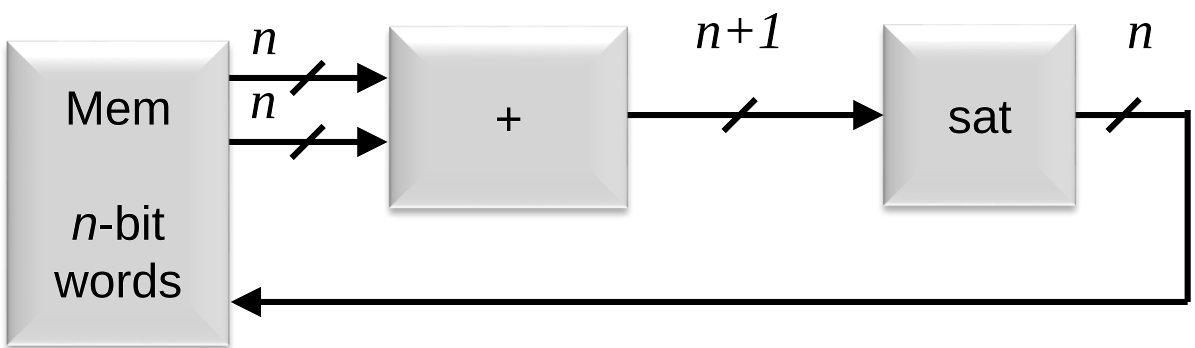
ROUNDING

Rounding

- Rounding is a fundamental method to reduce the size of a word, such as after arithmetic operations
  - For example to maintain the word width for memory storage



- Bits are removed from the LSB end of the word

```
XXXXXXXX
YYYYY
```

Rounding

- Another example: if we multiply two 5-bit words, the product will have 10 bits

  `xxxxx × yyyyy = zzzzzzzzzz`

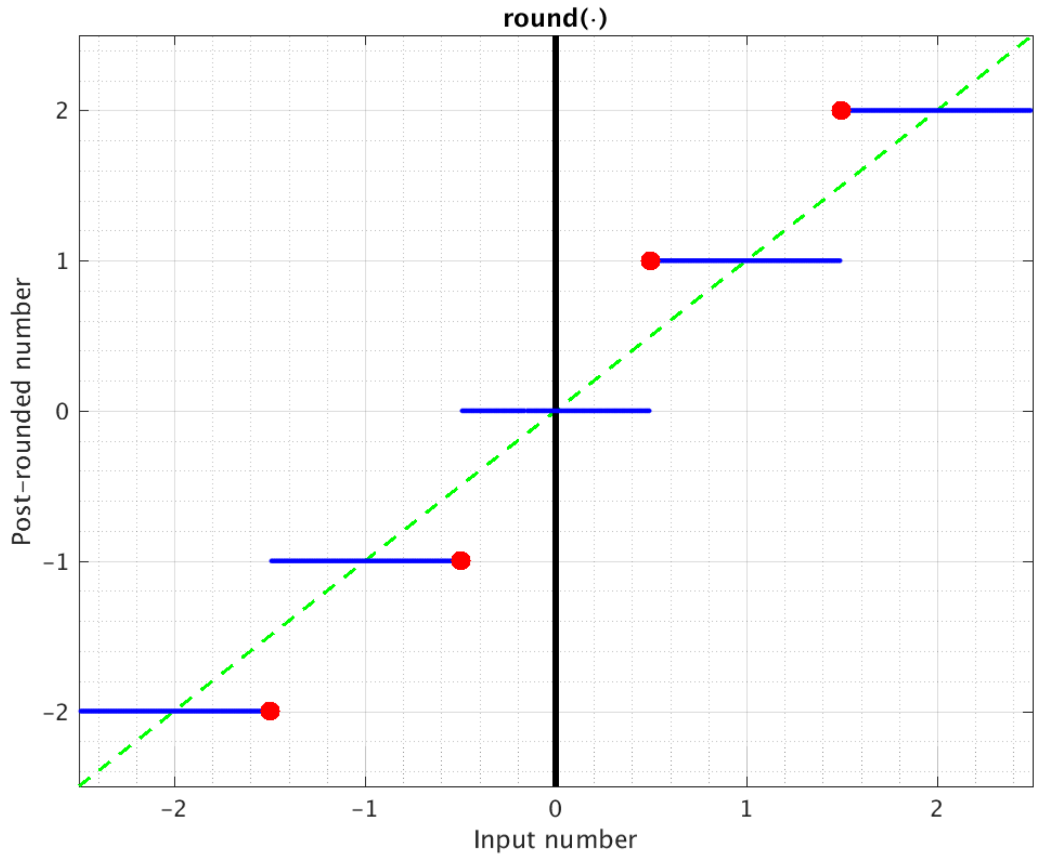
  and we likely can not handle or do not want or need all that precision
- More issues are present with signed data
- Issues vary for different formats:
  - unsigned
  - 2's complement
  - sign magnitude
  - etc.

