UMASS
UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

Part 3
Sequential Algorithms for Multiplication and Division

Israel Koren

Spring 2004

Sequential Multiplication

- X, A - multiplier and multiplicand
- $X = x_{n-1} x_{n-2} \ldots x_0$; $A = a_{n-1} a_{n-2} \ldots a_1 a_0$
- x_{n-1}, a_{n-1} - sign digits (sign-magnitude or complement methods)
- Sequential algorithm - $n-1$ steps
- Step j - multiplier bit x_j examined; product $x_j A$ added to $P^{(j)}$ - previously accumulated partial product $(P^{(0)} = 0)$

$$\boxed{P^{(j+1)} = (P^{(j)} + x_j \cdot A) \cdot 2^{-1} \quad ; \qquad j = 0, 1, 2, \cdots, n-2}$$

- Multiplying by 2^{-1} - shift by one position to the right - alignment necessary since the weight of x_{j+1} is double that of x_j

Sequential Multiplication - Proof

- Repeated substitution

$$
\begin{aligned}
P^{(n-1)} &= (P^{(n-2)} + x_{n-2} \cdot A) \cdot 2^{-1} \\
&= \Big((P^{(n-3)} + x_{n-3} \cdot A) \cdot 2^{-1} + x_{n-2} \cdot A\Big) \cdot 2^{-1} = \cdots \\
&= \Big(x_{n-2}2^{-1} + x_{n-3}2^{-2} + \cdots + x_0 2^{-(n-1)}\Big) \cdot A \\
&= \Big(\sum_{j=0}^{n-2} x_j \ 2^{-(n-1-j)}\Big) \cdot A = 2^{-(n-1)} \left(\sum_{j=0}^{n-2} x_j \ 2^j\right) \cdot A
\end{aligned}
$$

- If both operands positive ($x_{n-1}=a_{n-1}=0$) -

$$U = 2^{n-1} \cdot P^{(n-1)} = \Big(\sum_{j=0}^{n-2} x_j \ 2^j\Big) \cdot A = X \cdot A$$

- The result is a product consisting of 2(n-1) bits for its magnitude

Number of product bits

- Maximum value of U - when A and X are maximal

$$U_{max} = (2^{n-1}-1)(2^{n-1}-1) = 2^{2n-2} - 2^n + 1 = 2^{2n-3} + (2^{2n-3} - 2^n + 1)$$

- Last term positive for n≥3, therefore

$$2^{2n-3} < U_{max} < 2^{2n-2} \; ; \qquad n \geq 3$$

- 2n-2 bits required to represent the value - 2n-1 bits with the sign bit
- Signed-magnitude numbers - multiply two magnitudes and generate the sign separately (positive if both operands have the same sign and negative otherwise)

Negative operands

- For two's and one's complement - distinguish between multiplication with a negative multiplicand A and multiplication with a negative multiplier X
- If only multiplicand is negative - no need to change the previous algorithm - only add some multiple of a negative number that is represented in either two's or one's complement

Multiplication - Example

- *A* - negative, two's complement, *X* - positive, 4 bits
- Product - 7 bits, including sign bit
- Registers are 4 bits long - a double-length register required for storing the final product

A		1	0	1	1				-5
X	$\times$	0	0	1	1				3
$P^{(0)} = 0$		0	0	0	0				
$x_0 = 1 \Rightarrow$ Add A	$+$	1	0	1	1				
		1	0	1	1				
Shift		1	1	0	1	1			
$x_1 = 1 \Rightarrow$ Add A	$+$	1	0	1	1				
		1	0	0	0	1			
Shift		1	1	0	0	0	1		
$x_2 = 0 \Rightarrow$ Shift only		1	1	1	0	0	0	1	-15

- Vertical line separates most from least significant half - each stored in a single-length register
  - Bits in least significant half not used in the add operation

