Basic Division Scheme

台灣大學電子所
吳安宇

For Advanced VLSI Course

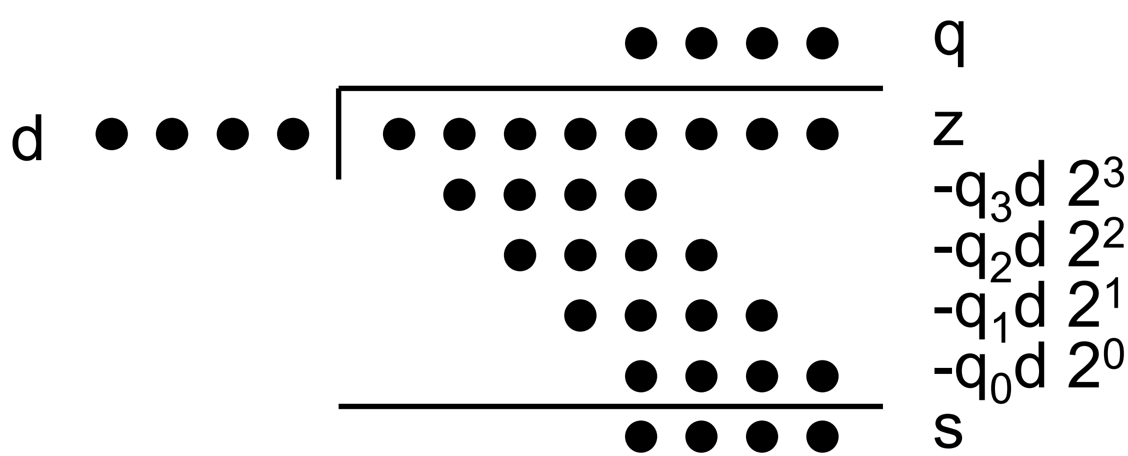
2002

Outline

- Shift/subtract division algorithm.
- Programmed division.
- Restoring hardware dividers.
- Nonstoring and signed division.
- Division by constants
- Radix-2 SRT division.
- High-Radix division.

Shift/Subtract Division Algorithms

z	*Dividend*	$z_{2k-1}z_{2k-2}\dots z_1 z_0$
d	*Divisor*	$d_{k-1}d_{k-2}\dots d_1 d_0$
q	*Quotient*	$q_{k-1}q_{k-2}\dots q_1 q_0$
s	*Remainder* $[z-(d\times q)]$	$s_{k-1}s_{k-2}\dots s_1 s_0$



Division can be done by a sequence of shifts and subtraction.

Overflow Check

- Multiplication:
  - product of two *k*-bit numbers is always 2*k* bits.

integer

$$z = [(d \times q)] + s \quad integer$$

$$z < (2^k - 1)d + d = 2^k d$$

- Division:
  - quotient of a 2*k*-bit number divided by a *k*-bit number may more than *k* bits.

fractions

$$2^{-2k} z = [(2^{-k} d) \times (2^{-k} q)] + 2^{-2k} s \quad fractions$$

$$z_{frac} = [(d_{frac} \times q_{frac})] + 2^{-k} s_{frac}$$

$$Z_{frac} < d_{frac}$$

Sequential Division Algorithm

- Left shift *partial remainder*, align to the term to be subtracted.

$$\underbrace{S^{(j)} = \underbrace{2s^{(j-1)}}_{\text{Shift left}} - q_{k-j}(2^k d)}_{\text{Subtract}} \quad with \quad s^{(0)} = z \quad and \quad s^{(k)} = 2^k s$$

After *k* iteration

$$S^{(k)} = 2^k s^{(0)} - q(2^k d) = 2^k[z - (q \times d)] = 2^k s$$

Fractional version

$$S^{(j)}_{frac} = 2s^{(j-1)}_{frac} - q_{-j} d_{frac} \quad with \quad s^{(0)}_{frac} = z_{frac} \quad and \quad s^{(k)}_{frac} = 2^k s_{frac}$$

Example of sequential division with integer and fractional operands

Integer division

z	0 1 1 1	0 1 0 1	
$2^4 d$	1 0 1 0		
$s^{(0)}$	0 1 1 1	0 1 0 1	
$2s^{(0)}$	0 1 1 1 0	1 0 1	
$-q_3 2^4 d$	1 0 1 0		$\{q_3 = 1\}$
$s^{(1)}$	0 1 0 0	1 0 1	
$2s^{(1)}$	0 1 0 0 1	0 1	
$-q_2 2^4 d$	0 0 0 0		$\{q_2 = 0\}$
$s^{(2)}$	1 0 0 1	0 1	
$2s^{(2)}$	1 0 0 1 0	1	
$-q_1 2^4 d$	1 0 1 0		$\{q_1 = 1\}$
$s^{(3)}$	1 0 0 0	1	
$2s^{(3)}$	1 0 0 0 1		
$-q_0 2^4 d$	1 0 1 0		$\{q_0 = 1\}$
$s^{(4)}$	0 1 1 1		
s		0 1 1 1	
q		1 0 1 1	

