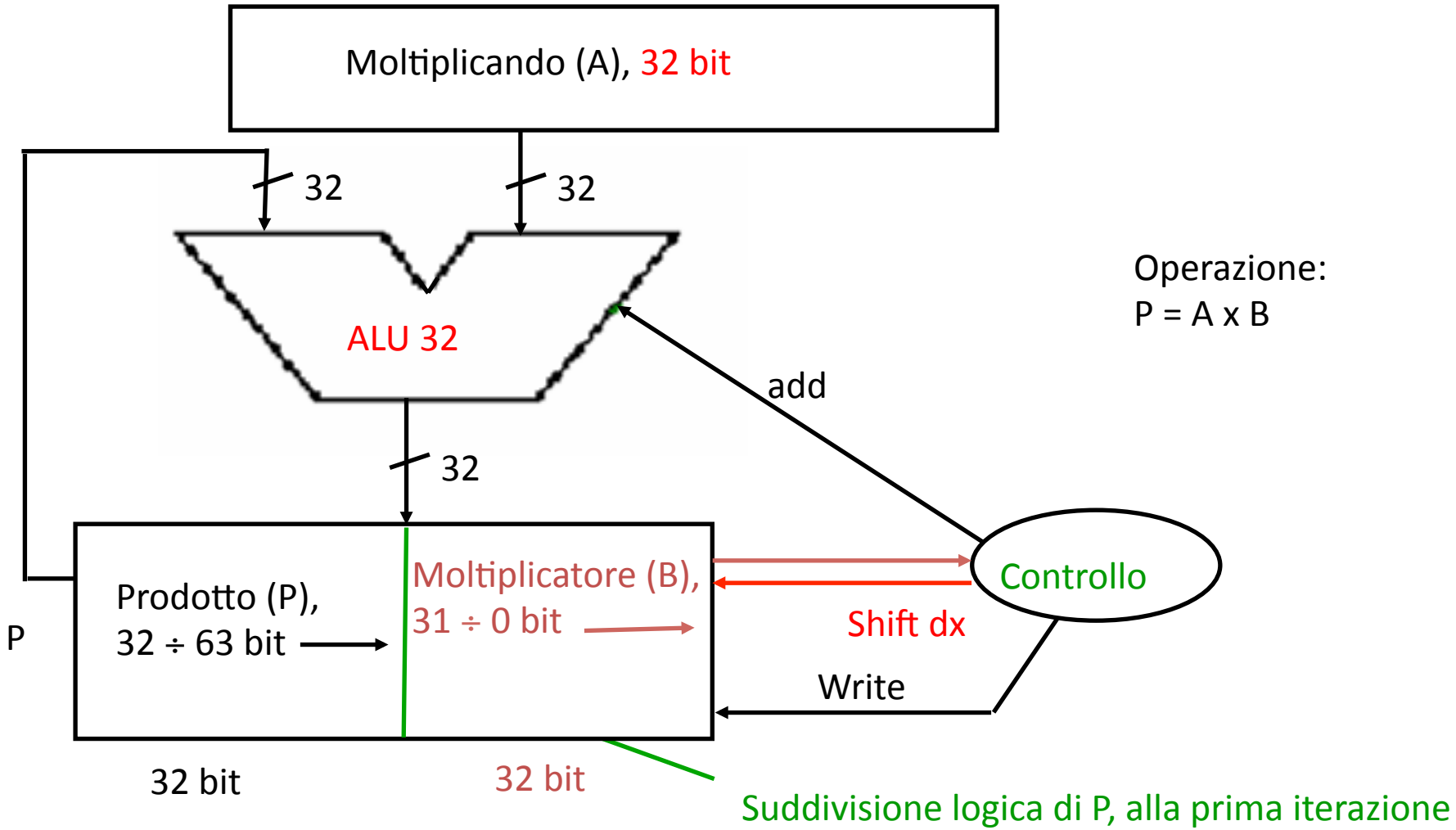
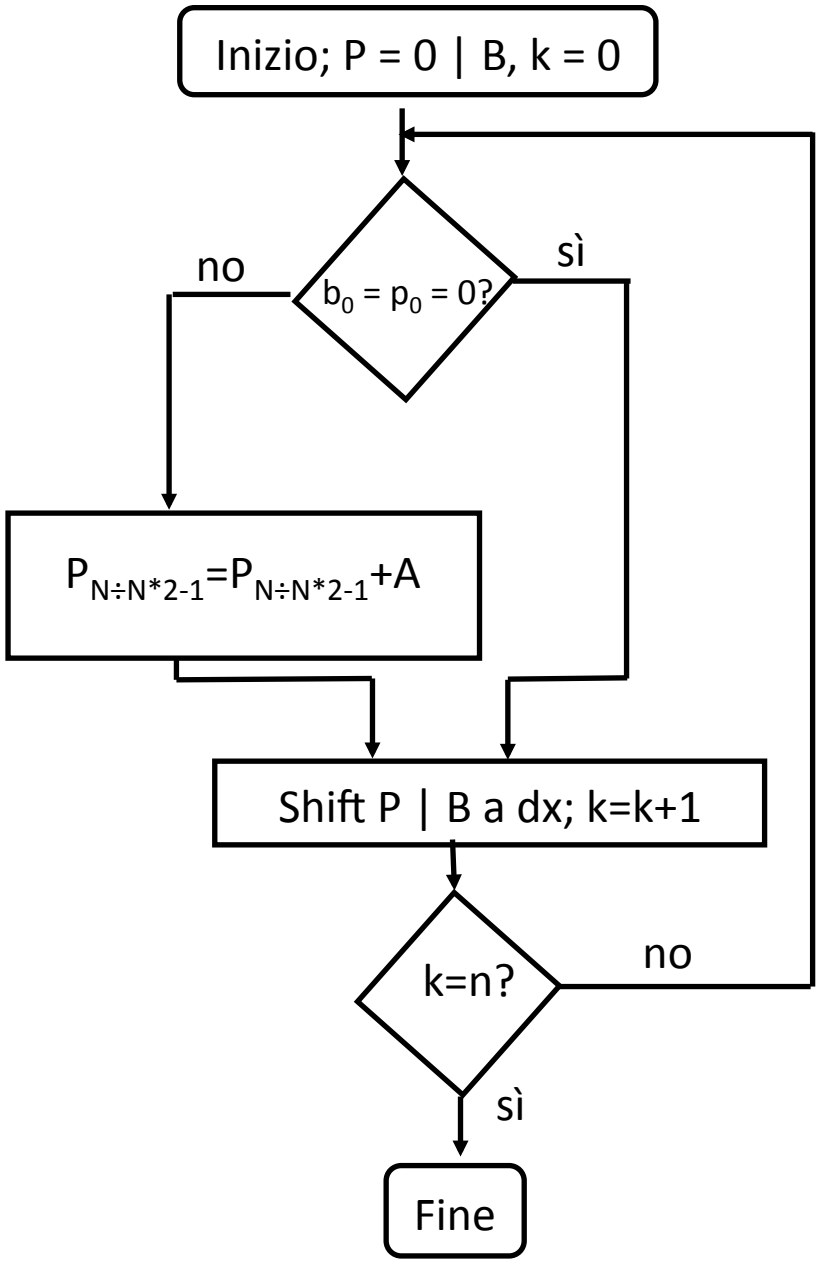
Es. 6
Moltiplicazione e divisione tra numeri interi (Firmware)