Least significant half of product

- Only 3 bit positions are utilized - least significant bit position unused - not necessarily final arrangement
- The 3 bits can be stored in 3 rightmost positions
- Sign bit of second register can be set in two ways
  - (1) Always set sign bit to 0, irrespective of sign of the product, since it is the least significant part of result
  - (2) Set sign bit equal to sign bit of first register
- Another possible arrangement -
  - Use all four bit positions in second register for the four least significant bits of the product
  - Use the rightmost two bit positions in the first register
  - Insert two copies of sign bit into remaining bit positions

Negative Multiplier - Two's Complement

- Each bit considered separately - sign bit (with negative weight) treated differently than other bits
- Two's complement numbers -

$$X = -x_{n-1}\ 2^{n-1} + \widetilde{X} \quad ; \quad \widetilde{X} = \sum_{j=0}^{n-2} x_j 2^j$$

- If sign bit of multiplier is ignored -

$$U = \widetilde{X} \cdot A = (X + x_{n-1} \cdot 2^{n-1}) \cdot A = X \cdot A + A \cdot x_{n-1} \cdot 2^{n-1}.$$

- $X \cdot A$ is the desired product - if $x_{n-1}=1$ - a correction is necessary

$$X \cdot A = U - A \cdot x_{n-1} \cdot 2^{n-1}$$

- The multiplicand A is subtracted from the most significant half of U

Negative Multiplier - Example

- **Multiplier and multiplicand - negative numbers in two's complement**

A		1	0	1	1					-5
X	$\times$	1	1	0	1					-3
$x_0 = 1 \;\Rightarrow$ Add A		1	0	1	1					
Shift		1	1	0	1	1				
$x_1 = 0 \;\Rightarrow$ Shift only		1	1	1	0	1	1			
$x_2 = 1 \;\Rightarrow$ Add A	$+$	1	0	1	1					
		1	0	0	1	1	1			
Shift		1	1	0	0	1	1	1		
$x_3 = 1 \;\Rightarrow$ Correct	$+$	0	1	0	1					
		0	0	0	1	1	1	1		$+15$

- **In correction step, subtraction of multiplicand is performed by adding its two's complement**

Negative Multiplier - One's Complement

$$X = -x_{n-1}(2^{n-1} - ulp) + \widetilde{X}$$

- and

$$X \cdot A = U - x_{n-1} \cdot 2^{n-1} \cdot A + x_{n-1} \cdot ulp \cdot A$$

- If $x_{n-1}=1$, start with $P^{(0)}=A$ - this takes care of the second correction term $x_{n-1} \cdot ulp \cdot A$
- At the end of the process - subtract the first correction term $x_{n-1} \cdot 2^{n-1} \cdot A$

Negative Multiplier - Example

- **Product of 5 and -3 - one's complement**

A		0	1	0	1					5
X	$\times$	1	1	0	0					-3
$x_3 = 1 \Rightarrow P^{(0)} = A$		0	1	0	1					
$x_0 = 0 \Rightarrow$ Shift		0	0	1	0	|	1			
$x_1 = 0 \Rightarrow$ Shift		0	0	0	1	|	0	1		
$x_2 = 1 \Rightarrow$ Add A	$+$	0	1	0	1					
		0	1	1	0	|	0	1		
Shift		0	0	1	1	|	0	0	1	
$x_3 = 1 \Rightarrow$ Correct	$+$	1	0	1	0	|	1	1	1	
		1	1	1	0	|	0	0	0	-15

- **As in previous example - subtraction of first correction term - adding its one's complement**
- **Unlike previous example - one's complement has to be expanded to double size using the sign digit - a double-length binary adder is needed**

Sequential Division

- Division - the most complex and time-consuming of the four basic arithmetic operations
- In general, result of division has two components
- Given a dividend X and a divisor D, generate a quotient Q and a remainder R such that
- $X = Q \cdot D + R$ (with $R < D$)
- Assumption - X,D,Q,R - positive
- If a double-length product is available after a multiply and we wish to allow the use of this result in a subsequent divide, then
- X may occupy a double-length register, while all other operands stored in single-length registers

