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 (Integer Slides)
- gdb introduction
- Floating Point layout

Feedback Survey
- Open on Canvas
- Anonymous: be honest!
- Due Wed 2/17 for 1 EP

Labs/HW
- Lab05: Bit operations
- HW05: Bits, Floats, GDB

Project 2
Release Thursday Friday
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:

\[ 123.406_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 0 \times 10^{-2} + 6 \times 10^{-3} \]

\[ 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \]

\[ 123 = 100 + 20 + 3 \]

\[ 0.406 = \frac{4}{10} + \frac{6}{1000} \]

\[ = 123.406_{10} \]

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

\[ 110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \]

\[ 6 = 4 + 2 \]

\[ 0.625 = \frac{1}{2} + \frac{1}{8} \]

\[ = 6.625_{10} \]

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Scientific</th>
<th><code>printf(&quot;%.4e&quot;,x);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>123.456</td>
<td>1.23456 × 10²</td>
<td>1.2346e+02</td>
</tr>
<tr>
<td>50.01</td>
<td>5.001 × 10¹</td>
<td>5.0010e+01</td>
</tr>
<tr>
<td>3.14159</td>
<td>3.14159 × 10⁰</td>
<td>3.1416e+00</td>
</tr>
<tr>
<td>0.54321</td>
<td>5.4321 × 10⁻¹</td>
<td>5.4321e-01</td>
</tr>
<tr>
<td>0.00789</td>
<td>7.89 × 10⁻³</td>
<td>7.8900e-03</td>
</tr>
</tbody>
</table>

- **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.

```plaintext
float fl = -248.75;
```

```
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
...
```

```
MANTISSA   EXPONENT
= -1.111100011 * 2^7
  0 123456789
```

Mantissa ≡ Significand ≡ Fractional Part
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 significand
- IBM: 32 bits, 1-bit sign, 7-bit exponent, 24-bit significand

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

---

1. Floating Point Arithmetic
2. IBM Hexadecimal Floats
IEEE 754 Format: The Standard for Floating Point

<table>
<thead>
<tr>
<th>Property</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total bits</td>
<td>72</td>
<td>15.95</td>
</tr>
<tr>
<td>Sign bits</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Exponent bits</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Fractional part</td>
<td>23</td>
<td>52</td>
</tr>
<tr>
<td>Decimal digits of accuracy</td>
<td>7.22</td>
<td>15.95</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized</td>
<td>most common like 1.0 and −9.56e37</td>
<td>mixed 0/1</td>
</tr>
<tr>
<td>Denormalized</td>
<td>very close to zero and 0.0</td>
<td>all 0’s</td>
</tr>
<tr>
<td>Special</td>
<td>extreme/error values like Inf and NaN</td>
<td>all 1’s</td>
</tr>
</tbody>
</table>

3 Wikipedia: IEEE 754
Example float Layout of -248.75: float_examples.c

**FLOATING POINT FORMAT IEEE-754, 32 BITS**

- **MSB**
- **LSB**
- **EXPONENT** 8 BITS
- **MANTISSA** 23 BITS

**SIGN BIT**
- 1 = NEGATIVE
- 0 = POSITIVE

**EXAMPLE:** -248.75

**HEXADECIMAL:** C3 78 C0 00

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

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

\[
1000_0110 - 0111_1111 = 128+4+2 - 127 = 134 - 127 = 7
\]

Fractional/Mantissa portion is

\[
1.111100011... \quad ^{|||\quad |||\quad |||\quad |||\quad |||}
\]

| 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: \(-248.75 = -1.111100011 \times 2^{7}\)

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

<table>
<thead>
<tr>
<th>Kind</th>
<th>Sign Bit</th>
<th>Exponent Bits</th>
<th>Bias</th>
<th>Exp Range</th>
<th>Mantissa Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>31 (1)</td>
<td>30-23 (8 bits)</td>
<td>-127</td>
<td>-126 to +127</td>
<td>22-0 (23 bits)</td>
</tr>
<tr>
<td>double</td>
<td>63 (1)</td>
<td>62-52 (11 bits)</td>
<td>-1023</td>
<td>-1022 to +1023</td>
<td>51-0 (52 bits)</td>
</tr>
</tbody>
</table>

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

```c
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?

The diagram above may help in recalling IEEE 754 layout

Special Cases: See float_examples.c

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*)

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 $0.0 / 0.0$
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 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 (!?!) (Why?)
- 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

```c
// 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, Nov 2017

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

[https://www.top500.org/lists/2017/11/]
# Top 5 Supercomputers Worldwide, Nov 2018

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

https://www.top500.org/list/2018/11/
# Top 5 Super Computers Worldwide, Nov 2019

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

[https://www.top500.org/list/2019/11/](https://www.top500.org/list/2019/11/)
### Top 5 Super Computers Worldwide, June 2020

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

[https://www.top500.org/lists/top500/2020/06/]
## Top 5 Super Computers Worldwide, November 2020

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

[https://www.top500.org/lists/top500/2020/06/](https://www.top500.org/lists/top500/2020/06/)