$$
\begin{array}{ccccccc}
 & & \mathbb{R} & \supset & D & \supset & R \\
 & & V\mathbb{R} & \supset & VD & \supset & VR \\
 & & M\mathbb{R} & \supset & MD & \supset & MR \\
 & & & & & & \\
\mathbb{P}\mathbb{R} & \supset & I\mathbb{R} & \supset & ID & \supset & IR \\
\mathbb{P}V\mathbb{R} & \supset & IV\mathbb{R} & \supset & IVD & \supset & IVR \\
\mathbb{P}M\mathbb{R} & \supset & IM\mathbb{R} & \supset & IMD & \supset & IMR \\
 & & & & & & \\
 & & \mathbb{C} & \supset & CD & \supset & CR \\
 & & V\mathbb{C} & \supset & VCD & \supset & VCR \\
 & & M\mathbb{C} & \supset & MCD & \supset & MCR \\
 & & & & & & \\
\mathbb{P}\mathbb{C} & \supset & I\mathbb{C} & \supset & ICD & \supset & ICR \\
\mathbb{P}V\mathbb{C} & \supset & IV\mathbb{C} & \supset & IVCD & \supset & IVCR \\
\mathbb{P}M\mathbb{C} & \supset & IM\mathbb{C} & \supset & IMCD & \supset & IMCR
\end{array}
$$

Operat. in $\mathbb{R}, V\mathbb{R}, M\mathbb{R}, \mathbb{C}, V\mathbb{C}, M\mathbb{C}$

$$\{M, *\},\ \{\mathbb{P}M, *\} : \quad \bigwedge_{X, Y \in \mathbb{P}M} X * Y := \{x * y \mid x \in X \wedge y \in Y\}$$

$\Rightarrow$ Operat. in $\mathbb{P}\mathbb{R}, \mathbb{P}V\mathbb{R}, \mathbb{P}M\mathbb{R}, \mathbb{P}\mathbb{C}, \mathbb{P}V\mathbb{C}, \mathbb{P}M\mathbb{C}$

$\boxed{\text{Kulisch} \\ \text{7 June 88}}$

1. trad. Arithm. $\{R, \boxplus, \boxminus, \boxtimes, \boxslash, \leqq\}$

$$\begin{tikzcd} \mathbb{R} \arrow[r] \arrow[d] \arrow[dd, bend right] & R \arrow[d] \arrow[dd, bend right] \\ \mathbb{C} & CR \\ M\mathbb{R} & MR \end{tikzcd}$$

$$\alpha = \alpha_1 + i\alpha_2 \,,\; \beta = \beta_1 + i\beta_2 \in CR$$

$$\alpha \boxtimes \beta := (\alpha_1 \boxtimes \beta_1 \boxminus \alpha_2 \boxtimes \beta_2 \,,\; \alpha_1 \boxtimes \beta_2 \boxplus \alpha_2 \boxtimes \beta_1)$$

$$a = (a_{ij}) \,,\; b = (b_{ij}) \in MR$$

$$a \boxtimes b := \left(\boxed{\sum_{k=1}^{n}} a_{ik} \boxtimes b_{kj} \right)$$

2. Semimorphism. $\square : M \longrightarrow N$

$$\begin{tikzcd} \mathbb{R} \arrow[r] \arrow[d] \arrow[dd, bend right] & R \\ \mathbb{C} \arrow[r] & CR \\ M\mathbb{R} \arrow[r] & MR \end{tikzcd}$$

$a \circledast b := \square(a * b)$ f.a. $a, b \in N$, $* \in \{+, -, \times, /\}$

$\square a = a$ f.a. $a \in N$

$a \leq b \Rightarrow \square a \leqq \square b$ f.a. $a, b \in M$

$\square(-a) = -\square a$ f.a. $a \in M$

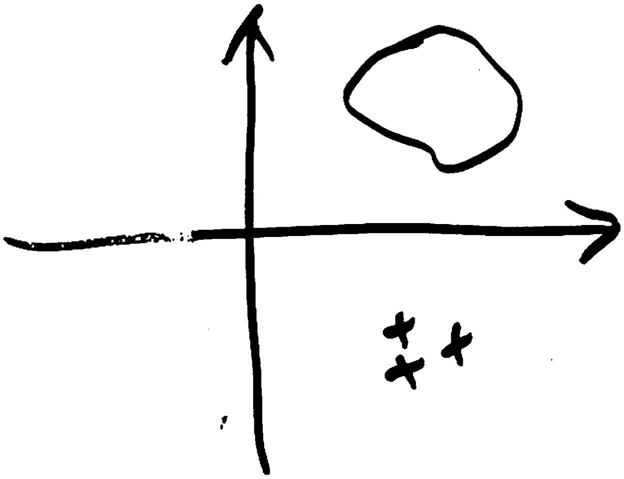
$a \subseteq \square a$ f.a. $a \in M$ $(= I\cdot\cdot)$

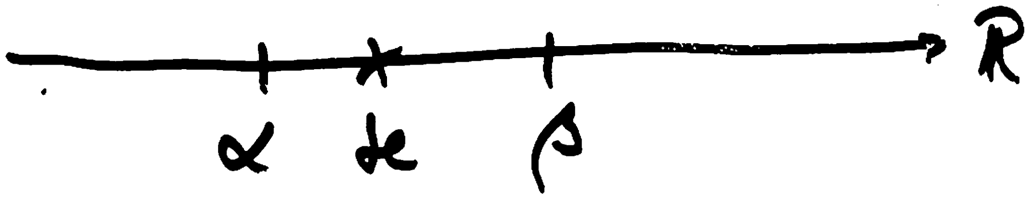
$$\alpha \boxtimes \beta = \square(\alpha \times \beta) = \square(\alpha_1 \times \beta_1 - \alpha_2 \times \beta_2 \,,\; \alpha_1 \times \beta_2 + \alpha_2 \times \beta_1)$$
$$= (\square(\alpha_1 \times \beta_1 - \alpha_2 \times \beta_2) \,,\; \square(\alpha_1 \times \beta_2 + \alpha_2 \times \beta_1))$$

$$a \boxtimes b = \square(a \times b) = \square\left(\sum_{k=1}^{n} a_{ik} b_{kj}\right) = \left(\square \sum_{k=1}^{n} a_{ik} b_{kj}\right)$$

$$a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

$a \le b$ $\quad$ $a_i \le b_i$ f.a. i





$\square / \quad \alpha \le x \le \beta$

([illegible]) $\quad \square\alpha \le \square x \le \square\beta$

(R1) $\quad \alpha \le \square x \le \beta$

$\alpha \le x * y \le \beta$

(R2) $\quad \square\alpha \le \square(x * y) \le \square\beta$

(R1) $\quad \alpha \le \square(x * y) \le \beta$

(R6) $\quad \alpha \le x \boxed{*} y \le \beta$

fixed-point arithmetic

0.2143769851
0.3214367534

0.7124342678
0.8123950690

0.1243467809

13.2467869097

0.1324678691

addition/subtraction in fixed-point arithmetic is 'error free'

scaling requirement → overscaling

0.0000021473

loss of accuracy

therefore: floating-point arithmetic

exponent part takes care of the scaling automaticall

multiplication and division rel. stable operations

addition/subtraction are problematic

ideal computer:

multiplication, division in floating-point arithm.

addition, subtraction in fixed-point arithm.