Rounding

- Rounding modes in IEEE 754 are much more complex than what is commonly needed in digital signal processing systems
- There are four fundamental rounding modes whose matlab function names are:
  1) round(·): towards nearest integer
     - Generally the best rounding algorithm
  2) fix(·): truncates towards zero
  3) floor(·): rounds towards negative infinity
  4) ceil(·): rounds towards positive infinity

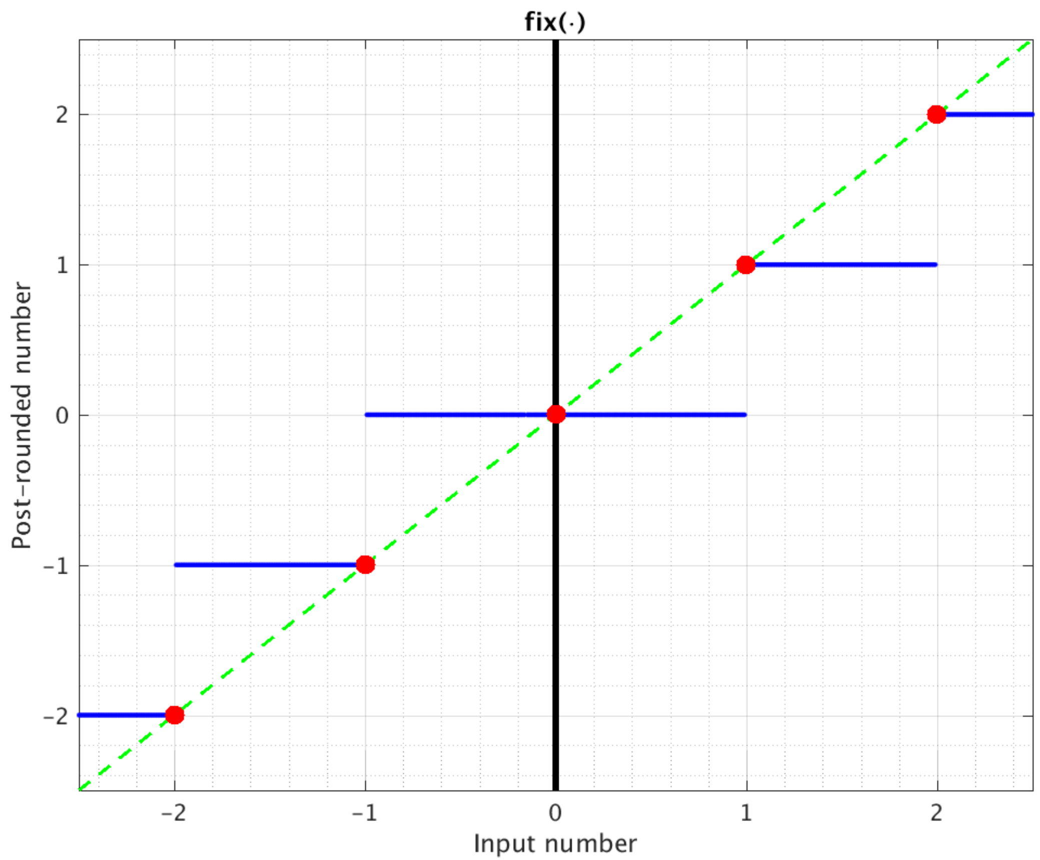
1) matlab round()

- Often the best general-purpose rounding mode
- “Unbiased” rounding
- Symmetric rounding for positive and negative numbers
- Max error ½ LSB