Overflow & Divide by zero

- $Q \leq$ largest number stored in a single-length register ($< 2^{n-1}$ for a register with n bits)
- 1. $X < 2^{n-1} D$ - otherwise an overflow indication must be produced by arithmetic unit
- Condition can be satisfied by preshifting either X or D (or both)
- Preshifting is simple when operands are floating-point numbers
- 2. $D \neq 0$ - otherwise - a divide by zero indication must be generated
- No corrective action can be taken when $D=0$

Division Algorithm - Fractions

- **Assumption** - dividend, divisor, quotient, remainder are fractions - divide overflow condition is $X<D$
- Obtain $Q=0.q_1 \cdots q_m$ ($m=n-1$) - sequence of subtractions and shifts
- Step i - remainder is compared to divisor D - if remainder larger - quotient bit $q_i=1$, otherwise 0
- ith step - $r_i = 2r_{i-1}-q_iD$; $i=1,2,\ldots,m$
- r_i is the new remainder and r_{i-1} is the previous remainder ($r_0=X$)
- q_i determined by comparing $2r_{i-1}$ to D - the most complicated operation in division process

Division Algorithm - Proof

- The remainder in the last step is r_m and repeated substitution of the basic expression yields

$$\begin{aligned} r_m &= 2r_{m-1} - q_m \cdot D \\ &= 2(2r_{m-2} - q_{m-1} \cdot D) - q_m \cdot D = \cdots \\ &= 2^m r_0 - (q_m + 2q_{m-1} + \cdots + 2^{m-1} q_1) \cdot D \end{aligned}$$

- Substituting $r_0 = X$ and dividing both sides by 2^m results in

$$r_m 2^{-m} = X - (q_1 2^{-1} + q_2 2^{-2} + \cdots + q_m 2^{-m}) \cdot D;$$

- hence $r_m 2^{-m} = X - Q \cdot D$ as required
- True final remainder is $R = r_m 2^{-m}$

Division - Example 1 - Fractions

* $X=(0.100000)_2=1/2$
* $D=(0.110)_2=3/4$
* Dividend occupies double-length reg.
* $X<D$ satisfied

$r_0 = X$			0	.1	0	0	0	0	0		
$2r_0$		0	1	.0	0	0	0	0			set $q_1 = 1$
Add $-D$	+	1	1	.0	1	0					
$r_1 = 2r_0 - D$		0	0	.0	1	0	0	0			
$2r_1$		0	0	.1	0	0	0				set $q_2 = 0$
$r_2 = 2r_1$		0	0	.1	0	0	0				
$2r_2$		0	1	.0	0	0					set $q_3 = 1$
Add $-D$	+	1	1	.0	1	0					
$r_3 = 2r_2 - D$		0	0	.0	1	0					

- Generation of $2r_0$ - no overflow
- An extra bit position in the arithmetic unit needed
- Final result : $Q=(0.101)_2=5/8$
- $R=r_m\, 2^{-m} = r_3 2^{-3} = 1/4 \cdot 2^{-3} = 1/32$
- Quotient and final remainder satisfy
  $X=Q \cdot D + R = 5/8 \cdot 3/4 + 1/32 = 16/32 = 1/2$
- Precise quotient is the infinite binary fraction
  $2/3=0.1010101 \cdots$