$$x = \begin{pmatrix} 10^{20} \\ 1223 \\ 10^{24} \\ 10^{18} \\ 3 \\ -10^{21} \end{pmatrix} \qquad y = \begin{pmatrix} 10^{30} \\ 2 \\ -10^{26} \\ 10^{22} \\ 2\,111 \\ 10^{19} \end{pmatrix}$$

$$x \cdot y = 10^{50} + 2446 - 10^{50} + 10^{40} + 6\,333 - 10^{40} = 8779$$

$$x \boxdot y = 0$$

$$x = 0.10005 \times 10^5 \qquad \begin{array}{r} 0.1000500 \times 10^5 \\ -\,0.0999730 \times 10^5 \end{array}$$

$$y = -0.99973 \times 10^4 \qquad \begin{array}{r} 0.0000770 \times 10^5 \\ 0.77000 \quad \times 10^1 \end{array}$$

$$x = \Box(x_1 \times x_2), \quad x_1 \times x_2 = 0.1000548241 \times 10^5$$

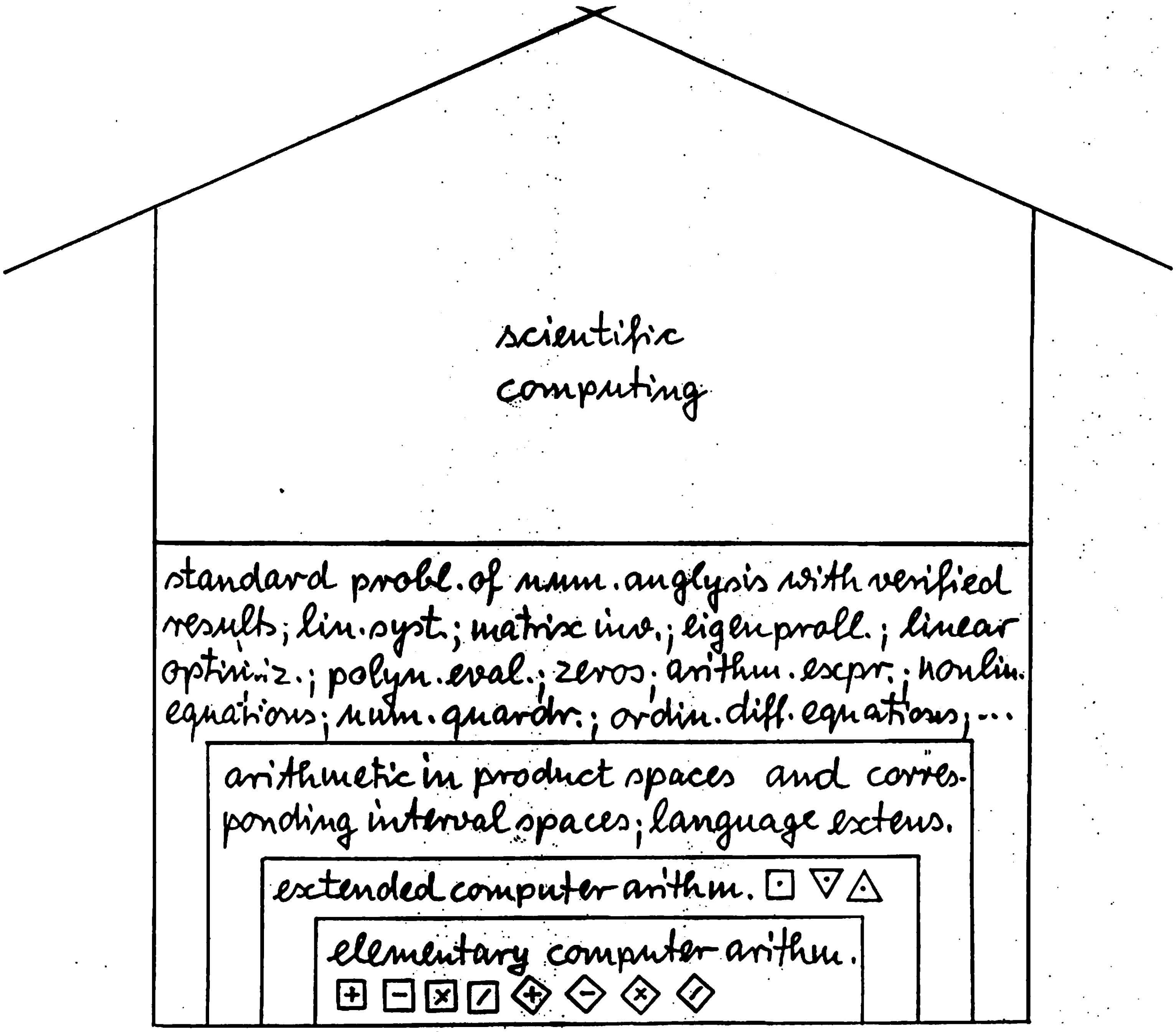
$$y = \Box(y_1 \times y_2), \quad y_1 \times y_2 = 0.9997342213 \times 10^4$$

$$x_1 \times x_2 - y_1 \times y_2 = \begin{array}{r} 0.1000548241\,0 \times 10^5 \\ -\,0.0999734221\,3 \times 10^5 \end{array}$$

$$= 0.00008140197 \times 10^5$$

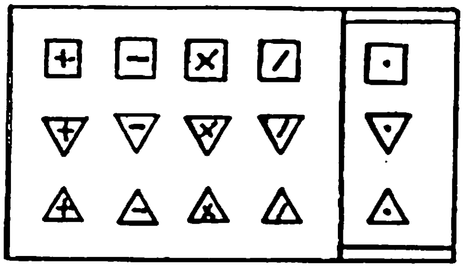
$$x_1 \times x_2 - y_1 \times y_2 = 0.8140197 \times 10^1$$

$$\Box(x_1 \times x_2 - y_1 \times y_2) = 0.81402 \times 10^1$$


scientific computing
standard probl. of num. analysis with verified results; lin. syst.; matrix inv.; eigenprobl.; linear optimiz.; polyn. eval.; zeros; arithm. expr.; nonlin. equations; num. quadr.; ordin. diff. equations; ...
arithmetic in product spaces and corresponding interval spaces; language extens.
extended computer arithm. $\boxdot$ $\triangledown$ $\vartriangle$
elementary computer arithm.
$\boxplus$ $\boxminus$ $\boxtimes$ $\boxslash$

implementation of semimorph. in

$R, VR, MR, CR, VCR, MCR, IR, IVR, IMR, ICR, IVCR, IMCR$:



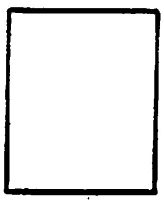
$$a \boxdot b = \square \sum_{i=1}^{n} a_i \cdot b_i$$

$$a \mathbin{\overset{\cdot}{\nabla}} b = \nabla \sum_{i=1}^{n} a_i \cdot b_i$$

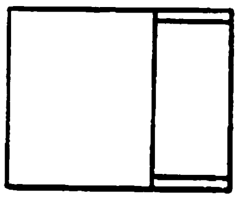
$$a \mathbin{\overset{\cdot}{\Delta}} b = \Delta \sum_{i=1}^{n} a_i \cdot b_i$$

$\boxed{*}$ traditional numerical analysis $* \in \{+, -, \times, /\}$

$\overset{*}{\nabla}, \overset{*}{\Delta}$ traditional interval arithmetic



Z23 (1967) in software, ext. ALGOL
X8 (1968) in hardware, ext. ALGOL
IEEE-arithmetic-standard (1983)



high accuracy, operations in product sets
Z 80 (1980) PASCAL-SC
IBM-PC (1983)
MOTOROLA 68000 (1982)
hardware unit (1983, G. Schweizer)
IBM /370, 4361, ACRITH (1983)
IBM 9377 (1986)
Siemens ARITHMOS (1986)
NAS, Hitachi, BASF, Nixdorf
GAMM-Resolution on Computer Arithmet
FORTRAN-SC

Higher Order Computer Arithmetic

linear systems, matrix inversion, eigen-problems, linear programming problems, expression evaluation with maximum quality

1. A, B matrices, x, y, z vectors
   x = x + A*y + B*z
   x = #*(x + A*y + B*z)
   x = #<( ), x = #>( ), x = ##( ), x = #( )

2. A_i, B_i, $i = 1(1)n$, vectors or matrices
   x = #*(sum(A(i)*B(i), i = 1, n))
   computes $x = \sum_{i=1}^{n} A_i \cdot B_i$ with max. quality

