
CLA: Valutazione delle Prestazioni

- Il ritardo totale per ottenere tutte le somme ed il riporto più a sinistra c_{i+1} è dato dalla somma di:
 - Un ritardo di porta per il calcolo delle funzioni di generazione e di propagazione ($G_i = x_i y_i$ e $P_i = x_i + y_i$)
 - Due ritardi di porta logica per calcolare il riporto i -esimo
 - Due ritardi di porta logica per calcolare la somma i -esima
 - Totale: 5 ritardi di porta logica
- Il ritardo è indipendente dalla lunghezza degli operandi
- Problema:
 - Realizzazione circuitale per operandi lunghi (ad esempio 32 bit) fa uso di porte con un *fan-in* molto elevato: non praticabile!!
- Soluzione: **addizionatore veloce a blocchi**

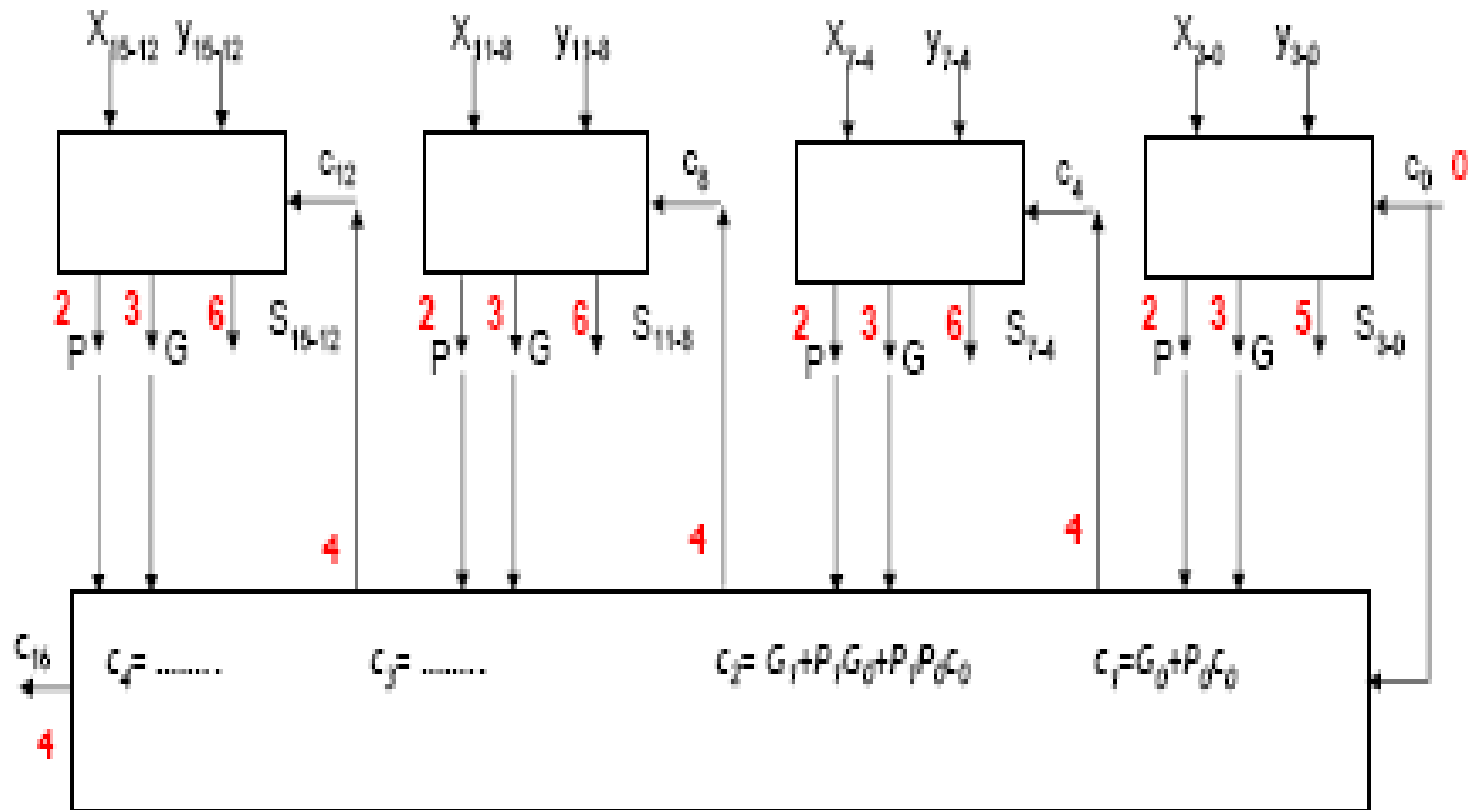
CLA: Schema a Blocchi

- Il sommatore completo a n bit è ottenuto utilizzando un insieme di blocchi costituiti da CLA a m bit
- Il blocco è costituito da un sommatore CLA a 4 bit (ragionevole).
- Il riporto finale di questo sommatore ha la seguente espressione:
 - $c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$
- che può essere riscritta come
 - $c_{uscita} = G + P C_0$
- con il tempo di ritardo per il calcolo di P e G:
 - P = attraversamento di 2 porte logiche (1 per calcolare P_3, P_2, P_1 e P_0 , 1 per calcolare il prodotto)
 - G = attraversamento di 3 porte logiche

Esempio: CLA a 4 Bit



CLA a 16 bit con CLA a 4 bit



Moltiplicatori Modulo M

- $\text{molt_mod_M}(X, Y, P, Q)$
 - $P = |X * Y|_M$
 - $Q = [X * Y / M]$
- In aritmetica estesa:
 - $Z = X * Y$: se $X, Y \in [0, M) \Rightarrow Z \in [0, M^2)$
 - $Z = |Z|_M + [Z / M] * m = P + QM$
- Quindi
 - $\text{molt_est_M}(X, Y, Z) \Leftrightarrow \text{molt_mod_M}(X, Y, P, Q)$
 - $Q \neq 0$ indica overflow

Moltiplicatore di Interi

- `molt_int_M(X,Y,Z, overflow)`
 - `molt_mod_M(X,Y,P,Q)`
 - `Z=P`
 - `overflow=(Q!=0)`

Moltiplicazione frazionari

- Essendo
 - $x' = X/M$; $y' = Y/M$; $z' = XY/M^2$; x,y,z in $[0,1)$
- Non si ha mai overflow e si può assumere Z come rappresentativo di z' (letto con l'apposito fattore di scala
 - $z' = Z/M^2 = Q/M + P/M^2$;
- Si può assumere Q come valore approssimato e P come indicativo dell'errore
 - $z'=Q/M$; $e'=P/M^2$
- `molt_fraz_M(X,Y,Z)`
 - `molt_mod_M(X,Y,P,Q)`
 - $Z=Q$

