La Moltiplicazione e la Divisione

Moltiplicazione in codifica binaria naturale

- Si seguono le regole note per la moltiplicazione decimale « in colonna »

	0	**1**
0	0	0
1	0	1

0 x	0 x	1 x	1 x
0 =	1 =	0 =	1 =
0	0	0	1

- Schema di moltiplicazione su 4 bit

				x_3	x_2	x_1	x_0	*
				y_3	y_2	y_1	y_0	=
				x_3y_0	x_2y_0	x_1y_0	x_0y_0	(P_0)
			x_3y_1	x_2y_1	x_1y_1	x_0y_1		(P_1)
		x_3y_2	x_2y_2	x_1y_2	x_0y_2			(P_2)
	x_3y_3	x_2y_3	x_1y_3	x_0y_3				(P_3)
q_7	q_6	q_5	q_4	q_3	q_2	q_1	q_0	(Q)

Il prodotto di parole di 4 bit richiede 8 bit:

Con n bit rappresento al massimo il valore 2^n-1 →

$$(2^n-1)(2^n-1) = 2^{2n} - 2^n - 2^n + 1 > 2^{2n-1}$$

Che è un valore rappresentabile su 2n bit

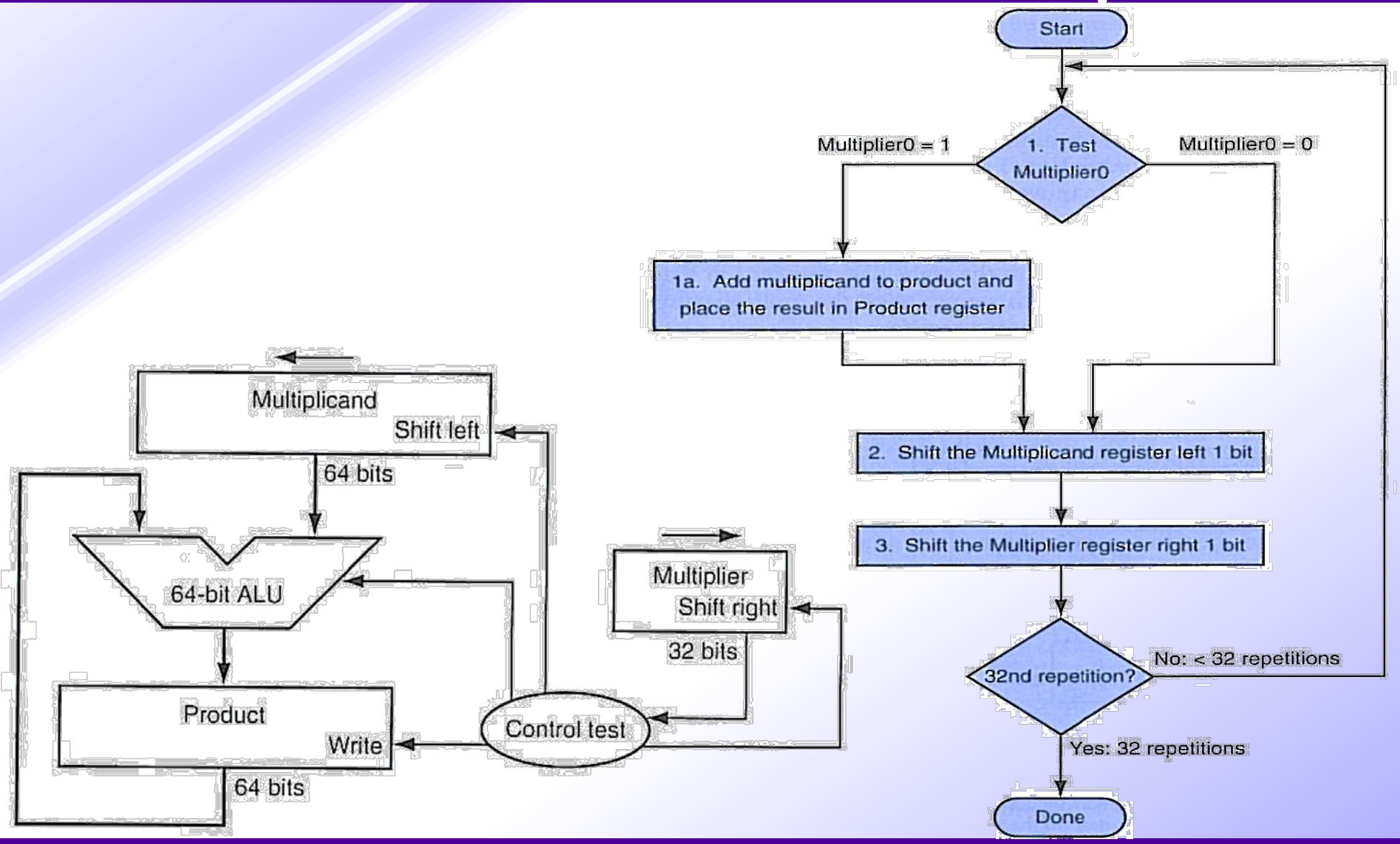
Moltiplicazione binaria - esempio

- Si consideri il prodotto $11_{10} \times 6_{10}$

				1	0	1	1	*
				0	1	1	0	=
				0	0	0	0	(P_0)
			1	0	1	1		(P_1)
		1	0	1	1			(P_2)
	0	0	0	0				(P_3)
0	1	0	0	0	0	1	0	(Q)

- I prodotti parziali P_i sono identicamente nulli oppure coincidono con il moltiplicando
  - $P_i = X$ se $y_i = 1$,
  - $P_i = 0$ altrimenti

