Department of Electrical and Computer Engineering
McGill University

ECSE-322 Computer Engineering

18 January 2008

Problem Set 1 - Solutions

1. Floating point representation:

(a) Show the IEEE-754 binary representation of the number –0.75 (i.e.- ¾ in base 10) using the single precision format.

Solution:

For IEEE, bit 31 is the sign but, bits 30-23 are the exponent bits and are stored in excess-127, and the rest are for the mantissa using hidden bit normalization.

$-0.75 = -3/4 = -3 * 2^{-2} = -1.5 * 2^{-1}$

The resultant representation is:
10111111010000000000000000000000, where the sign=-1, the exponent=-1 and the mantissa=1.5

(b) What floating point number (in IEEE-754 format) is represented by the following:
00110000001000000000000000000000

Solution:

$00110000001000000000000000000000 = -1^0 \times 2^{(96-127)} \times (1 + 2^{-2})$
$= 5.820766 \times 10^{-10}$

(c) Represent $0.125\ 16^5$ and -0.125×16^{-5} in USASI.

Solution:

Recall that, for USASI,
bit 0 = sign bit
bits 1-7 = exponent bits stored in excess-64
bits 8-31 mantissa bits (six hexadecimal digits normalized fraction)

The exponent 5 in excess 64 is equal to 69 (1000101)
The mantissa 0.125 is 1/8 = 2/16 = $2*16^{-1}$, thus 0.2 in hexadecimal (hence 0100 …)
The exponent –5 in excess 64 is equal to 59. Thus resulting representation follows:

0.125×16^5 is represented as 0 1000101 0010 0000 0000 0000 0000 0000
-0.125×16^{-5} is represented as 1 0111011 0010 0000 0000 0000 0000 0000

(d) Identify how infinity, not a number, and 0 are represented in IEEE-754 and USASI.

Solution:

IEEE-754:
Infinity: all exponent values are =1, all mantissa values are = 0.
NaN : all exponent values are = 1, mantissa = non zero
0: all bits are = 0

USASI:
Cannot represent infinity or NAN
0: Mantissa bits are 0

2. Determine the maximum relative error and minimum and maximum values of a real number stored using the following floating point formats:

(a) IEEE 754,
(b) USASI.

We define the relative error in terms of the difference between the number and its representation as follows:

Let rep(n) be the representation of n.
The relative error is err(n) = |rep(n) – n| / |n|, where |n| represents the absolute value of n.

(Note: This question is intended to be particularly challenging.)

Solution:

The floating point format of a number n is:

$$n = (Sign)(Base^{\exp})(Mantissa)$$

In digital floating point representations, the mantissa can only be represented up to a finite precision. Error is introduced because the least significant bits are dropped:

$$rep(n) = (Sign)(Base^{\exp})rep(Mantissa),$$
$$Mantissa \neq rep(Mantissa)$$

The relative error measures representation error of a number n relative to the size n:

$$err = \frac{|n - rep(n)|}{|n|}$$
$$= \frac{(Sign)(Base^{\exp})|Mantissa - rep(Mantissa)|}{(Sign)(Base^{\exp})|Mantissa|}$$
$$= \frac{|Mantissa - rep(Mantissa)|}{|Mantissa|}$$

Note that only the mantissa is relevant in the relative error calculation. To maximize the relative error, we want to find a mantissa that maximizes $|Mantissa - rep(Mantissa)|$ and minimizes $|Mantissa|$. This is the mantissa whose representable portion is the smallest possible, and whose unrepresentable portion is the largest possible.

(a)
The smallest representable mantissa is 1 (i.e., the mantissa field is all zeroes).
Assuming rounding of the least significant bits, the largest unrepresentable portion is 2^{-24}.
The maximum relative error is:

$$err = \frac{2^{-24}}{1 + 2^{-24}} \approx 2^{-24}$$

The largest exponent represented in IEEE 754 is 11111110 in excess-127, i.e. 254 – 127 = 127. The largest mantissa is 1.111… (all 23 bits in the mantissa field are 1) which is $2 - 2^{-23}$. The largest number is $2^{127} * (2-2^{-23}) = 2^{128} - 2^{104}$. The smallest number represented in IEEE 754 is therefore $-2^{128} + 2^{104}$.

