# A digit-serial divider for fine grain heterogeneous parallel-pipelined processing

Mario Fiallos-Aguilar, Jean Duprat

*Laboratoire de l'Informatique du Parallélisme*

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398


# *A digit-serial divider for fine grain heterogeneous parallel-pipelined processing*


Mario Fiallos Aguilar
Jean Duprat


septembre 1993


Research Report N$^{\text{O}}$ 93-26



**Ecole Normale Supérieure de Lyon**
46, Allée d'Italie, 69364 Lyon Cedex 07, France,
Téléphone : + 33 72 72 80 00; Télécopieur : + 33 72 72 80 80;
Adresses électroniques :
lip@frensl61.bitnet; lip@lip.ens−lyon.fr (uucp).

# A digit-serial divider for fine grain heterogeneous parallel-pipelined processing

Mario Fiallos Aguilar
Jean Duprat

septembre 1993

**Abstract**

We design a new radix 2 *digit on-line* (i.e., serial, most significant digit first) floating-point divider which performs its arithmetic operation in *digit on-line* mode both for the exponent and the mantissa. We have performed parallel discrete-event simulations of the circuit on a memory-distributed massively parallel computer.

**Keywords:** fine grain parallelisme, heterogeneous processing, digit on-line computation.

**Résumé**

Ce document décrit un diviseur "en-ligne" en vírgule flottante fonctionant en base 2. L'exposant comme la mantisse sont transmis chiffre à chiffre. Des simulations parallèles d'évènements discrets du circuit ont eté effectuées sur une machine parallèle à mémoire distribuée.

**Mots-clés:** parallélisme à ganularité fine, calcul hétérogène, calcul en-ligne.

# A digit-serial divider for fine grain heterogeneous parallel-pipelined processing*

Mario Fiallos Aguilar† and Jean Duprat
Laboratoire de l'Informatique du Parallélisme (LIP)
Ecole Normale Supérieure de Lyon.
46, Allée d'Italie 69364 Lyon cedex 07, France
mfiallos@lip.ens-lyon.fr

**Résumé**

We design a new radix 2 *digit on-line* (i.e., serial, most significant digit first) floating-point divider which performs its arithmetic operation in *digit on-line* mode both for the exponent and the mantissa. We have performed parallel discrete-event simulations of the circuit on a memory-distributed massively parallel computer.

## 1 Introduction

On-line arithmetic is a radical departure from conventional techniques for performing scientific computations [2],[5],[6],[12]. In such arithmetic, the digits circulate serially, most significant digit first. Since in classical (i.e. non redundant) number systems, carries are propagated from the least significant digit to the most significant one, *digit on-line* computations are not possible in these systems. Then, we need to use a redundant number system, which enables carry-free computations. Here, we use the BS ("borrow save") notation [7] which is a special bit-level implementation of the binary signed-digit representation [1].
The *digit on-line* arithmetic operators are characterized by their *delay*, that is the number $\delta$ such that $p$ digits of the result are deduced from $p + \delta$ digits of the input operands. When successive *digit on-line* operations are performed in digit pipelined mode, the resulting delay will be the sum of the individual delays of operations and communications, and the computation of large numerical jobs can be executed in an efficient manner. Here, we will assume that any communication has a delay of 1.
As we can see from figure 1, the computations in *digit on-line* mode can be described as a *dataflow graph, DFG.* These graphs consist of nodes, which indicate operations executed on arithmetic units, and edges from one node to another node, which indicate the flow of data between them. A nodal operation can be executed only when the required information, a digit from all the input edges is received. Typically a nodal operation requires one or two operands and produces one result. Once the node has been activated and the computations

*. This work is part of a project called CARESSE which is partially supported by the "PRC Architectures Nouvelles de Machine" of the French Ministère de la Recherche et de la Technologie and the Centre National de la Recherche Scientifique.
†. Supported by CNPq and Universidade Federal do Ceará, Brazil.

related to the input digits inside the arithmetic unit performed (i.e. the node has fired), the output digit is passed to the destination nodes. This process is repeated until all nodes have been activated and the final result obtained. Of course, more than one node can be *fired* simultaneously.

In this paper, we deal with the *digit on-line* floating-point implementation of the division.

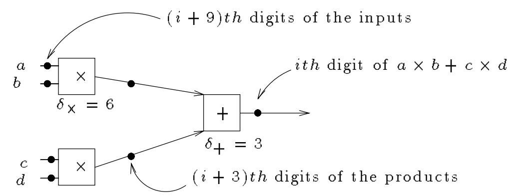


FIG. 1 - *Digit-level pipelining in digit on-line arithmetic*

We shall assume that both the exponents and the mantissas of numbers circulate in *digit on-line* mode and are represented in the BS system. We have already introduced *digit on-line* floating-point adders and multipliers [3], [4]. Recently, Tu[10], [11] has studied floating-point implementations of *digit on-line* operators, but in a slightly different manner: he assumes that the exponents enter the operators in parallel.

