EE 3755

Computer Arithmetic

Handout # 4

Floating Point (FLP) Arithmetic: Introduction

A floating point number consists of two fixed point (FXP) parts: a fraction f and an exponent e. Then, the floating point number A can be expressed as

$A = \pm f \times r^e$ or $A = (f, e)$

where

f = fraction of A (f is fractional or $f = .xx\cdots x$)
e = exponent of A
r = radix used (for decimal arithmetic r=10; for binary r=2 etc...).

Question: Why use floating point arithmetic?
Answer: Because with floating point systems we can get higher dynamic ranges than the ones we get with fixed point systems.

Multiple FLP representations of a number

A number can have many different FLP representations. For example consider decimal arithmetic (r=10); then the following show six different representations of the same number A:

$A = .0000068421 \times 10^3$
$A = .0000684210 \times 10^2$
$A = .0006842100 \times 10^1$
$A = .0068421000 \times 10^0$
$A = .0684210000 \times 10^{-1}$
$A = .6842100000 \times 10^{-2}$
↑
decimal point

Obviously

- The fraction can be shifted "k" digit positions to the left and simultaneously the value of the exponent should be decremented by "k" without changing the value of the number.

OR

- The fraction can be shifted "k" digit positions to the right and simultaneously the value of the exponent should be incremented by "k" without changing the value of the number.

Normalized FLP numbers

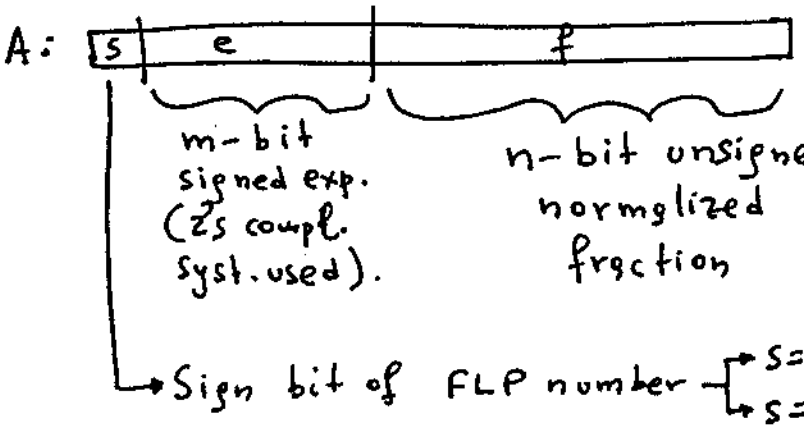
A floating point number is called <u>normalized</u> if the digit immediately to the right of the radix point in its fraction is <u>nonzero</u>.

For binary arithmetic ($r=2$), a normalized number will have a fraction of the form $f = .1xx\cdots xx$.

In the remaining we will assume binary arithmetic ($r=2$) and normalized numbers.

Dynamic Range

Consider the following binary FLP format:



→ obviously $A = \pm f \times 2^e$ or $A \equiv (-1)^s \times f \times 2^e$.

Problem: Compute the Dynamic Range for a floating point system based on the binary format shown on the previous page (m-bit signed exponent; n-bit unsigned normalized fraction).

Answer: Since the fraction is normalized, the range of the fraction is $0.5 \leq f \leq 1 - 2^{-n}$.

Since the exponent is an m-bit signed number (2's complement system used for representing signed numbers) its range is

