

CSCI 2021: Binary Floating Point Numbers

Chris Kauffman

*Last Updated:
Mon Feb 13 11:46:53 AM CST 2023*

Logistics

Reading Bryant/O'Hallaron

- ▶ Ch 2.4-5: Floats, Wed/Fri
- ▶ 2021 Quick Guide to GDB
- ▶ Next week: Ch 3.1-7:
Assembly Intro

Goals this Week

- ▶ Discuss Bitwise ops from
Integer Rep Slides
- ▶ Floating Point layout
- ▶ gdb introduction

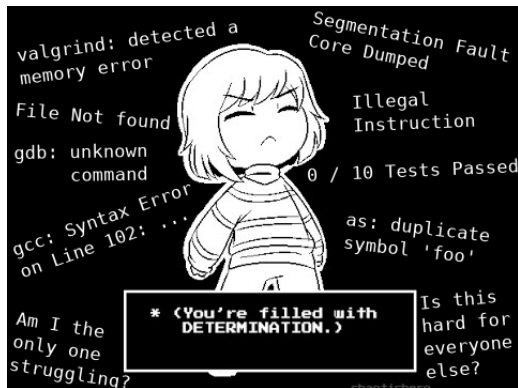
Assignments

- ▶ Canvas Feedback Survey
 - ▶ Anonymous: be honest!
 - ▶ Worthe 1 EP
 - ▶ Due Wed 15-Feb
 - ▶ 67% response rate so far
- ▶ HW04: Due Wed 11:59pm
- ▶ Lab05: Bit operations
- ▶ HW05: Bits, Floats, GDB

P2 To Be Released Tue

Bit shifting and Debugger Usage

Don't Give Up, Stay Determined!



- ▶ If Project 1 / Exam 1 went awesome, count yourself lucky
- ▶ If things did not go well, **Don't Give Up**
- ▶ Spend some time contemplating **why** things didn't go well, talk to course staff about it, learn from any mistakes
- ▶ There is a LOT of semester left and plenty of time to recover from a bad start

Parts of a Fractional Number

The meaning of the “decimal point” is as follows:

$$\begin{aligned} 123.406_{10} &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + & 123 &= 100 + 20 + 3 \\ &4 \times 10^{-1} + 0 \times 10^{-2} + 6 \times 10^{-3} & 0.406 &= \frac{4}{10} + \frac{6}{1000} \\ &= 123.406_{10} \end{aligned}$$

Changing to base 2 induces a “binary point” with similar meaning:

$$\begin{aligned} 110.101_2 &= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + & 6 &= 4 + 2 \\ &1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} & 0.625 &= \frac{1}{2} + \frac{1}{8} \\ &= 6.625_{10} \end{aligned}$$

One *could* represent fractional numbers with a **fixed point** e.g.

- ▶ 32 bit fractional number with
- ▶ 10 bits left of Binary Point (integer part)
- ▶ 22 bits right of Binary Point (fractional part)

BUT most applications require a more flexible scheme

Scientific Notation for Numbers

“Scientific” or “Engineering” notation for numbers with a fractional part is

| Standard | Scientific | <code>printf("%.4e",x);</code> |
|----------|-------------------------|--------------------------------|
| 123.456 | 1.23456×10^2 | 1.2346e+02 |
| 50.01 | 5.001×10^1 | 5.0010e+01 |
| 3.14159 | 3.14159×10^0 | 3.1416e+00 |
| 0.54321 | 5.4321×10^{-1} | 5.4321e-01 |
| 0.00789 | 7.89×10^{-3} | 7.8900e-03 |

- ▶ **Always** includes one **non-zero** digit left of decimal place
- ▶ Has some **significant** digits after the decimal place
- ▶ Multiplies by a **power of 10** to get actual number

Binary Floating Point Layout Uses Scientific Convention

- ▶ Some bits for integer/fractional part
- ▶ Some bits for exponent part
- ▶ All in base 2: 1's and 0's, powers of 2