# 2 The BS notation and the number format

## 2.1 The BS notation

An interesting implementation of a radix-2 carry-free redundant system is Borrow Save notation, *BS* for short. In *BS*, the $i^{th}$ digit $x_i$ of a number $x$ is represented by two bits $x_i^+$ and $x_i^-$ with $x_i = x_i^+ - x_i^-$. Then 0 has two representations, (0 0) and (1 1). The digit 1 is represented by (1 0) and the digit $-1$ (or $\overline{1}$) by (0 1). Using the BS number system, the addition can be computed without carry propagation [7]. Figure 2 shows some elementary fixed-point $BS$ circuits.

## 2.2 Floating-point number format

A *BS* floating-point number $X$ with $n$ digits of mantissa and $p$ digits of exponent is represented by $X = mx2^{ex}$, where $mx = \sum_{i=1}^{n} mx_i 2^{-i}$ and $ex = \sum_{i=0}^{p-1} ex_i 2^i$. In our system the exponents and the mantissas circulate in *digit on-line* mode, exponent first. See figure 3.

## 2.3 Pseudo-normalization

In classical binary floating-point representation, a number is said normalized if its mantissa belongs to $[1/2, 1[$ or $]\overline{1}, \overline{1}/2]$. Normalization of numbers leads to more accurate representations and consequently results. In $BS$ representation, to check if a number is normalized needs sometimes the examination of all its digits. For this reason, we adopt the concept of pseudo-normalized numbers. A number is said pseudo-normalized if its mantissa belongs to

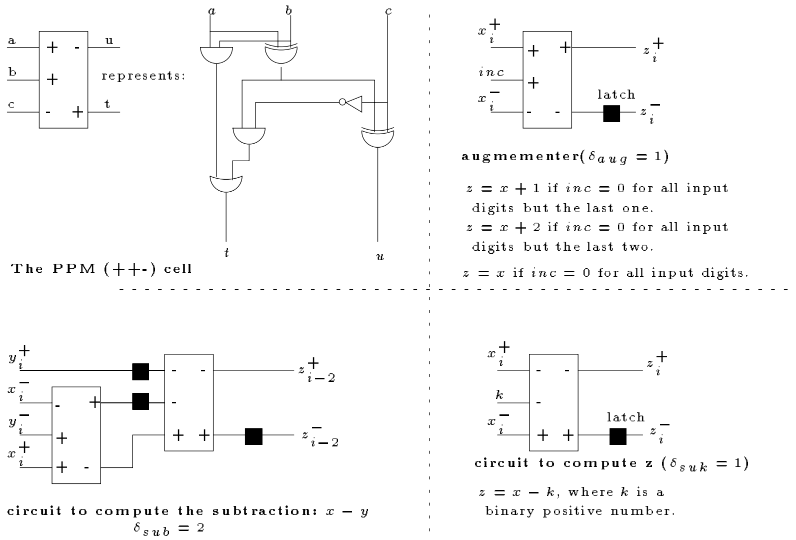


FIG. 2 - *Some elementary fixed-point BS circuits*

$$ex_{p-1} \cdots ex_0 \; mx_1 \cdots mx_n$$

<- - - - - - - - - - - - - - - - - - - - - - - -

*circulation of digits*

FIG. 3 - *The BS floating-point format*

$[1/4, 1[$ or $]\overline{1}, \overline{1}/4]$. It is easier and faster to ensure that a number is pseudo-normalized: it suffices to forbid a mantissa beginning by 01, $0\overline{1}$, $\overline{1}1$ or $1\overline{1}$. This pseudo-normalization is performed in two steps:

1. A four state automaton examines two consecutives digits and transforms the couples $(1\,\overline{1})$ and $(\overline{1}\,1)$ into $(0\,1)$ and $(0\,\overline{1})$ respectively and leaves the other couples unchanged. We call this operation an atomic pseudo-normalization. This automaton is shown in figure 4.

2. The second step consists in counting the zeroes generated by the previous computation and adding the same quantity to the exponent.

The divider could have a smaller delay if the divisor is guaranteed to be pseudo-normalized. In this case the output of all arithmetic operators (adders, multipliers, dividers, etc), must be pseudo-normalized.
But, as our principal goal is to perform computations in digit-level pipelined mode, it is preferable to pseudo-normalizer the inputs of the divider internally.
Note that the first solution makes the subtraction a variable delay operation. The second ones make the divider more complex, but allows the adders to have a fix *digit on-line* delay. This last solution is preferable because the division is less frequent than the addition is

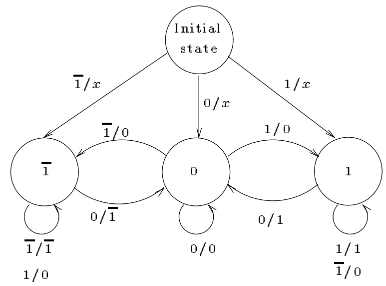


FIG. 4 - *The automaton of the pseudo-normalizer*

scientific computation.

