The Euclidean Definition of the Functions div and mod

RAYMOND T. BOUTE
University of Nijmegen

The definitions of the functions div and mod in the computer science literature and in programming languages are either similar to the Algol or Pascal definition (which is shown to be an unfortunate choice) or based on division by truncation (T-definition) or division by flooring as defined by Knuth (F-definition). The differences between various definitions that are in common usage are discussed, and an additional one is proposed, which is based on Euclid's theorem and therefore is called the *Euclidean* definition (E-definition). Its distinguishing feature is that $0 \leq D \bmod d < |d|$ irrespective of the signs of D and d. It is argued that the E- and F-definitions are superior to all other ones in regularity and useful mathematical properties and hence deserve serious consideration as the standard convention at the applications and language level. It is also shown that these definitions are the most suitable ones for describing number representation systems and the realization of arithmetic operations at the architecture and hardware level.

Categories and Subject Descriptors: B.2.m [**Arithmetic and Logic Structures**]: Miscellaneous—*arithmetic circuits, formal description*; D.3.0 [**Programming Languages**]: General—*standards*; D.3.3 [**Programming Languages**]: Language Constructs—*data types and structures*; G.1.0 [**Numerical Analysis**]: General—*computer arithmetic*; G.2.1 [**Discrete Mathematics**]: Combinatorics; J.2 [**Computer Applications**]: Physical Sciences and Engineering—*electronics, engineering, mathematics and statistics*

General Terms: Design, Standardization, Theory

Additional Key Words and Phrases: Decimation, div function, Euclid's theorem, hardware description, integer division, interpolation, mod function, number representation, predefined functions, remainder, residue, sampling

1. INTRODUCTION

The viewpoint presented here regarding the div and mod functions was originally developed in the context of formal description of computer architecture [5]. The present paper is motivated by subsequent experience with the role of these functions in various areas of applied mathematics and engineering. Indeed, the functions div and mod are very important concepts in discrete mathematics for certain problems in number theory, in computer

Author's current address: Department of Computer Science, University of Nijmegen, Toernooiveld 1, NL 6525 ED Nijmegen, The Netherlands.

science for reasoning about number representation systems, in communications engineering for a variety of issues ranging from coding to sampling and multiplexing, and so on.

Hence it is unfortunate that the definition of these functions appears to be handled rather casually in the computer science literature and in the design of programming languages, as one might infer from various poor "definitional engineering" decisions down to wrong or erroneous definitions, as in the ISO Standard for Pascal [11, 13], Algol 68 [20], and some other languages.

In this paper we clarify the differences between the various definitions, in particular those based on division by truncation (T-definition) and on division by flooring (F-definition) as defined by Knuth [14]. We also propose still another definition, which we call *Euclidean* because it is based on Euclid's theorem (E-definition). This alternative is rarely discussed in the literature, yet on closer analysis it is advantageous in terms of regularity and useful mathematical properties, both theoretically and in practical usage. The Euclidean definition usually emerged as the most straightforward choice, over a wide variety of representative application areas where we experienced the need for a div-mod function pair.

Comparison of the various definitions leads to a preference for the E- and F-definitions as the standard convention in mathematics and programming languages. Secondary definitions can always be expressed in terms of the primary ones to cater for the exceptional cases.

At the computer architecture level, the preferred definitions capture the most important number representation and computer arithmetic systems in a uniform way [5]. Although this suggests the possibility of simpler hardware realizations than for the other definitions, verifying this requires a thorough study in ALU design that is clearly beyond the scope of this paper. Difficulties such as the problem with arithmetic shifting pointed out by Steele [17] are also avoided.

Note. For positive values of dividend and divisor, all definitions of div and mod agree, as expected. Clearly, any reasons for preferring one convention over another must be derived from cases in which the definitions differ, viz., when either dividend or divisor or both are negative.

2. OVERVIEW OF DEFINITIONS

2.1 Conventions and Classification

Unless stated otherwise, D (the dividend) and d (the divisor) stand for real numbers that are arbitrary except for the fact that $d \neq 0$. There are several reasons for using the more general setting of real numbers rather than just integers.

(1) The generality comes at no extra cost whatsoever in terms of notation or conceptual understanding; it is perhaps even beneficial in this respect. The specialization of the discussion to integers is immediate, viz., by restricting D and d to values in $\mathbb{Z}$ without changing any of the formulas

presented. For integer values, $(a < b) \Leftrightarrow (a \leq b - 1)$, and in this case the inequality $|D \bmod d| < |d|$ may be replaced by the equivalent $|D \bmod d| \leq |d| - 1$ if desired.

(2) In certain application areas, the generality is needed, for example, a periodic function f with period p satisfies $fx = f(x \bmod p)$.

(3) Unnecessary restrictions in generality may cause certain issues to be overlooked, requiring revision of the definition at later stage.

As we shall see, also negative d-values arise naturally in certain applications (including number representation) and should not be ignored. Therefore, with few exceptions such as ISO Standard Pascal [11], most programming languages define both D div d and D mod d for negative d-values as well. Hence we can consider four "quadrants" according to the signs of d and D, respectively: q0 (+ +), q1 (+ −), q2 (− +), q3 (− −). The names of these quadrants appear as a clarification in later figures. The further generalization to complex numbers, for instance, in the manner proposed by McDonnell [15], falls outside the scope of this paper and can be considered at another occasion.

Presently, we require that the functions div and mod at least satisfy the following conditions:

(a) D div $d \in \mathbb{Z}$ (integer quotient must be integer),

(b) $D = d \cdot (D \text{ div } d) + D \bmod d$ (division rule),

(c) $|D \bmod d| < |d|$ ($|\,remainder\,| < |\,divisor\,|$),

in addition to the implicit basic condition that, if $D/d \in \mathbb{Z}$, then D div $d = D/d$. Definitions not satisfying these conditions are considered *wrong* because they lack the properties one expects from arithmetic and, in fact, have no useful mathematical properties at all. Condition (b) is the most important one in this respect. Note that, because of (b), the two functions must be considered as *paired*, that is, their definitions are not independent.

