Fixed-Point Arithmetic

Fixed-Point Notation

- A K-bit fixed-point number can be interpreted as either:
  - an integer (i.e., 20645)
  - a fractional number (i.e., 0.75)

Integer Fixed-Point Representation

- N-bit fixed point, 2's complement integer representation

$X = -b_{N-1} 2^{N-1} + b_{N-2} 2^{N-2} + \ldots + b_0 2^0$

- Difficult to use due to possible overflow
  - In a 16-bit processor, the dynamic range is -32,768 to 32,767.
    - Example: 200 × 350 = 70000, which is an overflow!

3

Fractional Fixed-Point Representation

- Also called Q-format
- Fractional representation suitable for DSP algorithms.
- Fractional number range is between 1 and -1
- Multiplying a fraction by a fraction always results in a fraction and will not produce an overflow (e.g., 0.99 x 0.9999 less than 1)
- Successive additions may cause overflow
- Represent numbers between
  - -1.0 and $1 - 2^{-(N-1)}$, when N is number of bits

4

Fractional Fixed-Point Representation

- Equivalent to scaling
- Q represents the "Quantity of fractional bits"
- Number following the Q indicates the number of bits that are used for the fraction.
- Q15 used in 16-bit DSP chip, resolution of the fraction will be 2^–15 or 30.518e–6
  - Q15 means scaling by $1/2^{15}$
  - Q15 means shifting to the right by 15
- Example: how to represent 0.2625 in memory:
  - Method 1 (Truncation): INT[$0.2625*2^{15}$]= INT[8601.6] = 8601 = 0010000110011001
  - Method 2 (Rounding): INT[$0.2625*2^{15}+0.5$]= INT[8602.1] = 8602 = 0010000110011010

5

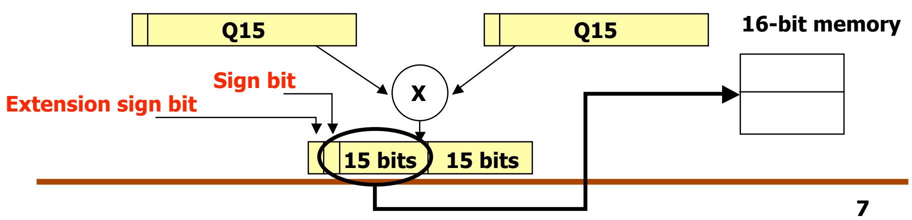
Truncating or Rounding?

- Which one is better?
- Truncation
  - Magnitude of truncated number always less than or equal to the original value
    - Consistent downward bias
- Rounding
  - Magnitude of rounded number could be smaller or greater than the original value
    - Error tends to be minimized (positive and negative biases)
  - Popular technique: rounding to the nearest integer
- Example:
  - INT[251.2] = 251 (Truncate or floor)
  - ROUND [251.2] = 252 (Round or ceil)
  - ROUNDNEAREST [251.2] = 251

6

Q format Multiplication

- Product of two Q15 numbers is Q30.
- So we must remember that the 32-bit product has *two bits* in front of the binary point.
  - Since NxN multiplication yields 2N-1 result
  - Addition MSB sign extension bit
- Typically, only the most significant 15 bits (plus the sign bit) are stored back into memory, so the *write operation requires a left shift by one*.



General Fixed-Point Representation

- Qm.n notation
  - m bits for integer portion
  - n bits for fractional portion
  - Total number of bits N = m + n + 1, for signed numbers
  - Example: 16-bit number (N=16) and Q2.13 format
    - 2 bits for integer portion
    - 13 bits for fractional portion
    - 1 signed bit (MSB)
  - Special cases:
    - 16-bit integer number (N=16) => Q15.0 format
    - 16-bit fractional number (N = 16) => Q0.15 format; also known as Q.15 or Q15

General Fixed-Point Representation

- N-bit number in Qm.n format:

$$\underbrace{b_{n+m}}_{N-1} b_{n+m-1} \ldots b_n . b_{n-1} \ldots b_1 b_o$$

Fixed Point

- Value of N-bit number in Qm.n format:

