Fixed Point Arithmetic
Fixed Point

* Intro
  - What is Fixed Point
  - Relation To Floating Point
  - Why Several Choices

* Architectural Choices
  - 2's Complement
  - 1's Complement
  - Signed Magnitude
  - Long Integer
  - BCD
  - Residue Numbers

* Microarchitecture Mechanisms
  - Addition (The Carry Problem)
    - LAC
    - Kogge-Stone
    - The Power Ball (2x Frequency)
  - Multiplication (The Iteration Problem)
    - Booth's Algorithm
Fixed Point
Equal Intervals

Floating Point
Not-equal Intervals

$1.00 - 0 \times 2^k$
$1.00 - 0 \times 2^k$
$10.00 - 0 \times 2^k$

$1.00 \times 2^{k+1}$
$1.00 \times 2^{k+1}$
$1.00 \times 2^{k+2}$
$1.00 \times 2^{k+3}$

$-\infty \rightarrow 0 \rightarrow +\infty$

$\text{NEAREST}$

$A - \Delta \rightarrow A \rightarrow A + \Delta$

Rounding

$1.0010 \times 2^k$
$1.0011 \times 2^k$
Fixed Point vs. Floating Point

**Fixed Point:** Binary point always in the same place

\[
\begin{array}{c}
0.00000 \\
111111 \\
0 \leq x \leq 31
\end{array}
\quad \begin{array}{c}
.00000 \\
11111 \\
0 \leq x \leq \frac{31}{32}
\end{array}
\quad \begin{array}{c}
00.000 \\
11.111 \\
0 \leq x \leq \frac{7}{8}
\end{array}
\]

**Interval = 1**

**Interval = \frac{1}{32}**

**Interval = \frac{1}{8}**

**Floating Point:** Binary point moves from binary to binary

![Number line diagram](image)

In Above Example, 2 bits of fraction. Therefore, 3 significant digits.

**Note:** Floating point moves

**Note:** Interval changes
Computer Arithmetic (Integers)

* Why several choices for representation

* The Choices
  - 2's complement
  - 1's complement
  - Signed magnitude
  - Long integers — Karatsuba's Trick
  - Decimal (BCD)
  - Residue Arithmetic
* Why several choices?

Application space should drive architecture

- Compute intensive, low I/O
- Arbitrarily large precision
- Generally within a fixed set, with option to go to multiples of that size

* The concept of "Long Integer" vs "Short Integer"
2's Complement 1's Complement

\[ \begin{array}{c}
0011 \\
1100 \\
1101 \leftarrow -3 \\
\end{array} \]

\[ -5 + 3 \]

Ridiculous

Long Integers:

\[ \text{ADDR} \]

\[ \text{ADC} \]

\[ \text{NOT a sign bit} \]
Three Most Common Schemes

- 2's complement
- 1's complement
- Signed magnitude

Example: (A 4-bit data path)

<table>
<thead>
<tr>
<th>Representation</th>
<th>2’s Comp</th>
<th>1’s Comp</th>
<th>Sign-Mag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>-8</td>
<td>-7</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>-7</td>
<td>-6</td>
<td>2</td>
</tr>
<tr>
<td>1010</td>
<td>-6</td>
<td>-5</td>
<td>2</td>
</tr>
<tr>
<td>1011</td>
<td>-5</td>
<td>-4</td>
<td>3</td>
</tr>
<tr>
<td>1100</td>
<td>-4</td>
<td>-3</td>
<td>4</td>
</tr>
<tr>
<td>1101</td>
<td>-3</td>
<td>-2</td>
<td>5</td>
</tr>
<tr>
<td>1110</td>
<td>-2</td>
<td>-1</td>
<td>6</td>
</tr>
<tr>
<td>1111</td>
<td>-1</td>
<td>-0</td>
<td>7</td>
</tr>
</tbody>
</table>

★ Why 2's Complement? (Easy for Computer)
★ Why 1's Complement (Self-Delusion)
★ Why Signed-Magnitude? (Easy for Humans)
BCD Arithmetic

\[
\begin{array}{c}
657 \\
0110 0101 0111 \\
0010 1001 0100 \\
1000 1110 1011
\end{array}
\]

\[
\begin{array}{c}
657 \\
294 \\
8\text{EB} \\
951
\end{array}
\]

\[
\begin{array}{c}
0110 0101 0111 \\
0110 0110 0110 \\
1100 1011 1101 \\
0010 0011 0100 \\
1111 0101 0001 \\
1
\end{array}
\]
Decimal Arithmetic

(Or, Variable Length, Packed BCD.)

* Each decimal digit represented by a 4-bit code

* Special ALU or 3 cycles per iteration

* A value requires two elements (Addr, length)

* Example: ADD 2F3 to 598

With binary ALU in one cycle:

\[
\begin{array}{c}
\text{0010 1000 0011} \\
\text{0101 1001 1000} \\
\text{1000 0001 1011}
\end{array}
\]

GARBAGE \rightarrow 8 1 B

Why?