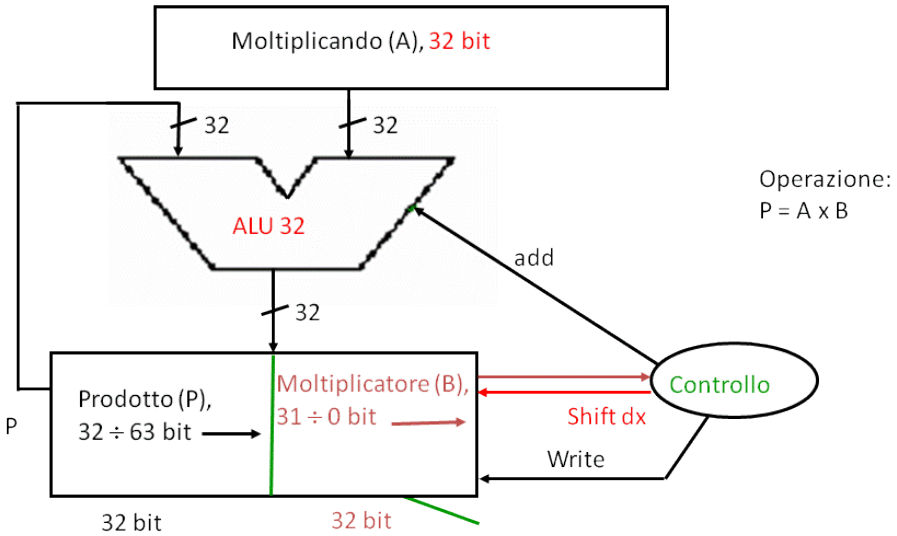
Circuito della moltiplicazione



Il moltiplicando è allineato sempre ai 32 bit più significativi del prodotto.
Ad ogni iterazione, il prodotto si allarga, il moltiplicatore si restringe.

Algoritmo della moltiplicazione





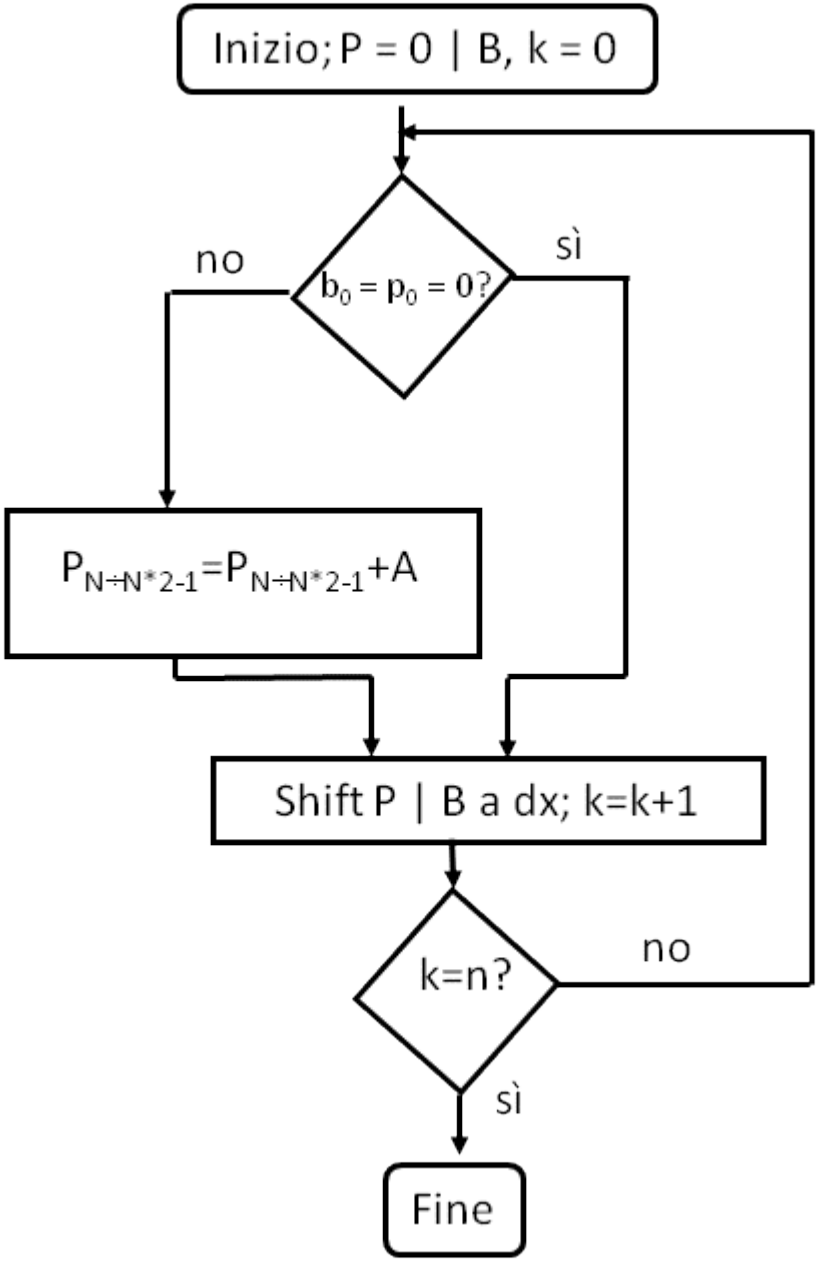
1010 x 0111

Es. 1

- Si effettui la moltiplicazione 1010 x 0111, evidenziando ad ogni passo il contenuto dei registri, le operazioni di somma e di shift. Si verifichi la correttezza del risultato finale. Si operi per semplicità su registri a 4 / 8 bit.

Sol. 1

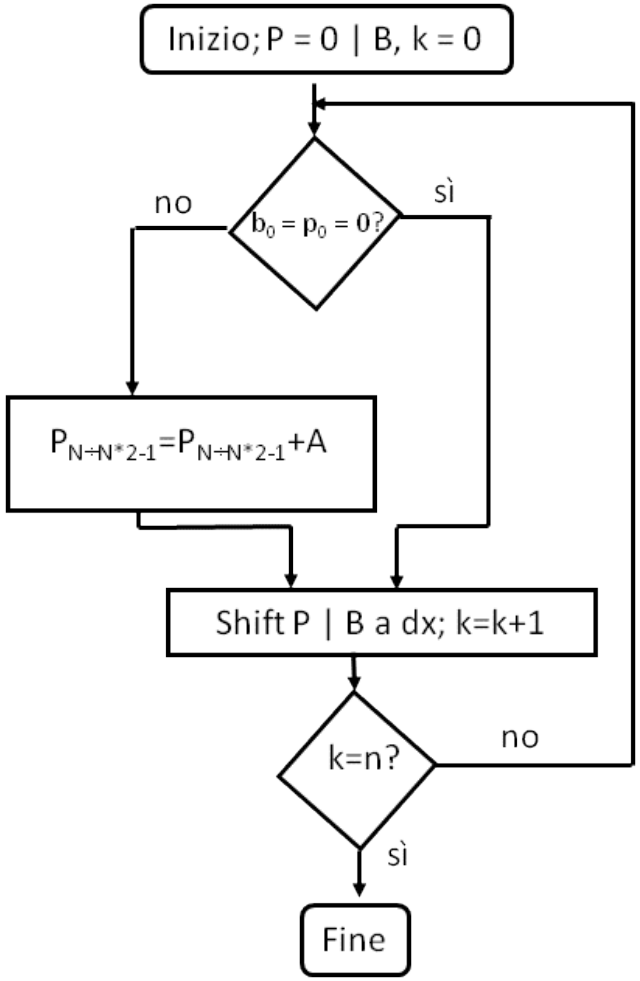
- P = A x B
- Dobbiamo effettuare al più 4 iterazioni su k per effettuare la moltiplicazione (ciascuna iterazione rende conto della moltiplicazione per un bit di B).
- Ciascuna iterazione richiede la somma tra P ed A, lo shift a dx di P => due righe di tabella per ogni iterazione.
- E' poi necessaria una riga iniziale per l'inizializzazione dei registri.
- Il numero di colonne della tabella è pari al numero di registri => 2, uno per A (a 4 bit), uno per P | B (a 8 bit).



Sol. 1

A =1010, B = 0111, P = A x B = ???