The above discussion leads to three classes of definitions:

(1) wrong definitions;

(2) div-dominant definitions;

(3) mod-dominant definitions.

Here "f-dominant" means that the definition of or a suitable restriction on the function f constitutes the primary definition, which together with the division rule (b) yields the definition of the other function of the div–mod pair.

2.2 Wrong Definitions

(1) *ISO Standard Pascal Definition.* The definition in the ISO Pascal Standard [11] is as follows (restricted to integers D and d, $d \neq 0$).

(i) For div: if $|D| < |d|$, then D div $d = 0$; otherwise D div d is defined by

$$|D| - |d| < |(D \text{ div } d) \cdot d| \leq |D|$$

together with $\text{sign}(D \text{ div } d) = \text{sign}(D/d)$.

(ii) For mod: if $d \leq 0$, then D mod d is an error; otherwise

$$D \bmod d = D - k \cdot d$$

for integral k such that $0 \leq D \bmod d < d$.

The ISO reference explicitly notes that the division rule is satisfied only for $D \geq 0$ and $d > 0$. For instance, if $D = -3$ and $d = 7$, then D div $d = 0$ and D mod $d = 4$ (corresponding to $k = -1$). Hence $(D \text{ div } d)d + D \bmod d = 4$, which violates the division rule.

(2) *Variant Pascal Definition.* The ISO Pascal User Manual [13, p. 17] defines the following variant (restricted to integers D and d, $d \neq 0$):

(i) For div: divide and truncate (for a precise definition, see Section 2.3).

(ii) For mod:

let $rem = D - (D \text{ div } d) \cdot d$;
if $rem < 0$ **then** $D \bmod d = rem + d$
otherwise $D \bmod d = rem$

The conditional expression deliberately violates the division rule if $rem < 0$. In fact, this definition coincides with the ISO Standard [11] for the range of d-values specified, viz., $d \neq 0$ for D div d and $d > 0$ for D mod d. However, for negative d it contains an additional (but most likely not deliberate) error that causes violation of condition (c). For instance, if $D = -3$ and $d = -7$, then D div $d = 0$, $rem = -3$, D mod $d = -3 - 7 = -10$. In the ISO Pascal User Manual [13, p. 167], this is avoided by proclaiming D mod d to be an error if $d \leq 0$. This restriction makes the definition fully agree with the ISO Standard [11], but of course the violation of the division rule remains.

(3) *Algol and Ada Definitions.* The aforementioned violation of condition (c) in the variant Pascal definition can be avoided by a less drastic measure than prohibiting negative d-values. For instance, the definition given in the Algol 68 report [20, p. 132], can be obtained from the variant Pascal definitions by using D mod $d = rem + |d|$ for the **then** part, which also covers the negative d-values. Due to the presence of the conditional, the Algol definition still violates condition (b) and hence belongs to the category of wrong definitions.

The Ada Reference Manual [1] complements integer division (/) with two functions **mod** and **rem**. The first of these violates the division rule, the second one does not, and its definition is given below.

Wrong definitions are not considered further.

2.3 Div-Dominant Definitions

(1) *Division with Truncation (T-definition).* Let us correct the ISO Pascal or Algol 68 definition by simply omitting the conditional statement. This yields

(a) D div $d = \text{trunc}\,(D/d)$,

(b) D mod $d = D - d \cdot (D \text{ div } d)$,

clearly satisfying the correspondingly labeled conditions, and also condition (c) in Section 2.1 because $|D/d - \text{trunc}(D/d)| < 1$. Restricted to integers, this corresponds to the definitions in Ada [1] for integer division (/) paired with the aforementioned **rem** function.

Properties

$$\begin{aligned} D \text{ div } (-d) &= -(D \text{ div } d) \\ D \bmod (-d) &= D \bmod d \\ (-D) \text{ div } d &= -(D \text{ div } d) \\ (-D) \bmod d &= -(D \bmod d). \end{aligned}$$

An illustration is given in Figure 1.

(2) *Division with Flooring (F-definition).* Knuth's definition [14] is as follows:

(a) $D \text{ div } d = \lfloor D/d \rfloor$;
(b) $D \bmod d = D - d \cdot (D \text{ div } d)$.

It clearly satisfies the correspondingly labeled conditions, and also condition (c) in Section 2.1 because $0 \le D/d - \lfloor D/d \rfloor < 1$. This definition i s used in a number of programming languages such as Miranda [19].

Properties

$$\begin{aligned} D \text{ div } (-d) &= -(D \text{ div } d) - I \\ D \bmod (-d) &= D \bmod d - d \cdot I \\ (-D) \text{ div } d &= -(D \text{ div } d) - I \\ (-D) \bmod d &= -(D \bmod d) + d \cdot I \end{aligned}$$

where $I =$ **if** $D/d \in \mathbb{Z}$ **then** 0 **else** 1.

An illustration is given in Figure 2. At the points of discontinuity, dots indicate which value is meant.

(3) *Other Definitions.* Common Lisp [18] provides four div-dominant definitions, thus supplementing the T- and F-definitions with two additional ones. The pairing of div and mod and the role of the division rule are made explicit by defining all functions as mapping (D, d) pairs onto (q, r) pairs:

$$f(D, d) = (q, r) \quad \text{satisfying} \quad r = D - d \cdot q.$$

The four definitions can be summarized as follows:

(a) $f =$ floor: $q = \lfloor D/d \rfloor$ (F-definition);
(b) $f =$ truncate: $q = \text{trunc}(D/d)$ (T-definition);
(c) $f =$ ceiling: $q = \lceil D/d \rceil$;
(d) $f =$ round: $q = D/d$ rounded to nearest integer (if D/d is equally distant to two integers, the even one is chosen).