Moltiplicatori interi in complementi

- Essendo
 - $\text{appr}(z) = |x^*y|_M = ||x|_M * |y|_M|_M = |X^*Y|_M = P$
- P rappresenta ancora il prodotto in assenza di overflow
- Q non diventa più rappresentativa di overflow
- Si può far riferimento ai resti modulo M^2 .
 - $X' = |x|_M^2$; $Y' = |y|_M^2$
- La macchina
 - $\text{Molt_mod_M}^2(X', Y', P', Q')$
- Fornisce il prodotto
 - $\text{appr}(z) = |x^*y|_M^2 = P' + Q' * M^2$

Moltiplicatori modulo b^n

- Si esegue l'algoritmo classico della moltiplicazione
 - $\text{molt_est_}b^n(X,Y,Z)$
 - $Z=0$
 - For $i=0$ to $n-1$ do $Z=Z+X*Y_i*b^i$
- Il prodotto di un numero per una cifra si ottiene nel modo classico ($C=X*\text{cifra}$)
 - C è quindi un numero in genere a $n+1$ cifre
 - --- Se $g=0$ $C=0$ ---
- Detti:
 - R la somma fra il prodotto C e la cifra i.ma del vecchio Q
 - $C=X*Y_i$
 - $R=Q+C$
- Il nuovo Q si ottiene STACCANDO da R la cifra meno significativa
 - $Q=[R/b]$
- Il nuovo P si ottiene AGGIUNGENDO in testa al precedente la cifra troncata di R
 - $P=P+|R|_b*b^i$

Moltiplicatori modulo b^n

- $\text{molt_mod_}b^n(X, Y, P, Q)$
 - $Q=0; P=0$
 - For $i=0$ to $n-1$ do
 - $\text{molt_cifra}(X, Y_i, C)$
 - $\text{add_int_}b^{n+1}(Q, C, R, \text{overflow})$
 - $P_i = R_0$ (equivale a $P = P + |R|_b * b^i$)
 - $Q = [R/b]$

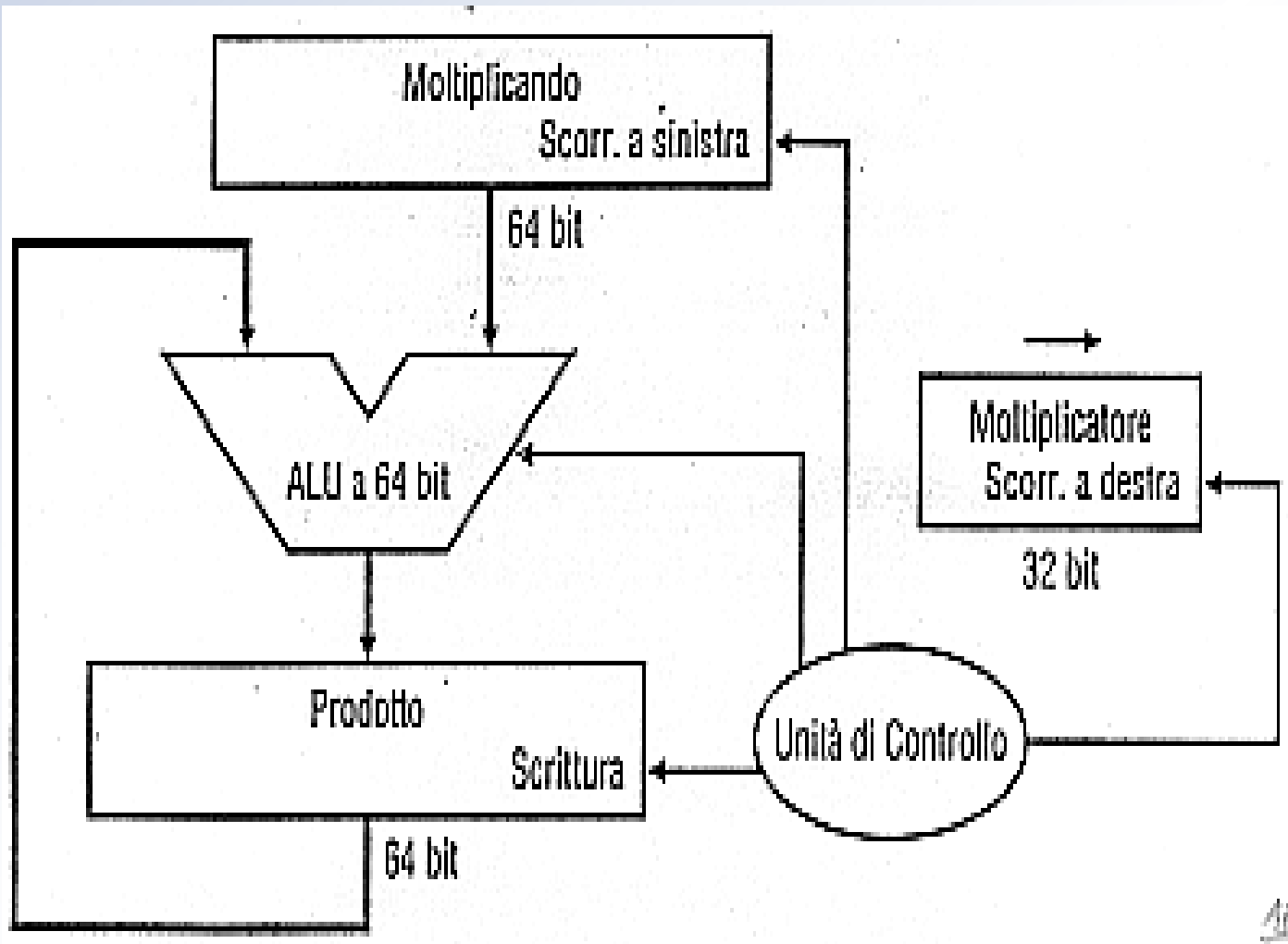
Moltiplicatore modulo b^n SERIALE

- `serial_molt_mod_bn(X,Y,Z)`
 - `Z.Q=0; Z.P=Y`
 - For `i=0` to `n-1` do
 - `molt_cifra(X,Z0,C)`
 - `add_int_bn+1(Z.Q,C,R,overflow)`
 - `{a:} Z.Q=R`
 - `{s:} shr(Z,0)`
- Dove `a:` e `s:` sono due segnali di tempificazione

