CSC 411

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

Prof. Marco Alvarez, University of Rhode Island

"95% of folks out there are completely clueless about floating-point."

James Gosling (founder and lead designer behind the Java programming language)

Denormalized and special values

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
- Key points:
  - if fraction field is 00…00, represents +0 or -0 (depending on sign bit)
  - 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

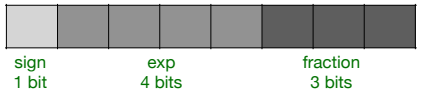
$(-1)^s M 2^E$

- Infinity
  - fraction field is 00…00
  - sign bit determines $+\infty$ or $-\infty$
- Not-a-number (NaN)
  - fraction field != 00…00
  - represents undefined or unrepresentable values (e.g., $0/0, \sqrt{-1}$)

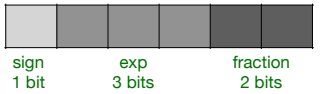
Final considerations

IEEE-like example formats

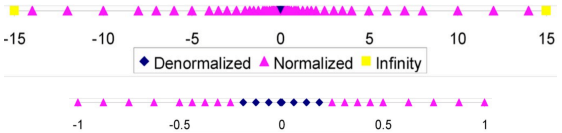
- Same general form can be extended to new formats
  - smaller formats trade precision and range for memory efficiency
- Using 8 bits
  - bias = 7



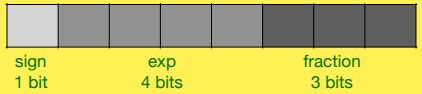
- Using 6 bits
  - bias = 3



  - distribution of values



Practice



- Using the 8-bit representation from the previous slide, provide the bit representation for the following 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 exactly 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 involve **overflow** or **rounding**)
- Round to nearest, ties to even (default)
  - 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

FP arithmetic cautions

- Rounding can break associativity and other mathematical 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 direct equality comparisons** `(a == b)`

Floating point in C

- Single precision (`float`) and double precision (`double`)
- 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
`if (d < 0.0) then ((d*2) < 0.0)`	True
`if (d > f) then (-f > -d)`	True
`d * d >= 0.0`	True
`(d + f) - d == f`	False

Practice

- Assume an 8-bit floating point representation
  - 1-4-3 bits for sign, exp, and fraction respectively
  - encode 0.3125 (0.0101)
    - E = exp - bias => exp = E + bias (bias = 7)
    - 1.0100000000 x 2^(-2)
    - exp = -2 + 7 = 5
    - 0 0101 010 = 0x2A
  - encode 7.3125 (111.0101)
    - 1.110101 x 2^2
    - exp = 2 + 7 = 9
    - 0 1001 111

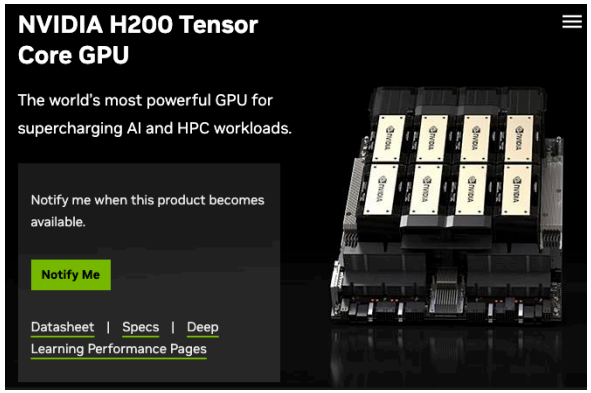
Precision

Format	Smallest positive value (*)	Largest positive value (*)	Precision (**)
single	$\sim 1.401 \cdot 10^{-45}$	$\sim 3.403 \cdot 10^{+38}$	6-9 digits
double	$\sim 4.941 \cdot 10^{-324}$	$\sim 1.798 \cdot 10^{+308}$	15-17 digits

(*) Smallest/largest negative values are the same as their positive counterparts, but negative.

(**) Precision refers to to the number of significant digits that can be represented in a number.

In C, FLT_MIN and DBL_MIN are defined as the smallest normalized numbers, and FLT_TRUE_MIN and DBL_TRUE_MIN represent the smallest positive value that can be represented by the float type, including denormalized numbers.

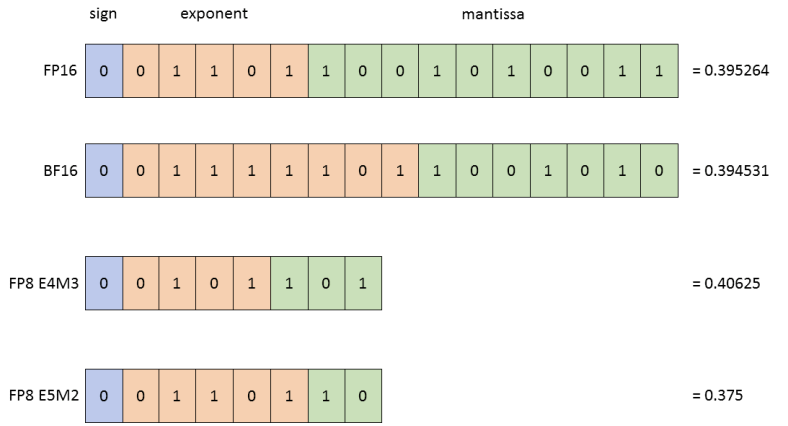


NVIDIA H200 Tensor Core GPU Quick Specs

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.