
Floating Point Representation

Textbook Section 8.4

Fixed Point Decimal Numbers

- 2's complement is a fixed point representation.
 - the radix (decimal) point is fixed to the immediate right of bit 0
- This gives positive and negative integers, centered on zero.
- By fixing a different radix point, fractions may be represented!

Fixed Point Example

Consider the following 8-bit number:

01100101_2

What decimal number is represented if the fixed point is:

- immediately right of bit 0? (i.e. normal binary)

76543210.

01100101.

$$2^6 + 2^5 + 2^2 + 2^0 = 64 + 32 + 4 + 1 = 101$$

Fixed Point Example

Consider the previous 8-bit number:

01100101_2

What decimal number is represented if the fixed point is:

- immediately right of bit 1?

7654321.0

0110010.1

$$2^5 + 2^4 + 2^1 = 32 + 16 + 2 = 50$$

$$2^{-1} = 1 / 2^1 = 0.5$$

Positions to the right of the fixed point are negative exponents starting at -1

Negative Powers of 2

$$2^{-1} = 1 / 2^1 = 0.5$$

$$2^{-2} = 1 / 2^2 = \frac{1}{2} * \frac{1}{2} = 0.25$$

$$2^{-3} = 1 / 2^3 = 0.125$$

$$2^{-4} = 1 / 2^4 = 0.0625$$

$$2^{-5} = 1 / 2^5 = 0.03125$$

What do you notice about all these #s?

What does this indicate about the values that can / cannot be represented?

Example Continued

So,

$01100101. = 101.0$

$0110010.1 = 50.5$

$011001.01 = 25.25$

...

Notice:

1. How conversion from binary to decimal is done – convert each portion, whole and fractional, **separately** and then combine.
2. What happens to ranges of each of the whole and fractional parts as the decimal point is shifted?

Exercise

Convert 19.3125_{10}

How to convert?

Split into whole and fractional portions and convert each separately.

Know how to convert the whole portions:

Decomposition or remainder method

$$19 = 16 + 2 + 1 = 10011_2$$

Exercise Continued

How to convert 0.3125_{10} ?

Division by two will simply make the value smaller. It will approach zero but never get there.

∴ will be an infinite decimal, but we know this value isn't infinite!

Convert Decimal Part to Binary

Repeatedly multiply by the base:

$$0.3125 \times 2 = 0.6250$$

Ignore the whole part and repeat until the fractional part becomes 0

$$0.625 \times 2 = 1.250$$

$$0.25 \times 2 = 0.50$$

$$0.5 \times 2 = 1.0$$



Now read the whole values from top down **.0101**

$$19.3125_{10} = 10011.0101_2$$

Fixed Point Conversions

Regardless of direction:

1. Separate the number at the fixed point into whole and fractional parts
2. Convert the whole part
3. Convert the fractional part
4. Recombine the two parts

Fixed Point Math

Remember that a representation is useless unless it allows operations!

Consider the following addition, if viewed as unsigned binary:

11001010	202
+ <u>01100101</u>	<u>101</u>
100101111	303

If viewed as fixed point with 2 decimal places:

110010.10	50.5
+ <u>011001.01</u>	<u>25.25</u>
1001011.11	75.75

Fixed Point Pros

Once the position of the fixed point is set it is fixed for all values because there is no way of specifying the position in the value!

New I/O Routines are required!

Pros:

- no new add/sub circuits required
- mul/div is easy, given integer mul/div

Fixed Point Cons

Cons:

- can't represent very large or very small numbers
 - limited range, variation between upper and lower limits, and precision, number of digits that can be represented)
- prone to rounding error
- lose negation in 2's comp
 - inverting only works with a fixed range of values

This representation has some benefits but also some significant limitations

Floating Point Basics

The biggest problems for fixed point is range and precision – how do scientist overcome these problems?

Represent the following decimal numbers conveniently:

0.0000000472

4.72×10^{-8}

472,000,000,000,000

4.72×10^{14}

In scientific notation the decimal point “floats” to a convenient location.

Scientific notation provides a simplified representation, but arithmetic is less convenient

Floating Point Basics

The use of scientific notation can be done in binary.

