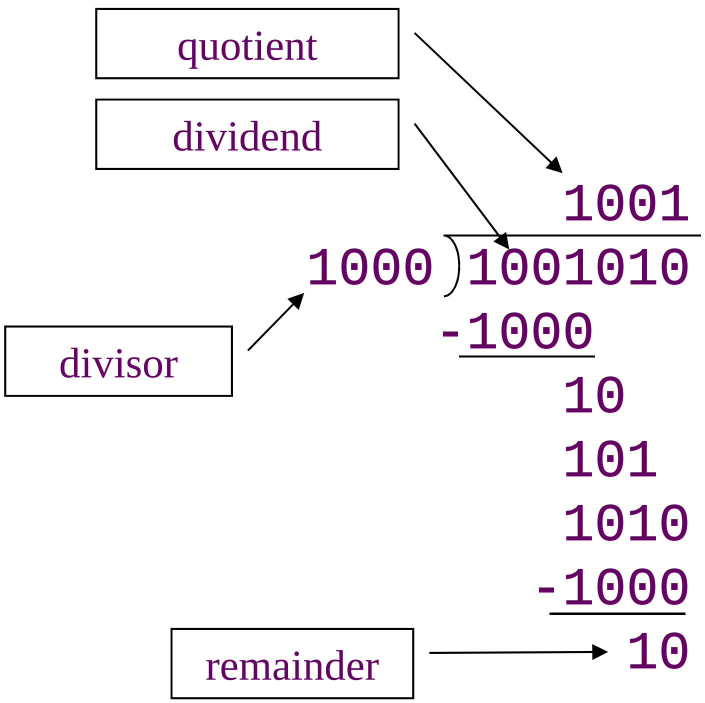
COMPUTER ORGANIZATION (IS F242)