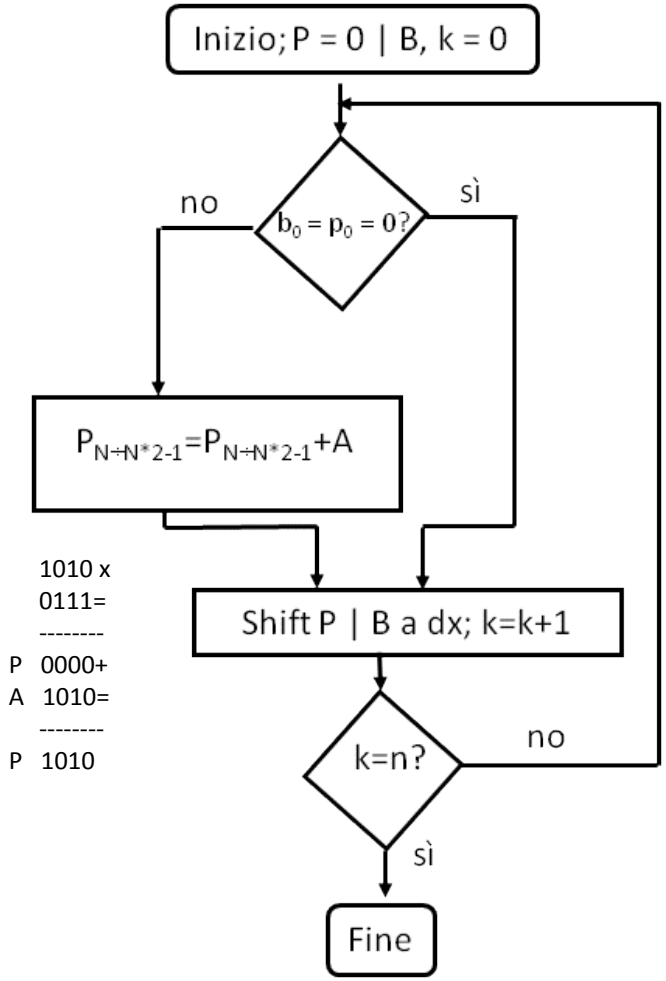
	A	P | B
Inizializzazione (k=0)	1010	0000 0111



Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111



1010 x
0111=

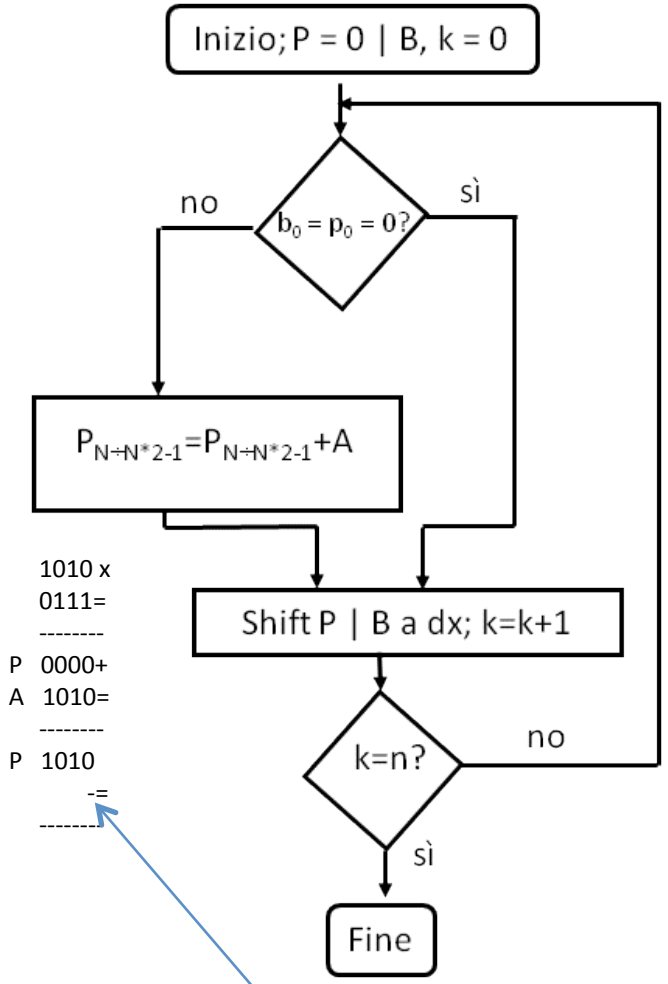
P 0000+
A 1010=

P 1010

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111
Shift a dx di P | B, k=0	1010	01010 011

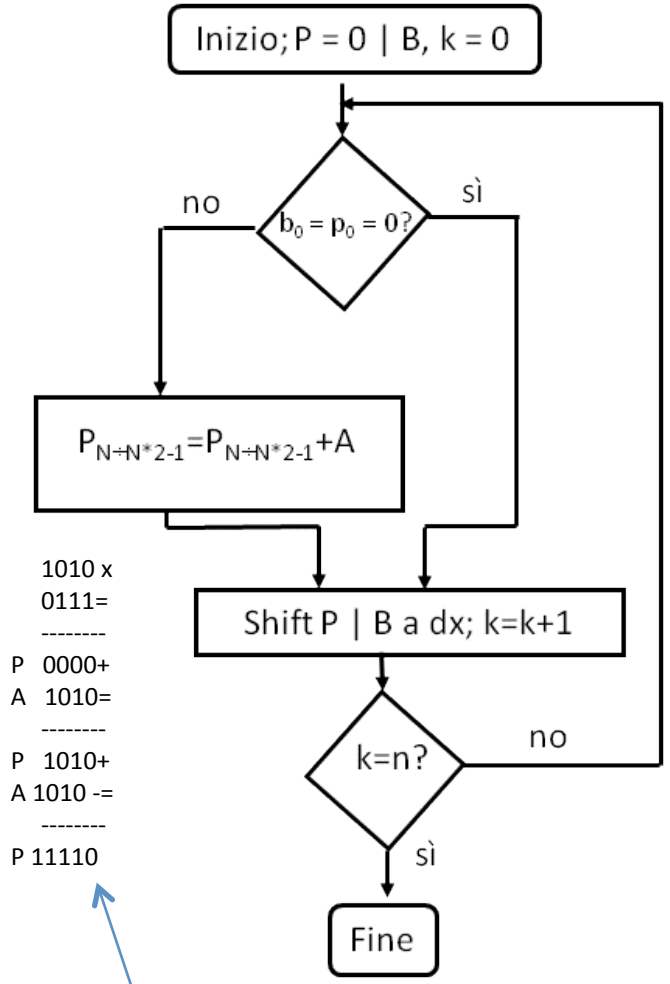


NB Lo shift a dx di P è equivalente ad uno shift a sx di A!

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111
Shift a dx di P | B, k=0	1010	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	11110 011
	1010	
	1010	
	1010	
	1010	
	1010	

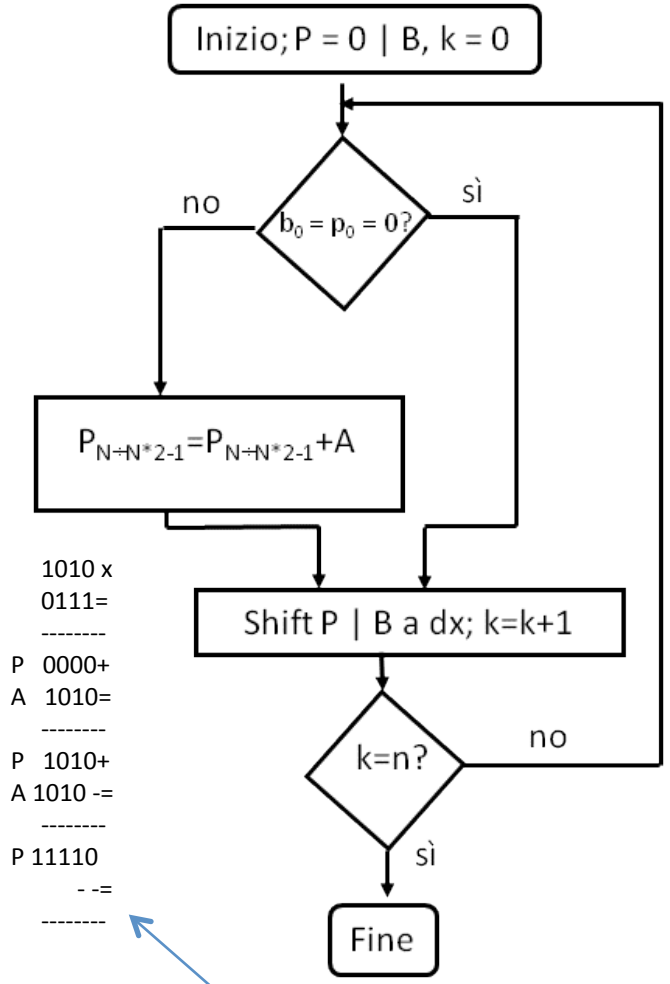


NB L'operazione di somma avviene sui primi 4 bit a sx – l'ultimo bit a dx è stabile (gli viene sommato uno zero!)