3. evaluation of expressions with max. quality
   b = #*(x + 4*(3.0e8 * y/z))
   b = #<(((4*x - 5)*x + 3)*x + 2.5e3)
   c = ##(sum(a(i)*x**i, i = 1, n))
   computes $\sum_{i=1}^{n} a_i x^i$ to max. quality

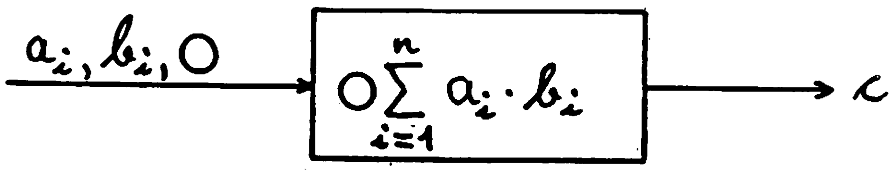
4. computation of program parts with max. quality
   <u>accurate</u> (x, y<, z>) <u>do</u>
   <u>begin</u> PROG
   <u>end</u>
   computes x, y, z to max. quality and rounds x to nearest, y downwards, z upwards

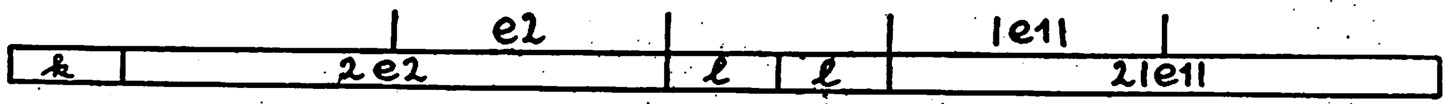
$$z = m \cdot b^{e}, \quad m = 0.9784231, \quad b = 10, \quad e = 12$$

$$a = (a_i), \quad b = (b_i), \qquad a_i, b_i \in R(b, l, e1, e2)$$

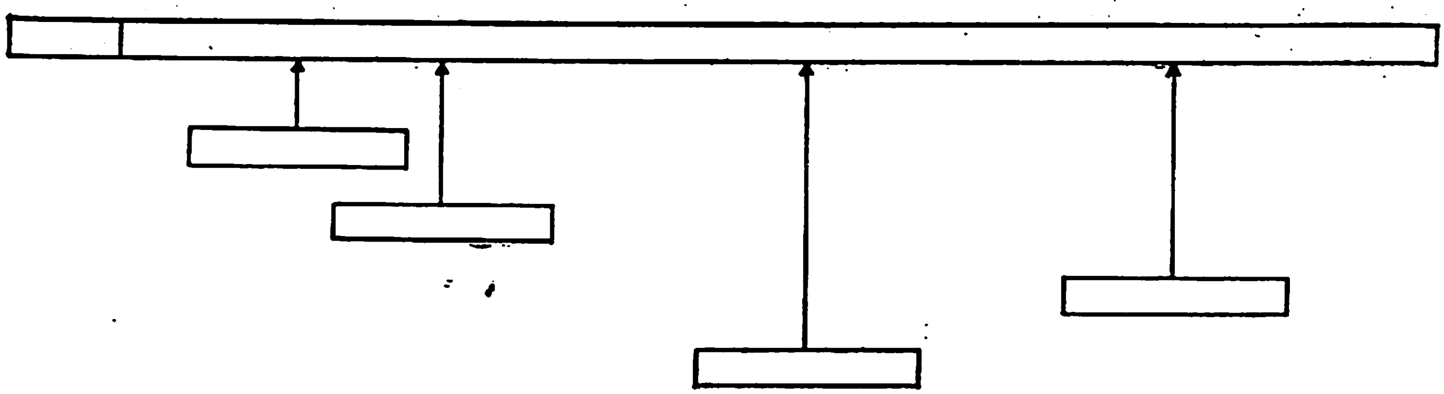
$$c = \bigcirc \sum_{i=1}^{n} a_i \cdot b_i, \qquad a_i \cdot b_i \in R(b, 2l, 2e1, 2e2)$$

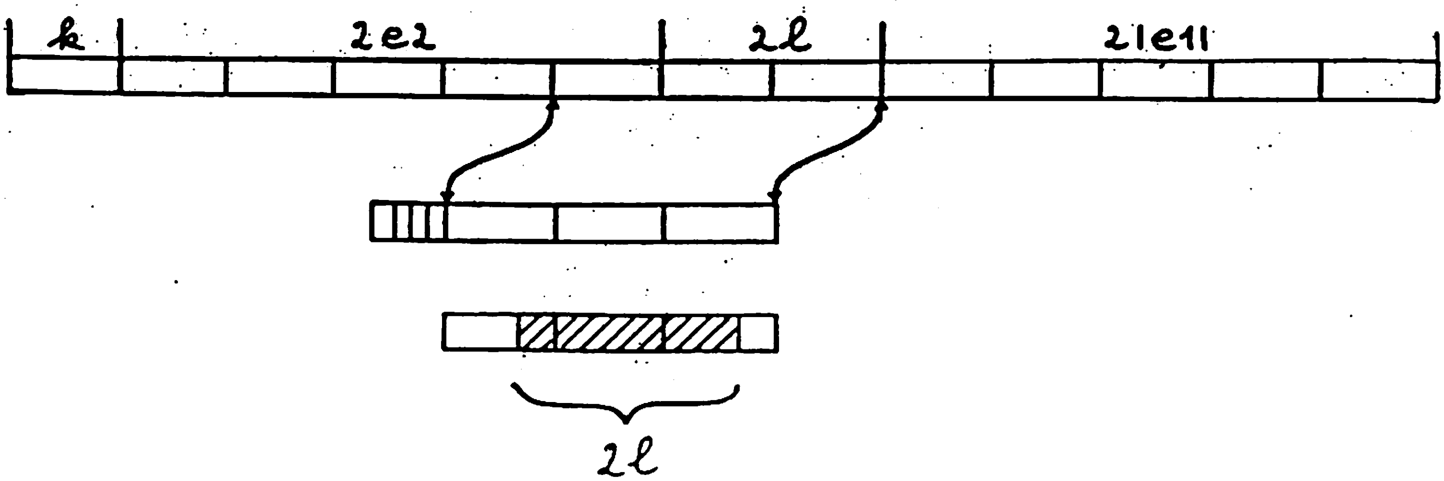
$$\bigcirc \in \{\square, \nabla, \Delta\}$$