To represent a number of the form:

$$\pm M \times 2^{\pm E}$$

where M is called the mantissa,
E is called the exponent,
2 is the exponent base

... we can use a 32-bit value:



Floating Point Features

The exponent base can be any value and does not need to be stored. However, the use of the value 2 is a compromise between range and precision. Also it allows for easy manipulation (shifting).

Bit 31 is the sign of M (0 means +, 1 means -), this is **signed magnitude**.

The exponent is biased. This means that a constant “bias” is subtracted from the biased exponent (e') to get the true exponent (e).

For k bits, the bias is typically $(2^{k-1} - 1)$.

For 8 bits, the bias is typically 127 (this is also called “excess 127” notation).

Thus, $e' = e + 127$ & $e = e' - 127$

Floating Point Exponent Range

The exponent range is:

True	Biased
-126 to +127	1 to 254

Complete the following conversions, assuming excess 127:

True	Biased
127	254
72	199
0	127
-90	37
-126	1

Floating Point in Binary

The normalized form a floating point number in binary is given by:

$$\pm 1.bbb\dots b \times 2^{\pm e}$$

To normalize simply:

- move the decimal point to be just right of the most significant 1
- adjust E accordingly

Normalizing Floating Point Exercise

Normalize each of the following (written using binary M and decimal E):

$$0.0110 \times 2^5$$

$$1.10 \times 2^3$$

shift 2 to the right

$$101101 \times 2^2$$

$$1.01101 \times 2^7$$

shift 5 to the left

$$101101 \times 2^{-2}$$

$$1.01101 \times 2^3$$

shift 5 to the left

Shifting left increases exponent by 1, shifting right decreases exponent by 1

What number cannot be normalized?

ZERO

Mantissa

1.bbb...b

Because a normalized number always leads with a 1, the most significant digit is implied and is therefore not stored. i.e. only store the .bbb...b

The stored 23-bit mantissa actually represents a 24-bit value.

Why do we normalize mantissas? (hint: think about uniqueness)

- Fixed format representation (good point for fixed point!)
- No loss of precision

Mantissa

1.bbb...b

Why do we use biased exponents? (hint: think about comparing floats)

- All biased exponents are positive so straightforward comparisons
- Always positive difference

Floating Point Conversion - Decimal to IEEE Packed

Convert 19.3125_{10} to IEEE Packed Notation

1. Identify whether the number is positive or negative and specify bit 31
 - It is positive so bit 31 = 0
2. Convert to binary
 - $19 = 10011$ and $.3125 = .0101$
 - $19.3125 = 10011.0101$
3. Specify in $\pm 1.bbb...b \times 2^{\pm E}$ notation
 - 10011.0101×2^0
4. Normalize
 - 1.00110101×2^4

Floating Point Conversion - Decimal to IEEE Packed

5. Convert the true exponent to a biased exponent
 - $e' = e + 127 = 4 + 127 = 131$
6. Convert the biased exponent to binary
 - $131 = 128 + 2 + 1 = 1000\ 0011$
7. Place each component in the appropriate place in the 32 bit notation and then break into groups of 4.
 - 0 1000 0011 0011 0101 0000 0000 0000 000
 - 0100 0001 1001 1010 1000 0000 0000 0000
8. Convert to Hex
 - 419A800016

Verified using the FtoH program on INS in the class directory

Floating Point Conversion - IEEE Packed to Decimal

Convert C35A480016 to decimal (reverse of decimal to IEEE Packed Notation)

1. Convert to binary
 - 1100 0011 0101 1010 0100 1000 0000 0000
2. Decompose into component parts and then group into 4s
 - 1 100 0011 0 101 1010 0100 1000 0000 0000
 - 1 1000 0110 1011 0100 1001 000 0000 0000
3. Identify the sign using bit 31
 - It is 1 so the number is negative

