THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

CS330 Booth’s

Spring 2022

Lab 9

Two’s Complement

Two's complement, or signed representation, uses the leftmost bits of a binary number to indicate the number's sign. These are known as the sign bit(s), and this process is how computers represent negative numbers since they otherwise have no concept of sign.

Positive numbers begin with 0s, such as 01000 (8)

Negative numbers begin with 1s, such as 11000 (-8)

To get the negative of a positive number, such as 00101011 (43):

1. Flip all the bits: 11010100.

2. Add one: 11010101 (-43). Reverse these steps to get the positive back.

Two’s Complement Addition/Subtraction

All the normal rules for addition apply

0001 + 0001 = 0010. 1 + 1 would be 2, but since there is no 2, we carry a 1 to the next place, like if we have 9+1 in decimal.

0111 + 0111 = 1110 by the same logic, but you'll note that this number now begins with 1. This is incorrect.

In two’s complement addition, if you add two numbers with the same sign and the result has the opposite sign, overflow occurs. This means the result is incorrect; for the upcoming exercises and homework, note that overflow has occurred when writing the result.

Two’s Complement addition/subtraction

Since we are emulating the way computers do addition and subtraction, there are two more important parts of the process:

1. All inputs and outputs have the same number of bits. If you are summing 5000 and 12 in binary, you must extend 12 to have as many bits as 5000, and the result will also have that many bits. Similarly, if you have two 4 bit numbers, the result of summation will have 4 bits.

2. You must *always* use the signed notation to perform addition and subtraction. This means being aware of the rules of overflow.

Otherwise, subtraction is exactly the same as addition, except you need to ensure the number is negative (begins with 1) before starting.

Arithmetic Right Shift, quick examples

11101100 --> 11110110. All digits shift to the right, and the sign bit is copied.
The last 0 goes off the number, and is gone.
The ones at the front, the sign bits, are copied when they “enter” the number.

00100101 --> 00010010
The last one goes off the number, and is gone.
The zeroes at the front, sign bits, are copied when they “enter” the number.

If you shift on multiple registers, simply put them all in the specified order and shift as though you shift on a single binary number.

Booth's Algorithm Multiplication

You'll want graph paper or an equivalent for this!

A is assigned 0. It represents the high bits of the result.

Q_{-1} is assigned 0. It holds the bit most recently shifted off of Q.

M is the Multiplicand, and Q is the Multiplier (both in binary). If we have 5 * 3, M is 5 and Q is 3. Importantly, M will never be re-assigned.

Count is the length of the binary number, i.e. number of bits used to represent the multiplicand (or multiplier).

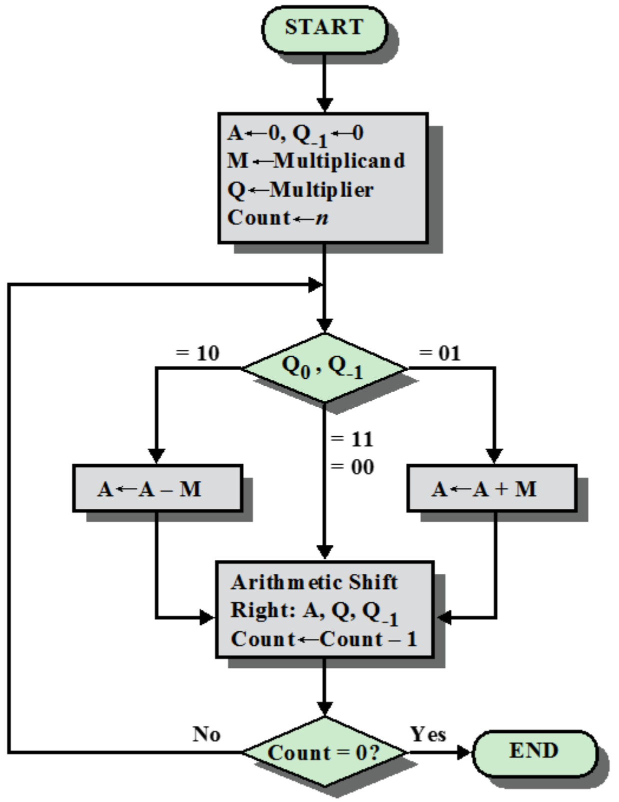


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

Multiplication, Continued

First, we check Q_0 (the rightmost digit of Q) and Q_{-1}

If they are 1 and 0 respectively, A is assigned A - M. Then you go to the Shift step.

If they are 0 and 1 respectively, A is assigned A + M. Then you go to the Shift step.

If they are 11 or 00, or when you finish either above step, you perform an Arithmetic Right Shift on all registers in the order A, Q, Q_{-1}. Decrement Count.

Lastly, if the Count is zero, you're finished, and the result is in A, Q. If not, go back to the start.