# 3 The digit on-line algorithm

The *digit on-line* floating-point division algorithm performs three operations: exponents calculation, mantissas centring and calculation. A synchronization is performed between the exponent and the mantissa. The algorithm part of mantissa computation is based on the algorithm presented in [6]. Let us present the algorithm.

## 3.1 The algorithm

We want to compute $Q = X/Y$ with $X = mx2^{ex}$, $Y = my2^{ey}$, $Q = mq2^{eq}$ and

$$\begin{aligned} 1/4 \leq my < 1 \\ |mx| \leq my \end{aligned}$$

We will see how to deal with the cases of $mx > my$ and negative divisor mantissa in the next sections. The algorithm can be stated as follows:

**Algorithm 1 (Digit on-line division algorithm)**

`Step 1 (Exponent computation)`

1. *Compute the subtraction of the exponents but its last two digits:* $eq_{p-1}, \cdots, eq_2$.

`Step 2 (Mantissas shifting and exponent computation)`

1. $MY_0^{'} = \sum_{i=1}^{5} my_i 2^{-i}$;
2. $A_0^{''} = \sum_{i=1}^{5} mx_i 2^{-i}$;
3. *if* $MY_0^{'} < 1/2$ *then* $MY_0 = 2 \times MY_0^{'}$; *else* $MY_0 = MY_0^{'}$;
4. *if* $(|A_0^{''}| + 1/32 \geq MY_0 - 1/32)$ *then* $A_0^{'} = A_0^{''}/2$; *else* $A_0^{'} = A_0^{''}$;

5. *if* $A_0^{'} = A_0^{''}/2$ *then increment* $eq$ *and compute* $eq_1$;

6. *if* $(|A_0^{'}| + 1/32 \geq MY_0 - 1/32)$ *then* $A_0 = A_0^{'}/2$; *else* $A_0 = A_0^{'}$;

7. *if* $A_0 = A_0^{'}/2$ *or* $MY_0 = 2 \times MY_0^{'}$ *then increment* $eq$ *and compute* $eq_0$;

Step 3 (Mantissa computation)

1. *for* $(j = 0; \quad j \leq n-1)$
   {
   1.1 *if* $A_j \geq 1/8$ *then* $mq_{j+1} = 1$;
   *else if* $A_j \leq -1/8$ *then* $mq_{j+1} = -1$;
   *else* $mq_{j+1} = 0$;
   1.2 *if* $MY_0^{'} < 1/2$ *then*
   {
   -- $MY_{j+1} = MY_j + my_{j+6}2^{-j-5}$;
   -- $A_{j+1} = 2A_j + mx_{j+6}2^{-5} - mq_{j+1}MY_{j+1} - Q_j my_{j+6}2^{-4}$;
   }
   *else*
   {
   -- $MY_{j+1} = MY_j + my_{j+6}2^{-j-6}$;
   -- $A_{j+1} = 2A_j + mx_{j+6}2^{-5} - mq_{j+1}MY_{j+1} - Q_j my_{j+6}2^{-5}$;
   }
   1.3 $Q_{j+1} = Q_j + mq_{j+1}2^{-j-1}$;
   }

## 3.2 Proof of correctness

It is obvious that the computation of the exponent of the result is correct. On the other hand, for the mantissas alignment and computation the situation is more complex. Let us explain this.

### 3.2.1 Mantissas shifting

We show why it may be necessary to shift $A_0^{''}$ and $A_0^{'}$ one time each.
According to the algorithm it must be guaranteed that $|m_x| \leq m_y$. Then, as the shift must be performed with only 5 digits of each mantissa, we may have the following situations:

- If $MY_0^{'} \geq 1/2$, $\frac{|A_0^{''}|}{MY_0} = \frac{0.11111}{0.10000}$ and, $\frac{mx}{my}$ may be equal to $\frac{0.11111\cdots\infty}{0.100000\overline{1}\cdots\infty}$. A shift is necessary. But as $\frac{|A_0^{'}|}{MY_0} = \frac{0.01111}{0.10000}$ another shift is necessary and then, $\frac{|A_0|}{MY_0} = \frac{0.00111}{0.10000}$. With this, it is guaranteed that $|m_x| \leq m_y$.

- If $MY_0^{'} < 1/2$ then, $MY_0^{'}$ is shifted of one position. The worst case is: $\frac{|A_0^{''}|}{MY_0} = \frac{0.11111}{1.0\overline{111}}$. Then, it is enough to shift $A_0$ one position to guaranteed that $|m_x| \leq m_y$. With this $MY/2 \geq 15/64$. Where, $MY$ is the mantissa of the divider.

Then, the exponent must be augmented in 0, 1 or 2.