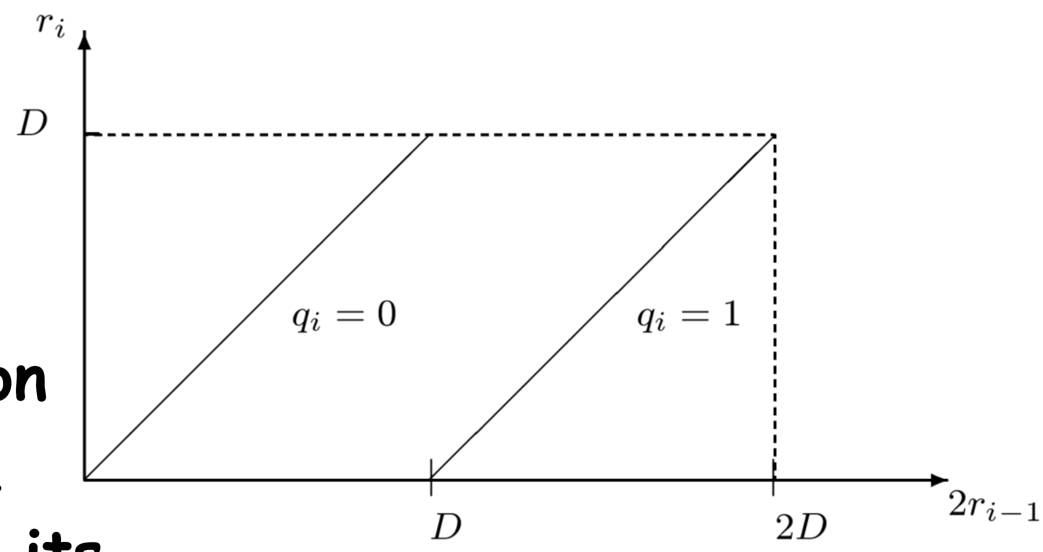
Division Algorithm - Integers

- Same procedure; Previous equation rewritten -

$$2^{2n-2}X_F = 2^{n-1}Q_F \cdot 2^{n-1}D_F + 2^{n-1}R_F$$ (X_F, D_F, Q_F, R_F -fractions)

- Dividing by 2^{2n-2} yields $X_F = Q_F \cdot D_F + 2^{-(n-1)}R_F$
- The condition $X < 2^{n-1} D$ becomes $X_F < D_F$
- $X=0100000_2=32$; $D=0110_2=6$
- Overflow condition $X<2^{n-1}D$ is tested by comparing the most significant half of X, 0100, to D, 0110
- The results of the division are $Q=0101_2=5$ and $R=0010_2=2$
- In final step the true remainder R is generated - no need to further multiply it by $2^{-(n-1)}$

Restoring Division



- Comparison - most difficult step in division
- If $2r_{i-1}-D<0$ - $q_i=0$ - remainder restored to its previous value - restoring division
- Robertson diagram - shows that if $r_{i-1}<D$, q_i is selected so that $r_i<D$
- Since $r_0=X<D$ - $R<D$
- m subtractions, m shift operations, an average of $m/2$ restore operations
  - * can be done by retaining a copy of the previous remainder - avoiding the time penalty

Nonrestoring Division - Remainder

- **Alternative** - quotient bit correction and remainder restoration postponed to later steps
- Restoring method - if $2r_{i-1}-D<0$ - remainder is restored to $2r_{i-1}$
- Then shifted and D again subtracted, obtaining $4r_{i-1}-D$ - process repeated as long as remainder negative
- Nonrestoring - restore operation avoided
- Negative remainder $2r_{i-1}-D<0$ shifted, then corrected by adding D, obtaining $2(2r_{i-1}-D)+D=4r_{i-1}-D$
- Same remainder obtained with restoring division

Nonrestoring Division - Quotient

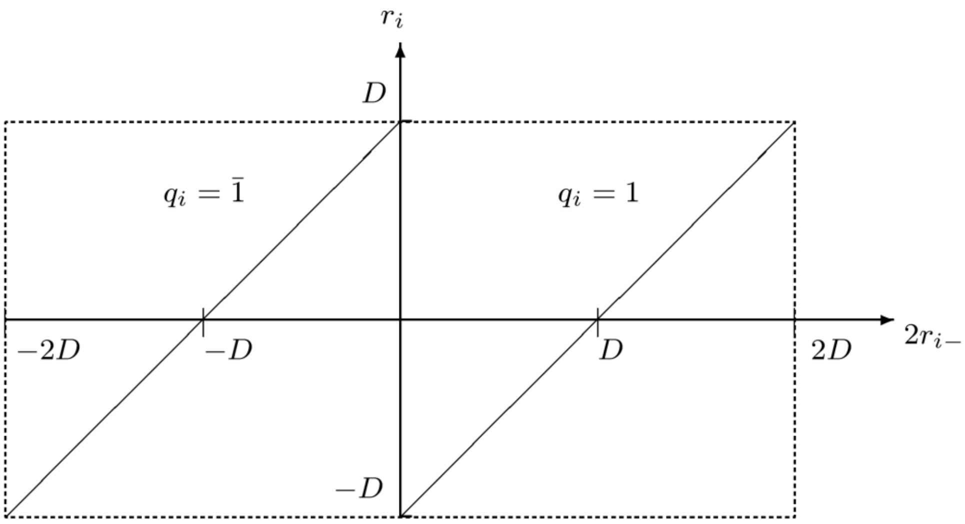
- Correcting a wrong selection of quotient bit in step i - next bit, q_{i+1}, can be negative - $\bar{1}$
- If q_i was incorrectly set to 1 - negative remainder - select $q_{i+1}=\bar{1}$ and add D to remainder
- Instead of $q_i\, q_{i+1}=10$ (too large) - $q_i\, q_{i+1}=1\bar{1}=01$
- Further correction - if needed - in next steps