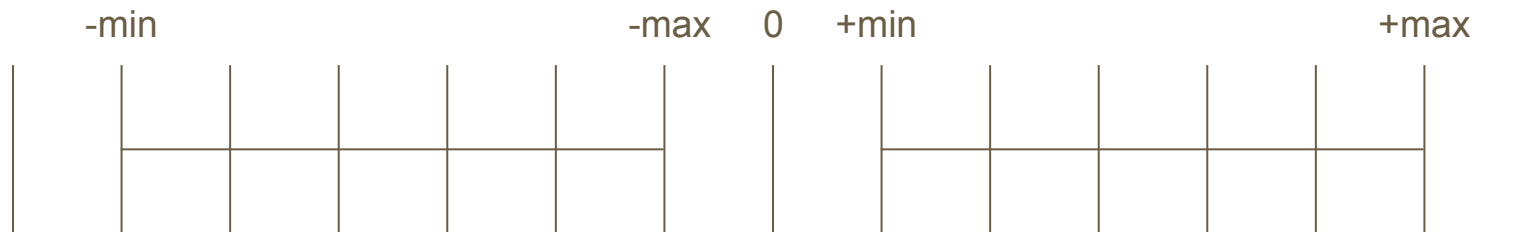
Floating Point Conversion - IEEE Packed to Decimal

4. Determine the true exponent, convert biased exponent to decimal and then subtract 127
 - $e' = 1000\ 0110 = 128 + 4 + 2 = 134$
 - $e = e' - 127 = 134 - 127 = 7$
 - aside: a shortcut if the high order bit is set,
 - i.e. e' contains a 128
 - $128 - 127 = 1 + \text{the remaining value}$
 - 128 is the value of the high order bit
 - 127 is the bias that needs to be subtracted

Floating Point Conversion - IEEE Packed to Decimal

5. Restore the hidden 1 in the mantissa, ignore any trailing zeros.
 - 1. 1011 0100 1001
6. Convert back to a normal floating point value, i.e. 2^0
 - Since $e = 7$ shift the decimal point 7 places to the right.
 - 1 1011 010.0 1001
7. Step 7. Convert the whole and fractional parts to decimal
 - $11011010 = 128 + 64 + 16 + 8 + 2 = 218$
 - $01001 = 2^{-2} + 2^{-5} = 0.25 + 0.03125 = 0.28125$
 - Then combine 218.28125
8. Apply the sign determined in step 3
 - -218.28125_{10}

Floating Point Number Line



- There are still only 2^{32} individual values!
- \therefore not all numbers in the range are precisely representable
 - rounding errors, i.e. $1 / 3$ or $0.333\dots$
- There is an uneven “density” of values
 - The highest density is around 1.0 and -1.0
 - The lowest density is near 0 and ∞

Floating Point Binary - Special Bit Patterns

$E=0, M=0$ means zero

$E=0, M \neq 0$ represents a “denormalized” number (for providing gradual underflow of very small numbers)

$E=\text{largest}, M=0$ means + or – infinity (depending on sign bit)

$E=\text{largest}, M \neq 0$ means NaN (“not a number”)

A detailed description of the latter three is beyond the scope of the course.

Note: Since there is a sign bit there exists both +ve and -ve zero

NaN Example

NHL - Mozilla Firefox

File Edit View History Bookmarks Tools Help

NHL

www.tsn.ca/nhl/teams/players/bio?id=4632

Bookmarks for Paul P... Most Visited Courses selectivecookiedelete ... Home | As It Happens... Philips Sonicare HX60... Minion Maker | Create... SP-Studio

NHL

GRABBER GENERAL TIRE



Career Stats Game-By-Game Buffalo Roster

Patrick Kaleta #36 - RW

Buffalo

Age: 27
DOB: 1986/06/08
POB: Buffalo, NY
Height: 6-1 Weight: 206lbs
Shoots: R
NHL Seasons: 7
Drafted by Buffalo in 2004 (6/176).