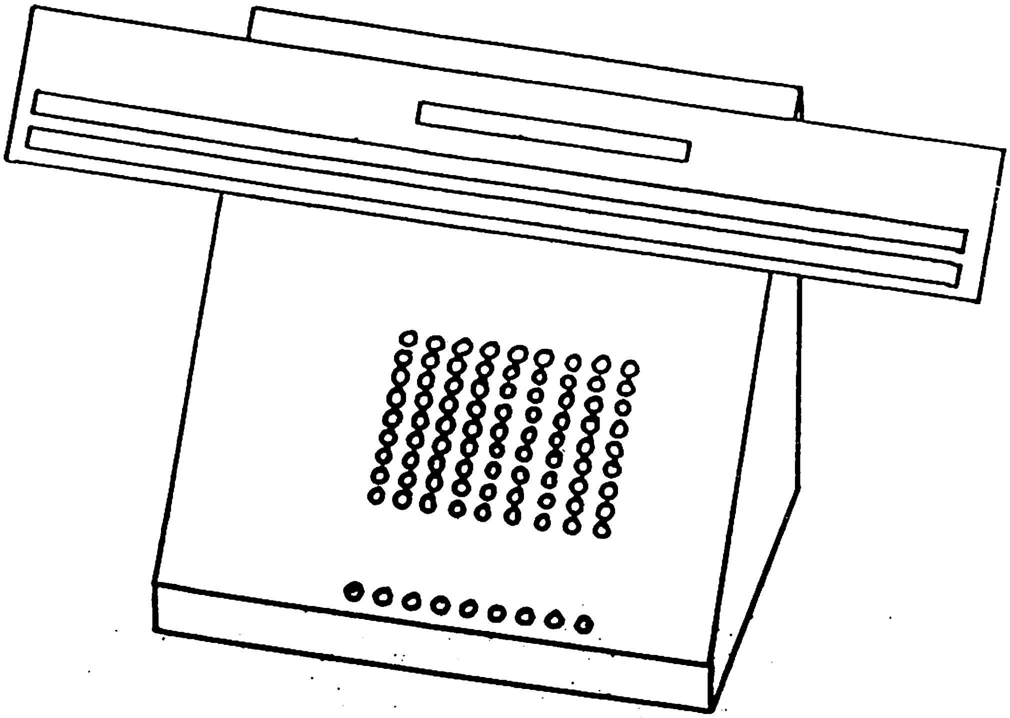


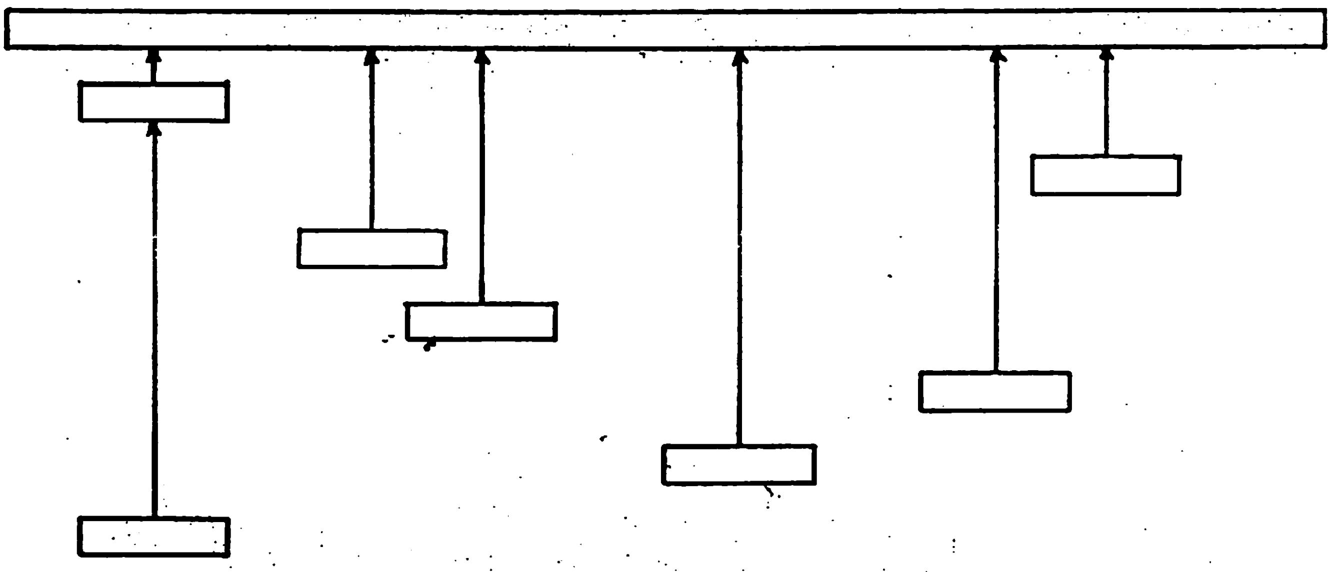


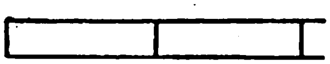
$$L = k + 2e2 + 2l + 2|e1|$$





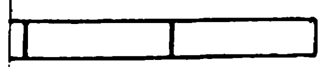






simpler applied as ofte as poss
faster
fully accurate
generalization of this operation for electronic computers has the same properties

simple



GESELLSCHAFT FÜR ANGEWANDTE MATHEMATIK UND MECHANIK (GAMM)

Resolution on Computer Arithmetic

The elementary floating-point operations +, -, *, / in electronic computers are currently required to be of highest machine accuracy: For any choice of operands, the computed result must coincide with the rounded exact result of the operation, rounded according to the rounding mode in use (if no overflow occurs). For reference, see the IEEE Arithmetic Standards 754 (binary floating-point arithmetic) and 854 (general floating-point arithmetic).

In recent years there has been a significant shift of numerical computation from general-purpose computers towards vector and parallel computers - so-called supercomputers. Along with the 4 elementary operations +, -, *, /, these computers usually offer compound operations as additional elementary operations. This leads to an increase of several orders of magnitude in computing power. Some of these elementary compound operations are:

- multiply and add:	a * b + c
- multiply and subtract:	a * b - c
- accumulate:	computes the sum of the components of a vector
- multiply and accumulate:	computes the inner (or scalar) product of two vectors

and others.

GAMM requires that all elementary compound operations be implemented by the manufacturer in such a way that guaranteed bounds are delivered for the deviation of the floating-point result from the exact result. It is desirable and usually achievable that for all possible data the computed result of such a compound floating-point operation agrees with the result that would be obtained if the exact result were computed and then rounded by the rounding in use (if no overflow occurs). In this case no explicit error bounds need be delivered. The user should not be obliged to perform an error analysis every time an elementary compound operation, predefined by the manufacturer, is employed.

All elementary compound operations should also be provided with directed roundings, a feature needed both for fast computation of reliable and narrow bounds in numerical algorithms and for verification of the correctness of computed results. It must be ensured that the final floating-point result can differ from the exact result only in the direction defined by the rounding in use. This is already required of the elementary floating-point operations by the arithmetic standards mentioned above.

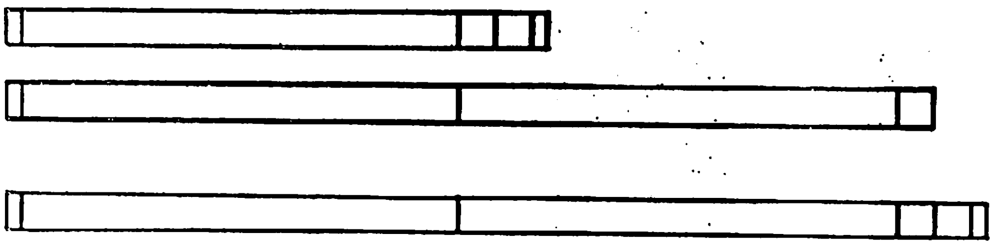
(single), double, (extended, quadruple)

$\underline{\textbf{real}}$ a, b, c, d, a_i, b_i

$a+b$		$a \oplus b$
$a-b$		$a \ominus b$
$a*b$		$a \circledast b$
a/b		$a \oslash b$
$a+c*d$	multiply and add	$a \oplus c \circledast d$
$a-c*d$	multiply and subtr.	$a \ominus c \circledast d$
$a_1+a_2+\cdots+a_n$	accumulate	$a_1 \oplus a_2 \oplus \cdots \oplus a_n$
$a_1*b_1+a_2*b_2+\cdots+a_n*b_n$	multiply and acc.	$a_1 \circledast b_1 \oplus a_2 \circledast b_2 \oplus \cdots \oplus a_n \circledast b_n$

$\square(a+b)$	$\nabla(a+b)$	$\triangle(a+b)$	(1
$\square(a-b)$	$\nabla(a-b)$	$\triangle(a-b)$	(2
$\square(a*b)$	$\nabla(a*b)$	$\triangle(a*b)$	(3
$\square(a/b)$	$\nabla(a/b)$	$\triangle(a/b)$	(4
$\square(a*b+c*d)$	$\nabla(a*b+c*d)$	$\triangle(a*b+c*d)$	(5
$\square(a*b-c*d)$	$\nabla(a*b-c*d)$	$\triangle(a*b-c*d)$	(6
$\square(a_1+a_2+\cdots+a_n)$	$\nabla(a_1+a_2+\cdots+a_n)$	$\triangle(a_1+a_2+\cdots+a_n)$	(7
$\square(a_1*b_1+a_2*b_2+\cdots+a_n*b_n)$	$\nabla(a_1*b_1+a_2*b_2+\cdots+a_n*b_n)$	$\triangle(a_1*b_1+a_2*b_2+\cdots+a_n*b_n)$	(8

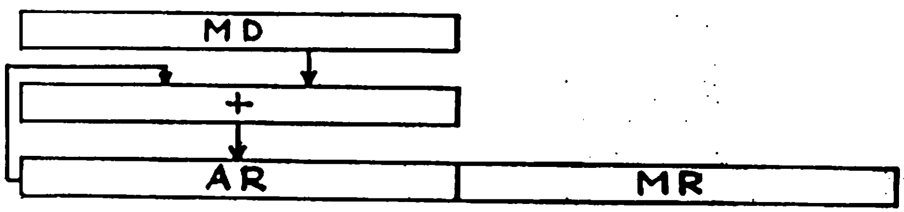
(1)–(4): JEEE-St.