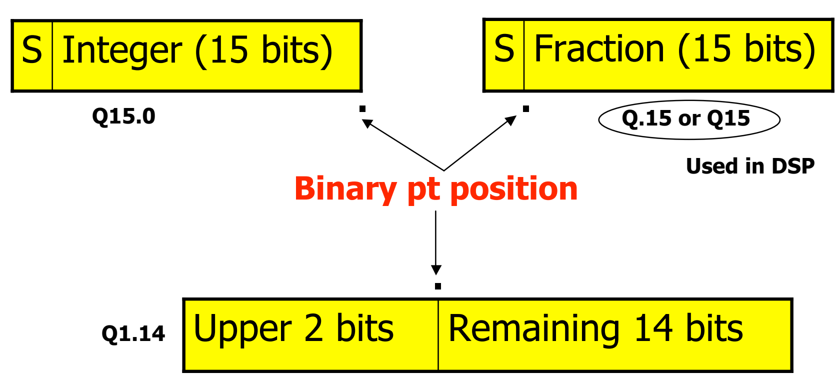
$$(-b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + b_{N-3}2^{N-3} + \ldots + b_1 2 + b_o)/2^n$$

$$= (-b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + b_{N-3}2^{N-3} + \ldots + b_1 2 + b_o)2^{-n}$$

$$= -b_{N-1}2^m + \sum_{l=0}^{N-2} b_l 2^{l-n}$$

9

Some Fractional Examples (16 bits)



10

How to Compute Fractional Number

Q m.n Format

$b'_s b'_{m-1} \ldots b'_0$	$b_{n-1} b_{n-2} \ldots b_0$

$$-2^m b'_s + \ldots + 2^1 b'_1 + 2^0 b'_0 + 2^{-1} b_{n-1} + 2^{-2} b_{n-2} \ldots + 2^{-n} b_0$$

Examples:

- 1110 Integer Representation Q3.0: $-2^3 + 2^2 + 2^1 = -2$
- 11.10 Fractional Q1.2 Representation: $-2^1 + 2^0 + 2^{-1} = -2 + 1 + 0.5 = -0.5$

  (Scaling by $1/2^2$)
- 1.110 Fractional Q3 Representation: $-2^0 + 2^{-1} + 2^{-2} = -1 + 0.5 + 0.25 = -0.25$ (Scaling by $1/2^3$)

General Fixed-Point Representation

Min and Max Decimal Values of Integer and Fractional 4-Bit Numbers (Kuo & Gan)

Unsigned integer	Signed integer
Smallest value: 0000 = (0) Largest value: 1111 = (15)	Most positive value: 0111 = (+7) Least negative value: 1000 = (−8)
Unsigned fractional	**Signed fractional**
Smallest value: .0000 = (0) Largest value: .1111 = (0.9375)	Most positive value: 0.111 = (+0.875) Least negative value: 1.000 = (−1)

General Fixed-Point Representation

- Dynamic Range
  - Ratio between the largest number and the smallest (positive) number
  - It can be expressed in dB (decibels) as follows:
    Dynamic Range (dB) = $20\log_{10}(Max/Min)$
  - Note: Dynamic Range depends only on N
    - N-bit Integer (Q(N-1).0):
      Min = 1; Max = 2^{N-1} - 1 => Max/Min = 2^{N-1} - 1
    - N-bit fractional number (Q(N-1)):
      Min = $2^{-(N-1)}$; Max = 1-$2^{-(N-1)}$ => Max/Min = 2^{N-1} – 1
    - General N-bit fixed-point number (Qm.n)
      => Max/Min = 2^{N-1} – 1

General Fixed-Point Representation

Dynamic Range and Precision of Integer and Fractional 16-Bit Numbers (Kuo & Gan)

	Dynamic range	Dynamic range in dB	Precision
Unsigned integer	0 to 65,536	$20\log_{10}(2^{16}) = 96$ dB	1
Signed integer	−32,768 to 32,767	$20\log_{10}(2^{15}) = 90$ dB	1
Unsigned fractional	0 to 0.99998474	96 dB	2^{-16}
Signed fractional	−1 to 0.99996948	90 dB	2^{-15}

General Fixed-Point Representation

- Precision
  - Smallest step (difference) between two consecutive N-bit numbers.
    Example:
    Q15.0 (integer) format => precision = 1
    Q15 format => precision = 2^{-15}
  - Tradeoff between dynamic range and precision
    Example: N = 16 bits
    Q15.0 => widest dynamic range (-32,768 to 32,767); worst precision (1)
    Q15 => narrowest dynamic range (-1 to $+1^{-}$); best precision (2^{-15})

15

General Fixed-Point Representation

Dynamic Range and Precision of 16-Bit Numbers for Different Q Formats (Kuo & Gan)