Algoritmo della moltiplicazione e hardware - qui

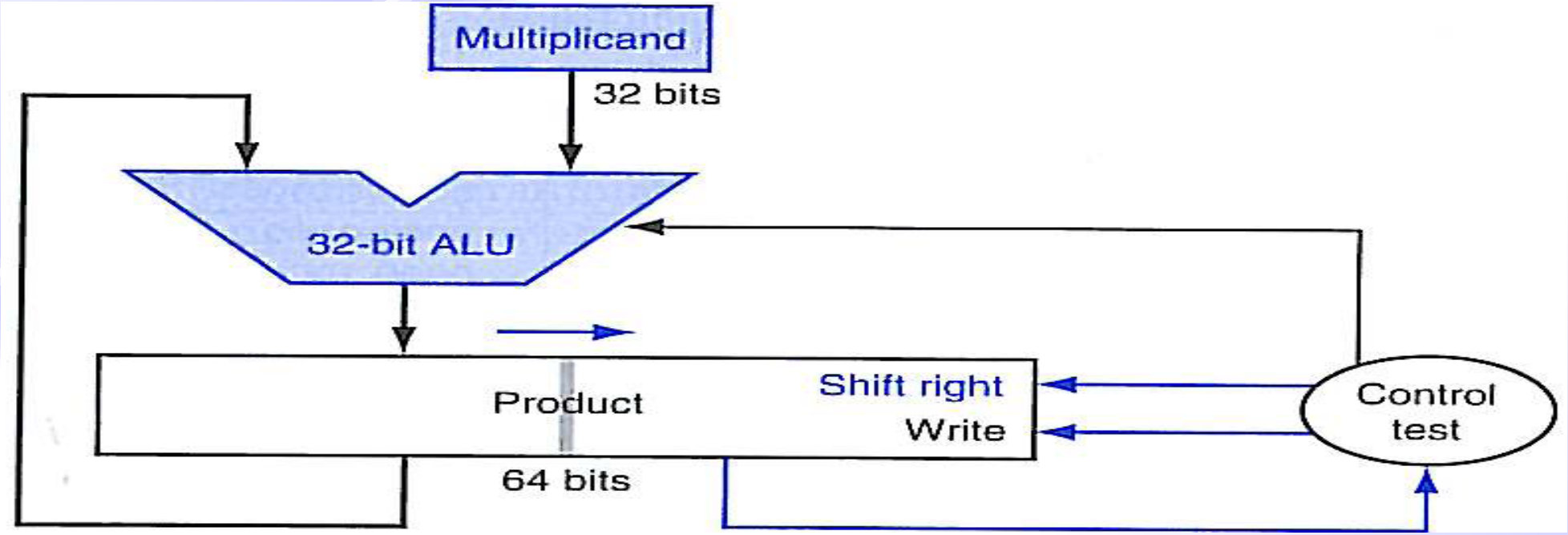


Algoritmo di moltiplicazione - Esempio

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

- Il bit esaminato per determinare il passo successivo dell'algoritmo è cerchiato

Ottimizzazione del circuito di moltiplicazione



- Registro moltiplicando e ALU a 32 bit
- Registro Prodotto a 64 bit che scorre a destra
- Il moltiplicatore è inserito nella metà destra del registro prodotto

Moltiplicazione in modulo e segno

- Si moltiplicano i valori assoluti e si calcola poi il segno del risultato
  - Operandi espressi su n bit
  - Valore assoluto su n-1 bit
  - Valori assoluto del risultato su 2(n-1) bit
  - Risultato con segno su 2n-1 bit

Segno di x	Segno di y	X_{n-1}	Y_{n-1}	Segno di Q	Q_{2n-1}
+	+	0	0	+	0
+	-	0	1	-	1
-	+	1	0	-	1
-	-	1	1	+	0

Moltiplicazione in complemento a 2

- In complemento a 2 a differenza della codifica naturale non è possibile aggiungere 0 a sinistra di un numero
  - Significherebbe cambiare di segno se il numero è negativo
- Si usa il procedimento di «estensione del segno»
  - I bit aggiunti a sinistra devono essere uguali al bit più significativo

				x_3	x_2	x_1	x_0	*
				y_3	y_2	y_1	y_0	=
	$\mathbf{x_3y_0}$	$\mathbf{x_3y_0}$	$\mathbf{x_3y_0}$	x_3y_0	x_2y_0	x_1y_0	x_0y_0	(P_0)
	$\mathbf{x_3y_1}$	$\mathbf{x_3y_1}$	x_3y_1	x_2y_1	x_1y_1	x_0y_1		(P_1)
	$\mathbf{x_3y_2}$	x_3y_2	x_2y_2	x_1y_2	x_0y_2			(P_2)
	x_3y_3	x_2y_3	x_1y_3	x_0y_3				(P_3)
q_7	q_6	q_5	q_4	q_3	q_2	q_1	q_0	(Q)

Algoritmo di Booth

- Questo algoritmo organizza la somma in modo da aumentare le prestazioni
- Si basa sulla seguente idea

$$A = 00011110 = 00100000 - 00000010$$

$$b \times a = b \times 00011110$$

$$= b \times (00100000 - 00000010)$$

$$= b \times a_1 - b \times a_2$$

- In generale $a = a_1 - a_2 + a_3 - a_4 + \ldots$

- Il numero delle coppie è uguale al numero di sequenze di 1 consecutivi

0	1	1	1	1	0

Consideriamo:

- un moltiplicatore su 5 bit in compl. a 2,
- aggiungiamo uno 0 a destra
- Si raggruppano 2 bit alla volta sovrapponendo un bit

$$[\ y_4\ \ y_3\ \ y_2\ \ y_1\ \ y_0\ \ 0\ ]$$

$$[y_4\ \ y_3]\ [y_3\ \ y_2]\ [y_2\ \ y_1]\ [y_1\ \ y_0]\ [y_0\ \ 0\ ]$$

y_i	y_{i-1}	$y_{i-1} - y_i$	Operazione
0	0	0	Nessuna
0	1	1	Somma di b
1	0	-1	Sottrazione di b
1	1	0	Nessuna

Bit corrente	Bit a destra	Spiegazione	Esempio
1	0	Inizio di una sequenza di 1	$0\ 0\ 0\ 0\ 1\ 1\ 1\ \boxed{1\ 0}\ 0\ 0_2$
1	1	Interno di una sequenza di 1	$0\ 0\ 0\ 0\ 1\ \boxed{1\ 1}\ 1\ 0\ 0\ 0_2$
0	0	Interno di una sequenza di 0	$0\ \boxed{0\ 0}\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0_2$
0	1	Fine di una sequenza di 1	$0\ 0\ 0\ \boxed{0\ 1}\ 1\ 1\ 1\ 0\ 0\ 0_2$

Algoritmo di Booth - Dimostrazione

- Sia il prodotto : $a_{31}\, a_{30} \ldots a_1\, a_0$

Alg. Booth:

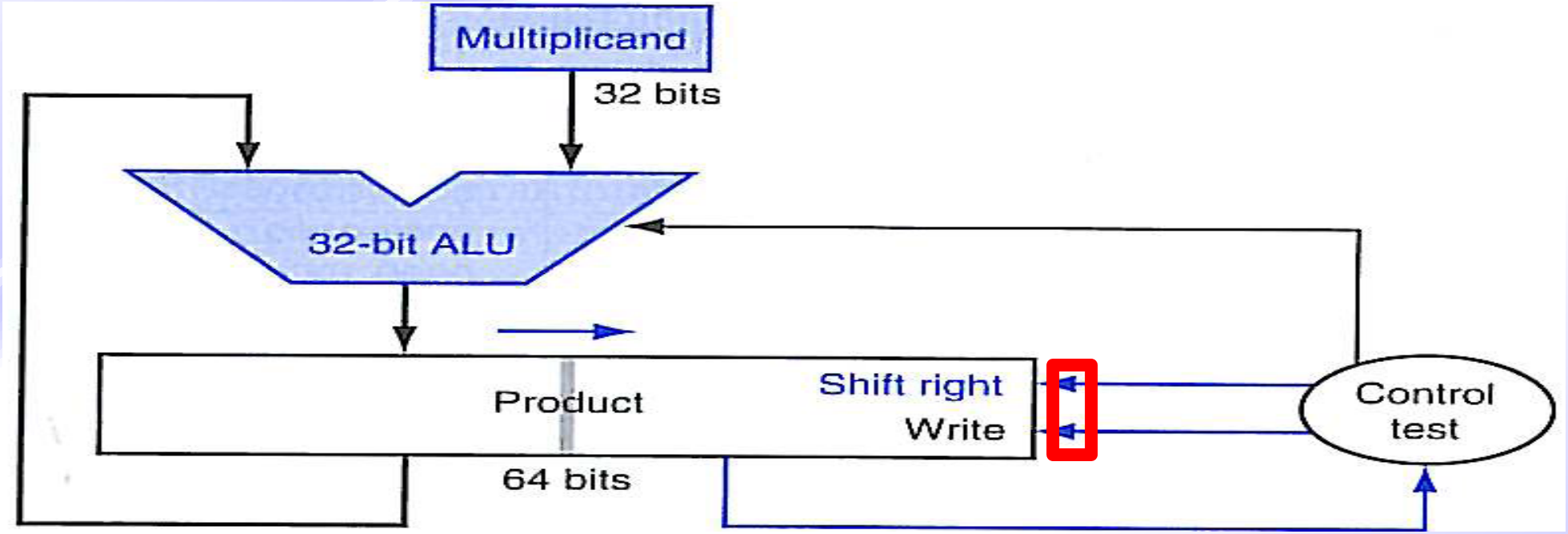
$$(a_{-1} - a_0) \times b \times 2^0 +$$
$$(a_0 - a_1) \times b \times 2^1 +$$
$$\vdots$$
$$(a_{29} - a_{30}) \times b \times 2^{30} +$$
$$(a_{30} - a_{31}) \times b \times 2^{31}$$

$a_{i-1} - a_i$	Operazione
0	Nessuna
1	Somma di b
-1	Sottrazione di b

- essendo $-a_i \times 2^i + a_i \times 2^{i+1} = a_i \times 2^i$
- diviene

$$b \times \left(\left(a_{31} \times -2^{31}\right) + \left(a_{30} \times 2^{30}\right) + \cdots + \left(a_1 \times 2^1\right) + \left(a_0 \times 2^0\right) \right) = b \times a$$

Circuiteria per Booth



- La stessa circuiteria per la moltiplicazione fra naturale con la aggiunta di un bit a destra di Prodotto (product-1) inizializzato a 0, testabile dal controllo, oltre al bit Product0

