

Algorithmen zur Division

- Umkehrung der Multiplikation: Berechnung von $q = a / b$ durch wiederholte bedingte Subtraktionen und Schiebeoperationen
- in jedem Schritt wird Divisor b testweise vom aktuellen Rest r subtrahiert: $q_i = 1$, falls $r = r - b > 0$
 $q_i = 0$ und Korrektur durch $r = r + b$, falls $r < 0$

- Beispiel: $103_{10} / 9_{10} = 11_{10}$ mit Rest 4_{10}

0001100111	/	01001	=	01011
- 01001				↑
00111				Quotient
- 01001				
11110				
+ 01001	←	Korrektur		
001111				
- 01001				
001101				
- 01001				
00100	←	Rest		

Algorithmen zur Division (Forts.)

- allgemein gilt: $a = q \cdot b + r$ mit Rest $r < b$
- im folgenden sei angenommen, daß b eine positive n -Bit Zahl und a eine positive $2n$ -Bit Zahl darstellen
 \Rightarrow es muß gelten: 1) $a < 2^{n-1} \cdot b$ bzw. $q < 2^{n-1}$
2) $b \neq 0$ (\Rightarrow Ausnahmebehandlung !)

- alle Divisionsalgorithmen führen in Schritt i folgende Operation aus: $r(i) = 2 \cdot r(i-1) - q_{n-i} \cdot 2^n \cdot b$ mit $i = 1, \dots, n$ und $r(0) = a$
- Es wird korrektes Ergebnis berechnet, da für Rest $r = r(n)$ gilt:

$$\begin{aligned}
 r(n) &= 2 \cdot r(n-1) - q_0 \cdot 2^n \cdot b \\
 &= 2 \cdot (2 \cdot r(n-2) - q_1 \cdot 2^n \cdot b) - q_0 \cdot 2^n \cdot b = \dots \\
 &= 2^n r(0) - (2^{n-1} q_{n-1} + \dots + 2q_1 + q_0) \cdot 2^n \cdot b
 \end{aligned}$$

Somit folgt: $a = r(0) = q \cdot b - r(n)/2^n$

Algorithmen zur Division (Forts.)

- Algorithmus mit Wiederherstellung des Rests durch Addition („*Restoring Division*“)
- statt einer $2n$ -Bit Addition $r = r + 2^n \times b$ genügt hier auch eine n -Bit Addition $r' = r' + b$, wenn r' die aktuellen höherwertigen n Bit von r darstellt

```
r = a
q = 0
for i = 0 to n-1 {
  shift left r by 1
  r = r - 2nb
  if (r >= 0)
    qbit = 1
  else
    qbit = 0
    r = r + 2nb
  q = 2q + qbit
}
```

Algorithmen zur Division (Forts.)

- Algorithmus mit Bildung eines neuen Rests nur in dem Fall, daß die Differenz nicht negativ ist („*Non-Performing Division*“)

```
r = a
q = 0
for i = 0 to n-1 {
  shift left r by 1
  tmp = r - 2nb
  if (tmp >= 0)
    qbit = 1
    r = tmp
  else
    qbit = 0
  q = 2q + qbit
}
```

Algorithmen zur Division (Forts.)

- Algorithmus mit Beibehaltung eines negativen Restes („*Non-Restoring Division*“)
 - korrigierende Addition(en) anstatt Subtraktion(en) in den Folgeschritt(en), bis Partialrest r wieder positiv ist
 - ggf. Korrektur bei negativem Rest erforderlich
- ```

r = a
q = 0
for i = 0 to n-1 {
 shift left r by 1
 if (r >= 0)
 r = r - 2nb
 qbit = 1
 else
 r = r + 2nb
 qbit = 0
 q = 2q + qbit
}
if (r < 0)
 r = r + 2nb

```

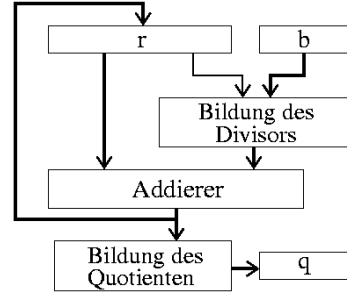
## Algorithmen zur Division (Forts.)