Format	Largest positive value	Least negative value	Precision
Q0.15	0.999969482421875	−1	0.00003051757813
Q1.14	1.99993896484375	−2	0.00006103515625
Q2.13	3.9998779296875	−4	0.00012207031250
Q3.12	7.999755859375	−8	0.00024414062500
Q4.11	15.99951171875	−16	0.00048828125000
Q5.10	31.9990234375	−32	0.00097656250000
Q6.9	63.998046875	−64	0.00195312500000
Q7.8	127.99609375	−128	0.00390625000000
Q8.7	255.9921875	−256	0.00781250000000
Q9.6	511.984375	−512	0.01562500000000
Q10.5	1023.96875	−1,024	0.03125000000000
Q11.4	2047.9375	−2,048	0.06250000000000
Q12.3	4095.875	−4,096	0.12500000000000
Q13.2	8191.75	−8,192	0.25000000000000
Q14.1	16383.5	−16,384	0.50000000000000
Q15.0	32,767	−32,768	1.00000000000000

16

General Fixed-Point Representation

Scaling Factor and Dynamic Range of 16-Bit Numbers (Kuo & Gan)

Format	Scaling factor (2^n)	Range in Hex (Decimal value)
Q0.15	$2^{15} = 32{,}768$	7FFFh (0.99) → 8000h (−1)
Q1.14	$2^{14} = 16{,}384$	7FFFh (1.99) → 8000h (−2)
Q2.13	$2^{13} = 8{,}192$	7FFFh (3.99) → 8000h (−4)
Q3.12	$2^{12} = 4{,}096$	7FFFh (7.99) → 8000h (−8)
Q4.11	$2^{11} = 2{,}048$	7FFFh (15.99) → 8000h (−16)
Q5.10	$2^{10} = 1{,}024$	7FFFh (31.99) → 8000h (−32)
Q6.9	$2^{9} = 512$	7FFFh (63.99) → 8000h (−64)
Q7.8	$2^{8} = 256$	7FFFh (127.99) → 8000h (−128)
Q8.7	$2^{7} = 128$	7FFFh (511.99) → 8000h (−512)
Q9.6	$2^{6} = 64$	7FFFh (1023.99) → 8000h (−1,024)
Q10.5	$2^{5} = 32$	7FFFh (2047.99) → 8000h (−2,048)
Q11.4	$2^{4} = 16$	7FFFh (4095.99) → 8000h (−4,096)
Q12.3	$2^{3} = 8$	7FFFh (4095.99) → 8000h (−4,096)
Q13.2	$2^{2} = 4$	7FFFh (8191.99) → 8000h (−8,192)
Q14.1	$2^{1} = 2$	7FFFh (16383.99) → 8000h (−16,384)
Q15.0	$2^{0} = 1$(Integer)	7FFFh (32,767) → 8000h (−32,768)

General Fixed-Point Representation

- Fixed-point DSPs use 2's complement fixed-point numbers in different Q formats
- Assembler only recognizes integer values
  - Need to know how to convert fixed-point number from a Q format to an integer value that can be stored in memory and that can be recognized by the assembler.
  - Programmer must keep track of the position of the binary point when manipulating fixed-point numbers in asembly programs.

How to convert fractional number into integer

- Conversion from fractional to integer value:
  - Step 1: normalize the decimal fractional number to the range determined by the desired Q format
  - Step 2: Multiply the normalized fractional number by 2^n
  - Step 3: Round the product to the nearest integer
  - Step 4: Write the decimal integer value in binary using N bits.
- Example:

Convert the value 3.5 into an integer value that can be recognized by a DSP assembler using the Q15 format => 1) Normalize: 3.5/4 = 0.875;

2) Scale: $0.875*2^{15}$= 28,672; 3) Round: 28,672

19

How to convert integer into fractional number

- Numbers and arithmetic results are stored in the DSP processor in integer form.
- Need to interpret as a fractional value depending on Q format
- Conversion of integer into a fractional number for Qm.n format:
  - Divide integer by scaling factor of Qm.n => divide by 2^n
- Example:

Which Q15 value does the integer number 2 represent? $2/2^{15}=2*2^{-15}=2^{-14}$

20

Finite-Wordlength Effects

- Wordlength effects occur when wordlength of memory (or register) is less than the precision needed to store the actual values.
- Wordlength effects introduce noise and non-ideal system responses
- Examples:
  - Quantization noise due to limited precision of Analog-to-Digital (A/D) converter, also called codec
  - Limited precision in representing input, filter coefficients, output and other parameters.
  - Overflow or underflow due to limited dynamic range
  - Roundoff/truncation errors due to rounding/truncation of double-precision data to single-precision data for storage in a register or memory.
    - Rounding results in an unbiased error; truncation results in a biased error => rounding more used in practice.