### 3.2.2 Mantissa computation

To perform the division correctly, the values of $mq_{j+1}$ chosen in step 2 of the algorithm must be compatibles with the Robertson's conditions [9]. They are:

1. if $MX_j < -MY/2$ then $mq_{j+1} = \overline{1}$.
2. if $-MY/2 \leq MX_j < 0$ then $mq_{j+1} = \overline{1}$ or $mq_{j+1} = 0$.
3. if $MX_j = 0$ then $mq_{j+1} = \overline{1}$ or $mq_{j+1} = 0$ or $mq_{j+1} = 1$.
4. if $0 < MX_j \leq MY/2$ then $mq_{j+1} = 0$ or $mq_{j+1} = 1$.
5. if $MX_j > MY/2$ then $mq_{j+1} = 1$.

The two following equations may be easily proved by induction.
If $MY_0^{'} \geq 1/2$:

$$A_j = 2^j \left[ \sum_{i=1}^{j+5} mx_i 2^{-i} - \left( \sum_{i=1}^{j} mq_i 2^{-i} \right) \left( \sum_{i=1}^{j+5} my_i 2^{-i} \right) \right] \tag{1}$$

else if $MY_0^{'} < 1/2$:

$$A_j = 2^j \left[ \sum_{i=1}^{j+5} mx_i 2^{-i} - \left( \sum_{i=1}^{j} mq_i 2^{-i} \right) \left( \sum_{i=1}^{j+5} my_i 2^{-i+1} \right) \right] \tag{2}$$

$A_j$ can be expressed also as:

$$A_j = 2^j \left( \sum_{i=1}^{j+5} mx_i 2^{-i} - \left( \sum_{i=1}^{j} mq_i 2^{-i} \right) MY_j \right) \tag{3}$$

$MY_j$ is the shifted mantissa of the divisor at step $j$.
We define a sequence as:

$$\left\{ \begin{array}{l} MX_0 = mx \\ MX_{j+1} = 2MX_j - mq_{j+1} MY \end{array} \right. \tag{4}$$

We find that:

$$MX_j = 2^j \left[ \sum_{i=1}^{n} mx_i 2^{-i} - \left( \sum_{i=1}^{j} mq_i 2^{-i} \right) MY \right] \tag{5}$$

$$MX_j - A_j = 2^j \left[ \sum_{i=j+6}^{n} mx_i 2^{-i} - \left( \sum_{i=1}^{j} mq_i 2^{-i} \right) \left( MY - MY_j \right) \right] \tag{6}$$

As:

$$MY_j = \left\{ \begin{array}{ll} \sum_{i=1}^{j+5} my_i 2^{-i} & \text{if } MY_0^{'} \geq 1/2 \\ \sum_{i=1}^{j+5} my_i 2^{-i+1} & \text{if } MY_0^{'} < 1/2 \end{array} \right\} \tag{7}$$

We have:

$$|MX_j - A_j \leq 2^j \left( \sum_{i=j+6}^{n} 2^{-i} + \left( \sum_{i=1}^{j} 2^{-i} \right) \left( |MY - MY_j| \right) \right) \tag{8}$$

As:

$$|MY - MY_j| \leq \left\{ \begin{array}{ll} 2^{-j}/32 & \text{if} MY_0' \geq 1/2 \\ 2^{-j}/16 & \text{if} MY_0' < 1/2 \end{array} \right\} \tag{9}$$

Then:

$$|MX_j - A_j| \leq 1/32 + 1/16 = 3/32 \tag{10}$$

According to step 3 of the algorithm:

- if $mq_{j+1} = 1$ then, $A_j \geq 1/8$. From equation 10 we find that if $A_j \geq 1/8$ then $MX_j \geq 1/32$. Robertson's conditions 4 and 5 are satisfied.
- Similarly, if $mq_{j+1} = \overline{1}$ then $A_j \leq \overline{1}/8$. Then, $MX_j \leq \overline{1}/32$. Robertson's conditions 1 and 2 are satisfied.
- if $mq_{j+1} = 0$, then, $\overline{4}/32 < A_j < 4/32$. From equation 10, we find that $\overline{7}/32 < MX_j < 7/32$ and as, $|MY|/2 \geq 15/64$ then, the Roberson's conditions 2, 3 and 4 are satisfied.

Hence, the algorithm computes the division correctly.
However, this algorithm can be improved. The sequence of tests:

**Test 1 (Test of $A_j$)**

-- *if* $A_j \geq 1/8$ ***then*** $mq_{j+1} = 1$
*else if* $A_j \leq -1/8$ ***then*** $mq_{j+1} = -1$
*else* $mq_{j+1} = 0$

needs the examination of all the digits of $A_j$ (i.e., $j+5$). This examination involves a needless loss of time (the arithmetic operations on step 3 of the algorithm may be performed in parallel, without carry propagation, using the BS number system). Therefore this sequence of test is the most time-consuming part of the algorithm. In order to avoid this drawback, we examine all the digits of $A_j$ between the most significant one and the digit which power is $2^{-5}$. Namely, $A_j^* = \sum_{i=0}^{5} 2^{-i} a_{j,i}$ [1]. Then, the test will be performed on $A_j^*$ instead of $A_j$ as following:

