Number Representation

Sandhya Dwarkadas

Number Representation

An $n$ digit number can be represented in any base as

\[
\begin{array}{ll}
\text{MSD} & \ldots \\
\text{n-1} & \ldots \\
\end{array}
\]

LSD

The value of the $i$th digit $d$ is $d \times \text{base}^i$, where $i$ starts at 0 and increases from right to left.

Decimal (base 10) is the natural human representation, binary (base 2) is the natural computer representation.

E.g. $1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 12_{10}$
**Negative Number Representation**

Sign-Magnitude - MSB represents sign (0 for +ve, 1 for -ve)

One's Complement \((1's\ complement \ of \ x = 2^n - x - 1)\)

**Problem:** Balanced representation, but two values for 0

**Solution:** Two’s Complement \((2's\ complement \ of \ x = 2^n - x)\)

- single bit pattern for 0
- ensures that \(x + (-x)\) is 0
- still keeps 1 in MSB for a -ve number (sign bit)
- 100... represents the most -ve number
- E.g. 4-bit 2’s complement number
  \[1100_2 = -1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = -4_{10}\]

**Exceptions**

**Overflow:** number too large to be represented in \(n\) bits

Overflow condition for \(O = A + B\):
\[
MSBA.MSBB.MSBO + MSBA.MSBB.MSBO
\]

Detection of overflow ignored in C, required in Fortran

Memory addressing arithmetic on unsigned numbers as well

Therefore, MIPS provides both signed and unsigned arithmetic, with an exception/interrupt generated on overflow for the signed arithmetic
Shortcuts

2’s complement = 1’s complement + 1

2’s complement representation of $n$ digit number as $n + m$ digit number - sign extend

Recall Scientific Notation

$65.4 = 6.54 \times 10^1$ (normal form - 1 non-zero leading digit)
Mantissa = 6.54, Exponent = 1, Radix (base) = 10

$-0.0654 = -6.54 \times 10^{-2}$
IEEE 754 Floating Point Representation

Sign and magnitude Mantissa representation

Biased notation for Exponent (0 represents most negative/smallest exponent)

**Single precision** - $\pm 1.M \times 2^{E-127}$

```
<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>M</th>
</tr>
</thead>
</table>
```

1 8 23

<table>
<thead>
<tr>
<th>sign exponent</th>
<th>significand</th>
</tr>
</thead>
</table>

**Double precision** - $\pm 1.M \times 2^{E-1023}$

```
<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>M</th>
</tr>
</thead>
</table>
```

1 11 52

Trade-offs between range and accuracy
**Floating Point Addition**

Align decimal point by matching larger exponent

Add significands (possibly unnormalized)

Normalize - shift sum/adjust exponent

Round number to fit in significand field

---

**Floating Point - Additional Bits for Accuracy**

Borrow/carry bit - to take into account overflow (carry/borrow) in result

Guard bit - to take care of normalizing left shift

Round bit - for correct rounding

Sticky bit - fine-tune rounding - additional bit to the right of round that is set to 1 if any 1 bit falls off the end of the round digit
IEEE 754 Floating Point Rounding

Four modes:
1. Round to $+\infty$ - add 1 if round or sticky bit set and result $\geq 0$
2. Round to $-\infty$ - add 1 if round or sticky bit set and result $< 0$
3. Round to 0 (truncate)
4. Round to nearest number (default) - round to even when exactly half-way
   - add 1 if round bit and least significant bit of result are set, or, if round bit and sticky bit are set
   - minimizes mean error introduced by rounding

Floating Point Multiplication

Exponent of product = sum of exponents - bias

Multiply significands - decimal point location = sum of digits to the right of decimals

Normalize - check for overflow/underflow

Round

Sign - +ve if both are the same, -ve otherwise
Zero, Infinity, NaN

Zero - 0|0...0|0...0

+/- - ∞ - S|1...1|0...0

NaN - S|1...1|non-zero

Exceptions

IEEE standard specifies defaults and allows traps to permit user to handle exception

- Invalid operation: e.g., sqrt of -ve number, 0/0, ∞/∞ - result is NaN
- Overflow: result is +/- - ∞ unless overflow exception is enabled
- Divide by 0: result is +/- - ∞ if exception not enables
- Underflow: non-zero result underflows to 0
- Inexact: rounded result not the actual result
Converting $0.15_{10}$ to binary

<table>
<thead>
<tr>
<th>remainder</th>
<th>remainder $\times 2$</th>
<th>leading digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>0.30</td>
<td>0</td>
</tr>
<tr>
<td>0.30</td>
<td>0.60</td>
<td>0</td>
</tr>
<tr>
<td>0.60</td>
<td>1.20</td>
<td>1</td>
</tr>
<tr>
<td>0.20</td>
<td>0.40</td>
<td>0</td>
</tr>
<tr>
<td>0.40</td>
<td>0.80</td>
<td>0</td>
</tr>
<tr>
<td>0.80</td>
<td>1.60</td>
<td>1</td>
</tr>
<tr>
<td>0.60</td>
<td>1.20</td>
<td>1</td>
</tr>
</tbody>
</table>

Repeats...

$0.15_{10} = 0.00100110011..._2$