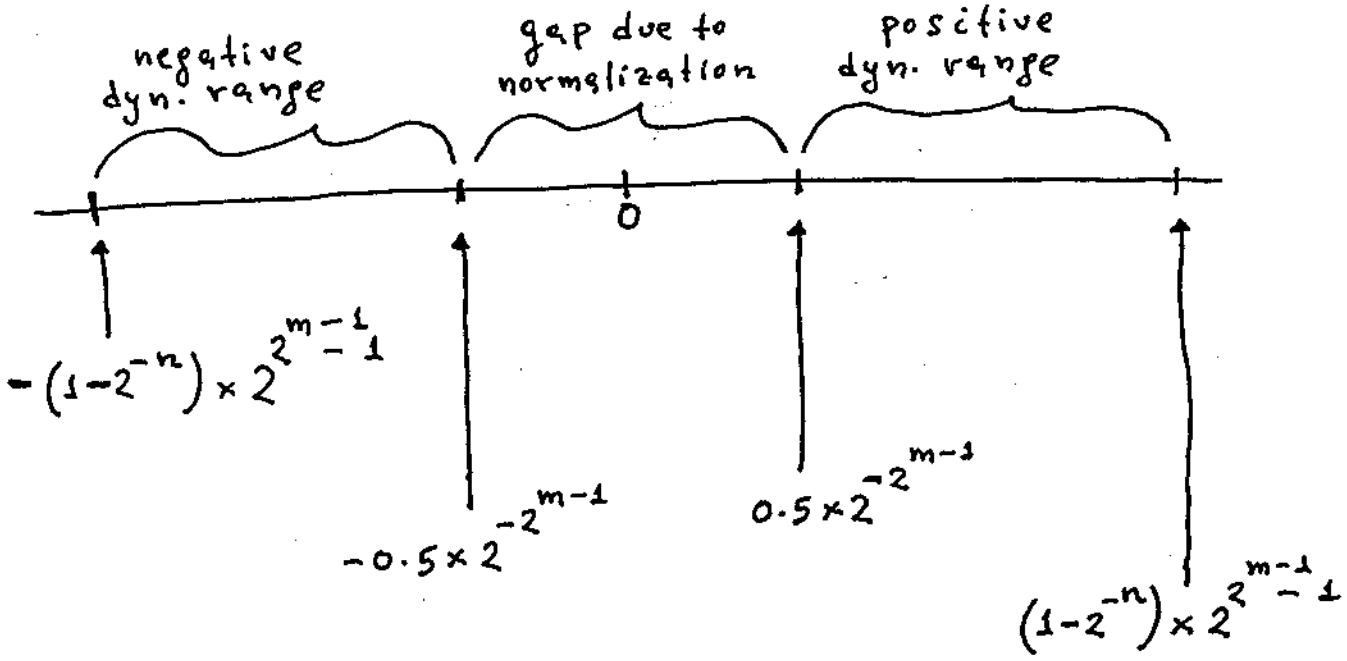
$$-2^{m-1} \leq e \leq 2^{m-1} - 1 .$$

Then, the positive FLP dynamic range is

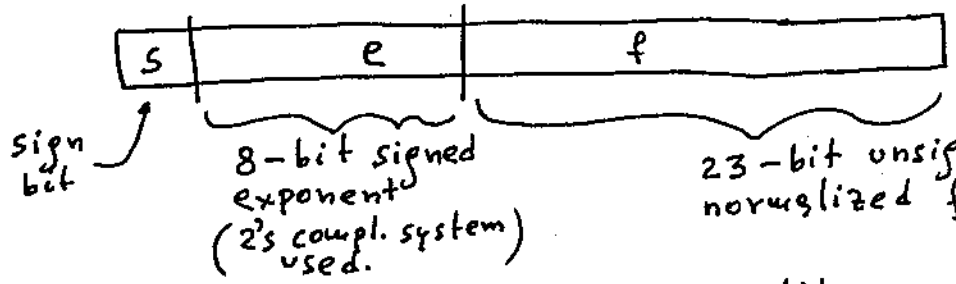
$$0.5 \times 2^{-2^{m-1}} \leq A^{+} \leq \left(1 - 2^{-n}\right) \times 2^{2^{m-1} - 1}$$

while the negative FLP dynamic range is

$$-\left(1 - 2^{-n}\right) \times 2^{2^{m-1} - 1} \leq A^{-} \leq -0.5 \times 2^{-2^{m-1}}$$



Example 1: Compute the Dynamic Range for a floating point system based on the 32-bit binary FLP format shown below:



Answer: Since fraction is 23 bits normalized, its range is $0.5 \leq f \leq 1-2^{-23}$
Since exponent is 8-bit signed (2's compl. syst. used) its range is $-2^7 \leq e \leq 2^7-1$ or $-128 \leq e \leq 127$.
Thus the positive dynamic range is

$$0.5 \times 2^{-128} \leq A^+ \leq (1-2^{-23}) \times 2^{127}$$

while the negative dynamic range is

$$-(1-2^{-23}) \times 2^{127} \leq A^- \leq -0.5 \times 2^{-128}$$

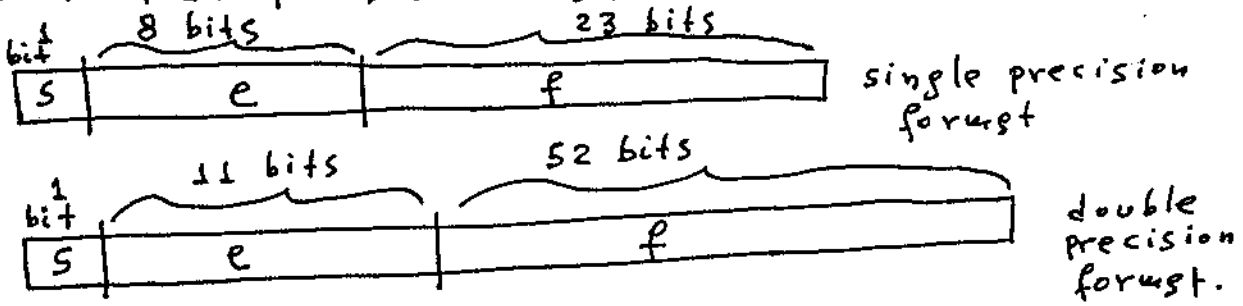
Compare the above dynamic range with the range of a 32-bit fixed point system.

Length of fraction and length of exponent

- The length "n" of the fraction dictates the precision of the system. The larger "n" becomes, the larger the precision by which we can represent FLP numbers becomes.
- The length "m" of the exponent dictates the size of the dynamic range. The larger "m" becomes, the larger the dynamic range becomes.