Moltiplicatore binario

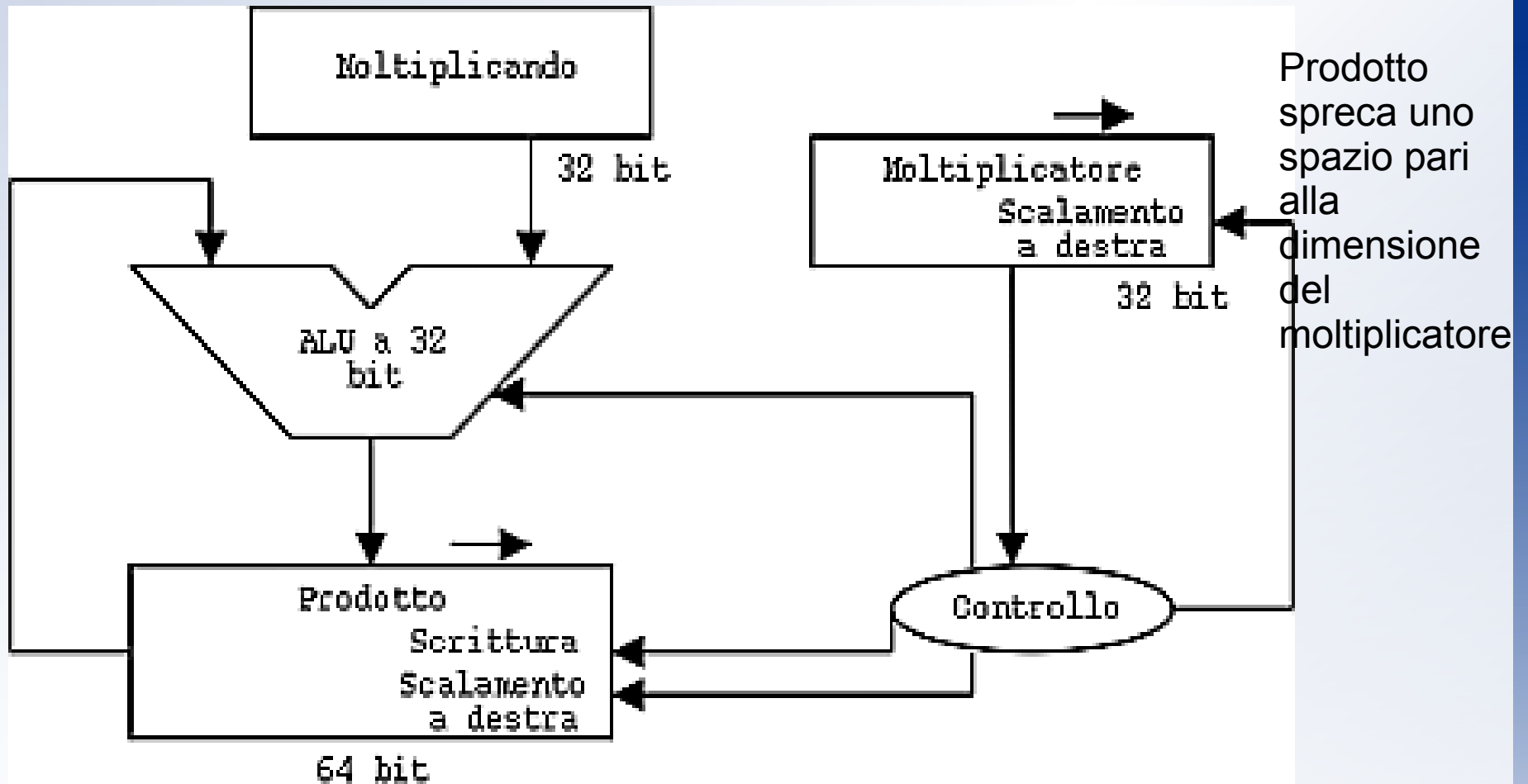
- Moltiplicazione per raddoppio e dimezzamento
 - $X*Y = (2*X)+(Y/2)$ se Y è pari
 - $X*Y = (2*X)*(Y-1)/2 +X$ se Y è dispari
- $Z=0$
- While $Y \geq 1$ do
 - If disparty (y) then $Z=Z+X$
 - $Y=[Y/2]$
 - $X=2*X$

