FLOATING POINT

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- **Notes**
  - Homework 5 is due tonight
    - Do not try to modify the given pseudocodes
  - No class on Tuesday Feb 19th
  - Midterm on Thursday Feb 21st
    - In class, including today’s lecture

- **This lecture**
  - Floating point
A comparison requires a subtract; the sign of the result is examined; if the result is negative, the divisor must be added back.

Similar to multiply, results are placed in Hi (remainder) and Lo (quotient)
MIPS Instructions for Division

- MIPS ignores overflow in division

- Signed division (div)
  
  \[
  \begin{align*}
  \text{div} & \quad \text{s2, s3} \\
  \text{mfhi} & \quad \text{s0} \\
  \text{mflo} & \quad \text{s1}
  \end{align*}
  \]
  
  computes the division and stores it in two “internal” registers that can be referred to as hi and lo
  
  moves the remainder into \text{s0}
  
  moves the quotient into \text{s1}

- Unsigned division (divu)
  
  \[
  \begin{align*}
  \text{divu} & \quad \text{s2, s3} \\
  \text{mfhi} & \quad \text{s0} \\
  \text{mflo} & \quad \text{s1}
  \end{align*}
  \]
Signed Division

- **Simplest solution:** convert to positive and adjust sign later

- Note that multiple solutions exist for the equation
  
  \[
  \text{Dividend} = \text{Quotient} \times \text{Divisor} + \text{Remainder}
  \]

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Quo</th>
<th>Rem</th>
</tr>
</thead>
<tbody>
<tr>
<td>+7 div +2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7 div +2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+7 div -2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7 div -2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Signed Division

- **Simplest solution**: convert to positive and adjust sign later

- Note that multiple solutions exist for the equation
  
  \[ \text{Dividend} = \text{Quotient} \times \text{Divisor} + \text{Remainder} \]

  - \[ +7 \div +2 \quad \text{Quo} = +3 \quad \text{Rem} = +1 \]
  - \[ -7 \div +2 \quad \text{Quo} = -3 \quad \text{Rem} = -1 \]
  - \[ +7 \div -2 \quad \text{Quo} = -3 \quad \text{Rem} = +1 \]
  - \[ -7 \div -2 \quad \text{Quo} = +3 \quad \text{Rem} = -1 \]
Signed Division

- **Simplest solution:** convert to positive and adjust sign later

- Note that multiple solutions exist for the equation
  - **Dividend = Quotient \times Divisor + Remainder**
    - \( +7 \div +2 \) Quo = +3 Rem = +1
    - \( -7 \div +2 \) Quo = -3 Rem = -1
    - \( +7 \div -2 \) Quo = -3 Rem = +1
    - \( -7 \div -2 \) Quo = +3 Rem = -1

- **Convention**
  - Dividend and remainder have the same sign
  - Quotient is negative if signs disagree
  - These rules fulfil the equation above
Floating Point Numbers

- Example: how to represent $32.5$
  - weights smaller than 1
    $100000.100_{\text{two}}$

- Normalized scientific notation: leave a single non-zero digit to the left of the point
  - $3.25 \times 10^1$
    $1.00000100_{\text{two}} \times 2^5 = (1 \times 2^0 + 1 \times 2^{-6}) \times 2^5$

- The IEEE 754 standard
Sign-Magnitude Representation

- Since we are only representing normalized numbers, we are guaranteed that the number is of the form $1.xxxx$.

- Every 32-bit number has three fields: sign (S), exponent (E), and fraction (F)
  - value $= (-1)^S \times (1+F) \times 2^{E-127}$

- Example: 32.5
  - $32.5 = 100000.100 = 1.00000100_{two} \times 2^5$
Sign-Magnitude Representation

- Since we are only representing normalized numbers, we are guaranteed that the number is of the form $1.xxxx$.

- Every 32-bit number has three fields: sign ($S$), exponent ($E$), and fraction ($F$).
  \[
  \text{value} = (-1)^S \times (1 + F) \times 2^{E-127}
  \]

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- Example: 32.5
  \[
  32.5 = 100000.100 = 1.00000100_{\text{two}} \times 2^5
  \]
Sign-Magnitude Representation

- Since we are only representing normalized numbers, we are guaranteed that the number is of the form $1.xxxx$.

- Every 32-bit number has three fields: sign (S), exponent (E), and fraction (F)
  - value = $(-1)^S \times (1+F) \times 2^{E-127}$

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- Using more bits
  - Increase E bits to represent a wider range of numbers
  - Increase F bits to represent more precision

- Double precision format with 64 bits
  1 bit | 11 bits | 52 bits

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>11 bits</td>
<td>52 bits</td>
</tr>
</tbody>
</table>
Single Precision Floating Point

- Largest number that can be represented
- Smallest number that can be represented
Single Precision Floating Point

- Largest number that can be represented
  - $2.0 \times 2^{128} = 2.0 \times 10^{38}$

- Smallest number that can be represented
  - $1.0 \times 2^{-127} = 2.0 \times 10^{-38}$

- Overflow
  - representing a number larger than the one above

- Underflow
  - representing a number smaller than the one above
The number “0” has a special code so that the implicit 1 does not get added
- see discussion of denorms (pg. 222) in the textbook

```
0000000000000000000000000000000000000000000000000000000000 = 0
```

1 \times 2^{-126} \times 0

The largest exponent value (with zero fraction) represents +/- infinity

The largest exponent value (with non-zero fraction) represents NaN (not a number) — for the result of 0/0 or (infinity minus infinity)
Special Notes