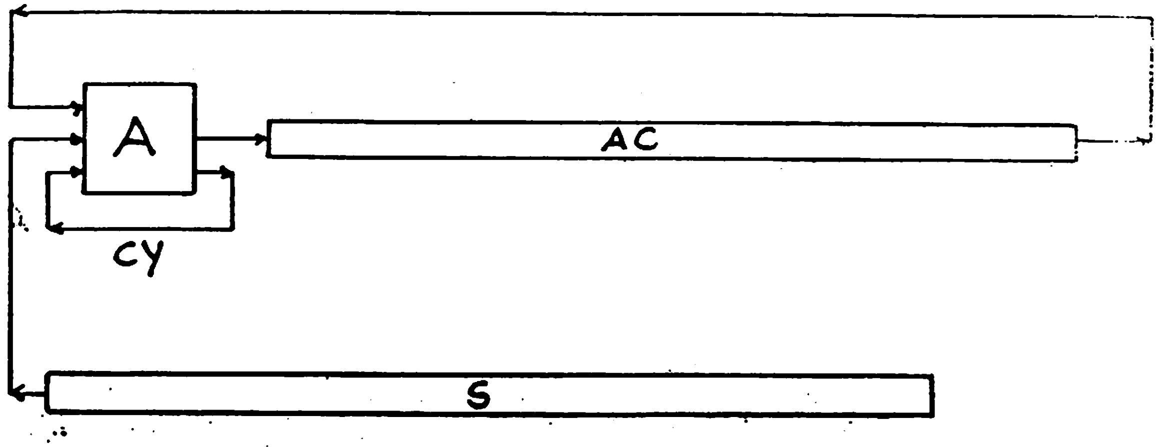
(1) ··· (4) JEEE-Pr.

(1)' ··· (4)

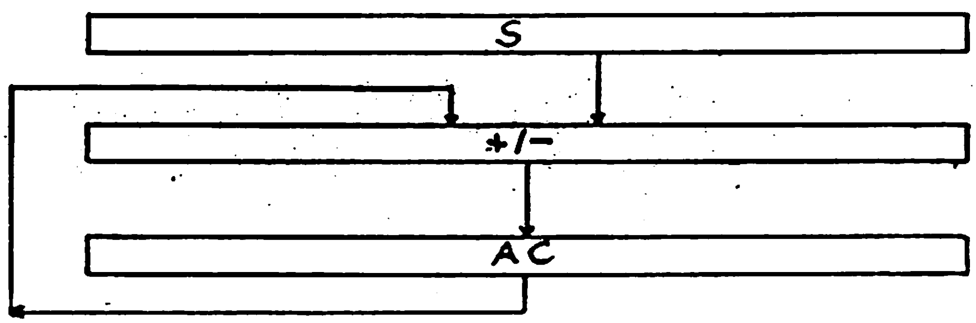
(1)'' ··· (6) & double real

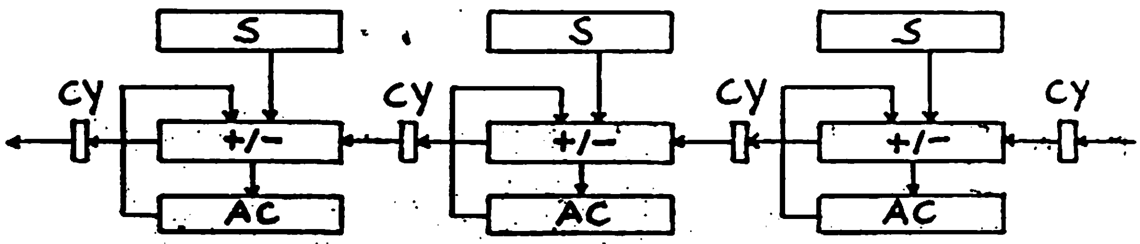


Serienaddierer

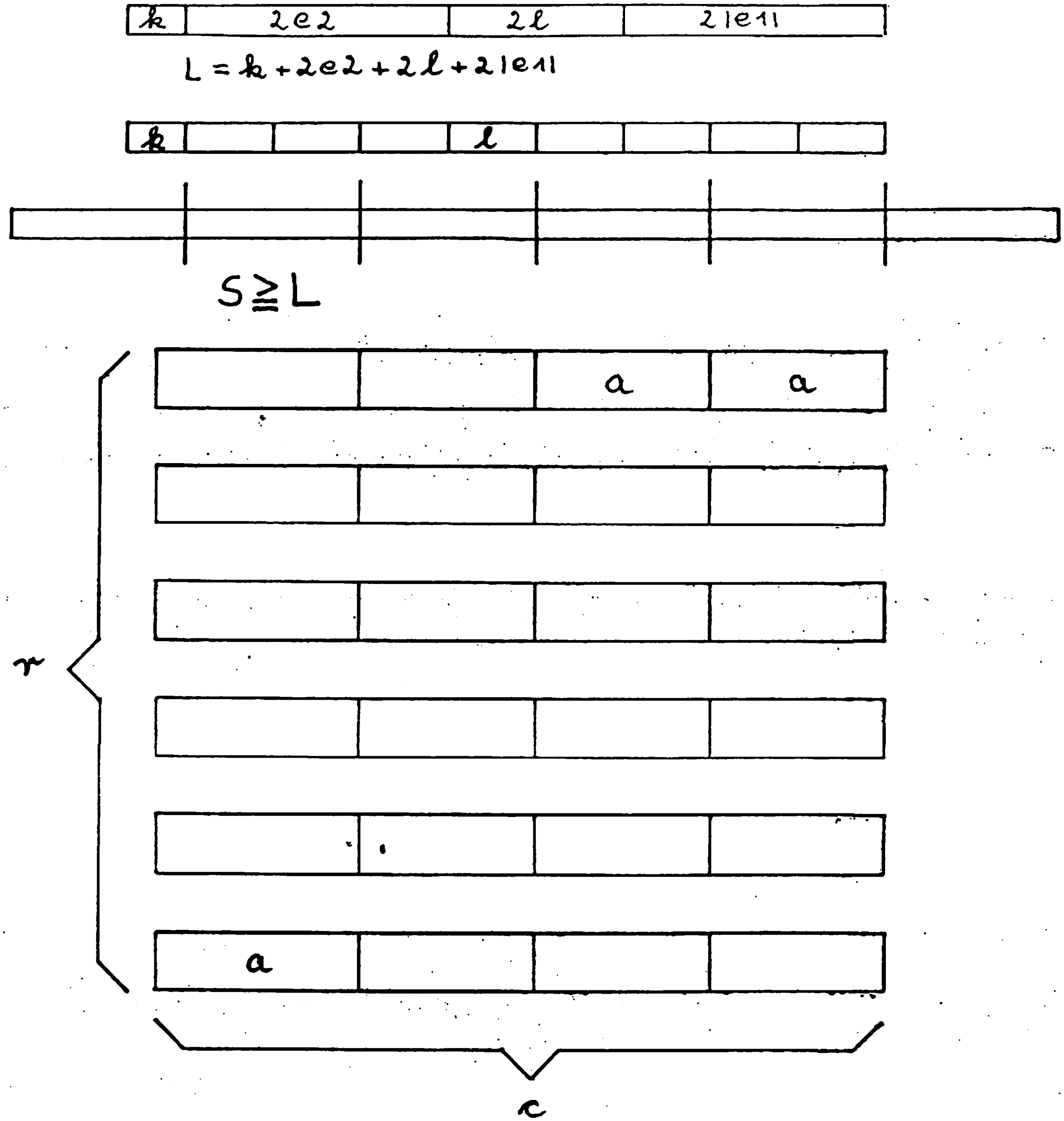


Paralleladdierer





aufgebaut aus Teilstücken



$S = r \cdot c \cdot a$ Ziffern der Basis b

a : Anzahl der Ziffern eines Teil<u>a</u>ddierers

r : Anzahl der Zeilen (<u>R</u>eihen, <u>r</u>ows)

c : Anzahl der Spalten (<u>c</u>olumns)