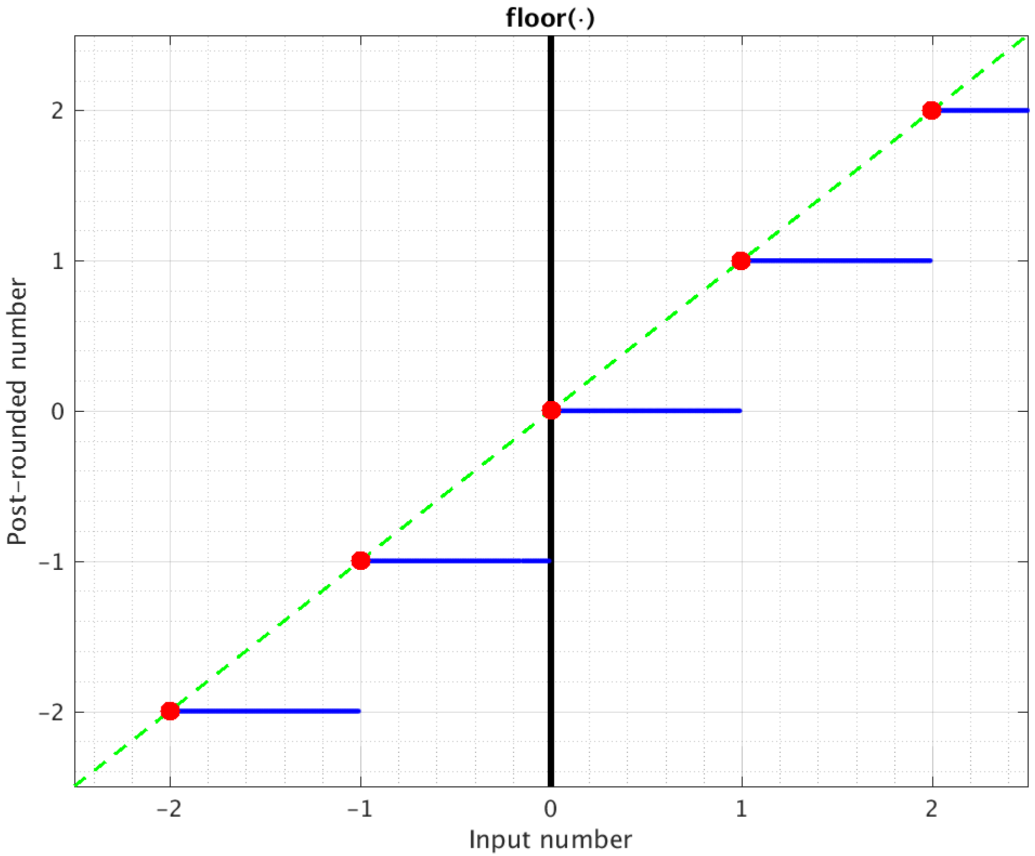
2) matlab fix()

- Truncates toward zero
- Numerical performance is poor
- Symmetric rounding for positive and negative numbers
- Very simple hardware for the magnitude of sign magnitude (simple truncation)
  - `xxxxxx` in
    `xxxx--` out
- Max error 1 LSB



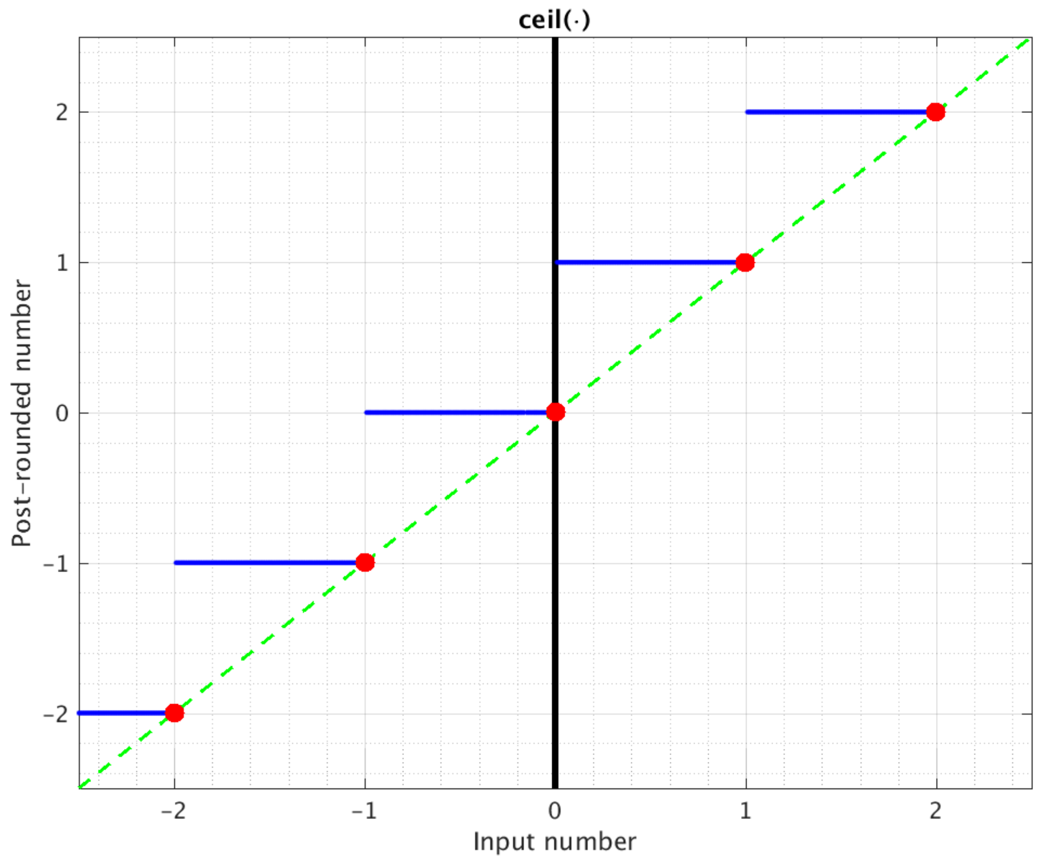
3) matlab floor()

