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

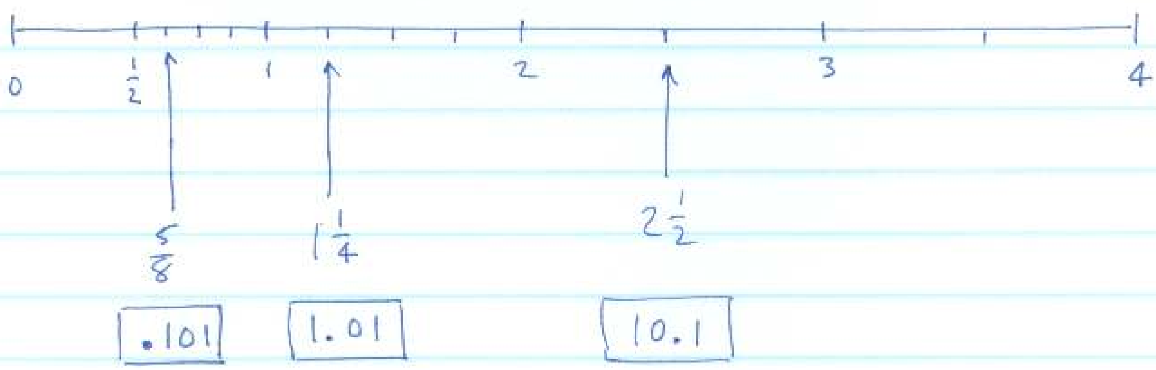
* MICROARCHITECTURE MECHANISMS
  - ADDITION (THE CARRY PROBLEM)
    - LAC
    - KOGGE-STONE
    - THE POWER BALL (2X FREQUENCY)
  - MULTIPLICATION (THE ITERATION PROBLEM)
    - BOOTH'S ALGORITHM

FIXED POINT VS. FLOATING POINT

FIXED POINT : BINARY POINT ALWAYS IN THE SAME PLACE

00000.	.00000	00.000
⋮	⋮	⋮
11111.	.11111	11.111
$0 \le X \le 31$	$0 \le x \le \frac{31}{32}$	$0 \le x \le 3\frac{7}{8}$
$\boxed{\text{INTERVAL} = 1}$	$\boxed{\text{INTERVAL} = \frac{1}{32}}$	$\boxed{\text{INTERVAL} = \frac{1}{8}}$

FLOATING POINT : BINARY POINT **MOVES** FROM BINADE TO BINADE



IN ABOVE EXAMPLE, 2 BITS OF FRACTION.
THEREFORE 3 SIGNIFICANT DIGITS.

NOTE : ~~FLOATING~~ BINARY 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"

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:

```
  0010 1000 0011
  0101 1001 1000
  --------------
  1000 0001 1011
```

GARBAGE → 8 1 B

WHY?

WITH CONSTANT 666 AND THREE CYCLES

(1) 283 + 666 → 8E9
(2) 8E9 + 598 → E8*1*
(3) E81 − 600 → 881

$\boxed{\text{WHY SUBTRACT 600?}}$

ResidueArithmetic

* 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?

$$\begin{tikzcd}
a,b \arrow[r, "\text{Transform to Residue Representation}", "\text{(Slow)}"'] & f(a), f(b) \arrow[d, "\text{Perform the operation (VERY FAST)}"] \\
a * b & f(a) * f(b) \arrow[l, "\text{Inverse transform}"', "\text{(Slow)}"]
\end{tikzcd}$$

RESIDUE ARITHMETIC (CONTINUED)

* IN GREATER DETAIL,

  - PICK A SET OF MODULI $p_1, p_2 \cdots p_K$ SUCH THAT THEY ARE ALL RELATIVELY PRIME.

  - WE CAN REPRESENT X AS $X_1 X_2 \cdots X_K$ WHERE $X_i = X \bmod p_i$

  - IF $0 \leq X < \prod p_i$, OR MORE REALISTICALLY

    IF $-\frac{\prod p_i}{2} \leq X < +\frac{\prod p_i}{2}$,

    THEN THIS REPRESENTATION FOR X IS UNIQUE

  - FROM WHICH $X+Y$ AND $X*Y$ CAN BE COMPUTED CONCURRENTLY BY K PROCESSING ~~EL~~ ELEMENTS, EACH ONE COMPUTING THE RESULT $\bmod p_i$.

* WHY DON'T WE DO IT?