- Rule for q_i :

$$q_i = \begin{cases} 1 & \text{if } 2r_{i-1} \geq 0 \\ \bar{1} & \text{if } 2r_{i-1} < 0 \end{cases}$$

Nonrestoring Division - Diagram

- Simpler and faster than selection rule for restoring division - $2r_{i-1}$ compared to 0 instead of to D
- Same equation for remainder - $q_i = 2r_{i-1} - q_i D$
- Divisor D subtracted if $2r_{i-1} > 0$, added if < 0
  - * $|r_{i-1}| < D$
  - * q_i selected so $|r_i| < D$
  - * $q_i \neq 0$ - at each step, either addition or subtraction is performed



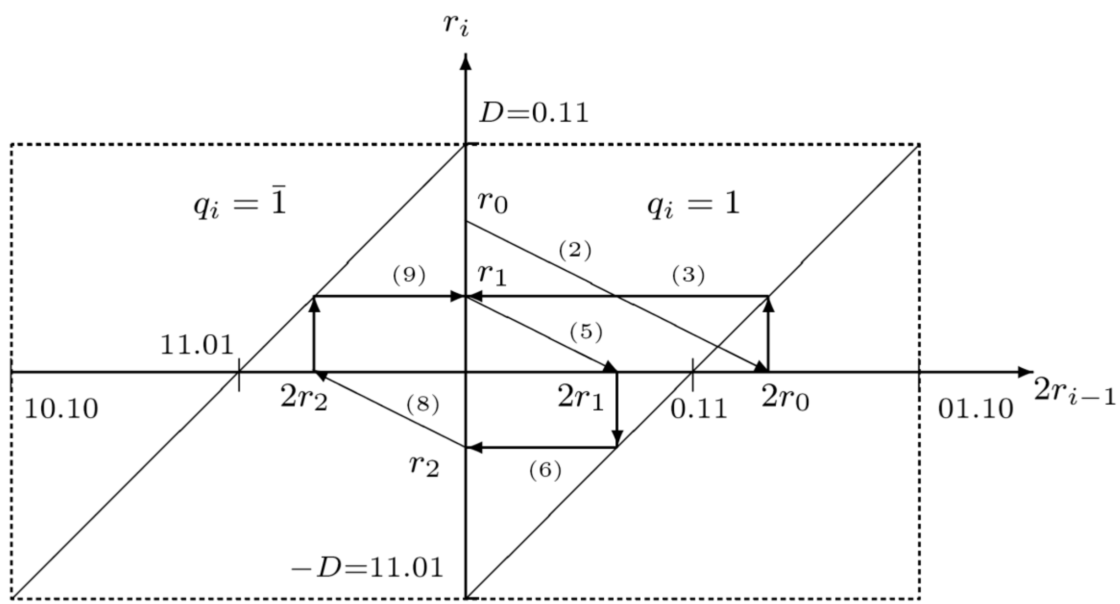
- Not SD representation no redundancy in representation of quotient
- Exactly m add/subtract and shift operations

Nonrestoring Division - Example 1

- $X=(0.100000)_2=1/2$
- $D=(0.110)_2=3/4$
- Final remainder - as before
- $Q=0.11\bar{1}=0.101_2=5/8$