**Test 2 (Test of $A_j^*$)**

-- *if* $A_j^* \geq 1/8$ ***then*** $mq_{j+1} = 1$
*else if* $A_j^* \leq -1/8$ ***then*** $mq_{j+1} = -1$
*else* $mq_{j+1} = 0$

The proof of the improved algorithm is similar to the previous one:
We obtain the obvious relation:

$$|A_j - A_j^*| \leq 1/32 \tag{11}$$

Then, according to the modified Step 3 of the algorithm:

- if $mq_{j+1} = 1$ then, $A_j^* \geq 1/8$. From equation 11 we find that if $A_j^* \geq 1/8$ then, $A_j \geq 3/32$ and from 10 we find that $MX_j \geq 0$.

1. by now let us assume that $A_j^*$ can be represented as a 6 digits expression.

- Similarly, if $mq_{j+1} = \overline{1}$ then, $A_j^* \leq \overline{1}/8$. Then, $A_j \leq \overline{3}/32$ and $MX_j \leq 0$.
- if $mq_{j+1} = 0$, then, $\overline{1}/8 < A_j^* < 1/8$. As $A_j^*$ is a multiple of $1/32$, we have: $\overline{3}/32 \leq A_j^* \leq 3/32$. From equation 11 we find: $\overline{4}/32 \leq A_j \leq 4/32$ and, from equation 10, we find that $\overline{7}/32 \leq MX_j \leq 7/32$.

### 3.2.3 Pseudo-normalization

If the inputs of the floating-point divider are pseudo-normalized then its output is also pseudo-normalized. Let us prove that:

- If $MY_0^{'} \geq 1/2$ then, the worst case is: $\frac{|X|}{Y} = \frac{0.10\overline{1}\cdots\infty}{0.1\cdots\infty} = \frac{1}{4}$ and the quotient is pseudo-normalized.
- If $MY_0^{'} < 1/2$ then the worst case is: $\frac{|X|}{Y} = \frac{0.10\overline{1}\cdots\infty}{1.000\overline{1}1\cdots\infty} = \frac{1}{4}$ and the quotient is pseudo-normalized.

# 4 The architecture

The floating-point divider consists of several blocks (figure 5):

- A serial circuit to compute the difference between the exponents.
- A serial *augmenter* to increase the exponent by 0, 1 or 2.
- A serial automaton that computes the absolute value of $Y$.
- A serial overflow detector.
- A pseudo-normalizer, which ensures that $1/4 \leq Y < 1$.
- A serial shifter/synchronizer for the mantissas.
- A serial divider for the mantissas.

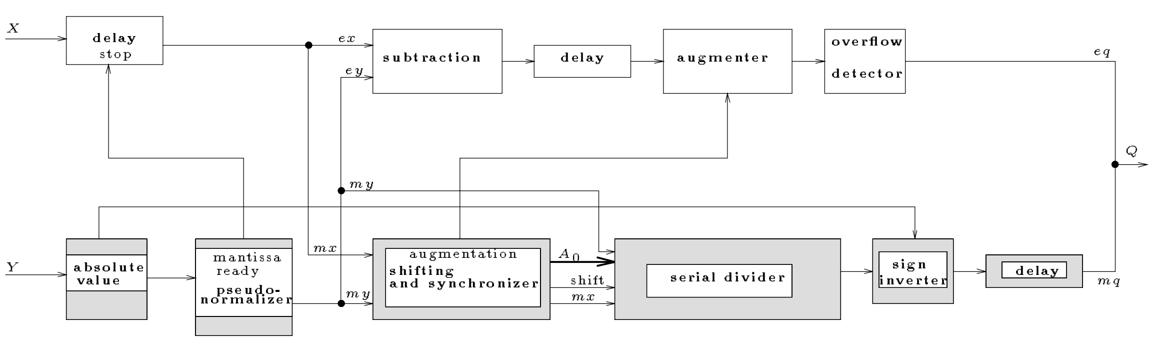


FIG. 5 - *The on-line floating-point divider*

The first two computations are performed with the circuits of figure 2.
The automaton that computes the absolute value of $Y$ is shown in figure 6. The sign inverter changes the sign of the mantissa of the result if the state of the maximum value automaton is $\overline{1}$.
The detection of the overflow is done at the output of the incrementer. A small automaton

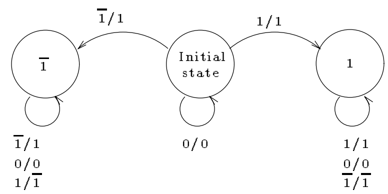


FIG. 6 - *The absolute value automaton*

tries to find a representation of the exponent so that to have the carry digit equal to 0 (in order to keep the $p$ exponent of the format). Figure 7 shows this automaton.

