CS 33
Introduction to Computer Organization

Qi Zhao
Week 5
11/2/2018
Reminder

• Second half starts (From 5\textsuperscript{th} week till the end)

• Office hours:
  • Starting from 6\textsuperscript{th} week
  • Location: Eng VI, 3\textsuperscript{rd} floor, common study area
  • 1C: Monday 11am – 1pm
  • 1D: Thursday 9:30am – 11:30am
Agenda

• *ALL* Floating Point
So far

• We have one way of interpreting binary. Bit ‘n’ contributes 0 or $2^n$ to the total value of the number. This means a 32-bit number can represent $2^{32}$ values.

• As you can probably guess, this alone is not sufficient.

• For example, without some modification, this won't allow for the representation of non-integer numbers.
So far

- With this in mind, let's try extending our existing binary representation by adding a decimal point. Consider a 5-bit representation.
- Instead of 00000 to 11111, we'll have 000.00 to 111.11
- The bit immediately to the left of the point contributes 0 or $2^0$ while the bit immediately to the right contributes 0 or $2^{-1}$. The next bit to the right contributes 0 or $2^{-2}$
- This is known as “fixed point” binary.
Fixed Point Binary

• Consider an 8-bit fixed point binary value where the point is between bits 3 and 4 (0000.0000 to 1111.1111)

• 0101.0110

• $= 2^2 + 2^0 + 2^{-2} + 2^{-3} = 5.175$

• Because we have 4 fractional bits, we can express a granularity of $2^{-4}$ or .0625.

• What's the cost of doing things this way?
Fixed Point Binary

• Originally, an 8-bit signed number represents values from -128 to 127
• Now our 8-bit signed number represents the following values:
  • 1000.0000 to 0111.1111 or:
    • -8 to 7.9375
• With this 8 bit signed representation, in order to represent up to fractional digits, we've reduced our overall range from 256 to approximately 16.
Fixed Point Binary

- Well, that was a dummy example anyway. What if we tried this in a real system that uses 32 bit values?
- For each bit we use for a fractional bit, we reduce the approximate range of values by 2.
- This means that if we use 5 bits to achieve a granularity of $2^{-5} (0.03125)$, the range of values drops to about $2^{27}$ (approximately 134 million)
Fixed Point Binary

• We paid a huge cost to in order to represent only 5 fractional digits.
• ...and we still can't even represent .1 precisely!
• .00011 = .09375
• .00100 = .125
• The fact is, regardless of what we do, there will be “simple” decimal fractional numbers that we can't represent in binary.
Fixed Point Binary

• With this finite, discrete binary representation, we will always have limited precision and limited range. However, perhaps there is a representation that loses precision in an acceptable way.

• Consider an unsigned 32-bit fixed point where 8 bits are used for the fractional bits. This means we can represent an unsigned number (nearly) as large as $2^{24}$ with a granularity of $2^{-8}$. 
Fixed Point Binary

• However, if I'm representing a number in the range of $2^{23}$, how interested are we in a value of $2^{-8}$?
• Given $8.398 \times 10^6$, what is $8.398 \times 10^6 + 1$?
• It's just going to be $8.398 \times 10^6$ simply because 1 is entirely overshadowed by the other number.
• That's the beauty of scientific notation right?
Fixed Point Binary

• The goal for floating point is essentially to be able to represent base 2 scientific notation

• Generic:
  • $F \times B^E$

• Base 10:
  • $4.87 \times 10^2$

• Base 2:
  • $1.101_2 \times 2^4$
Fixed Point Binary

• Each binary configuration is split into three components:
  • s: Sign bit
  • e: Exponential part
  • f: Fraction part

• \([s][\text{exponent}][\text{fractional}]\)

• 0 1101 100
Fixed Point Binary

- Single precision: 8 exp bits, 23 frac bits, for 32 bits total
- Double precision: 11 exp bits, 52 frac bits, for 64 bits total
- Extended precision: 15 exp bits, 63 frac bits (only Intel-compatible machines)
Normalized Numeric Values

- $V = (-1)^s \times 2^E \times M$

- **Condition**: exp $\neq 000 \ldots 0$ and exp $\neq 111 \ldots 1$

- **Exponent is coded as a biased value**
  - $E = \text{Exp} - \text{Bias}$
    - Exp: unsigned value denoted by $\text{exp}$.
    - Bias: Bias value
      - In general: $\text{Bias} = 2^{e-1} - 1$, where $e$ is the number of exponent bits
      - Single precision: $2^{8-1}-1 = 127$ (Exp: $1 \ldots 254$, E: $-126 \ldots 127$)
      - Double precision: $2^{11-1}-1 = 1023$ (Exp: $1 \ldots 2046$, E: $-1022 \ldots 1023$)

- **Significand coded with implied leading 1**
  - $M = 1.\text{xxx} \ldots x_2$
    - xxx $\ldots$ x: bits of frac
    - Minimum when $000 \ldots 0$ (M = 1.0)
    - Maximum when $111 \ldots 1$ (M = 2.0 $-$ e)
    - We get the extra leading bit “for free.”
Normalized Numeric Values