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111
Shift a dx di P | B, k=0	1010	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	11110 011
Shift a dx di P | B, k=1	1010	011110 01
	1010	
	1010	
	1010	
	1010	

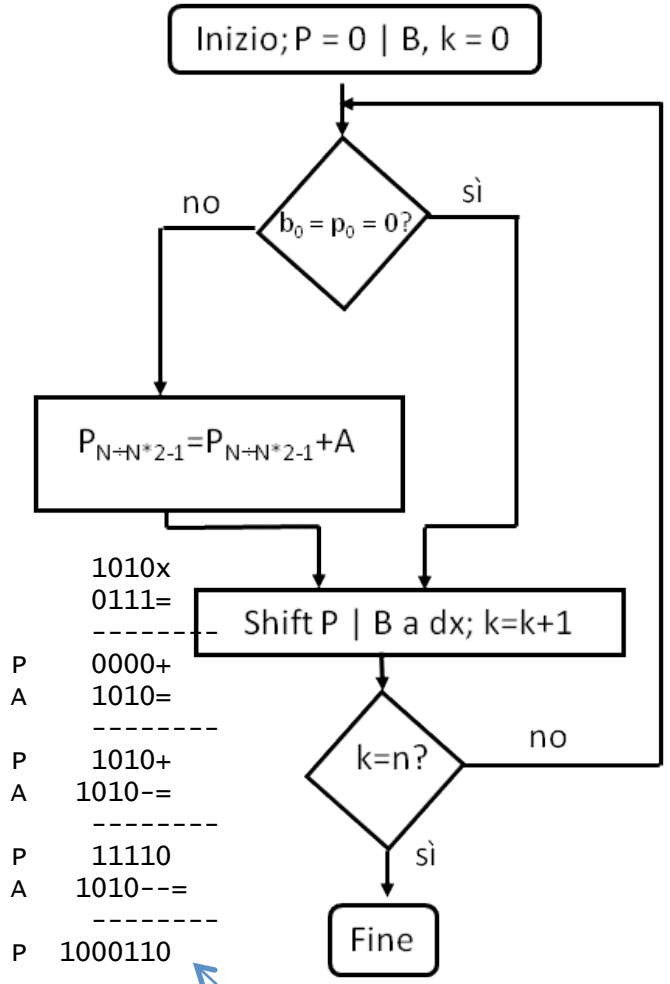


NB Lo shift a dx di P è equivalente ad uno shift a sx di A!

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111
Shift a dx di P | B, k=0	1010	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	11110 011
Shift a dx di P | B, k=1	1010	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	000110 01
	1010	
	1010	
	1010	

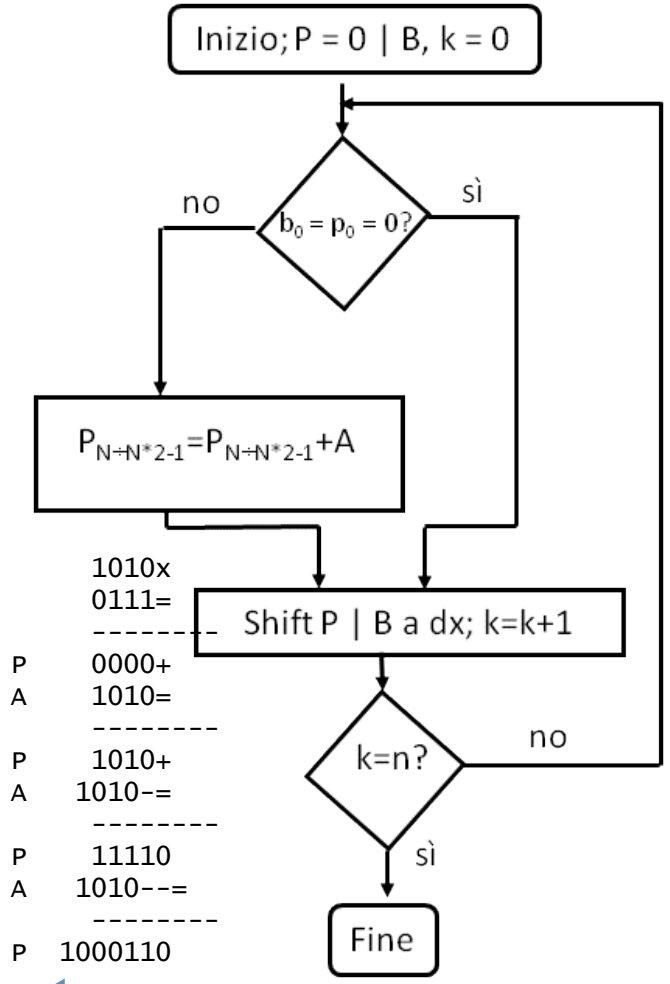


NB Gli ultimi due bit a dx sono stabili, la somma è effettiva solo sui 4 bit a sx

Sol. 1

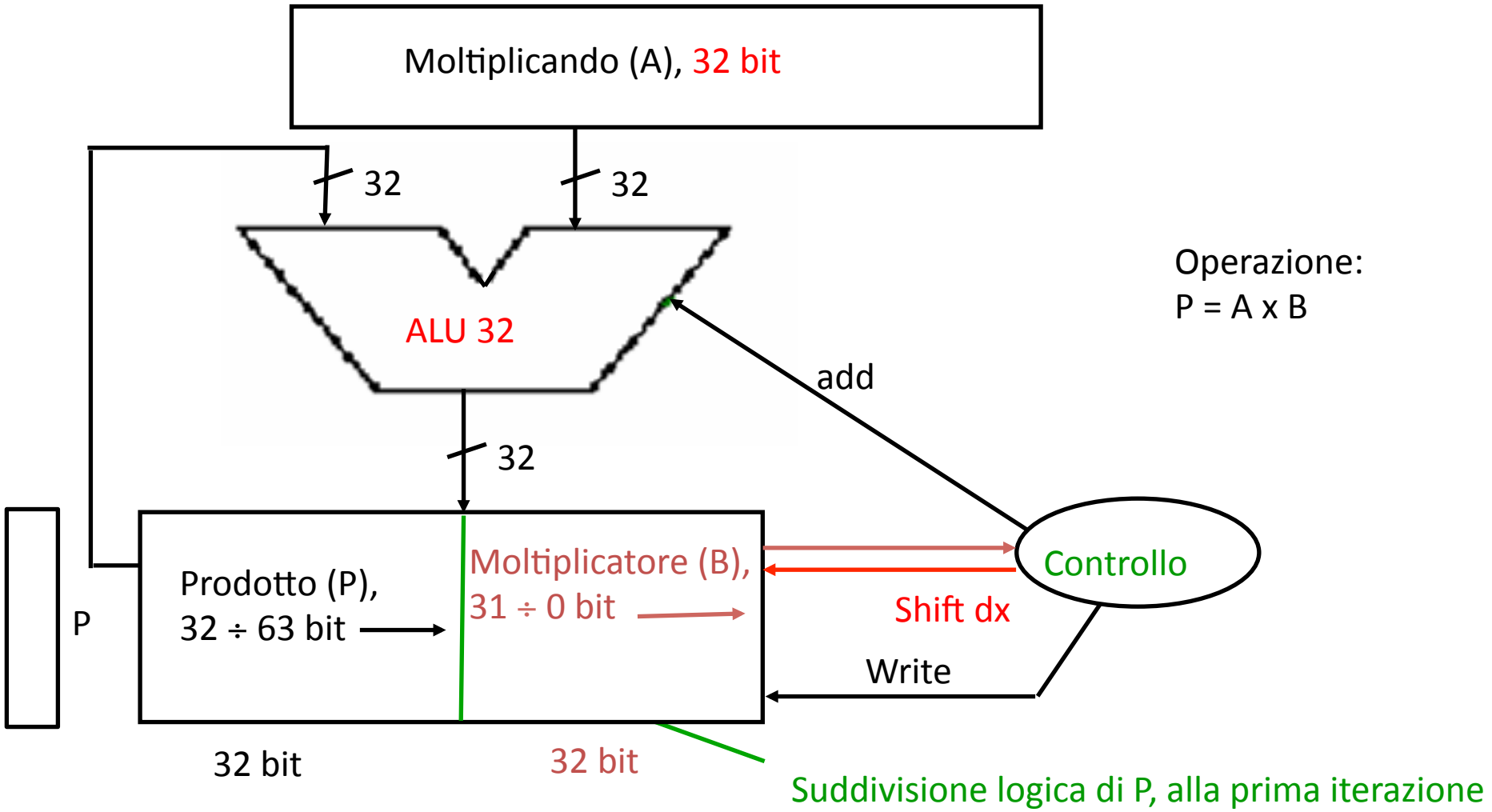
A =1010, B = 0111, P = A x B = ???

