ALTERNATIVE ON-THE-FLY CONVERSION OF REDUNDANT INTO CONVENTIONAL REPRESENTATIONS

Milos D. Ercegovac
Tomas Lang

November 1986
CSD-860027

Alternative On-the-Fly Conversion of Redundant into Conventional Representations

Miloš D. Ercegovac and Tomas Lang
UCLA Computer Science Department
University of California, Los Angeles

Abstract

In a previous paper an algorithm to convert redundant number representations into conventional representations was presented. The conversion is performed concurrently with the digit-by-digit generation of redundant forms by schemes such as division, square root, and on-line operations in which redundantly represented results are generated in a digit-by-digit manner, from most significant to least significant. Here we discuss a variation of this algorithm that results in a more effective implementation for some on-line algorithms and combinational implementations.

1. Introduction

In a previous paper [1] we presented an algorithm to convert a signed fraction from a redundant (signed digit) into a conventional range-complement representation. Such a conversion is necessary, for example, in SRT division or square root algorithms that produce this redundant result [2,3,4,5,6], or to convert operands and results in on-line algorithms into the equivalent conventional forms [7,8,9,13]. The algorithm differs from the standard approach for the conversion, which consists of a carry-propagate addition of the positive and negative digits, since it is done on-the-fly while the digits of the redundant representation are produced from most to least significant and has a delay of approximately one carry-save adder.

The previously proposed algorithm generates two conditional forms, in a technique similar to that used in conditional-sum addition [10], and selects one of them when a nonzero digit is produced. Here we propose a variation of this algorithm which requires just one form and one control bit per digit. This variation has a more efficient implementation for higher radices and for combinational implementations of right-to-left algorithms, such as multiplication [12].

We first review the previously proposed algorithm and the corresponding implementation and then present the new variation and compare both.

2. Previous Algorithm and Implementation

We want to convert the fractional signed-digit representation

$$p = \sum_{i=1}^{m} p_i r^{-i}, \; p_i \in \{-a, \ldots, 0, \ldots, a\} \;\; r/2 \le |a| \le r-1 \tag{2.1}$$

into the conventional range-complement representation

$$q = -q_0 + \sum_{i=1}^{m} q_i r^{-i}, \; q_i \in \{0, \ldots, r-1\}, \;\; q_0 \in \{0,1\} \tag{2.2}$$

The digits of p are produced from most significant to least significant and the conversion is done on-the-fly as each digit is produced. Let us call $q[k]$ the converted fraction after k digits have been produced. The algorithm generates two conditional forms $A[k]$ and $B[k]$ such that

$$q[k] = A[k] \tag{2.3}$$

and

$$B[k] = A[k] - r^{-k} \tag{2.4}$$

This results in the following recurrence:

For $k > 1$

$$A[k+1] = \begin{cases} A[k] + p_{k+1} r^{-(k+1)} & \textbf{if } p_{k+1} \geq 0 \\ B[k] + (r - |p_{k+1}|) r^{-(k+1)} & \textbf{if } p_{k+1} < 0 \end{cases} \tag{2.5}$$

$$B[k+1] = \begin{cases} A[k] + (p_{k+1} - 1) r^{-(k+1)} & \textbf{if } p_{k+1} > 0 \\ B[k] + ((r-1) - |p_{k+1}|) r^{-(k+1)} & \textbf{if } p_{k+1} \leq 0 \end{cases} \tag{2.6}$$

with the initial condition

$$A[1] = \begin{cases} +p_1 r^{-1} \quad (0.p_1) & \textbf{if } p > 0 \\ -|p_1| r^{-1} \quad (1.(r - |p_1|)) & \textbf{if } p < 0 \end{cases} \tag{2.7}$$

$$B[1] = \begin{cases} +(p_1 - 1) r^{-1} \quad (0.(p_1 - 1))^* & \textbf{if } p > 0 \\ -(|p_1| + 1) r^{-1} \quad (1.(r - 1 - |p_1|)) & \textbf{if } p < 0 \end{cases} \tag{2.8}$$

* $p_1 > 0$ since p is normalized.