Some more estimations, not required: The smallest exponent is 00000001 in excess-127, i.e.. 1 – 127 = -126, and the smallest mantissa is 1. The smallest *positive* number represented in IEEE 754 is therefore 2^{-126}. The dynamic range is approximately $2^{128}/2^{-126} = 2^{254}$.

(b)
The smallest representable mantissa is 2^{-4}, corresponding to mantissa 0001 0000 0000 0000 0000 0000 (the first hexadecimal digit must be non-zero for any non-zero number in USASI).
Assuming rounding of least significant bits, the largest unrepresentable portion is 2^{-25}.
The maximum relative error is:

$$err = \frac{2^{-25}}{2^{-4} + 2^{-25}} \approx 2^{-21}$$

The largest exponent represented in USASI is 1111111 in excess-64, i.e. 127-64 = 63. The largest mantissa is 0.111… (all 24 bits in the mantissa field are 1) which is $1 - 2^{-24}$. The largest number represented in USASI is $16^{63} * (1-2^{-24}) = 2^{(4*63)}(1-2^{-25}) = 2^{252}(1-2^{-25}) = 2^{252} - 2^{228}$. The smallest number represented in USASI is therefore $-2^{252} + 2^{228}$.

Some more estimations, not required: The smallest exponent is 0000000 in excess-64, i.e.. –64, and the smallest mantissa is 1/16. The smallest *positive* number represented in USASI is

therefore $16^{-64} * 16^{-1} = 16^{-65} = 2^{(-4*65)} = 2^{-160}$. The dynamic range is approximately $2^{252}/2^{-160} = 2^{512}$.

3. Hashing:

Consider a list of words to be placed into an array of size M=11 using the following hash function:

h(x) = (sum of the ASCII decimal values of the first and last letters of the word) mod M

(a) Draw the resulting array (referred to as a hash table) after inserting, in order, the following words *ibex, hare, ape, bat, koala, mud, dog, carp, stork* if collisions are dealt with through linear (sequential) probes.

Solution

h(ibex) = (105+120)mod 11 = 225 mod 11 =5
h(ape) = (97+101)mod 11 = 198 mod 11 = 0
h(bat) =(98+116)mod 11 = 214 mod 11 =5 – collision! 6
h(koala) = (107+97)mod 11= 204 mod 11 =6 – collision! 7
h(mud) = (109+100)mod 11= 209 mod 11 =0 – collision! 1
h(dog) = (100+103)mod 11= 203 mod 11 = 5 – collision! 6- collision! 7 – collision - 8
h(carp) = (99+112)mod 11= 211 mod 11 = 2
h(stork) =(115+107)mod 11= 222 mod 11 =2 – collision! 3

0	*ape*
1	*mud*
2	*carp*
3	*stork*
4	
5	*ibex*
6	*bat*
7	*koala*
8	*dog*
9	
10	

(b) How many collisions does this result in?

Solution:

7 collisions

(c) Draw a picture of the resulting hash table that uses bucket hashing instead.

Solution:

Bucket Hashing			
0	*ape*	*mud*	
1			
2	*carp*	*stork*	
3			
4			
5	*ibex*	*bat*	*dog*
6	*koala*		
7			
8			
9			
10			

4. A problem commonly encountered with linear hashing at medium levels of occupancy is "clustering": data items tend to gather in clusters of consecutive locations in the hash table. Explain what causes the phenomenon of clustering, and which hashing functions are less likely to create clustering.

Solution:

The trouble with a linear family of hashing functions is that, after a collision, two streams of hashing are merged, thus doubling the likelihood of further collisions at the respective location. To see that, notice that h1(i)=h0(j) $\Rightarrow$ h2(i)=h1(j) $\Rightarrow$ h3(i)=h2(j) $\Rightarrow$ …; hence, if i takes j's location, then other items in i's family will compete with items in j's family for locations h1(i), h2(i), h3(i), etc, which are the same as h0(j), h1(j), h2(j), etc, and the items from two streams will cluster in these locations. (Recall that linear hashing uses hk(i) = (i + k) mod N, where N is the capacity of the hash table.)

For the same reason, clustering will also occur with variations of linear hashing, such as hk(i) = [(i + k) / N], where [/] yields the quotient of integer division.

To avoid clustering, select a family of hash functions that avoid complete overlap of locations that are tried consecutively for different items; for example, take hk(i) = (i + 2^k) mod N, where N

is the capacity of the hash table. The terms 2^k can be computed cheaply by shifting, or can be vectored.