Fractional division

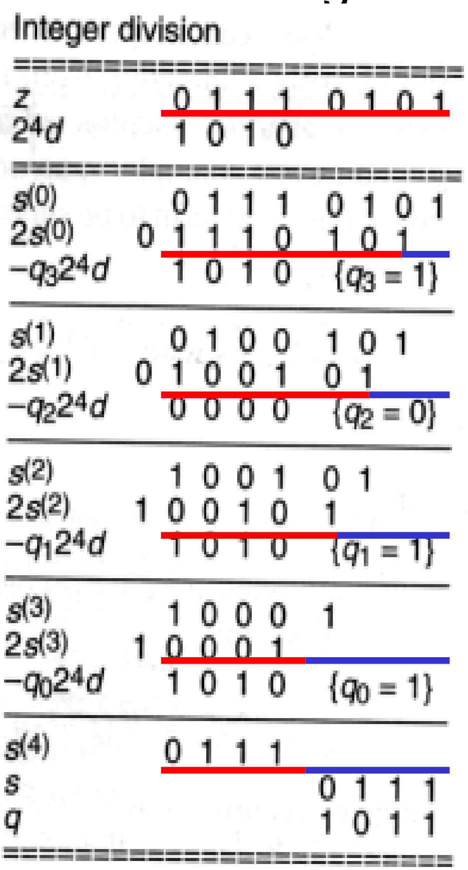
z_{frac}	.0 1 1 1	0 1 0 1	
d_{frac}	.1 0 1 0		
$s^{(0)}$	.0 1 1 1	0 1 0 1	
$2s^{(0)}$	0.1 1 1 0	1 0 1	
$-q_{-1} d$	.1 0 1 0		$\{q_{-1} = 1\}$
$s^{(1)}$	.0 1 0 0	1 0 1	
$2s^{(1)}$	0.1 0 0 1	0 1	
$-q_{-2} d$	.0 0 0 0		$\{q_{-2} = 0\}$
$s^{(2)}$	.1 0 0 1	0 1	
$2s^{(2)}$	1.0 0 1 0	1	
$-q_{-3} d$	.1 0 1 0		$\{q_{-3} = 1\}$
$s^{(3)}$	.1 0 0 0	1	
$2s^{(3)}$	1.0 0 0 1		
$-q_{-4} d$	.1 0 1 0		$\{q_{-4} = 1\}$
$s^{(4)}$	.0 1 1 1		
s_{frac}	.0 0 0 0	0 1 1 1	
q_{frac}	.1 0 1 1		

Programmed Division

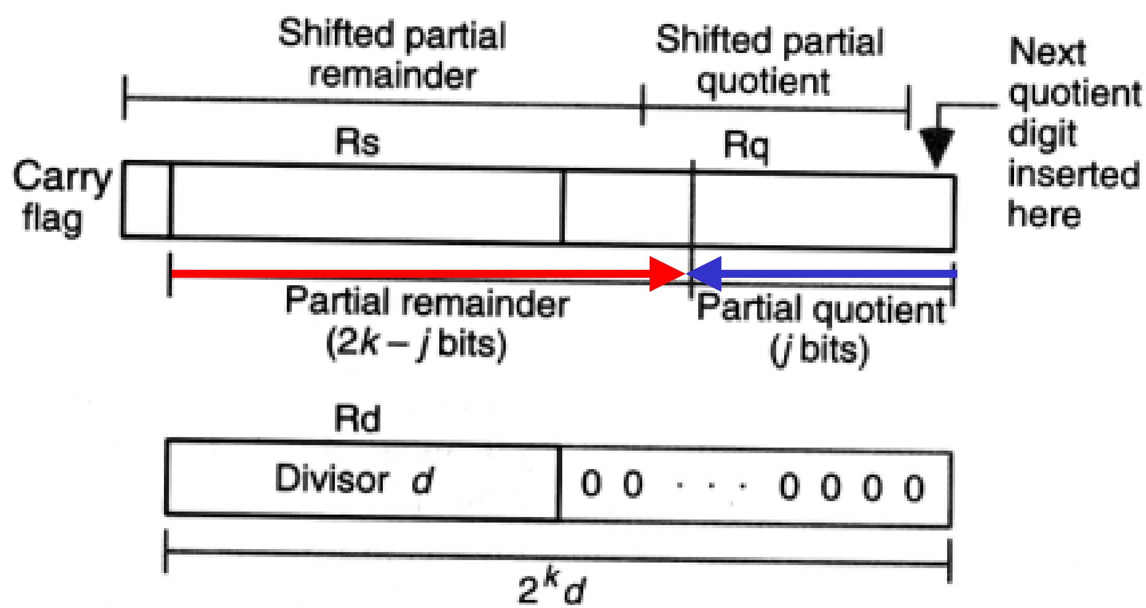
```
{Using left shifts, divide unsigned 2k-bit dividend,
z_high|z_low, storing the k-bit quotient and remainder.
Registers:     R0 holds 0               Rc for counter
               Rd for divisor           Rs for z_high & remainder
               Rq for z_low & quotient}

{Load operands into registers Rd, Rs, and Rq}

     div:      load       Rd with divisor
               load       Rs with z_high
               load       Rq with z_low

{Check for exceptions}

               branch     d_by_0 if Rd = R0
               branch     d_ovfl if Rs > Rd

{Initialize counter}

               load       k into Rc

{Begin division loop}

 d_loop:       shift      Rq left 1        {zero to LSB, MSB to carry}
               rotate     Rs left 1        {carry to LSB, MSB to carry}
               skip       if carry = 1
               branch     no_sub if Rs < Rd
               sub        Rd from Rs
               incr       Rq               {set quotient digit to 1}
 no_sub:       decr       Rc               {decrement counter by 1}
               branch     d_loop if Rc ≠ 0

{Store the quotient and remainder}

               store      Rq into quotient
               store      Rs into remainder
 d_by_0:       . . .
 d_ovfl:       . . .
 d_done:       . . .
```