LECT 11: DIVISION, FLOATING POINT

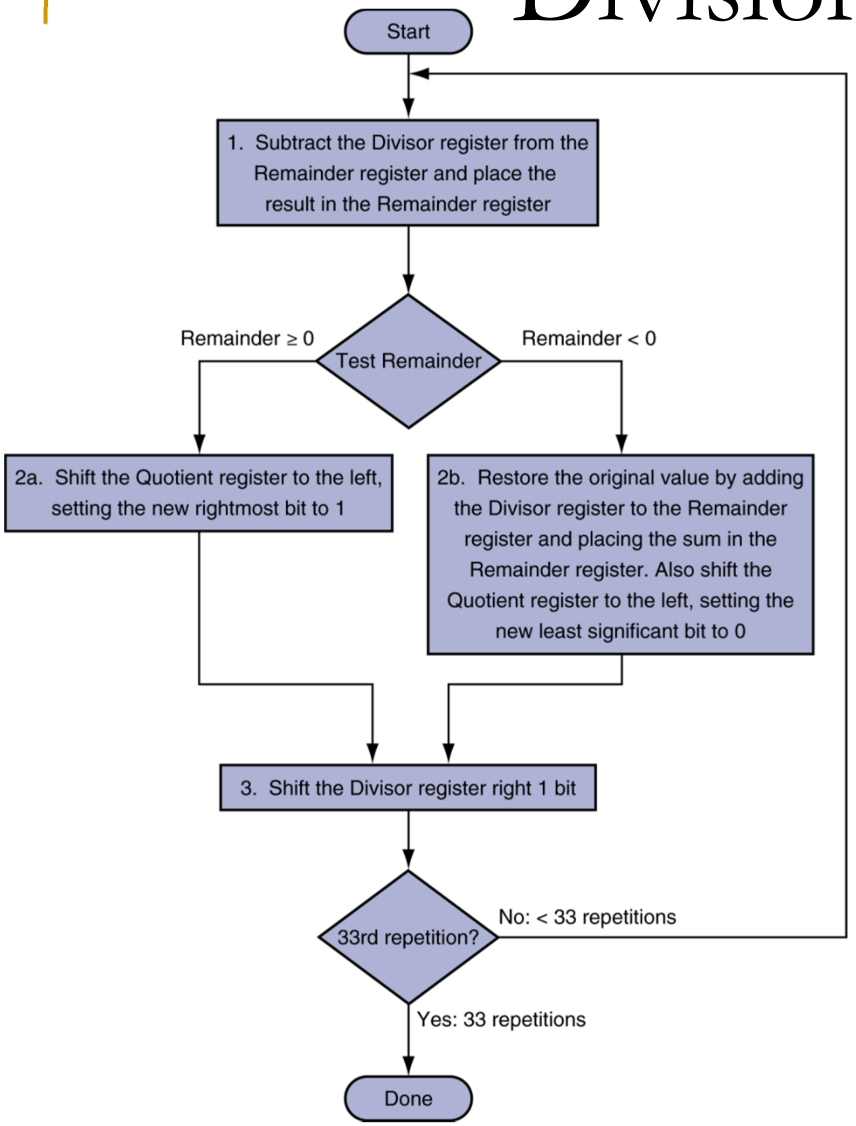
Division

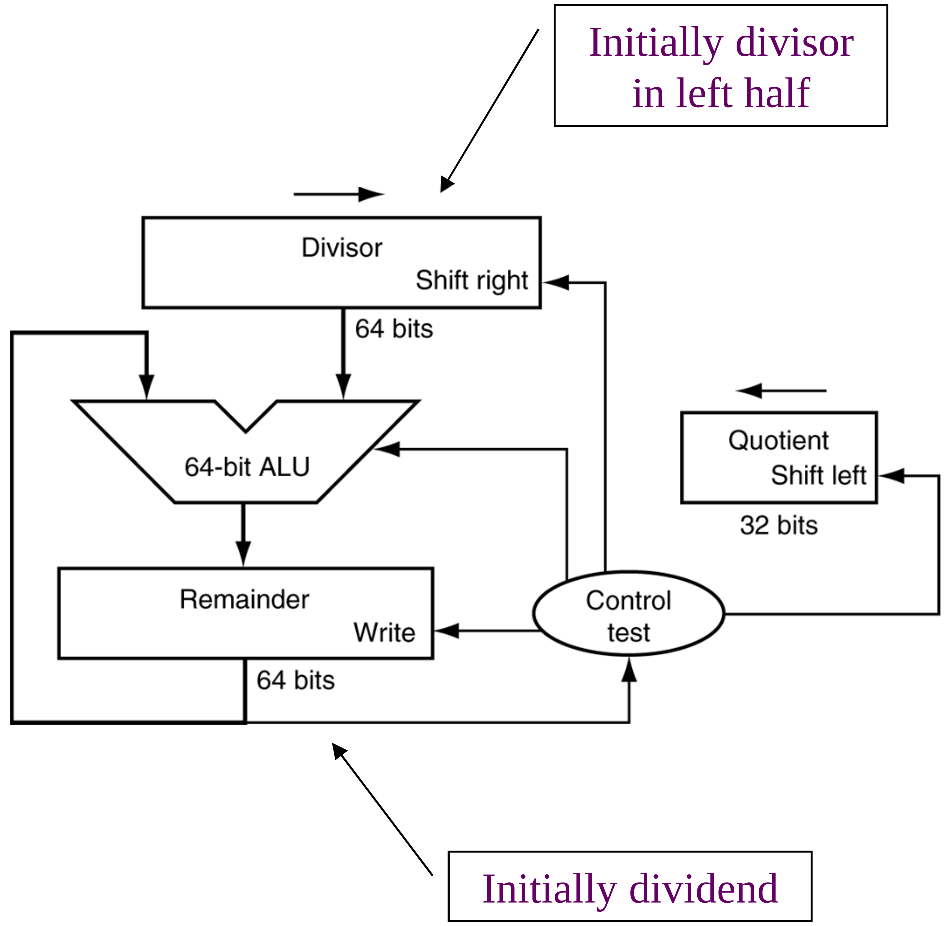


n-bit operands yield *n*-bit quotient and remainder

- Check for 0 divisor
- Long division approach
  - If divisor ≤ dividend bits
    - 1 bit in quotient, subtract
  - Otherwise
    - 0 bit in quotient, bring down next dividend bit
- Restoring division
  - Do the subtract, and if remainder goes < 0, add divisor back
- Signed division
  - Divide using absolute values
  - Adjust sign of quotient and remainder as required