- Numbers rounded down towards –infinity
- Numerical performance is poor
- Very simple hardware for 2's complement (simple truncation)
  - xxxxxxx in
xxxx-- out
- Max error 1 LSB



4) matlab ceil()

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



Hardware Rounding: A) Truncation

A. The easiest hardware method is truncation
   - `xxx.xxxxx`
     `xxx.xx---`
   - Simply neglect the truncated bits and remove all hardware which calculates only those bits
   - Maximum rounding error ~1 post-rounded LSB
   - Sign magnitude format numbers (obviously the magnitude portion)
     - Positive and negative numbers both truncate towards zero
     - Same as matlab `fix(•)`
   - 2's complement format numbers
     - All numbers truncate towards negative infinity
     - Same as matlab `floor(•)`
   - Unsigned format numbers
     - All numbers truncate towards zero (negative infinity)
     - Same as matlab `fix(•)` or `floor(•)`

Hardware Rounding: B) Add ½ LSB and Truncate

B. Method #5. Add ½ LSB (that is, one half of the LSB of the *output*) and then truncate

- This does not correspond to any of the matlab rounding functions for all binary formats
- Maximum rounding error ½ of the post-rounded LSB

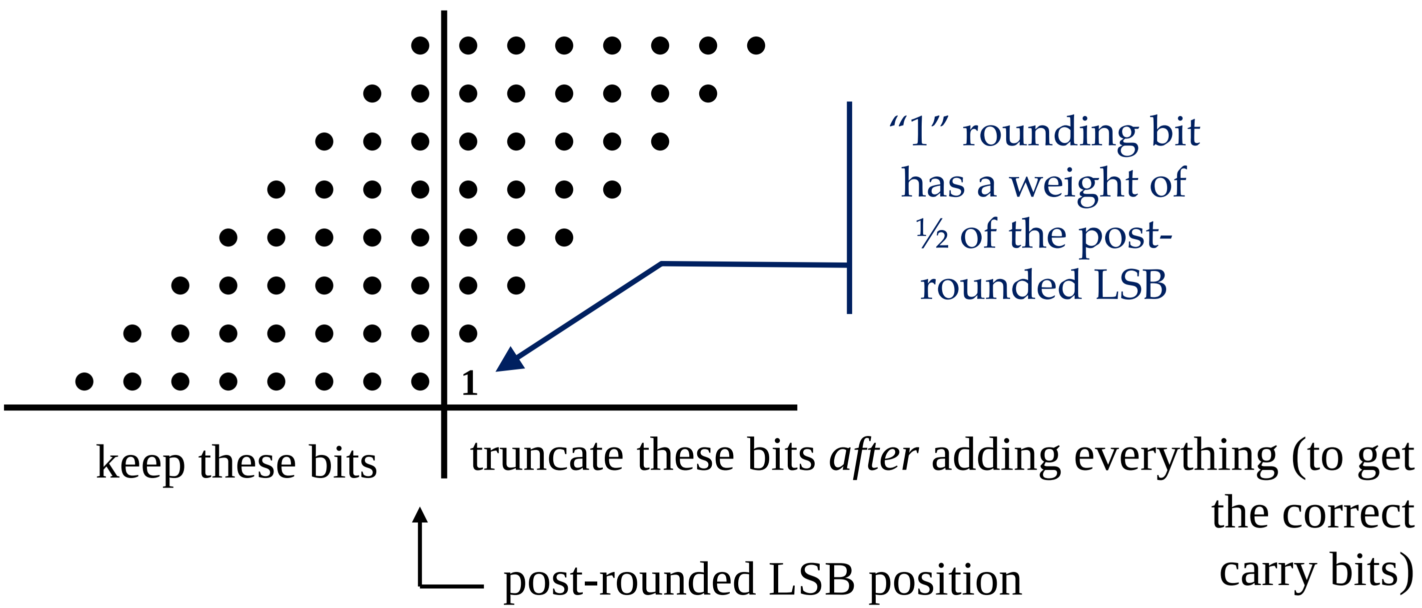
```
          1
+ xxxxx.xxx    input
  ---------
  yyyyy.yxx    intermediate sum
  ---------
  yyyyy.---    rounded output
```

