Fixedpt/1

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

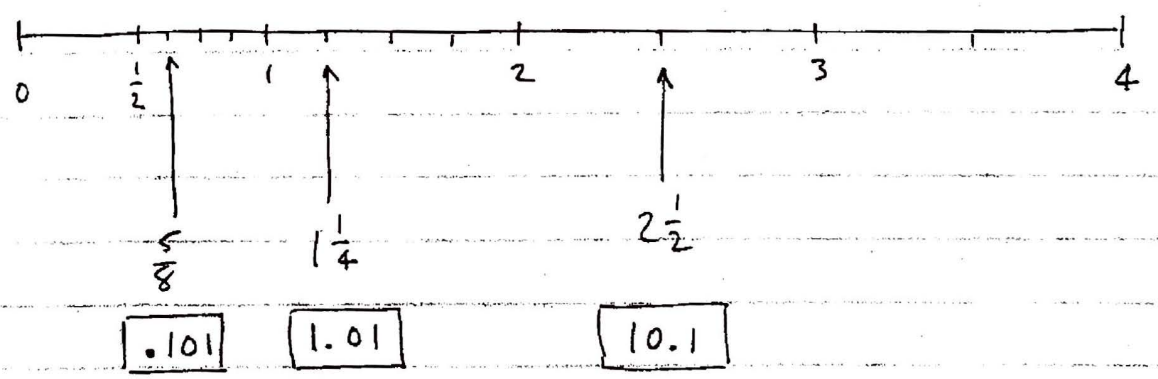
* MICROARCHITECTURE MECHANISMS
  - ADDITION (THE CARRY PROBLEM)
    - LAC
    - KOGGE-STONE
    - THE POWER BALL (2x FREQUENCY)
  - MULTIPLICATION (THE ITERATION PROBLEM)
    - BOOTH'S ALGORITHM

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

Fixed Pt. /4

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"

Three Most Common Schemes

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

Example: (A 4-bit data path)

Representation	What is being Represented		
	2's Comp	1's Comp	Sign-Mag
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-8	-7	0
1001	-7	-6	2
1010	-6	-5	2
1011	-5	-4	3
1100	-4	-3	4
1101	-3	-2	5
1110	-2	-1	6
1111	-1	-0	7

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

0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101 ← 5
6	6	0110
7	7	0111
-7	-8	1000
-6	-7	1001
-5	-6	1010 ← -5
-4	-5	1011
-3	-4	1100
-2	-3	1101 ← -3
-1	-2	1110
-0	-1	1111

1's Complement

2's Complement

2's complement

1's complement

Fixed Pt/1

```
  0011
  1100
     1
  1101 ← -3
```

-5 + 3

```
  1111
     1
  0000 → 0
```

-3 + 5

Long Integers:

Sign bit

```
1 ✓ 0 ✓ A
  ✓ 0 ✓ B
  ------
1   1   ✓
```

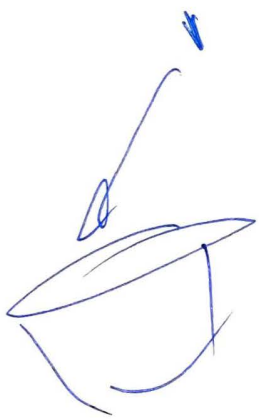
NOT a sign bit

Ridiculous!

ADDR

ADC

```
 1110   -2
 0001
 ----   -1
 1111
 0001   0
 ----
 0000
```



WHY THE CARRY BIT CAN GO IN THE TRASH!

(FOR 2'S COMPLEMENT)

WHAT DO WE DO IN 1'S COMPLEMENT?

WHY?