Current Status: On roster

FREE SHIPPING
ON ALL CONTACTS LENS ORDERS OVER \$149

ClearlyContacts.ca **SHOP NOW**

	GP	G	A	PTS	+/-	PPG	PPA	SHG	SHA	GWG	PIM	Shots	PCT	Hits
Year to date	5	0	0	0	-1	0	0	0	0	0	5	0	NaN	11
On Pace	72	0	0	0	-14	0	0	0	0	0	72	0	NaN	-

TRANSACTIONS / INJURIES / SUSPENSIONS

- 2013/11/01 Missed 10 games (suspended by nhl).
- 2013/10/15 Suspended by the NHL for 10 games.
- 2013/04/26 Missed the last 3 regular season games (hand injury).
- 2013/04/22 Hand injury, day-to-day.
- 2013/03/17 Missed 5 games (suspended by nhl).
- 2013/03/04 Suspended by the NHL for 5 games.
- 2013/02/09 Missed 5 games (neck injury).
- 2013/02/05 Neck injury, injured reserve.

SCOREBOARD

- CHICAGO WINNIPEG 1:00PM
Gameday
- ANAHEIM BUFFALO 5:00PM
Gameday
- ST. LOUIS TAMPA BAY 5:00PM
Gameday
- PHILADELPHIA NEW JERSEY 5:00PM
Gameday
- BOSTON NY ISLANDERS 5:00PM
Gameday
- CAROLINA NY RANGERS 5:00PM
Gameday
- FLORIDA WASHINGTON 5:00PM
Gameday
- PITTSBURGH COLUMBUS 5:00PM
Gameday
- TORONTO VANCOUVER 5:00PM
Gameday
- MONTREAL COLORADO 8:00PM
Gameday
- DETROIT EDMONTON 8:00PM
Gameday
- PHOENIX 8:30PM

Start NHL - Mozilla Firefox apps 1:13 PM

IEEE Packed Notation - Extra notes

- There is a tradeoff between the number of bits dedicated to M vs. E
- More bits for E (i.e. less for M)
 - gives greater range at the expense of less precision
- More bits for M (i.e. less for E)
 - gives greater precision at the expense of less range

IEEE Standard for Floating Point Numbers

The IEEE standard for floating point is widely used. It defines:

- a 32-bit format (8 bit exponent, 23 bit mantissa)
 - called “single” in C/C++ this is a float
- a 64-bit format (11 bit exponent, 52 bit mantissa)
 - called “double” in C/C++ this is a double
- These bit representations are called packed (or “internal”) format
- In order to implement operations (+, ×, etc.) on these numbers, they are typically converted to unpacked (or “external”) format
- This means the sign, exponent and mantissa are extracted and stored in individual registers. This allows easy independent manipulation.

Mathematical Operations

- Modern CISC CPUs often provide hardware support for floating point numbers:
 - special registers for storing floating point values
 - special instructions for performing floating point operations
- Such CPUs may have an onboard FPU (floating point unit), or may emulate one using the ALU
- Older CPUs (up to mid-90's) may support connection of an optional floating point coprocessor

Mathematical Operations

- Some CPUs provide no support at all
- If no hardware support exists, floating point operations must be implemented in software
- Note: at a minimum, I/O routines must be implemented in software.
- In a high level language, these are provided as part of a standard library

Unpacking an IEEE Number

- In order to perform mathematics on an IEEE number it has to be unpacked, i.e. decomposed into its component parts
- Each component needs to be moved into a separate location: register or memory location
 - Sign into a byte or word
 - Biased exponent into a byte or word
 - Mantissa into a longword with the hidden bit restored
- Where to put the mantissa's 24 bits in the longword?
 - Typically with bit 23 in bit 30 of the longword. This has 1 leading extra bit in front and 7 trailing extra bits
 - These extra bits are called guard bits

Addition

1. Check for zeros – then result is other operand
2. Unpack both operands
 - Aside: can you directly add 1.23×10^5 and 7.89×10^{-3} ?
 - No – what needs to be done first?
3. Align mantissas
 - This means shifting one of the mantissas.
 - Which one and in what direction?
 - The smaller one and to the right, because if the shift is large enough it is possible to lose significant bits, but they will be least significant bits. Plus you have 7 extra bits in this direction