Division Hardware





Reminder Divisor Quotient

0100 1010 1000 0000 0000 Initial Values

R-D < 0, So Reminder remains same, Shift D right; Shift Q left & Q0 = 0

0100 1010 0100 0000 0000 Shift D right & Q left, Q0=0

R-D > 0, So R = R – D; Shift D right; Shift Q left & Q0 = 1.

0000 1010 0010 0000 0001 R=R-D; Shift D right & Q left; Q0=1

R-D < 0, So Reminder remains same, Shift D right; Shift Q left & Q0 = 0

0000 1010 0001 0000 0010 Shift D right & Q left, Q0=0

R-D < 0, So Reminder remains same, Shift D right; Shift Q left & Q0 = 0

0000 1010 0000 1000 0100 Shift D right & Q left, Q0=0

R-D > 0, So R = R – D; Shift D right; Shift Q left & Q0 = 1.

0000 0010 0000 0100 1001 R=R-D; Shift D right & Q left; Q0=1

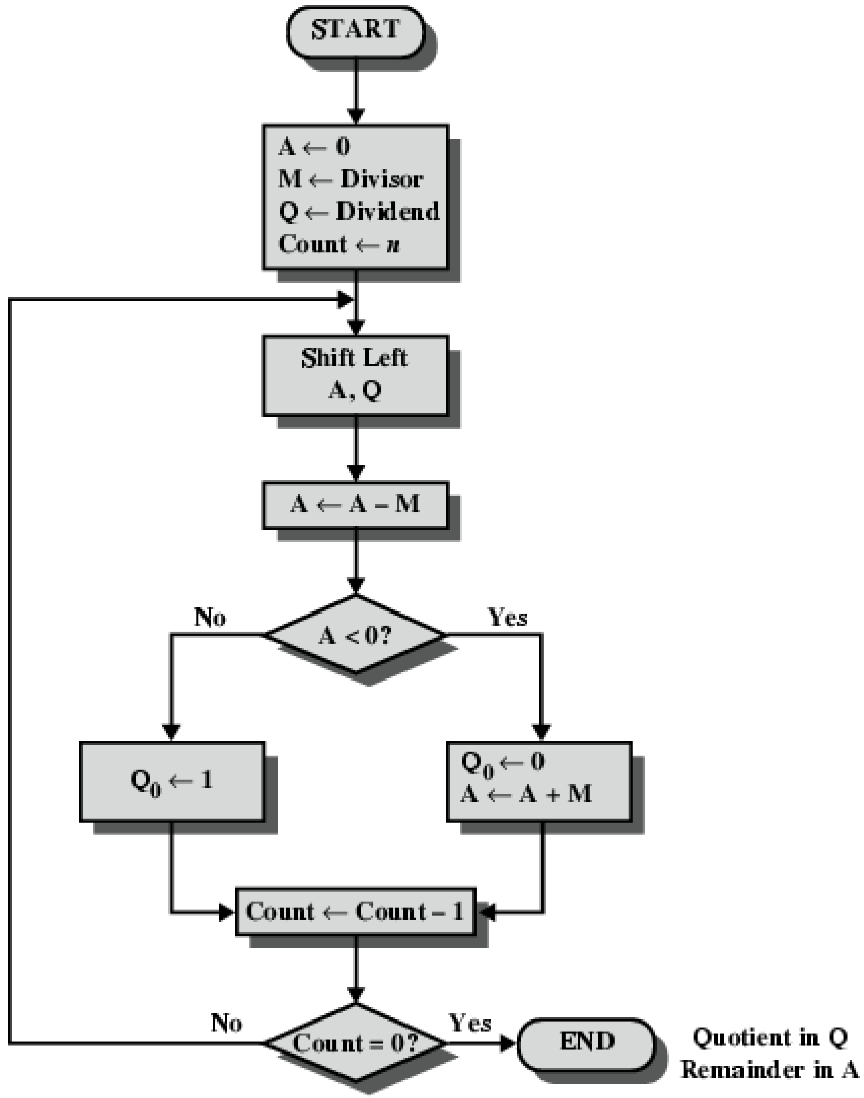
Final result is in Quotient register [1001] and Reminder in Reminder register [0010]