Programmed Division (con't)

- Use *shift* and *add* to perform integer division by a processor.
- Two *k*-bit register to store the *partial remainder* and the *quotient*.



```
Integer division
=========================
z            0 1 1 1   0 1 0 1
2^4 d        1 0 1 0
=========================
s(0)         0 1 1 1   0 1 0 1
2s(0)      0 1 1 1 0   1 0 1
−q3 2^4 d    1 0 1 0   {q3 = 1}
-------------------------
s(1)         0 1 0 0   1 0 1
2s(1)      0 1 0 0 1   0 1
−q2 2^4 d    0 0 0 0   {q2 = 0}
-------------------------
s(2)         1 0 0 1   0 1
2s(2)      1 0 0 1 0   1
−q1 2^4 d    1 0 1 0   {q1 = 1}
-------------------------
s(3)         1 0 0 0   1
2s(3)      1 0 0 0 1
−q0 2^4 d    1 0 1 0   {q0 = 1}
-------------------------
s(4)         0 1 1 1
s                      0 1 1 1
q                      1 0 1 1
=========================
```



Restoring Hardware Dividers

- The basic eq. for signed division is

$$(d \times q) + s \quad along\ with$$

$$sign(s) = sign(z) \quad and \quad |s| < |d|$$

- For example

$$z = 5 \quad d = 3 \quad \Rightarrow \quad q = 1 \quad s = 2$$
$$z = 5 \quad d = -3 \quad \Rightarrow \quad q = -1 \quad s = 2$$
$$z = -5 \quad d = 3 \quad \Rightarrow \quad q = -1 \quad s = -2$$
$$z = -5 \quad d = -3 \quad \Rightarrow \quad q = 1 \quad s = -2$$

- The magnitudes of *q* and *s* are unaffected by the input signs.

Restoring Hardware Dividers (con't)

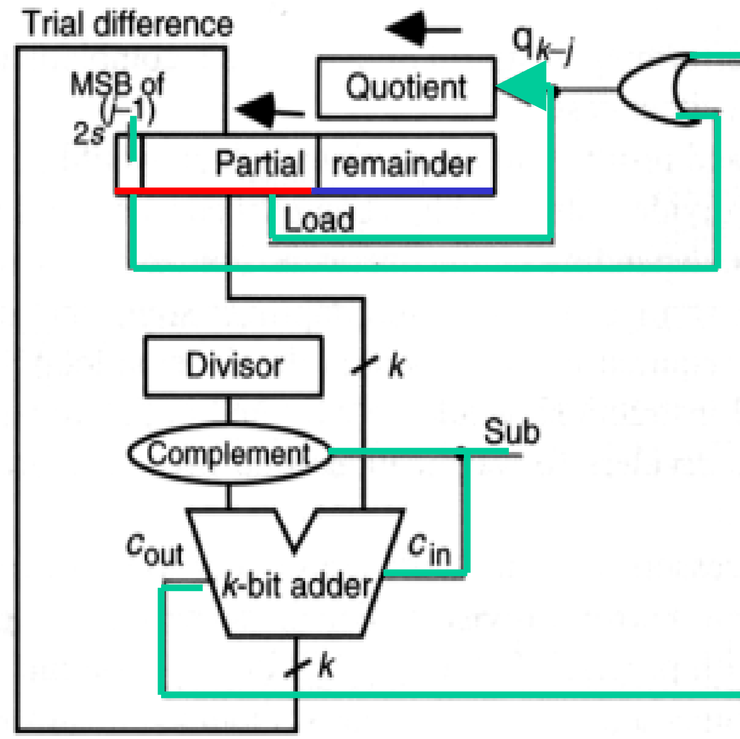
- Because the magnitudes of q and s are unaffected by the input signs. Signed division can be converted into unsigned values and, at the end, the signs is determined by the sign bits or via complementation.
- This is the method of choice with the restoring division algorithm.

Restoring Hardware Dividers (con't)

z	0 1 1 1	0 1 0 1	
$2^4 d$	0 1 0 1 0		No overflow, since: $(0111)_{two} < (1010)_{two}$
$-2^4 d$	1 0 1 1 0		
$s^{(0)}$	0 0 1 1 1	0 1 0 1	
$2s^{(0)}$	0 1 1 1 0	1 0 1	
$+(-2^4 d)$	1 0 1 1 0		
$s^{(1)}$	0 0 1 0 0	1 0 1	Positive, so set $q_3 = 1$
$2s^{(1)}$	0 1 0 0 1	0 1	
$+(-2^4 d)$	1 0 1 1 0		
$s^{(2)}$	1 1 1 1 1	0 1	Negative, so set $q_2 = 0$ and restore
$s^{(2)} = 2s^{(1)}$	0 1 0 0 1	0 1	
$2s^{(2)}$	1 0 0 1 0	1	
$+(-2^4 d)$	1 0 1 1 0		
$s^{(3)}$	0 1 0 0 0	1	Positive, so set $q_1 = 1$
$2s^{(3)}$	1 0 0 0 1		
$+(-2^4 d)$	1 0 1 1 0		
$s^{(4)}$	0 0 1 1 1		Positive, so set $q_0 = 1$
s		0 1 1 1	
q		1 0 1 1	

restore

unchanged



Drawback of Restoring Division

- Timing issues because each *k* cycles must be long enough to allow following events in sequence:
  - Shifting of the registers.
  - Propagation of signals through the adder (check carry)
  - Storing of the quotient digit. (storing)
- So the sign of the trial difference must be sampled near the negative edge. (drawback)
- To avoid such timing issues, *nonrestoring division algorithm* can be used.