Moltiplicatori: Implementazioni

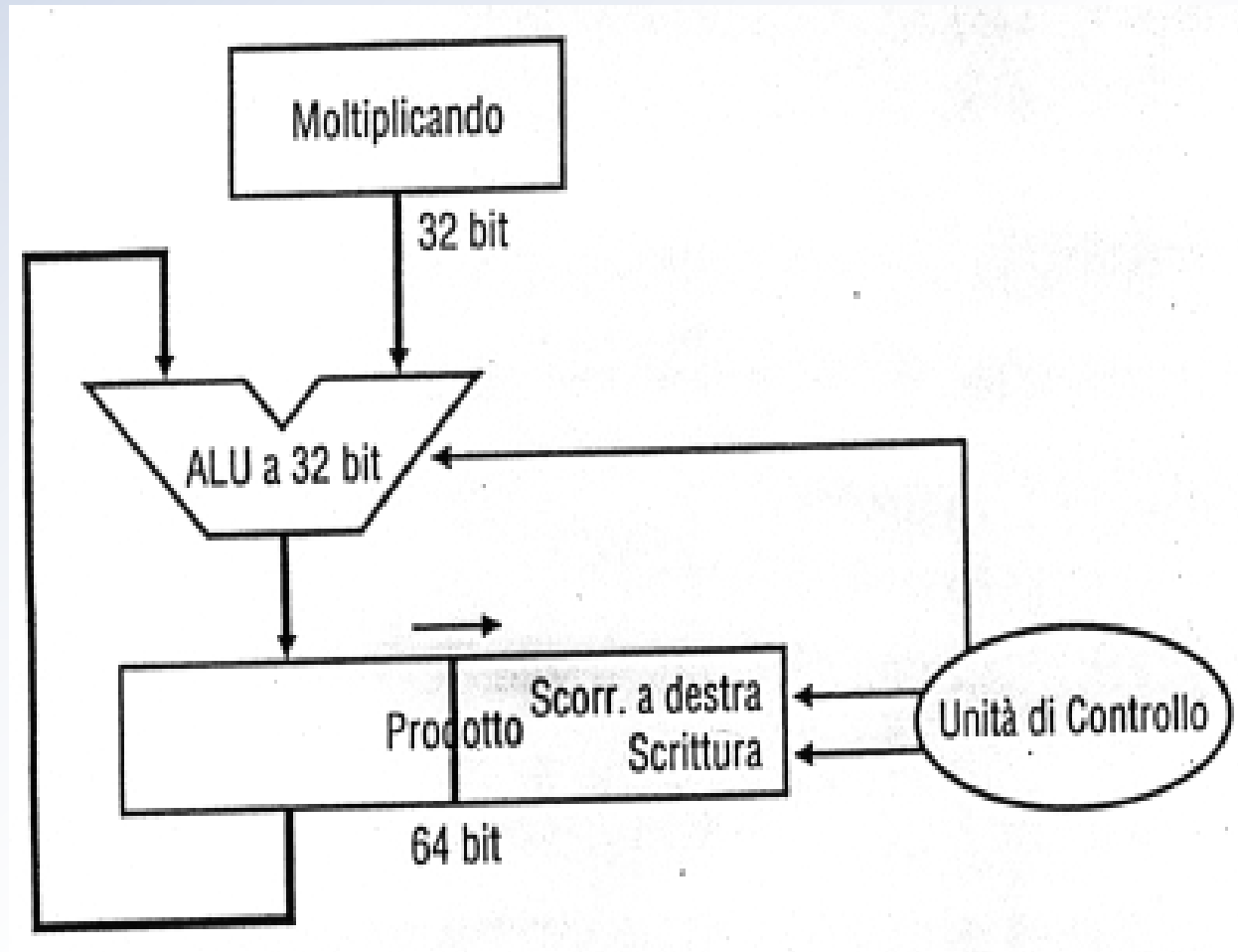


Metà del contenuto del moltiplicand o contiene 0

Moltiplicatori: Implementazioni



Moltiplicatori: Implementazioni



Moltiplicatori Veloci: Algoritmo di Booth

- $0011111111111000 \Leftrightarrow 010000000000-1000$
- Se ci sono molti 1... non si fanno parecchie moltiplicazioni
- In media funziona come gli altri ;)

Divisori (in breve ;)

Divide: Paper & Pencil

```

      1001  Quotient
Divisor 1000 | 1001010  Dividend
      -1000
      -----
         10
          101
           1010
            -1000
             ---
              10  Remainder (or Modulo result)

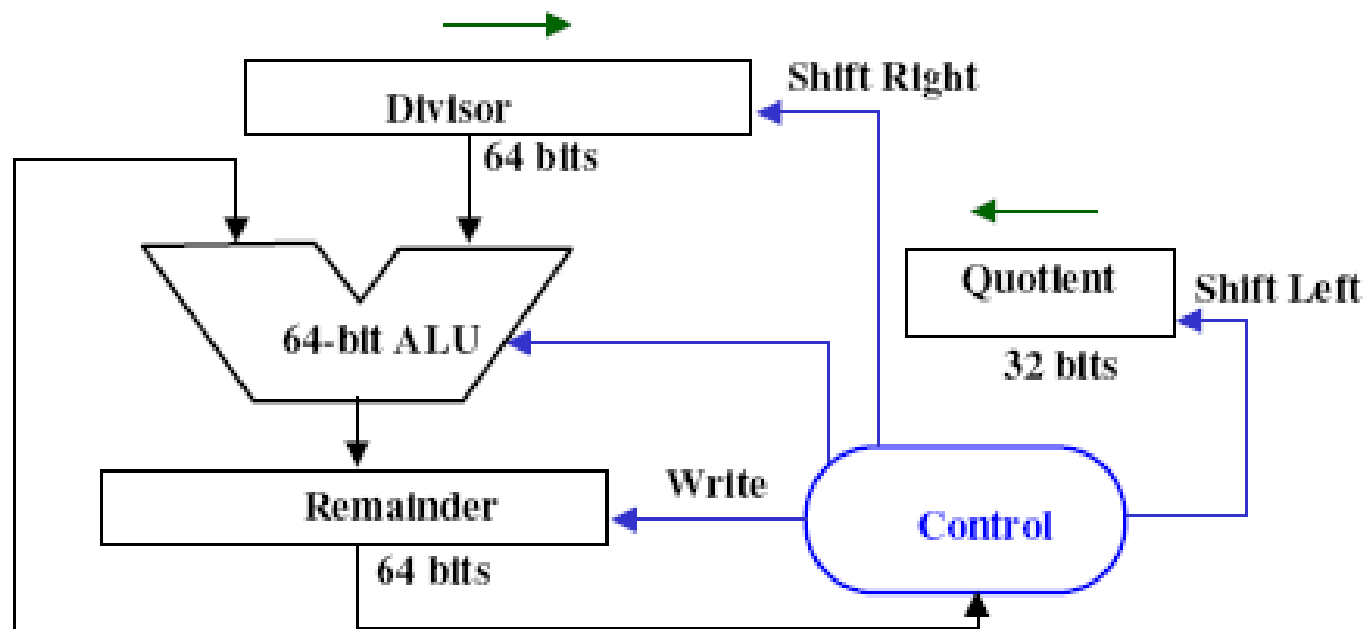
```

- See how big a number can be subtracted, creating quotient bit on each step
- Binary \Rightarrow 1 * divisor or 0 * divisor
- Dividend = Quotient x Divisor + Remainder
 \Rightarrow | Dividend | = | Quotient | + | Divisor |
- 3 versions of divide, successive refinement

Divisori (in breve ;)