The algorithm is illustrated for $r = 4$ and $p_i \in \{-3, \ldots, 3\}$ in Figure 1.

k	p_k	$A[k]$	$B[k]$	$q[k]$
1	-2	1.2	1.1	1.2
2	0	1.20	1.13	1.20
3	3	1.203	1.202	1.203
4	-1	1.2023	1.2022	1.2023
5	0	1.20230	1.20223	1.20230
6	-1	1.202233	1.202232	1.202233
7	2	1.2022332	1.2022331	1.2022332
8	-3	1.20223311	1.20223310	1.20223311

Figure 1. Example

The sequential implementation of the algorithm requires two registers to hold $A[k]$ and $B[k]$, respectively. These registers are shifted left with insertion of a new digit. They also require parallel loading capabilities to load $A[k]$ with $B[k]$ and vice versa. This implementation is shown in Figure 2a.

In a combinational implementation, two vectors are kept and the selection is done using multiplexers as shown in Figure 2b.

3. Variation of the Algorithm and Implementation

The algorithm we present now is based on the following observations:

- In the previous algorithm the value of the j-th digit of $B[k]$ is equal or one less than the corresponding digit of $A[k]$. Consequently, it is not necessary to keep both forms, but just one form and one bit per digit to indicate whether both digits are the same or not.