Nonrestoring and Signed Division

- Always store difference in the partial remainder register.
- Allow partial remainder being temporarily incorrect (hence the name "nonrestoring").
- For example:

[Restoring]:	[Nonrestoring]:
$cycle\ n$	$cycle\ n$
$incorrect\ \ partial\ remainder\ u-2^k d$	$incorrect\ \ partial\ remainder\ u-2^k d$
$restore\ to\ u$	$skip\ restore$
$cycle\ n+1$	$cycle\ n+1$
$2u-2^k d$	$2(u-2^k d)+2^k d = 2u-2^k d$
	$(the\ same\ as\ restoring)$

Nonrestoring and Signed Division (con't)

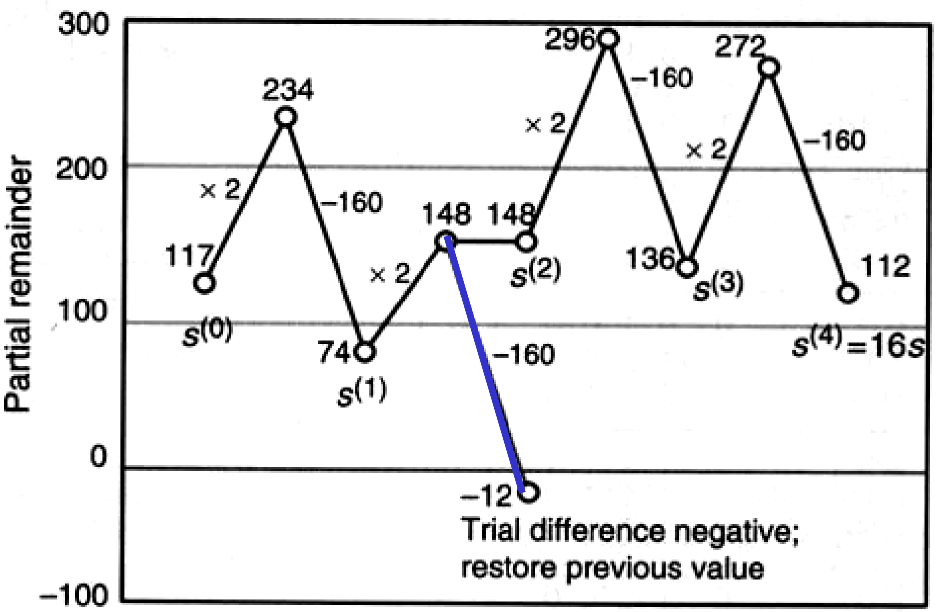
- Quotient digits are selected from the set {1,-1}, (1→sub,-1→add).
- Goal is to end up with a remainder matches the sign of the dividend. (dividend can be positive or negative).
- The rule for quotient digit selection becomes:

$$if\ \ sign(s) = sign(d)\ then\ q_{k-j} = 1\ else\ q_{k-j} = -1$$

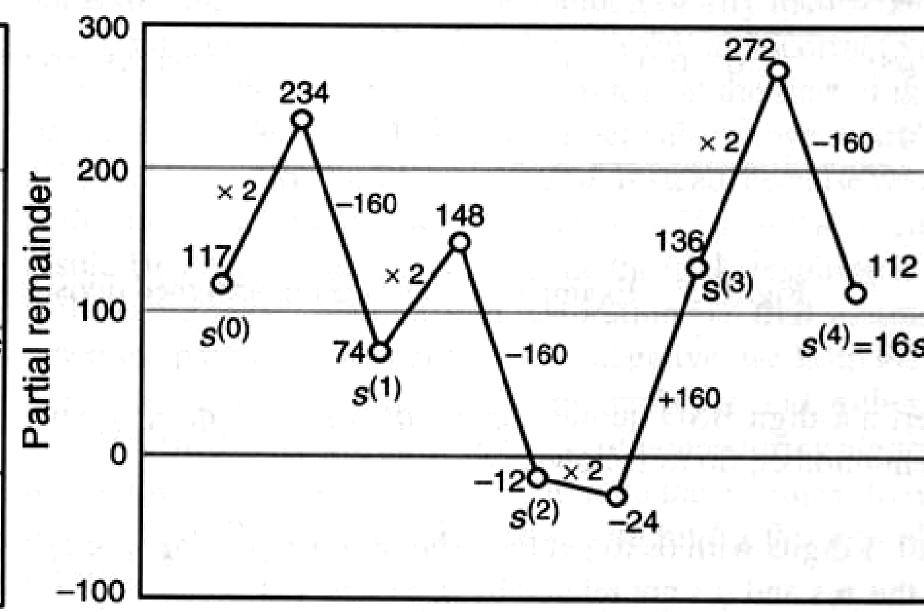
Nonrestoring Unsigned division example

z		0 1 1 1	0 1 0 1	No overflow, since: $(0111)_{two} < (1010)_{two}$
$2^4 d$	0	1 0 1 0		
$-2^4 d$	1	0 1 1 0		
$s^{(0)}$	0	0 1 1 1	0 1 0 1	
$2s^{(0)}$	0	1 1 1 0	1 0 1	Positive, so subtract
$+(-2^4 d)$	1	0 1 1 0		
$s^{(1)}$	0	0 1 0 0	1 0 1	
$2s^{(1)}$	0	1 0 0 1	0 1	Positive, so set $q_3 = 1$ and subtract
$+(-2^4 d)$	1	0 1 1 0		
$s^{(2)}$	1	1 1 1 1	0 1	
$2s^{(2)}$	1	1 1 1 0	1	Negative, so set $q_2 = 0$ and add
$+2^4 d$	0	1 0 1 0		
$s^{(3)}$	0	1 0 0 0	1	
$2s^{(3)}$	1	0 0 0 1		Positive, so set $q_1 = 1$ and subtract
$+(-2^4 d)$	1	0 1 1 0		
$s^{(4)}$	0	0 1 1 1		Positive, so set $q_0 = 1$
s			0 1 1 1	
q			1 0 1 1	

Partial Remainder variation for restoring and nonrestoring dividsion



(a) Restoring.



(b) Nonrestoring.

Nonstoring Signed Division: Two Problems

- The quotient with digits 1 and -1 must be converted to standard binary.
- If the final remainder *s* has a sign opposite that of *z*, a correction step addition ±d to the remainder and subtraction of ±1 from the quotient, is needed.
  - Convert a *k*-digit BSD quotient to a *k*-bit 2's complement number.

    A. replace all -1 digits with 0s to get the k-bit number

    $$p = p_{k-1}p_{k-2}\cdots p_0,\ p_i \in \{0,1\}$$

    B. complement p_{k-1} and then shift *p* left by 1 bit, inserting 1 to the LSB, get

    $$q = (\overline{p}_{k-1}p_{k-2}\cdots p_0 1)_{2's-compl}.$$

Convert a *k*-digit BSD quotient to a *k*-bit 2's complement number

Proof:

$$(\overline{p}_{k-1} p_{k-2} \dots p_0 1)_{2's-compl.} = -(1 - p_{k-1})2^k + 1 + \sum_{i=0}^{k-2} p_i 2^{i+1}$$