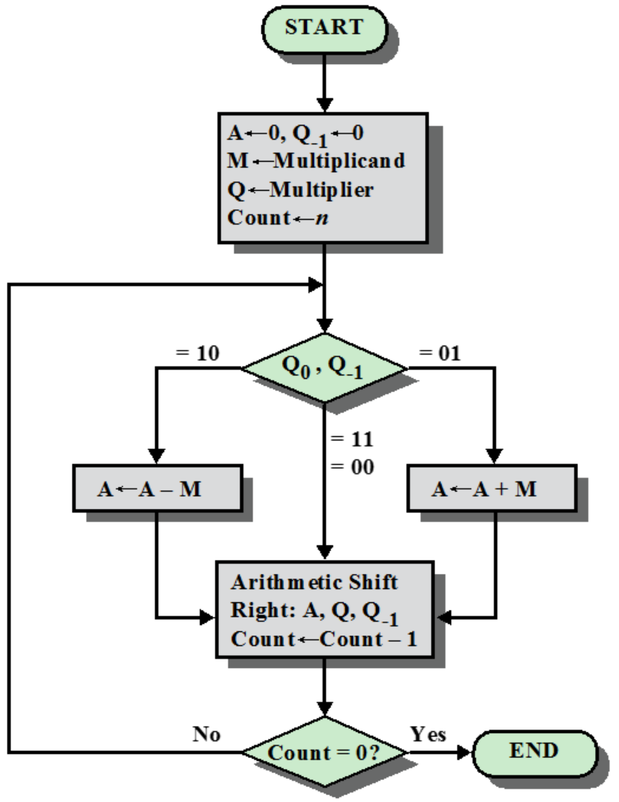


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

Booth’s Algorithm, Division

A is assigned 0. It represents the remainder after division is complete.

M is the divisor, and Q is the Dividend. For reference, Dividend means the top number (to be divided), and divisor means the bottom number (to divide by).

In other words, the operation we're doing is Q divided by M.

Q will be the quotient (result) after we finish.

Lastly, Count is the length of the divisor. Divisor and dividend should be the same length.

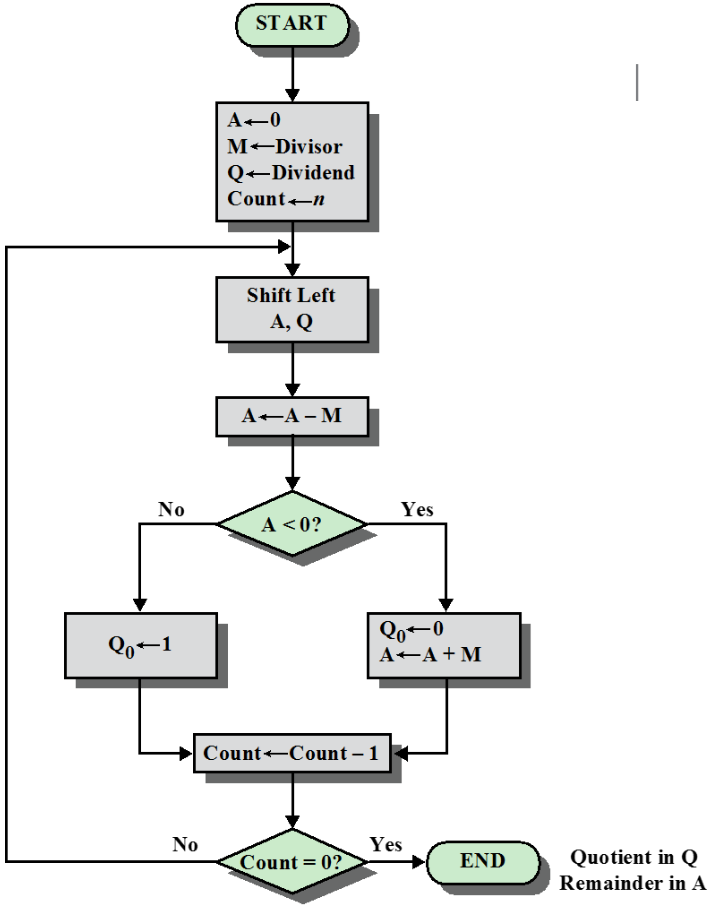


Figure 10.16 Flowchart for Unsigned Binary Division

Division, Continued

First, we shift left A and Q. Left shift is exactly the same as right, except there is no ‘arithmetic’ component that cares about the sign bit. We only care that each bit is shifted left, and 0 comes on at the right (low) end.

A is assigned A - M.

If A is less than zero (check sign bit), the last bit of Q becomes zero, and A is assigned A + M.

If A is not less than zero, the last bit of Q becomes one.

Then we decrement Count. If it is zero, we’re finished; quotient is in Q, and remainder is in A. If not, go back to the start.