Division of Unsigned Binary Integers

More complex than multiplication
Negative numbers are really bad!
Based on long division

```
                     00001101     <-- Quotient
                   ----------
Divisor --> 1011  | 10010011      <-- Dividend
                     1011
                    ------
                    001110
Partial               1011
Remainders           ------
                      001111
                        1011
                       ------
                         100      <-- Remainder
```

Flowchart for Unsigned Binary Division

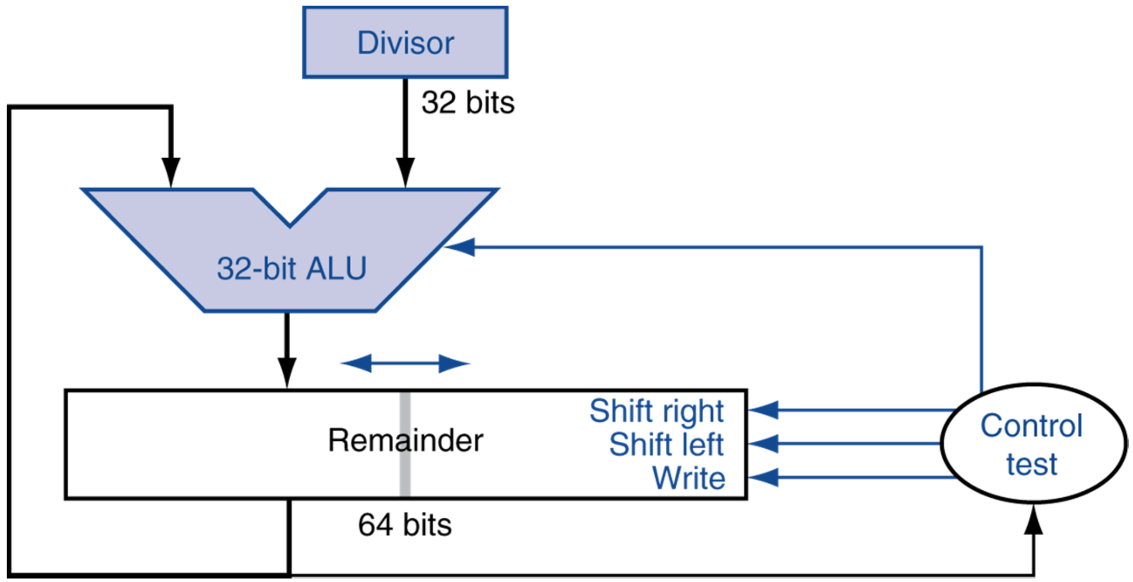


A (N+1 bits)	Q	M	
00000	1001 0011	1011	Initial Values
00001	0010 0110	1011	Shift A,Q left
A-M<0; A remains same; Q0=0			
00001	0010 0110	1011	A=A; Q0=0
00010	0100 1100	1011	Shift A,Q left
A-M<0; A remains same; Q0=0			
00010	0100 1100	1011	A=A; Q0=0
00100	1001 1000	1011	Shift A,Q left
A-M<0; A remains same; Q0=0			
00100	1001 1000	1011	A=A; Q0=0
01001	0011 0000	1011	Shift A,Q left
A-M<0; A remains same; Q0=0			
01001	0011 0000	1011	A=A; Q0=0

