Chapter One
ENGR 3703 Notes

<u>1.1 Background</u>

Please read this section… we have discussed this in class. Bottom line: lots of engineering problems can't be solved by hand because they are too hard (or the math does not exist to solve them) or they would take too long… so we use computers to <u>*estimate*</u> and <u>*automate*</u>.

<u>1.2 Representation of Numbers on the Computer</u>

<u>*Base-10 (Decimal)*</u>

$$16{,}234.02 = 1\,x\,10^{4} + 6\,x\,10^{3} + 2\,x\,10^{2} + 3\,x\,10^{1} + 4\,x\,10^{0} + 0\,x\,10^{-1} + 2\,x\,10^{-2}$$

We could rewrite in scientific notation as:

$1.623402\,x\,10^{4}$ OR *d.dddddd x 10* ^{p}

Base-2 (Binary)

$$28 = 1\,x\,2^{4} + 1\,x\,2^{3} + 1\,x\,2^{2} + 0\,x\,2^{1} + 0\,x\,2^{0} = 11100$$

To write the equivalent scientific notation in binary…

$$1.bbbbbb\,x\,2^{bb}$$

Note we will call the *.bbbbb* part the *mantissa* and *bb* exponent part, simply *exponent*.

Other Note: <u>*binary digit = bit*</u>

Other Note: <u>*8 bits is a byte.*</u>

First you have to divide by the largest factor of the form *2*^{n} where *n* is an integer… this will make the first binary digit *1*. At this point we will use *50* as our number so you can compare to the book example.

$\frac{50}{2^{5}}\,x\,2^{5} = 1.5625\,x\,2^{5}$ but how do we put the *.5625* in binary??

Let's try the following:

$1\,x\,2^{-1} + 0\,x\,2^{-2} + 0\,x\,2^{-3} + 1\,x\,2^{-4} = \frac{1}{2} + \frac{1}{16} = 0.5 + 0.0625 = 0.5625$ Note… you can learn to do this very soon!

So, in the end…

$1.1001\, x\, 2^{101}$ in fully binary form.

Another example.

IEEE 754 Standard
https://ieeexplore.ieee.org/document/4610935/

Representation of floating point data

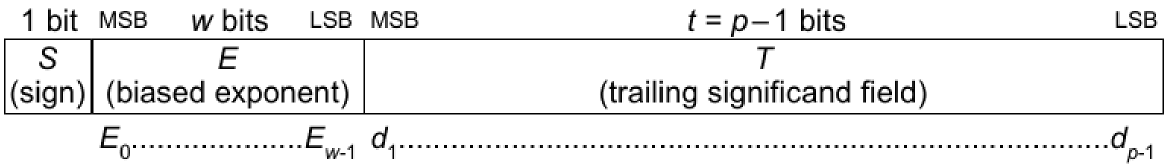
- $(-1)^{sign} \times b^{exponent} \times significand$
- $+\infty, -\infty$
- qNaN (quiet), sNaN (signaling)

Sets of floating-point numbers (FPN) are determined by:

- *b* = radix (or base), 2 or 10
- *p* = number of digits in the significand (precision)
- *emax* = maximum exponent
- *emin* = minimum exponent (emin = 1-*emax*)

FPN in decimal form:

1. $(-1)^s \times b^e \times m$ *m* is scientific format
   - *s* is 0 or 1
   - *e* is any integer: *emin* ≤ *e* ≤ *emax*
   - *m* is a digit string
     - d_0• *d1d2*...d_{p-1} where d_i is an integer $0 \le d_i < b$
   - $+\infty, -\infty$
   - qNaN (quiet), sNaN (signaling)
2. $(-1)^s \times b^q \times c$
   1. *s* is 0 or 1
   2. q is any integer: *emin* ≤ *q+p-1* ≤ *emax*
   3. *c* is a digit string
      1. $d_0d_1d_2...d_{p-1}$ where d_i is an integer $0 \le d_i < b$ ($0 \le c < b^p$)
   4. $+\infty, -\infty$
   5. qNaN (quiet), sNaN (signaling)