$$= -(2^k - 1) + 2\sum_{i=0}^{k-1} p_i 2^i$$

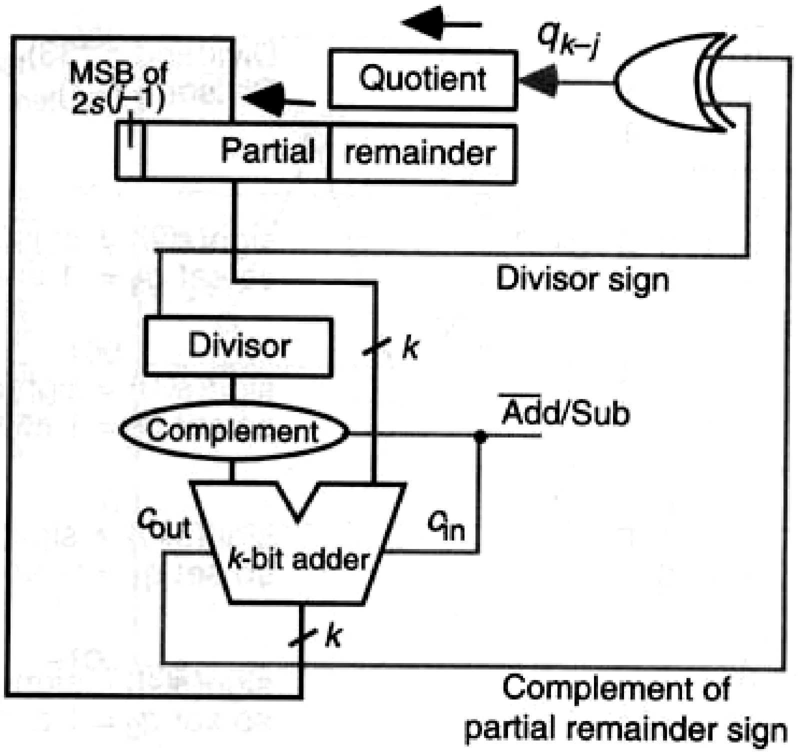
$$= \sum_{i=0}^{k-1} (2p_i - 1)2^i$$

$$= \sum_{i=0}^{k-1} q_i 2^i = q$$

$$note: \quad (1)\ q_i = 2p_i - 1 \quad (2) \sum_{i=0}^{k-1} 2^i = 2^k - 1$$

Nonstoring Signed Division: example

z		0 0 1 0 0 0 0 1	Dividend = $(33)_{ten}$ Divisor = $(-7)_{ten}$
2^4d	1	1 0 0 1	
-2^4d	0	0 1 1 1	
$s^{(0)}$	0	0 0 1 0 0 0 0 1	
$2s^{(0)}$	0	0 1 0 0 0 0 1	sign($s^{(0)}$) ≠ sign(d), so set $q_3 = {}^{-}1$ and add
$+2^4d$	1	1 0 0 1	
$s^{(1)}$	1	1 1 0 1 0 0 1	
$2s^{(1)}$	1	1 0 1 0 0 1	sign($s^{(1)}$) = sign(d), so set $q_2 = 1$ and subtract
$+(-2^4d)$	0	0 1 1 1	
$s^{(2)}$	0	0 0 0 1 0 1	
$2s^{(2)}$	0	0 0 1 0 1	sign($s^{(2)}$) ≠ sign(d), so set $q_1 = {}^{-}1$ and add
$+2^4d$	1	1 0 0 1	
$s^{(3)}$	1	1 0 1 1 1	
$2s^{(3)}$	1	0 1 1 1	sign($s^{(3)}$) = sign(d), so set $q_0 = 1$ and subtract
$+(-2^4d)$	0	0 1 1 1	
$s^{(4)}$	1	1 1 1 0	sign($s^{(4)}$) ≠ sign(d) Corrective subtraction
$+(-2^4d)$	0	0 1 1 1	
$s^{(4)}$	0	0 1 0 1	
s		0 1 0 1	Remainder = $(5)_{ten}$
q		${}^{-}1$ 1 ${}^{-}1$ 1	Ucorrected BSD quotient
p		0 1 0 1	${}^{-}1$s replaced by 0s
		/ / / /	
Shifted p		1 1 0 1 1	Add 1 to correct
$q_{2\text{'s-compl}}$		1 1 0 0	Quotient = $(-4)_{ten}$



Division by Constants

- Here, we will consider only division by odd integers, since division by even integer can be performed by dividing by a odd integer then shift the result.
- For example: K/20=K/(5*4)=(K/5)*(1/4)=(K/5)>>2.
- If only a limited number of constant divisor are of interest, their reciprocals can be precomputed with an appropriate precision and stored in a table.

Division by Constants (con't)

- Faster constant division can be obtained for many small odd divisors by using that: for each odd integer *d* there exists an odd integer *m* such that $d*m=2^n-1$.

$$\frac{1}{d}=\frac{m}{2^n-1}=\frac{m}{2^n(1-2^{-n})}$$

$$=\frac{m}{2^n}(1+2^{-n})(1+2^{-2n})(1+2^{-4n})\ldots$$

- For example: d=5, m=3 and n=4. Thus for 24 bits of precision,

$$\frac{z}{5}=\frac{3z}{2^4-1}=\frac{3z}{16(1-2^{-4})}$$

$$=\frac{3z}{16}(1+2^{-4})(1+2^{-8})(1+2^{-16})\ldots$$

Note that the next term $(1+2^{-32})$ would shift out the entire operand.

Division by Constants (con't)

❖ Follow preceding example, to effect division by 5

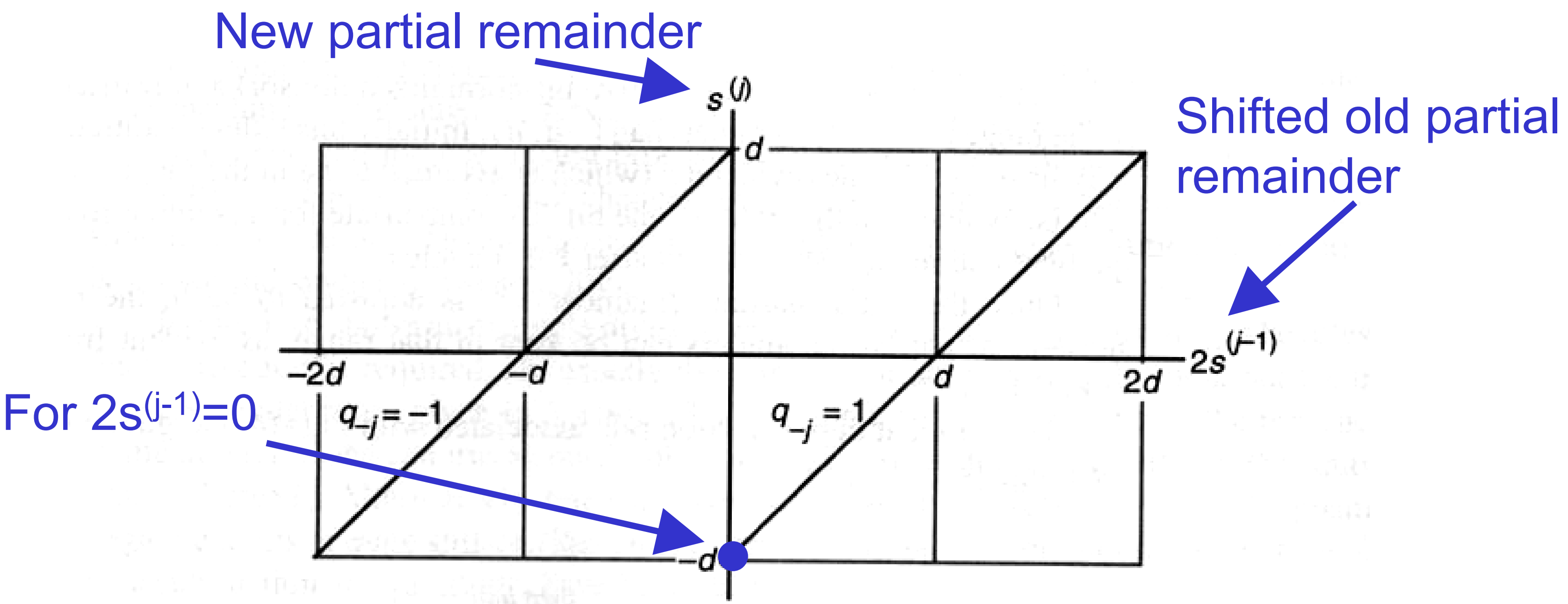
$$
\begin{array}{ll}
q \leftarrow z + z \text{ shift-left } 1 & \{3z \text{ computed}\} \\
q \leftarrow q + q \text{ shift-right } 4 & \{3z(1+2^{-4})\} \\
q \leftarrow q + q \text{ shift-right } 8 & \{3z(1+2^{-4})(1+2^{-8})\} \\
q \leftarrow q + q \text{ shift-right } 16 & \{3z(1+2^{-4})(1+2^{-8})(1+2^{16})\} \\
q \leftarrow q \text{ shift-right } 4 & \{3z(1+2^{-4})(1+2^{-8})(1+2^{-16})/16\}
\end{array}
$$

