Architettura degli Elaboratori

Moltiplicazione e divisione tra numeri interi: algoritmi e circuiti

slide a cura di Salvatore Orlando, Andrea Torsello, Marta Simeoni

Operazioni aritmetiche e logiche

Abbiamo visto che le ALU sono in grado di eseguire operazioni di

- somma
- sottrazione
- salto
- confronto
- operazioni logiche: AND e OR

Rimangono però altre due operazioni da realizzare: MOLTIPLICAZIONE e DIVISIONE di numeri interi. Nel MIPS abbiamo:

```
mult $5 $4      # Hi,Lo = $5 x $4  prodotto con segno
multu $5 $4     # Hi,Lo = $5 x $4  prodotto senza segno
div $3 $4       # Hi = $3 mod $4,  Lo = $3 / $4 divisione con segno
divu $3 $4      # Hi = $3 mod $4,  Lo = $3 / $4 divisione senza segno
```

Vediamo come vengono realizzate…

Moltiplicazione: algoritmo carta e penna

Come funziona l'algoritmo "carta e penna" per la moltiplicazione ?

```
     105 *        Moltiplicando
     204 =        Moltiplicatore
 ----------
      420
     000
    210
 ----------
   21420          Prodotto
```

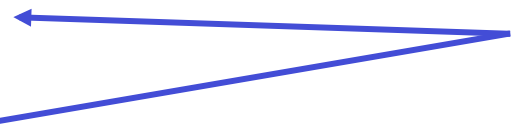
I passi dell'algoritmo sono i seguenti :

- considerare le cifre del moltiplicatore una alla volta, da destra a sinistra
- moltiplicare il moltiplicando per la singola cifra del moltiplicatore
- scalare il prodotto intermedio di una cifra alla volta verso sx (shift a sx)
- il prodotto finale si ottiene dalla somma di tutti i prodotti intermedi

Algoritmo carta e penna (base 2)

In base 2, la presenza delle sole cifre 0 e 1 semplifica i vari prodotti?

```
    1011 *        Moltiplicando  11(10)
     101 =        Moltiplicatore  5(10)
----------
    1011
   0000
  1011
----------
  110111          Prodotto  55(10)
```



Addendi della somma uguali al Moltiplicando (shiftato a sx opportunamente)

Osservazioni:

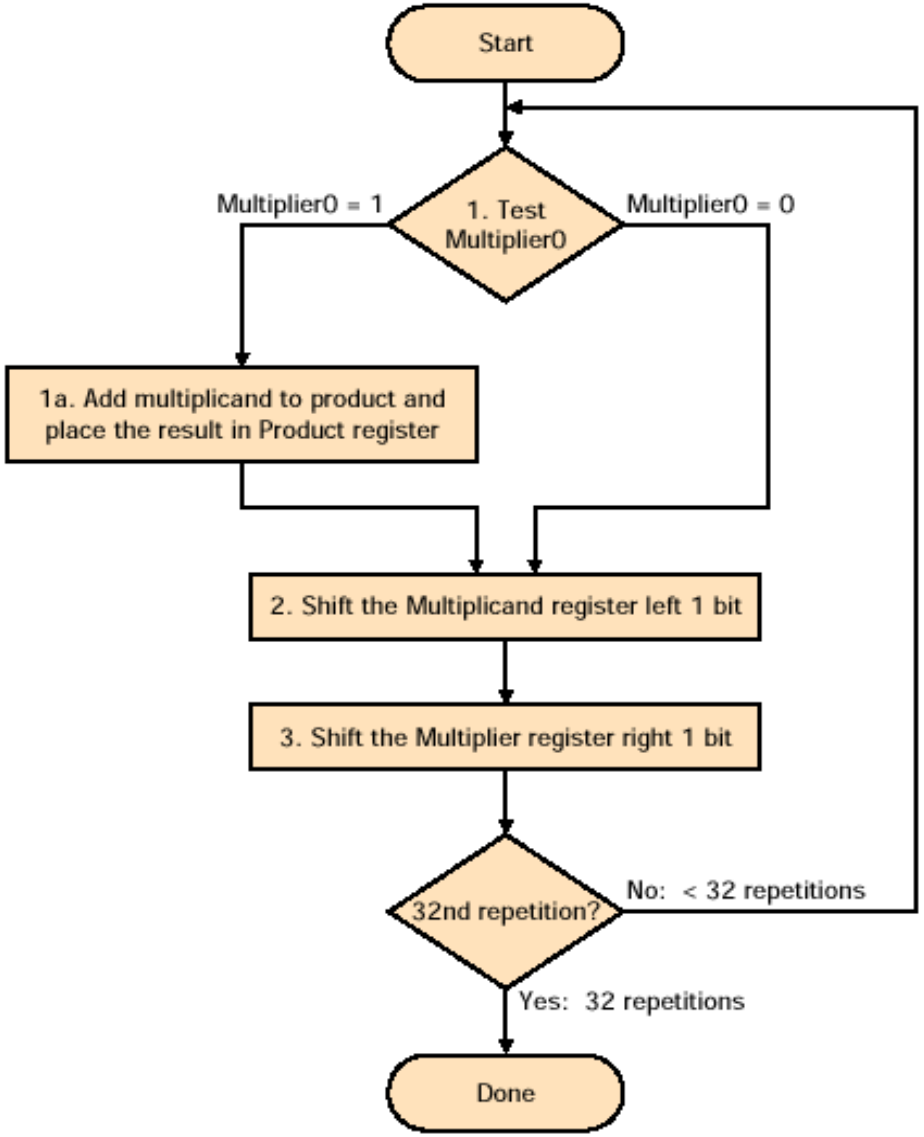
- il numero di cifre del prodotto è molto più grande rispetto alle cifre del moltiplicatore e del moltiplicando.
- Ignorando i bit di segno si ha:
  moltiplicatore (n bit) moltiplicando (n bit) ==> prodotto (2n bit)
- In generale, per rappresentare il risultato abbiamo bisogno del doppio dei bit dei due fattori del prodotto