21

Multiplication & Division

Fast Multiplication

- What do we do?
  - Let Verilog do it: Write a= b * c
  - Design fast multiplier circuit
  - Use built-in hardware multipliers

Fast Division

- More difficult problem-- no hardware divider
- Traditional division is slow
- So, what to do?

Fast Division

- Find alternative solutions:
  - Multiply by the reciprocal : A / D = A * 1 / D
    - Great for constants
    - Use Newton’s method for calculation of the reciprocal of D
  - Pipeline and use a slow algorithm (next time)
  - Speed up the slower algorithms

Newton-Raphson division

Newton-Raphson uses Newton's method to converge to the quotient.

The strategy of Newton-Raphson is to find the reciprocal of D, and multiply that reciprocal by N to find the final quotient Q.

Newton-Raphson division

The steps of Newton-Raphson are:

1. Calculate an estimate for the reciprocal of the divisor *(D)*: X_0

2. Compute successively more accurate estimates of the reciprocal: $(X_1, \ldots, X_k)$

3. Compute the quotient by multiplying the dividend by the reciprocal of the divisor: $Q = NX_k$

27

Newton's method to find reciprocal of D

- find a function f(X) which has a zero at X = 1 / D
- a function which works is f(X) = 1 / X − D
- The Newton-Raphson iteration gives:

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} = X_i - \frac{1/X_i - D}{-1/X_i^2} = X_i + (X_i - DX_i^2) = X_i(2 - DX_i)$$

- which can be calculated from Xi using only multiplication and subtraction.
- Google for more details

28

Division Overview

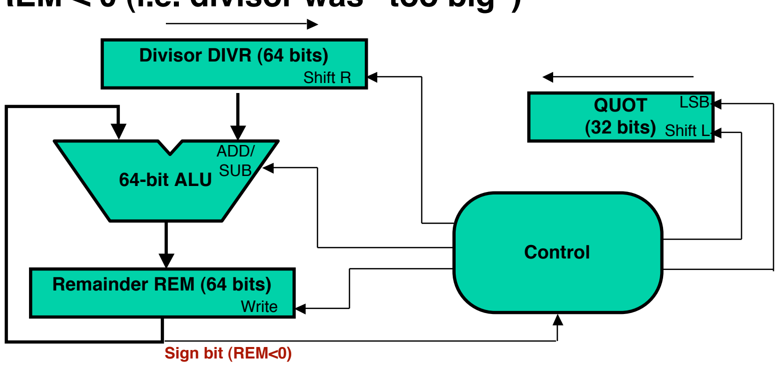
- Grade school algorithm: long division
  - Subtract shifted divisor from dividend when it "fits"
  - Quotient bit: 1 or 0
- Question: how can hardware tell "when it fits?"

```
                   1001    Quotient
  Divisor 1000 |1001010    Dividend
               -1000
                   1010
                  -1000
                     10    Remainder
```

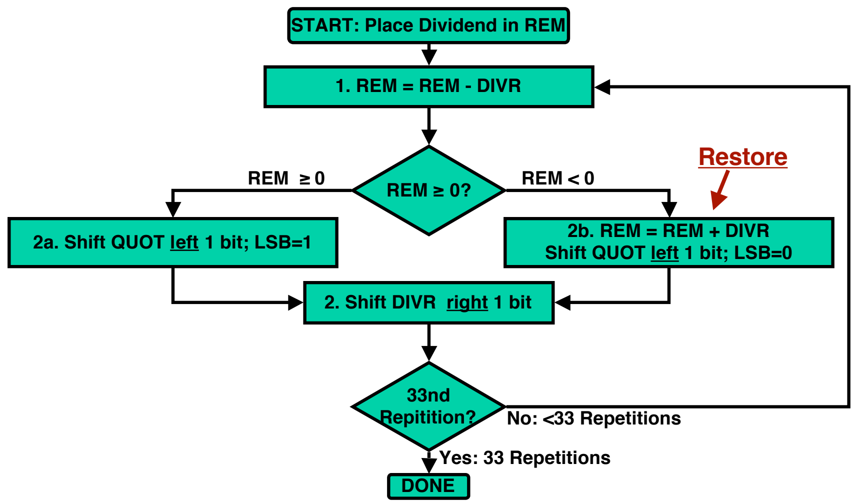
`Dividend = Quotient X Divisor + Remainder`