(1)	$r_0 = X$			0	.1	0	0	
(2)	$2r_0$		0	1	.0	0	0	set $q_1 = 1$
(3)	Add $-D$	+	1	1	.0	1	0	
(4)	r_1		0	0	.0	1	0	
(5)	$2r_1$		0	0	.1	0	0	set $q_2 = 1$
(6)	Add $-D$	+	1	1	.0	1	0	
(7)	r_2		1	1	.1	1	0	
(8)	$2r_2$		1	1	.1	0	0	set $q_3 = \bar{1}$
(9)	Add D	+	0	0	.1	1	0	
(10)	r_3		0	0	.0	1	0	

- Graphical representation
  - Horizontal lines - add ±D
  - Diagonal lines - multiply by 2



Nonrestoring Division - Advantage

- Important feature of nonrestoring division - easily extended to two's complement negative numbers
- Generalized selection rule for q_i -

$$q_i = \begin{cases} 1 & \text{if } 2r_{i-1} \text{ and } D \text{ have the same sign} \\ \bar{1} & \text{if } 2r_{i-1} \text{ and } D \text{ have opposite signs} \end{cases}$$

- Remainder changes signs during process - nothing special about a negative dividend X

Nonrestoring Division - Example 2

- $X=(0.100)_2=1/2$
- $D=(1.010)_2=-3/4$

$r_0 = X$			0	.1	0	0	
$2r_0$		0	1	.0	0	0	set $q_1 = \bar{1}$
Add D		1	1	.0	1	0	
r_1		0	0	.0	1	0	
$2r_1$		0	0	.1	0	0	set $q_2 = \bar{1}$
Add D	+	1	1	.0	1	0	
r_2		1	1	.1	1	0	
$2r_2$		1	1	.1	0	0	set $q_3 = 1$
Add $-D$	+	0	0	.1	1	0	
r_3		0	0	.0	1	0	

- Final quotient - $Q=0.\bar{1}\bar{1}1=0.\bar{1}0\bar{1}_2=-0.101_2=-5/8$ $=1.011$ in two's complement
- Final remainder = $1/32$ - same sign as the dividend X

Nonrestoring Division - sign of remainder

- Sign of final remainder - same as dividend
- Example - dividing 5 by 3 - $Q=1$, $R=2$, not $Q=2$, $R=-1$ (although $|R|<D$)
- If sign of final remainder different from that of dividend - correction needed - results from quotient digits being restricted to $1,\bar{1}$
- Last digit can not be 0 - an "even" quotient can not be generated

Nonrestoring Division - Example 3

- $X = 0.110_2 = 5/8$
- $D = 0.110_2 = 3/4$

$r_0 = X$			0	.1	0	1	
$2r_0$		0	1	.0	1	0	set $q_1 = 1$
Add $-D$	$+$	1	1	.0	1	0	
r_1		0	0	.1	0	0	
$2r_1$		0	1	.0	0	0	set $q_2 = 1$
Add $-D$	$+$	1	1	.0	1	0	
r_2		0	0	.0	1	0	
$2r_2$		0	0	.1	0	0	set $q_3 = 1$
Add $-D$	$+$	1	1	.0	1	0	
r_3		1	1	.1	1	0	

- **Final remainder negative - dividend positive**
- **Correct final remainder by adding D to r_3 - 1.110+0.110=0.100**
- **Correct quotient - $Q_{corrected} = Q - ulp$**
- **Q=0.111 - $Q_{corrected} = 0.110_2 = 3/4$**

Nonrestoring Division - Cont.

- If final remainder and dividend have opposite signs - correction needed
- If dividend and divisor have same sign - remainder r_m corrected by adding D and quotient corrected by subtracting ulp
- If dividend and divisor have opposite signs - subtract D from r_m and correct quotient by adding ulp
- Another consequence of the fact that 0 is not an allowed digit in non-restoring division - need for correction if a 0 remainder is generated in an intermediate step

Nonrestoring Division - Example 4

$r_0 = X$			1	.1	0	1	
$2r_0$		1	1	.0	1	0	set $q_1 = \bar{1}$
Add D	$+$	0	0	.1	1	0	
r_1		0	0	.0	0	0	zero remainder
$2r_1$		0	0	.0	0	0	set $q_2 = 1$
Add $-D$	$+$	1	1	.0	1	0	
r_2		1	1	.0	1	0	
$2r_2$		1	0	.1	0	0	set $q_3 = \bar{1}$
Add D	$+$	0	0	.1	1	0	
r_3		1	1	.0	1	0	

- $X=1.101_2=-3/8$
- $D=0.110_2=3/4$
- Correct result of division - $Q=-1/2; \; R=0$
- Although final remainder and dividend have same sign - correction needed due to a zero intermediate remainder
- This must be detected and corrected -
- $r_{3(\text{corrected})} = r_3+D=1.010+0.110=0.000$
- Correcting the quotient $Q=0.\bar{1}1\bar{1}=0.\bar{1}01$ by subtracting ulp : $Q(\text{corrected}) = 0.\bar{1}00_2=-1/2$

Generating a Two's Complement Quotient in Nonrestoring Division

- Converting from using $1,\bar{1}$ to two's complement
- Previous algorithms require all digits of quotient before conversion starts - increasing execution time
- Preferable - conversion on the fly - serially from most to least significant digit as they become available
- Quotient digit assumes two values - single bit sufficient for representation - 0 and 1 assigned to $\bar{1}$ and 1
- Resulting binary number - $0.p_1...p_m$ ($p_i=1/2(q_i+1)$)

Conversion Algorithm

- **Step 1**: Shift number one bit position to left
- **Step 2**: Complement most significant bit
- **Step 3**: Shift a 1 into least significant position

- **Result** - $(1-p_1).p_2p_3...p_m1$ - has same numerical value as original quotient Q
- **Proof**: Value of above sequence in two's complement -

$$-(1-p_1)2^0 + \sum_{i=2}^{m} p_i 2^{-i+1} + 2^{-m}$$

- Substituting $p_i=1/2(q_i+1)$ -

$$q_1 2^{-1} - 2^{-1} + \sum_{i=2}^{m}(q_i+1)2^{-i} + 2 = q_1 2^{-1} - (2^{-1} - 2^{-m}) + \sum_{i=2}^{m} q_i 2^{-i} + \sum_{i=2}^{m} 2^{-i}.$$

- Last term = $2^{-1} - 2^{-m}$

$$= q_1 2^{-1} + \sum_{i=2}^{m} q_i 2^{-i} = \sum_{i=1}^{m} q_i 2^{-i} = Q$$

Conversion Algorithm - Example

- Algorithm can be executed in a bit-serial fashion
- Example - X=1.101 ; D=0.110
- Instead of generating the quotient bits $.\bar{1}1\bar{1}$ - generate the bits (1-0).101=1.101
- After correction step - Q-ulp=1.100 - correct representation of -1/2 in two's complement
- Exercise - The same on-the-fly conversion algorithm can be derived from the general SD to two's complement conversion algorithm presented before

Square Root Extraction - Restoring

- The conventional completing the square method for square root extraction is conceptually similar to restoring division
- X - the radicant - a positive fraction ; $Q=(0.q_1\ q_2 \ldots q_m)$ - its square root
- The bits of Q generated in m steps - one per step
- $Q_i = \sum_{k=1}^{i} q_k 2^{-k}$ - partially developed root at step i

  ($Q_m=Q$) ; r_i - remainder in step i
- Calculation of next remainder -

$$r_i = 2r_{i-1} - q_i \cdot (2Q_{i-1} + q_i 2^{-i}).$$

- Square root extraction can be viewed as division with a changing divisor - $\hat{D}_i = (2Q_{i-1} + q_i 2^{-i})$

Square Root Extraction - Cont.