Moltiplicazione tra numeri interi

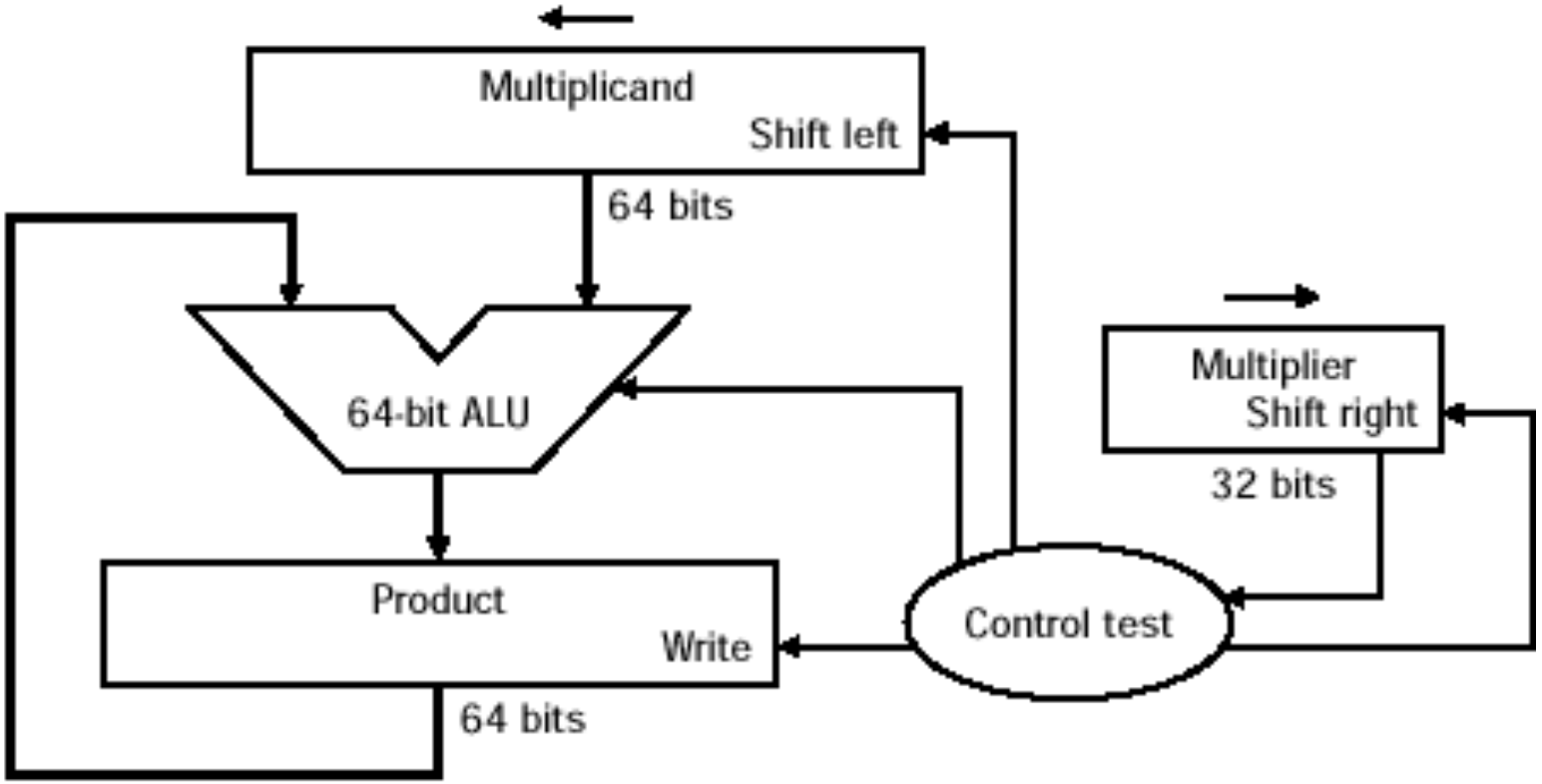
Si basa su:

- shift a sx del moltiplicando per rappresentare i prodotti parziali
- shift a dx del moltiplicatore per considerare, una alla volta, tutte le sue cifre (dalla meno significativa alla più significativa)
- somma di moltiplicando e prodotto

Moltiplicando	Moltiplicatore	Prodotto
00001011	**0101**	0000 1011
00010110	0**010**	1011
00101100	00**01**	1011 110111
01011000	000**0**	110111
10110000	0000	110111



Circuito per la moltiplicazione



Moltiplicazione: ottimizzazione

L'algoritmo di moltiplicazione precedente usa un registro a 64bit per il moltiplicando e, conseguentemente, una ALU a 64bit per la somma.
Infatti, Il moltiplicando viene scalato a sinistra di una cifra ad ogni passo, in modo da non influenzare i bit meno significativi del prodotto

Vediamo ora un algoritmo che, per arrivare allo stesso obiettivo, prevede di scalare il prodotto a destra.
In tal modo il moltiplicando può rimanere fisso e quindi sono necessari solo 32bit sia per il registro corrispondente che per l'ALU.