- zur Äquivalenz von „*Restoring*“ und „*Non-Restoring*“ Division:
  - „*Restoring*“:  
 $r(i) = 2 \cdot r(i-1) - 2^n \cdot b < 0 \Rightarrow r(i+1) = 2 \cdot r(i) - 2^n \cdot b = 4 \cdot r(i-1) - 2^n \cdot b$
  - „*Non-Restoring*“:  
 $r(i) = 2 \cdot r(i-1) - 2^n \cdot b < 0 \Rightarrow r(i+1) = 2 \cdot r(i) + 2^n \cdot b = 4 \cdot r(i-1) - 2^n \cdot b$
- Aufwandsanalyse: im Worst-Case Fall sind genau  $n - 1$  Nullen im Quotient  $q$  enthalten
  - „*Restoring*“ Division:  
 $n + n - 1$  Additionen/Subtraktionen
  - „*Non-Performing*“ Division:  
 $n$  Subtraktionen und  $n - 1$  Kopieroperationen
  - „*Non-Restoring*“ Division:  
 $n$  Additionen/Subtraktionen (ggf. +1 Korrekturaddition)

$\Rightarrow$  „*Non-Restoring*“ Division ist das schnellste Verfahren !

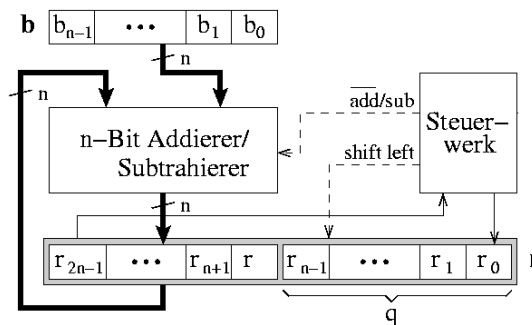
## Implementierung:

- allgemeiner Aufbau eines Dividierers:
- Behandlung negativer Dividenden und Divisoren sehr umständlich:
  - es gibt kein Äquivalent zum Booth-Algorithmus !
  - i.a. Umwandlung in Vorzeichen und Betrag
- es gibt verschiedene Möglichkeiten, die Division zu beschleunigen, z.B.:
  - Verwendung von schnellen Addierern bzw. Carry-Save Addierern
  - Überspringen von  $k$  führenden Nullen im Rest  $r$ : schiebe Rest  $r$  um  $k$  Positionen nach links und setze die ersten  $k$  Bits von  $q$  auf 0
  - simultane Generierung mehrerer Quotientenbits durch Subtraktion des Vielfachen von  $b$
  - parallele Subtraktionen sind jedoch nicht möglich !



## sequentieller Dividierer

- sequentielle Division ist direkt in Hardware implementierbar
- mit
- $n$ -Bit Register  $b$ ,
  - $2n$ -Bit Register  $r$ ,
  - $n$ -Bit Addierer/Subtrahierer



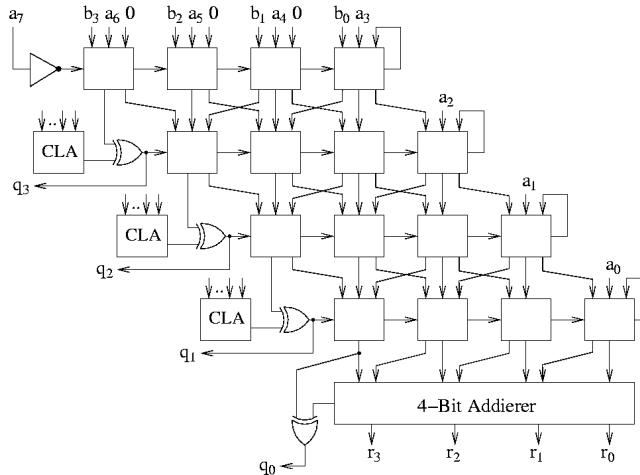
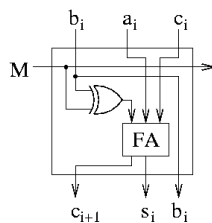
- nach  $n$  Schritten steht Quotient in  $r_{n-1} \dots r_0$ , Rest in  $r_{2n-1} \dots r_n$
- das Steuerwerk implementiert Algorithmus, z.B. gilt für „Non-Restoring“ Division:  $\text{add/sub} := r_{2n-1}$  und  $r_0 := r_{2n-1}$
- Zeit:  $n \cdot (\Delta_{\text{Add}} + 3\hat{\delta})$