Division Hardware - 1st Version

- Shift register moves divisor (DIVR) to right
- ALU subtracts DIVR, then <u>restores</u> (adds back) if REM < 0 (i.e. divisor was "too big")



Division Algorithm - First Version



31

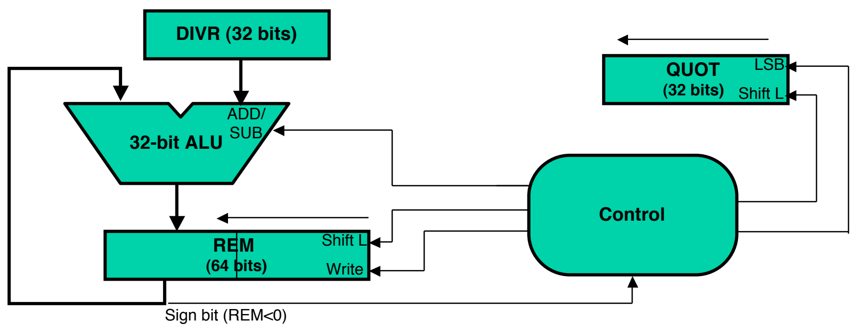
Divide 1st Version - Observations

- We only subtract 32 bits in each iteration
  - Idea: Instead of shifting divisor to right, shift remainder to left
- First step cannot produce a 1 in quotient bit
  - Switch order to shift first, then subtract
  - Save 1 iteration

32

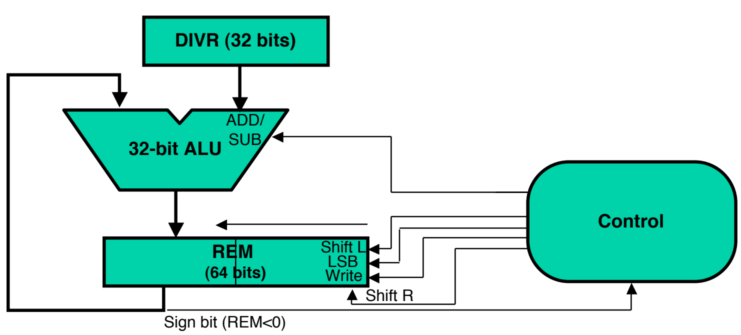
Divide Hardware - 2nd Version

- Divisor Holds Still
- Dividend/Remainder Shifts Left
- End Result: Remainder in upper half of register

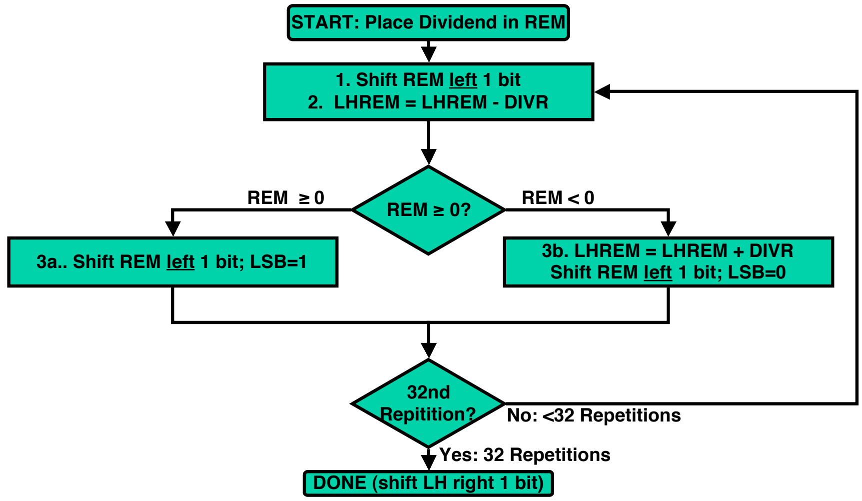


Divide Hardware - 3rd Version

- Combine quotient with remainder register



Divide Algorithm - 3rd Version



35

Dividing Signed Numbers

- Check sign of divisor, dividend
- Negate quotient if signs of operands are opposite
- Make remainder sign match dividend (if nonzero)