Addition

4. Add the mantissas
 - Realize that these are signed magnitude values so signed magnitude addition must be performed
5. Normalize the result
 - Why?
 - Try $1.0 + 1.0 = 2.0$
 - The MSbit has carried into the upper guard bit
 - $1.0 + -0.9999999 = 0.0000001$
 - The MSbit is now many bits to the right of bit 23
 - In steps 4 and 5 may need to handle overflow or underflow (infinity, NaN or denormalized)

Addition

6. Round result
 - Handling over/underflow and rounding are beyond the scope of this class
7. Pack components into IEEE format

Other Mathematical Operations

- Subtraction
 - Add the negation (i.e. $X - Y = X + -1 * Y$)
- Multiplication
 - Simpler than addition, why?
 - Multiply the mantissas
 - Add the exponents
 - Renormalize
 - XOR signs to get result sign
 - $1.23 \times 10^5 \times 7.89 \times 10^{-3} =$
- Division
 - Similar to multiplication

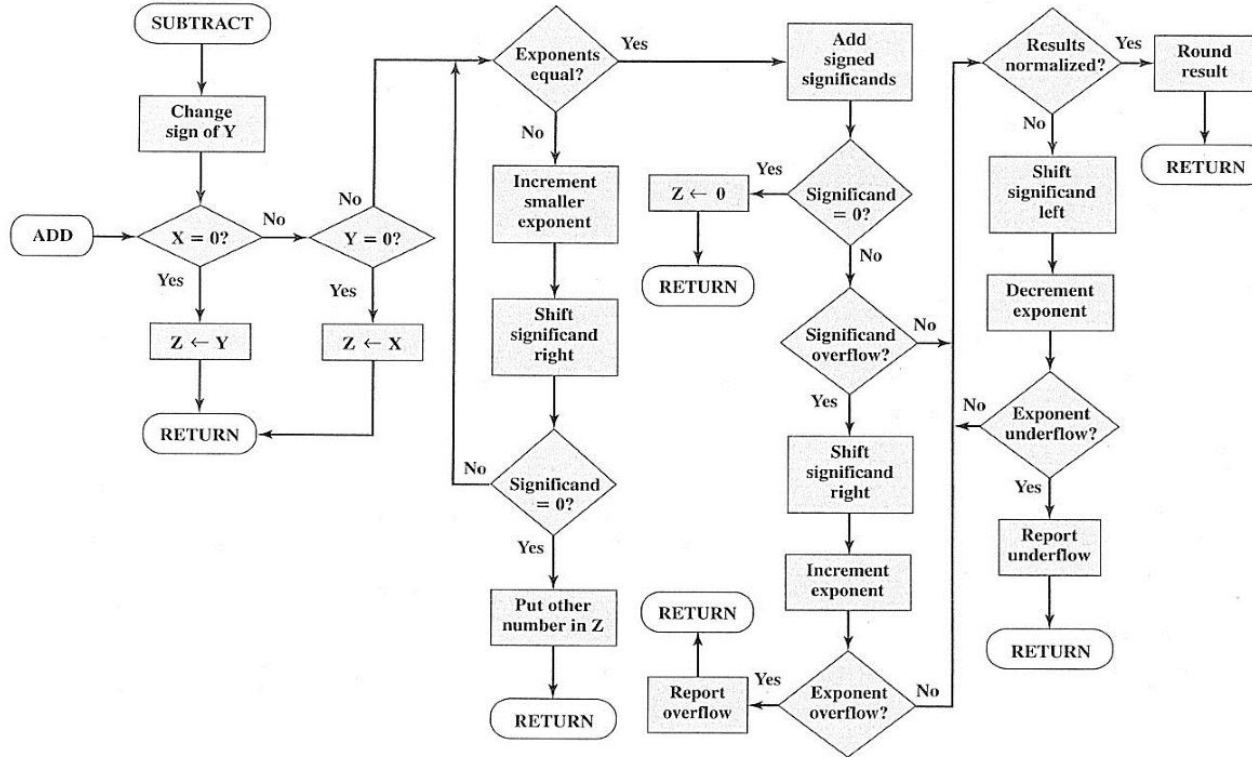


Figure 9.22 Floating-Point Addition and Subtraction ($Z \leftarrow X \pm Y$)