DIVIDE HARDWARE Version 1

- 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg

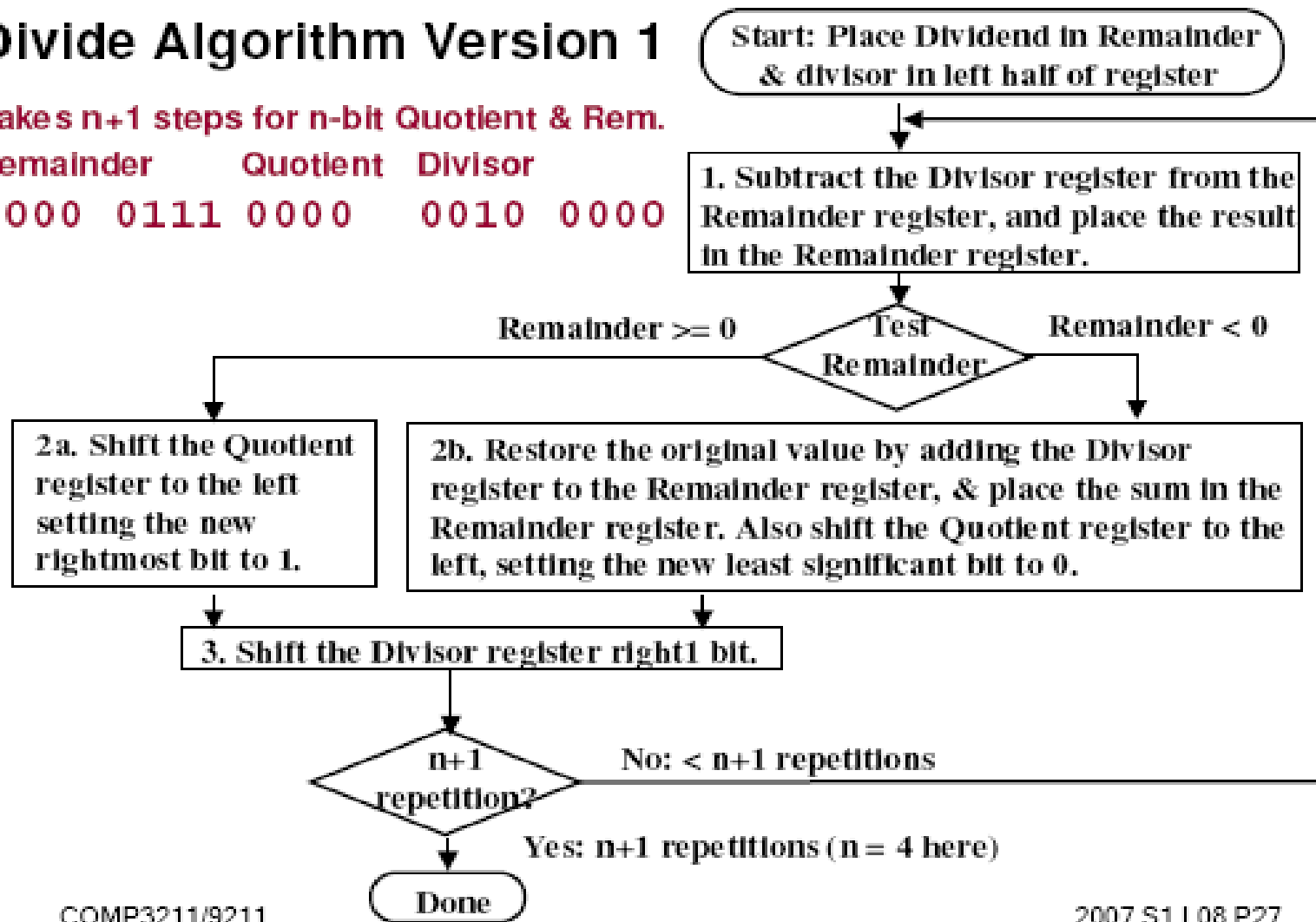


Divisori (in breve ;)

Divide Algorithm Version 1

Takes $n+1$ steps for n -bit Quotient & Rem.

Remainder	Quotient	Divisor
0000	0111	0000
0010	0000	0000



Divisori (in breve ;))

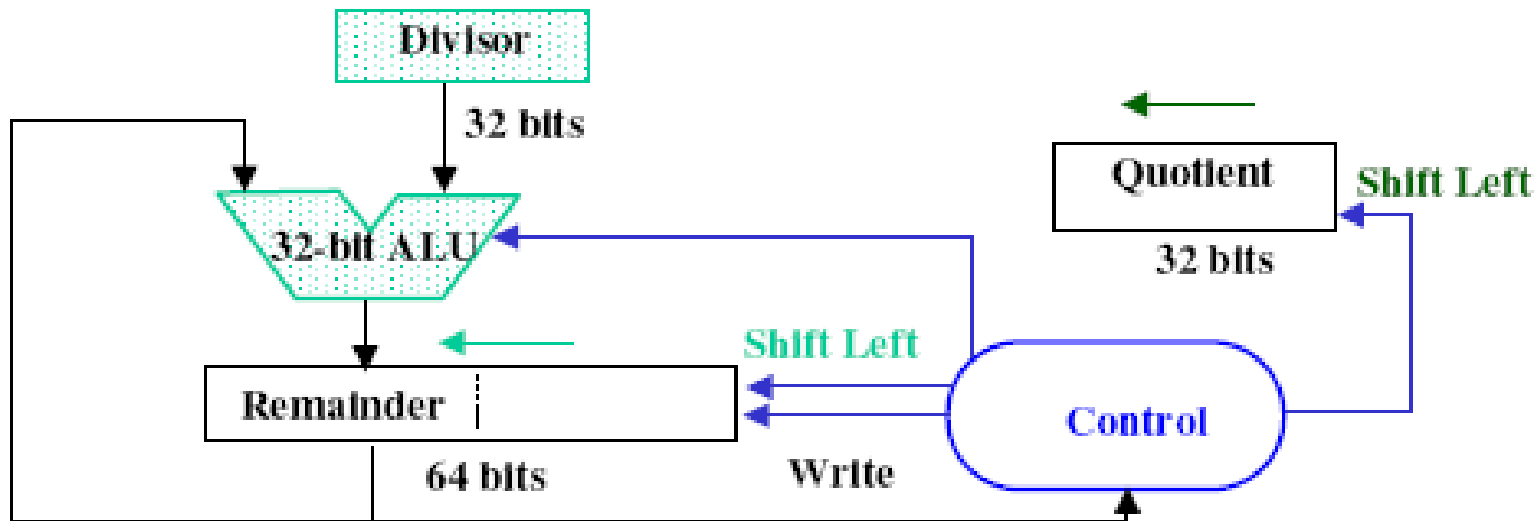
Observations on Divide Version 1

- **1/2 bits in divisor always 0**
 - 1/2 of 64-bit adder is wasted
 - 1/2 of divisor is wasted
- **Instead of shifting divisor to right, shift remainder to left?**
- **1st step cannot produce a 1 in quotient bit (otherwise too big)**
 - switch order to shift first and then subtract, can save 1 iteration

Divisori (in breve ;)