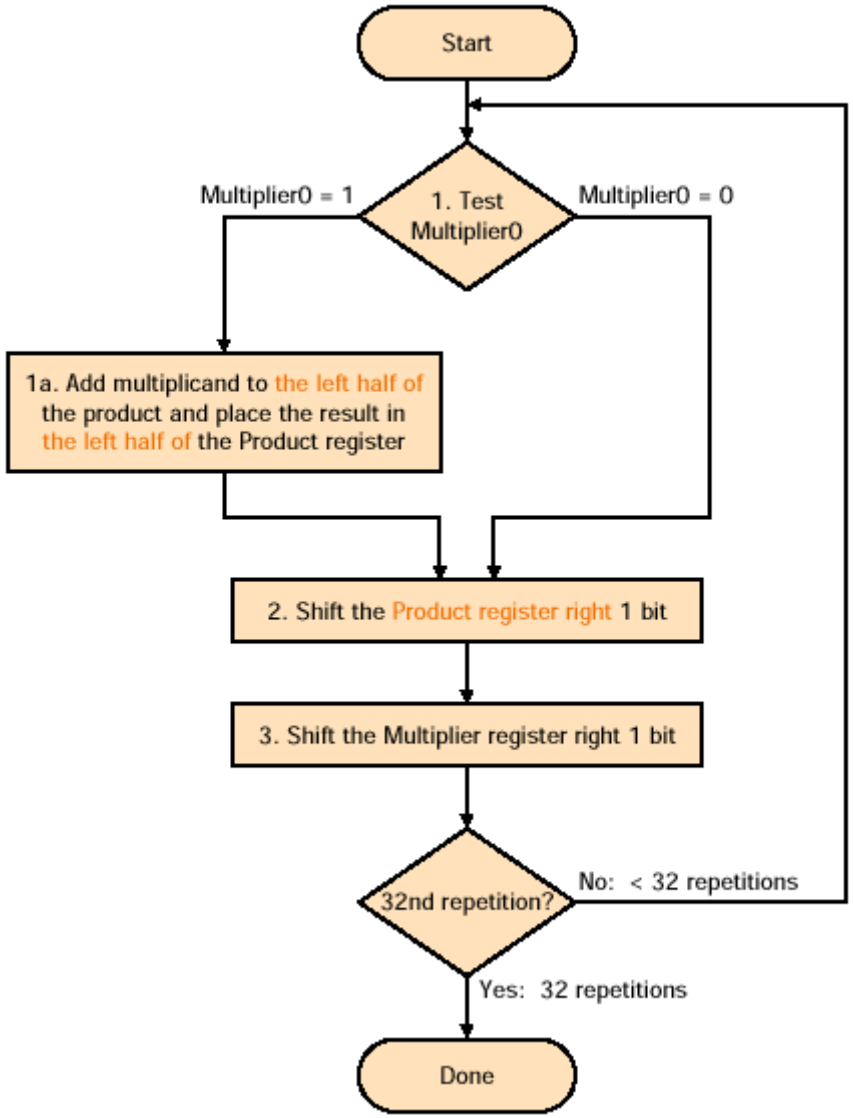
Risultato: il circuito diventa più semplice e la moltiplicazione più veloce

Moltiplicazione intera: ottimizzazione

Si basa su:

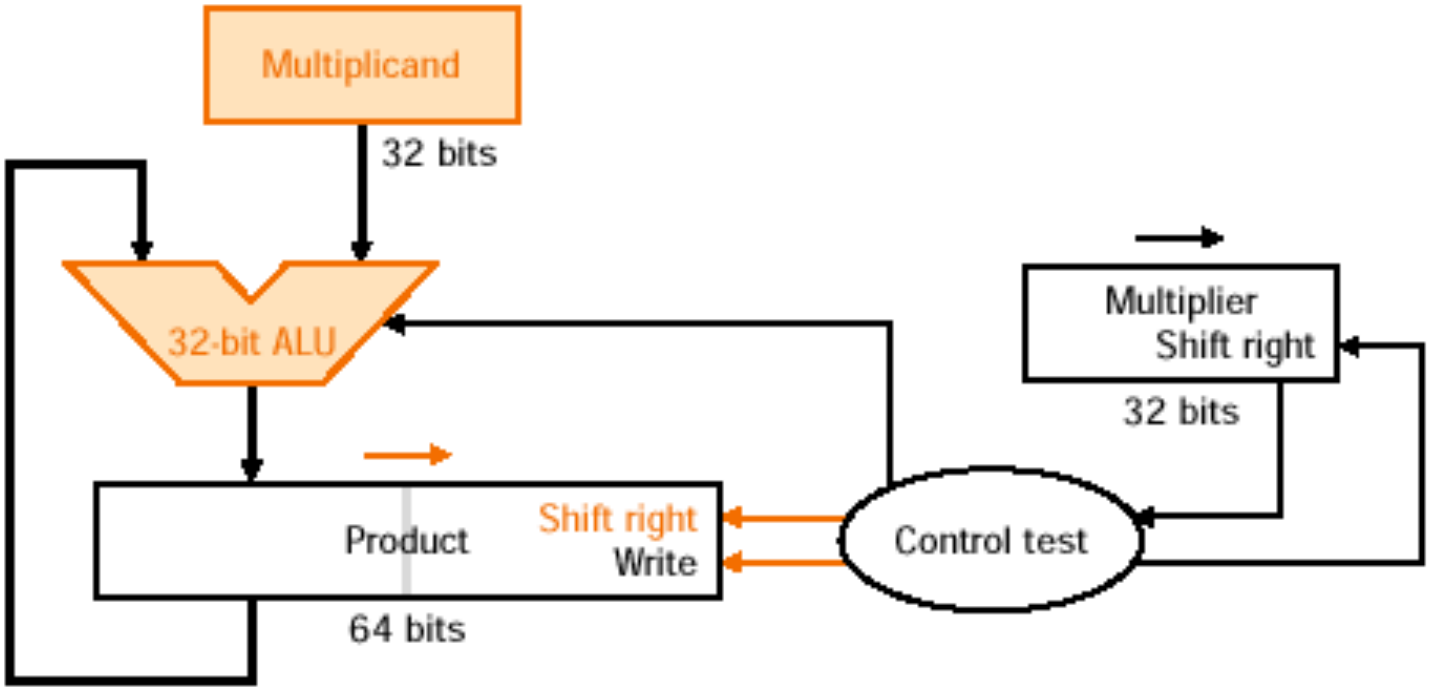
- shift a destra del moltiplicatore per considerare, una alla volta, tutte le sue cifre (dalla meno significativa alla più significativa)
- shift a destra del prodotto per non influenzare le cifre meno significative dei prodotti parziali
- ad ogni passo vengono sommati solo i 32bit più significativi del prodotto