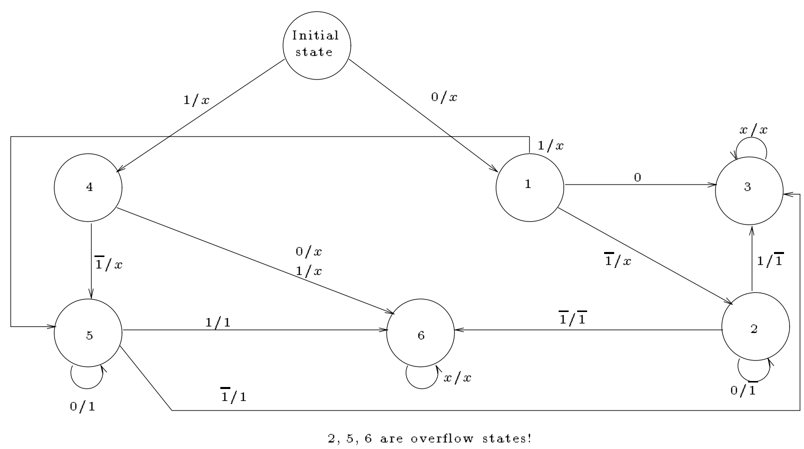


FIG. 7 - *The overflow detector automaton*

The shifter/synchronizer guarantees that if shifts have been performed, then the exponent is augmented and otherwise the exponent remains unchanged. We will explain with more detail the pseudo-normalizer, the shifter/synchronizer and the serial divider.

## 4.1 Pseudo-normalizer

The pseudo-normalizer is shown in figure 8. The automaton is shown in figure 4. A *binary counter* stores the number that the exponent must be decreased. A *zero tester* is used to avoid the delay of the serial circuit when the subtraction of the exponents is not performed. The overflow detector is similar to the ones shown in figure 7. The delay of the pseudo-normalizer ($\delta_{pno}$) is variable and depends on the degree of pseudo-normalization of the operands. If $le$ is the number of digits of the exponent and $lbs$ the number of digits to represent the floating-point number, then:

$$le + 1 \leq \delta_{pno} \leq lbs + 1 \tag{12}$$

Then the delay of the normalizer may be, in the worst case, as great as the length of the number representation plus 1. On the other hand, if the input operand is already pseudo-normalized, $\delta_{pno}$ has its minimum value. Figure 9 shows an example.
If the zero tester is not used a simplified design is obtained, but the minimum value of the delay will be augmented by 1. The serial subtraction can be replaced also by its parallel version.

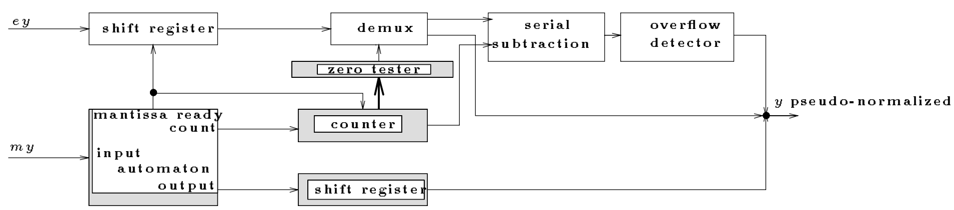


FIG. 8 - *The pseudo-normalizer*

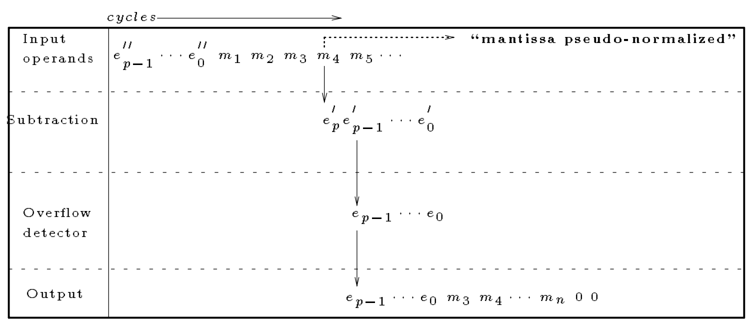


FIG. 9 - *Example of the internal synchronization on the pseudo-normalizer ($my = 0.0010\cdots$)*

## 4.2 Shifting the mantissas

The circuit performs the comparisons of the mantissas. The comparison on $MY_0^{'}$ is performed before the comparison with $mx$. A second comparison delays $mx$ of 1 or 2 cycles if necessary. None digit of $mx$ is lost, but delayed. It is assumed that these operations can be performed in one cycle.

## 4.3 The serial divider

The serial divider is shown in figure 11. The upper part of it computes the term $mq_{j+1}MY_{j+1}$. Similarly, the lower ones computes $Q_j m_{j+6}$. The $BS$ four-input parallel adder computes the

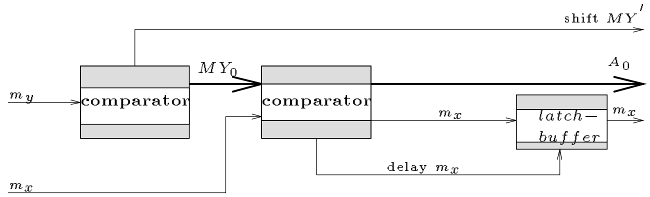


FIG. 10 - *The circuit for shifting the mantissas*

term $A_j$. It is made up with 3 2-input $BS$ parallel adders. A 2-input parallel adder is proposed in [7]. The format control is very simple and requires only the test of the digit with power $2^1$. If the value of this digit is different form zero, then the digit with power $2^0$ is inverted (remember, $|A_j| \leq 3/8$). This technique was originally proposed by Kla [8]:

– Let $Z = z_n \cdots z_1 z_0 . z_{-1} z_{-k} = N z_1 z_0 . K$ such that $|Z| \leq 1$.
  if $z_1 = 0 \Rightarrow Z = z_0.K$ else $Z = \overline{z}_0.K$