36

Fast Division - SRT Algorithm

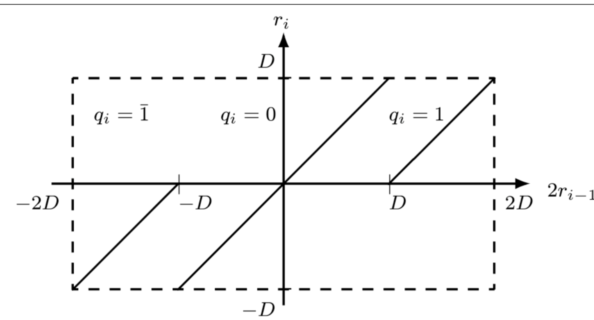
- **2 approaches**:
  - First - conventional - uses add/subtract+shift, number of operations linearly proportional to word size n
  - Second - uses multiplication, number of operations logarithmic in n, but each step more complex
  - SRT - first approach
- **Most well known division algorithm - named after Sweeney, Robertson, and Tocher**
- **Speed up nonrestoring division (n add/subtracts) - allows 0 as a quotient digit - no add/subtract:**

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

$$r_i = 2r_{i-1} - q_i \cdot D$$

37

Modified Nonrestoring Division



- **Problem**: full comparison of $2r_{i-1}$ with either D or -D required
- **Solution**: restricting D to normalized fraction $1/2 \leq |D| < 1$
- **Region of $2r_{i-1}$ for which $q_i=0$ reduced to**

$$-D \leq -\frac{1}{2} \leq 2r_{i-1} < \frac{1}{2} \leq D$$

38

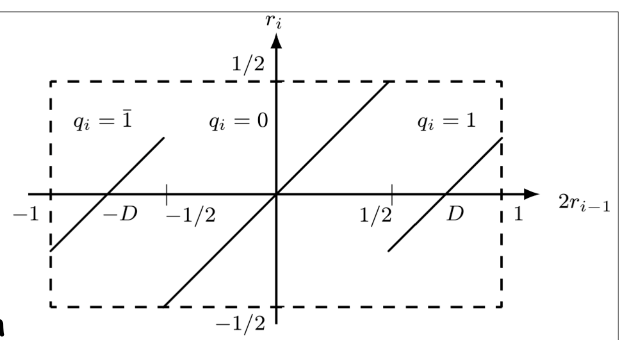
Modified Nonrestoring → SRT

- **Advantage:** Comparing partial remainder $2r_{i-1}$ to 1/2 or -1/2, not D or -D
- Binary fraction in two's complement representation
  - * ≥ 1/2 if and only if it starts with 0.1
  - * ≤ -1/2 if and only if it starts with 1.0
- Only 2 bits of $2r_{i-1}$ examined - not full comparison between $2r_{i-1}$ and D
  - * In some cases (e.g., dividend X>1/2) - shifted partial remainder needs an integer bit in addition to sign bit - 3 bits of $2r_{i-1}$ examined
- Selecting quotient digit:

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

39

SRT Division Algorithm



- Quotient digits selected so $|r_i| \leq |D|$ ⇒ final remainder < |D|
- Process starts with normalized divisor - normalizing partial remainder by shifting over leading 0's/1's if positive/negative
- **Example:**
  - * $2r_{i-1}$=0.001xxxx (x - 0/1); $2r_{i-1}$<1/2 - set q_i=0, $2r_i$=0.01xxxx and so on
  - * $2r_{i-1}$=1.110xxxx; $2r_{i-1}$>-1/2 - set q_i=0, $2r_i$=1.10xxxx
- SRT is nonrestoring division with normalized divisor and remainder

40

Extension to Negative Divisors

$$q_i = \begin{cases} 0 & \text{if } |2r_{i-1}| < 1/2 \\ 1 & \text{if } |2r_{i-1}| \geq 1/2 \ \& \ r_{i-1} \text{ and } D \text{ have the same sign} \\ \bar{1} & \text{if } |2r_{i-1}| \geq 1/2 \ \& \ r_{i-1} \text{ and } D \text{ have opposite signs} \end{cases}$$