• **Value**
  - float F = 15213.0;
  - $15213_{10} = 11101101101101_{2} = 1.1101101101101_{2} \times 2^{13}$

• **Significand**
  - $M = 1.1101101101101_{2}$
  - frac = 1101101101101000000000000

• **Exponent**
  - $E = 13$
  - Bias = 127
  - Exp = $140 = 10001100$
Normalized Numeric Values

- **Floating Point Representation**
- Hex: 466DB400
- Binary: 0100 0110 0110 1101 1011 0100 0000 0000
- 140: 100 0110 0
- 15213: 110 1101 1011 01
Normalized Numeric Values

• \([s][\text{exponent}][\text{fractional}]\)
• 0 1101 100
• \(V = (-1)^s \times 2^E \times M_2\)
Normalized Numeric Values

- \([s][exponent][fractional]\]
- \[0 \ 1101 \ 100\]
- \[V = (-1)^s * 2^E * M_2\]
- \[V = (-1)^0 * 2^{13-7} * 1.100_2\]
- \[= 26 * 1.5 = 96\]
Normalized Example

- Given the bit string \texttt{0x40500000}, what floating point number does it represent?
Normalized Example

• Given the bit string **0x40500000**, what floating point number does it represent?

• Writing this as a bit string gives us:

  0 10000000 101000000000000000000000000000000

• We see that this is a positive, normalized number. 

  \[ \text{exp} = 128 - 127 = 1 \]

• So, this number is:

  \[ 1.101_2 \times 2^1 = 11.01_2 = 3.25_{10} \]
Normalized Numeric Values

• This is “normalized”, because the fractional field always begins with a 1
  • If \( f = 1010 \), the fractional field is 1.1010
  • If \( f = 0000 \), the fractional field is 1.0000

• The reason for this is so that there is only one way to represent a single value. We don't want:
  • \( V = 24 \times .11 \) where \( e = 11 \) and \( f = 11 \)
  • \( V = 25 \times .011 \) where \( e = 12 \) and \( f = 011 \)
Normalized Numeric Values

• However, what's the minimum value we can represent like this?

• The minimum value is limited by the smallest exponent that e can be. The f will always contribute a value that is \([1,2)\).

• Maybe we want to represent smaller numbers somehow?
Denormalized Numeric Values

- \( V = (-1)^s \times 2^{1-Bias} \times .f_2 \)
- **Condition**: \( \text{exp} = 000...0 \)
- **Value**
- Exponent values: \( E = -\text{Bias} + 1 \) **Why this value?**
  - Floats: \(-126\); Doubles: \(-1022\)
  - Significand value: \( M = 0.xxx \ldots x_2 \), where \( xxx \ldots x \) are the bits of \( \text{frac} \).
- **Cases**
  - \( \text{exp} = 000 \ldots 0 \) and \( \text{frac} = 000 \ldots 0 \)
    - represents values of 0
    - notice that we have distinct +0 and -0
  - \( \text{exp} = 000 \ldots 0 \) and \( \text{frac} \neq 000 \ldots 0 \)
    - These are numbers very close to 0.0
    - Lose precision as they get smaller
    - Experience “gradual underflow”
Denormalized Numeric Values

- $[s][\text{exponent}][\text{fractional}]$
- $0 \ 0000 \ 100$
- $V = (-1)^s \times 2^{1-\text{Bias}} \times .f_2$
Denormalized Numeric Values

• $[s][\text{exponent}][\text{fractional}]$
• $0 \ 0000 \ 100$
• $V = (-1)^s \times 2^{1-\text{Bias}} \times .f_2$
• Bias = 7
• $V = 2^{1-7} \times .100 = 2^{-6} \times .5 = .0078125$

• Note that even though $e = 0000$, the exponent part is not $0 - \text{Bias}$, it's $1-\text{Bias}$. Thus, the minimum exponent part is technically $-2^{k-1} + 2$. 
Denormalized Example

• Given the bit string 0x80600000, what floating point number does it represent?
Denormalized Example

- Given the bit string 0x80600000, what floating point number does it represent?

- Writing this as a bit string gives us: 1 00000000 11000000000000000000000

- We see that this is a negative, denormalized number with value:
  
  \[ -0.11_2 \times 2^{-126} = -1.1_2 \times 2^{-127} \]
Denormalized Numeric Values

• $E = -\text{Bias} + 1$  Why this value?
• Let’s think about why we use denormalized values
• 0 00000 000
• In this representation, what is the minimum positive normalized value?
Denormalized Numeric Values

• $E = -\text{Bias} + 1$ Why this value?

• Let’s think about why we use denormalized values

• 0 0000 000

• In this representation, what is the minimum positive normalized value?

