Drake Bridgewater & Ryan Phillips
April 14, 2014
CS 472 HW 2

(2.5)

Calculations are to be performed to a precision of 0.001%. How many bits does this require?

$.001 = 2^{-x}$

x $= 9.9$

It will require at least 10 bits to achieve this precision. This doesn't take into account the size of the numbers, as it will require extra bits to describe the possible range of magnitudes.

(2.13)

Perform the following calculations in the stated bases.

a.

$$\begin{array}{rl} & 00110111_2 \\ + & 01011011_2 \\ \hline & 10010010_2 \end{array}$$

b.

$$\begin{array}{rl} & 00111111_2 \\ + & 01001001_2 \\ \hline & 10001000_2 \end{array}$$

c.

$$\begin{array}{rl} & 00120121_{16} \\ + & 0A015031_{16} \\ \hline & A135152_{16} \end{array}$$

d.

$$\begin{array}{rl} & 00ABCD1F_{16} \\ + & 0F00800F_{16} \\ \hline & FAC4D2E_{16} \end{array}$$

(2.14)

What is arithmetic overflow? When does it occur and how can it be detected?

Arithmetic overflow is what occurs when the result of an arithmetic calculation will not fit in the provided number of bits. In this case, the carry of the sum *overflows.* It can be detected by checking the leading bits of the inputs, outputs and the type of operation. If the operation is addition, and either one or both of the inputs have a 1 in the most significant bit, and the answer does not, then an overflow has occurred. Similarly, if the operation is subtract, and both numbers are negative, the same check can be performed.

(2.16)

Convert 1234.125 into 32-bit IEEE floating-point format.

Convert to binary	10011010010.001_2
Normalize	1.0011010010001_2
Number bits shifted	11_10 or 1001_2
Adjust because of bias	10001001_2
sign digit	0_2
exponent	10001001_2
significand	$00110100100010000000000_2$

Or in hexidecimal: $00449a44_{16}$

(2.17)

What is the decimal equivalent of the 32-bit IEEE floating-point value CC4C 0000

First, as binary:

1 10011000 10011000000000000000000

$negative, 2^{25}, fraction : 0.59375$

$-1.59375 * 2^{25} = -53477376$

(2.22)

What is the difference between a *truncation* error and a *rounding* error?

Truncation is the simplest method of maintaining a constant number of bits in the significand; it drops any bits that don't fit. This means that a truncated number will always be less than what it should be, which is a biased error.

Rounding is the other method. The bits that don't fit in the significand are rounded, and this is reflected in the least significant bit (and it propagates through the rest of the bits if necessary). Rounding error just refers to difference between a rounded value and the original value. [1, p.78-79]

(2.40)

Draw a truth table for the circuit in Figure P2.40 and explain what it does.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

This is a way of making an XOR gate out of NAND gates.

(2.45)

It is possible to have n-input AND, OR, NAND, and NOR gates, where n>2. Can you have an n-input XOR gate for n>2? Explain your answer with a truth table.

XOR

A	B	$A \oplus B$
1	1	0
1	0	1
0	1	1
0	0	0

No, by definition you can't have an XOR gate with 3 or more inputs.

References

[1] Alan Clements. *Computer Organization and Architecture*. Global Engineering: Christopher M. Shortt, themes and variations edition, 2014.