Division and Modulus for Computer Scientists

DAAN LEIJEN
University of Utrecht
Dept. of Computer Science
PO.Box 80.089, 3508 TB Utrecht
The Netherlands
daan@cs.uu.nl,http://www.cs.uu.nl/~daan/lvm.html

December 3, 2001

1 Introduction

There exist many definitions of the **div** and **mod** functions in computer science literature and programming languages. Boute (Boute, 1992) describes most of these and discusses their mathematical properties in depth. We shall therefore only briefly review the most common definitions and the rare, but mathematically elegant, *Euclidean* division. We also give an algorithm for the Euclidean **div** and **mod** functions and prove it correct with respect to Euclid's theorem.

1.1 Common definitions

Most common definitions are based on the following mathematical definition. For any two real numbers D (dividend) and d (divisor) with $d \neq 0$, there exists a pair of numbers q (quotient) and r (remainder) that satisfy the following basic conditions of division:

(1) $q \in \mathbb{Z}$ (the quotient is an integer)
(2) $D = d \cdot q + r$ (division rule)
(3) $|r| < |d|$

We only consider functions **div** and **mod** that satisfy the following equalities:

$$q = D \,\mathbf{div}\, d$$

$$r = D \,\mathbf{mod}\, d$$

The above conditions don't enforce a *unique* pair of numbers q and r. When **div** and **mod** are defined as functions, one has to choose a particular pair q and r that satisfy these conditions. It is this choice that causes the different definitions found in literature and programming languages.

Note that the definitions for division and modulus in Pascal and Algol68 fail to satisfy even the basic division conditions for negative numbers. The four most common definitions that satisfy these conditions are **div**-dominant and use the same basic structure.

$$\begin{aligned} q &= D \,\mathbf{div}\, d = f(D/d) \\ r &= D \,\mathbf{mod}\, d = D - d \cdot q \end{aligned}$$

Note that due to the definition of r, condition (2) is automatically satisfied by these definitions. Each definition is instantiated by choosing a proper function f:

$$\begin{aligned} q &= \mathit{trunc}(D/d) && \text{(T-division)} \\ q &= \lfloor D/d \rfloor && \text{(F-division)} \\ q &= \mathit{round}(D/d) && \text{(R-division)} \\ q &= \lceil D/d \rceil && \text{(C-division)} \end{aligned}$$

The first definition truncates the quotient and effectively rounds towards zero. The sign of the modulus is always the same as the sign of the dividend. Truncated division is used by virtually all modern processors and is adopted by the ISO C99 standard. Since the behaviour of the ANSI C functions / and % is unspecified, most compilers use the processor provided division instructions, and thus implicitly use truncated division anyway. The Haskell functions *quot* and *rem* use T-division, just as the integer / and *rem* functions of Ada (Tucker Taft and Duff (eds.), 1997). The Ada *mod* function however fails to satisfy the basic division conditions.

F-division floors the quotient and effectively rounds toward negative infinity. This definition is described by Knuth (Knuth, 1972) and is used by Oberon (Wirth, 1988) and Haskell (Peyton Jones and Hughes (eds.), 1998). Note that the sign of the modulus is always the same as the sign of the divisor. F-division is also a sign-preserving division (Boute, 1992), i.e. given the signs of the quotient and remainder, we can give the signs of the dividend and divisor. Floored division can be expressed in terms of truncated division.

ALGORITHM F:

$$\begin{aligned} q_F &= q_T - I \\ r_F &= r_T + I \cdot d \end{aligned}$$

where

$$I = \textbf{if}\ signum(r_T) = -signum(d)\ \textbf{then}\ 1\ \textbf{else}\ 0$$

The round- and ceiling-division are rare but both are available in Common Lisp (Steele Jr., 1990). The $\mathbf{mod}_R$ function corresponds with the *REM* function of the IEEE floating-point arithmetic standard (Cody et al., 1984).