## paralleler Dividierer

- wiederholte Subtraktion auch durch **Felddividierer** („array divider“) mit CSA-Addierern implementierbar:

- Zeit:  
 $\Delta_{\text{Add}} + (n-1)8\delta$   
 $+ 5\delta$

- Aufbau einer Zelle:



## SRT-Dividierer

- benannt nach den Entwicklern Sweeny, Robertson und Tocher, die diesen Algorithmus fast gleichzeitig vorstellten (1958)
- Algorithmus mit dreiwertiger Kodierung der Quotientenbits  
 $q'_i \in \{\bar{1}, 0, 1\}$  bzw.  
 $q'_i \in \{-1, 0, 1\}$  :
  - Quotient  $q$  wird somit **redundant** kodiert
  - Zahl der Additionen und Subtraktionen im Vergleich zur „Non-Restoring“ Division reduziert

```

r = a
q' = 0
for i = 1 to n {
 shift left r by 1
 if (r >= 2^n b)
 qbit = 1
 else if (r < -2^n b)
 qbit = -1
 else
 qbit = 0
 r = r - qbit * 2^n b
 q'_{n-i} = qbit
}
if (r < 0)
 r = r + 2^n b

```

## SRT-Dividierer (Forts.)

- *Problem:* SRT-Dividierer benötigt je Schritt Vergleich von  $r$  sowohl mit  $2^n b$  als auch mit  $-2^n b$
- *Lösung:* der Divisor  $b$  wird zuvor durch Schiebeoperationen derart *normalisiert*, daß hinter dem Vorzeichenbit das erste Nachkommabit  $\neq 0$  ist, d.h. es gilt:  $\frac{1}{2} \leq b < 1$
- der Vergleich wird dann wie folgt angenähert:  
Dividend  $a$  und Rest  $r$  sind hierbei auch Zahlen aus  $[0,1)$
- aufgrund der redundanten Darstellung von  $q$  ist Ergebnis jedoch korrekt !
- Rückwandlung von  $q$  in binär kodierte Zahl erforderlich !

```
if (r >= 1/2)
 qbit = 1
else if (r < -1/2)
 qbit = -1
else
 qbit = 0
r = r - qbit·b
```

## SRT-Dividierer (Forts.)

- Zahl der erforderlichen Operationen ist datenabhängig!
- für einen  $n$ -Bit Dividenden benötigt SRT Algorithmus im Mittel nur  $n/2.67$  Additionen
- weitere Beschleunigung durch simultane Generierung mehrerer Quotientenbits, d.h. je Schritt Bestimmung einer Quotientenziffer  $q_i \in \{-\alpha, -\alpha+1, \dots, -1, 0, 1, \dots, \alpha-1, \alpha\}$  (dies wird auch als Radix- $2^\alpha$  SRT Verfahren bezeichnet)
- Abschätzung von  $q_i$  erfolgt i.a. über in PLAs gespeicherten Tabellen in Abhängigkeit von den höherwertigen Bits des aktuellen Rests  $r$  und den höherwertigen Bits des Divisors  $b$
- bei dem 1994 im Intel Pentium Prozessor entdeckten Bug waren 5 Einträge in einer solchen Tabelle falsch !