Moltiplicando	Moltiplicatore	Prodotto
1011	**0101**	00000000
		10110000
1011	0**010**	01011000
1011	00**01**	00101100
		11011100
1011	000**0**	01101110
1011	0000	00110111



Moltiplicazione intera: secondo circuito

Il circuito ottimizzato che realizza la moltiplicazione è il seguente:



Moltiplicazione intera: terzo circuito

Il secondo algoritmo spreca inizialmente i 32bit bassi del prodotto, che sono esattamente lo spazio necessario per memorizzare il moltiplicatore

Man mano che lo spazio sprecato del prodotto diminuisce (per via dello shift a destra) diminuiscono anche le cifre significative del moltiplicatore

Il terzo algoritmo prevede quindi la memorizzazione del moltiplicatore nella parte bassa del prodotto

Con questa soluzione:

- si elimina un registro (quello che prima memorizzava il moltiplicatore)
- il test sulla cifra del moltiplicatore da considerare ad ogni passo diventa il test sul bit meno significativo del prodotto

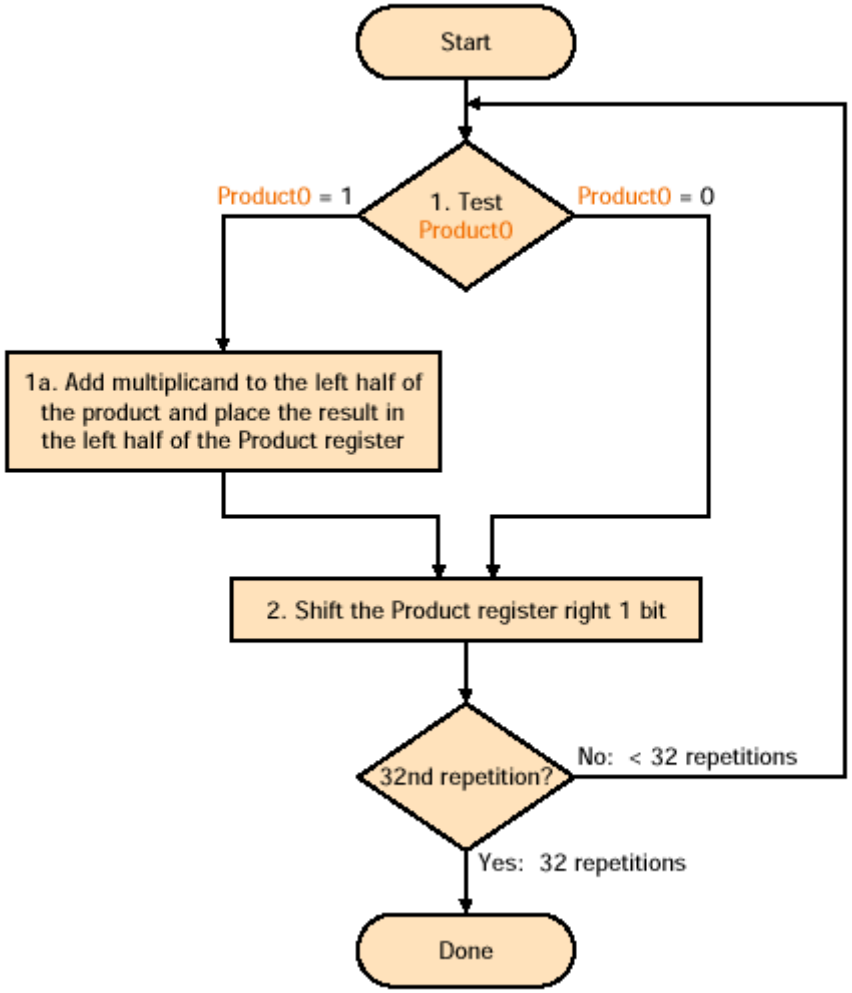
Vediamo nei dettagli il terzo (e ultimo!!!) algoritmo per la moltiplicazione...

Moltiplicazione intera: terzo circuito

Si basa su:

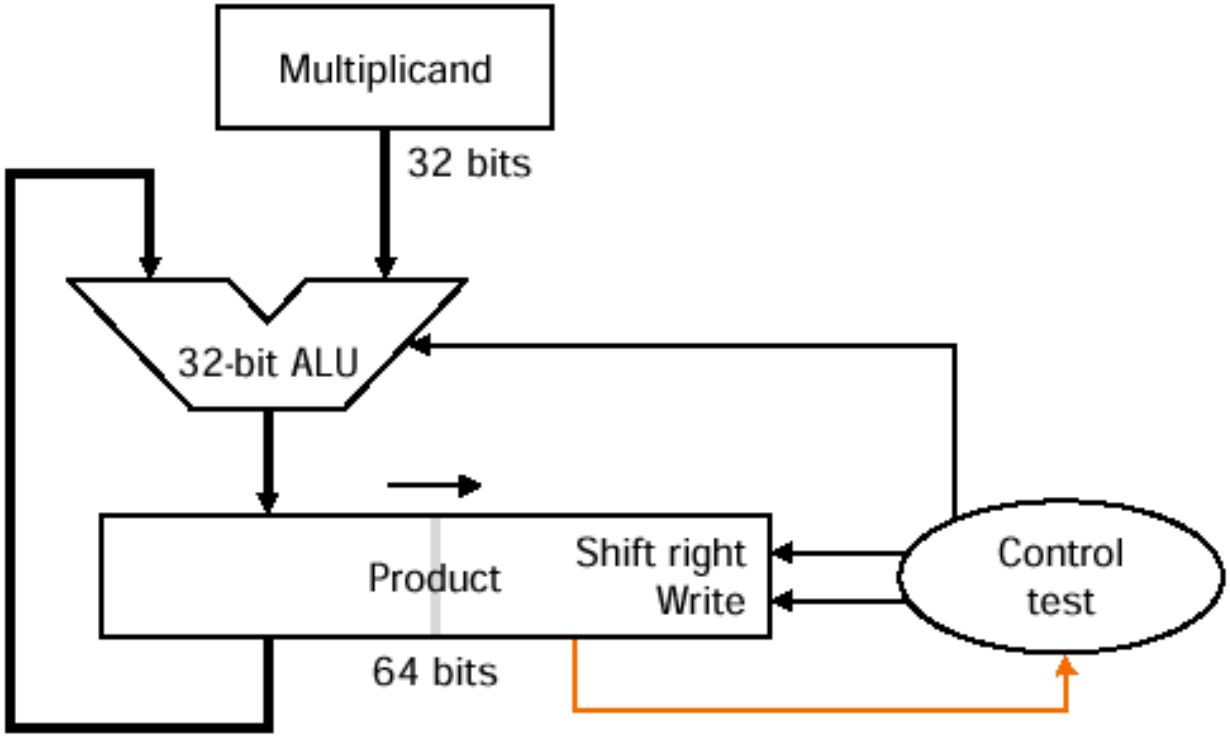
- memorizzazione del moltiplicatore nella parte bassa del prodotto
- shift a destra del prodotto per non influenzare le cifre meno significative dei prodotti parziali
- ad ogni passo vengono sommati solo i 32bit più significativi del prodotto