Conversion Example

Below steps convert a decimal number to a fractional binary number equivalent then adjusts to scientific representation.

```
float f1 = -248.75;
```

$$\begin{array}{rcll} & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 \\ -248.75 & = & -(128+64+32+16+8+0+0+0) & . & (1/2+1/4) \\ & = & -11111000.11 & *2^0 \\ & & 76543210 & 12 \\ & = & -1111100.011 & *2^1 \\ & & 6543210 & 123 \\ & = & -111110.0011 & *2^2 \\ & & 543210 & 1234 \\ & & \dots & \\ & & \text{MANTISSA} & \text{EXPONENT} \\ & = & -1.111100011 & * 2^7 \\ & & 0 & 123456789 \end{array}$$

Mantissa \equiv Significand \equiv Fractional Part

Principle and Practice of Binary Floating Point Numbers

- ▶ In early computing, computer manufacturers used similar principles for floating point numbers but varied specifics
- ▶ Example of Early float data/hardware
 - ▶ Univac: 36 bits, 1-bit sign, 8-bit exponent, 27-bit significand¹
 - ▶ IBM: 32 bits, 1-bit sign, 7-bit exponent, 24-bit significand²
- ▶ Manufacturers implemented circuits with different rounding behavior, with/without infinity, and other inconsistencies
- ▶ Troublesome for reliability: code produced different results on different machines
- ▶ This was resolved with the adoption of the **IEEE 754 Floating Point Standard** which specifies
 - ▶ Bit layout of 32-bit float and 64-bit double
 - ▶ Rounding behavior, special values like Infinity
- ▶ **Turing Award** to **William Kahan** for his work on the standard

¹Floating Point Arithmetic

²IBM Hexadecimal Floats

IEEE 754 Format: *The Standard for Floating Point*

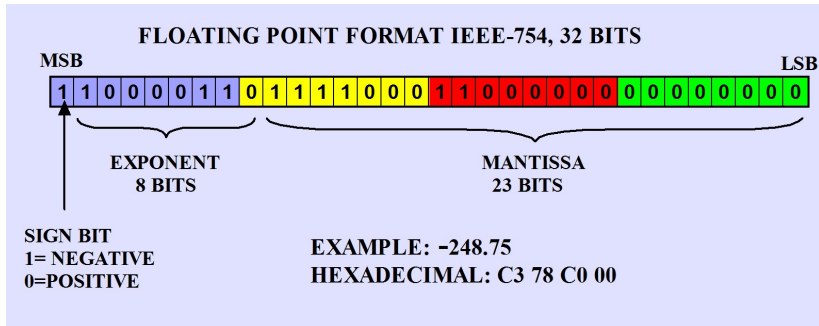
| float | double | Property |
|-------|--------|---|
| 32 | 64 | Total bits |
| 1 | 1 | Bits for sign (1 neg / 0 pos) |
| 8 | 11 | Bits for Exponent multiplier (power of 2) |
| 23 | 52 | Bits for Fractional part or mantissa |
| 7.22 | 15.95 | Decimal digits of accuracy ³ |

- ▶ Most commonly implemented format for floating point numbers in hardware to do arithmetic: processor has physical circuits to add/mult/etc. for this bit layout of floats
- ▶ Numbers/Bit Patterns divided into three categories

| Category | Description | Exponent |
|--------------|---------------------------------------|-----------|
| Normalized | most common like 1.0 and -9.56e37 | mixed 0/1 |
| Denormalized | very close to zero and 0.0 | all 0's |
| Special | extreme/error values like Inf and NaN | all 1's |

³[Wikipedia: IEEE 754](#)

Example float Layout of -248.75: float_examples.c



Source: IEEE-754 Tutorial, www.puntoflotante.net

Color: 8-bit blocks, **Negative**: highest bit, leading 1

Exponent: high 8 bits, 2^7 encoded with bias of -127

$$\begin{aligned} &1000_0110 - 0111_1111 \\ &= 128+4+2 - 127 \\ &= 134 - 127 \\ &= 7 \end{aligned}$$

Fractional/Mantissa portion is

1.111100011...

