ROUNDING

Rounding

- Eliminates LSB bits
- Need to reduce the number of bits due to word growth
  - For example, if we multiply two 5-bit words, the product will have 10 bits

    xxxxx × yyyyy = zzzzzzzzzz

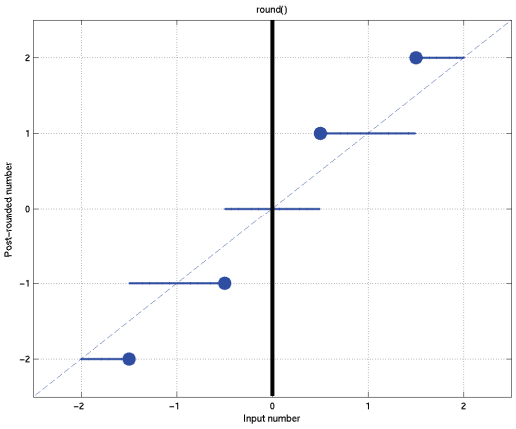
    and we likely don't want or need all that precision

Rounding

- ANSI/IEEE rounding more complex
- Matlab rounding
  1) round(·): towards nearest integer
     - Pos. and neg. numbers are rounded symmetrically about zero
     - Generally the best possible rounding algorithm
  2) fix(·): truncates towards zero
     - Pos. and neg. numbers are rounded symmetrically about zero
  3) floor(·): rounds towards negative infinity
  4) ceil(·): rounds towards positive infinity

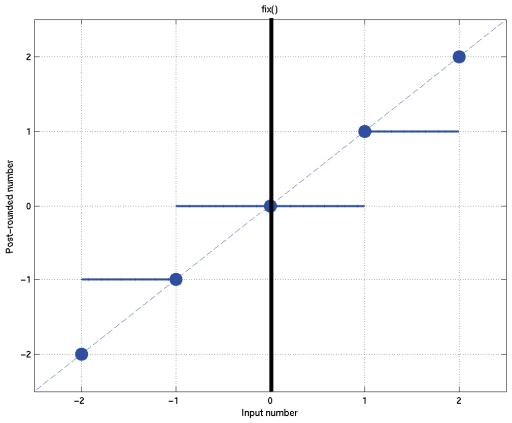
1) matlab round()

- One of the best rounding modes
- "Unbiased" rounding
- Symmetric rounding for positive and negative numbers
- Max error ½ LSB



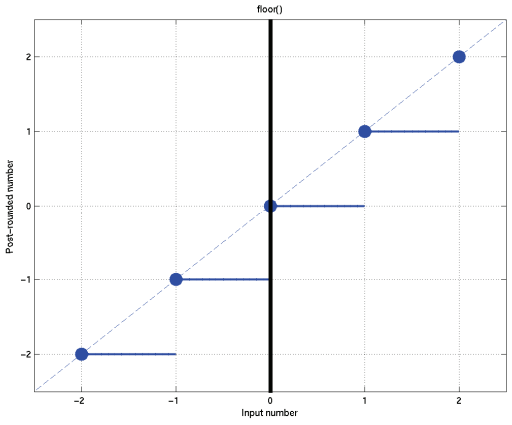
2) matlab fix()

- Truncates toward zero
- Numerical performance poor
- Symmetric rounding for positive and negative numbers
- Max error 1 LSB



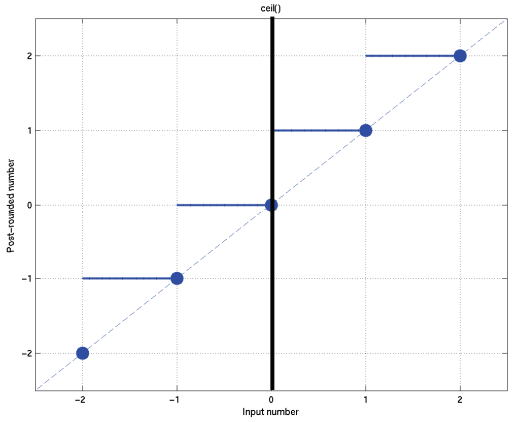
3) Truncation, or matlab floor()

- Numbers rounded down towards –infinity
- Numerical performance poor
- Very simple hardware
- `xxxxxx` in
  `xxxx--` out
- Max error 1 LSB



4) matlab ceil()

- Numbers rounded up towards +infinity
- Numerical performance poor
- Max error 1 LSB



Hardware Rounding

A. Easiest is truncation
   - `xxx.xxxxx`
     `xxx.xx---`
   - Maximum rounding error ~1 post-rounded LSB
   - Signed magnitude
     - Positive and negative numbers both truncate towards zero
     - Matlab fix(·)
   - 2's complement and unsigned
     - All numbers truncate towards negative infinity
     - Matlab floor(·)