Moltiplicando	Prodotto
1011	00000101
1011	10110101
	01011010
1011	01011010
	00101101
1011	11011101
	01101110
1011	01101110
	00110111



Moltiplicazione intera: terzo circuito

Il circuito che realizza il terzo algoritmo per la moltiplicazione è il seguente:



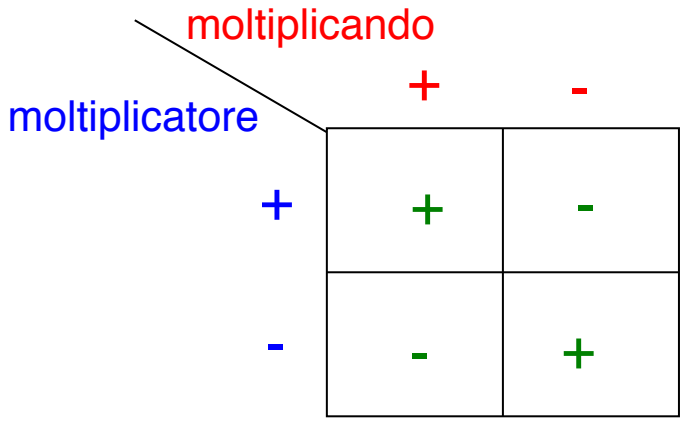
Segno della moltiplicazione

Finora non abbiamo considerato il SEGNO di moltiplicando e moltiplicatore

Il modo più semplice di gestire il segno è il seguente:

- convertire moltiplicando e moltiplicatore in numeri positivi
- eseguire la moltiplicazione
- stabilire il segno del prodotto secondo la regola dei segni, sulla base dei segni originali, e complementare se necessario

ATTENZIONE: gli algoritmi possono limitarsi ad effettuare solo 31 iterazioni, lasciando i segni fuori dal calcolo



Altri algoritmi di Moltiplicazione

Esiste un algoritmo, detto Algoritmo di Booth, che consente di moltiplicare direttamente numeri con segno.

Divisione tra numeri interi

Come funziona l'algoritmo di divisione "carta e penna" ?

Dividendo Divisore Quoziente

```
  100     :    6        =   016
  10
  06
   40
   36
    4   Resto
```

Si ha: Dividendo = Quoziente * Divisore + Resto
da cui: Resto = Dividendo - Quoziente * Divisore

I passi dell'algoritmo sono i seguenti:

- considerare gruppi di cifre del dividendo, partendo dalla cifra più significativa
- vedere quante volte il divisore sta nel gruppo di cifre del dividendo
- scrivere la cifra corrispondente nel quoziente
- sottrarre il multiplo del divisore dal dividendo
- alla fine il resto dovrà essere 0 o essere minore del divisore

ATTENZIONE: la divisione per ZERO è erronea

Divisione tra numeri interi (base 2)

Cosa succede in base 2 ?

Dividendo Divisore Quoziente

```
1101001  :   101    =   0010101
1
11
110
101
 11
 110
 101
  10
  101
  101
    0   Resto
```

- Quante volte il divisore sta nella porzione di dividendo considerata?
  ci sono solo due alternative: *0 volte* oppure *1 volta*
- Questo semplifica l'algoritmo di divisione
- Consideriamo per ora solo numeri positivi

Algoritmo di divisione tra interi

Supponiamo di dover dividere $1101_2 : 0101_2$ (cioè $13_{10} : 5_{10}$)

- Per semplificare, interi rappresentati su 4 bit e somma effettuata su 8 bit

L'algoritmo di divisione procede come segue:

- quoziente = 0, memorizzato su un registro a 8 bit
- divisore = 01010000, memorizzato sulla parte alta di un registro a 8 bit
- resto memorizzato su reg. a 8 bit, inizializzato col valore del dividendo: resto = 00001101 = dividendo

I circuiti non capiscono "al volo" quando il divisore è più piccolo della porzione di dividendo considerata

- ad ogni passo si effettua quindi la sottrazione (dividendo - divisore) e si controlla il segno del risultato. Poiché il dividendo è memorizzato nel registro resto la sottrazione da effettuare è (resto – divisore)

Ad ogni passo si esegue lo shift a dx di una posizione del divisore

Algoritmo di divisione tra interi

```
resto    =  00001101-     resto    = 00001101-     resto   = 00001101-
divisore =  01010000     divisore = 00101000     divisore = 00010100
            10111101                 11100101                 11111001
```