^ |||||

| explicit low 23 bits

|

implied leading 1

not in binary layout

Normalized Floating Point: General Case

- ▶ A “normalized” floating point number is in the standard range for float/double, bit layout follows previous slide
- ▶ Example: $-248.75 = -1.111100011 * 2^7$

Exponent is in **Bias Form** (not Two's Complement)

- ▶ Unsigned positive integer minus constant **bias number**
- ▶ **Consequence:** exponent of 0 is not bitstring of 0's
- ▶ **Consequence:** tiny exponents like -125 close to bitstring of 0's; this makes resulting number close to 0
- ▶ 8-bit exponent 1000 0110 = $128+4+2 = 134$
so exponent value is $134 - 127 = 7$

Integer and Mantissa Parts

- ▶ The leading 1 before the binary point is **implied** so does not show up in the bit string
- ▶ Remaining fractional/mantissa portion shows up in the low-order bits

Fixed Bit Standards for Floating Point

IEEE Standard Layouts

| Kind | Sign Bit | Exponent Bits | Bias | Exp Range | Mantissa Bits |
|--------|----------|-----------------|-------|----------------|----------------|
| float | 31 (1) | 30-23 (8 bits) | -127 | -126 to +127 | 22-0 (23 bits) |
| double | 63 (1) | 62-52 (11 bits) | -1023 | -1022 to +1023 | 51-0 (52 bits) |

Standard allows hardware to be created that is as efficient as possible to do calculation on these numbers

Consequences of Fixed Bits

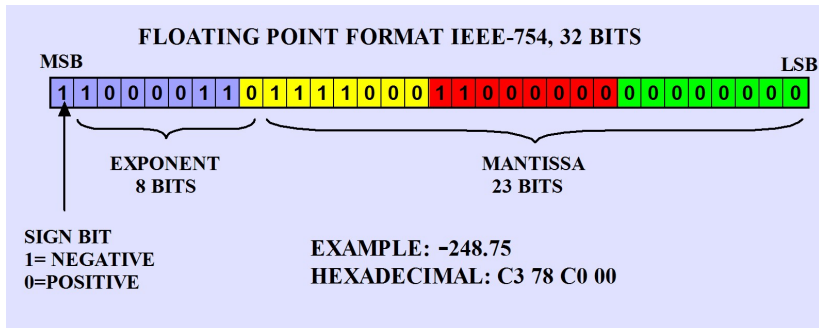
- ▶ Since a fixed # of bit is used, **some numbers cannot be exactly represented**, happens in any numbering system:
- ▶ Base 10 and Base 2 cannot represent $\frac{1}{3}$ in finite digits
- ▶ Base 2 cannot represent $\frac{1}{10}$ in finite digits

```
float f = 0.1;
printf("0.1 = %.20e\n",f);
0.1 = 1.00000001490116119385e-01
```

Try `show_float.c` to see this in action

Exercise: Quick Checks

1. What distinct parts are represented by bits in a floating point number (according to IEEE)
2. What is the “bias” of the exponent for 32-bit floats
3. Represent 7.125 in binary using “binary point” notation
4. Lay out 7.125 in IEEE-754 format
5. What does the number 1.0 look like as a float?



Source: IEEE-754 Tutorial, www.puntoflotante.net

The diagram above may help in recalling IEEE 754 layout

Special Cases: See `float_examples.c`

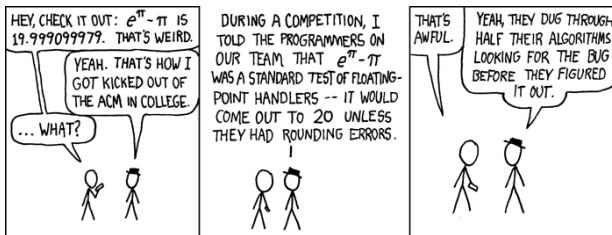
Special Values

- ▶ **Infinity**: exponent bits all 1, fraction all 0, sign bit indicates $+\infty$ or $-\infty$
- ▶ Infinity results from overflow/underflow or certain ops like `float x = 1.0 / 0.0;`
- ▶ `#include <math.h>` gets macro `INFINITY` and `-INFINITY`
- ▶ **NaN**: not a number, exponent bits all 1, fraction has some 1s
- ▶ Errors in floating point like `0.0 / 0.0`

Denormalized values: Exponent bits all 0

- ▶ Fractional/Mantissa portion evaluates *without* implied leading one, still an unsigned integer though
- ▶ Exponent is $Bias + 1$: 2^{-126} for `float`
- ▶ Result: very small numbers close to zero, smaller than any other representation, degrade uniformly to 0
- ▶ Zero: bit string of all 0s, optional leading 1 (*negative zero*);