rounding bit added here

the *post-rounded* LSB position

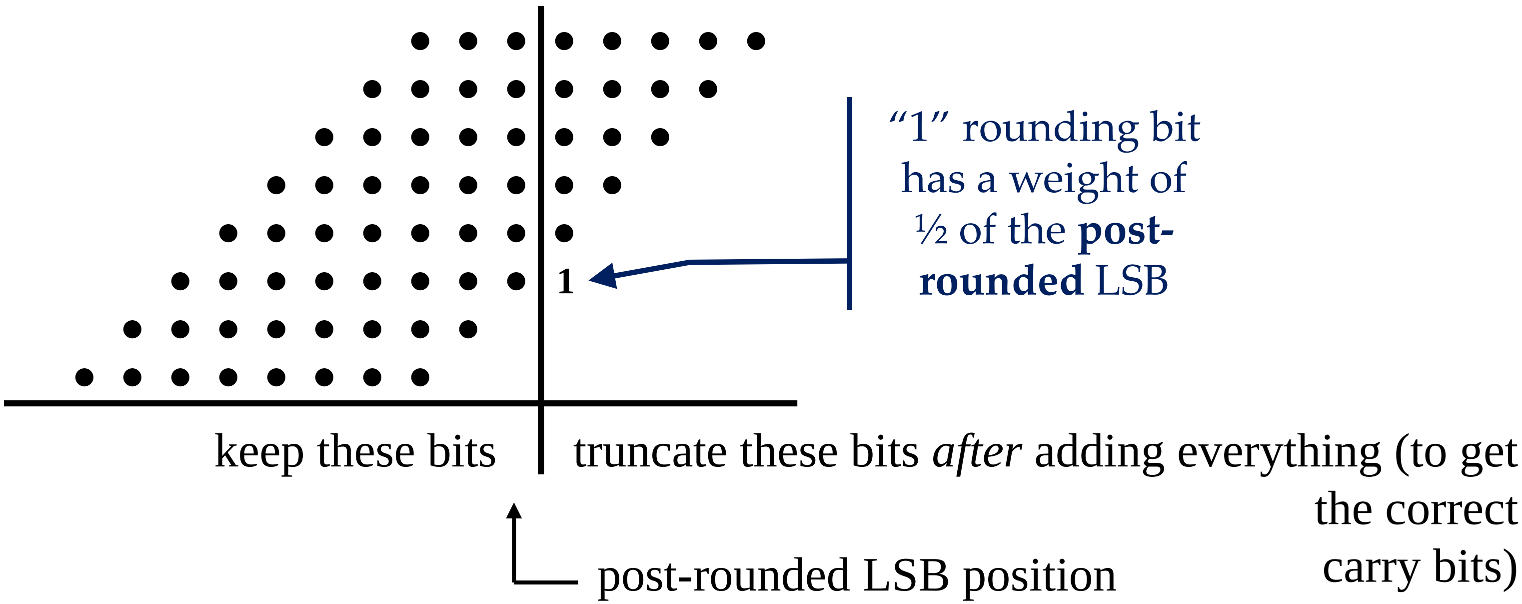
Hardware Rounding: B) Add ½ LSB and Truncate

- It is often not difficult to find a place to add the extra "1" in a complex datapath if you plan ahead



Hardware Rounding: B) Add ½ LSB and Truncate

- It is often not difficult to find a place to add the extra "1" in a complex datapath if you plan ahead