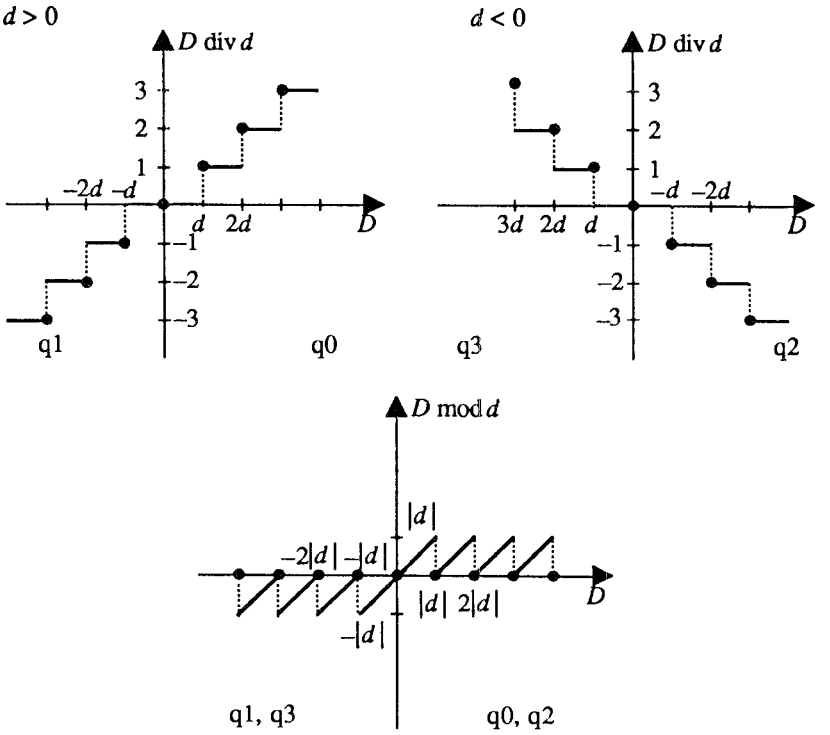

Fig. 1. The functions div and mod based on truncation.

Common Lisp also provides two functions mod and rem mapping the (D, d) pair to the *second* component of floor (D, d) and truncate (D, d), respectively. Of course, in Lisp function application is written fDd).

The second component of round (D, d) corresponds to the image of (D, d) under the REM function defined in the IEEE Floating-Point Arithmetic Standard [7].

2.4 Mod-Dominant Definition Based on Euclid's Theorem

Let us recall Euclid's theorem.

THEOREM. *For any real numbers D and d with* $d \neq 0$, *there exists a unique pair of numbers q, r satisfying the following conditions*:

(a) $q \in \mathbb{Z}$;
(b) $D = d \cdot q + r$;
(c) $0 \leq r < |d|$.

For the particular case of integers, the conditions (a)–(c) correspond to part of the axioms for Euclidean rings [9].

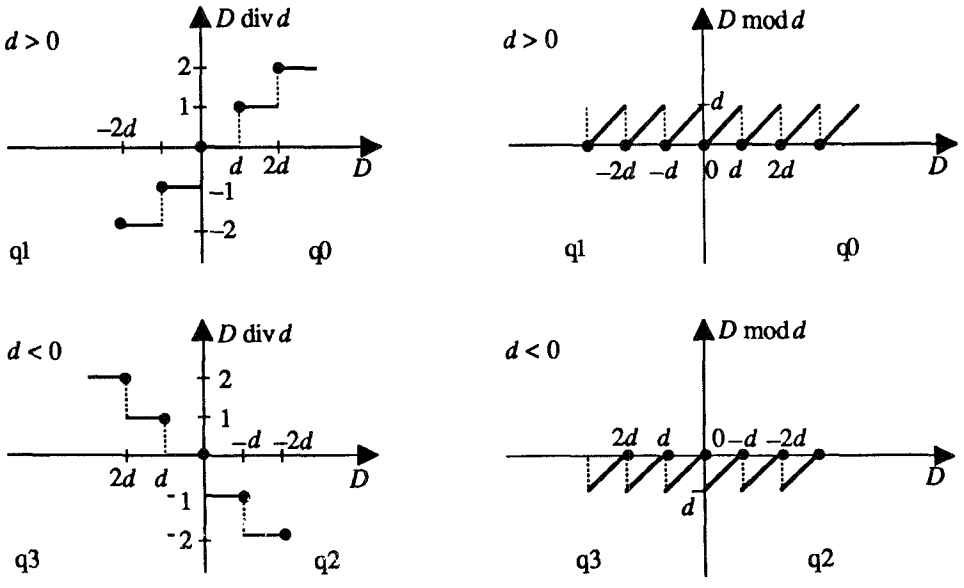

Fig. 2. The functions div and mod based on flooring.

With the notational conventions of the theorem, we define D div $d = q$ and D mod $d = r$. Clearly the correspondingly labeled conditions are satisfied.

Properties

$$D \text{ div } (-d) = -(D \text{ div } d)$$

$$D \text{ mod } (-d) = D \text{ mod } d$$

$$(-D) \text{ div } d = -(D \text{ div } d) - I \cdot J$$

$$(-D) \text{ mod } d = -(D \text{ mod } d) + d \cdot I \cdot J$$

where $J = \text{sign } d$ and I is as defined earlier.

An illustration is given in Figure 3.

3. DISCUSSION

With the earlier convention for the quadrants q0–q3, we can express the correspondences between the definitions as follows. In q0, all definitions agree; in q1, F and E agree; in q2, E and T agree, whereas in q3, T and F agree.