Other Float Notes



Source: XKCD #217

Approximations and Roundings

- ▶ Approximate $\frac{2}{3}$ with 4 digits, usually 0.6667 with standard rounding in base 10
- ▶ Similarly, some numbers cannot be exactly represented with fixed number of bits: $\frac{1}{10}$ approximated
- ▶ IEEE 754 specifies various rounding modes to approximate numbers

Clever Engineering

- ▶ IEEE 754 allows floating point numbers to sort using signed integer sorting routines
- ▶ Bit patterns for float follows are ordered nearly the same as bit patterns for signed int
- ▶ Integer comparisons are usually fewer clock cycles than floating comparisons

Sidebar: The Weird and Wonderful Union

- ▶ Bitwise operations like `&` are not valid for `float/double`
- ▶ Can use pointers/casting to get around this OR...
- ▶ Use a **union**: somewhat unique construct to C
- ▶ Defined like a struct with several fields
- ▶ BUT fields occupy the same memory location (!!!)
- ▶ Allows one to treat a byte position as multiple different types, ex: `int` / `float` / `char` []
- ▶ Memory size of the union is the **max** of its fields

```
// union.c
typedef union { // shared memory
    float fl;    // an float
    int in;      // a int
    char ch[4];  // char array
} flint_t;      // 4 bytes total

int main(){
    flint_t flint;
    flint.in = 0xC378C000;
    printf("%.4f\n", flint.fl);
    printf("%08x %d\n", flint.in, flint.in);
    for(int i=0; i<4; i++){
        unsigned char c = flint.ch[i];
        printf("%d: %02x '%c'\n", i, c, c);
    }
}
```

| Symbol | Mem | Val |
|-------------------|-------|------|
| flint.ch[3] | #1027 | 0xC3 |
| flint.ch[2] | #1026 | 0x78 |
| flint.ch[1] | #1025 | 0xC0 |
| flint.in/fl/ch[0] | #1024 | 0x00 |
| i | #1020 | ? |

Floating Point Operation Efficiencies

- ▶ Floating Point Operations per Second, **FLOPS** is a major measure for numerical code/hardware efficiency
- ▶ Often used to benchmark and evaluate scientific computer resources, (e.g. [top super computers in the world](#))
- ▶ Tricky to evaluate because of
 - ▶ A single FLOP (add/sub/mul/div) may take 3 clock cycles to finish: **latency 3**
 - ▶ Another FLOP **can start** before the first one finishes:
pipelined
 - ▶ Enough FLOPs lined up can get **average 1 FLOP per cycle**
 - ▶ FP Instructions may automatically operate on multiple FPs stored in memory to feed pipeline: **vectorized ops**
 - ▶ Generally referred to as **superscalar**
 - ▶ Processors schedule things **out of order** too
- ▶ All of this makes micro-evaluation error-prone and pointless
- ▶ Run a real application like an N-body simulation and compute

$$\text{FLOPS} = \frac{\text{number of floating ops done}}{\text{time taken in seconds}}$$

Top 5 Super Computers Worldwide, June 2022

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power* (kW) |
|------|--|-----------|-------------------|--------------------|----------------|
| 1 | Frontier, <i>USA / Oak Ridge</i> Cray EX235a, AMD EPYC 2GHz (x86-64) | 8,730,112 | 1,102.00 | 1,685.65 | 21,100 |
| 2 | Fugaku, <i>Japan / Fujitsu</i> Fujitsu A64FX 2.2GHz (Arm) | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 3 | LUMI <i>Finland / EuroHPC</i> Cray EX235a, AMD EPYC 2GHz (x86-64) | 1,110,144 | 151.90 | 214.35 | 2,942 |
| 4 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz (Power) | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 5 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz (Power) | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |

<https://www.top500.org/lists/top500/2022/06/>

*: An average US Home uses 909 kWh of power per month

Top 5 Super Computers Worldwide, June 2021

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|--|------------|-------------------|--------------------|---------------|
| 1 | Fugaku, <i>Japan</i> / <i>Fujitsu</i> Fujitsu A64FX 2.2GhZ (Arm) | 7,630,848 | 442,010.0 | 537,212.0 | 29,899 |
| 2 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz (Power) | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 3 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz (Power) | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight <i>China</i> Sunway SW26010 (custom RISC) | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Perlmutter, <i>United States</i> AMD EPYC 2.45GHz, Cray (x86-64) | 706,304 | 64,590.0 | 89,794.5 | 2,528 |