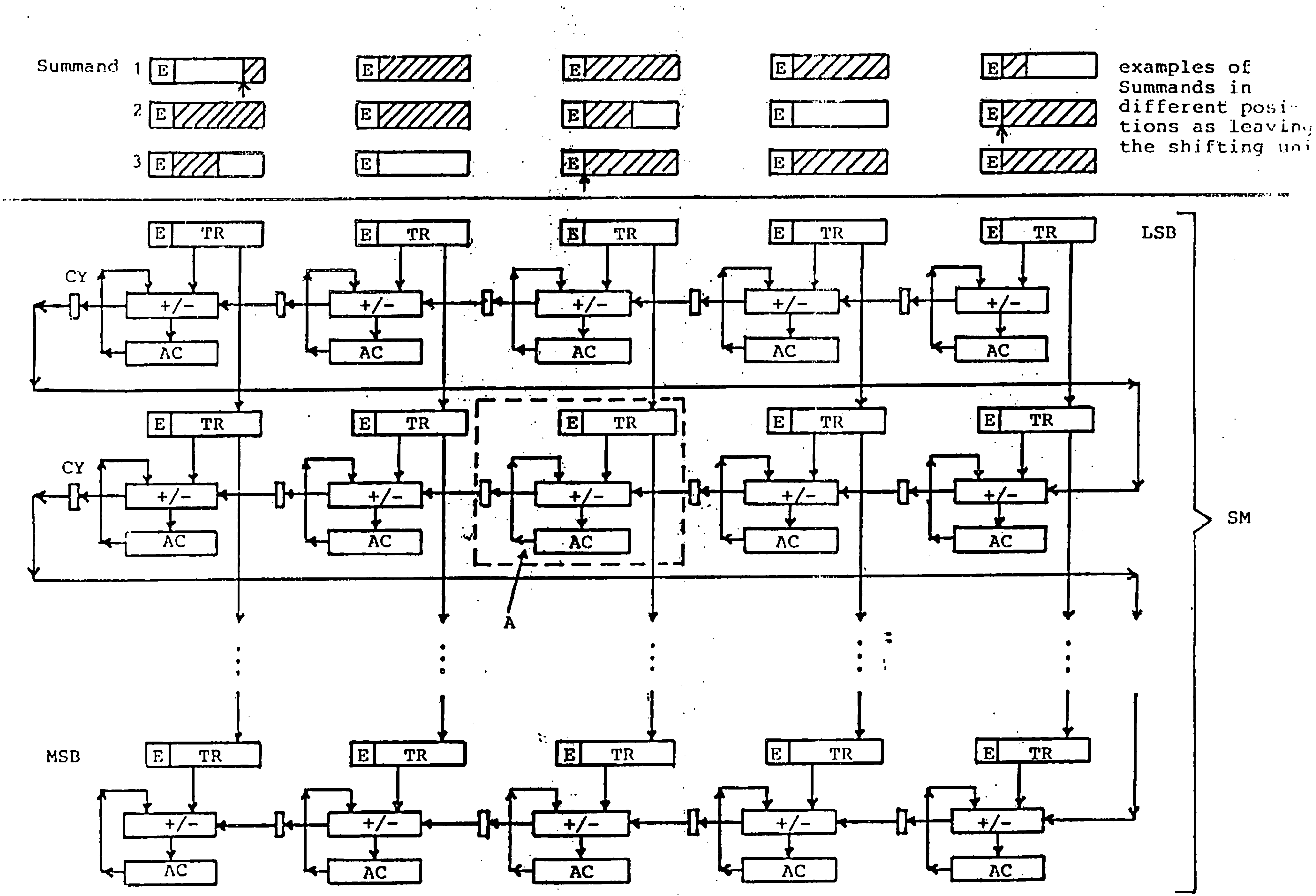


Figure 6: Summing matrix SM consisting of h=c · r independent adders A
E: tag-register for exponent identification, TR: transfer register,
AC: accumulator register, CY: carry, ↑: most significant digit of summand

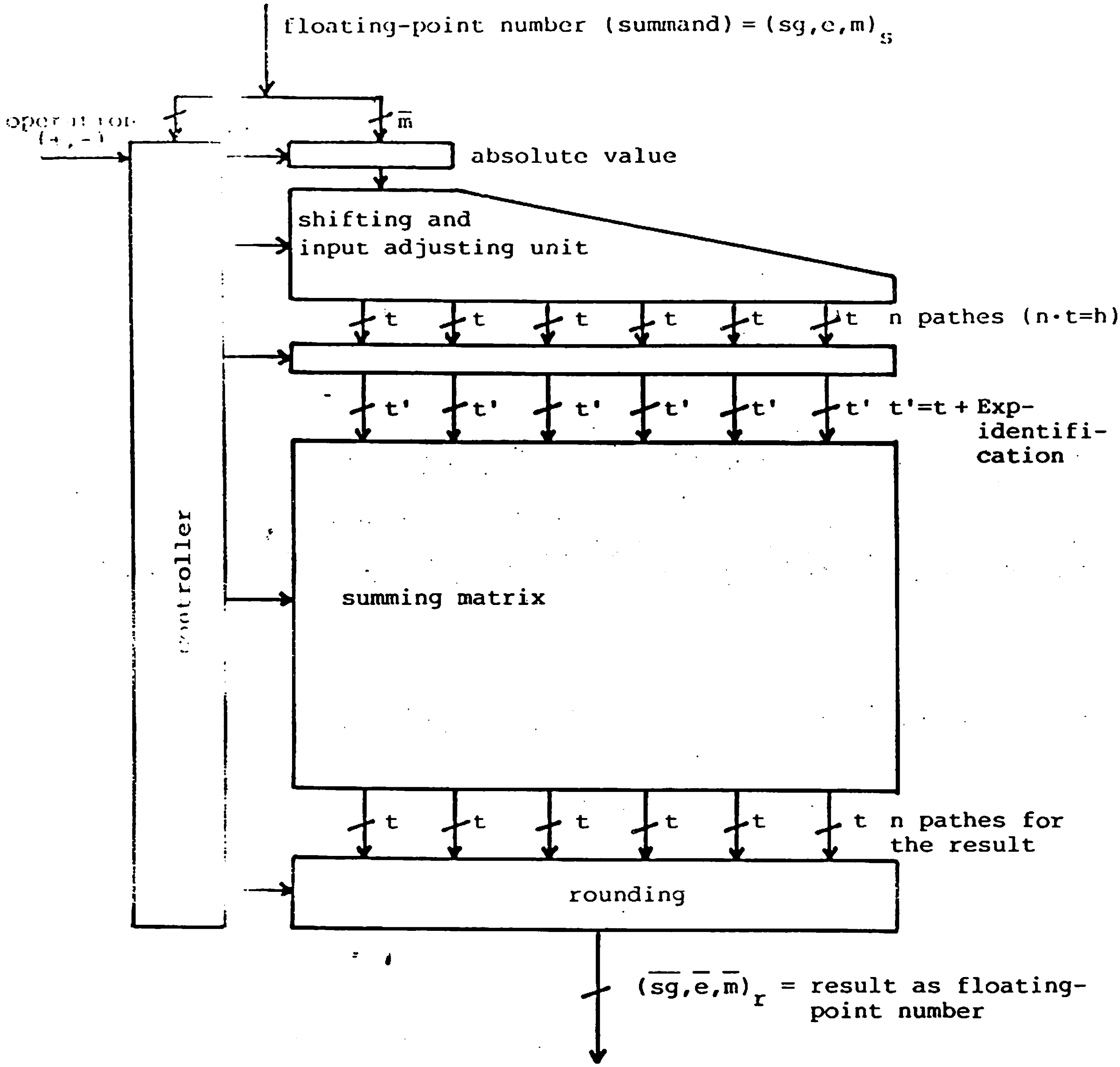
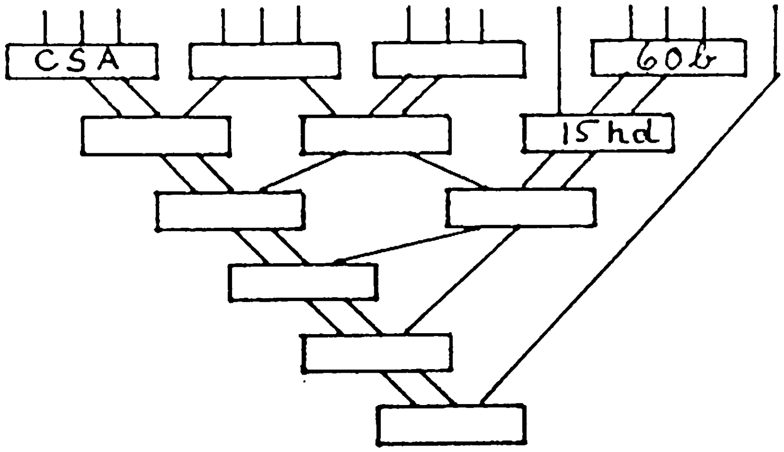
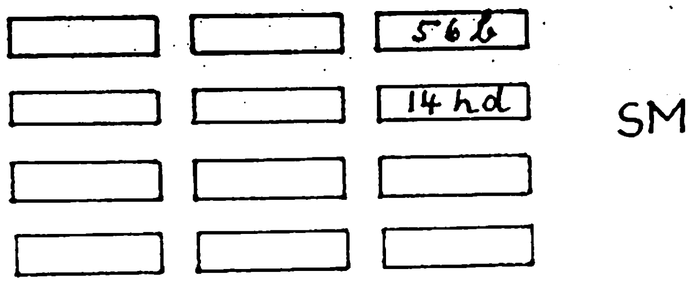