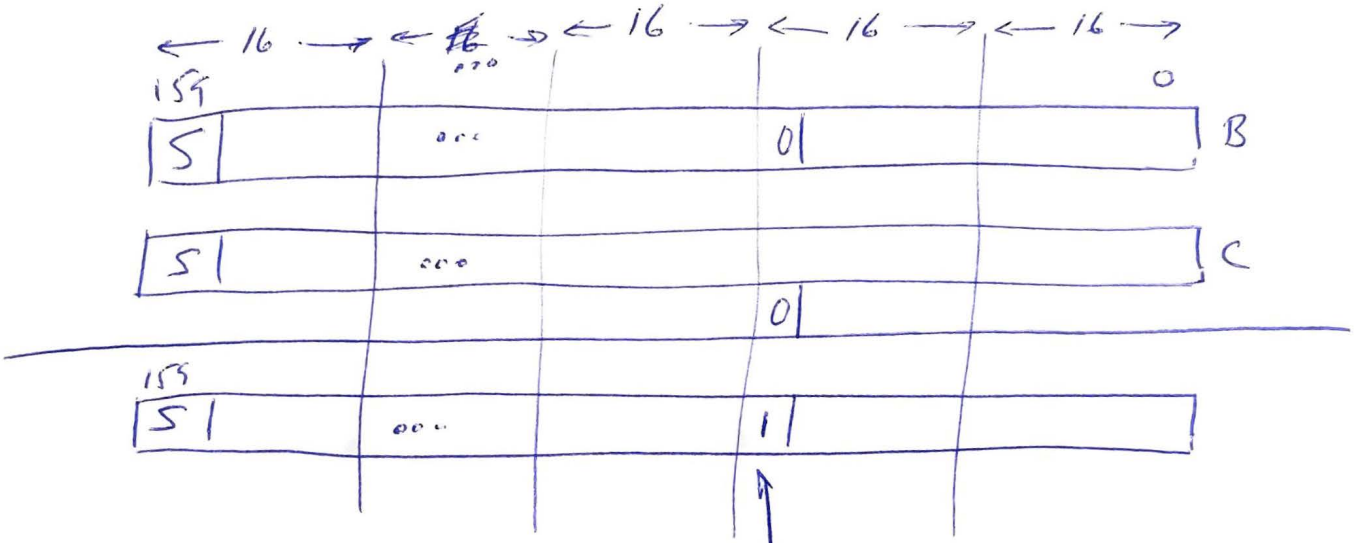
Another Example

Long Integers

ADDR A, B, C

↑

Ridiculous



WHY IS THIS NOT A PROBLEM

$\boxed{\text{ADDC } R1, R2, R3 \rightarrow R1 \leftarrow R2 + R3 + C}$

160 BIT ADD → 10 ITERATIONS

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

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

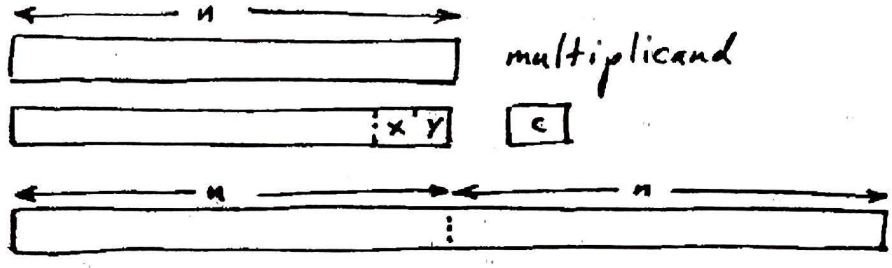
BCD Arithmetic
(Another Example)

6	5	7
0110	0101	0111
0010	1001	0100
1000	1110	1011

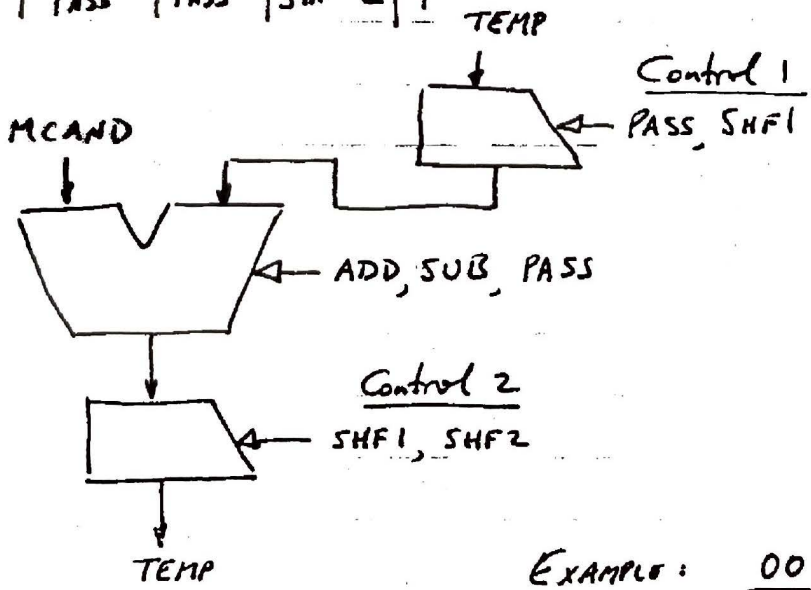
```
  1 1
  6 5 7
  2 9 4
  -----
  8 E B
  -----
  9 5 1
```

0110	0101	0111
0110	0110	0110
1100	1011	1101
0010	1001	0100
1111	0101	0001
		1

Booth's Algorithm (A Variation)



x	y	c	Control 1	ALU	Control 2	c'
0	0	0	PASS	PASS	SHF 2	0
0	1	0	PASS	ADD	SHF 2	0
1	0	0	SHF 1	ADD	SHF 1	0
1	1	0	PASS	SUB	SHF 2	1
0	0	1	PASS	ADD	SHF 2	0
0	1	1	SHF 1	ADD	SHF 1	0
1	0	1	PASS	SUB	SHF 2	1
1	1	1	PASS	PASS	SHF 2	1