FLP formats

There are numerous floating point formats. Different manufacturers use different formats. The standards proposed by the IEEE for single precision and double precision FLP formats are shown below:



Biased exponents

When adding or subtracting two floating point numbers, their exponents must be compared and made equal, resulting in a shift operation on one of the fractions. It is known that it is easier to compare unsigned numbers rather than signed. In order to take advantage of the above fact, the exponents can be converted to unsigned by adding a positive constant named "bias". This way, the actual or unbiased exponent becomes biased.

For m-bit signed exponents (2's compl. system used for representing signed numbers) the range of the true or unbiased exponent is

$$-2^{m-1} \leq e_{unbiased} \leq 2^{m-1} - 1$$

To translate the exponents into unsigned numbers we perform

$$e_{biased} = e_{unbiased} + bias$$

where the bias amount is

$$bias = 2^{m-1}$$

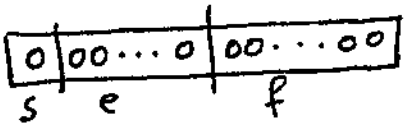
The range of the biased exponent now is

$$0 \leq e_{biased} \leq 2^m - 1$$

Representation of the floating point zero

There can be many floating point zero representations (observe that any floating point number with $f=0$ has a value of zero). This way we have a non-unique zero representation. However, a unique zero representation is always desired.

A unique FLP zero is a FLP number with sign bit $s=0$, fraction $f=0$ and exponent e = smallest possible exponent. If the exponents are biased then $(e_{biased})_{min} = 0$. As a result of this if we use biased exponents the unique FLP zero corresponds to the number with $s=0$, $f=0$, $e=0$ or



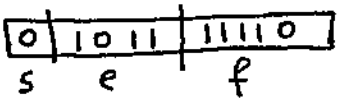
The above representation of zero is the familiar representation of zero in fixed point systems and this is another reason of why biased exponents are prefered.

Note: If a floating point operation is expected to produce a result of zero (like $A-A$; $0 \times A$; $\frac{0}{A}$ etc) then the result should be forced to the unique (or true) zero.

Note: The biased exponent is not the actual or true exponent. If you want to compute the dynamic range of a floating point system or find the

value of a floating point number, then the true (unbiased) exponent should be used.

Example 2: Consider the following floating point number

A: 

where its exponent e is in biased form. Find the value of the number A.

Solution: Here $e_{biased} = (1011)_2 = (11)_{10}$. Since the exponent length is $m = 4$ bits, the bias amount is $bias = 2^{m-1} = 2^3 = 8$. Thus $e_{true} = e_{unbiased} = e_{biased} - bias = 11 - 8 = 3$. The value of A now is $A = +(.11110) \times 2^3 = +(111.10)_2 = (7.5)_{10}$

Question: How can we perform an addition or subtraction between two floating point numbers?

Answer: Consider two binary FLP numbers $A_1 = (-1)^{s_1} \times f_1 \times 2^{e_1}$ and $A_2 = (-1)^{s_2} \times f_2 \times 2^{e_2}$. In order to perform an addition or subtraction between A_1 and A_2, their exponents must be made equal. Suppose that $e_1 > e_2$ and suppose $e_1 - e_2 = k$. Then it seems like that we can follow one of the following two approaches:

1. Try to make the two exponents equal to the smaller exponent e_2. This would mean that we would have to shift the fraction corresponding to the larger exponent f_1 k bit locations to the left so that A_1 becomes $A_1 = (-1)^{s_1} \times f_1' \times 2^{e_2}$ (where f_1' = result of shifting f_1 k bit locations to the left). This approach relies on left shifts and

this way we might lose the most significant bits of the left-shifted fraction. We will never follow this approach.

OR

2. Try to make the two exponents equal to the larger exponent e_1. This would mean that we would have to shift f_2 k bit locations to the right so that A_2 becomes $A_2 = (-1)^{s_2} \times f_2' \times 2^{e_1}$ (where f_2' = result of shifting f_2 k bit locations to the right). This approach relies on right shifts and this way we might only lose the least significant bits of the fraction to be right-shifted. We will always follow this approach.

So the conclusion is that before performing an addition or subtraction between two floating point numbers their exponents must be made equal to the larger exponent.

Overflows and Underflows

- Fraction overflow: A fraction overflow is defined to be the situation where the resulting fraction exceeds its n-bit fractional space (ie; the resulting fraction is $\underbrace{1.\underbrace{x\ldots x}_{n \text{ bits}}}_{n+1 \text{ bits}}$).

  The fraction overflow is correctable by shifting the overflowed result one bit to the right and incrementing the exponent by one.

- Fraction underflow: A fraction underflow is defined to be the loss of right most bits of a fraction as a result of a right shift operation.

- Exponent overflow: If the exponent of the result of some floating point operation is larger than the maximum allowable exponent value, we then have an exponent overflow. In such a case the FLP processor will set an exponent overflow flag.
- Exponent underflow: If the exponent of the result of some floating point operation is smaller than the minimum allowable exponent value, we then have an exponent underflow. In this case the result of the floating point operation is forced to become equal to the unique zero.