  - TRANSFORMATIONS EXPENSIVE
  - COMPARISONS UNWIELDLY

RESIDUE ARITHMETIC (EXAMPLES)

AS IN CLASS : $p_1 = 7$, $p_2 = 8$, $p_3 = 9$; $\prod p_i = 504$

FOR THESE EXAMPLES, LET'S USE ONLY POSITIVES.

$$0 \leq X < 503$$

(1) REPRESENTATIONS :

$$19 = 5\ 3\ 1$$
$$24 = 3\ 0\ 6$$

(2) ADDITION :

$$\begin{array}{r} 19 \\ +24 \\ \hline 43 \end{array} \qquad \begin{array}{ccc} 5 & 3 & 1 \\ 3 & 0 & 6 \\ \hline 1 & 3 & 7 \end{array}$$

(3) MULTIPLICATION :

$$\begin{array}{r} 19 \\ 24 \\ \hline 76 \\ 38 \\ \hline 456 \end{array} \qquad \begin{array}{ccc} 5 & 3 & 1 \\ 3 & 0 & 6 \\ \hline 1 & 0 & 6 \end{array}$$

(4) WHY IT WORKS:

$$A * B = (m p_i + a) * (n p_i + b)$$
$$= p_i(mn p_i + an + bm) + ab$$

$$(A * B) \bmod p_i = ab$$

RESIDUE ARITHMETIC (CONTINUED)

INVERSE TRANSFORMATION

LET X BE REPRESENTED AS $X_1 X_2 X_3$.
WHAT IS X?

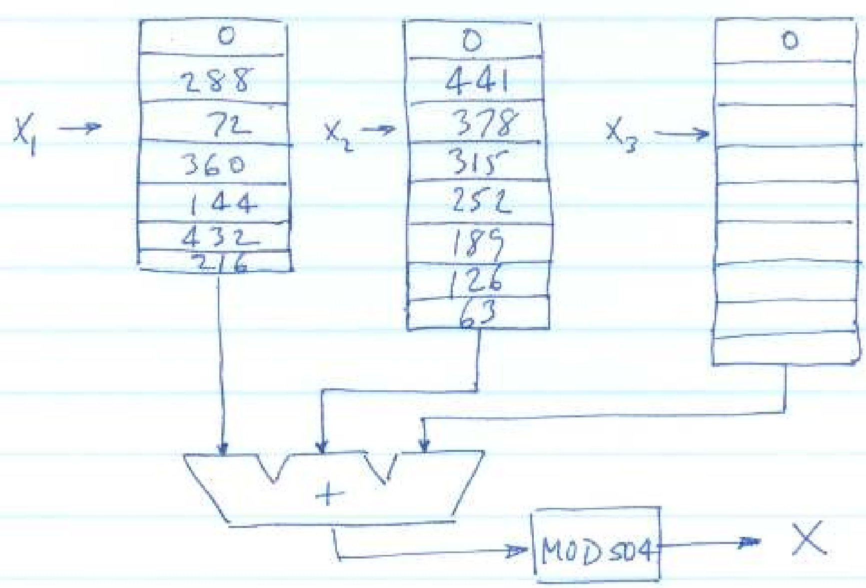
$$X_1 X_2 X_3 = X_1(100) + X_2(010) + X_3(001)$$

100 IS A MULTIPLE OF 72 THAT HAS A RESIDUE OF 1 FOR P = 7. i.e. 288.

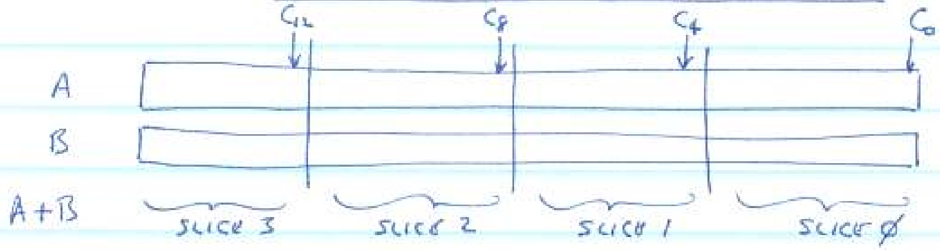
SIMILARLY, 010 IS A MULTIPLE OF 63. i.e. 441
SIMILARLY, 001 IS A MULTIPLE OF 56. i.e. 280.

THUS X CAN BE OBTAINED BY ADDING
$X_1 * 288 + X_2 * 441 + X_3 * 280$, AND
FINDING THE RESIDUE MOD 504.

A SIMPLER HARDWARE MECHANISM:



LOOK-AHEAD CARRY GENERATOR



IN ORDER TO GET S_{12}, FOR EXAMPLE, WE NEED A_{12}, B_{12}, C_{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 (G_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 i PRODUCES G_i, P_i.

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 TO EACH SLICE.