Cairo University
Electronics and Communications Department

Computer Arithmetic:
What is Computer Arithmetic?

Hossam A. H. Fahmy

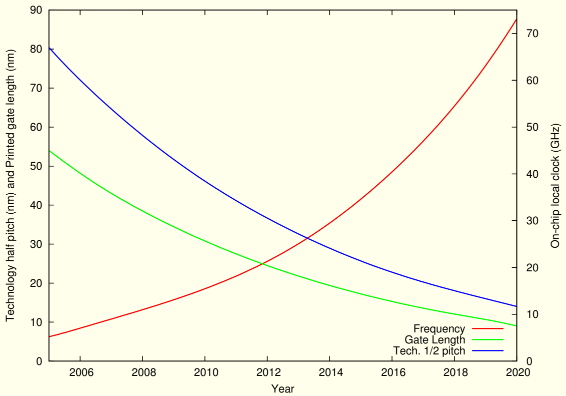
What are the arithmetic blocks? Where do they fit in digital circuits?

$x + y$ $\quad \frac{x}{y} \quad$ $\sqrt{x}$

$x * y$ $\quad \sin x \quad$ e^x

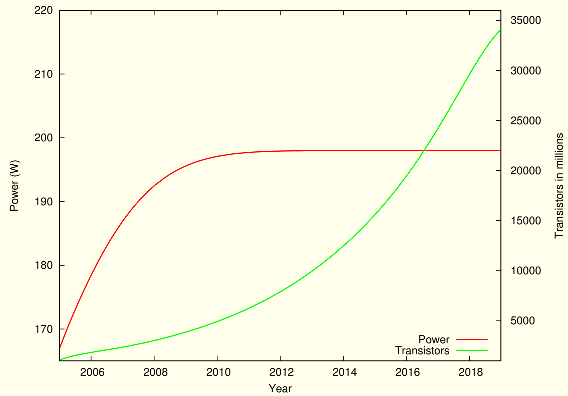
- Floating point calculations in high-end microprocessors
- Digital signal processors and graphics accelerators
- Program counters, basic ALU, branch target calculation, . . .

Looking forward



Predicted transistor gate length and maximum clock frequency trends in high performance chips. (Original data from `http://public.itrs.net`)

Looking forward



Predicted maximum allowable power and number of transistors trends in high performance chips. (Original data from `http://public.itrs.net`)

What is arithmetic on computers?

Number representations:

Integers, Floating Point, Redundant Representations, Residue Number System, Logarithmic Number System,. . .

What is the best for the specific application? Why?

Operations:

Addition, Subtraction, Multiplication, Division, Square root, exponential, log, trigonometric,. . .

Which implementation is better? How do you define better?

We always optimize according to some purpose (application) that sets the conditions of the problem.

Finitude

Computers have finite resources (datapath width, memory locations).

Example 1 In a decimal system with 5 digits after the point, can you represent 1234567/500000 = 2.469134?

The REAL issue

- Integer numbers are infinite.

  ⇒ Upper and lower bound on representable numbers and on their precision.

  ⇒ Modular arithmetic.

- Irrational numbers ($\sqrt{2}, \pi, e$) have infinitely many digits.

  ⇒ We must map from the infinite to the finite.

  ⇒ Represent all numbers with "integers".

Modular arithmetic: congruence

Two integers N and M are *congruent* modulo μ (μ is a positive integer), if and only if there exists an integer K such that

$$N - M = K\mu.$$

Hence,

$$N\mathbf{mod}_\mu \equiv M\mathbf{mod}_\mu,$$

where μ is called the modulus.

Modular arithmetic: properties

If $N' = N\mathbf{mod}_\mu$ and $M' = M\mathbf{mod}_\mu$, then

$$\begin{aligned}(N+M)\mathbf{mod}_\mu &= (N'+M')\mathbf{mod}_\mu \\ (N-M)\mathbf{mod}_\mu &= (N'-M')\mathbf{mod}_\mu \\ (N\times M)\mathbf{mod}_\mu &= (N'\times M')\mathbf{mod}_\mu\end{aligned}$$

Not for division!

Mapping the real to integers

- Approximate irrational numbers and rational numbers by some terminating sequences of digits.
- Operate on all numbers as if they were integers (provided scaling and rounding are done properly).