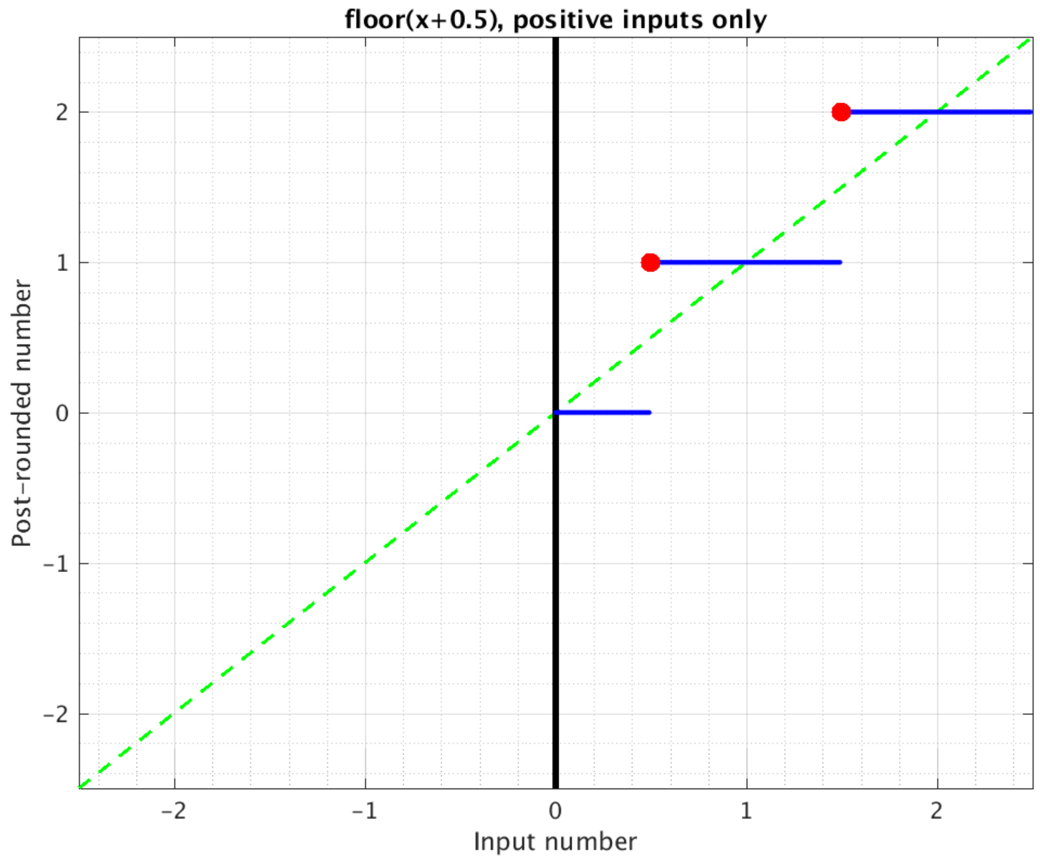


Hardware Rounding: B) Add ½ LSB and Truncate

- The exact behavior depends on the number format being used:
  - Unsigned
    - Unbiased rounding
  - Magnitude portion of Sign magnitude
    - Unbiased rounding
  - 2's complement
    - Both positive and negative `xxxx.1000` cases round towards positive infinity as explained previously
    - The behavior requires a little more analysis

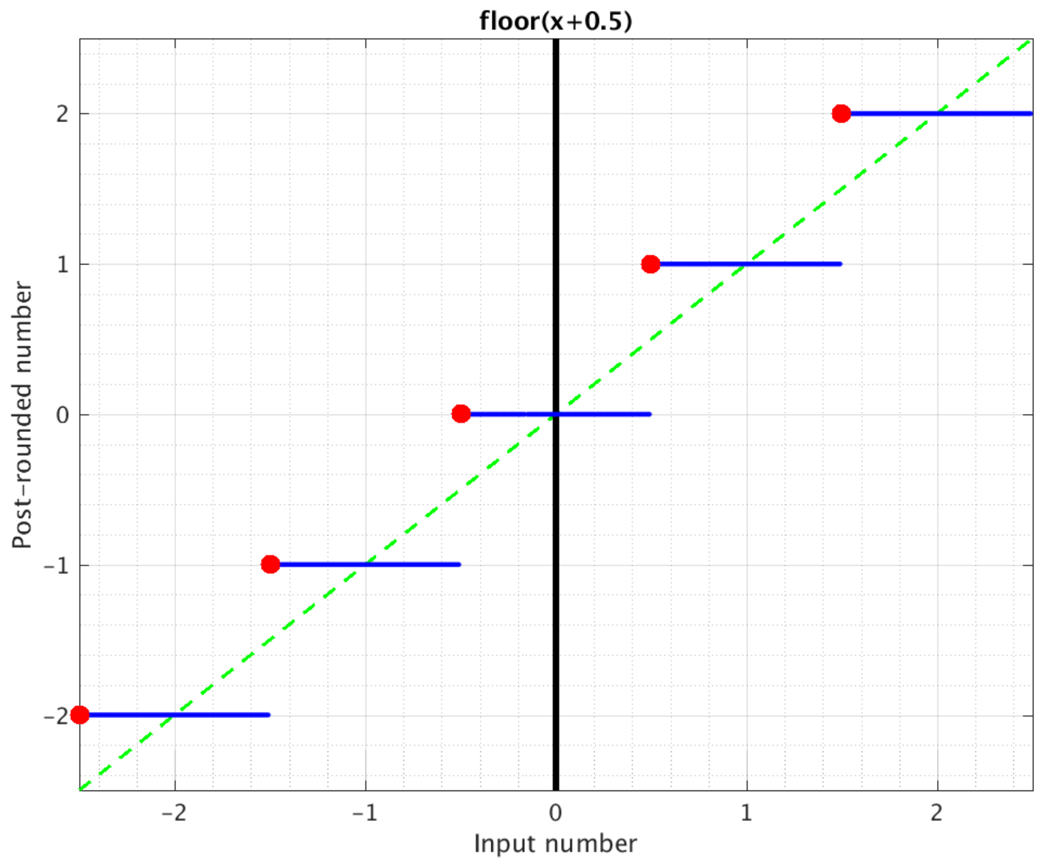
B) Add ½ LSB and Truncate
Unsigned, Sign Magnitude

- matlab **`floor(x+1/2)`**
- matlab **`fix(x+1/2)`**
- Both positive and negative xxxx.1000 cases round away from zero just like round()
- Functions the same as matlab round() which is the best of our four matlab rounding functions
- Max error ½ LSB