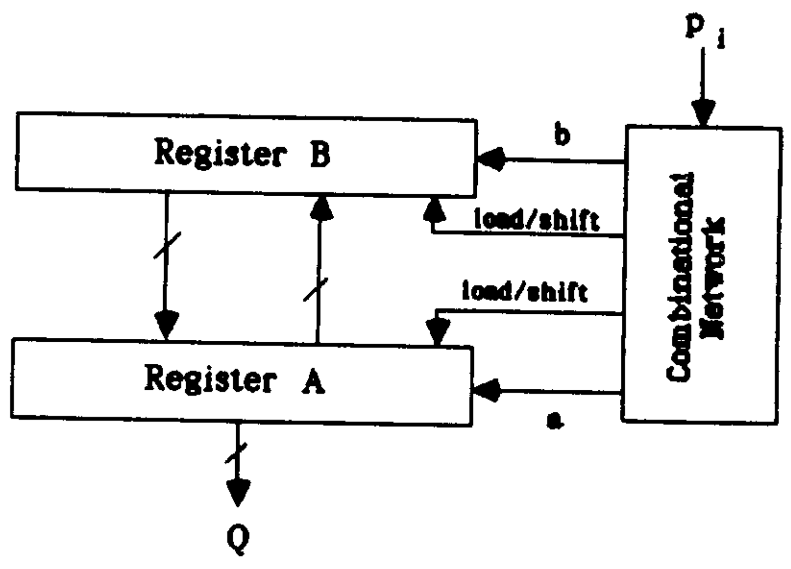

Figure 2a. Sequential Implementation of Basic Scheme

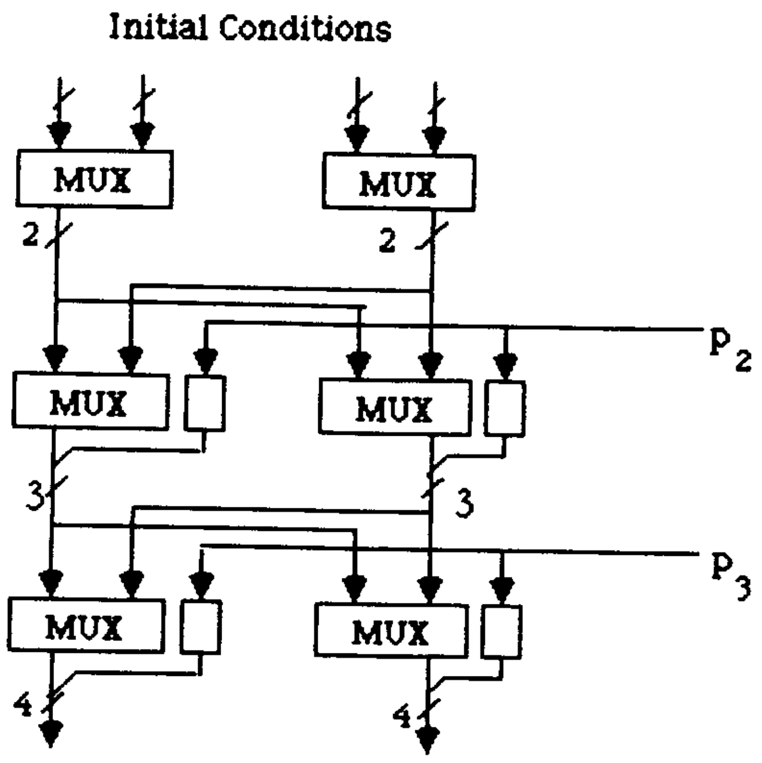

Figure 2b: Combinational Implementation of Basic Scheme

- Whenever $p_k \neq 0$ all digits, with exception of the least significant, of $A[k]$ are equal to the corresponding digit of $B[k]$ and will remain unchanged in later steps (we say that they are already resolved).

- To produce a digit of $A[k+1]$, the corresponding digit of $A[k]$ is decremented whenever it has not been resolved and $p_{k+1} < 0$.

According to these remarks, instead of keeping two conditional forms, it is sufficient to keep just $A[k]$ and one bit per digit indicating whether it has been already resolved. The algorithm is the following:

For $j \leq k$,

$$A_j[k+1] = \begin{cases} A_j[k] & \textbf{if } R_j[k]=1 \textbf{ or } p_{k+1} \geq 0 \\ (A_j[k]-1) \mathit{ mod } r & \textbf{if } R_j[k]=0 \textbf{ and } p_{k+1} < 0 \end{cases} \tag{3.1}$$

$$R_j[k+1] = \begin{cases} 1 & \textbf{if } R_j[k]=1 \textbf{ or } p_{k+1} \neq 0 \\ 0 & \textbf{if } R_j[k]=0 \textbf{ and } p_{k+1}=0 \end{cases} \tag{3.2}$$

For $j = k+1$,

$$A_{k+1}[k+1] = p_{k+1} \mathit{ mod } r \tag{3.3}$$

$$R_{k+1}[k+1] = 0 \tag{3.4}$$

An example is shown in Figure 3.

k	p_k	$A[k]$	$R[k]$
1	3	0.3	1.0
2	-2	0.22	1.10
3	1	0.221	1.110
4	0	0.2210	1.1100
5	0	0.22100	1.11000
6	-3	0.220331	1.111110
7	2	0.2203312	1.1111110
8	-1	0.22033113	1.11111110

Figure 3. Example of Alternative Scheme

The sequential implementation consists of two registers, containing $A[k]$ and $R[k]$, respectively. To each digit of $A[k]$ there is an associated decrementer. The decrementation is controlled by $R[k]$ and the sign of p_{k+1}. In addition, $R[k+1]$ is determined by $R[k]$ and the fact that $p_{k+1} \neq 0$. This implementation is shown in Figure 4a.

The combinational implementation is shown in Figure 4b. As indicated it requires $m^2/2$ decrementers. To avoid this, it is possible to postpone the decrementation until the last level. This requires another bit-vector, $D[k]$, which indicates whether the corresponding digit of $A[k]$ has to be decremented. The recurrence for $D[k]$ is

$$D_j[k+1] = \begin{cases} 1 & \textbf{if } D_j[k]=1 \textbf{ or } (R_j[k]=0 \textbf{ and } p_{k+1} \neq 0) \\ 0 & \textbf{otherwise} \end{cases} \tag{3.5}$$

