exponent: E

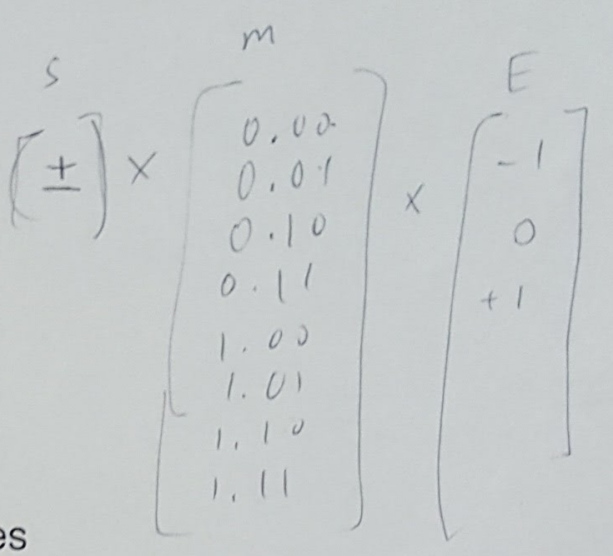
Example 1 (1):

```
b =  2
p =  3
L = -1
U =  1
```

start enumerating possibilities:

```
+-  m     E
+- 0.00  -1  -> 0
+- 0.00   0  -> 0
+- 0.00  +1  -> 0
+- 0.01  -1  -> 0.001
+- 0.01   0  -> 0.01
+- 0.01  +1  -> 0.1
+- 0.10  -1  -> 0.01
+- 0.10   0  -> 0.1
+- 0.10  +1  -> 1.0
```

duplicates!



$2 \times 8 \times 3 = 48$

In general, number of possibilities
2 * b^p * (U - L + 1)
but
- lots of duplicates
- non-unique representation

Normalization

- require the leading digit to be non-zero
- so mantissa, m
  1 <= m < b
- nice because:
  - representation is now *unique*
  - don't waste digits on any leading 0's
  - for binary base, leading digit must be 1
    - so don't need to store it, just assume number is 1.d1d2..dp
      - gain an extra bit of precision!

$b=10 \quad (10.00)_{10} = (10)_{10}$
$b=2 \quad (10.00)_2 = (2)_{10}$
$b=3 \quad (10.00)_3 = (3)_{10}$

Properties

- finite and discrete system
- finite: how many (normalized) numbers can be represented?

count them:

```
2 * (b - 1) * b^(p-1) * (U - L + 1) + 1
```

$\pm$ d_0 $d_1 \cdots d_{p-1}$ E 0

Ex. 1 $2 \cdot 1 \cdot 2^2 \cdot 3 + 1 = 25$

- what's the smallest (positive) normalized number? or "underflow level (UFL)"

Ex. 1 UFL = .5

```
1.0 ... 0 * b^L = b^L
```

- what's the biggest normalized number? or "overflow level (OFL)"

```
(b-1).(b-1) ... (b-1) * b^U
    = ( b - b^(-(p-1)) ) * b^U
    = ( 1 - b^(-p) ) * b^(U+1)
```

Ex 1

$OFL = (1 - 2^{-3}) \cdot 2^2 = (1 - \frac{1}{8}) \cdot 4 = 4 - \frac{1}{2} = 3.5$

```
Example 1 (2):
--------------
      b =  2
      p =  3
      L = -1
      U =  1
```

- number of normalized

```
2 * (b - 1) * b^(p-1) * (U - L + 1) + 1
    = 2 * (2 - 1) * 2^(3-1) * (1 - -1 + 1) + 1
    = 2 * 1 * 4 * 3 + 1
    = 25
```

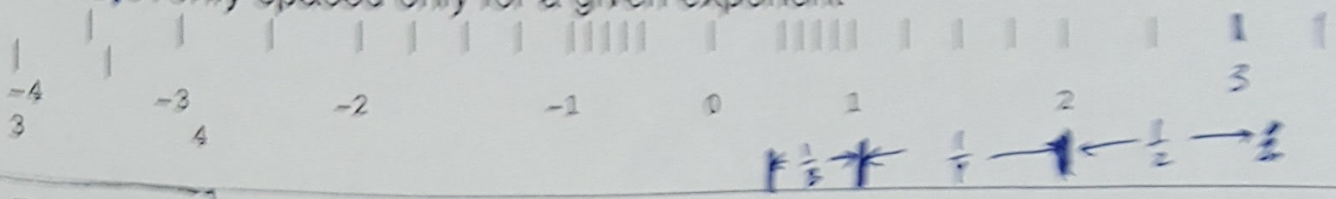
- UFL

```
b^L
    = 2^-1
    = .5
```

- OFL

```
( 1 - b^(-p) ) * b^(U+1)
  = ( 1 - 2^(-3) ) * 2^2
  = 3.5
```

PICTURE of representable numbers
- note, evenly spaced only for a given exponent



Rounding

- floating point system is discrete!
  - not all real numbers representable
  - those that are called "machine numbers"
  - others must be *rounded*

```
x  <-- fl(x)
```