The integers

In our days, humans mainly use the Indo-Arabic weighted positional number system.

A number N is represented as $d_{n-1}\, d_{n-2}\, d_{n-3} \cdots d_1\, d_0$ in radix β.

$$N = \sum_{i=0}^{i=n-1} d_i \beta^i$$

How do *you* represent negative numbers? How does the machine represent them?

Negative numbers

Sign plus magnitude:

- An additional high-order symbol represents the sign.
- Natural for humans, but unnatural for a modular computer system.

Complement codes: Two types are commonly used;

Radix Complement code ($\mathbf{RC}$)
Diminished Radix Complement code ($\mathbf{DRC}$)

Complement coding is natural for computers, since no special sign symbology or computation is required.

In binary arithmetic (base $= 2$), the $\mathbf{RC}$ code is called *two's complement* and the $\mathbf{DRC}$ is called *ones' complement*.

Radix complement

If N has n digits and $\mathbf{RC}(N) = \beta^n - N$, then

$$\mathbf{RC}(N)\mathbf{mod}_{\beta^n} = (\beta^n - N)\mathbf{mod}_{\beta^n} = (-N)\mathbf{mod}_{\beta^n}$$

$P - N$ is more accurately $(P - N)\mathbf{mod}_{\beta^n}$, and

$$\begin{aligned}(P-N)\mathbf{mod}_{\beta^n} &= (P\mathbf{mod}_{\beta^n} - N\mathbf{mod}_{\beta^n})\mathbf{mod}_{\beta^n} \\ &= \left(P\mathbf{mod}_{\beta^n} + (\beta^n - N)\mathbf{mod}_{\beta^n}\right)\mathbf{mod}_{\beta^n}\end{aligned}$$

Computation of $\mathrm{RC}(N) = \beta^n - N$

1. Scan the digits of N from the least significant side till you reach the first non-zero digit. Assume this non-zero digit is at position $i + 1$.

2. The digits of $\mathbf{RC}(N)$ are given by

$$\mathbf{RC}(N)_j = \begin{cases} 0 & 0 \le j \le i \\ \beta - d_j & j = i+1 \\ \beta - 1 - d_j & i+2 \le j \le m \end{cases}$$

Also, $\mathbf{RC}(N) = \mathbf{DRC}(N) + 1 = \left(\sum_{i=0}^{i=n-1}((\beta - 1) - d_i) \times \beta^i\right) + 1$.

Which is better RC or DRC?

The calculation of $\mathbf{DRC}$ is much faster. *Why?*

However, the addition and subtraction in $\mathbf{DRC}$ needs some fixing.

(i) $P = 47$, $N = 24$:

$$\begin{array}{r} 47 \\ +24 \\ \hline 071 \end{array} \quad 71\mathbf{mod}_{100} \equiv 71\mathbf{mod}_{99} = \text{result.}$$

(ii) $P = 47$, $N = 57$:

$$\begin{array}{r} 47 \\ +57 \\ \hline 104 \\ +1 \\ \hline 05 \end{array} \quad 4\mathbf{mod}_{100} \equiv 5\mathbf{mod}_{99} = \text{result.}$$

We also have two “zeros” in $\mathbf{DRC}$.

Overflow in binary addition

Consider $P+N$ for two’s complement representations with C_{n-1} the carry-in to the sign bit and C_n the carry-out of the sign bit.

Case	P	N	Sum of Signs	C_{n-1}	C_n	Overflow	Notes
1a	Pos	Pos	0	0	0	no	
1b	Pos	Pos	0	1	0	yes	
2a	Neg	Neg	0	1	1	no	
2b	Neg	Neg	0	0	1	yes	
3	Pos	Neg	1	0	0	no	$\lvert P\rvert < \lvert N\rvert$
4	Pos	Neg	1	1	1	no	$\lvert P\rvert > \lvert N\rvert$

$$\text{OVERFLOW} = C_{n-1} \oplus C_n.$$

(Same for ones’ complement)

Shifts

A left shift multiplies the number by the radix. A right shift divides it.

Logical shift: *All bits* of a word are shifted right or left by the indicated amount with zeros filling the end bits.

Arithmetic shift: The sign bit is fixed.

For arithmetic right shift, fix the sign bit and fill the higher order bits with the value of the sign bit.