- First step - remainder=radicand X ; $Q_0=0$
- Performed calculation -

$$r_1 = 2r_0 - q_1(0 + q_1 2^{-1}) = 2X - q_1(0 + q_1 2^{-1})$$

- To determine q_i in the restoring scheme - calculate a tentative remainder

$$2r_{i-1} - (2Q_{i-1} + 2^{-i})$$

- $q_1.q_2 \ldots q_{i-1}01 = 2Q_{i-1}+2^{-i}$ - simple to calculate
- If tentative remainder positive - its value is stored in r_i and $q_i=1$
- Otherwise - $r_i=2r_{i-1}$ and $q_i=0$

Proof of Algorithm

- Repeated substitution in the expression for r_m -

$$
\begin{aligned}
r_m &= 2r_{m-1} - q_m(2Q_{m-1} + q_m 2^{-m}) \\
&= 2^2 r_{m-2} - 2q_{m-1}(2Q_{m-2} + q_{m-1}) - q_m(2Q_{m-1} + q_m 2^{-m}) \\
&\;\vdots \\
&= 2^m \cdot r_0 - 2^m \left[(q_1 2^{-1})^2 + (q_2 2^{-2})^2 \; + \cdots + (q_m 2^{-m})^2\right] \\
&\quad -2^m \left[2q_2 2^{-2} q_1 2^{-1} + \cdots + 2q_m 2^{-m} \sum_{i=1}^{m-1} q_i 2^{-i}\right] \\
&= 2^m X - 2^m \left(\sum_{i=1}^{m} q_i 2^{i}\right)^2 = 2^m (X - Q^2).
\end{aligned}
$$

- Dividing by 2^m results in the expected relation with $r_m 2^{-m}$ as the final remainder

Example - Square root (Restoring)

- $X=0.1011_2=11/16=176/256$

$r_0 = X$			0	.1	0	1	1	
$2r_0$		0	1	.0	1	1	0	
$-(0+2^{-1})$	$-$	0	0	.1	0	0	0	
r_1		0	0	.1	1	1	0	set $q_1=1,\ Q_1=0.1$
$2r_1$		0	1	.1	1	0	0	
$-(2Q_1+2^{-2})$	$-$	0	1	.0	1	0	0	
r_2		0	0	.1	0	0	0	set $q_2=1,\ Q_2=0.11$
$2r_2$		0	1	.0	0	0	0	is smaller than $(2Q_2+2^{-3})$ $=1.101$
$r_3 = r_2$		0	1	.0	0	0	0	set $q_3=0,\ Q_3=0.110$
$2r_3$		1	0	.0	0	0	0	still a positive number
$-(2Q_3+2^{-4})$	$-$	0	1	.1	0	0	1	
r_4		0	0	.0	1	1	1	set $q_4=1,\ Q_4=0.1101$

- $Q=0.1101_2=13/16$
- Final remainder$=2^{-4}r_4=7/256=X-Q^2=(176-169)/256$

Different Algorithm - Nonrestoring

- **Second algorithm - similar to nonrestoring division**

$$q_i = \begin{cases} 1 & \text{if } 2r_{i-1} \geq 0 \\ \bar{1} & \text{if } 2r_{i-1} < 0 \end{cases}$$

- **Example -**
  - * $X=0.011001_2$ $=25/64$
- **Square root -**
  - * $Q=0.11\bar{1}$ $=0.101_2=5/8$

$r_0 = X$			0	.0	1	1	0	0	1	
$2r_0$			0	.1	1	0	0	1	0	set $q_1=1,\ Q_1=0.1$
$-(0+2^{-1})$	$-$		0	.1	0	0	0	0	0	
r_1			0	.0	1	0	0	1	0	
$2r_1$			0	.1	0	0	1	0	0	set $q_2=1,\ Q_2=0.11$
$-(2Q_1+2^{-2})$	$-$	0	1	.0	1	0	0	0	0	
r_2		1	1	.0	1	0	1	0	0	
$2r_2$		1	0	.1	0	1	0	0	0	set $q_3=\bar{1},\ Q_3=0.11\bar{1}$
$+(2Q_2-2^{-3})$	$+$	0	1	.1	0	$\bar{1}$	0	0	0	
r_3		0	0	.0	0	0	0	0	0	

- **Converting the digits of Q to two's complement representation - similarly to nonrestoring division**
- **Faster algorithms for square root extraction exist**