- **Example:** Dividend X=(0.0101)2 =5/16 Divisor D=(0.1100)2 =3/4

$r_0 = X$		0	.0	1	0	1	
$2r_0$		0	.1	0	1	0	$\geq 1/2$ set $q_1 = 1$
Add $-D$	+	1	.0	1	0	0	
r_1		1	.1	1	1	0	
$2r_1 = r_2$		1	.1	1	0	0	$\geq -1/2$ set $q_2 = 0$
$2r_2 = r_3$		1	.1	0	0	0	$\geq -1/2$ set $q_3 = 0$
$2r_3$		1	.0	0	0	0	$< -1/2$ set $q_4 = \bar{1}$
Add D	+	0	.1	1	0	0	
r_4		1	.1	1	0	0	negative remainder & positive X
Add D	+	0	.1	1	0	0	correction
r_4		0	.1	0	0	0	corrected final remainder

- **Before correction** Q=0.100$\bar{1}$ - minimal SD repr. of Q=0.0111 - minimal number of add/subtracts
- **After correction,** Q = 0.0111-ulp = 0.0110$_2$ = 3/8 ; final remainder = $1/2 \cdot 2^{-4}$ =1/32

41

Example

- X=(0.00111111)2=63/256 D=(0.1001)2=9/16

$r_0 = X$		0	.0	0	1	1	1	1	1	1	
$2r_0$		0	.0	1	1	1	1	1	1	0	$< 1/2$ set $q_1 = 0$
$2r_1$		0	.1	1	1	1	1	1	0	0	$\geq 1/2$ set $q_2 = 1$
Add $-D$	+	1	.0	1	1	1					
r_2		0	.0	1	1	0	1	1	0	0	
$2r_2$		0	.1	1	0	1	1	0	0	0	$\geq 1/2$ set $q_3 = 1$
Add $-D$	+	1	.0	1	1	1					
r_3		0	.0	1	0	0	1	0	0	0	
$2r_3$		0	.1	0	0	1	0	0	0	0	$\geq 1/2$ set $q_4 = 1$
Add $-D$	+	1	.0	1	1	1					
r_4		0	.0	0	0	0	0	0	0	0	zero final remainder

- Q =0.0111$_2$=7/16 - not a minimal representation in SD form
- Conclusion: Number of add/subtracts can be reduced further

42

Properties of SRT

- Based on simulations and analysis:
- 1.Average "shift"=2.67 - n/2.67 operations for dividend of length n
  - 24/2.67 ~ 9 operations on average for n=24
- 2.Actual number of operations depends on divisor D - smallest when $17/28 \leq D \leq 3/4$ - average shift of 3
- If D out of range ($3/5 \leq D \leq 3/4$) - SRT can be modified to reduce number of add/subtracts
- 2 ways to modify SRT

Two Modifications of SRT

- Scheme 1: In some steps during division -
  - If D too small - use a multiple of D like 2D
  - If D too large - use D/2
  - Subtracting 2D (D/2) instead of D - equivalent to performing subtraction one position earlier (later)
- Motivation for Scheme 1:
  - Small D may generate a sequence of 1's in quotient one bit at a time, with subtract operation per each bit
  - Subtracting 2D instead of D (equivalent to subtracting D in previous step) may generate negative partial remainder, generating sequence of 0's as quotient bits while normalizing partial remainder
- Scheme 2: Change comparison constant K=1/2 if D outside optimal range - allowed because ratio D/K matters - partial remainder compared to K not D

Example - Scheme 1 (Using 2D)

- Same as previous example -
- $X=(0.00111111)_2=63/256$ $D=(0.1001)_2=9/16$

$r_0 = X$			0	.0	0	1	1	1	1	1	1	
$2r_0$			0	.0	1	1	1	1	1	1	0	$< 1/2$ set $q_1 = 0$
$2r_1$		0	0	.1	1	1	1	1	1	0	0	subtract $2D$
Add $-2D$	+	1	0	.1	1	1						instead of D
r_2		1	1	.1	1	0	1	1	1	0	0	set $q_1 = 1$ and $q_2 = 0$
$2r_2$			1	.1	0	1	1	1	0	0	0	set $q_3 = 0$
$2r_3$			1	.0	1	1	1	0	0	0	0	$\leq -1/2$ set $q_4 = \bar{1}$
Add D	+		0	.1	0	0	1					
r_4			0	.0	0	0	0	0	0	0	0	zero final remainder

- $Q = 0.100\bar{1}_2 = 7/16$ - minimal SD representation

Scheme 1 (Using D/2)