A (N+1 bits)	Q	M	
01001	0011 0000	1011	Initial Values
10010	0110 0000	1011	Shift A,Q left
A-M>=0; A = A - M; Q0=1			
00111	0110 0001	1011	A=A-M; Q0=1
01110	1100 0010	1011	Shift A,Q left
A-M>=0; A = A - M; Q0=1			
00011	1100 0011	1011	A=A-M; Q0=1
00111	1000 0110	1011	Shift A,Q left
A-M<0; A remains same; Q0=0			
00111	1000 0110	1011	A=A; Q0=0
01111	0000 1100	1011	Shift A,Q left
A-M>=0; A = A - M; Q0=1			
00100	0000 1101	1011	A=A-M; Q0=1

Final result is in Quotient register [1101] and Reminder in A [0100]

Optimized Divider



- One cycle per partial-remainder subtraction
- Looks a lot like a multiplier!
  - Same hardware can be used for both

Remainder (2N+1 bits)		Divisor	
01001	0011	1011	Initial Values

R[4-8] – D < 0; Shift R; R[0]=0

10010	0110	1011	Shift R; R[0]=0

R[4-8] – D >= 0; R[4-8] = R-4-8] – D; Shift R; R[0]=1

00111	0111	1011	R[4-8]=R[4-8]-D; R[0]=1
01110	1110	1011	Shift R

R[4-8] – D >= 0; R[4-8] = R-4-8] – D; Shift R; R[0]=1

00011	1111	1011	R[4-8]=R[4-8]-D; R[0]=1
00111	1110	1011	Shift R

R[4-8] – D < 0; Shift R; R[0]=0

01111	1100	1011	Shift R; R[0]=0

R[4-8] – D >= 0; R[4-8] = R-4-8] – D; Shift R; R[0]=1

00100	1101	1011	R[4-8]=R[4-8]-D; R[0]=1

Representation of Number – (Refer CP)

- Binary representation
  - Unsigned
  - Signed
    - Signed Magnitude
    - 1's Complement
    - 2's Complement
- Bit Vector representation
- ASCII representation

Fractions: Fixed-Point

- How can we represent fractions?
  - Use a “binary point” to separate positive from negative powers of two -- just like “decimal point.”

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

Example:

01111110.110

Conversion from Binary to Decimal(Fractions)

Convert to decimal: **00101000.101**

Step1: Take integer part and convert to decimal

$2^3 + 2^5 = 40$

Step2: Take fraction part and convert to decimal

$2^{-1}+2^{-3} = .625$

Step3: Add result of Step1 and Step2

40.625

Arithmetic on Fractions

2's comp addition and subtraction still work.

- If binary points are aligned

Example:

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

00101000.101 (40.625)

+ 11111110.110 (-1.25)

00100111.011 (39.375)

Note: 11111110 = -2 (decimal)

.110 = 0.75

So 11111110.110 gives -2 + 0.75 = -1.25

Representation of Real Numbers

- Numbers with fraction
- Could be done in pure binary
- Where is the binary point?
- Fixed Binary point?
  - Very limited
- Varying Binary point?
  - How do you show where is the binary point?

Floating Point Representation types

- IBM – 370 Format
- IEEE – 754 Format

IEEE 754 Floating Point Representation

- Single Precision – 4 bytes (32 bits)
- Double Precision – 8 bytes (64 bits)
- Extended Double – 10 bytes (80 bits)
- Quadruple precision – 16 bytes (128 bits)

Floating point representation

- Representation of floating point Number
  - Example: 01110.101 x 2^{-20} (+14.625 * 2^{-20})
- Stored as 3 parts
  - Sign
  - Exponent
  - Mantissa (Significant or fraction)
- More bits for fraction gives more precision
- More bits for exponent increases range