The above format requires

- 1-bit sign *(0 or 1)*
- *w-bit* biased exponent *(E = e + bias)*
- *(t=p-1)-bit* trailing significand digit string, $T = d_0d_1d_2...d_{p-1}$

For ***binary16*** representation:

- k = 16 (total storage width)
- p = 11
- emax = 15
- bias = 15
- w = 5
- t = 10

For ***binary32*** representation:

- k = 32 (total storage width)
- p = 24
- emax = 127
- bias = 127
- w = 8
- t = 23

For ***binary64*** representation:

- k = 64 (total storage width)
- p = 53
- emax = 1023
- bias = 1023
- w = 11
- t = 52

For ***binary128*** representation:

- k = 128 (total storage width)
- p = 113
- emax = 16383
- bias = 16383
- w = 15
- t = 112

Here is how to convert to decimal to binary16,32,64,128:
(this was adapted from: http://sandbox.mc.edu/~bennet/cs110/flt/dtof.html)

The rules for converting a decimal number into floating point are as follows:

1. Convert the absolute value of the number to binary, perhaps with a fractional part after the binary point. This can be done by converting the integral and fractional parts separately. The integral part is converted (see example below). The fractional part can be converted by multiplication. *This is basically the inverse of the division method: we repeatedly multiply by 2, and harvest each one bit as it appears left of the decimal.*

2. Append $\times 2^0$ to the end of the binary number (which does not change its value).

3. Normalize the number. Move the binary point so that it is one bit from the left. Adjust the exponent of two so that the value does not change.

4. Place the mantissa into the mantissa field of the number. Omit the leading one, and fill with zeros on the right.

5.Add the bias to the exponent of two, and place it in the exponent field. The bias is $2^{k-1} - 1$, where k is the number of bits in the exponent field. For the eight-bit format, $k = 3$, so the bias is $2^{3-1} - 1 = 3$. For IEEE 32-bit, $k = 8$, so the bias is $2^{8-1} - 1 = 127$.

6.Set the sign bit, 1 for negative, 0 for positive, according to the sign of the original number.

Convert *2.625* to ***binary16***:

1. Convert the *2* to binary… *10*
   Convert .625 to binary
   1. 0.625 x 2 = *1*.25 → 1 (msb)
   2. 0.25 x 2 = *0*.5 → 0
   3. 0.5 x 2 = *1*.0 → 1
   4. 0.0 x 2 = *0*.0 → 0…
   5. So – 0.625 = 0.1010…
   6. we will end up needing more zeros on the end for *binary16*
2. We now have 10.1010 – express as $10.101x2^0$
3. Normalize this $10.101x2^0$ → $1.0101x2^1$
4. Mantissa = 0101…
5. Exponent = 1+bias = 1+15 = 16 → 10000
6. Sign = 0.

Result is: 0 (s) 10000 (E) 0101000000 (T)

Convert -1313.3125 to ***binary16***:

7. Convert the 1313 to binary…
   1313/2 = 656 R=*1* → 1 (lsb)
   656/2 = 328 R=*0* → 0
   328/2 = 164 R=*0* → 0
   164/2 = 82 R=*0* → 0
   82/2 = 41 R=*0* → 0
   41/2 = 20 R=*1*→ 1
   20/2 = 10 R=*0* → 0
   10/2 = 5 R=*0* → 0
   5/2 = 2 R=*1* → 1
   2/2 = 1 R=*0* → 0
   1/2 = 0 R=*1* → 1 (msb)
   10100100001
   Convert 0.3125 to binary
   0.3125x2 = *0*.625 → 0
   0.625x2 = *1*.25 → 1
   0.25x2 = *0*.5 → 0
   0.5x2 = *1*.0 → 1
   0.0x2 = *0*.0 → 0…

.01010...

8. Now we have 10100100001.01010 = 10100100001.01010x2^0
9. Normalize as 10100100001.01010x$2^{0=}$ 1.010010000101010x2^{10}
10. Mantissa = 010010000101010 (only use first 10 bits)
11. Exponent = 10 + bias = 10 + 15 = 25 → 11001
12. Sign = 1.

Result is: 1 (s) 11001 (E) 0100100001 (T)