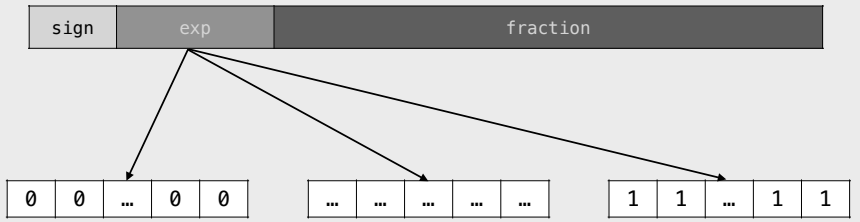
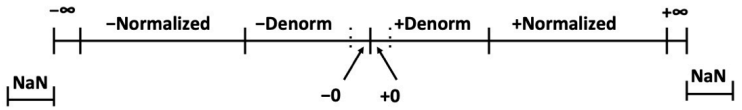
CSC 411

Computer Organization (Spring 2024)
Lecture 8: Floating Point (part 2)

Prof. Marco Alvarez, University of Rhode Island

Floating point encodings





Denormalized values

$$(-1)^s M 2^E$$

- **E = 1 - bias**
  - exp field is 00…00
- **M = 0.bb…bb**
  - bb…bb are the bits in the fraction field
  - M is the decimal that corresponds to 0.bb…bb
    - note that M has an implied leading 0
- Comments
  - note that if the fraction field is 00…00, we can represent +0 and -0
  - for all other cases, denormalized values are numbers close to 0.0

Practice (decoding)

$$(-1)^s M 2^E$$

- Assume a float F = 0x00580000
  - write the binary
    - divide into **s**, **exp**, **frac**
    - 0 00000000 10110000000000000000000
  - calculate M
    - M = 0.10110000000000000000000 = 0.6875
  - calculate E
    - E = 1 - bias = 1 - 127 = -126
  - write final number
    - $(-1)^0 * 0.6875 * 2^{-126} = 8.0815236619 * 10^{-39}$

Special values

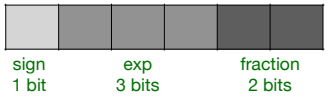
$$(-1)^s M 2^E$$

- Infinity
  - fraction field is 00…00
  - we can represent $+\infty$ and $-\infty$
- Not-a-number
  - fraction field != 00…00
  - no numeric value can be determined
  - e.g., $0/0, \sqrt{-1}, \pm\infty / \pm\infty$

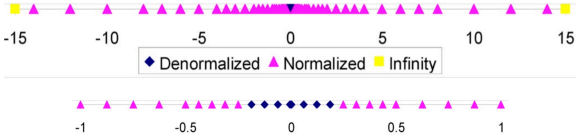
IEEE-like example formats

- Same general form can be extended to new formats
  - normalized, denormalized, and special values
- Using 8 bits
  - bias = 7

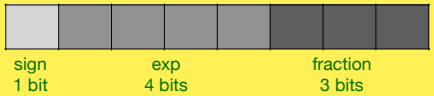
- Using 6 bits
  - bias = 3



  - distribution of values



Practice



- Using the 8-bit representation from the previous slide, provide the bit representation for the following **interesting numbers**:
  - zero
  - one
  - smallest positive denormalized
  - largest positive denormalized
  - smallest positive normalized
  - largest positive normalized

FP operations and rounding

- Operations using floating point numbers might produce results that can't be represented
  - e.g., x + y, x * y ⇐ will cover FP addition and multiplication later in the course
- Steps
  - perform the operation and compute the exact result
  - fit into the desired precision
    - may **overflow**
    - may need **rounding**
- Rounding (to nearest even)
  - default rounding mode
  - when exactly halfway between two values, round so that the least significant digit is **even**

Practice

- Rounding decimals
  - halfway when decimal digits to right of rounding position = 500…

4.32313333	4.32313333	4.32	
4.32500001	4.32500001	4.33	
4.31500000	4.31500000	4.32	to nearest even
4.34500000	4.34500000	4.34	to nearest even

- Rounding fractional binary numbers
  - halfway when binary digits to right of rounding position = 100…