- The number “0” has a special code so that the implicit 1 does not get added
  - see discussion of denoms (pg. 222) in the textbook

```
00000000000000000000000000000000 = 0
1 \times 2^{-126} \times 0
```

- The largest exponent value (with zero fraction) represents +/- infinity

```
11111111000000000000000000000000 = -Infinity
-1 \times 2^{128} \times 1
```

- The largest exponent value (with non-zero fraction) represents NaN (not a number) — for the result of 0/0 or (infinity minus infinity)
Special Notes

- The number “0” has a special code so that the implicit 1 does not get added
  - see discussion of denorms (pg. 222) in the textbook
    \[
    0\ldots0000000000000000000000 = 0
    \]
    \[
    1 \times 2^{-126} \times 0
    \]

- The largest exponent value (with zero fraction) represents +/- infinity
  \[
  1\ldots11111110000000000000000000000000 = -\text{Infinity}
  \]
  \[
  -1 \times 2^{128} \times 1
  \]

- The largest exponent value (with non-zero fraction) represents NaN (not a number) — for the result of 0/0 or (infinity minus infinity)
  \[
  1\ldots11111111000000000000000000000001 = \text{NaN}
  \]
  \[
  -1 \times 2^{128} \times 1.00000001
  \]
Special Notes

- The number “0” has a special code so that the implicit 1 does not get added
  - see discussion of denorms (pg. 222) in the textbook

- The largest exponent value (with zero fraction) represents +/- infinity

- The largest exponent value (with non-zero fraction) represents NaN (not a number) — for the result of 0/0 or (infinity minus infinity)

These choices impact the smallest and largest numbers that can be represented
Example 1

- Represent $-0.75_{\text{ten}}$ in single- and double-precision formats

  \[
  \text{value} = (-1)^S \times (1 + F) \times 2^{(E - 127)}
  \]

- Single

  \[
  \begin{array}{ccc}
  \text{1 bit} & \text{8 bits} & \text{23 bits} \\
  S & E & F
  \end{array}
  \]

- Double

  \[
  \begin{array}{ccc}
  \text{1 bit} & \text{11 bits} & \text{52 bits} \\
  S & E & F
  \end{array}
  \]
Example 1

- Represent -0.75\text{ten} in single- and double-precision formats

- \(-0.75 = -1 \times 0.11_{\text{two}} = -1 \times 1.1_{\text{two}} \times 2^{-1} = (-1)^1 \times (1 + F) \times 2^{(E - 127)}\)

- Single

- **1 bit**
- **8 bits**
- **23 bits**

1 \[
\begin{array}{c}
1 \\
01111110 \\
\end{array}
\]
126

10000000000000000000000000000000...000
0.5

- **Double**

- \(-1 \times 1.1_{\text{two}} \times 2^{-1} = (-1)^1 \times (1 + F) \times 2^{(E - 1023)}\)

- **1 bit**
- **11 bits**
- **52 bits**

1 \[
\begin{array}{c}
1 \\
01111111110 \\
\end{array}
\]
1022

10 000000000000000000000000000000000000...000
0.5
Example 2

- Represent $3.40625_{\text{ten}}$ in single- and double-precision formats
  
  \[
  \text{value} = (-1)^S \times (1 + F) \times 2^{(E - 127)}
  \]

- Single

<table>
<thead>
<tr>
<th>1 bit</th>
<th>8 bits</th>
<th>23 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

- Double

<table>
<thead>
<tr>
<th>1 bit</th>
<th>11 bits</th>
<th>52 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>
Example 2

- Represent $3.40625_{\text{ten}}$ in single- and double-precision formats

  $3.40625 = 11.01101_{\text{two}} = 1.101101_{\text{two}} \times 2^1 = (-1)^0 \times (1 + F) \times 2^{(E - 127)}$

- Single

  - 1 bit
    - 0
  - 8 bits
    - 10000000
    - 128
  - 23 bits
    - 10110100000000000000000000000000
    - 0.703125

- Double

  - 1 bit
    - 0
  - 11 bits
    - 10000000000
    - 1024
  - 52 bits
    - 1011010000000000000000000000000000...000
    - 0.703125
Floating Point Addition

- Numbers maintain only 4 decimal digits and 2 exponent digits
  - $9.999 \times 10^1 + 1.610 \times 10^{-1}$

- Convert to the larger exponent
  - $9.999 \times 10^1 + 0.016 \times 10^1$

- Add
  - $10.015 \times 10^1$

- Normalize
  - $1.0015 \times 10^2$

- Check for overflow/underflow

- Round
  - $1.002 \times 10^2$

- Re-normalize
Floating Point Addition

- Numbers maintain only 4 decimal digits and 2 exponent digits
  - $9.999 \times 10^1 + 1.610 \times 10^{-1}$

- Convert to the larger exponent
  - $9.999 \times 10^1 + 0.016 \times 10^1$

- Add
  - $10.015 \times 10^1$

- Normalize
  - $1.0015 \times 10^2$

- Check for overflow/underflow
- Round
  - $1.002 \times 10^2$

- Re-normalize

If we had more fraction bits, these errors would be minimized
Floating Point Addition

- Numbers maintain only 4 binary digits and 2 exponent digits
  - $1.010 \times 2^1 + 1.100 \times 2^3$

- Convert to the larger exponent
  - $0.0101 \times 2^3 + 1.100 \times 2^3$

- Add
  - $1.1101 \times 2^3$

- Normalize
  - $1.1101 \times 2^3$

- Check for overflow/underflow

- IEEE 754 format
  
  0 10000010 11010000000000000000000000000000