Prima sottrazione: controlla se ci sta il divisore in zero cifre del dividendo;

Seconda sottrazione: controlla se ci sta il divisore in una cifra del dividendo;

Terza sottrazione: controlla se ci sta il divisore in due cifre del dividendo;

se il segno di (Resto – Divisore) è positivo

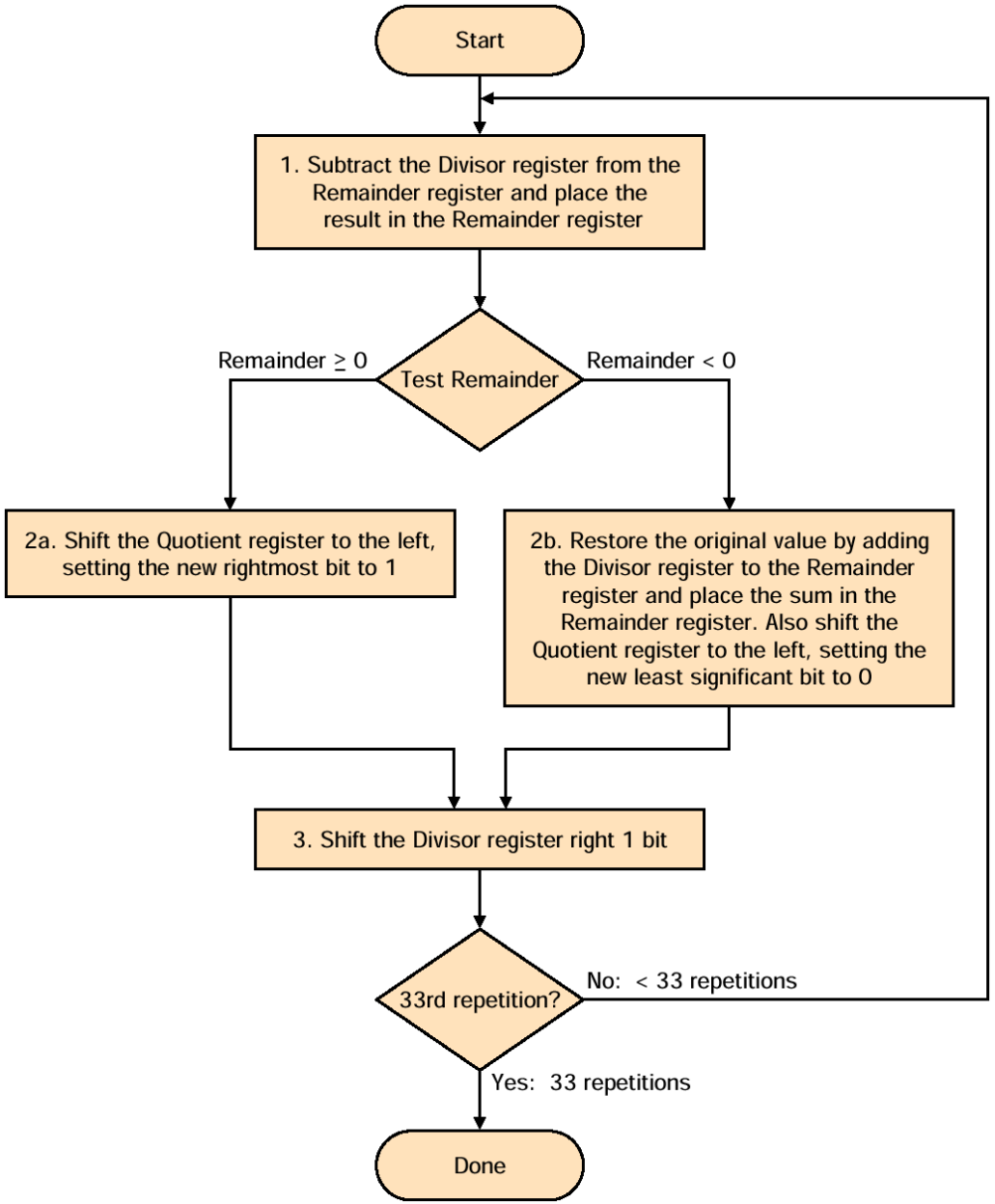
- shift a sinistra del quoziente e inserimento di 1

se il segno di (Resto – Divisore) è negativo

- shift a sinistra del quoziente e inserimento di 0
- ripristino del valore precedente di resto