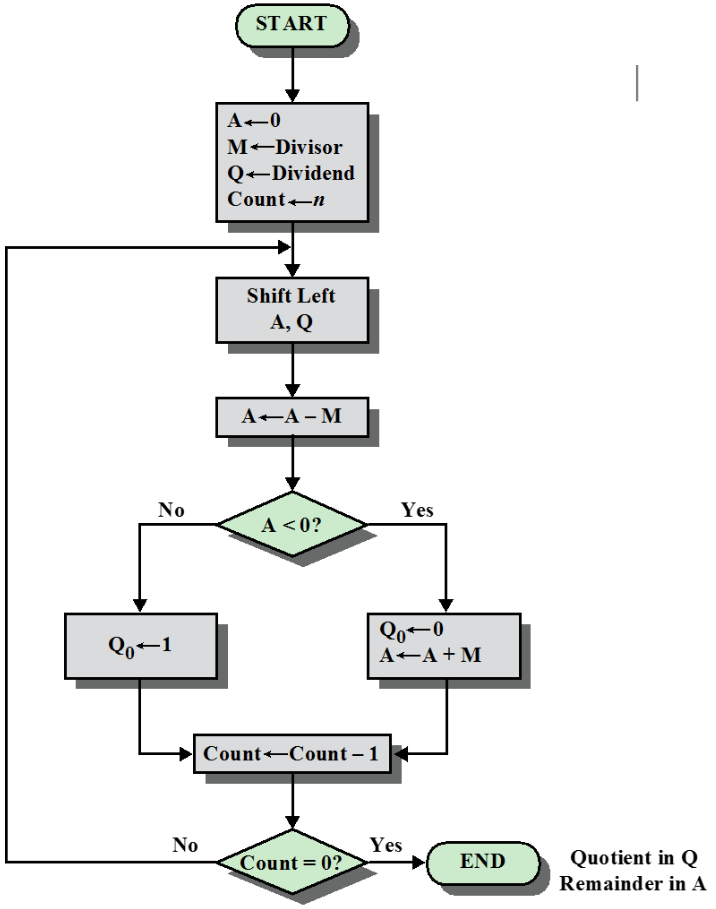


Figure 10.16 Flowchart for Unsigned Binary Division

Exercises for Multiplication

a. 3 * 5 (0011 * 0101)

b. 3 * -7

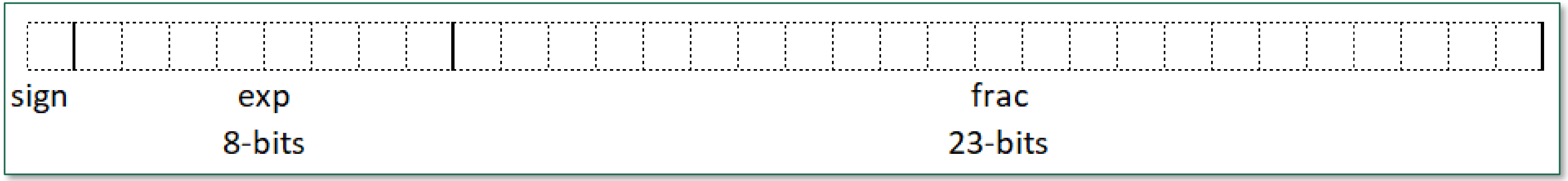
THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

Backup

IEEE754

- Example: 125.125

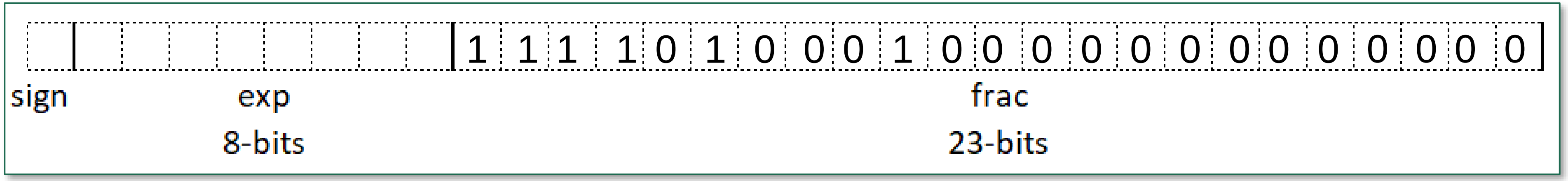
1. Convert to binary:
   125 == 1111101
   $\frac{1}{2}\ \frac{1}{4}\ \frac{1}{8}$
   .125 == 125/1000 == 1/8 == 0.001
   1111101.0001