DIVIDE HARDWARE Version 2

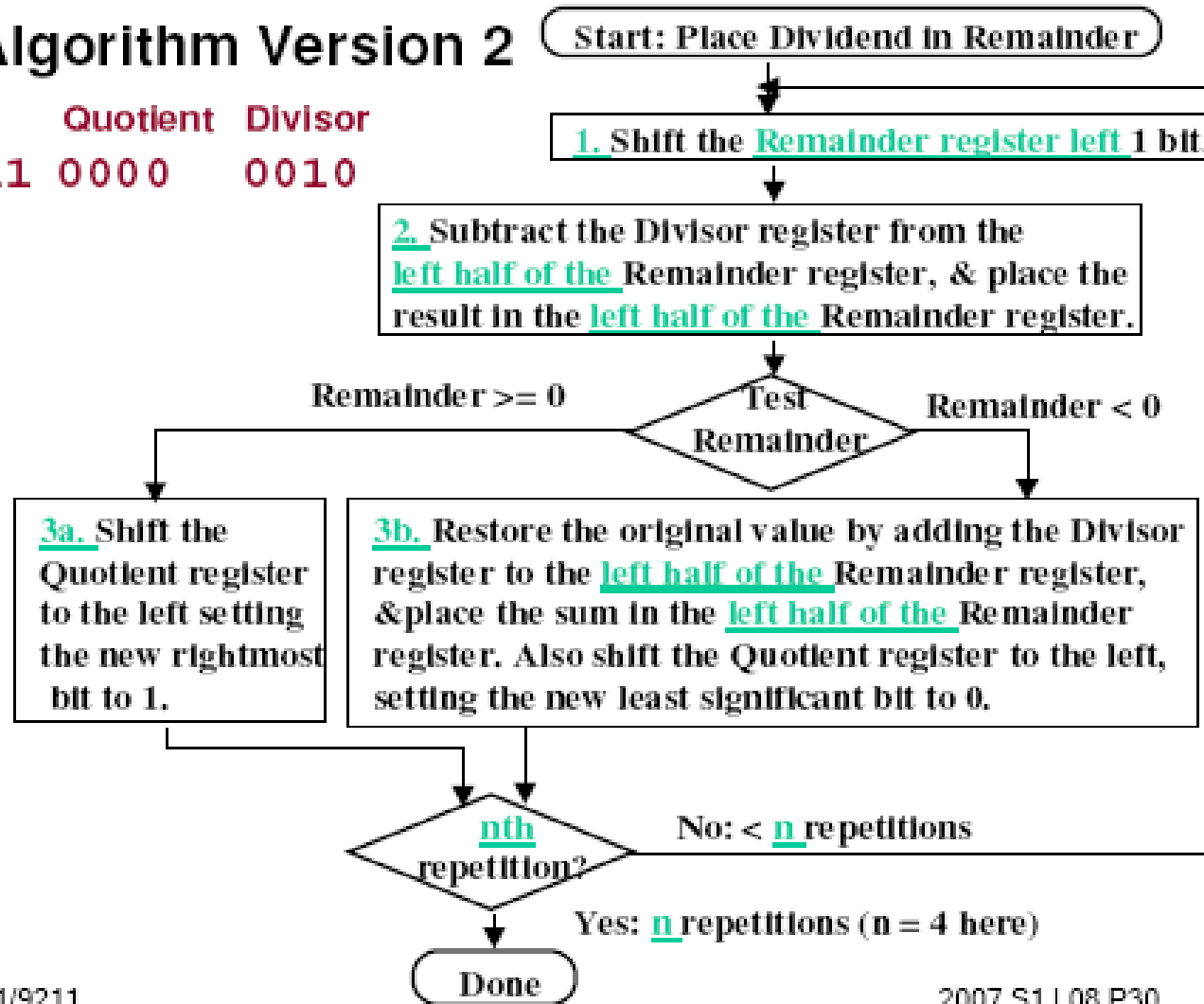
- 32-bit Divisor reg, 32-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg



Divisori (in breve ;)

Divide Algorithm Version 2

Remainder	Quotient	Divisor
0000	0111	0010



Divisori (in breve ;))

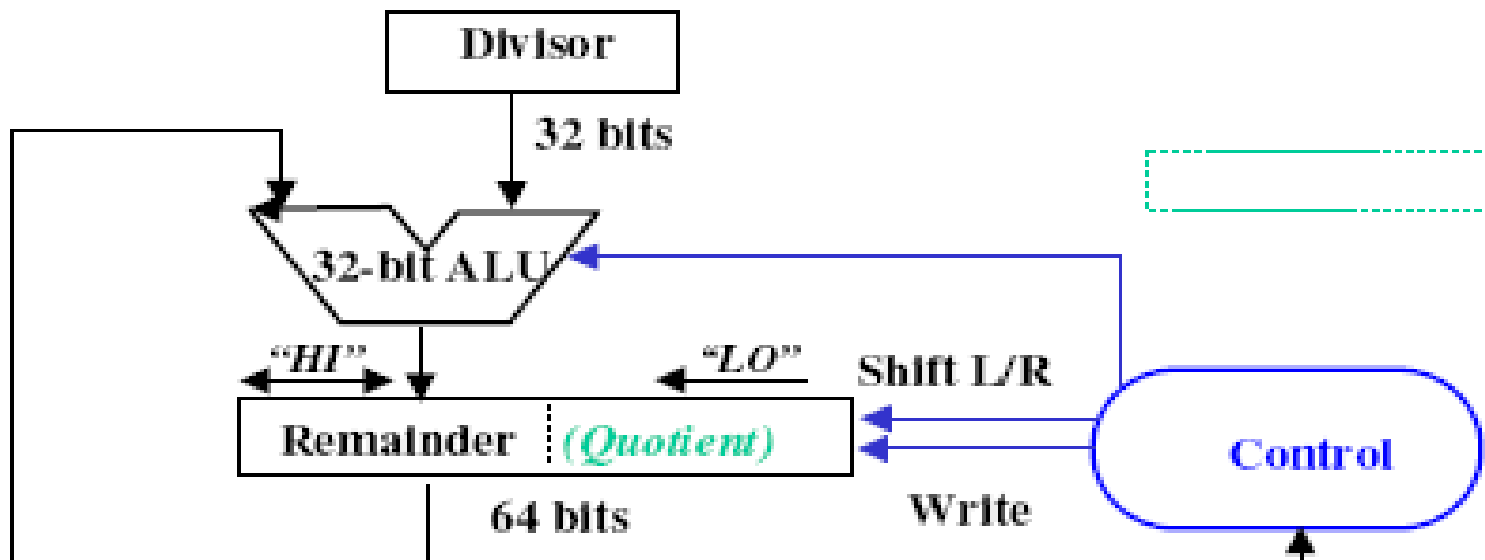
Observations on Divide Version 2

- **Eliminate Quotient register by combining with Remainder as shifted left**
 - Start by shifting the Remainder left as before.
 - Thereafter loop contains only two steps because the shifting of the Remainder register shifts both the remainder in the left half and the quotient in the right half
 - The consequence of combining the two registers together and the new order of the operations in the loop is that the remainder will be shifted left one time too many.
 - Thus a final correction step must shift back only the remainder in the left half of the register

Divisori (in breve ;)