B) Add ½ LSB and Truncate
2's Complement

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


  ```

B) Add ½ LSB and Truncate *2's Complement*

- There are three key cases to consider for 2's complement:
  a. When the input is of the form xxxxx.100 (base 2) in the example above, and positive
     - Rounding is towards positive infinity which **is the same** as **`round(•)`**
  b. When the input is of the form xxxxx.100 (base 2) in the example above, and negative
     - Rounding is towards positive infinity which **is NOT the same** as **`round(•)`**
  c. Otherwise
     - It performs the same as matlab **`round(•)`**

B) Add ½ LSB and Truncate *2's Complement*

- The biased rounding in the `xxx.1000` cases when using 2's complement may be fine in many cases, especially when many bits are being rounded off, but if only a few bits are being rounded off, the case that differs from round() occurs more often.
- Example: `xxxxxx.x` rounded to `yyyyyy`

Pre-rounded value	Rounding action	Net effect
`(+) xxxxxx.0`	No change in value	Same as round()
`(+) xxxxxx.1`	Rounds to integer +0.5	Same as round()
`(-) xxxxxx.0`	No change in value	Same as round()
`(-) xxxxxx.1`	Rounds to integer +0.5	Same as round() **+1**

B) Add ½ LSB and Truncate
2's Complement

- Example: positive values of `xxxxxx.xxxxxx` rounded to `yyyyyy`

Pre-rounded value	Rounding action	Net effect
`(+) xxxxxx.000000`	No change in value	Same as round()
`(+) xxxxxx.000001`	Rounds down to integer	Same as round()
`(+) xxxxxx.000010`	Rounds down to integer	Same as round()
. . .	. . .	. . .
`(+) xxxxxx.100000`	Rounds up to integer	Same as round()
. . .	. . .	. . .
`(+) xxxxxx.111101`	Rounds up to integer	Same as round()
`(+) xxxxxx.111110`	Rounds up to integer	Same as round()
`(+) xxxxxx.111111`	Rounds up to integer	Same as round()

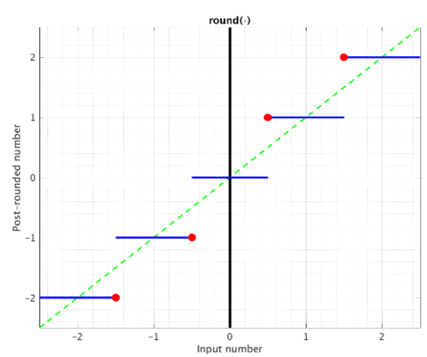
B) Add ½ LSB and Truncate
2's Complement

- Example: negative values of `xxxxxx.xxxxxx` rounded to `yyyyyy`

Pre-rounded value	Rounding action	Net effect
`(-) xxxxxx.000000`	No change in value	Same as round()
`(-) xxxxxx.000001`	Rounds down to integer	Same as round()
`(-) xxxxxx.000010`	Rounds down to integer	Same as round()
...	...	...
`(-) xxxxxx.100000`	Rounds up to integer	Same as round() **+1**
...	...	...
`(-) xxxxxx.111101`	Rounds up to integer	Same as round()
`(-) xxxxxx.111110`	Rounds up to integer	Same as round()
`(-) xxxxxx.111111`	Rounds up to integer	Same as round()

Hardware Rounding: C) Unbiased for 2's Complement

C. Unbiased Rounding: the same as matlab round(·)
   - For cases where a "DC" bias is unacceptable, positive and negative numbers must be rounded differently with 2's complement
   - Although logically simple, implementing an unbiased rounding with 2's complement numbers can increase the critical path delay significantly
   - The calculation is not so complex if the only operation is rounding, but this is uncommon. Things get interesting in the common case when rounding is the last step in a series of calculations.

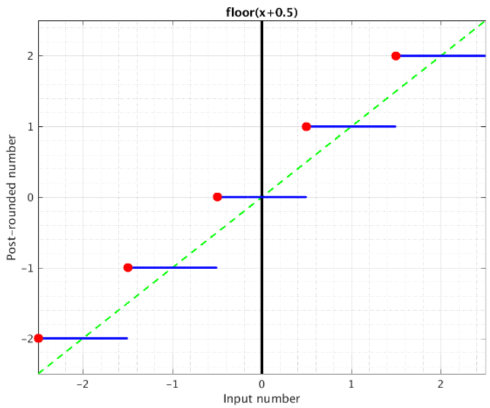


Hardware Rounding: C) Unbiased for 2's Complement

– Here is one basic algorithm (there are others)

1) *Remove* the normal ½ LSB rounding bit
2) Keep the output when the **result**(!) is:
    i. Negative and # Unrounded result is
    ii. Of the form `xxxxx.1000` # −xxx.5000 (base 10)
        - Equivalently, we could also not add the ½ LSB when the result is in the range:
          `xxxxx.0000` to `xxxxx.1000`
          Do you see why?
3) Otherwise, *add* the ½ LSB rounding bit back into the input and *recalculate the output*
4) Truncate as with method (B)

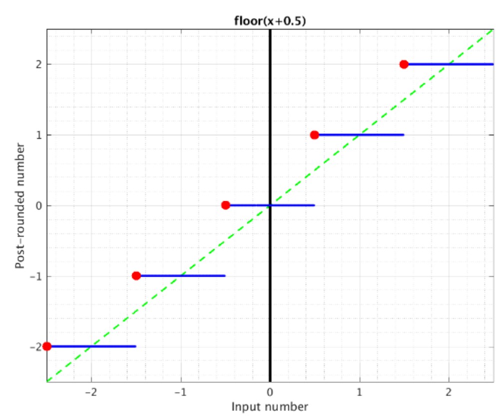
```
         0
+ xxx.xxxxx
-----------
  yyy.yyyxx
-----------
  yyy.yy---
```



Hardware Rounding: C) Unbiased for 2's Complement

– Here is a second basic algorithm
  1) *Add* the normal ½ LSB rounding bit
  2) Keep the output when the **result**(!) is *not*:
     i. (Negative and # A negative integer
     ii. of the form `xxxxx.0000`)
     iii. Or zero # If input was −0.5
  3) Otherwise, *remove* the ½ LSB rounding from the input and *recalculate the output*
  4) Truncate as with method (B)

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



Hardware Rounding: C) Unbiased for 2's Complement

- A third option is to calculate the result two times in parallel:
  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 using one of the previously-described algorithms or another
  - This is faster than the other two approaches however it requires about twice as much hardware which could be unacceptably expensive in area and energy dissipation

matlab for previous plots

- copy, paste, and try it out