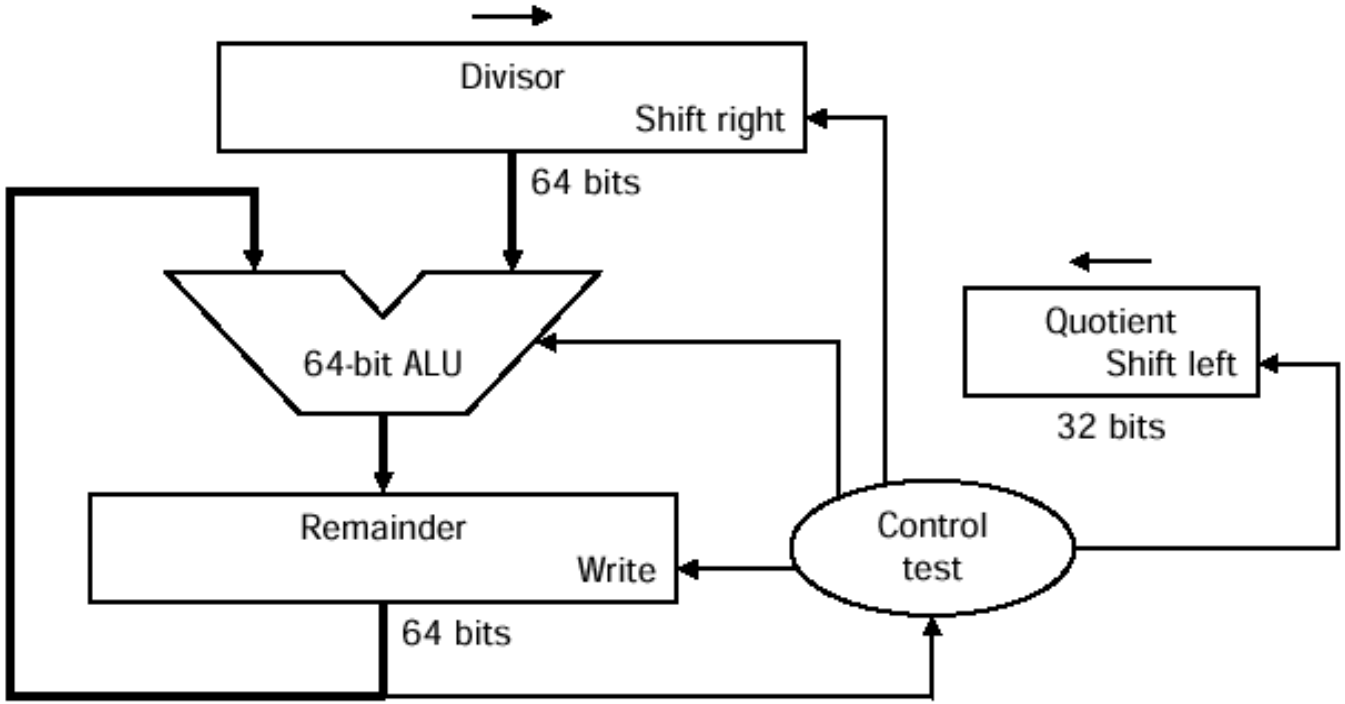
Algoritmo di divisione tra interi

Resto	Divisore	Quoziente
00001101- 01010000= 10111101	01010000	0000
00001101- 00101000= 11100101	00101000	0000
00001101- 00010100= 11111001	00010100	0000
00001101- 00001010= 00000011	00001010	0000
00000011- 00000101= 11111110	00000101	0001
00000011	00000010	0010



Divisione tra interi: circuito

Il circuito che implementa il primo algoritmo è il seguente:

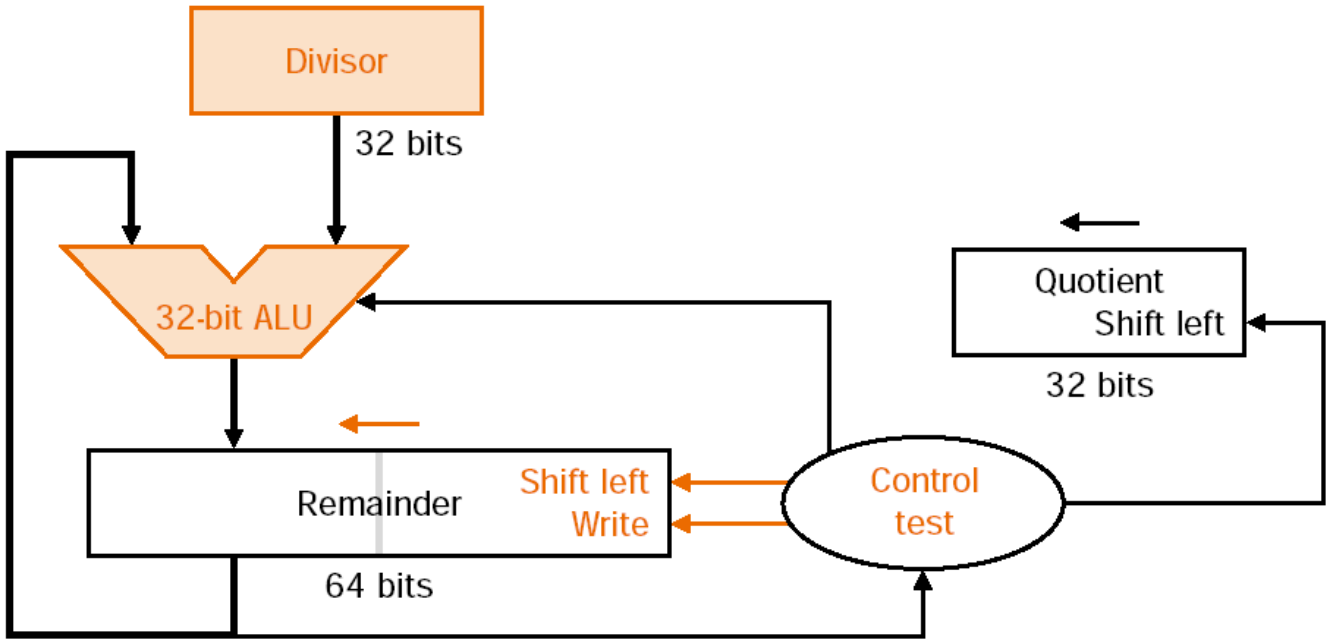


Divisione tra interi (ottimizzazioni)

Analogamente al caso della moltiplicazione, sono stati studiati dei raffinamenti per l'algoritmo della divisione. L'obiettivo è sempre quello di semplificare e rendere più veloce il circuito che implementa la divisione

Vediamo ad esempio un circuito in cui si usa un'ALU a 32 bit

- Il registro del resto viene shiftato a sinistra
- Il divisore viene sottratto solo dalla parte alta del registro Resto



NOTA: un'ulteriore ottimizzazione prevede di memorizzare il quoziente nella parte bassa del resto.
Si elimina in questo modo il registro per il quoziente

Segno del quoziente

Finora non abbiamo considerato il SEGNO di dividendo e divisore

Analogamente al caso della moltiplicazione, il modo più semplice di gestire il segno è il seguente:

- convertire dividendo e divisore in numeri positivi
- eseguire la divisione lasciando i bit di segno fuori dal calcolo
- stabilire il segno del quoziente mediante la regola dei segni, ricordando i segni originali (quoziente negativo se i segni di dividendo e divisore sono discordi, positivo altrimenti)

Segno del resto

Stabilire il segno del resto mediante la seguente regola:

- dividendo e resto devono avere lo stesso segno

Resto positivo:

- Q = 5 / 2 = 2 R = 5 % 2 = 1 D = 5 d = 2
  - D = Q * d + R = 2 * 2 + 1 = 4+1=5
- Q = 5 / (-2) = -2 R = 5 % (-2) = 1
  - D = Q * d + R = (-2)*(-2) + 1 = 4+1=5

Resto negativo:

- Q = (-5) / 2 = -2 R = (-5) % 2 = -1 D = -5 d = 2
  - D = Q * d + R = (-2) * 2 + (-1) = -4-1 = -5
- Q = (-5) / (-2) = 2 R = (-5) % (-2) = -1 D = -5 d = -2
  - D = Q * d + R = (-2)*2 + (-1) = -4-1 = -5

Somma floating-point

Riprendiamo infine l'algoritmo relativo alla somma di numeri floating-point, allo scopo di vedere il circuito che lo realizza…

Esempio: 5.0 + 3.62510 con precisione 4 bit

$5.0_{10} = 101_2 = 1.01 \cdot 2^2$

$3.625_{10} = 11.101_2 \cdot 2^0 = 1.1101 \cdot 2^1$

Shifting: $3.62510 = 0.11101 \cdot 2^2$

Somma:
$$\begin{array}{r} 1.01000 + \\ 0.11101 = \\ \hline 10.00101 \cdot 2^2 \end{array}$$

Normalizzazione: $1.000101 \cdot 2^3$

Arrotondamento: $1.0001 \cdot 2^3$

Questo passo potrebbe dover essere iterato se l'arrotondamento richiede di sommare un'unità nella posizione meno significativa della mantissa

≠ troncamento

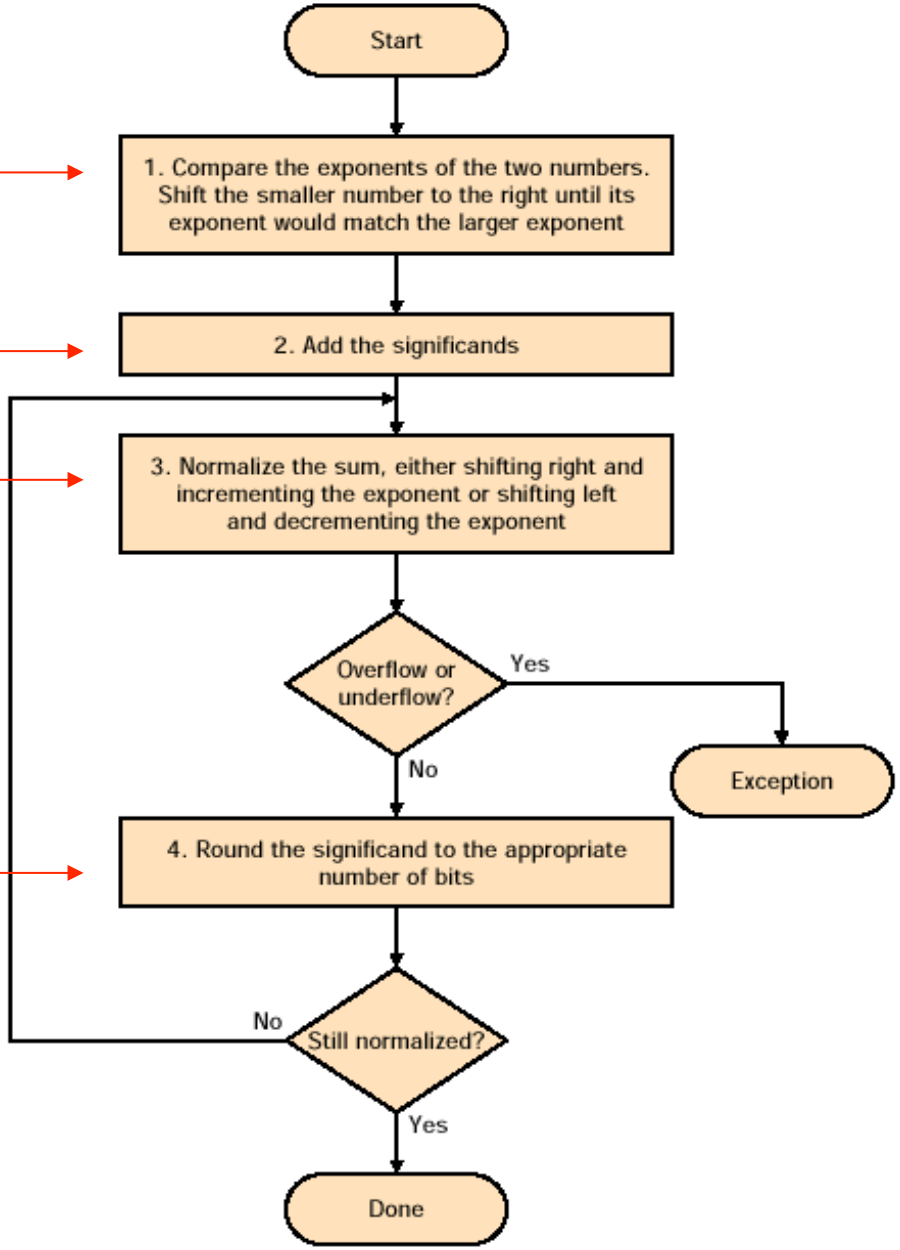
Algoritmo di somma floating-point

Shift

Somma

Normalizzazione

Arrotondamento



Circuito per la somma FP