IEEE754

- Example: 125.125

2. Convert to 1.xxxx format:
$$1111101.0001 == 1.\underbrace{1111010001}_{\text{Mantissa (fraction)}} \times 2^6$$



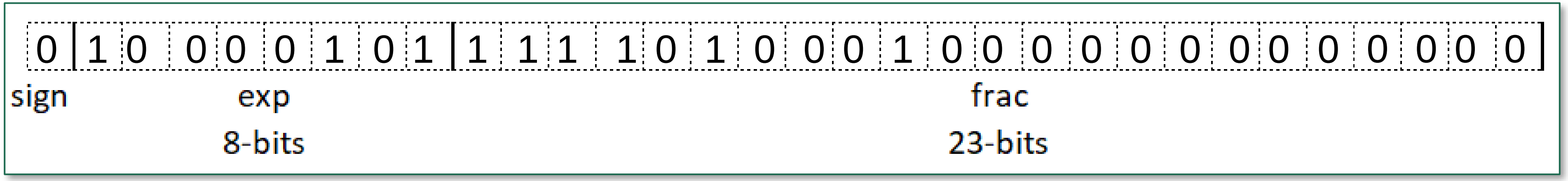
IEEE754

- Example: 125.125

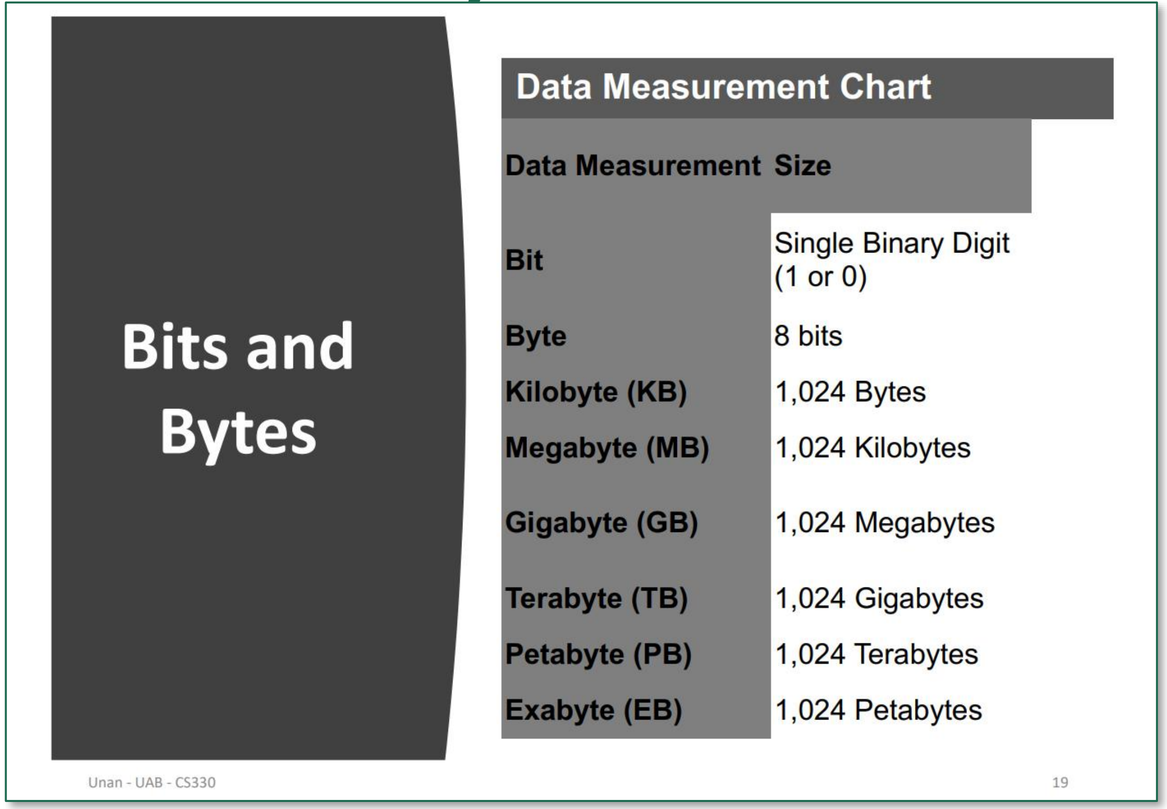
3. Calculate Exponent:
   1.1111010001×2^6

   $E = Exp - Bias$ $\qquad\qquad$ $6 = Exp - 127$

   $Exp = 133 == 10000101$



From Lecture 4, Chart 19



Bits and Bytes

Data Measurement Chart

Data Measurement	**Size**
Bit	Single Binary Digit (1 or 0)
Byte	8 bits
Kilobyte (KB)	1,024 Bytes
Megabyte (MB)	1,024 Kilobytes
Gigabyte (GB)	1,024 Megabytes
Terabyte (TB)	1,024 Gigabytes
Petabyte (PB)	1,024 Terabytes
Exabyte (EB)	1,024 Petabytes

Unan - UAB - CS330 19