	A	P | B
Inizializzazione (k=0)	1010	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	1010 0111
Shift a dx di P | B, k=0	1010	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	11110 011
Shift a dx di P | B, k=1	1010	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	000110 01
	1010	
	1010	
	1010	



Attenzione! Il bit che si è generato con il riporto non ci sta nel registro P|B fino a quando non effettuiamo lo shift!!! La somma a 4 bit di 1010 + 111 ha generato un overflow (a 4 bit), dobbiamo aggiungere un bit a sx di P|B per tenerne conto...

Circuito della moltiplicazione (agg.)

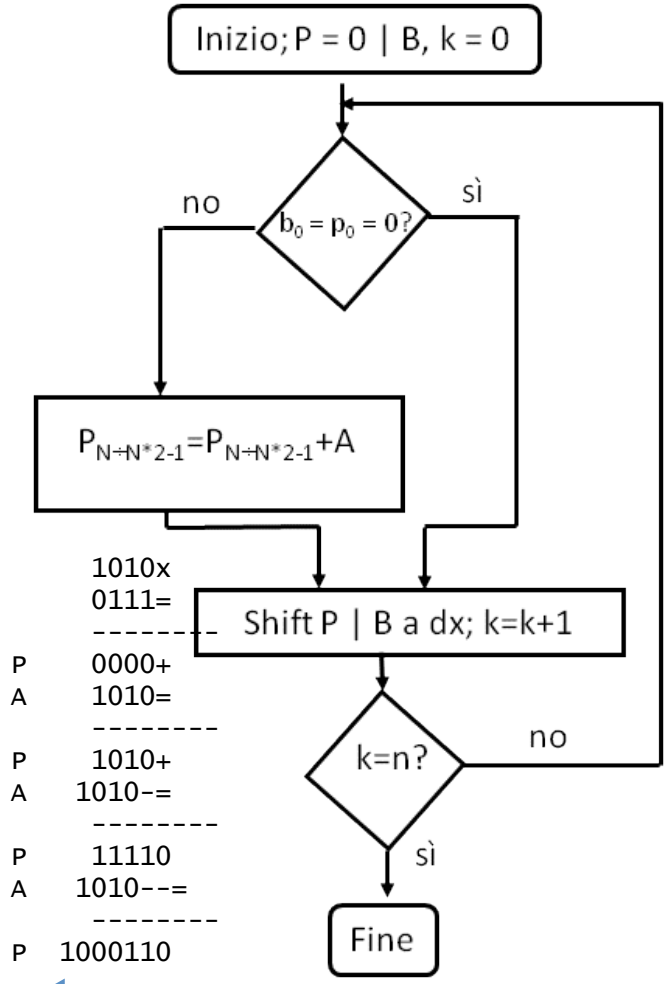


Bit per l'oveflow nelle somme parziali – va a riempire l'ultimo bit a sx di P|B
al momento dello shift verso dx di P|B!

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	O	P | B
Inizializzazione (k=0)	1010	0	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	0	1010 0111
Shift a dx di P | B, k=0	1010	0	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	0	11110 011
Shift a dx di P | B, k=1	1010	0	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	1	000110 01
	1010		
	1010		
	1010		

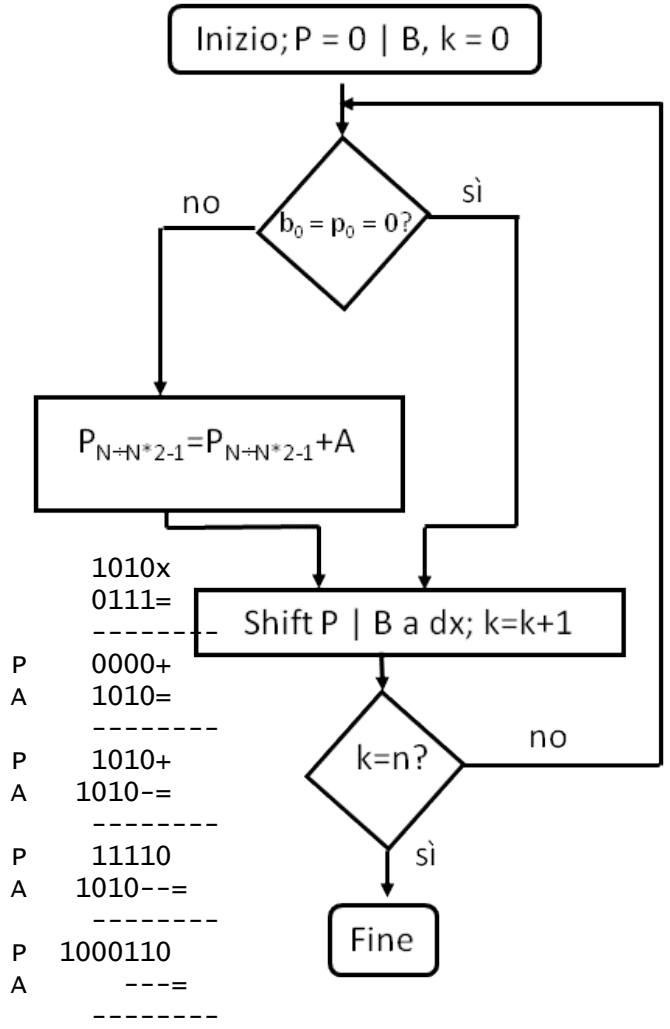


Metto il bit di riporto in O, dovrò tenerne conto al momento dello shift...
Per tutte le operazioni precedenti, il bit di riporto era pari a 0.

Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	O	P | B
Inizializzazione (k=0)	1010	0	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	0	1010 0111
Shift a dx di P | B, k=0	1010	0	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	0	11110 011
Shift a dx di P | B, k=1	1010	0	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	1	000110 01
Shift a dx di P | B, k=2	1010	0	1000110 0
	1010		
	1010		

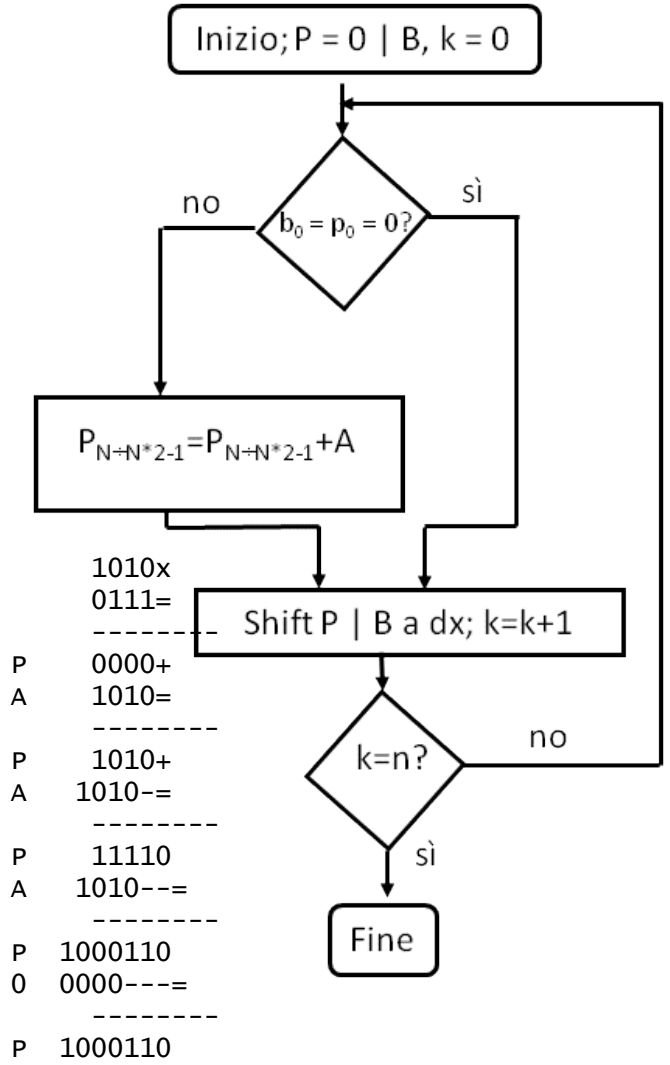


Nell'effettuare lo shift, ho inserito 1 invece che 0 nel bit più a sx.
Metto a 0 il valore di O per evitare problemi nei passi successivi...

Sol. 1

A =1010, B = 0111, P = A x B = ???

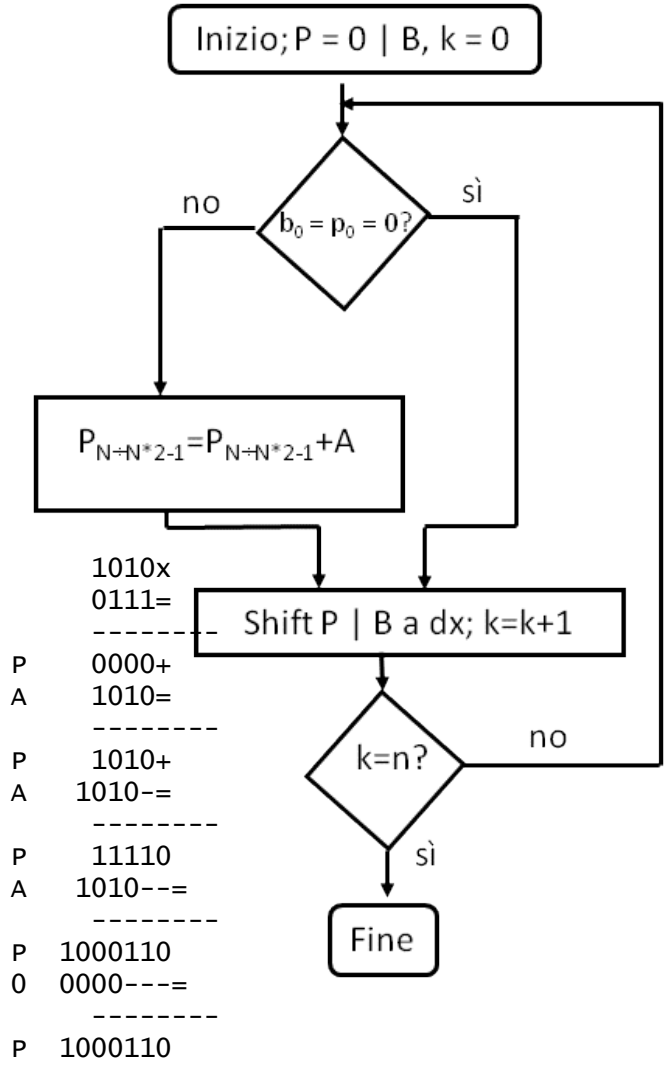
	A	O	P | B
Inizializzazione (k=0)	1010	0	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	0	1010 0111
Shift a dx di P | B, k=0	1010	0	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	0	11110 011
Shift a dx di P | B, k=1	1010	0	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	1	000110 01
Shift a dx di P | B, k=2	1010	0	1000110 0
b_0 = 0, no agg., k=2	1010	0	1000110 0
	1010		



Sol. 1

A =1010, B = 0111, P = A x B = ???

	A	O	P | B
Inizializzazione (k=0)	1010	0	0000 0111
b_0 = 1, aggiorno P = P + A, k=0	1010	0	1010 0111
Shift a dx di P | B, k=0	1010	0	01010 011
b_0 = 1, aggiorno P = P + A, k=1	1010	0	11110 011
Shift a dx di P | B, k=1	1010	0	011110 01
b_0 = 1, aggiorno P = P + A, k=2	1010	1	000110 01
Shift a dx di P | B, k=2	1010	0	1000110 0
b_0 = 0, no agg., k=2	1010	0	1000110 0
Shift a dx di P | B, k=3	1010	0	01000110



Se non avessi messo a 0 il valore di O, avrei portato in P un bit erroneamente a 1...

Sol. 1 - verifica

- A =1010, B = 0111, P = A x B = 01000110

- A = 2 + 8 = 10
- B = 1 + 2 + 4 = 7

- P = 2 + 4 + 64 = 70 = 10 x 7

QED

Es. 2

- Si effettui la moltiplicazione 1111 x 1111, evidenziando ad ogni passo il contenuto dei registri, le operazioni di somma e di shift. Si verifichi la correttezza del risultato finale. Si operi per semplicità su registri a 4 / 8 bit.

- Si verifica overflow?

Sol. 2

- Risultati numerici -> a voi...

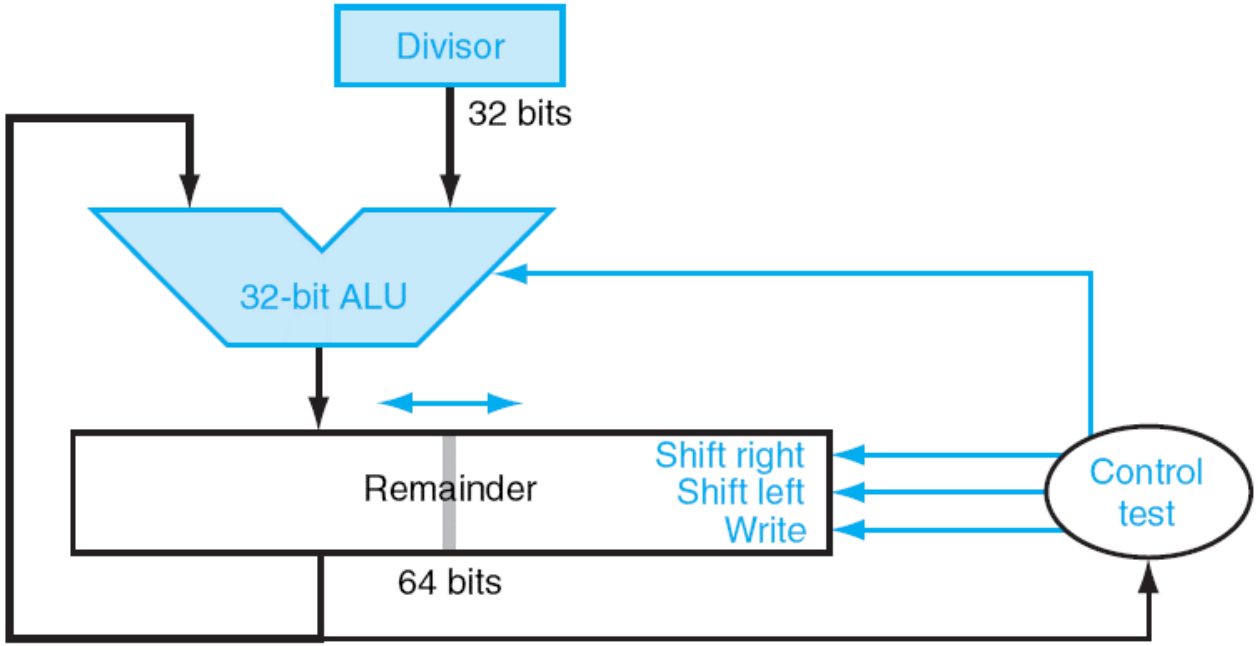
- Oveflow -> con le moltiplicazioni tra numeri interi, l'overflow non si verifica... Ad esempio:

  15x15 = 225 < 255

  Moltiplicando due numeri a 4 bit (15, massimo rappresentabile) otteniamo un numero ampiamente all'interno dell'intervallo rappresentabile con 8 bit!

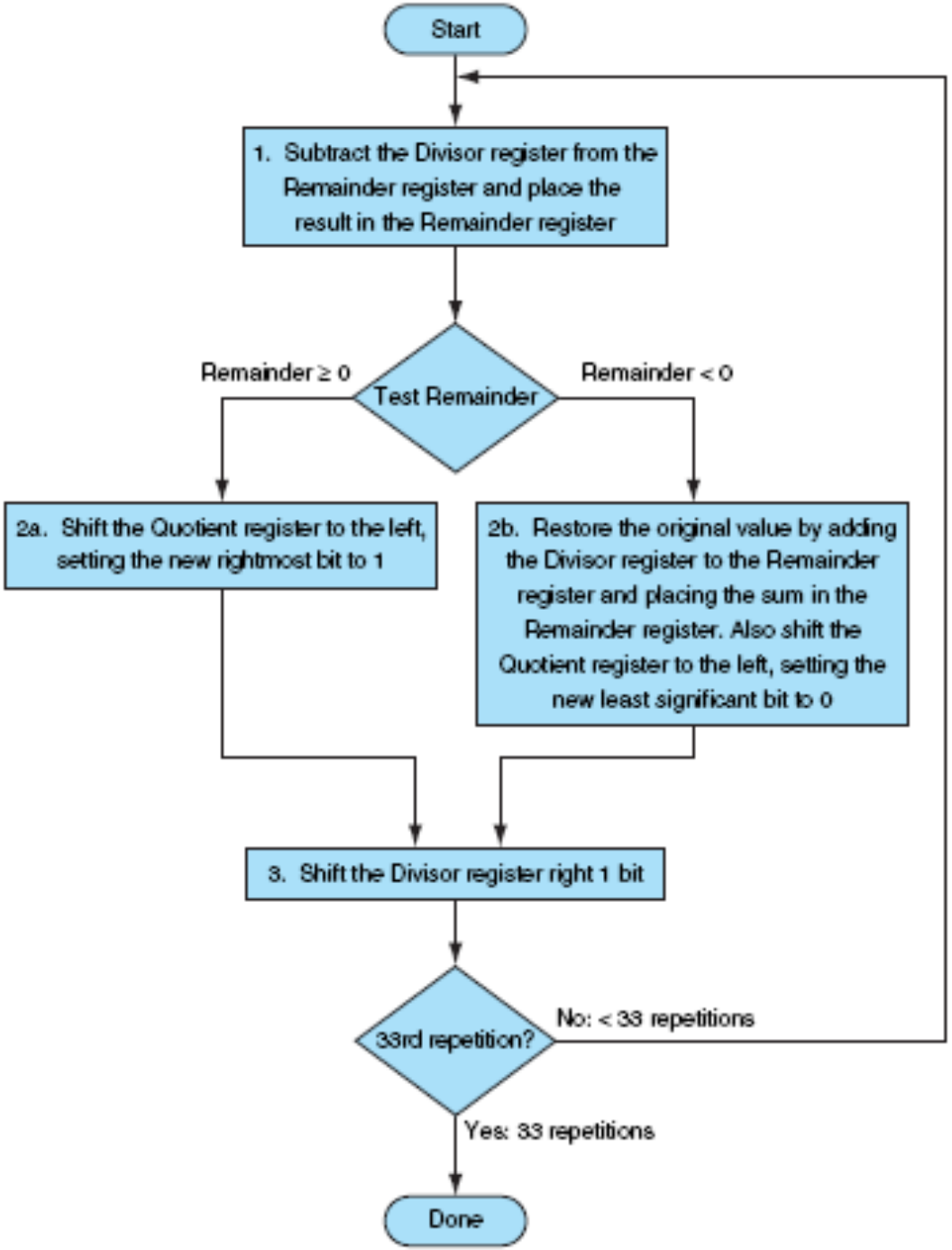
Circuito della divisione

Inizializzazione: Resto = 0 | Dividendo

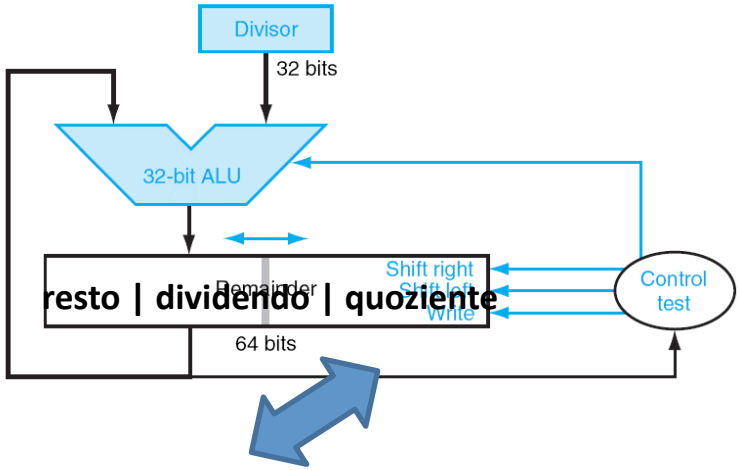


E' lo stesso circuito della moltiplicazione!

Algoritmo della divisione



Inizializzazione: Resto = 0 | Dividendo



104 : 5=020
1
10
04
4

Man mano si procede nel calcolo, il dividendo (104) può essere scartato, il resto ha dimensione costante, il quoziente cresce...