Radix-2 SRT Division: Review of nonrestoring division

- Reconsider radix-2 nonrestoring division algorithm for fractional operands.

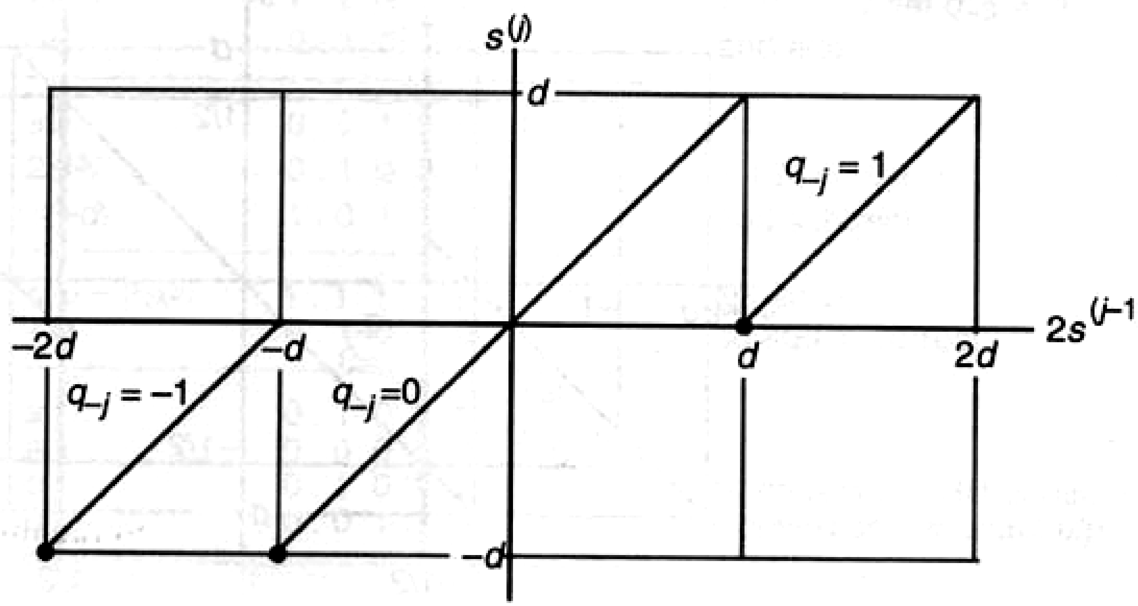
$$S^{(j)} = 2s^{(j-1)} - q_{-j}d \quad with \quad s^{(0)} = z \quad and \quad s^{(k)} = 2^k s$$

- Quotient is obtained with the digit set {-1,1} and is then converted to the standard digit set {0,1}.



Radix-2 SRT Division (con't)

- Quotient is obtained using digit set {-1,0,1}.
- Quotient "0" is selected when $q_{-j}=0$ for $-d \leq 2s^{(j-1)} < d$
- Quotient "0" is simple shift, can speed up the division operation.
- But determined $-d \leq 2s^{(j-1)} < d$ need trial subtraction. Would consume more time than they save!



Radix-2 SRT Division (con't)

- ***SRT***: *Sweeney*, *Roberson*, and *Tocher* discovered SRT division about the same time.
- *Normalized* divisor and *normalized* partial dividend.
- Divisor and partial dividend is limited in the range [1/2,1) or (-1,-1/2].
- Easier comparison can be used due to normalized divisor.

Radix-2 SRT Division (con't)

- Because of normalized divisor. comparison become:
  - $2s^{(j-1)} > +½ = (0.1)_{2\text{'s-compl}}$ implies $2s^{(j-1)} = (0.1u_{-2}u_{-3}\ldots)_{2\text{'s-compl}}$
  - $2s^{(j-1)} < -½ = (1.1)_{2\text{'s-compl}}$ implies $2s^{(j-1)} = (1.0u_{-2}u_{-3}\ldots)_{2\text{'s-compl}}$
- $2s^{(j-1)} > +½$ is given by $\overline{u}_0 u_{-1}$, and $2s^{(j-1)} < -½$ is given by $u_0\overline{u}_{-1}$. Much easier!.