With constant 666 and three cycles:

(1) 283 + 666 \rightarrow 8E9

(2) 8E9 + 598 \rightarrow E8*1

(3) E81 - 600 \rightarrow 881

Why subtract 600?
Résidue Arithmétique (Proof)

\[ A = p_1 m + a_1 \]
\[ B = p_1 n + a_2 \]
\[ A \times B = \]

\[
\begin{pmatrix}
A \\
B
\end{pmatrix}
= \begin{pmatrix}
p_1 m + a_1 \\
p_1 n + a_2
\end{pmatrix}
\]

\((p_1 m + a_1)(p_1 n + a_2)\)
\[
p_1 m n + p_1 m a_2 + p_1 a_1 n + a_1 a_2
\]

\[
A + B = p_1 m + p_1 n + a_1 + a_2
\]

\[
\begin{pmatrix}
0 \rightarrow \\
\infty \rightarrow
\end{pmatrix}
\]

\[
\begin{pmatrix}
f(c) \rightarrow \\
f(\infty) \rightarrow
\end{pmatrix}
\]
Residue Arithmetic

* When?

- Inputs, outputs are short integers
- Intermediate results may be very large
- Internally compute intensive, as opposed to having to do substantial I/O

* How?

```
Transform to Residue Representation
a, b  ---->  (Slow)  f(a), f(b)
     |                    Perform the operation (VERY FAST)
     v                    
Inverse transform
a * b  ---->  (Slow)  f(a) * f(b)
```

Residue Arithmetic (Continued)

* In greater detail,

- Pick a set of moduli $p_1, p_2 \ldots p_k$ such that they are all relatively prime.

- We can represent $x$ as $x_1 x_2 \ldots x_k$ where $x_i = x \mod p_i$.

- If $0 \leq x < \frac{\prod p_i}{2}$, or more realistically

  $\frac{\prod p_i}{2} \leq x < \frac{\prod p_i}{2}$

  Then this representation for $x$ is unique.

- From which $x + y$ and $x \times y$ can be computed concurrently by $k$ processing elements, each one computing the result $\mod p_i$.

* Why don't we do it?

- Transformations expensive
- Comparisons unwieldy
Residue Arithmetic (Examples)

As in class: \( p_1 = 7 \), \( p_2 = 8 \), \( p_3 = 9 \); \( \Pi p_i = 504 \)

For these examples, let's use only positives:

\[ 0 \leq x < 503 \]

1. Representations:
   
   \[ 19 = 531 \]
   \[ 24 = 306 \]

2. Addition:
   
   \[
   \begin{array}{c}
   19 \\
   + 24
   \end{array}
   \]
   
   \[
   \begin{array}{c}
   531 \\
   306
   \end{array}
   \]
   
   \[
   \begin{array}{c}
   47 \\
   137
   \end{array}
   \]

3. Multiplication:
   
   \[
   \begin{array}{c}
   19 \\
   \times 24
   \end{array}
   \]
   
   \[
   \begin{array}{c}
   531 \\
   306
   \end{array}
   \]
   
   \[
   \begin{array}{c}
   76 \\
   106
   \end{array}
   \]
   
   \[
   \begin{array}{c}
   38 \\
   456
   \end{array}
   \]

4. Why it works:

   \[
   A \times B = (m p_i + a) \times (n p_i + b)
   \]
   
   \[
   = p_i (m n p_i + a n + b m) + a b
   \]

   \[ (A \times B) \mod p_i = a b \]
Residue Arithmetic (continued)

Inverse Transformation

Let \( X \) be represented as \( X_1 X_2 X_3 \). What is \( X \)?

\[
X_1 X_2 X_3 = X_1 (100) + X_2 (010) + X_3 (001)
\]

100 is a multiple of 72 that has a residue of 1 for \( p = 7 \), i.e. 288.

Similarly, 010 is a multiple of 63, i.e. 441

Similarly, 001 is a multiple of 56, i.e. 280.

Thus \( X \) can be obtained by adding \( X_1 \times 288 + X_2 \times 441 + X_3 \times 280 \), and finding the residue mod 504.

A simpler hardware mechanism:

\[
\begin{array}{c|c|c|c}
X_1 & 0 & 288 & 72 \\
260 & 144 & 432 & 216 \\
\hline
X_2 & 0 & 441 & 378 \\
315 & 252 & 189 & 126 \\
\hline
X_3 & 0 & \\
\hline
\end{array}
\]

\[+ \]

\[\text{mod 504} \]

\[X\]
Booth's Algorithm (A Variation)

\[
\begin{array}{cccc}
\text{Control 1} & \text{ALU} & \text{Control 2} & \text{C'} \\
\hline
000 & \text{Pass} & \text{Pass} & \text{SHT 2} & 0 \\
010 & \text{Pass} & \text{Add} & \text{SHT 2} & 0 \\
000 & \text{SHT 1} & \text{Add} & \text{SHT 1} & 0 \\
110 & \text{Pass} & \text{Sub} & \text{SHT 2} & 1 \\
001 & \text{Pass} & \text{Add} & \text{SHT 2} & 0 \\
011 & \text{SHT 1} & \text{Add} & \text{SHT 1} & 0 \\
101 & \text{Pass} & \text{Sub} & \text{SHT 2} & 1 \\
111 & \text{Pass} & \text{Pass} & \text{SHT 2} & 1 \\
\end{array}
\]

\[
\text{Example: } \frac{001011}{2} = 11
\]

\[
\begin{align*}
\text{SHT1} & \\
00 & 11 \quad (-1) \\
01 & \quad (-1) \\
\text{+1} & \\
\end{align*}
\]

\[1 \times 4^2 - 1 \times 4^1 - 1 \times 4^0\]