<https://www.top500.org/lists/top500/2021/06/>

Top 5 Super Computers Worldwide, Nov 2020

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|--|------------|-------------------|--------------------|---------------|
| 1 | Fugaku, <i>Japan</i> / <i>Fujitsu</i> Fujitsu A64FX 2.2GHz (Arm) | 7,299,072 | 415,530.0 | 513,854.7 | 28,335 |
| 2 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz (Power) | 2,397,824 | 143,500.0 | 200,794.9 | 10,096 |
| 3 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz (Power) | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight <i>China</i> Sunway SW26010 (custom RISC) | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Selene <i>USA</i> , <i>NVIDIA/AMD</i> AMD EPYC 7742 64C 2.25GHz (x86-64) | 555,520 | 63,460.0 | 79,215.0 | 2,646 |

<https://www.top500.org/lists/top500/2020/06/>

Top 5 Super Computers Worldwide, June 2020

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|--|------------|-------------------|--------------------|---------------|
| 1 | Fugaku, <i>Japan</i> / <i>Fujitsu</i> Fujitsu A64FX 2.2GhZ (Arm) | 7,299,072 | 415,530.0 | 513,854.7 | 28,335 |
| 2 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz (Power) | 2,397,824 | 143,500.0 | 200,794.9 | 10,096 |
| 3 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz (Power) | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 4 | Sunway TaihuLight <i>China</i> Sunway SW26010 (custom RISC) | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 5 | Tianhe-2A <i>China</i> Intel Xeon 2.2GHz (x86-64) | 4,981,760 | 61,444.5 | 100,678.7 | 18,482 |

<https://www.top500.org/lists/top500/2020/06/>

Top 5 Super Computers Worldwide, Nov 2019

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|---|------------|-------------------|--------------------|---------------|
| 1 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz | 2,397,824 | 143,500.0 | 200,794.9 | 9,783 |
| 2 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz, | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 3 | Sunway TaihuLight <i>China</i> Sunway MPP | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 4 | Tianhe-2A <i>China</i> Xeon 2.2GHz | 4,981,760 | 61,444.5 | 100,678.7 | 18,482 |
| 5 | Frontera, <i>United States</i> Dell 6420, Xeons 2.7GHz | 448,448 | 23,516.4 | 38,745.9 | ?? |

<https://www.top500.org/list/2019/11/>

Top 5 Super Computers Worldwide, Nov 2018

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|---|------------|-------------------|--------------------|---------------|
| 1 | Summit <i>United States</i> IBM POWER9 22C 3.07GHz | 2,397,824 | 143,500.0 | 200,794.9 | 9,783 |
| 2 | Sierra <i>United States</i> IBM POWER9 22C 3.1GHz, | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 3 | Sunway TaihuLight <i>China</i> Sunway MPP | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 4 | Tianhe-2A <i>China</i> TH-IVB-FEP Cluster | 4,981,760 | 61,444.5 | 100,678.7 | 18,482 |
| 5 | Piz Daint <i>Switzerland</i> Cray XC50, Xeon E5-2690v3 | 387,872 | 21,230.0 | 27,154.3 | 2,384 |

<https://www.top500.org/list/2018/11/>

Top 5 Super Computers Worldwide, Nov 2017

| Rank | System | #Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|------|--|------------|-------------------|--------------------|---------------|
| 1 | Sunway TaihuLight <i>China</i> Sunway MPP | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 2 | Tianhe-2 (MilkyWay-2) <i>China</i> TH-IVB-FEP Cluster | 3,120,000 | 33,862.7 | 54,902.4 | 17,808 |
| 3 | Piz Daint <i>Switzerland</i> Cray XC50 | 361,760 | 19,590.0 | 25,326.3 | 2,272 |
| 4 | Gyokou <i>Japan</i> ZettaScaler-2.2 HPC system | 19,860,000 | 19,135.8 | 28,192.0 | 1,350 |
| 5 | Titan <i>USA</i> Cray XK7 | 560,640 | 17,590.0 | 27,112.5 | 8,209 |

<https://www.top500.org/lists/2017/11/>