• 0 0001 000

• $V = 2^{1-\text{Bias}} * 1.000_2 = 2^{-6} * (1)_2$

• $= 2^{-6}$
Denormalized Numeric Values

• 0 0000 000

• In this representation, what is the maximum positive denormalized value?
Denormalized Numeric Values

- 0 0000 000
- In this representation, what is the maximum positive denormalized value?
  - 0 0000 111
  - \( V = 2^{1-\text{Bias}} \times .111_2 = 2^{-6} \times (2^{-1} + 2^{-2} + 2^{-3}) \)
  - \( = 2^{-6} \times (1 - 2^{-3}) \)
Denormalized Numeric Values

• 0 0000 000
• In this representation, what is the maximum positive denormalized value?
• 0 0000 111
• $V = 2^{1-Bias} \times .111_2 = 2^{-6} \times (2^{-1} + 2^{-2} + 2^{-3})$
• $= 2^{-6} \times (1 - 2^{-3})$
• Note: Denormalized numbers in this range have a granularity of $2^{-9}$. Thus, We don't use $2^{0-Bias}$ in order to maintain this gradual transition from normalized to denormalized (i.e. MIN_NORM – MAX_DENORM = $2^{-9}$).
Denormalized Numeric Values

• Note that the smallest norm and the largest denormalized are incredibly close together. How close? Thus, the normalized range flows naturally into the denormalized range because of this choice of exponent for denormalized.
Floating Point

• In class the rationale for why we have floating point values was overflow.
• To be clear, regardless of how we represent numbers, overflow is unavoidable in a representation with limited bits.
• However, what happens when an integer overflows? Consider a 4-bit signed with wrap semantics:
  • $0111 + 1 = 1000$
  • $7 + 1 = -8$
Floating Point

• With integer overflow, you get an incredibly wrong answer that looks sort of legitimate.
• Maybe with floating point, we can handle overflow by producing a less wrong answer that's easy to identify.
• One way of doing this, is to allow for the representation of infinity. If a value overflows, it becomes infinity
Floating Point: Infinity

• Here's how floating point does it.
• If e = 111...111 and f = 0, then the number is either + or – infinity:
  • 0 1111 000 = +Inf
  • 1 1111 000 = -Inf
• Now when you have an overflow, the answer is still sort of correct.
  • ex. 7 + 1 = infinity
  • Eh, good enough
Floating Point: NaN

• While we're at it, let's allow floating point to represent numbers that aren't numbers. We’ll cleverly call them not-a-numbers (NaN)?
• If e= 111...111 and f != 0, then the value is NaN
• 0 1111 000 = NaN
• 0 1111 101 = NaN
• These numbers are not a numbers.
• How many 32-bit NaN’s are there?
Floating Point: NaN

• Notes on NaN:
  1. NaN's are infectious: any operation that deals with NaN result in NaN. Ex. 1 + NaN = NaN.
  2. Despite the fact that two NaN's can have the same binary representation, (NaN == NaN) = false.
    • Ex:
      • float nan = 0.0/0.0;
      • nan == nan will evaluate to false.
Values Related to the Exponent

<table>
<thead>
<tr>
<th>Exp</th>
<th>exp</th>
<th>E</th>
<th>$2^E$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>-6</td>
<td>1/64</td>
<td>(denoms)</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>-6</td>
<td>1/64</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>-5</td>
<td>1/32</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>-4</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>-3</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>-2</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>-1</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>+1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>+2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>+3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>+4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>+5</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>+6</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>+7</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>n/a</td>
<td>(inf, NaN)</td>
<td></td>
</tr>
</tbody>
</table>
## Dynamic Range

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
<th>E</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>000</td>
<td>-6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>001</td>
<td>-6</td>
<td>$1/8 \times 1/64 = 1/512$</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>010</td>
<td>-6</td>
<td>$2/8 \times 1/64 = 2/512$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Denormalized numbers</strong></td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>110</td>
<td>-6</td>
<td>$6/8 \times 1/64 = 6/512$</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>111</td>
<td>-6</td>
<td>$7/8 \times 1/64 = 7/512$</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>000</td>
<td>-6</td>
<td>$8/8 \times 1/64 = 8/512$</td>
</tr>
<tr>
<td>0</td>
<td>0001</td>
<td>001</td>
<td>-6</td>
<td>$9/8 \times 1/64 = 9/512$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Normalized numbers</strong></td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>110</td>
<td>-1</td>
<td>$14/8 \times 1/2 = 14/16$</td>
</tr>
<tr>
<td>0</td>
<td>0110</td>
<td>111</td>
<td>-1</td>
<td>$15/8 \times 1/2 = 15/16$</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>000</td>
<td>0</td>
<td>$8/8 \times 1 = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>001</td>
<td>0</td>
<td>$9/8 \times 1 = 9/8$</td>
</tr>
<tr>
<td>0</td>
<td>0111</td>
<td>010</td>
<td>0</td>
<td>$10/8 \times 1 = 10/8$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>$14/8 \times 128 = 224$</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>110</td>
<td>7</td>
<td>$15/8 \times 128 = 240$</td>
</tr>
<tr>
<td>0</td>
<td>1111</td>
<td>000</td>
<td>n/a</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
END OF WEEK 5
ONLY 5 WEEKS MORE