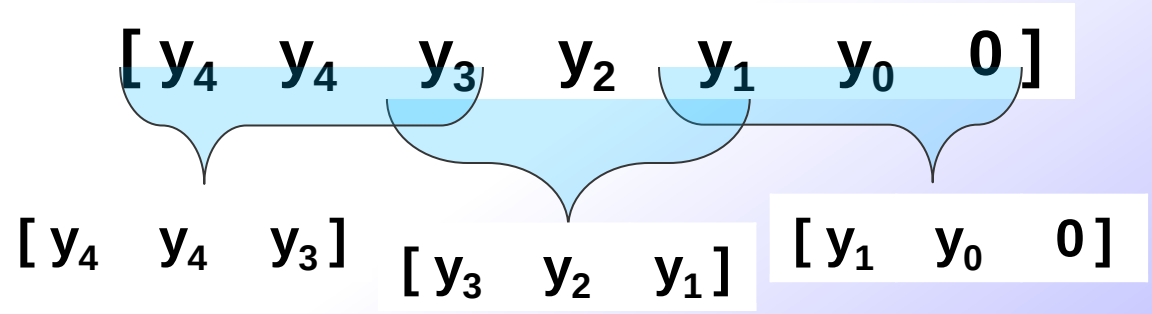
Algoritmo di Booth - esempio

- Consideriamo la moltiplicazione di:
  - Moltiplicando: $0010_{2,C2} = 2_{10}$
  - Moltiplicatore: $1101_{2,C2} = -3_{10}$

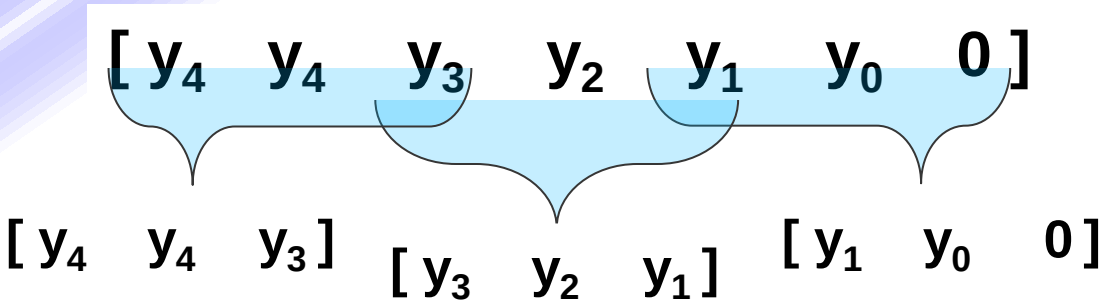
Passo	Fase	Moltiplicando	Prodotto
0	Preparazione dei valori iniziali	0010	0000 1101 0
1	1c: 10 => Prodotto = Prodotto – Moltiplicando	0010	1110 1101 0
	2 : scorrimento a destra del prodotto	0010	1111 0110 1
2	1b: 01 => Prodotto = Prodotto + Moltiplicando	0010	0001 0110 1
	2 : scorrimento a destra del prodotto	0010	0000 1011 0
3	1c: 10 => Prodotto = Prodotto – Moltiplicando	0010	1110 1011 0
	2 : scorrimento a destra del prodotto	0010	1111 0101 1
4	1d: 11 => Nessuna Operazione	0010	1111 0101 1
	2 : scorrimento a destra del prodotto	0010	1111 1010 1

Moltiplicazione con codifica di Booth-2

- Riduce il numero di prodotti parziali in operazioni di moltiplicazione con operandi minori di 0
- Organizza la somma in modo da aumentare le prestazioni
- Consideriamo:
  - un moltiplicatore di 5 bit in compl. a 2,
  - aggiungiamo uno 0 a destra
  - Si raggruppano 3 bit alla volta sovrapponendo un bit ed estendendo il segno per completare l'ultimo gruppo



Calcolo del coefficiente di Booth-2



- Per ogni raggruppamento si calcola un coefficiente:

$$f(y_{n+1}, y_n, y_{n-1}) = -2 \cdot y_{n+1} + y_n + y_{n-1}$$

- E lo si pesa con 2^n ottenendo una parola in codifica di booth-2

$$Y_{b2} = f(y_4, y_4, y_3)2^4 + f(y_3, y_2, y_1)2^2 + f(y_1, y_0, 0)2^0$$

$$Y_{2,Booth2} = [f(y_4, y_4, y_3), f(y_3, y_2, y_1), f(y_1, y_0, 0)]$$

Equivalenza fra Codifica di Booth-2 e C2

- Dimostriamo l'equivalenza fra la parola di codice di partenza e quella in codice di Booth-2

$$Y_{2,Booth2} = [f(y_4, y_4, y_3), f(y_3, y_2, y_1), f(y_1, y_0, 0)]$$

$$\begin{aligned} Y_{b2} &= (-2y_4 + y_4 + y_3)2^4 + (-2y_3 + y_2 + y_1)2^2 + (-2y_1 + y_0 + 0)2^0 \\ &= -y_4 2^5 + y_4 2^4 + y_3 2^4 - y_3 2^3 + y_2 2^2 + y_1 2^2 - y_1 2^0 + y_0 2^0 = \\ &= y_4\left(-2^5 + 2^4\right) + y_3\left(2^4 - 2^3\right) + y_2 2^2 + y_1\left(2^2 - 2^0\right) + y_0 2^0 = \\ &= -y_4 2^4 + y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0 \end{aligned}$$

$$Y_{2,C2} = [y_4 y_3 y_2 y_1 y_0]$$

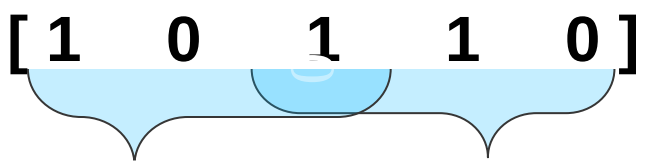
- Utilizzando questa codifica la moltiplicazione fra parole 5 bit richiede solamente 3 prodotti parziali:
- Inoltre i coefficienti della codifica di Booth-2 può assumere solamente valori pari a 0, ±1, ±2.