- leads to "rounding error" or "roundoff error"

- How to round?
1. Chop - truncate digits - "round to zero"
2. Round to nearest
   - in case of tie go to even

Example: Rounding

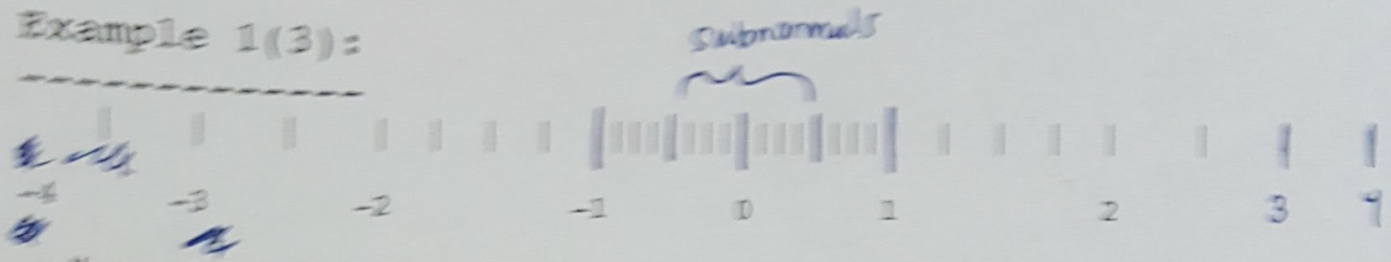
Number	Chop	Round to nearest
1.649	1.6	1.6
1.650	1.6	1.6 (tie - round to even)
1.651	1.6	1.7
1.699	1.6	1.7
1.749	1.7	1.7
1.750	1.7	1.8 (tie - round to even)
1.751	1.7	1.8
1.799	1.7	1.8

Machine Precision

Subnormals

- normalized numbers: gap between 0 and b^L
- fill in by allowing denormalized or subnormal numbers
- can make use of capacity for non-normalized numbers by allowing leading 0's
- though precision won't be full precision, since have leading 0's

Example 1(3):



- allows 6 new numbers around 0
- new smallest number is (0.01)_2 * 2^-1 = (0.125)_10

- called "gradual underflow" because we gradually lose precision

- implementation: reserved value of exponent field
  - leading bit not stored

Exceptional values

- Inf
  - dividing finite number by 0
  - exceeding OFL
- NaN
  - undefined operation 0/0, Inf/Inf, 0*Inf
- implemented through reserved values of exponent field

Floating Point Math

- adding or subtracting
  - match exponents first
  - must shift smaller number

Rounding

- floating point system is discrete!
  - not all real numbers representable
  - those that are called "machine numbers"
  - others must be *rounded*

```
x  <-- fl(x)
```

- leads to "rounding error" or "roundoff error"

- How to round?

1. Chop - truncate digits - "round to zero"
2. Round to nearest
   - in case of tie go to even

Example: Rounding

```
Number     Chop    Round to nearest
------    -----    ---------------
1.649       1.6    1.6
1.650       1.6    1.6 (tie - round to even)
1.651       1.6    1.7
1.699       1.6    1.7
1.749       1.7    1.7
1.750       1.7    1.8 (tie - round to even)
1.751       1.7    1.8
1.799       1.7    1.8
```

EXAMPLE : !!!!! warning: don't compare fp numbers with == !!!!!

octave-online.net

```
>> 4/3-1 == 1/3
ans = 0

>> single((4/3-1))==single(1/3)
ans = 1
```

```
>> (4/3-1)-1/3
ans = -5.5511e-17
```

right way to compare:

```
>> abs((4/3 - 1) - 1/3) <= 1e-16
ans = 1
```

Machine Precision

`eps_mach` ε_{mach}

- characterizes accuracy
- "machine epsilon", "machine precision", "unit roundoff"
- depends on rounding rule

```
_ . _ _ _ ...   _     x x x ... x
0 . 1 2 3 ... (p-1) p ...
```

- chop: (chop everything at and after b^p position)

```
b^-(p-1) = b^(1-p)
```

lose up to this amount

- round: (lose up to half of chop)

```
1/2 b^(1-p)
```

- ~~tells us~~ bound the max possible relative error in representation

$$\frac{|fl(x) - x|}{|x|} \leq \text{eps\_mach}$$

(for normalized #'s)

- check:

```
<= eps_mach * b^e / | x |
 = eps_mach * b^e / (m * b^e)
 = eps_mach / m
<= eps_mach
```

, because mantissa is at least 1

$m \geq 1$