Figure 4: Structure of the whole circuitry
$\overline{\text{value}}$ denotes the number of figures of 'value'



W.Tr.: $15 \times 12 = \underline{\underline{180 \text{ hd}}}$ 720 b



SM: $64+28+64 = \underline{\underline{156 \text{ hd}}} < 42 \times 4 = 168 \text{ hd}$

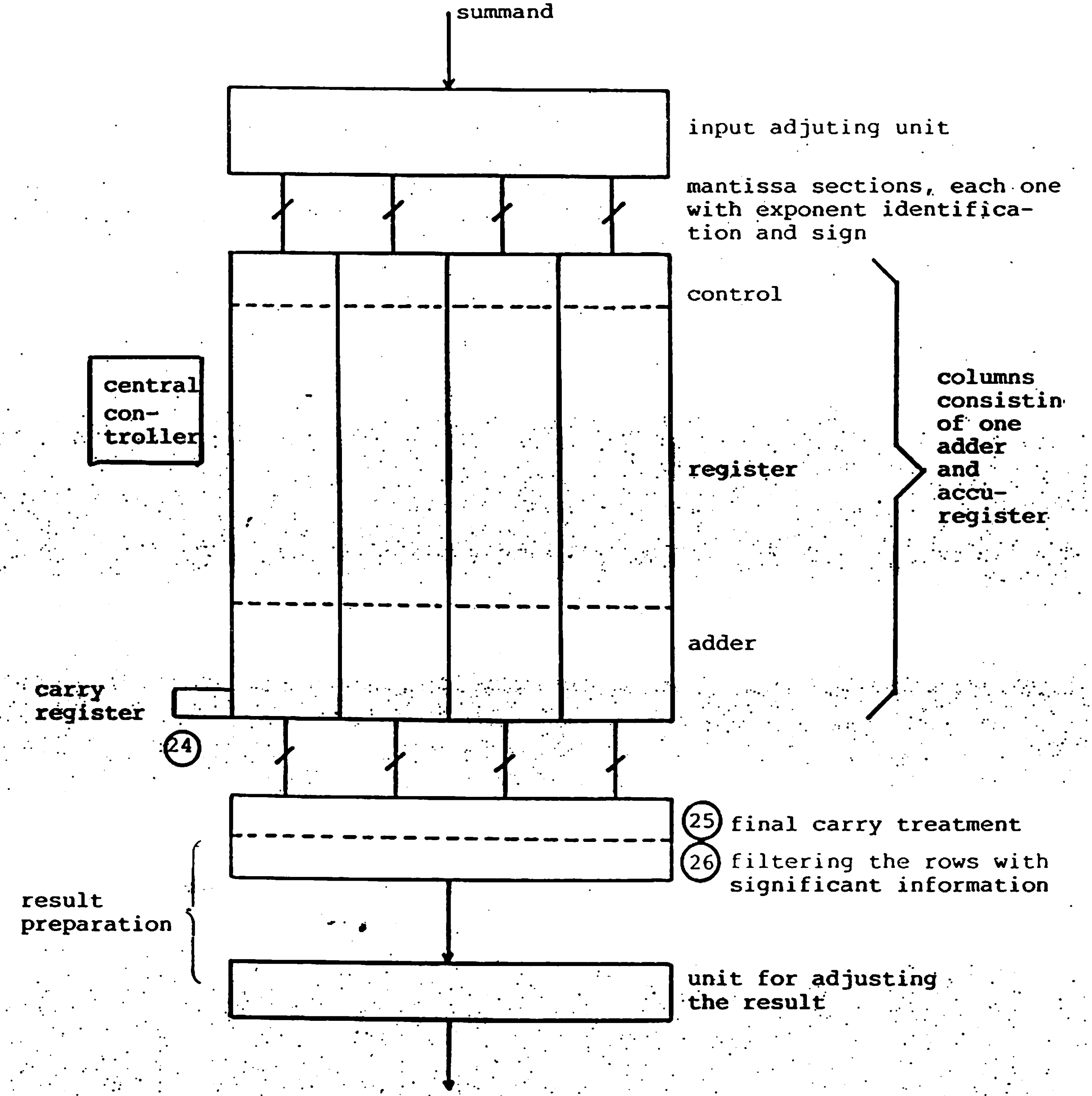


Figure 16: Structure of the summing unit with only one row of adders

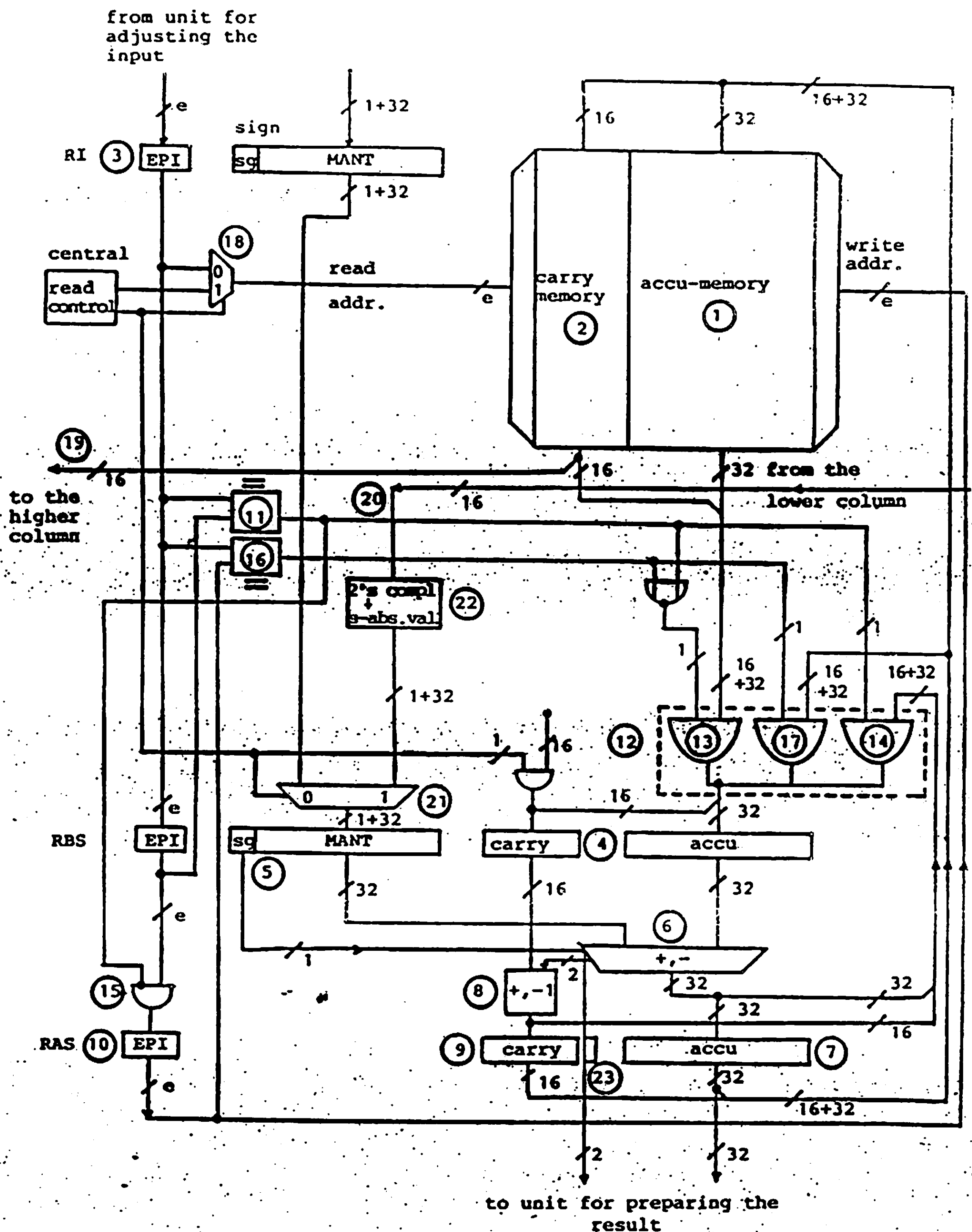


Figure 17: Structure of a "column" of the addition unit

Mehrfach genaue Operationen

lassen sich mit optimalem Skalarprodukt schnell und fehlerfrei ausführen:

1. doppelt lange Arithmetik

 1.1 Summe und Differenz

 $a+b$

 $a+b+c+\dots+z$

 Summen von Matrizen ebenso

 1.2 Produkt

 $a\cdot b=(a_1+a_2)(b_1+b_2)=a_1b_1+a_2b_2+a_1b_2+a_2b_1$

 $a\cdot b\cdot c\cdot d=$ [redacted] $=\sum_{i=1}^{8} a_i\cdot\sum_{j=1}^{8} c_i=\sum_{i=1}^{8}\sum_{j=1}^{8} a_i c_j$

 Produkte von Matrizen

2. dreifach lange Arithmetik

3. vierfach lange Arithmetik

 3.1 Summe und Differenz

 $a+b=a_1+a_2+b_1+b_2$

 $a+b+c+\dots+z=a_1+a_2+b_1+b_2+c_1+c_2+\dots+z_1+z_2$

 Summen von Matrizen ebenso

 3.2 Produkt

 $a\cdot b=(a_1+a_2+a_3+a_4)(b_1+b_2+b_3+b_4)=\sum_{i=1}^{4}\sum_{j=1}^{4} a_i b_j$

 $a\cdot b\cdot c\cdot d=(a\cdot b)\cdot(c\cdot d)=\left(\sum_{i=1}^{4}\sum_{j=1}^{4} a_i b_j\right)\left(\sum_{i=1}^{4}\sum_{j=1}^{4} c_i d_j\right)=$

 $=\left(\sum_{i=1}^{32} a^i\right)\left(\sum_{j=1}^{32} c^j\right)=\sum_{i=1}^{32}\sum_{j=1}^{32} a^i\cdot c^j$

 Produkte von Matrizen

Polynomial and Arithmetic Expression Evaluation:

$$p(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 = ((a_3 t + a_2)t + a_1)t + a_0$$

$$a_0, a_1, a_2, a_3, t \in R$$

$$\begin{aligned} x_1 &= a_3 \\ x_2 &= x_1 t + a_2 \\ x_3 &= x_2 t + a_1 \\ x_4 &= x_3 t + a_0 \end{aligned} \qquad \begin{array}{rrrrl} x_1 & & & & = a_3 \\ -t x_1 + & x_2 & & & = a_2 \\ & -t x_2 + & x_3 & & = a_1 \\ & & -t x_3 + & x_4 & = a_0 \end{array}$$