DIVIDE HARDWARE Version 3

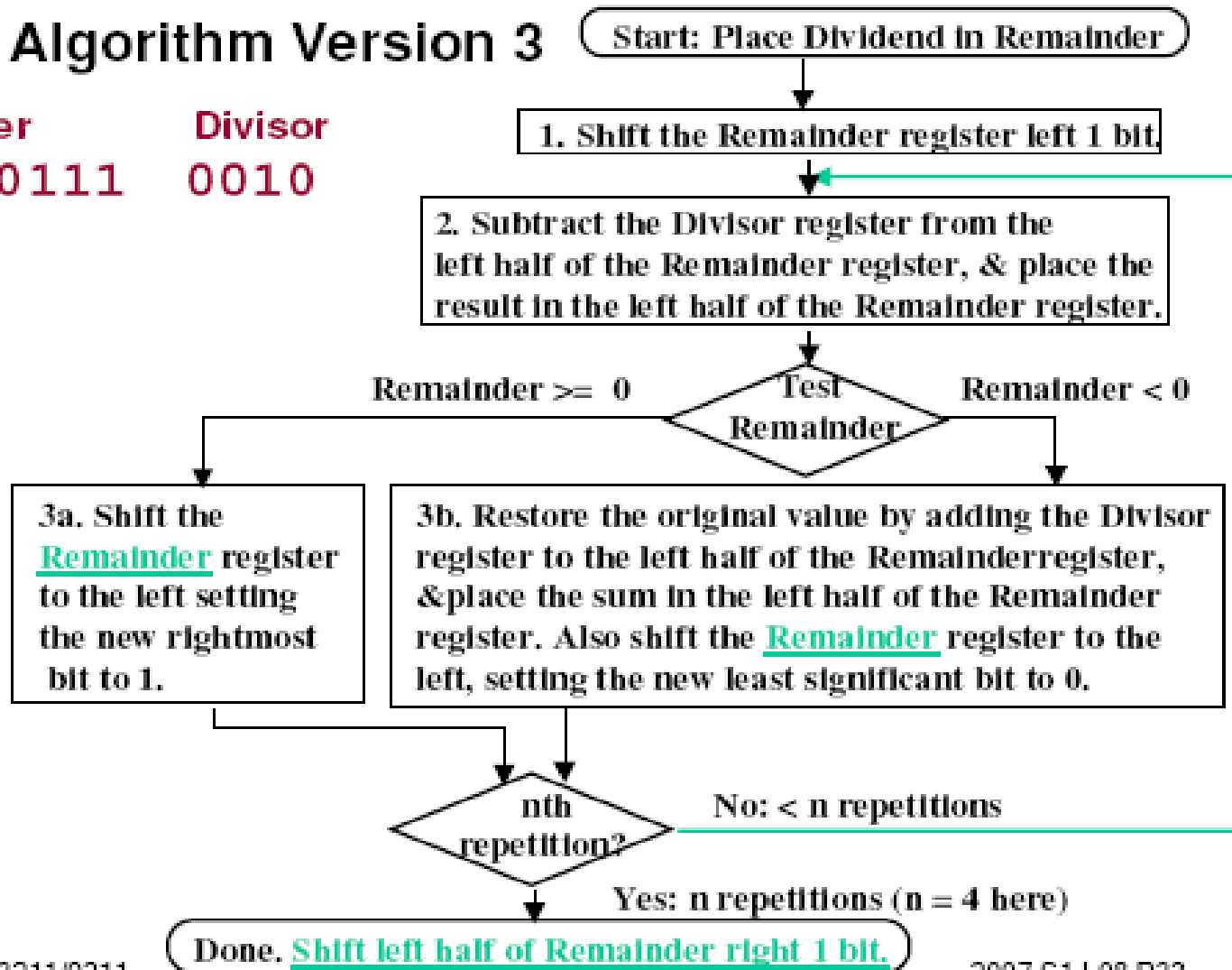
- 32-bit Divisor reg, 32-bit ALU, 64-bit Remainder reg, (0-bit Quotient reg)



Moltiplicatori: Implementazioni

Divide Algorithm Version 3

Remainder Divisor
 0000 0111 0010



Moltiplicatori: Implementazioni

Observations on Divide Version 3

- **Same Hardware as Multiply:** just need ALU to add or subtract, and 63-bit register to shift left or shift right
- **Hi and Lo registers in MIPS** combine to act as 64-bit register for multiply and divide
- **Signed Divides:** Simplest is to remember signs, make positive, and complement quotient and remainder if necessary
 - **Note:** Dividend and Remainder must have same sign
 - **Note:** Quotient negated if Divisor sign & Dividend sign disagree
e.g., $-7 \div 2 = -3$, remainder = -1

Aritmetica FP

FP addition example

Let's say you want to compute $9.999_{10} \times 10^1 + 1.610_{10} \times 10^{-1}$

Assume we can store 4 **significand** digits and 2 **exponent** digits

Step 1: Align decimal point of number with smaller exponent:

$$1.610_{10} \times 10^{-1} = 0.1610_{10} \times 10^0 = 0.01610_{10} \times 10^1 = 0.016_{10} \times 10^1$$

Step 2: Add the significands:

$$9.999_{10} + 0.016_{10} = 10.015_{10} \times 10^1$$

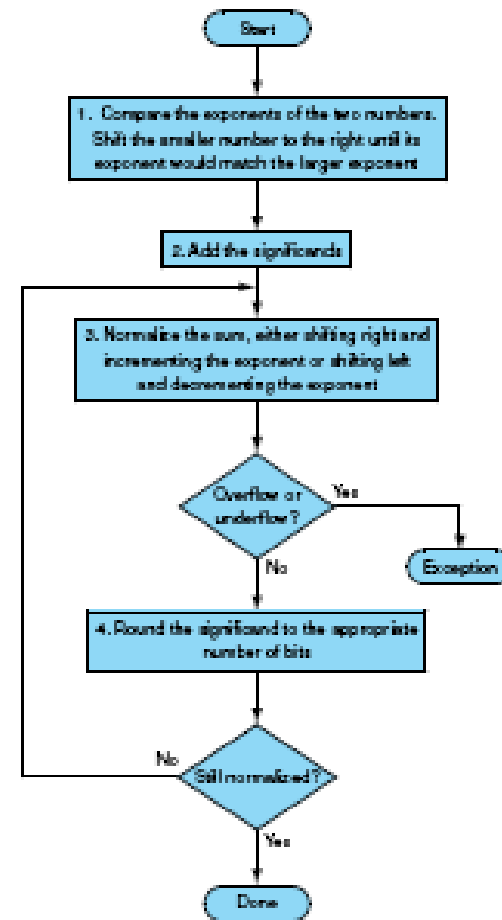
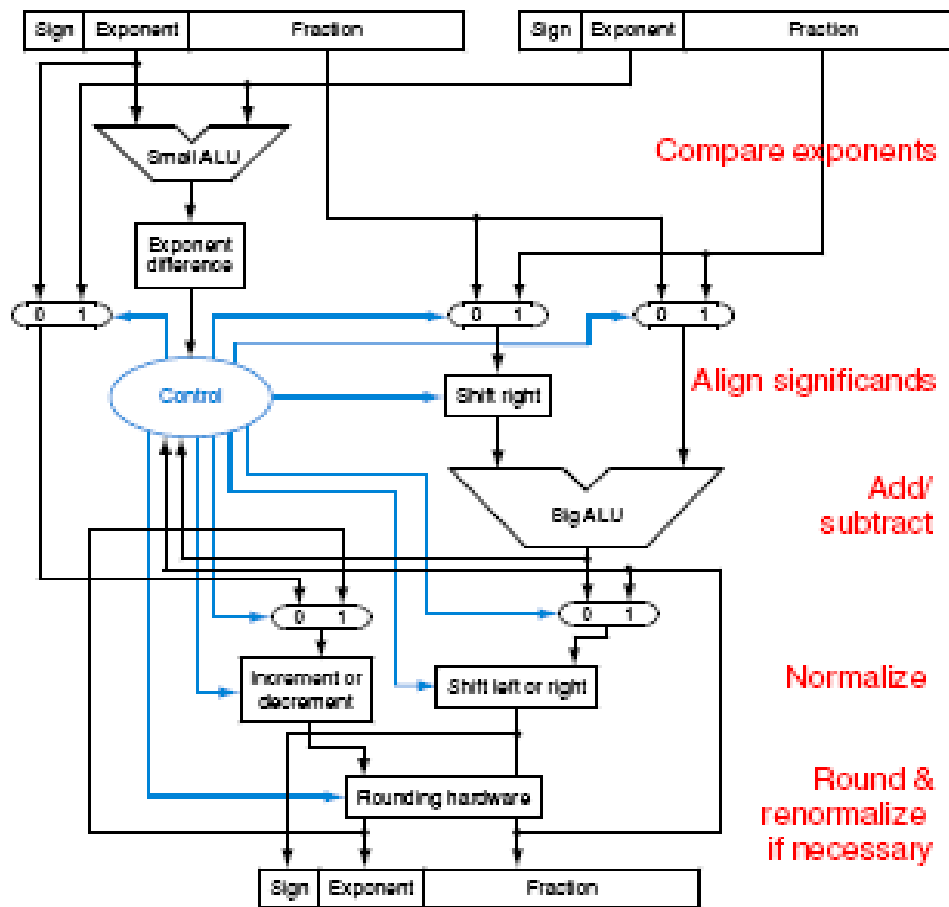
Step 3: Normalize the result & check for overflow/underflow:

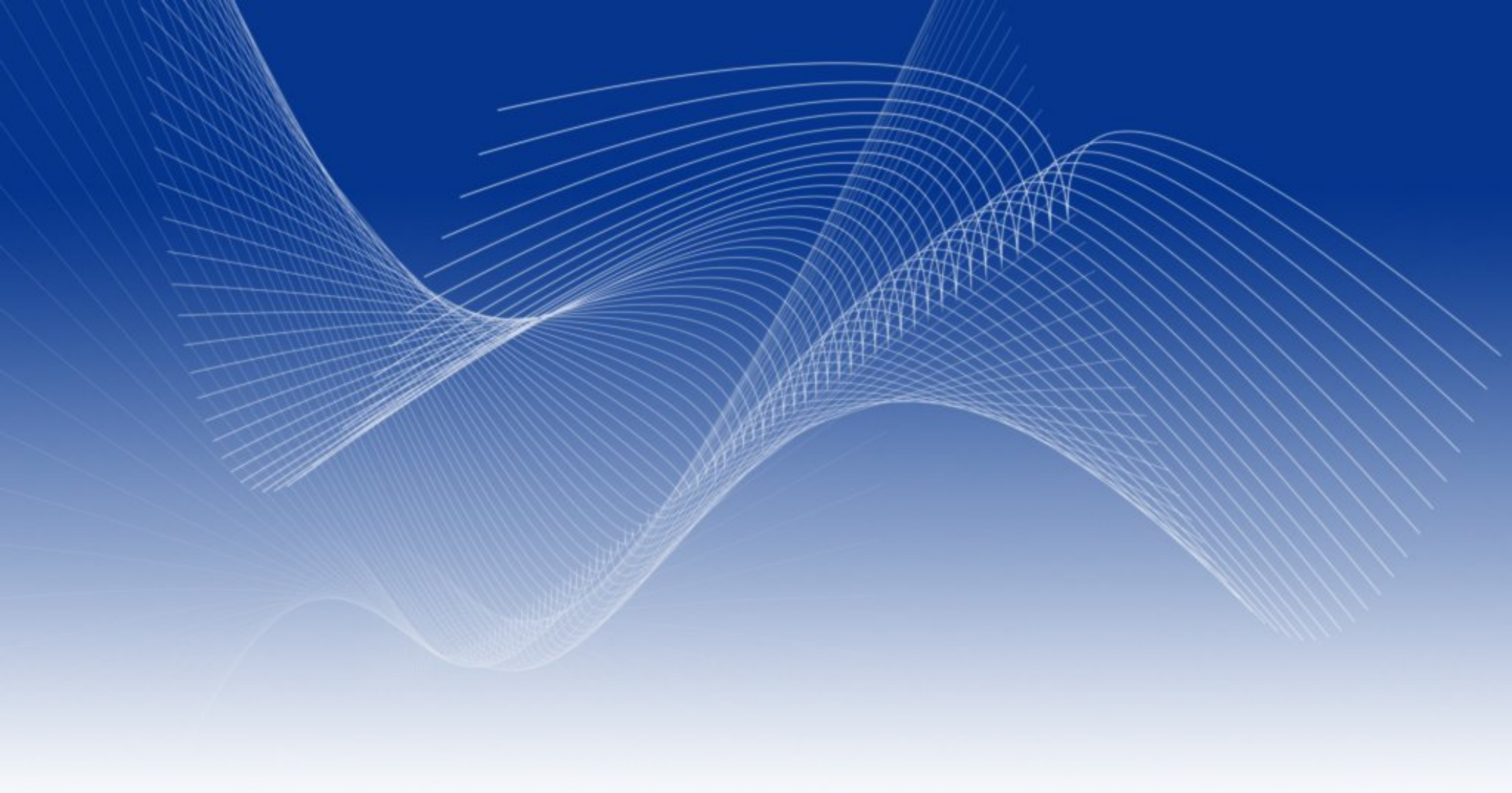
$$10.015_{10} \times 10^1 = 1.0015_{10} \times 10^2$$

Step 4: Round and renormalize if necessary:

$$1.0015_{10} \times 10^2 = 1.002_{10} \times 10^2$$

Floating point addition hardware





Your Name
Your Title

Your Organization (Line #1)
Your Organization (Line #2)