For arithmetic left shift, fix the sign bit and fill the lower order bits with zeros regardless of the sign bit.

Multiplication

In unsigned data representation, multiplying two operands, one with n bits and the other with m bits, requires that the result will be $n+m$ bits. *Can you prove it?*

In signed numbers, each n bits, the product requires only $2n-1$ bits, since the product has only one sign bit.

Exception: In the two's complement code, -2^n is representable in n bits but $(-2^n)\times(-2^n) = +2^{2n}$ is not representable in $2n-1$ bits.

Division

Division is the most difficult operation of the four basic arithmetic operations.

1. *Overflow:* Even when the dividend is n bits long and the divisor is n bits long, an overflow may occur. A special case is a zero divisor.
2. *Inaccurate results:* In most cases, dividing two numbers gives a quotient that is an approximation to the actual rational number.

By definition, $\frac{a}{b} = q + \frac{r}{b}$, $a = b \times q + r$

a	dividend	q	quotient
b	divisor	r	remainder.

If $r=0$, the division is the exact converse of multiplication. Otherwise, it is not!

Types of division

A difficulty in division is the multiplicity of valid results depending upon the sign conventions.

Modular division	$-7 \div_m 3$	$=-3,$	$r=2.$
Signed division	$-7 \div_s 3$	$=-2,$	$r=-1.$

As well as other possibilities.

If the hardware provides one and you wish another, you must make a correction.

Going far and beyond

It is possible to generalize the formula $N = \sum_{i=0}^{i=n-1} d_i \beta^i$ where β is a positive integer and $0 \le d_i < \beta$.

1. Use $N = \sum_{i=\ell}^{i=n-1} d_i \beta^i$ with $\ell \le 0$ to get a representation of fractions.
2. Use $\beta = -2$ or $\beta = -1 + j$ to get special purpose systems.
3. Have more than β possibilities for the digits to get a redundant representation. $\Rightarrow$ *Leads to carry-free addition!*
4. Do not use $N = \sum_{i=\ell}^{i=n-1} d_i \beta^i$!

Redundant representations

Assume a weighted positional signed digit system with base β where the digits d_i are such that $\alpha < d_i < \gamma$ with $\alpha < 0 < \gamma$ and $\gamma - \alpha \ge \beta + 1$.

1. At each position i, form the primary sum $p_i = x_i + y_i$ of the two operands x and y.
2. If $p_i \ge \gamma$ generate a carry $c_{i+1} = 1$. If $p_i \le \alpha$ generate a carry $c_{i+1} = -1$. Otherwise, $c_{i+1} = 0$.
3. The intermediate sum at position i is $w_i = p_i - \beta c_{i+1}$.
4. The final sum at position i is $s_i = w_i + c_i$.

Carry-free addition

Example 2 Using $\beta = 10$ and $d_i \in \{-9, \ldots, 9\}$, apply the previous rules to $202 + 189$ and $212 + 189$.

Solution: Obviously, the results are 391 and 401 but let us see the detailed operations:

2 0 2		2 1 2
+1 8 9		+1 8 9
3 8 11	$p_i \ge \lvert\gamma\rvert$?	3 9 11
0 1	c_i	1 1
3 8 1	w_i	3 $\bar{1}$ 1
3 9 1	s_i	4 0 1

Mixed radix

The elapsed time in 2 weeks, 3 days, 2 hours, 23 minutes, and 17 seconds is

Time	2 weeks	3 days	2 hours	23 minutes	17 seconds
Weights	$7 \times 24 \times 60 \times 60$	$24 \times 60 \times 60$	60×60	60	1
Value	$2 \times 7 \times 24 \times 60 \times 60 +$	$3 \times 24 \times 60 \times 60 +$	$2 \times 60 \times 60 +$	$23 \times 60 +$	$17 \times 1 = 1\,477\,397$s.

In mixed radix systems, it is important to clearly specify the possible set of digit values. In the case of time, the digit values for seconds and minutes is $\in \{0, \ldots, 59\}$ while for hours it is $\in \{0, \ldots, 23\}$ or $\{1, \ldots, 12\}$.

Summary

- Arithmetic blocks are everywhere in digital circuits.
- The finitude of computers leads to modular arithmetic.
- Negative numbers are usualy represented in $\mathbf{RC}$.
- It is possible to change the representation in order to ease the implementation of certain tasks.