The following discussion compares the various definitions in view of their usage in numerical and discrete mathematics, in computer science, and in programming languages. A synoptic table summarizing the results is presented at the end of this discussion (see Table I).

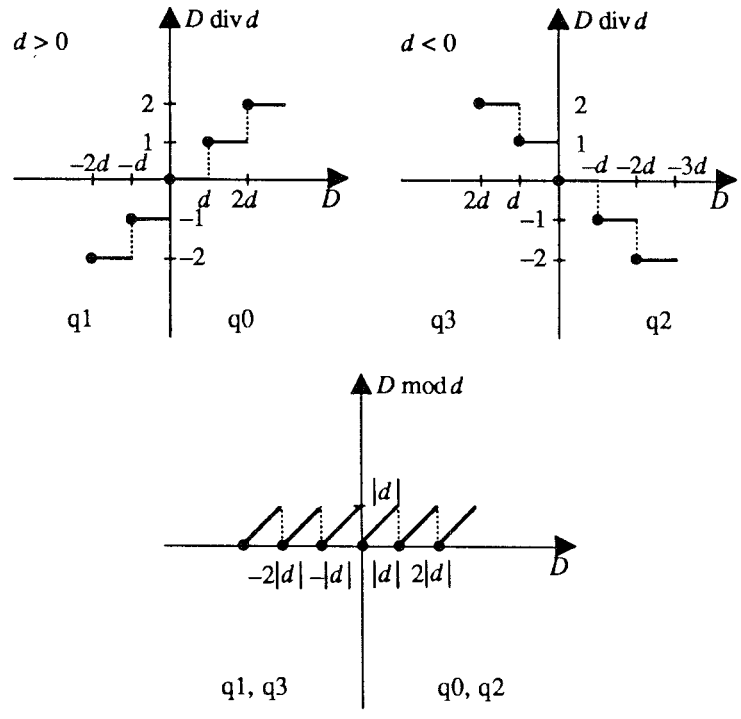

Fig. 3. The Euclidean definitions of div and mod.

Table I. Comparison of Definitions

Criterion	Definition		
	T	F	E
Regularity	1	2	3
Periodicity		√	√
Preservation of information (sign of d)		√	
Nonnegative unique representative			√
Unsigned integer representation	√	√	√
Radix complement representation		√	√
Negative radix representation		2	3
Repeated shift, 2s complement		√	√
Repeated shift, 1s complement	√		

Note. Numbers represent figures of merit, with 1 indicating the poorest and 3 indicating the best.

3.1 Mathematical Considerations

(1) An important relation when dealing with numbers is the so-called equivalence modulo d, denoted by $\equiv_d$ and defined as follows:

$$x \equiv_d y \Leftrightarrow \exists k \in \mathbb{Z}.(x - y = k \cdot d).$$

This equivalence relation (which is, in fact, also a congruence with respect to addition and multiplication) induces many useful group-theoretic and other

algebraic properties [9]. To "inherit" these properties, the mod function should satisfy

$$x \bmod d = y \bmod d \Leftrightarrow x \equiv_d y.$$

This *periodicity* criterion is satisfied by the F- and E-definitions only. For the T-definition, the "platform" of the div d function in the interval $(-|d|, +|d|)$ destroys not only regularity but also periodicity.

For comparing the F- and E-definitions, we consider the mod d function as mapping any number onto a unique representative in its equivalence class modulo d. Unless this unique representative is zero, it has the same sign as d with the F-definition and is always positive with the E-definition. Hence the F-definition is slightly more *information-preserving* w.r.t. the sign of d. On the other hand, in view of regularity one might prefer not letting the sign of the representative depend on the sign of d. In fact, the definition of $\equiv_d$ suggests the opposite; since the sign of k in the equality $x - y = k \cdot d$ has no impact on the sign of the unique representative, there appears little reason for letting the sign of d influence the choice (although the roles of k and d are not really symmetrical: in view of the variable bindings, d is free and k is existentially quantified). This *nonnegative unique representative* criterion leads to a preference for definition E.

(2) The Euclidean definition coincides with the definition in algebra that is generalizable to Euclidean rings other than integers (e.g., polynomials [9]). It is also clearly the more regular of the three.

Of course, symmetry and regularity arguments such as the above are rather speculative by themselves, and it remains to be seen to what extent they lead to more orthogonal formulations and to simpler or more general ways of reasoning and of proving theorems in actual applications. These issues are considered next.

3.2 Application-Oriented Considerations

As a result of their mathematical properties, the E- and F-definitions of div and mod can be expected to be the more suitable ones in any situation that involves the number-theoretic aspects of the function pair. This is confirmed by experience with formal system description in various technical applications, for instance, raster scan display generation [2], time division multiplexing [4], and other issues in coding, signal processing, and communications. Reproducing these detailed examples here is beyond the scope of this paper. Hence we provide only a small illustration that conveys the flavor of the paired usage of div and mod in some of these applications. This example is derived from a formalization of the interpolation and decimation functions in the signal processing language Silage [10].

Example. Decimation is the conversion of a synchronous serial stream of data samples with rate f into another synchronous serial stream where the original data samples are packed into n-tuples occurring at rate f/n. This can be seen as serial-to-parallel conversion. Interpolation is the inverse process. It is convenient to introduce two auxiliary definitions: $I_T = \{n \cdot T \mid n$

$\in \mathbb{N}\}$ for the set of sampling instants spaced T apart in time, and $I_T \to A$ for the family of signals sampled at rate I_T and taking instantaneous values in the set A. We also use the notational conventions of Funmath (functional mathematics) [3, 6], which by design is sufficiently close to commonly used mathematical notation to be largely self-explanatory. A sentence of the form

def $f \in A \to B$ **with** $f\,a = expr$

defines a function f with domain A, codomain B, and defining equation $fa = expr$. With these conventions, decimation and interpolation are formally described by the definitions

def $decim_n \in (I_{T/n} \to A) \to (I_T \to A^n)$ **with**
$decim_n\,x\,t\,i = x\,(t + i \cdot T/n)$

def $inter_n \in (I_T \to A^n) \to (I_{T/n} \to A)$ **with**
$inter_n\,x\,t = x\,((t \text{ div } T) \cdot T)\,((t \text{ mod } T) \cdot n/T)$

Here A^n denotes the set of n-tuples over A, whereas for the higher order functions we use the standard convention that $x\,a\,b$ stands for $(x\,a)\,b$. It can be shown that, with the E- and F-definitions of div and mod, the composition $decim_n \circ inter_n$ is the identity function over $I_T \to A^n$, as one is entitled to expect.

As pointed out by one of the referees, the choice of the definition for the div–mod function pair may provide a good mathematical framework for illuminating the choices of protocol in data transmission that are (still) a matter of debate in the so-called endian wars [8, 12]. The issue under discussion is whether serialized data should be transmitted most-significant-bit first ("big-endian") or last ("little-endian"), with similar questions arising regarding the byte and word order.

Thus far we have assumed positive T. At first sight, the F-definition covers negative T-values as well, reflecting reversal of the order within a tuple, but also introducing a time shift over one period. It appears that neither the F- nor the E-definition can describe order reversal adequately without case distinction. Exploring this interesting issue is left as an exercise.

In the following subsection we provide a few examples that are closer to home from the computer scientist's point of view, namely, in the area of computer arithmetic.

3.3 Number Representation and Computer Arithmetic

This subsection is intended to show that, for describing computer arithmetic and hardware realization, the Euclidean definition is the most natural choice. The three definitions (T, F, E) are suitable for unsigned integer representation (quadrant q0), the T-definition drops off for radix complement (e.g., 2's complement) and negative radix representation (quadrants q1, q2, q3), whereas the F-definition is slightly less elegant than the E-definition for

negative radix representation (quadrants q2 and q3). The E-definition covers all cases in the most uniform way.

Our mathematical conventions are the following: we write A^* for the set of strings (finite sequences) over A, including the empty sequence ϵ; we write (x, y, z) for the sequence containing the elements x, y, z; we use the symbol # for the *length* function, a colon for the prefix *cons* function, a double colon for the postfix *cons*, and the symbol ++ for *concatenation*.

When dealing with number representations, we write b for the base (or radix) which may be any integer, B for the set $\{m \in \mathbb{N} \mid m < |b|\}$, and "$xyz$" as an abbreviation for (x, y, z). To avoid an explicit function for expressing the trivial mapping between values in B (numbers) and their representations (single symbols, e.g., 0, 1, 2, . . . , 9, A, B, . . .), we identify values in the range B with the corresponding symbols. Also for notational simplicity, in concrete numerical examples integer values are denoted by their standard decimal representation; for example, 9365 denotes a value (number) and "9365" an (uninterpreted) string of digits. These conventions result in minimal nomenclature without causing ambiguity. For the sake of generality, we use variable word-length representations as the starting basis for our discussion.

Remark. Endian difficulties also arise in number representation. Arabic writing is from right to left, also for numbers, which implies that the least significant digit is written first. This is the most reasonable convention, because the weight of a digit is determined at the instant it is written down (i.e., does not depend on digits written subsequently). When adopting the Arabic number system in Western writing (left to right), the spatial ordering was kept unchanged, and hence the most significant digit is written first. Since elements in sequences are usually indexed in the order of writing (e.g., if s = "example," then $s\ 0$ = e, $s\ 1$ = x, etc.), the most significant digit in a sequence representing a number has the lowest index. This is unfortunate, since one would prefer writing $b^i \cdot s\ i$ rather than $b^{\#s-1-i} \cdot s\ i$ for the value associated with digit $s\ i$. There is no satisfactory way out of this situation, apart from conforming to the more consistent Arabic convention of writing the least significant digit first.

(1) *Variable-Word-Length Number Representation Systems.* A *denotation* function interprets digit strings as numbers; a *representation* function represents numbers by digit strings. More specifically,

(a) For the unsigned integer system ($b > 0$):

def $du \in B^* \to \mathbb{N}$ **with**
$du\ \epsilon = 0$
$du\ (x::a) = b \cdot (du\ x) + a$

def $ru \in \mathbb{N} \to B^*$ **with**
$ru\ 0 = \epsilon$
$ru\ m = ru(m \text{ div } b)::(m \text{ mod } b)$

(b) For signed integers in b's complement ($b > 0$):

def $ds \in C \to \mathbb{Z}$ **with**
$ds\ \langle b-1\rangle = -1$
$ds\ \langle 0\rangle = 0$
$ds\ (x::a) = b \cdot (ds\, x) + a$

def $rs \in \mathbb{Z} \to C$ **with**
$rs\ (-1) = \langle b-1\rangle$
$rs\ 0 = \langle 0\rangle$
$rs\, m = rs\ (m \text{ div } b)::(m \text{ mod } b)$

Here $\langle\ \rangle \in A \to A^1$ maps elements of A into sequences of length 1 over A. Also, C denotes the set of *canonical* representations [5]:

$$C = \{a{:}x \mid (a = 0 \vee a = b-1) \wedge x \in B^* \wedge \forall a' \in B,\ y \in B^*.(x = a'{:}y \Rightarrow a' \neq a)\}$$

that is, the set of strings starting with the symbols 0 or $b-1$, and in which the two highest order digits are unequal.

(c) For signed integers in negative radix, positive digit (NRPD) representation ($b < 0$):

def $dnp \in B^* \to \mathbb{Z}$ **with**
$dnp\ \epsilon = 0$
$dnp\ (x::a) = b \cdot (dnp\, x) + a$

def $rnp \in \mathbb{Z} \to B^*$ **with**
$rnp\ 0 = \epsilon$
$rnp\, m = rnp(m \text{ div } b)::(m \text{ mod } b)$

Notice the following:

(i) Obviously, the *correctness criterion* for the denotation and representation functions is that they must be each other's inverse (up to suitable domain restrictions or equivalence relations on representations) for every case.

(ii) In the preceding definitions, the last lines are formally the same in every case: the differences reside only in the boundary conditions. This results in a simple, uniform treatment.

This uniformity is possible only with the E-definition. The F-definition can be used for describing the negative radix negative digit (NRND) system, but the corresponding denotation and representation functions *dnn* and *rnn* require a minus sign in the third line of the definitions:

$$dnn(x::a) = b \cdot (dnn\, x) - a; \qquad rnn\, m = rnn(m \text{ div } b) :: -(m \text{ mod } b)$$

with div and mod according to the F-definition. The presence of the minus sign slightly damages uniformity, but this is only a minor issue of style: uniformity can be restored by factorizing out the minus sign, using a separate function expressing the mapping between (single) *symbols* and the *numbers* in the range $(b+1)..0$ represented by them.

Notice that the NRPD (*dnp*, *rnp*) and NRND (*dnn*, *rnn*) systems have the following very simple relationship to each other:

$$dnn\,x = -(dnp\,x); \qquad rnn\,m = rnp(-m)$$

The following numerical examples illustrate the three cases in a more concrete way. Successive digits of the representation are derived in tabular form with numbers and digits arranged in the following pattern:

m	m div b	$\cdots$	(numbers)
	m mod b	$\cdots$	(digits)

(a) For the unsigned integer system ($b = 10$):

384	38	3	0	—
	4	8	3	—

Hence, *ru* 384 = "384."

(b) For the signed integer system ($b = 10$):

384	38	3	0	—	−384	−39	−4	−1	—
	4	8	3	0		6	1	6	9

Hence, *rs* 384 = "0384," and *rs*(−384) = "9616."

(c) For the negative radix system ($b = -10$):

384	−38	4	0	—	−384	39	−3	1	0	—
	4	2	4	—		6	9	7	1	—

Hence, *rnp* 384 = "424" and *rnp* (−384) = "1796."

A final example illustrates the usage of the F-definition in computing *rnn*:

384	−39	3	−1	0	—	−384	38	−4	0
	−6	−9	−7	−1	—		−4	−2	−4

Hence, *rnn* 384 = "1796" and *rnn* (−384) = "424."

(2) *Intermezzo: Relating Variable to Fixed Word Length.* The concepts, definitions, and theorems that are useful for the general and uniform mathematical treatment of number systems are discussed in Boute [5]. It is shown in particular that, for b's complement representation, the *ds* functions can be extended to interpret all strings in B^+ rather than just the canonical strings in C. For instance, if we make the additional design decision to distribute the digit strings evenly over nonnegative and negative number s (which requires b even), the redefined *ds* function and the matching *rs* function are

def $ds \in B^+ \to \mathbb{Z}$ **with**
$ds\langle d\rangle = (0 \le d < b/2)?d \mid d - b$
$ds(x::a) = b \cdot (ds\,x) + a$

def $rs \in \mathbb{Z} \to B^+$ **with**
$rs\,m = (0 \le m < b/2)?\langle m\rangle \mid (-b/2 \le m < 0)?\langle m + b\rangle$
$\mid rs(m \text{ div } b)::(m \text{ mod } b)$

Here $c?a \mid b$ should be seen as a compact notation for **if** c **then** a **else** b.

Examples.

m	-500	-499	-384	-6	-5	-1	0	1	4	5	38	61	499
$rs\,m$	"500"	"501"	"616"	"94"	"5"	"9"	"0"	"1"	"4"	"05"	"38"	"061"	"499"

Notice that rs is not surjective because it maps every integer onto its *shortest* representation, whereas ds is not injective and maps *any* string in B^+ onto the integer it represents. Of course, $ds \circ rs = id_{\mathbb{Z}}$.

If we define an equivalence relation $\equiv$ on B^+ by

$$x \equiv y \Leftrightarrow ds\,x = ds\,y$$

it is easy to see that, in general, equivalence is preserved by removing leading 0's, provided the leading digit d of the remaining string satisfies $0 \le d < b/2$, or by removing leading digits with value $b - 1$ as long as the leading digit d of the remaining string satisfies $b/2 \le d < b$. The first case is obvious; the second case is explained by the fact that the digit in the most significant position of an n-digit word is interpreted as $(d - b) \cdot b^{n-1}$ which equals $d \cdot b^{n-1} + ((b - 1) - b) \cdot b^n$. For instance, "00384" ≡ "0384" ≡ "384" and "99843" ≡ "9843" ≡ "843." The reverse transformation (padding with leading digits) should now be obvious.

For the unsigned integer and the negative radix system, the equivalence-preserving transformations simply consist in adding or removing zeros.

These transformations are the key to fitting a variable word length representation into a fixed word length representation (provided the length of the shortest representation does not exceed the word length, of course).

(3) *On-Fixed-Word-Length Computer Arithmetic.* We conclude by showing that the Euclidean div and mod functions are also the ones that most adequately express the meaning of the arithmetic operations in a computer. Of course, for quadrants $q0$ and $q1$ it shares these advantages with the F-definition.

In Boute [5] it is shown that b's complement representation with fixed word length n can be defined in terms of unsigned integer fixed word length representation as follows:

def $ru_n \in 0..(b^n - 1) \to B^n$ **with** $ru_n\,m\,k = (m \text{ div } b^{n-1-k}) \bmod b$
def $rs_n \in -b^n/2..(b^n/2 - 1) \to B^n$ **with** $rs_n\,m = ru_n(m \bmod b^n)$

As a consequence of the second definition, performing the *add operation* on b's complement *representation* using the digit manipulations corresponding to unsigned integer representation amounts to *addition modulo* b^n on the represented *numbers*. This is discussed extensively in Boute [5].

Here we briefly concentrate on the issue raised in Steele [17] of integer division in conjunction with the arithmetic shift right (asr) operation when dealing with 2's complement representation. The criticism on the asr in this reference is based on the assumption that the div function is defined as divide and truncate. With this assumption, -1 div $2 = 0$ and the asr yield the wrong result.

However, it follows from the observations thus far, and from Table I, that the T-definition is far inferior to the other ones, and hence is not the one we wish to implement anyway. We now show that asr realizes the div operation on 2's complement representation *correctly* if the Euclidean definition is chosen.

The relationship between the arithmetic shift right and the functions div and mod follows for even radix b from the properties [5]:

$$ds(x +\!\!+ y) \text{ div } b^{\#y} = ds\,x \qquad \text{provided } x \neq \epsilon$$
$$ds(x +\!\!+ y) \text{ mod } b^{\#y} = du\,y \qquad \text{(warning: } du\text{)}$$

Arithmetic shift right implements the transition from $x +\!\!+ y$ to x in fixed word length by shifting y out to the right and padding x on the left to fit the original word length. Expressed formally, for word length n and even r adix b:

def $asr_n \in B^n \to B^n$ **with**
$asr_n\,xk = k = 0?(0 \le x0 < b/2?0 \mid b - 1) \mid x(k-1)$

This function has the property that, for an m-fold shift of an n-digit word x,

$$(ds\,x) \text{ div } b^m = ds(asr_n^m x)$$

where asr_n^m denotes the m-fold composition of asr_n with itself.

Example. (with $b = 10$)

x	"413"	"041"	"004"	"587"	"958"	"995"
$asr_3\,x$	"041"	"004"	"000"	"958"	"995"	"999"
$ds_3\,x$	413	41	4	−413	− 42	− 5
$ds_3(asr_3\,x)$	41	4	0	− 42	− 5	− 1

Notice the following:

(i) The "shifted-out" part in an m-fold shift represents $(ds_n\,x)$ mod b^m. This can be obtained as part of the result of the "combined" arithmetic shift right *asrc*:

def $asrc_n \in \mathbb{N} \ni m \to B^n \to (B^n \times B^m)$ **with**
$asrc_n\,m\,x = (q, r)$ **where**
$q\,k = k < m?(0 \le x0 < b/2?0 \mid b - 1) \mid x(k - m)$
$r\,k = x(k + n - m)$

The first argument is the size of the shift. This function has the property that, if $asrc_n\,m\ x = (q, r)$, then $ds\,q = (ds\,x)$ div b^m and $du\,r = (ds\,x)$ mod b^m.

(ii) The special case of 2s complement ($b = 2$) corresponds to the way *asr* is implemented in 2s complement ALUs.

(iii) In 1's complement representation, the *asr* corresponds to div 2 according to the T-definition. This may be another argument against 1's comple-

ment representation, in addition to the undesirability of representations of even numbers ending with a 1 instead of a 0.

3.4 Precision Considerations

We consider the div and mod functions primarily as mathematical functions at the applications level and defined for "genuine," that is, mathematical integer or real numbers. However, when envisaging their implementation as primitive functions in some programming language involving floating-point approximations to real numbers, precision issues arise.

In the author's opinion, these issues should *not* be the primary factor in the choice of the definition of the div and mod functions: this definition should be mainly governed by its usefulness in applications, whereas it is the implementor's task to engineer satisfactory realizations. Yet, for the sake of completeness, we devote a few remarks to precision aspects, referring to the literature for more detailed discussions.

The IEEE Standard for Floating-Point Arithmetic [7] defines the REM function as mentioned earlier in the context of Common Lisp. It is defined explicitly to satisfy the division rule $r = D - d \cdot q$ and has the property that r is exactly representable in the precision of D and d, even if q is not exactly representable in the precision of D, d, and r (i.e., when the significand of q requires more digits than specified or the exponent of q falls outside the range). The example given in Cody [7] is the following:

> precision: $b = 10$, $p = 7$ (digits in significand), $E_{\min} = -99$, $E_{\max} = +99$
> numbers: $D = 10^{75}$, $d = 3 \times 10^{-75}$

Then, with the given definition for REM, $q = \lfloor D/d \rfloor$ yields $q = \lfloor 1/3 \times 10^{150} \rfloor$, yet D REM $d = D - d \cdot \lfloor D/d \rfloor = 10^{-75}$.

This property is inherited by all definitions of mod that satisfy the division rule and whose range is not larger than the range of REM because, as explained in Cody [7], such mod functions can always be computed from the REM function without loss in precision. The range of REM is $[-|d|/2, +|d|/2]$, and the range of the Euclidean mod, being $[0, |d|)$, is never larger.

Some caution is required with language implementations performing "hidden" rounding. As pointed out by McDonnell [16], in certain versions of APL the so-called comparison tolerance (CT), which in principle is meant as the user-specified minimum distance between numbers to yield inequality in comparisons, is also used to round ("fuzz") noninteger arguments of certain functions to the nearest integer before applying the function. This is also the case for the floor and ceiling functions, for instance, if $CT = 10^{-13}$ and $x = 2 - 10^{-14}$, then $\lfloor x$ returns 2. Moreover, in these implementations the "residue" $r = D - d \cdot \lfloor D/d \rfloor$ is computed *without* rounding of D/d to the nearest integer prior to computing $\lfloor D/d \rfloor$. This causes a violation of the div ision rule. In McDonnell [16] it is explained how to introduce fuzz on the resi due in such a fashion that the division rule is obeyed.

3.5 Comparison of Definitions

In Table I, checkmarks indicate which criteria are satisfied by the various definitions. Whenever this is a matter of degree, relative figures of merit are given (1 = poorest, 3 = best). This table is not intended to be complete, since

other criteria could be imagined and since for certain problems the elegance of the solution may depend as much on the style of formulation as on the chosen definition of div and mod.

3.6 Implementation Considerations

There still appears to exist an ongoing debate between two schools of thought regarding the direction for the engineering tradeoffs involved in programming language design when choosing definitions for primitive functions.

The first school adheres to the principle that programming languages should support those primitive function definitions that are most convenient for programming and problem solving. With this principle, it is up to the compiler designer to bridge to gap (if any) between the definitions and the machine, and to the computer architecture designer to reduce (or annihilate) such a gap, both in the most economical way (regarding time and resources).

The other school proclaims that the tradeoffs in programming language design must be made in the direction of catering for the properties of machines.

Whereas this author in general favors the first viewpoint, it is not clear whether in this particular case the issue even arises, in the sense that the proposed E-definition need not be more costly to implement than the other ones. The very origin of this definition, viz., in architecture description, suggests the opposite, but elaborating this topic is more appropriate in a paper on computer architecture.

In any case, it appears evident that any of the considered div and mod functions can be implemented on any machine on a "pay for what you use" basis, because the differences arise only for negative D- and d-values. Hence the simple situation of positive D- and d-values can be handled identically and hence equally economically for all definitions. The next simple situation (positive d, arbitrary D) is handled in the same way for the E- and F-definitions, whereas, as we have seen, the T-definition is of lesser interest. The most "difficult" situation (negative d) only requires a sign check on d to handle the difference between the E- and F-definitions.

4. CONCLUSION

It is remarkable that, in computer science, the Euclidean definition is used very infrequently, if at all, whereas it is perhaps the most convenient one in terms of mathematical properties and practical applications (only the F-definition being comparable). Precisely because this definition appears not to be well known, its discussion may be of interest even to readers who find our arguments insufficiently conclusive for choosing the Euclidean definition as a standard for programming languages. More specifically, it is hoped that this paper will at least broaden the basis for making appropriate "definitional engineering" decisions regarding the div and mod functions in the future.

ACKNOWLEDGMENT

The author wishes to thank the referees for their useful suggestions and for providing additional references to interesting related work. In particular, one

of the referees has indicated many other opportunities and areas to be investigated further, from which it is clear that this topic is still far from exhausted.

REFERENCES

1. American National Standards Institute. ANSI/MIL-Std 1815 A, Reference Manual for the Ada Programming Language. Alsys, France, Jan. 1983.
2. BOUTE, R. Formal description of digital systems. In *Methodologies for Computer System Design*. W. K. Giloi and B. D. Shriver, Eds., North-Holland, Amsterdam, 1985, pp. 291-306.
3. BOUTE, R. Funmath: Towards a general formalism for system description in engineering applications. In *Advances in Electrical Engineering Software*, P. P. Silvester, Ed., Springer-Verlag, New York, 1990, pp. 215-226.
4. BOUTE, R. On the equivalence of time-division and frequency-division multiplexing. *IEEE Trans. Commun. COM-33*, 1 (Jan. 1985), 97-99.
5 BOUTE, R. Representational and denotational semantics of digital systems. *IEEE Trans. Comput. 38*, 7 (July 1989), 986-999.
6. BOUTE, R. Syntactic and semantic aspects of formal system description. *Microprocessing and Microprogramming 27* (1989), 155-162.
7. CODY, W. J., ET AL.. A proposed radix- and word-length-independent standard for floating-point arithmetic. *IEEE Micro 4*, 4 (Aug. 1984), 86-100.
8. COHEN, D. On holy wars and a plea for peace. *IEEE Comput. 14*, 10 (Oct. 1981), 48-54.
9 HERSTEIN, I. N. *Topics in Algebra*. Xerox College Publ., Lexington, Mass., 1964.
10. HILFINGER, P. N. Silage Reference Manual, Rev. 1.3. University of California, Berkeley, Dec. 1987.
11. ISO/IEC 7185, Information Technology—Programming Languages—Pascal. International Standard, 2nd ed., ISO/IEC 7185, 1990 (E).
12. JAMES, D. V. Multiplexed buses: The endian wars continue," *IEEE Micro 10*, 3 (June 1990), 9-21.
13. JENSEN, K., AND WIRTH, N. *Pascal User Manual and Report*. 3rd ed., Springer-Verlag, New York, 1985.
14. KNUTH, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. 2nd ed. Addison-Wesley, Reading, Mass., 1972.
15. MCDONNELL, E. E Complex floor. In *APL Congress 73*, P. Gjerløv et al., Eds., North-Holland, Amsterdam, 1973, pp. 299-305.
16. MCDONNELL, E. E. Fuzzy residue. In *Proceedings of the APL'79 Conference*. Rochester, N.Y., June 1979, pp 42-46.
17. STEELE, G. L., JR., Arithmetic shifting considered harmful. *SIGPLAN Not. 12*, 11 (Nov. 1977), 61-68.
18. STEELE, G. L., JR., *Common LISP: The Language*. Digital Press, Billerica, Mass., 1984.
19. TURNER, D. Miranda System Manual, Version 1.009. Research Software Ltd., Canterbury, Kent, May 1987.
20. VAN WIJNGAARDEN, A., ET AL. *Revised Report on the Algorithmic Language Algol 68*. Springer-Verlag, New York, 1976.

Received July 1989; revised April 1990 and May 1991; accepted June 1991