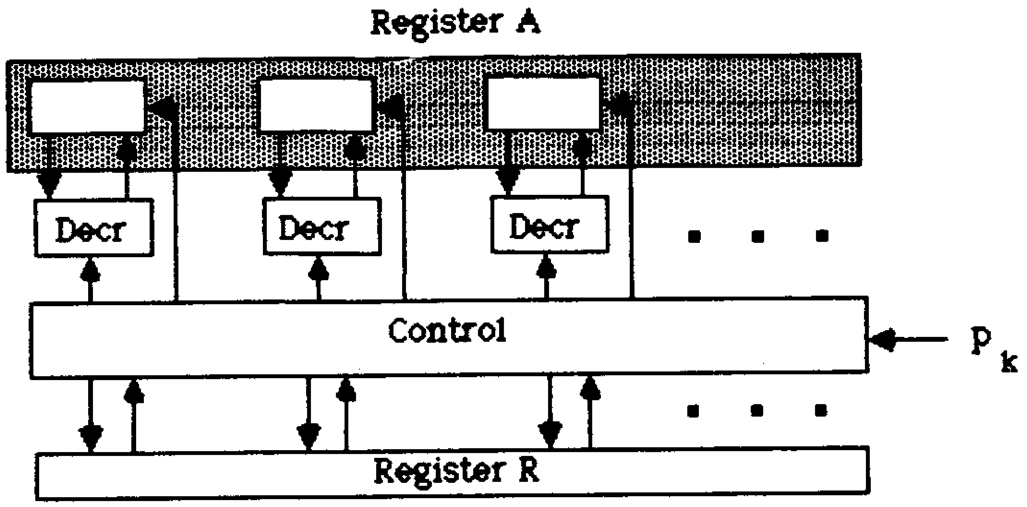

Figure 4a: Sequential Implementation

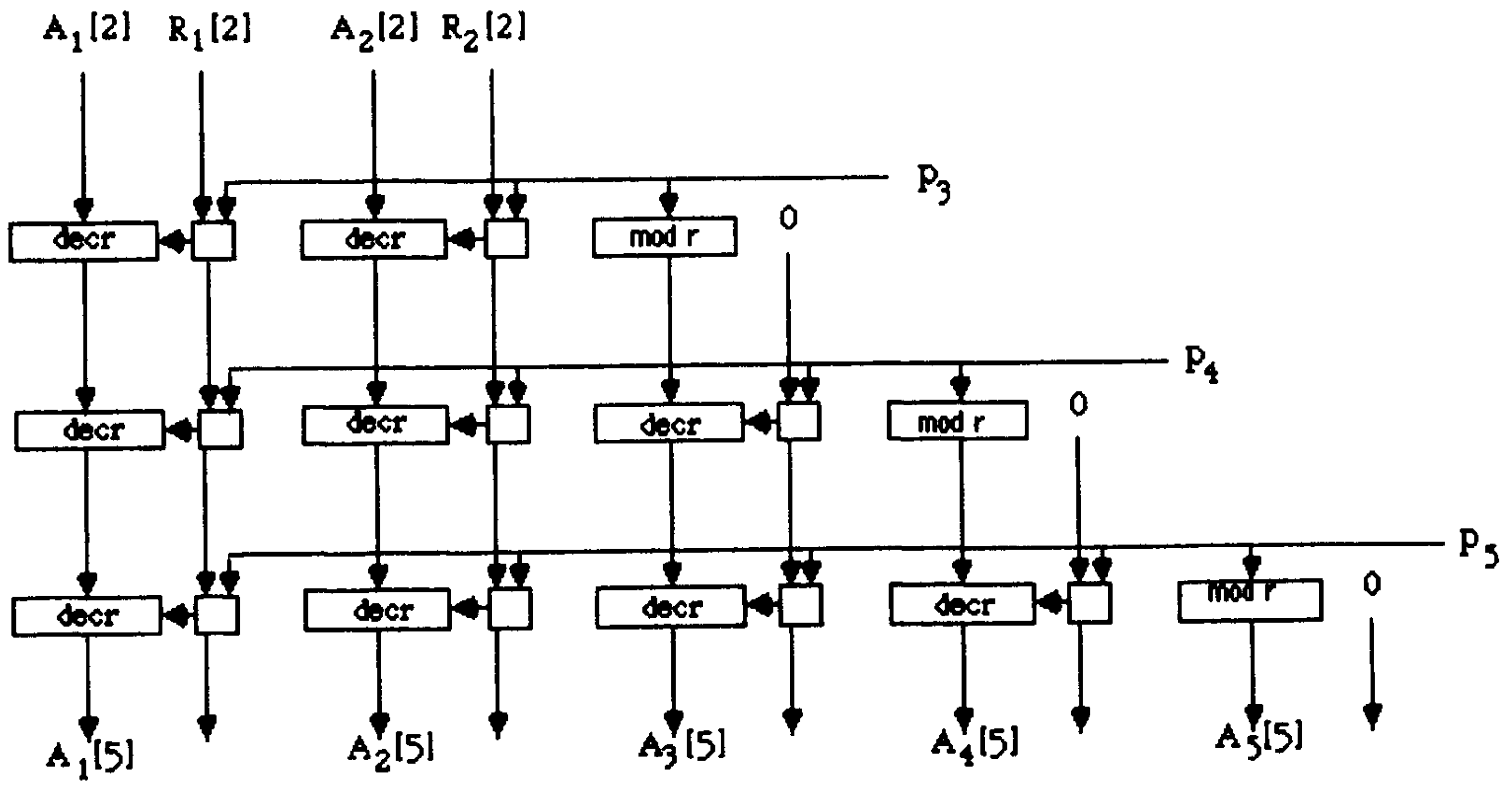

Figure 4b: Combinational Implementation
- A Segment

This implementation is shown in Figure 4c.

The comparison between Figures 2 and 4 shows the following:

- In the sequential case, we are replacing a "radix-r" register by a bit-register and a set of decrementers. This does not seem a good alternative. However, in some on-line algorithms [11] in which it is necessary to append the new digit instead of shifting, the decrementers have to be there anyhow. In such cases, the implementation produced by the new algorithm might be preferable.

- In the combinational case, the number of lines and the complexity of the cells is reduced. This has to be compared with the additional decrementers needed.

4. Summary

An algorithm for converting redundant forms into range complement conventional forms concurrently with digit-by-digit generation is presented. The algorithm has simple sequential and combinational implementations. In the sequential implementation, its delay (independent of the working precision), roughly equals two logic levels plus a register shift/load time. In the combinational case, there is a delay per stage equivalent to three gate delays. The algorithm is applicable in nonrestoring division and square algorithms, in producing conventional results in on-line algorithms, and in left-to-right multiplication [12].

Acknowledgements. This research has been supported in part by the Contract No. N00014-85-K-0159 of the Office of Naval Research.