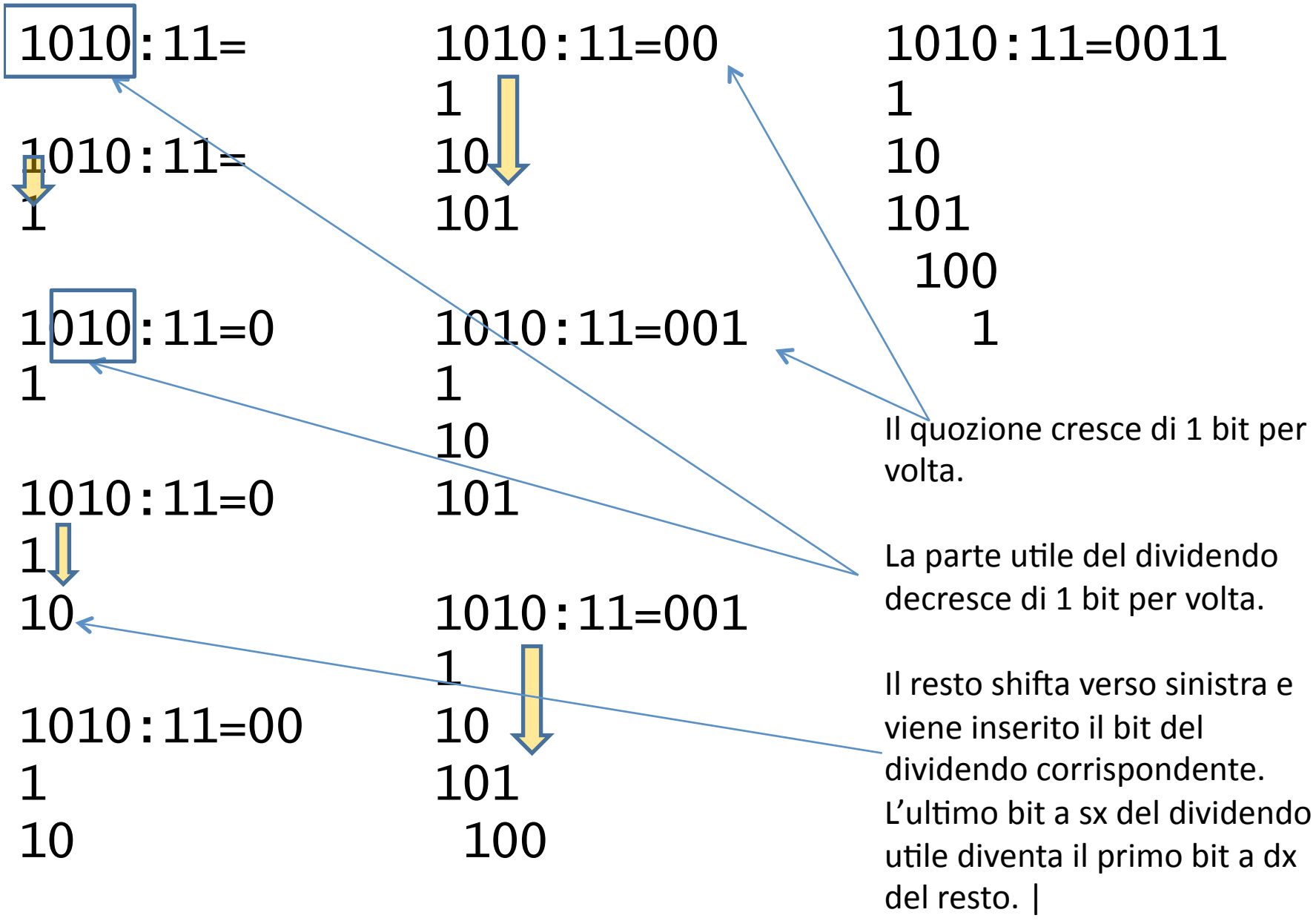
Algoritmo della divisione

```
1010:11=

1010:11=
1

1010:11=0
1

1010:11=0
1
10

1010:11=00
1
10
```

```
1010:11=00
1
10
101

1010:11=001
1
10
101

1010:11=001
1
10
101
 100
```

```
1010:11=0011
1
10
101
 100
   1

Test:
1010 = 10
11 = 3

10/3 = 3, r = 1.
```

Algoritmo della divisione



Es. 3

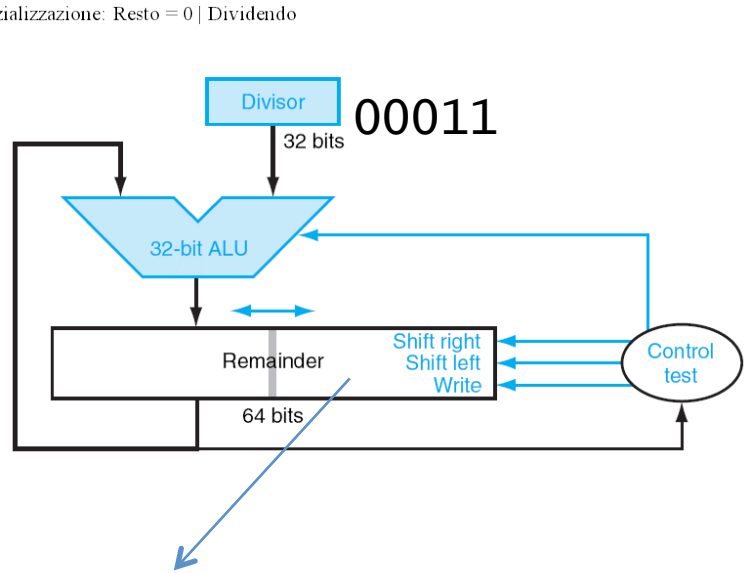
- Si effettui la divisione a 5 bit 10011:11, evidenziando il contenuto dei registri.
- Hint -> il registro del divisore è costante (dimensione 5 bit)
- Hint -> il registro che contiene il risultato alla fine dell'operazione (a 10 bit) contiene (resto parziale | parte utile del dividendo | quoziente).

Sol. 3

```
10011:00011=00110
1
10
100
  11
   01
    1

Test:
10011 = 19
11 = 3
110 = 6
19 / 3 = 6, r = 1.
```

10011:00011=00110
1
10
100
  11
   01
    1

Sol. 3



[resto parziale | parte utile del dividendo | quoziente]

Algoritmo (circuito ottimizzato):

0a. Inizializza il registro Divisor; inizializza il registro di uscita a (00...00 | dividendo).
0b. Shift a sx del registro di uscita (nuovo bit = 0).
1. Rem = Rem – Divisor.
2a. Se Rem < 0, allora Rem = Rem + Divisor, shift a sx del registro di uscita (nuovo bit = 0).
2b. Altrimenti, shift a sx del registro di uscita (nuovo bit = 1).
3. Se i bit non sono terminati, torna al punto 1.
4. Shift a dx della parte sx del registro di uscita (Rem).

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem - Divisor	(00010-00011<0)|0110|0	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
		00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
		00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
3, 1 Rem = Rem – Divisor	00000|10|001	00011
		00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
3, 1 Rem = Rem – Divisor	00000|10|001	00011
2b, shift L, nuovo bit 1	00001|0|0011	00011
		00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
3, 1 Rem = Rem – Divisor	00000|10|001	00011
2b, shift L, nuovo bit 1	00001|0|0011	00011
3, 1 Rem = Rem – Divisor	(00001-00011<0)|0|0011	00011
		00011
		00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
3, 1 Rem = Rem – Divisor	00000|10|001	00011
2b, shift L, nuovo bit 1	00001|0|0011	00011
3, 1 Rem = Rem – Divisor	(00001-00011<0)|0|0011	00011
2a, ripristina Rem	00001|0|0011	00011
2a, shift L, nuovo bit 0	00010||00110	00011
		00011

Sol. 3 (10011:00011=00110)

	Remainder | Useful dividend | Quotient	Divisor
0a	00000|10011|	00011
0b	00001|0011|0	00011
1, Rem = Rem - Divisor	(00001-00011<0)|0011|0	00011
2a, ripristina Rem	00001|0011|0	00011
2a, shift L, nuovo bit 0	00010|0110|0	00011
3, 1 Rem = Rem – Divisor	(00010-00011<0)|0110|0	00011
2a, ripristina Rem	00010|0110|0	00011
2a, shift L, nuovo bit 0	00100|110|00	00011
3, 1 Rem = Rem – Divisor	00001|110|00	00011
2b, shift L, nuovo bit 1	00011|10|001	00011
3, 1 Rem = Rem – Divisor	00000|10|001	00011
2b, shift L, nuovo bit 1	00001|0|0011	00011
3, 1 Rem = Rem – Divisor	(00001-00011<0)|0|0011	00011
2a, ripristina Rem	00001|0|0011	00011
2a, shift L, nuovo bit 0	00010||00110	00011
3, 4 shift R Rem	00001||00110	00011

Es. 4

- Si effettui la divisione a 4 bit 1101/101, utilizzando l'algoritmo evidenziato, evidenziando il contenuto dei registri ad ogni passo.