$$A x = b, \text{ with } A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -t & 1 & 0 & 0 \\ 0 & -t & 1 & 0 \\ 0 & 0 & -t & 1 \end{pmatrix}, \; x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \; b = \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$p(t) = x_4$$

Arithmetic Expressions: $a, b, c, d, e \in R$

$(a+b)\cdot c - d/e$

$$\begin{aligned} x_1 &= a \\ x_2 &= x_1 + b \\ x_3 &= c \cdot x_2 \\ x_4 &= d \\ e \cdot x_5 &= x_4 \\ x_6 &= x_3 - x_5 \end{aligned}$$

$(a+b)(c+d)$

$$\begin{aligned} x_1 &= a \\ x_2 &= x_1 + b \\ x_3 &= c \\ x_4 &= x_3 + d \\ x_5 &= x_2 \cdot x_4 \end{aligned}$$

All such systems: simple form;
can be solved by non linear system solv. technique
or transfered into a linear system by an
algebraic transformation process

Step: diadic to multiadic operations
with maximum accuracy

Linear Systems of Equations $\quad A \cdot x = b$

$\hat{x}$: solution ; $\tilde{x}$: approximation ; $e = \hat{x} - \tilde{x}$: error ;

$$\begin{array}{l} b - A\tilde{x} = d \\ \underline{b - A\hat{x} = 0} \\ \quad A e = d \end{array}$$

: defect can be computed with full accuracy

If $e = \hat{x} - \tilde{x} \in E \Rightarrow \hat{x} \in \tilde{x} + E$.

Interval iteration scheme:

$$E_{n+1} := (J - RA)\, E_n + R d \qquad (*)$$

converges for every $E_0 \in V_n \mathbb{I}\mathbb{R}$ to the unique fixed poi[nt]

iff $\qquad \varrho(|J - RA|) < 1 \qquad$ (contraction)

not easy to verify.

Retraction easier to verify

$$E_{n+1} \subset \mathring{E}_n$$

$\Rightarrow$ R and A not singular and $e \in E_{n+1}$

$$\Rightarrow \quad \hat{x} \in \tilde{x} + E_{n+1}$$

Choose R as an approximate inverse of A;

then $\varrho(|J - RA|) < 1$ practically always holds.

E_0 is obtained by adding a small interval to $\tilde{x}$;

then usually $E_{n+1} \subset \mathring{E}_n$ after one or two steps.

(*) is very sensitive towards roundings;

round as little as possible; apply opt. scalar product