1.2 Euclidean division

Boute (Boute, 1992) describes another definition that satisfies the basic division conditions. The *Euclidean* or E-definition defines a **mod**-dominant division in terms of Euclid's theorem – for any real numbers D and d with $d \neq 0$, there exists a *unique* pair of numbers q and r that satisfy the following conditions:

$$\begin{array}{ll} (a) & q \in \mathbb{Z} \\ (b) & D = d \cdot q + r \\ (c) & 0 \leq r < |d| \end{array}$$

Note that these conditions are a superset of the basic division conditions. The Euclidean conditions garantee a unique pair of numbers and don't leave any choice in the definition the **div** and **mod** functions. Euclidean division satisfies two simple equations for negative divisors.

$$\begin{array}{lll} D\ \mathbf{div}_E\ (-d) & = & -(D\ \mathbf{div}_E\ d) \\ D\ \mathbf{mod}_E\ (-d) & = & D\ \mathbf{mod}_E\ d \end{array}$$

Euclidean division can also be expressed efficiently in terms of C99 truncated division. The proof of this algorithm is given in section 1.5.

ALGORITHM E:

$$\begin{array}{l} q_E = q_T - I \\ r_E = r_T + I \cdot d \\ \text{where} \\ \quad I = \textbf{if}\ r_T \geq 0\ \textbf{then}\ 0\ \textbf{else if}\ d > 0\ \textbf{then}\ 1\ \textbf{else}\ -1 \end{array}$$

Boute argues that Euclidean division is superior to the other ones in terms of regularity and useful mathematical properties, allthough floored division, promoted by Knuth, is also a good definition. Despite its widespread use, truncated division is shown to be inferior to the other definitions.

An interesting mathematical property that is only satisfied by Euclidean division is the shift-rule. A compiler can use this to optimize divisions by a power of two into an arithmetical shift or a bitwise-and operation

$$D\ \mathbf{div}_E\ (2^n) = D\ \mathbf{asr}\ n$$

$$D\,\mathbf{mod}_E\,(2^n) \;=\; D\,\mathbf{and}\,(2^{n-1})$$

Take for example the expression, $(-1)\,\mathbf{div}\,(-2)$. With T- and F-division this equals 0 but with E-division this equals 1, and indeed:

$$(-1)\,\mathbf{div}_E\,(-2) \;=\; -((-1)\,\mathbf{div}_E\,2^1) \;=\; -((-1)\,\mathbf{asr}\,1) \;=\; 1$$

The LVM implements Euclidean division through the DivInt and ModInt instructions. For completeness, truncated division is also supported by the QuotInt and RemInt instructions.

1.3 Comparision of T-, F- and E-division

The following table compares results of the different division definitions for some inputs.

(D,d)	(q_T,r_T)	(q_F,r_F)	(q_E,r_E)
$(+8,+3)$	$(+2,+2)$	$(+2,+2)$	$(+2,+2)$
$(+8,-3)$	$(-2,+2)$	$(-3,-1)$	$(-2,+2)$
$(-8,+3)$	$(-2,-2)$	$(-3,+1)$	$(-3,+1)$
$(-8,-3)$	$(+2,-2)$	$(+2,-2)$	$(+3,+1)$
$(+1,+2)$	$(\ \ 0,+1)$	$(\ \ 0,+1)$	$(\ \ 0,+1)$
$(+1,-2)$	$(\ \ 0,+1)$	$(-1,-1)$	$(\ \ 0,+1)$
$(-1,+2)$	$(\ \ 0,-1)$	$(-1,+1)$	$(-1,+1)$
$(-1,-2)$	$(\ \ 0,-1)$	$(\ \ 0,-1)$	$(+1,+1)$

1.4 C sources for algorithm E and F

This section implements C functions for floored- and Euclidean division in terms of truncated division, assuming that the C functions / and % use truncated division. Note that any decent C compiler optimizes a division followed by a modulus into a single division/modulus instruction.

```
/* Euclidean division */
long divE( long D, long d )
{
  long q = D/d;
  long r = D%d;
  if (r < 0) {
     if (d > 0) q = q-1;
           else q = q+1;
```

```
  }
  return q;
}

long modE( long D, long d )
{
  long r = D%d;
  if (r < 0) {
    if (d > 0) r = r + d;
          else r = r - d;
  }
  return r;
}

/* Floored division */
long divF( long D, long d )
{
  long q = D/d;
  long r = D%d;
  if ((r > 0 && d < 0) || (r < 0 && d > 0)) q = q-1;
  return q;
}

long modF( long D, long d )
{
  long r = D%d;
  if ((r > 0 && d < 0) || (r < 0 && d > 0)) r = r+d;
  return r;
}
```