10.00011	10.00011	10.00	
10.00110	10.00110	10.01	
10.11100	10.11100	11.00	to nearest even
10.10100	10.10100	10.10	to nearest even

Caution

- Rounding breaks associativity and other properties

```c
#include <stdio.h>

int main() {
    float a = 1e20;
    float b = -1e20;
    float c = 1;

    if (((a + b) + c) == (a + (b + c))) {
        printf("equal\n");
    } else {
        printf("different\n");
    }

    return 0;
}
```

- When comparing floating point values:
  - use `(abs(a-b) < eps)` with a small value in `eps`
  - avoid `(a == b)`

Floating point in C

- Single (`float`) and double (`double`) precision
- Casting and conversions
  - casting between two's complement signed/unsigned integer types **does not change** the bit representation
    - bit-extension, truncating may be applied
  - casting between integer and floating point types **does change** the bit representation

From	To	Action
double/float	integer	truncate fractional part
integer	float	can't guarantee exact conversion, possibly rounding
integer	double	exact conversion, as long as integer size < 53 bits

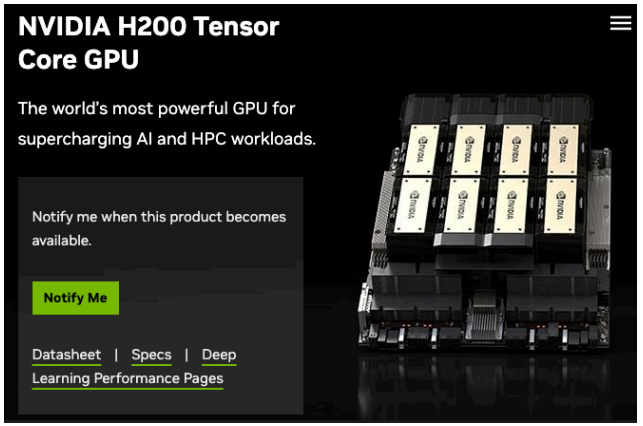
Practice

- Will the following statements always be **`true`**?
  - `i == (int) ((float) i)`
    - assume i is an int
  - `f == (float) ((int) f))`
    - assume f is a float

Floating point in C

- Consider an `int x`, a `float f`, and a `double d`
  - assuming d and f are not special values, what is the output of the following expressions?

Expression	Output
`x == (int) (float) x`	False
`x == (int) (double) x`	True
`f == (float) (double) f`	True
`d == (double) (float) d`	False
`f == -(-f)`	True
`2/3 == 2/3.0`	False
`(d < 0.0) then ((d*2) < 0.0)`	True
`(d > f) then (-f > -d)`	True
`d * d >= 0.0`	True
`(d + f) - d == f`	False

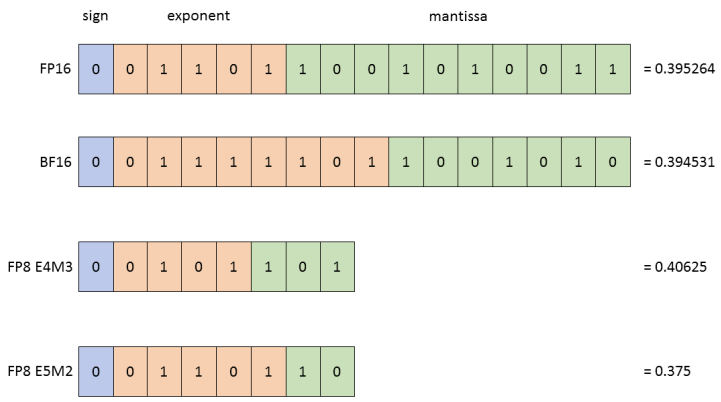


NVIDIA H200 Tensor Core GPU Quick Specs

GPU Memory	141GB
GPU Memory Bandwidth	4.8TB/s
FP8 Tensor Core Performance	4 PetaFLOPS

H100 GPU introduced support for a new datatype, FP8 (8-bit floating point), enabling higher throughput of matrix multiplies and convolutions.

NVIDIA H200 Tensor Core GPU



Structure of the floating point datatypes. All of the values shown are the closest representations of value 0.3952.