- Large D - one 0 in sequence of 1's in quotient may result in 2 consecutive add/subtracts instead of one
- Adding D/2 instead of D for last 1 before the single 0 - equivalent to performing addition one position later - may generate negative partial remainder
- Allows properly handling single 0
- Then continue normalizing partial remainder until end of sequence of 1's

Example

- $X=(0.01100)_2=3/8$; $D=(0.11101)_2=29/32$
- Correct 5-bit quotient - $Q=(0.01101)_2=13/32$
- Applying basic SRT algorithm - $Q=0.10\bar{1}1\bar{1}$ - single 0 not handled efficiently
- Using multiple D/2 -

$r_0 = X$		0	.0	1	1	0	0		
$2r_0$		0	.1	1	0	0	0		$\geq 1/2$ set $q_1 = 1$
Add $-D$	+	1	.0	0	0	1	1		
r_1		1	.1	1	0	1	1		
$2r_1$		1	.1	0	1	1	0		set $q_2 = 0$
$2r_2$		1	.0	1	1	0	0	0	add $D/2$ $q_3 = \bar{1}$)
Add $D/2$	+	0	.0	1	1	1	0	1	instead of D
r_3		1	.1	1	0	1	0	1	set $q_3 = 0$ and
$2r_3$		1	.1	0	1	0	1		$q_4 = \bar{1}$
$2r_4$		1	.0	1	0	1	0		$\leq -1/2$ set $q_5 = \bar{1}$
Add D	+	0	.1	1	1	0	1		
r_5		0	.0	0	1	1	1		final remainder $= 7/32 \cdot 2^{-5}$

- $Q=(0.100\bar{1}\bar{1})_2=13/32$ - single 0 handled properly

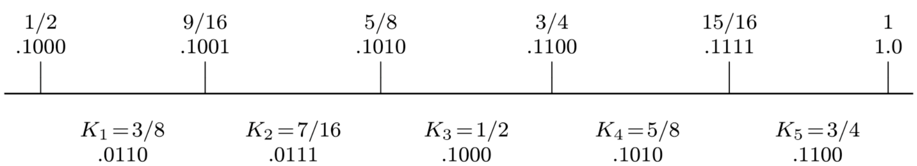
Implementing Scheme 1

- Two adders needed
  - One to add or subtract D
  - Second to add/subtract 2D if D too small (starts with 0.10 in its true form) or add/subtract D/2 if D too large (starts with 0.11)
- Output of primary adder used, unless output of alternate adder has larger normalizing shift
- Additional multiples of D possible - 3D/2 or 3D/4
- Provide higher overall average shift - about 3.7 - but more complex implementation

Modifying SRT - Scheme 2

- For $K=1/2$, ratio D/K in optimal range $3/5 \le D \le 3/4$ is
  $6/5 \le D/K = D/(1/2) \le 3/2$ or
  $(6/5)K \le D \le (3/2)K$
- If D not in optimal range for $K=1/2$ - choose a different comparison constant K
- Region $1/2 \le |D| < 1$ can be divided into 5 (not equally sized) sub-regions
- Each has a different comparison constant K_i

Division into Sub-regions



- 4 bits of divisor examined for selecting comparison constant
- It has only 4 bits compared to 4 most significant bits of remainder
- Determination of sub-regions for divisor and comparison constants - numerical search
- Reason: Both are binary fractions with small number of bits to simplify division algorithm

Example

- $X=(0.00111111)_2=63/256$; $D=(0.1001)_2=9/16$
- Appropriate comparison constant - $K_2=7/16=0.0111_2$
- If remainder negative - compare to two's complement of $K_2=1.1001_2$

$r_0 = X$		0	.0	0	1	1	1	1	1	1	
$2r_0$		0	.0	1	1	1	1	1	1	0	≥ 0.0111 set $q_1 = 1$
Add $-D$	+	1	.0	1	1	1					
r_1		1	.1	1	1	0	1	1	1	0	
$2r_1 = r_2$		1	.1	1	0	1	1	1	0	0	≥ 1.1001 set $q_2 = 0$
$2r_2 = r_3$		1	.1	0	1	1	1	0	0	0	≥ 1.1001 set $q_3 = 0$
$2r_3$		1	.0	1	1	1	0	0	0	0	< 1.1001 set $q_4 = \bar{1}$
Add D	+	0	.1	0	0	1					
r_4		0	.0	0	0	0	0	0	0	0	zero final remainder

- $Q=0.100\underline{1}=0.0111_2=7/16$ - minimal SD form