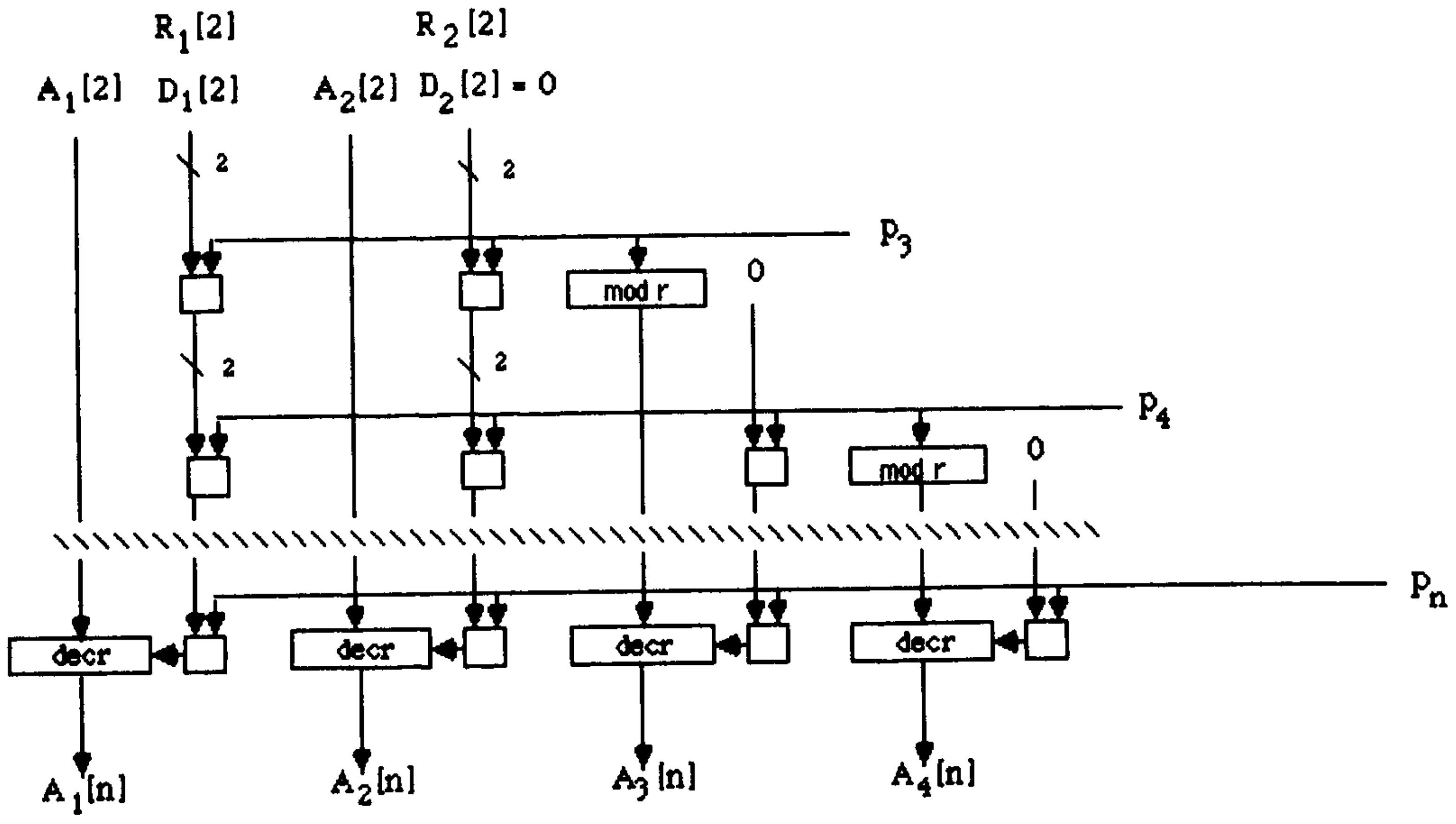

Figure 4c: Combinational Implementation with Decrementers at the Last stage

- A Segment

References

[1] M. D. Ercegovac and T. Lang, "On-the -Fly Conversion of Redundant into Conventional Representations", to be published in IEEE Transactions on Computers, 1987.

[2] J.E. Robertson, "A New Class of Digital Division Methods", *IRE Trans. on Electronic Computers,* Vol.EC-7, September 1958, pp.218-222.

[3] D.E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainders", *IEEE Trans. Computers,* Vol.C-17, October 1968, pp.925-934.

[4] G. Metze, "Minimal Square Rooting", *IEEE Trans. Electronic Computers,* Vol.EC-14, February 1965, pp.181-185.

[5] M.D. Ercegovac and T. Lang, "A Division Algorithm with Prediction of Quotient Digits", *Proc. 7th IEEE Symposium on Computer Arithmetic,* Urbana, Illinois, 1985, pp.51-56.

[6] G.S. Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages", *Proc. 7th IEEE Symposium on Computer Arithmetic,* Urbana, Illinois, 1985, pp. 64-71.

[7] M.D. Ercegovac, "A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer", *IEEE Trans. Computers,* Vol.C-26, July 1977, pp.667-680.

[8] K.S. Trivedi and M.D. Ercegovac, "On-Line Algorithms for Division and Multiplication", *IEEE Trans. Computers,* Vol.C-26, July 1977, pp.667-680.

[9] M.D. Ercegovac, "On-Line Arithmetic: An Overview", *Proc. SPIE 1984,* Vol.495 - Real Time Signal Processing VII, 1984, pp.86-93.

[10] J. Sklansky, "Conditional Sum Addition Logic", *IRE Trans. Electron. Computers,* Vol.EC-9, June 1960, pp.226-231.

[11] M. D. Ercegovac and T. Lang, "On-Line Scheme for Computing Rotation Factors", UCLA Computer Science Department, TR CSD No. 860031, September 1986.

[12] M. D. Ercegovac and T. Lang, "Fast Multiplication without Carry Propagate Addition", Patent Application, 1985.

[13] D. M. Tullsen, "A Very Large Scale Integration Implementation of an On-Line Arithmetic Unit", UCLA Computer Science Department, (MS Thesis), June 1986, Technical Report CSD-860094.