```
% rounding.m
%
% Look at various rounding modes
%
% 2005/02/03  Written
% 2018/03/05  Updated colors and a number of details. Added PrintOn on line 40
%             to print plots in .tif files.
%
% The only reason to declare the main body as a function is because matlab
% chokes if a function is declared inside a non-function.
function rounding

%----- Definitions
x               = -2.5 : 0.01 : 2.5;     % "continuous" samples
xpos            =   0  : 0.01 : 2.5;     % "continuous" non-negative samples
xcirc5          = -2.5 : 1    : 2.5;     % circles at x.500 points
xcircint        = -2   : 1    : 2;       % circles at x.000 points
xcircpos        =  0.5 : 1    : 2.5;     % circles at x.500 points (positive)
axis_limits     = [-2.5 2.5 -2.5 2.5];   % for axis limits on plots
ticks           = [-2 -1 0 1 2];         % points on axis to draw lines

%----- Main
plot_one(1, x, round(x), xcirc5, round(xcirc5), ...
   axis_limits, ticks, 'round(\cdot)');
plot_one(2, x, fix(x), xcircint, fix(xcircint), ...
   axis_limits, ticks, 'fix(\cdot)');
plot_one(3, x, floor(x), xcircint, floor(xcircint), ...
   axis_limits, ticks, 'floor(\cdot)');
plot_one(4, x, ceil(x), xcircint, ceil(xcircint), ...
   axis_limits, ticks, 'ceil(\cdot)');
plot_one(5, x, floor(x+0.5), xcirc5, floor(xcirc5+0.5), ...
   axis_limits, ticks, 'floor(x+0.5)');
plot_one(6, xpos, floor(xpos+0.5), xcircpos, floor(xcircpos+0.5), ...
   axis_limits, ticks, 'floor(x+0.5), positive inputs only');
return;

%----- Function to plot one figure
function plot_one(fignum, x, y, xi, yi, axis_limits, ticks, nametitle)

PrintOn = 0;            % select whether to print plots to .tif and .eps files

figure(fignum); clf
% plot diagonal dashed line
plot([axis_limits(1) axis_limits(4)], [axis_limits(1) axis_limits(4)], ...
   '--g', 'linewidth', 1.5);
hold on;                % don't erase prior plots
% plot black vertical y-axis line
plot([0 0], [axis_limits(1) axis_limits(4)], '-k', 'linewidth', 3);
% print horizontal lines
plot(x, y, '.b');
% print large circles
plot(xi, yi, '.r', 'markersize', 30, 'linewidth', 2);

% touch up the plot
axis(axis_limits);
set(gca, 'XTick', ticks);
set(gca, 'YTick', ticks);
grid on;
grid minor;
xlabel('Input number');
ylabel('Post-rounded number');
title(nametitle);

% print out .eps and .tiff figures
filename = strcat('rounding', num2str(fignum));
if (PrintOn)
%print('-deps',  filename);
print('-dtiff', filename);
end

return;
```