Example: 00 10 11 = 11

Step 1: 00 11 (-1)

2: 01 (-1)

3: (+1)

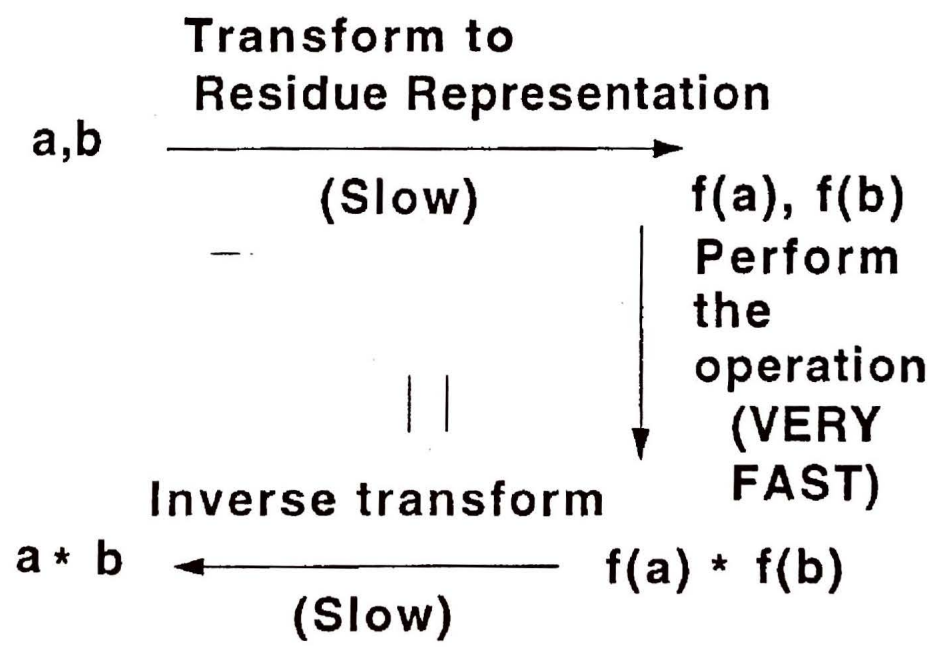
$1 \times 4^2 - 1 \times 4^1 - 1 \times 4^0$

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?



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 $0 \le X < \Pi p_i$, OR MORE REALISTICALLY

    IF $-\frac{\Pi p_i}{2} \le x < +\frac{\Pi p_i}{2}$,

    THEN THIS REPRESENTATION FOR X IS <u>UNIQUE</u>

  - FROM WHICH $X+Y$ AND $X*Y$ CAN BE COMPUTED CONCURRENTLY BY K PROCESSING 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.

$$0 \le X < 503$$

(1) REPRESENTATIONS :

$$19 = 531$$
$$24 = 306$$

(2) ADDITION :

$$\begin{array}{r} 19 \\ +24 \\ \hline 43 \end{array} \qquad \begin{array}{r} 531 \\ 306 \\ \hline 137 \end{array}$$

(3) MULTIPLICATION :

$$\begin{array}{r} 19 \\ 24 \\ \hline 76 \\ 38 \\ \hline 456 \end{array} \qquad \begin{array}{r} 531 \\ 306 \\ \hline 106 \end{array}$$

(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)

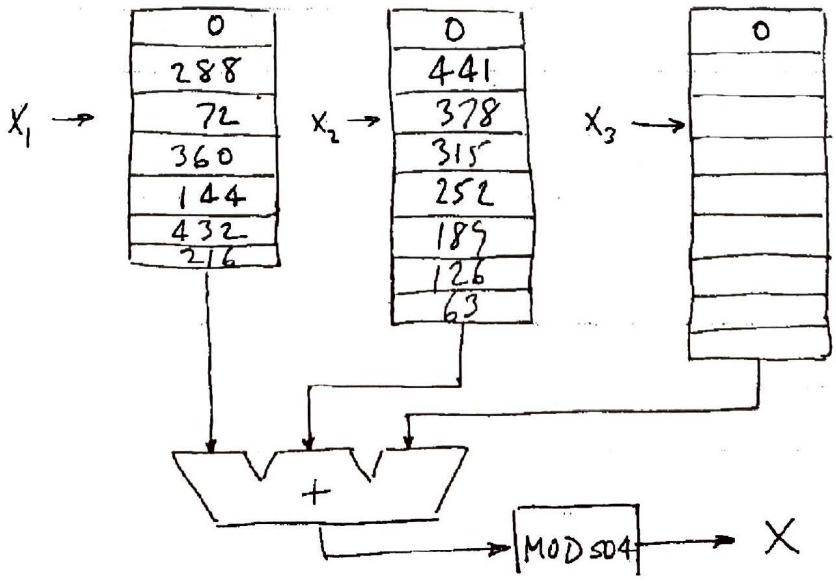
INVERSE TRANSFORMATION

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: