Axiomatizations of Floating Point Arithmetics

Włodzimierz Zadrożny

North Texas State University
Department of Computer Science
Denton, Texas 76203-3886

Abstract. We present a universal scheme for axiomatizing floating point arithmetic. The schema can be used to axiomatize any floating point arithmetic. It consists of a labeled graph with vertices describing some arithmetical properties and edges containing appropriate axioms. The language of floating point arithmetic is developed gradually in this scheme. The scheme can provide a vehicle for studying and implementing various versions of floating point arithmetic.

Introduction.

The purpose of this work was to develop an axiomatization of floating point arithmetic for the project "A System for a Gradual Development and Verification of Microprograms and Machine Architecture" at the Technische Hochschule in Aachen. Since the purpose of the program is to build a universal machine capable of verifying different types of firmware and microprograms, the axiomatization was supposed to be similarly universal.

However the axioms induced, say, by different mantissa representations can be contradictory e.g. sometimes $-(x-y) = y-x$ and sometimes $-(x-y) \neq y-x$ (in twos complement representation), and there are many examples of that kind.

To the extent the author knows the literature, this difficulty has never been overcome in the previous attempts to axiomatize the floating point arithmetic, (cf. [10], [11], [12], [13]). People usually tried to find a list of algebraic properties (axioms) describing behaviour of a floating point representation, or to find a system with nice algebraic properties, that can be applicable for developing arithmetic of real computers.

Although this approach can produce elegant theories it is unsatisfactory because, clearly, there cannot be one axiom system describing the floating point arithmetic for all computers.

On the other hand, floating point organizations of different machines share many common properties. Therefore they should not be treated separately. The scheme presented in this paper allows a uniform treatment of different floating point organizations, with a possible much larger range of applications. The solution consists of a labeled graph, whose vertices specify some arithmetic properties and edges are labeled by (names of) lists of axioms (cf. Fig. 1).

Paths of a fixed length through the graph completely determine different floating point organizations, hence the approximation of the arithmetic of real numbers is achieved by a collection of floating point arithmetics, and not by any single floating point organization.

The proposed system is universal i.e. any system of floating point arithmetic can be included in this scheme.

Figure 1 presents the graph. The dotted paths denote two floating point arithmetics implemented in a Z80 based microcomputer under the CP/M operating system [8]. A similar path for an IBM 370 computer would require much bigger 'boxes' on the Level 5 to contain accurate "descriptions of how operations are performed". The labels on the edges are omitted.

Section 1. The scheme.

The graph is hierarchically built. It contains seven levels. Some levels have a specified, finite number of vertices (Levels: 0,1,4). Other levels contain an arbitrary number of vertices. But edges can connect only vertices on different levels. Therefore the longest path always

---

* The research described in this paper was sponsored by the Alexander von Humboldt Fellowship.

** A complete version of this paper has been submitted for publication in IEEE Transactions on Computers.

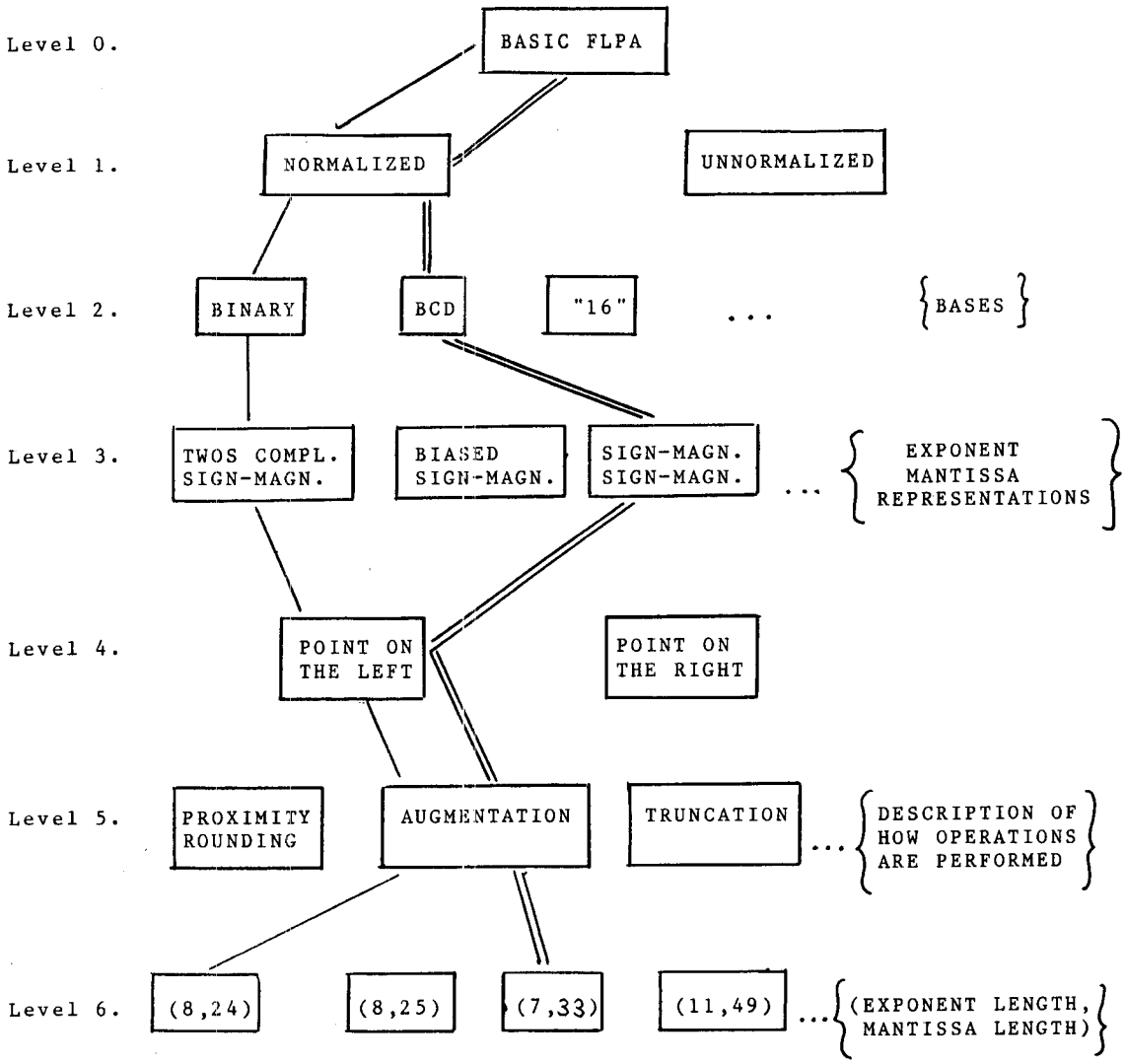

Fig.1

has length six. Vertices of the graph specify the arithmetic properties of different floating point organizations, like the usage of normalized or unnormalized numbers, describe rounding schemes etc. The edges have the form:

$$A \xrightarrow{\quad L+ \quad} B$$

where A is on level i and B is on level i+1 , and L+ is a set of labels of the form:

If A0,A1,... ,Ai(=A),B
then l0,l1,... ,ls.

If A0´,A1´,...,Ai(=A),B
then l0´,l1´,... ,lk´.

.
.
.
etc.

Aj´s denote here the names of vertices, i.e. describe a path up to vertex B, lj´s are names of group of axioms (and inference rules) that hold for any system with the properties described by the vertices.

Example. Consider the edge

BINARY --------- 2-COMPL./SIGN-MAGN.

We will not list all the axioms that appear here, but only show how the scheme works. We have then:

L+ : If NORMALIZED, BINARY,
TWOS COMPL./SIGN-MAGN.
then 10,11,...,1s.

If UNNORMALIZED,BINARY,
TWOS COMPL./SIGN-MAGN.
then 10´,11´,...,1u´.

Where, for example,

10 : x=x ; x=y ---> y=x
(reflexivity and symmetry)

11 : x=y & y=z ---> x=z (transitivity)

12 : y=-x ---> $\forall$ i < tol-1 (x[i]=y[i]
& y[tol-1]=1-x[tol-1]

The symbol = denotes the equality of floating point numbers. This relation is always reflexive and symmetric, but not necessarily transitive.

For example

.1234 E8 =.0012 E10 and
.1201 E8 = .0012 E10 but
.1234 E8 $\neq$ .1201 E8 .

The axiom denoted by 12 says that there is only one representation of -x , for any floating point number x. The symbols (cf. Section 3.0) have the following meaning: "tol" stands for "total length" (in bits) of a floating point number, x[i] denotes the i´th bit of x . The sign bit is the last bit of a number.

13 : x+y=y+x ; x·y = y·x

Since 11 and 12 do not hold for unnormalized numbers, they will not appear among 1j´s . The axioms 10 and 13 will.

The levels have been introduced in order to develop gradually the language of floating point arithmetic. For example, on Level 0 we do not have in our language bit strings. Symbols for different roundings or predicates describing the fields of exponent and mantisssa appear later on when we make a floating point system more complete. The following facts are easy to prove and intuitively obvious:

Proposition. Any path going through all the levels of the graph completely determines a floating point arithmetic.

Proposition. If a path P reaches Level 5 (going through Levels 1-4) then P uniformly determines a decision procedure for all floating point arithmetics given by paths extending P .

Section 2. Advantages of the scheme.

The proposed approach clarifies the relationships between different types of floating point organizations. Moreover it is much more flexible than the approach of developed in some of U.Kulisch papers ([10-13]): He proposed a system of floating point arithmetic which can be practically implemented and possesses a reasonable mathematical structure (i.e. the structure of ringoid). To this end some special conditions are imposed on allowed roundings, in particular, any rounding $\square$ should satisfy the antisymmetry condition:

$$\square(-x) = -\square(x)$$

On the other hand, Yohe [17] (cf.also [5] p.328) has suggested that any computer should be capable of executing the following rounding models:
∇ , Δ , T, A, P .

These roundings are called: optimal downward directed, optimal upward directed, truncation, augmentation and proximity (respectively). It is not difficult ot see that not all of them satisfy the above antisymmetry condition. Moreover, in practice, other roundings are being used, for example, ROM-based roundings (cf. [5]).

The diversity of used systems of floating point arithmetic proves indirectly that there is no single system of floating point organization which would suit best for all purposes. On the other hand, formalization of properties of a floating point arithmetic is necessary for understanding of computer arithmetic. In our scheme we impose no restrictions on allowed finite arithmetics (that gives the necessary flexibility) but require that they must be developed according to some scheme (in order to allow a uniform formalization).

Section 3. The language and axioms.

We now come to a more close examination of the levels, in particular we will show how the language is developed. We will describe the language and the axioms level by level. When analyzing the levels, we will also list some axioms and show how they appear on the edges of the graph. This will help to clarify the meaning of the symbols.

In order to avoid ambiquities the arithmetic operations will be denoted as follows:

Real: $\boxplus$, $\boxminus$, $\boxdot$, $\boxslash$

Floating point : + , - , · , /

Digit strings : $\oplus$, $\ominus$, $\odot$, $\oslash$

Moreover one should distinguish between the equality of floating point numbers, denoted by = in this paper, and the equality of real numbers, denoted by $\underline{=}$.

The symbol = will also be used for denoting equality of bits and bit strings, because this does not lead to any confusion. We examine now the graph level by level:

Level 0. On this level we introduce a significant part of the language:

Variables: x,y,z,x1,y1,z1,x2,...

They will be interpreted later as floating point numbers or -equivalently- as bit (digit) strings.

Constants: zero, one

Additional constants: GRTST,SMPLST,SMMGN

These letters stand for: "the greatest", "the smallest" , and "the smallest magnitude" - numbers.

Functional symbols: +, -, · ,/ , abs,

(the latter stands for the "absolute value").

Predicates: =, <, UNFLOW, OVFLOW, DIVISION_BY_ZERO .

Additional predicates: sm(,), >, $\geq$, $\leq$,

(the interpretation is given below). The lexical object above preceded by the word "additional" need not be used at all, or they can be replaced by other constructs. However all other objects that appear above seem to be indispensable in analyzing any floating point organization.

The two edges linking Level 0 and Level 1 contain , in the L+ - part, labels for the following axioms (among others):

$x=x$, $x=y \rightarrow y=x$,

$x=y \leftrightarrow x-y=zero$,

$zero=-zero$, $x+zero=x$

$\neg(x<x)$, $x<y \rightarrow \neg(y<x)$,

$x<y \ \&\ y<z \rightarrow x<z$

$x<GRTST \ \vee \ x=GRTST$,

$SMLST<x \ \vee \ x=SMLST$

$zero<x \rightarrow OVFLOW(GRTST+x)$,

$x<zero \rightarrow OVFLOW(SMLST+x)$

$zero \leq x \rightarrow abs(x)=x$,

$x<zero \rightarrow abs(x)=-x$

$-(zero<x \ \&\ x<SMMGN)$,

$zero<SMMGN$

$sm(r,x) \leftrightarrow x \neq zero \ \&\ x^{r-1} \neq zero$,

where x^r abbreviates $x \cdot x \cdot \ldots \cdot x$, r-times.

Moreover some basic substitution axioms should be present in this list, like the ones expressing the following properties:

Sub < :

Assume $zero<x\leq w$ and $y<z$. Then a, b, c, d hold.

a) $(y \leq zero \ \vee \ \neg OVFLOW(x+y)) \rightarrow (x+y<w+z \ \vee \ OVFLOW(w+z))$

b) $(\neg UNFLOW(w\ z) \ \&\ \neg OVFLOW(x\ y)) \rightarrow (x\ y<w\ z \ \vee OVFLOW(w\ z))$

c) $zero \leq y \rightarrow (y/w<z/x \ \vee \ \vee \ UNFLOW(z/x))$

d) $zero \leq y \rightarrow x-z<w-y$

Sub = :

$x=w \ \&\ y=z \rightarrow (x.op.y = w.op.z \ \&\ \&\ flag.op.)$,

where .op. stands for +,-,· or / , and flag.op denotes

$(OVFLOW(x.op.y) \leftrightarrow OVFLOW(w.op.z)) \ \&$

$(UNFLOW(x.op.y) \leftrightarrow UNFLOW(w.OP.z))$

[& (DIVISION_BY_ZERO(x/y) <--->

(DIVISION_BY_ZERO(w/z)) - in the case of division].

The axioms listed below appear in the the edge going to NORMALIZED .

$x=y \ \&\ y=z \rightarrow x=z$ (transitivty of =)

$[\neg OVFLOW(x+y) \ \&\ \neg OVFLOW(y+z) \ \&$

& ¬ OVFLOW((x+y)+z) & ¬ UNFLOW(x+y) &

& ¬ UNFLOW(y+z) & ¬ UNFLOW((x+y)+z)]

---> (x+y)+z=x+(y+z) ,

(associativity of +)

$(x \cdot y) \cdot z = x \cdot (y \cdot z)$,

under similar conditions ,

(associativity of ·)

One can list more axioms that appear on the edges linking the top two levels but this is not necessary for the purpose of exposition. Note that we do not have yet

$-(-x) = x$ nor $(x-y) = -(y-x)$,

since our arithmetic is not determined yet enough, for example, in the twos complement arithmetic these rules do not hold. In order to express the number representation we must add new objects to the language.

Level 1. At this level most of the language is introduced, however it will take more steps to determine most of the properties of the introduced objects.

We introduce here:

Variables of the second kind:

i,j,k,i1,j1,k1,...

ranging over the numerals.

Constants: 0,1,2,3,...,199,...
denoting numerals,

0,1 - denoting bits.

Functional symbol: [ ] ,

where [ ] denotes the function from Bit Strings × Numerals into {0,1} .

We will write as usual x[i] for [ ](x,i) .

Convention. Our aim is to be able to talk about floating point numbers and their properties. We need a convention about representation of numbers in our system. Without loss of generality we can assume that any number consists of two fields: The field of the exponent and the field of the mantissa. It is necessary to be able to say in our language that a certain bit is a bit of an exponent or of a mantissa. To this end we introduce the predicates EXPF and MANF . We need also the variables for the length of these fields: expfl and manfl. It is also useful to introduce the variable tol (total length). We then have :

tol = expfl + manfl

There are many representations that can be chosen for exponent and mantissa. Some of them do not use the sign explicitly (like the twos complement), therefore we adopt the convention that if the sign bit must be described explicitly (as in the sign-magnitude notation) then it will be the last bit in any of these two fields. For the sake of completeness and convenience we add another two predicates to our list of additional predicates:

EXPS and MANS
(exponent-sign, mantissa-sign).

We augment our language with:

Predicates: EXPF, MANF

Additional predicates: EXPS, MANS

Variables: tol, expfl, manfl

The properties of the above objects are stated by the axioms:

tol = expfl + manfl

$\bigwedge_{i<exfl}$ EXPF(x[i]) ,

(we will abbreviate $\bigwedge_{i}$ by $\forall i$)

$\forall i$ (expfl $\leq$ i < tol ---> MANFL(x[i]))

We need also

Variables: expl, manl

(the length of exponent and mantissa without the sign bit), with the properties (axioms):

- EXPS(x[expfl-1]) ---> expl=expfl-1

EXPS(x[expfl-1]) ---> expl=expfl

(i.e. if there is no sign bit then the length of the exponent equals the length of the field of the exponent), and with similar axioms for the length of the mantissa - manl .

Now the language is rich enough to express the property like "a number is represented in base 2" or "a number is represeted by the BCD code". It should be clear how to write the appropriate axioms for the edges going from Level 1 to Level 2 . The same applies to the

edges connecting Level 2 and Level 3. There we describe properties like

$\forall i$ ((i < expf1 ---> -x[i] = x[i])&

& (expf1 $\leq$ i $\leq$ man1-1 --->

---> -x[i] = 1 - x[i])),

for the edge going to

.../ ONES COMPLEMENT.

Level 2. NO NEW LEXICAL OBJECTS !

Level 3. At this level we already know how the numbers are represented, except for the decimal (binary) point. In order to complete the description we introduce:

Predicates: POINTL , POINTR .

The edges connecting Level 3 and Level 4 contain labels for the axioms:

POINTL(x) <---> - POINTR(x) , and either

POINTL(x) or POINTR(x).

Level 4. The edges linking the vertices on this level with the vertices of the next level describe how the arithmetic operations are performed . This description should be done in terms of operations on strings of bits.

Thus we must introduce names for functions dealing with bit strings. One should also take into account that results of arithmetic operations on mantissas do not preserve the length of input, i. e. the result can be a string of bits longer than the operands. Therefore it would be rather difficult difficult to avoid operations on strings of variable length.So we need a new kind of

Variables: e, e1, e2, ...

that will be interpreted as bit strings of variable lengths, and

Functional symbols: $\frown$, $\restriction$, lngth,

denoting respectively : concatenation, restriction and length of a string.

Lack of space does not permit for a full description of arithmetic operations for different kinds of floating point systems. These descriptions can be written in a not-too-difficult way using radix polynomials introduced in the paper of J. Kent [9] , (cf. also [5]).

We need also

Functional symbols: exp, man

denoting functions with the properties

e1=exp(x) <---> $\forall$ i< expf1 (e1[i]=x[i]),

similarly for e2 = man(x).

(These functions return the strings of exponent and mantissa bits - respectively. We use here the base 2 only, but this approach works for all positive bases (cf. [7], [5]).

It is easy to formalize operations on mantissas as operations on corresponding to them binary radix polynomials. The only problem arising here is that the number of nonzero coefficients in the resulting polynomial is greater then the length of mantissas in most cases. In order to describe this phenomenon it is convenient to have functions that deal with bit strings of mantissas as binary radix polyomials. We introduce

Functional symbols: k_plus, k_times, k_divid

that are interpreted as arithmetical operations corresponding to them, the parameter k specifies the length of result i. e. the number of coefficients in the resulting polynomial.

If some uniform coding of binary radix polynomials into bit strings is assumed then the axioms describing the functions introduced above are common for all the edges leading to Level 5 . Using the operations defined on mantissas one can define arithmetic functions on floating point numbers which will give as results extended floating point numbers i. e. bit strings of the form

! a floating point number ! extension !

where the bits of the mantissa of the floating point number in this string, together with the extension, give the code of the appropriate polynomial.

We will use

Functional symbols : $\oplus$, $\ominus$, $\odot$, $\oslash$

for these extended arithmetical operations.

After a normalization and rounding or truncation to tol-many bits we will get a result in the form of a floating point number. Therefore we need:

Functional symbols :

k_shl , denoting the shifting of a mantissa by k bits left, increasing the exponent by k.

The domain of k_shl is the set of extended floating point numbers.

k_shr , denoting the shifting of the mantissa k bits to the right, decreasing the exponent, and increasing the length of the extension accordingly.

k_T , truncation of mantissa to k digits (k $\geq$ manfl).

T , truncation of the mantissa to manfl digits.

As in Section 2 we introduce: k_∇ , ∇ k_Δ , k_A , A , k_P , P , where, for the functions with the prefix k_ , M is the class of floating point numbers with k-bit mantissas, and for the functions without the prefix, M is the class of floating point numbers with mantissas of the length manfl .

norm , which normalizes an extended floating point number i.e. sets the leading coefficient to one, but does not change the total length of the extended mantissa.

Note that for different bases norm has different meanings, for example, in base 16 the first digit of a normalized mantissa can be zero. It is clear that the descriptions of the above functions by sets of axioms will vary for POINTL and POINTR but it is easy to see how.

We will treat k in k_F as a parameter (a numeral) of the function F, so we allow writing sentences of the form :

k = manfl - 7 & el=k_shl(xl).

We also need

Predicates: LSSTH , LESS

where LSSTH(e1,e2) states that the value of e1 as a binary radix polynomial is less than the value of e2 as a binary radix polynomial. LESS(x,y) holds iff LSSTH(exp(x),exp(y)) or exponents are equal as binary radix polynomials but man(x) is less (LSSTH) than man(y) as a binary radix polynomial.

Using the above defined functions and predicates we describe the floating point arithmetical operations in terms of operations on string of bits.

Example. If a path through the graph describes the arithmetic of IBM 370 computer then the label L+ of the edge going to Level 5 contains the definition of addition:

Assuming LESS(x,y)

x+y = k_T norm (x $\oplus$ k+1_T (i_shl(y))

where k is manl that will be set to 25 on the edge going to Level 6, i is the difference of the exponents, and normalization is in base 16.

Level 5. We do not introduce any new constructs to our language, it is already rich enough and complicated enough. We need however

Functional symbols :

error, relerr, err1, err2, ...

denoting different kinds of errors, say, the absolute error, the relative error etc. These symbols are not necessary to describe fully the numbers and operations of any floating point arithmetic, but because of their importance for any kind of numerical analysis they must appear in our scheme.

Level 6. The vertices on the last level contain only label of the form manfl=8 & expfl=24 . Thus the length of the exponent and mantissa are set to concrete values here.

Section 4. Final remarks.

Remark 1. The arithmetics are already divided into NORMALIZED and UNNORMALIZED at Level 1. This division is supported both by practice and theory. Almost all computer system use normalized numbers (in particular IBM370 , VAX'es, and all microcomputers), unnormalized arithmetic is given as an option for Cyber.

An axiomatic analysis of the latter arithmetic is contained in [19]. An analysis of errors for both types of arithmetic is given in [9].

Remark 2. Strictly speaking, the computing of errors is not a part of a floating point arithmetic, because error is a difference between a received value and some "true" value i.e. requires a superstructure of the usual arithmetic of real numbers. Therefore we can say that errors belong to "meta-floating point arithmetic". On the other hand , error analysis is indispensable in numerical problems, so it must be included in any complete axiomatization of a floating point organization. In our scheme we can formalize a substantial part of the error analysis in a floating point arithmetic itself. One possibility is to formalize

the interval arithmetic within the presented system ([14],[9],[18] might be helpful).

Remark 3. The approach used here seems to have more applications in the areas where it is necessary to approximate (models of) rather large systems using smaller systems or theories (e. g. in expert systems in artificial intelligence).

Such an approximation can be done by a gradual development of a language together with a partial axiomatization of interesting phenomena, i.e. by creating a graph of the kind presented in this paper. Then a most convenient, for a given task, system (i. e. path through the graph) can be chosen.

* * *

I would like to thank the Alexander von Humboldt Stiftung for their generous support during my stay in West Germany. I would also like to express my gratitude to Michael Richter for inviting me to Aachen. Discussions with him and his colleagues contributed to my better understanding of the subject.

Finally, special thanks are due to the anonymous referees whose remarks helped me to avoid many mistakes and to clarify the paper.

REFERENCES :

[1] S.D.Crocker, L.Marcus, D.van-Mierop, "The ISI Microcode Verification System", in: G.Chroust (ed.), "Firmware, Microprogramming and Restructurable Hardware", Proc. of the IFIP Working Conference, North Holland, 1980.

[2] S.Dasgupta, "Some aspects of high level microprogramming", Computing Surveys ,12,No.3,1980.

[3] S.Dasgupta, "Towards a microprogramming language schema", Proc.11 Annual Workshop on Microprogramming (ACM), 1978.

[4] I.Flores, "The Logic of Computer Arithmetic", Prentice Hall,1963.

[5] K.Hwang, "Computer Arithmetic", Wiley & Sons, 1979.

[6] H.Katzan, "Microprogramming Primer", McGraw-Hill, 1977.

[7] J.G.Kent, "Highlights of a study of floating point instructions", IEEE Transactions on Computers, vol.C-26,July 1977.

[8] H.J.Kirschbaum, "Floating point arithmetic fuer Z80", Technische Hochschule Aachen, notes, 1983.

[9] D.Knuth, "The Art of Computer Programming",vol.II, Addison Wesley, 1981.

[10] U.Kulisch, "Ein Konzept fur eine allgemeine Theorie der Rechnerarithmetic, in: Computing, Supplementum 1, 1977.

[11] U.Kulisch, "Mathematical Foundations of Computer Science", IEEE Transactions on Computers, vol.c-26, July 1977.

[12] U.Kulisch, "Uber die beim numerischen Rechnen mit Rechenanlagen auftretende Raume", in: Computing, Supplementum 1, 1977.

[13] U.Kulisch, W.Miranker, "Arithmetic Operations in Interval Spaces", in: Computing, Supplemetum 2, 1980.

[14] D.W.Lozier, "The use of Floating -Point and Interval Arithmetic in the Computation of Error Bounds", IEEE Trans. on Computers, vol.C-32, April 1983.

[15] K.Merkel, "Maschinenspecifikationsprache MDL", Diplomaarbeit, Technische Hochschule Aachen, 1983.

[16] A.S.Tanenbaum, "Structured Computer Organization", Prentice Hall, 1976.

[17] J.M.Yohe, "Roundings in Floating Point Arithmetic", IEEE Transactions on Computers,vol.C-22,June 1973.

[18] O.Watanuki, M.D.Ercegovac, "Error Analysis of Certain Floating Point On-Line Algorithms", IEEE Trans. on Computers, vol.C-32, April 1983.

[19] A.v.Wijngaarden, "Numerical analysis as an independent science",BIT,6,1966.