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 Integers
  - BCD
  - Residue Numbers

* Microarchitecture Mechanisms
  - Addition (The Carry Problem)
    • CAC
    • Kogge-Stone
    • The Power Ball (2x Frequency)
  - Multiplication (The Iteration Problem)
    • Booth's Algorithm
**Fixed Point vs. Floating Point**

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

\[ 0.00000 \quad .00000 \quad 00.000 \]

\[ \begin{array}{c}
0 \leq x \leq 31 \\
\text{interval} = 1
\end{array} \quad \begin{array}{c}
0 \leq x \leq \frac{7}{32} \\
\text{interval} = \frac{1}{32}
\end{array} \quad \begin{array}{c}
0 \leq x \leq \frac{7}{8} \\
\text{interval} = \frac{1}{8}
\end{array} \]

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

\[ \begin{array}{c}
0.101 \\
1.01 \\
10.1
\end{array} \]

In above example, 2 bits of fraction. Therefore 3 significant digits.

**Binary 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
- 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"**
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 283 TO 598

   WITH BINARY ALU IN ONE CYCLE:
   \[ \begin{array}{c}
   0010 \ 1000 \ 0011 \\
   0101 \ 1001 \ 1000 \\
   \hline
   1000 \ 0001 \ 1011
   \end{array} \]

GARBAGE \[ \rightarrow \]

WHY?

WITH CONSTANT 666 AND THREE CYCLES

(1) 283 + 666 \[ \rightarrow \] 8E9

(2) 8E9 + 598 \[ \rightarrow \] E81*  

(3) E81 - 600 \[ \rightarrow \] 881

WHY SUBTRACT 600?
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?

\[
\begin{align*}
& a, b \\
& \quad \xrightarrow{\text{Transform to Residue Representation}} \\
& \quad \quad \quad \quad \quad \quad \quad (\text{Slow}) \\
& \quad \quad \quad \quad \quad \quad \quad f(a), f(b) \\
& \quad \quad \quad \quad \quad \quad \quad \xrightarrow{\text{Perform the operation}} \\
& \quad \quad \quad \quad \quad \quad \quad (\text{VERY FAST}) \\
& \quad \quad \quad \quad \quad \quad \quad \xrightarrow{\text{Inverse transform}} \\
& a \times b \\
& \quad \quad \quad \quad \quad \quad \quad (\text{Slow}) \\
& \quad \quad \quad \quad \quad \quad \quad f(a) \times f(b)
\end{align*}
\]
Residue Arithmetic (continued)

* IN GREATER DETAIL,

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

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

- If $0 \leq x < \pi p_i$, or more realistically

$$\frac{-\pi p_i}{2} < x < \frac{\pi p_i}{2}$$

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: $p_1 = 7, p_2 = 8, p_3 = 9; \sum p_i = 504$

For these examples, let's use only positives.

$0 \leq x < 503$

(1) Representations:

$19 = 531$

$24 = 306$

(2) Addition:

\[
\begin{array}{c}
19 \\
+ 24 \\
\hline
43
\end{array}
\begin{array}{c}
531 \\
306 \\
\hline
137
\end{array}
\]

(3) Multiplication:

\[
\begin{array}{c}
19 \\
\times 24 \\
\hline
76 \\
38
\end{array}
\begin{array}{c}
531 \\
306 \\
\hline
456
\end{array}
\]

(4) Why it works:

\[
A \times B = (mp_i + a) \times (np_i + b)
\]

\[
= p_i (mn p_i + an + bm) + ab
\]

\[
\text{Example: } (A \times B) \text{ mod } p_i = ab
\]
**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:
Look-Ahead Carry Generator

A
B
A+B

Slice 1  Slice 2  Slice 3  Slice 4

In order to get $S_{12}$, for example, we need $A_{12}, B_{12}, S_{12}$. Rather than wait for $C_{12}$ to propagate (ripple) from $C_0$, we note that $C_{12}$ is 1 if Slice 2 produces a carry.

Slice 2 produces a carry if either Slice 2 by itself produces a carry ($C_2$) or if the sum of the corresponding bits by themselves add to 1111 and $C_8 = 1$. We say in the latter case, Slice 2 propagates a carry ($P_2$).

Thus:
1. We see (in parallel) if Slice 2 produces $G_2, P_2$.
2. We feed $G_0, P_0, G_1, P_1, G_2, P_2, G_3, P_3$ into a two-level combinational logic circuit (the LAC), which produces:

$$C_{12} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_8 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_4 = G_0 + P_0 C_0$$

3. We can now add the Slices in parallel with the correct Carry in $\overline{C_0}$ each slice.