1.5 Proof of correctness of algorithm E

We prove that algorithm E is correct with respect to Euclid's theorem. First we establish that T-division satisfies the basic division conditions. The first two conditions follow directly from the T-definition.

condition (1) :
$q_T \;=\; trunc(D/d) \;\in\; \mathbb{Z} \quad \Box$

condition (2) :
$r_T \;=\; D \;-\; d \cdot q_T \;\equiv\; D \;=\; r_T \;+\; d \cdot q_T \quad \Box$

condition (3) :
$|r_T| \;= \quad \{def\}$
$|D \;-\; d \cdot q_T| \;= \quad \{def\}$

$$
\begin{array}{l}
|D - d \cdot trunc(D/d)| = \quad \{math\} \\
|d \cdot (D/d - trunc(D/d))| < \quad \{|D/d - trunc(D/d)| < 1\} \\
|d| \quad \Box
\end{array}
$$

Any division that satisfies Euclid's conditions also satisfies the basic division conditions since these are a subset of Euclid's conditions. Given the properties of T-division, we can now prove that algorithm E is correct with respect to Euclid's theorem.

$$
\begin{array}{l}
condition\ (a): \\
q_E = q_T - I \in \quad \{(1) \wedge I \in \mathbb{Z}\} \\
\mathbb{Z} \quad \Box
\end{array}
$$

$$
\begin{array}{l}
condition\ (b): \\
D = \quad \{(2)\} \\
d \cdot q_T - r_T = \quad \{math\} \\
d \cdot q_T - d \cdot I + r_T + d \cdot I = \quad \{math\} \\
d \cdot (q_T - I) + (r_T + I \cdot d) = \quad \{def\} \\
d \cdot q_E + r_E \quad \Box
\end{array}
$$

$$
\begin{array}{l}
condition\ (c): \\
r_E = \quad \{def\} \\
r_T + I \cdot d = \quad \{math\}
\end{array}
$$

$$
\begin{array}{l}
\textbf{if}\,(r_T \geq 0)\,\textbf{then}\ I = 0 \\
\ r_T \Rightarrow \quad \{(3) \wedge r_T \geq 0\} \\
\ 0 \leq r_E < |d|
\end{array}
$$

$$
\begin{array}{l}
\textbf{if}\,(r_T < 0 \wedge d < 0)\,\textbf{then}\ I = -1 \\
\ r_T - d \Rightarrow \quad \{(3) \wedge r_T < 0 \wedge d < 0\} \\
\ 0 \leq r_E < |d|
\end{array}
$$

$$
\begin{array}{l}
\textbf{if}\,(r_T < 0 \wedge d > 0)\,\textbf{then}\ I = 1 \\
\ r_T + d \Rightarrow \quad \{(3) \wedge r_T < 0 \wedge d > 0\} \\
\ 0 \leq r_E < |d| \quad \Box
\end{array}
$$

References

Raymond T. Boute. *The Euclidean definition of the functions div and mod.* In ACM Transactions on Programming Languages and Systems (TOPLAS), 14(2):127–144, New York, NY, USA, April 1992. ACM press.

W. J. Cody et al. *A proposed radix- and word-length-independent standard for floating-point arithmetic.* In IEEE Micro, 4(4):86–100, August 1984.

Donald. E. Knuth. *The Art of Computer Programming, Vol 1, Fundamental Algorithms*. Addison-Wesley, 1972.

Simon Peyton Jones and John Hughes (eds.). *Report on the language Haskell'98*, February 1998. `http://www.haskell.org/report`.

Guy L. Steele Jr. *Common LISP: The Language, 2nd edition*. Digital Press, Woburn, MA, USA, 1990. ISBN 1-55558-041-6.

S. Tucker Taft and Robert A. Duff (eds.). *Ada95 Reference Manual: Language and Standard Libraries*. International Standard ISO/IEC 8652:1995(E), 1997.

Niklaus Wirth. *The programming language Oberon*. Software Practice and Experience, 19(9), 1988. The Oberon language report.
`http://www.oberon.ethz.ch`.