Y_{n+1}	y_n	Y_{n-1}	$-2*y_{n+1}+y_n+y_{n-1}$
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

- La moltiplicazione dei prodotti parziali diventa estremamente semplice
  - Moltiplicare per due equivale ad uno spostamento verso sinistra

Esempio di moltiplicazione usando Booth-2

- Esempio: si consideri la moltiplicazione X * Y:
- $X = -4_{10} = 1100_{2,C2}$
- $Y = -5_{10} = 1011_{2,C2}$
- Eseguiamo i raggruppamenti nel moltiplicatore:



$$Y_{2,Booth2} = [-1, -1]$$

- Il Prodotto X*Y è quindi la somma dei prodotti parziali:

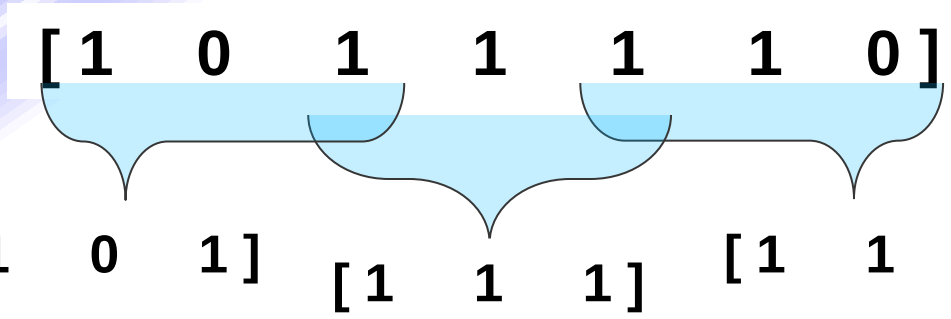
$$-4_{10} \times -1 \cdot 2^2 + -4_{10} \times -1 \cdot 2^0$$

0	0	0	0	1	0	0	+
0	0	1	0	0			=
0	0	1	0	1	0	0	

Altro esempio

$$X = 18_{10} = 010010_{2,C2} \quad , \quad Y = -17_{10} = 101111_{2,C2}$$

- Eseguiamo i raggruppamenti nel moltiplicatore:



$$Y_{2,Booth2} = [-1, 0, -1]$$

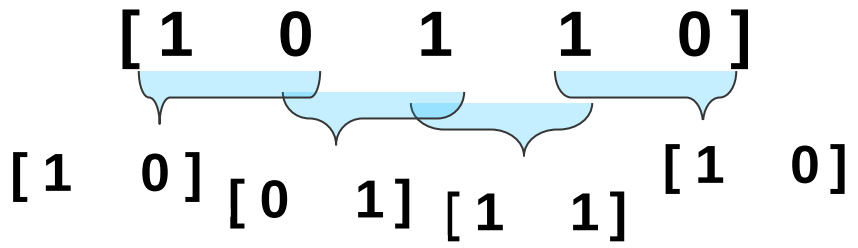
- Il Prodotto X*Y è quindi la somma di tre prodotti parziali:

$$18_{10} \times -1 \cdot 2^4 + 18_{10} \times 0 \cdot 2^2 + 18_{10} \times -1 \cdot 2^0$$

1	1	1	1	1	1	0	1	1	1	0	+
0	0	0	0	0	0	0	0	0			+
1	1	0	1	1	1	0					=
1	1	0	1	1	0	0	1	1	1	0	

Codifica di Booth-1

- Al metodo di codifica di Booth-2 se ne aggiungono altri che codificano gruppi di bit di dimensione k
- Fra questi il più semplice è proprio l'algoritmo di Booth, detto Booth-1, che usa gruppi di 2 bit
- Es: -4 * -5



$$f(y_n, y_{n-1}) = -y_n + y_{n-1}$$

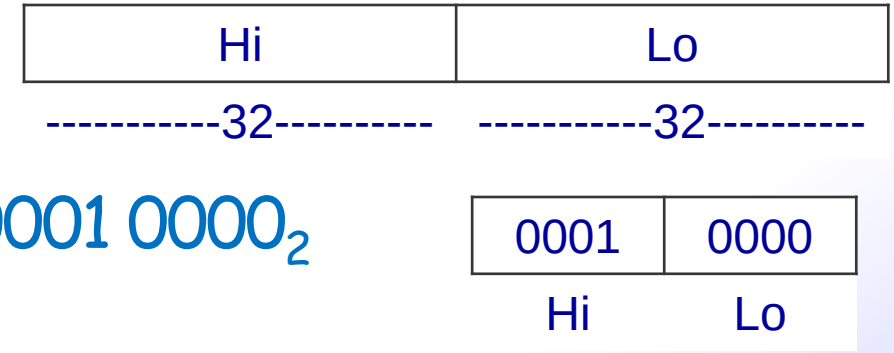
$$Y_{2,Booth1} = [-1, 1, 0, -1]$$

$$-4_{10} \times -1 \cdot 2^3 +$$
$$-4_{10} \times +1 \cdot 2^2 +$$
$$-4_{10} \times \ \ 0 \cdot 2^1 +$$
$$-4_{10} \times -1 \cdot 2^0$$

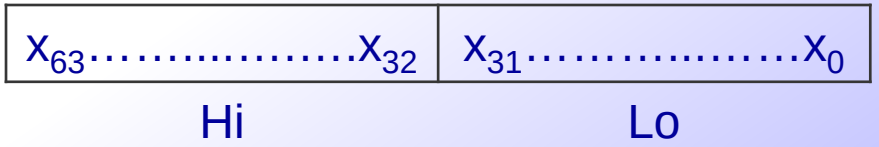
0	0	0	0	1	0	0	+
0	0	0	0	0	0		+
1	1	1	0	0			+
0	1	0	0				=
0	0	1	0	1	0	0	

Overflow nella moltiplicazione

- MULTU (multiply unsigned) – fra naturali
  - Se Hi ≠ 0…0 allora c'è overflow
  - Esempio su 4 bit:
  - $4_{10} \times 4_{10} = 16_{10} = 0100_2 \times 0100_2 = 0001\ 0000_2$
  - Hi ≠ 0…0 => overflow



- MULT (multiply signed) – fra interi
  - Se Hi ≠ x31 allora c'è overflow
  - Esempio su 4 bit:
  - $-3_{10} \times 3_{10} = -9_{10} = 1101_{2,C2} \times 0011_{2,C2} = 1111\ 0111_{2,C2}$
  - Hi ≠ Lo => overflow



La divisione

- $897123_{10} : 810_{10}$ Dividendo : Divisore
- Dividendo = Quoziente x Divisore + Resto

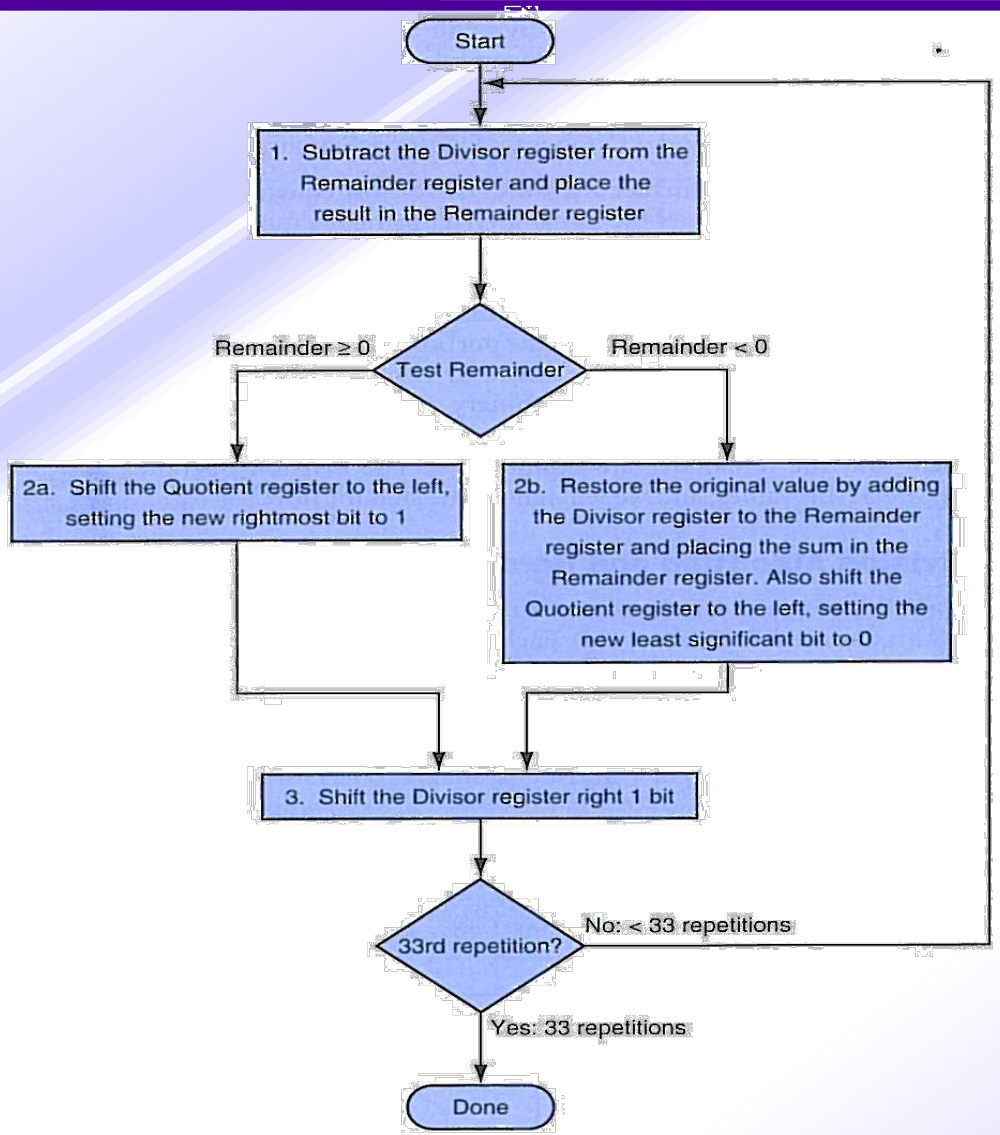
- Si usa l'algoritmo appreso alle scuole elementari ricavando il quoziente una cifra alla volta

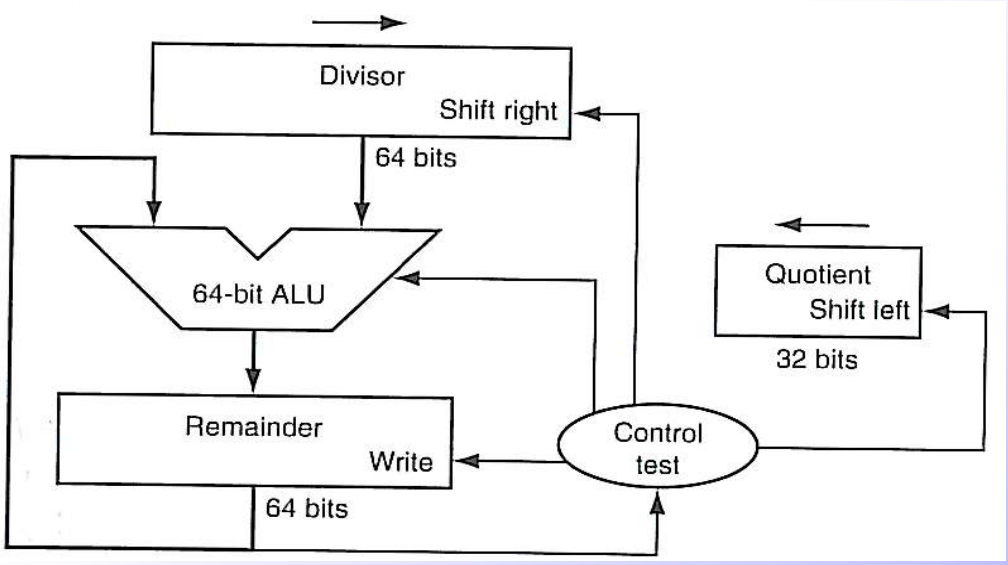
```
8 9 7 1 2 3 -
8 1 0
-------
  8 7         1
  8 7 1 -
  8 1 0
-------
    6 1       1
    6 1 2     0
    6 1 2 3
    5 6 7 0 -
-------
      4 5 3   7
```

...analogo...

```
1 0 0 1 0 1 0
1 0 0 0
-------
      1       1
      1 0     0
      1 0 1   0
      1 0 1 0
      1 0 0 0
-------
        1 0   1
```

Algoritmo ed hardware per la divisione

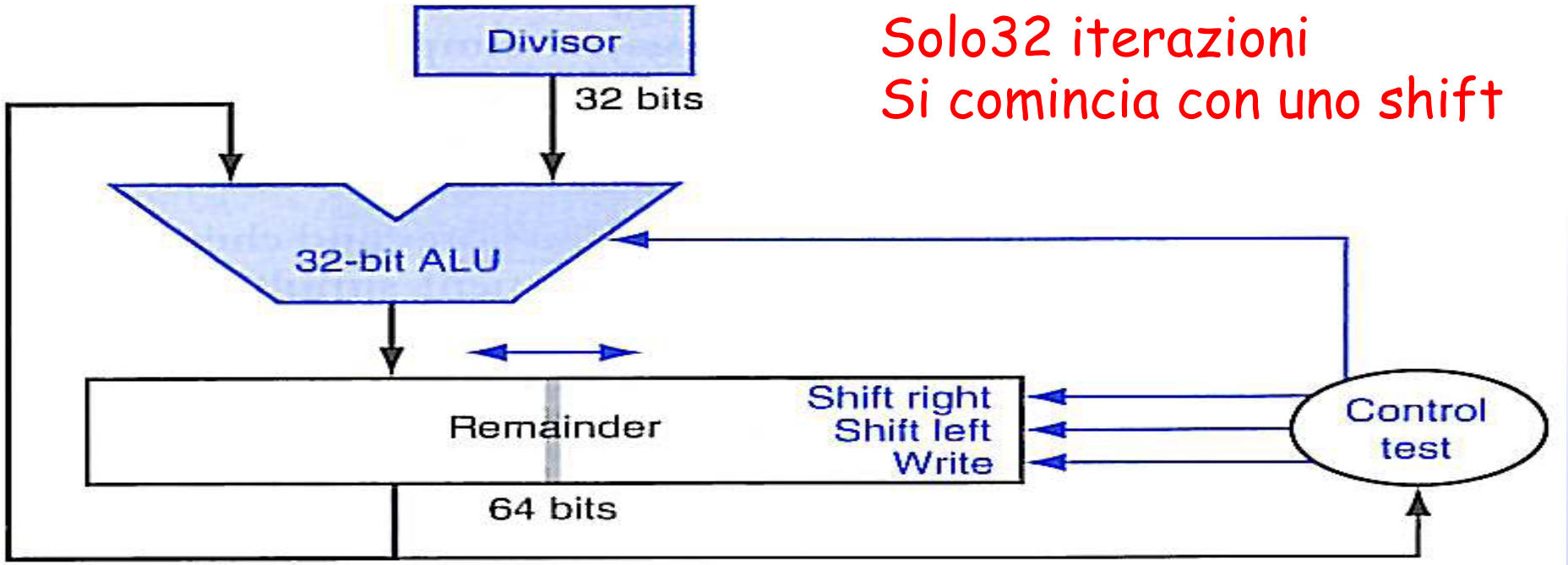




Algoritmo di divisione - Esempio

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	(1)110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	(1)111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	(1)111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	(0)000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	(0)000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Hardware per la divisione migliorato



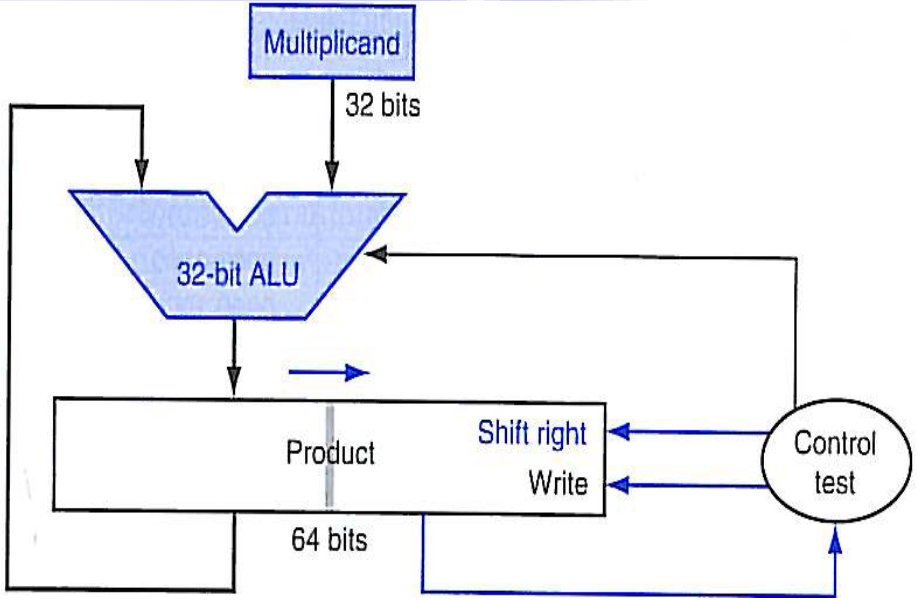
Solo32 iterazioni
Si comincia con uno shift

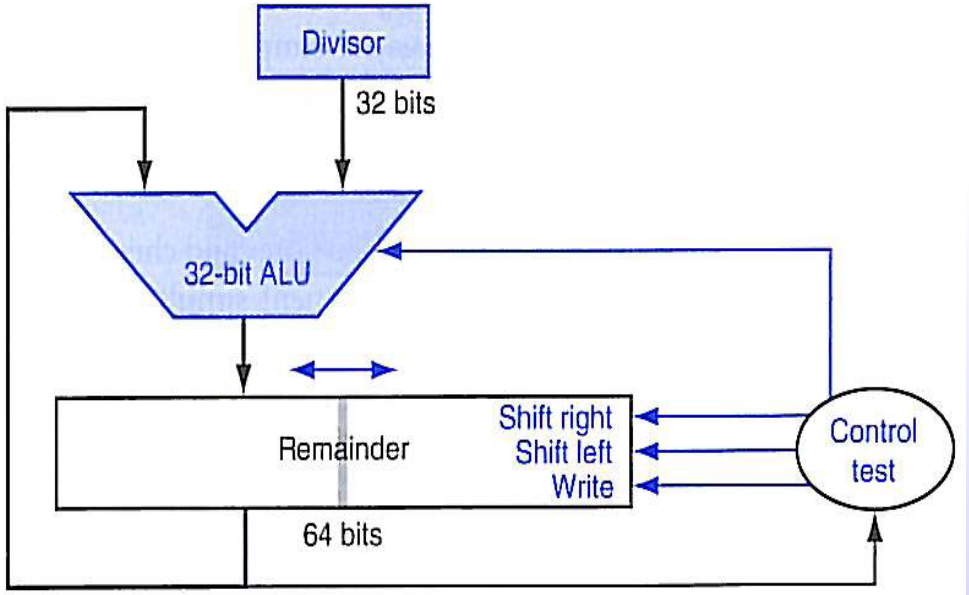
- Registro divisore e ALU a 32 bit - Registro resto a 64 bit
- Il dividendo viene inizialmente inserito nei 32 bit meno significativi del registro Resto.
- Alla fine i 32 bit meno significativi del registro Resto contengono il quoziente e i piu' significativi il resto

Esempio con hardware migliorato

Passo	Fase	Divisore	Resto
0	Preparazione dei valori iniziali	0010	0000 0111
	Scorrimento a sinistra del resto	0010	0000 1110
1	Resto = Resto - Divisore	0010	1110 1110
	Resto < 0 =>somma del div., scorr a sin. del resto, $Resto_0$ =0	0010	0001 1100
2	Resto = Resto - Divisore	0010	1111 1100
	Resto < 0 =>somma del div., scorr a sin. del resto, $Resto_0$ =0	0010	0011 1000
3	Resto = Resto - Divisore	0010	0001 1000
	Resto >= 0 => scorr a sin. del resto, $Resto_0$=1	0010	0011 0001
4	Resto = Resto - Divisore	0010	0001 0001
	Resto >= 0 => scorr a sin. del resto, $Resto_0$=1	0010	0010 0011
	Scorr. A destra della metà sin. di Resto	0010	0001 0011

Stesso hardware per prodotto e divisione





- Prodotto e divisione utilizzano lo stesso circuito formato da:
  - Alu a 32 bit per somme e sottrazioni
  - Un registro a 32 bit
  - Un registro a 64 bit con shift aritmetico a destra e sinistra
  - Una logica di controllo

Divisione di numeri interi relativi

1. Si memorizza il segno del divisore e del dividendo
2. Si nega il quoziente se i segni sono discordi, al termine dell'esecuzione
3. Per calcolare il segno del resto si ricorda che deve valere :
   - Dividendo=Quoziente x Divisore + resto
     (il resto ha lo stesso segno del Dividendo)

- Esempio:

```
-7: 2   q=-3  r=-1
-7:-2   q= 3  r=-1
 7:-2   q=-3  r= 1
 7: 2   q= 3  r= 1
```