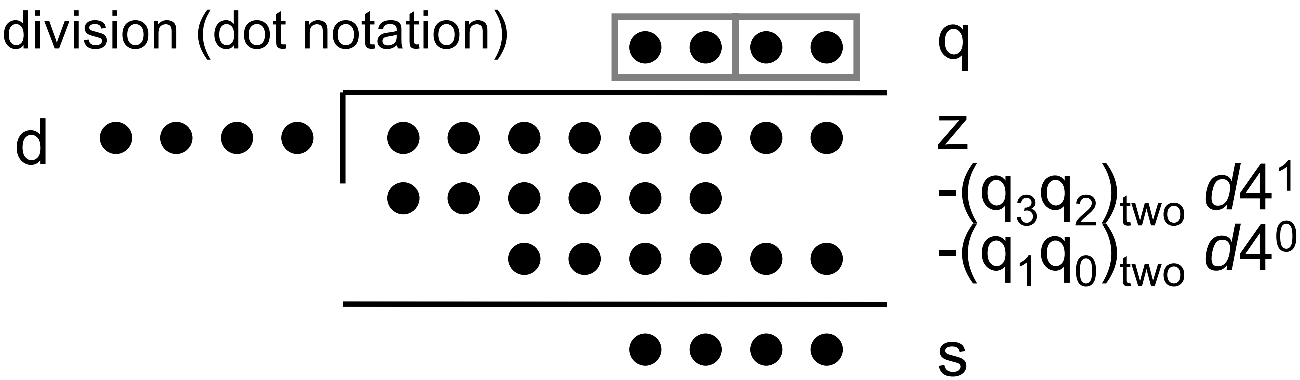
Example of Radix-2 SRT Division

z	.0100 0101	In [–1/2, 1/2), so OK
d	.1010	In [1/2, 1), so OK
$-d$	1.0110	
$s^{(0)}$	0.0100 0101	
$2s^{(0)}$	0.1000 101	$\geq 1/2$, so set $q_{-1} = 1$ and subtract
$+(-d)$	1.0110	
$s^{(1)}$	1.1110 101	
$2s^{(1)}$	$\boxed{1.1}$101 01	In [–1/2, 1/2), so set $q_{-2} = 0$
$s^{(2)} = 2s^{(1)}$	1.1101 01	
$2s^{(2)}$	1.1010 1	$< -1/2$, so set $q_{-3} = {}^{-}1$ and add
$+d$	0.1010	
$s^{(3)}$	0.0100 1	
$2s^{(3)}$	0.1001	$\geq 1/2$, so set $q_{-4} = 1$ and subtract
$+(-d)$	1.0110	
$s^{(4)} = 2s^{(3)}$	1.1111	Negative, so add to correct
$+d$	0.1010	
$s^{(4)}$	0.1001	
s	0.0000 1001	
q	0.10^{-}1 1	Uncorrected BSD quotient
q	0.0110	Convert and subtract *ulp*

Basic of High-Radix Division

z	*Dividend*	$z_{2k-1}z_{2k-2}\ldots z_1 z_0$
d	*Divisor*	$d_{k-1}d_{k-2}\ldots d_1 d_0$
q	*Quotient*	$q_{k-1}q_{k-2}\ldots q_1 q_0$
s	*Remainder*$[z-(d\times q)]$	$s_{k-1}s_{k-2}\ldots s_1 s_0$

Ex: Radix-4 division (dot notation)



High-Radix Division example

Radix-4 integer division

z	0 1 2 3	1 1 2 3	
$4^4 d$	1 2 0 3		
$s^{(0)}$	0 1 2 3	1 1 2 3	
$4s^{(0)}$	0 1 2 3 1	1 2 3	
$-q_3 4^4 d$	0 1 2 0 3		$\{q_3 = 1\}$
$s^{(1)}$	0 0 2 2	1 2 3	
$4s^{(1)}$	0 0 2 2 1	2 3	
$-q_2 4^4 d$	0 0 0 0 0		$\{q_2 = 0\}$
$s^{(2)}$	0 2 2 1	2 3	
$4s^{(2)}$	0 2 2 1 2	3	
$-q_1 4^4 d$	0 1 2 0 3		$\{q_1 = 1\}$
$s^{(3)}$	1 0 0 3	3	
$4s^{(3)}$	1 0 0 3 3		
$-q_0 4^4 d$	0 3 0 1 2		$\{q_0 = 2\}$
$s^{(4)}$	1 0 2 1		
s		1 0 2 1	
q		1 0 1 2	

Radix-10 fractional division

z_{frac}	. 7 0 0 3	
d_{frac}	. 9 9	
$s^{(0)}$	. 7 0 0 3	
$10s^{(0)}$	7 . 0 0 3	
$-q_{-1} d$	6 . 9 3	$\{q_{-1} = 7\}$
$s^{(1)}$	. 0 7 3	
$10s^{(1)}$	0 . 7 3	
$-q_{-2} d$	0 . 0 0	$\{q_{-2} = 0\}$
$s^{(2)}$	. 7 3	
s_{frac}	. 0 0 7 3	
q_{frac}	. 7 0	

Features of High-Radix Division

- Dividing binary number in radix 2b reduces the cycles required by a factor of b, but each cycle is more difficult to implement:
  - The higher radix makes the guessing of the correct quotient digit more difficult.
  - The value to be subtracted are determined sequentially, one per cycle. Possible value to be subtracted become harder to generate.
- Other variations in division and divider please consult reference “Computer Arithmetic: algorithm and hardware designs / Behrooz Parhami, OXFORD university press” ch13~ch16.