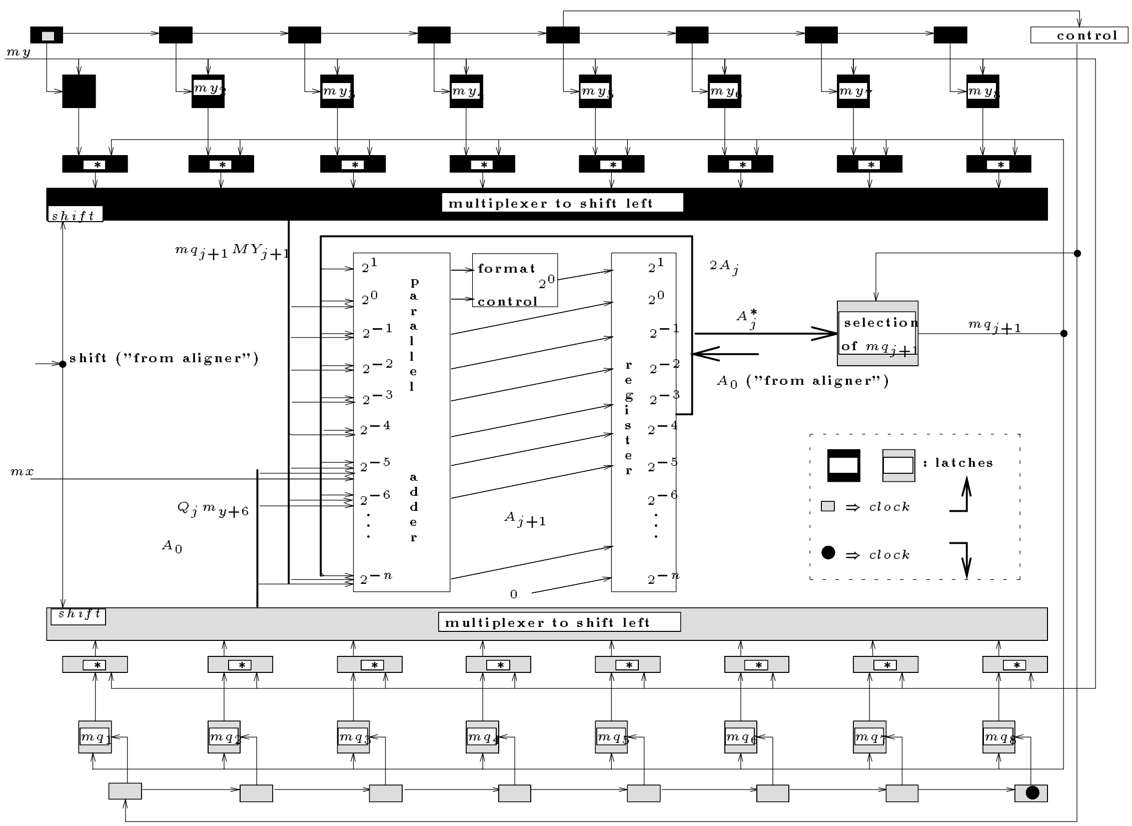


FIG. 11 - *The serial divider*

### 4.4 Internal synchronization of the floating-point divider

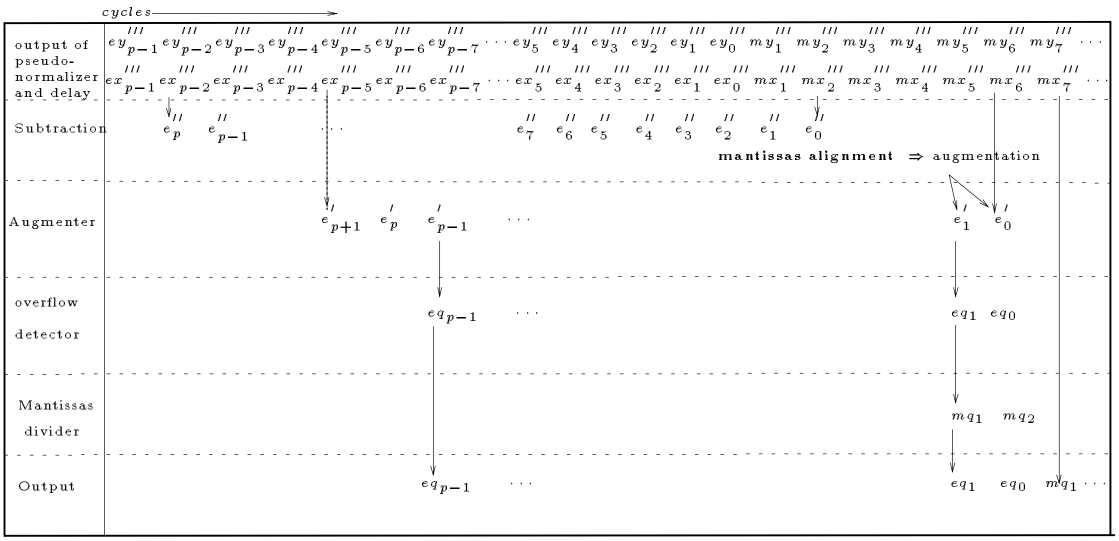


FIG. 12 - *The internal synchronization on the on-line floating-point divider*

As we can see from figure 12, the decision on augmenting or not the exponent can be taken when their last two digits go through the incrementer. As the last two digits of the exponent are outputting, the first five digits of the mantissas are available, and then it is possible to subtract 0, 1, or 2 from the exponent of the result. Using figures 9 and 12 we obtain the interval values of the *digit on-line* delay of the floating-point divider ($\delta_{div}$):

$$le + 7 \leq \delta_{div} \leq lbs + 7 \tag{13}$$

Note that if the inputs are guaranteed to be pseudo-normalized, the delay of the divider would be 6.

## 5 Conclusion

We have described a new radix 2 *digit on-line* divider. This arithmetic unit has a variable *digit on-line* delay which depends on the pseudo-normalization degree of the divisor.
This architecture is fully simulated using parallel discrete-event simulations. It works on MaPar MP-1, a memory-distributed massively parallel computer, where several operators work in parallel.
With this operator and the adders and multipliers already introduced, it is possible to perform in a digit-level pipelined mode, complex computations such as the Gauss elimination algorithm to solve linear equations.
We are working in a project to simulate and to build a *digit on-line* machine called CARESSE, the french abbreviation of Serial Redundant Scientific Computer, that will made up of heterogeneous *digit on-line* arithmetic units.

## Références


[1] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:pp 389–400, 1961.

[2] J. Duprat and M. Fiallos Aguilar. Datafow dot product on networks of heterogeneous digit-serial arithmetic units. In *submitted to IEEE 5th Symposium on Parallel and Distributed Processing (SPDP93)*, 1993.

[3] J. Duprat and M. Fiallos. On the simulation of pipelining of fully digit on-line floating-point adder networks on massively parallel computers. In *Second Joint Conference on Vector and Parallel Processing*, Lecture Notes in Computer Science, pages 707–712. Springer-Verlag, September 1992.

[4] J. Duprat, M. Fiallos, J. M. Muller, and H. J. Yeh. Delays of on-line floating-point operators in borrow save notation. In *Algorithms and Parallel VLSI Architectures II*, pages 273–278. Noth Holland, 1991.

[5] M.D. Ercegovac. On-line arithmetic: an overview. In SPIE, editor, *SPIE, Real Time Signal Processing VII*, pages pp 86–93, 1984.

[6] M.D. Ercegovac and K.S Trivedi. On-line algorithms for division and multiplication. *IEEE Trans. Comp.*, C-26(7):pp 681–687, 1977.

[7] A. Guyot, Y. Herreros, and J. M. Muller. Janus, an on-line multiplier/divider for manipulating large numbers. In *IEEE 9th Symposium on Computer Arithmetic*, pages 106–111. IEEE Computer Society Press, 1989.

[8] Sylvanus Kla. *Calcul Parallèle et En-Ligne des Fonctions Arithmétiques.* PhD thesis, Ecole Normale Supérieure de Lyon, France, February 1993.

[9] J.M. Muller. *Arithmetique des Ordinateurs.* Masson, 1989.

[10] P. K. Tu. *On-line Arithmetic Algorithms for Efficient Implementation.* PhD thesis, Computer Science Department, UCLA, 1990.

[11] P.K. Tu and M.D. Ercegovac. Design of on-line division unit. In *9th Symposium on Computer arithmetic*, pages 42–49. IEEE, 1989.

[12] R. J. Zaccone and J. L. Barlow. Eliminating the normalization problem in digit on-line arithmetic. *IEEE Transaction on Computers*, C-36(1):36–46, January 1987.