Hardware Rounding

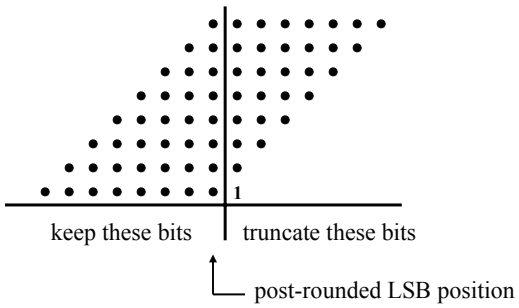
B. Better rounding numerically is to add ½ lsb and then truncate

```
          1
  + xxx.xxxxx
  -----------
    yyy.yyyxx
    yyy.yy---
```

- Our 5th rounding method
- Maximum rounding error ½ post-rounded LSB
- Two cases:
  a. When the input is xxxx.5000 (base 10) (or xxx.xx100 (base 2) in the example above)
     - Rounding is towards positive infinity (for both positive and negative numbers)
     - matlab ceil(·)
  b. Otherwise
     - Performs best rounding: matlab round(·)

Hardware Rounding

- Often not difficult to find a place to add the extra "1" if you plan ahead

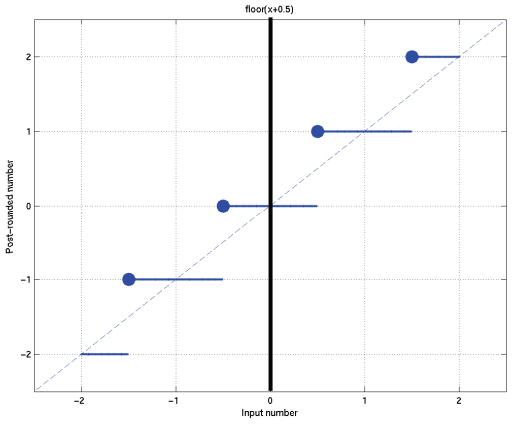


Hardware Rounding

- But there is a biased rounding of the `xxx.1000` cases
  - Is fine in many cases, especially when many bits are being rounded off (then the `xxxx.5000` case is less frequent)
  - Exact behavior depends on the number format being used:
    - Signed magnitude
      - Both positive and negative `xxxx.5000` cases round away from zero
    - 2's complement and unsigned
      - Both positive and negative `xxxx.5000` cases round towards positive infinity

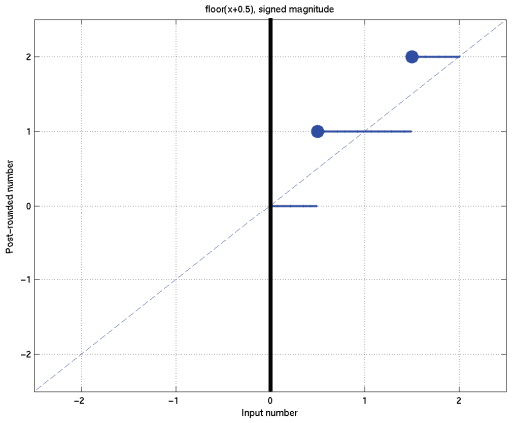
Add ½ LSB and Truncate 2's Complement

- matlab floor(x+1/2)
- Numerical performance often sufficient
- ```
      1
  + xxxxxx
  ------
    yyyyxx
  ------
    yyy---
  ```
- Biased rounding for 2's complement
- Max error ½ LSB


  ```

Add ½ LSB and Truncate Signed Magnitude

- matlab floor(x+1/2)
- Functions same as matlab round()
- Unbiased rounding for signed magnitude
- Max error ½ LSB



Unbiased Rounding

C. Unbiased rounding
  - For cases where a "DC" bias is unacceptable, positive and negative numbers must be rounded differently
  - Implement matlab round(·)
  - Basic algorithm (there are others)
    - Add ½ lsb normally
    - Do not add a ½ lsb when:
      1. The **result**(!) is negative, and
      2. The result is of the form `xxxxx.1000`
         - Equivalently, we could also specify values in the range `xxxxx.0001` to `xxxxx.1000` Do you see why?
    - Truncate as with method (B)

Unbiased Rounding

C. Unbiased rounding (continued)
- Although logically simple, this requirement can increase the critical path delay significantly
- If very high speed is required, it may be necessary to calculate the result two times: 1) with ½ LSB added in, 2) without ½ LSB added in. The correct answer is then selected with a mux when it is known which result is correct.