Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit

by E. Hokenek
R. K. Montoye

This paper presents a novel technique used in the multiply-add-fused (MAF) unit of the IBM RISC System/6000* (RS/6000) processor for normalizing the floating-point results. Unlike the conventional procedures applied thus far, the so-called leading-zero anticipator (LZA) of the RS/6000 carries out processing of the leading zeros and ones in parallel with floating-point addition. Therefore, the new circuitry reduces the total latency of the MAF unit by enabling the normalization and addition to take place in a single cycle.

Introduction

Normalization is used as a means of referencing a number to a fixed radix point. Normalization strips out all leading sign bits so that the two bits immediately adjacent to the radix point are of opposite polarity. **Table 1** represents a 32-bit register containing various floating-point numbers.

Using the conventional techniques shown in **Figure 1**, the following three steps must be performed in order to normalize a floating-point addition:

1. The two terms must be added [a process requiring a minimum of log (N) time] [1].
2. The result must be searched for the leading zero or one (depending upon the sign of the result).
3. The result of the addition must be shifted by the appropriate amount and the exponent of the floating-point result must be adjusted accordingly.

Optimizing the RISC System/6000* (RS/6000) architecture and machine organization [2, 3] with a tightly coupled floating-point unit (FPU) which performs the dot-product operations $(A \times B) + C$ required significant innovation in the multiply-add-fused (MAF) design. A major contribution was made by the LZA to accomplish the essential RISC fundamentals of implementing simple, self-contained, low-latency hardware. As mentioned in a companion paper in this issue [4], a minimum of two-cycle latency and a second pipeline delay comparable in time to the multiplication/shifting path of the MAF unit can be

* RISC System/6000 is a trademark of International Business Machines Corporation.

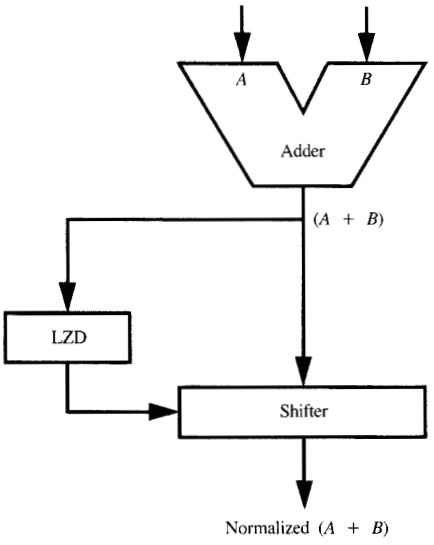

Figure 1

Conventional adder and normalizer.

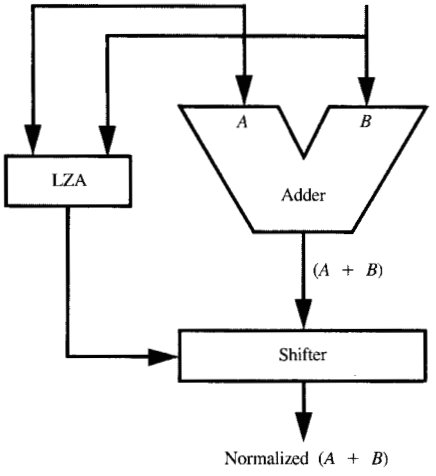

Figure 2

Schematic of MAF normalization.

Table 1 Floating-point numbers in a 32-bit register.

Unnormalized positive number	0 00000010111100011011001111000100 MSB	LSB
After normalization	0 10111100011011001111000100000000 MSB	LSB

achieved by overlapping normalization and addition. However, the question at hand is this: "Can the leading-zero/one detection (LZD) be begun without waiting for the result of the floating-point addition?" or "Can the LZD be performed using only the two operands of the addition?"

This paper describes the novel concept which enables the leading-zero/one detection to be performed in parallel with the addition (subject to a single bit correction). The leading-zero/one detection is subject to normalization after the addition is finished, and the leading-zero/one anticipation occurs concurrently with the addition. The leading-zero anticipator (LZA) of the MAF unit in the RS/6000 processor is described for all values of the operands denoted by A and B (see **Figure 2**).

Algorithm

Suppose that the result $(A + B)$ is an unnormalized floating-point number. There are four possible cases:

1. $A > 0, \quad B > 0, \quad A + B > 0$
   (unnormalized positive number).
2. $A < 0, \quad B < 0, \quad A + B < 0$
   (unnormalized negative number).
3. $A > 0, \quad B < 0, \quad A + B > 0$
   (unnormalized positive number).
4. $A > 0, \quad B < 0, \quad A + B < 0$
   (unnormalized negative number).

To determine the shift amount, the LZA uses the P, G, Z signals that define the bit-to-bit relations of the two operands A and B:

$$P_i = \text{XOR}\,(a_i, b_i), \tag{1a}$$

$$G_i = \text{AND}\,(a_i, b_i), \tag{1b}$$

$$Z_i = \text{NOR}\,(a_i, b_i). \tag{1c}$$

The circuitry required to generate the P and G signals is not an added cost for the LZA: P and G signals are already required for the carry-lookahead adder (CLA) and $Z_i = \text{NOR}\,(P_i, G_i)$.

We now discuss the four cases given above and construct the finite-state representation of the LZA.

Case 1: $A > 0$, $B > 0$, $A + B > 0$

Two possible combinations of A and B which yield the same result are given in **Table 2**.

First, leading-zero/one anticipation should be carried out, starting from the most significant bit (MSB, or sign bit) side of the addition. Considering the examples in Table 2, the state description for the LZA can be summarized as follows:

- The Z-signal at the MSB implies the addition of two positive numbers. The LZA enters a Z-state and

remains unchanged as long as the Z-signal is true, i.e., (1). For each successive Z-input, it should generate a shift signal (SHL).

- The leading-zero/one anticipation is finished when the kth Z-input is false, namely (0). Subsequently, the LZA should take into account the carry into the $(k-1)$th Z-position and create an adjustment signal (AD) accordingly:

$$AD = carry. \tag{2}$$

The adjustment is a single right-shift signal resulting in a total shift:

$$SH = SHL - AD. \tag{3}$$

Case 2: $A < 0$, $B < 0$, $A + B < 0$
Starting from the examples presented in **Table 3**,

- The G-signal guarantees a negative result. The LZA enters into the G-state and remains unchanged as long as the G-signal is true, i.e., (1). For each G-input, the LZA generates a shift signal (SHL).
- The leading-zero/one anticipation is finished when the kth G-input is false, namely (0). Subsequently, the LZA takes into account the carry into the $(k-1)$th G-position and creates an adjustment signal (AD) accordingly:

$$AD = \text{INV}\,(carry), \tag{4}$$

where INV ≡ INVERT.

Notice that simply NORing the two positive operands or ANDing the two negative numbers produces a result which differs from the final normalization by only one bit, i.e., carry. This duality originates from complementing the operands. Naturally, performing the LZA for

$$(-A) + (-B) \rightarrow (-)(A + B) \tag{5}$$

should yield the same shift amounts as $(A + B)$.

Case 3: $A > 0$, $B < 0$, $A + B > 0$
This case corresponds to a subtraction resulting in a positive number (**Table 4**).

We extend the statements given in the previous case using the above examples:

- If the MSB is a P-signal indicating a subtraction, the LZA enters a P-state and remains unchanged as long as the P-signal is true, i.e., (1). Note that the P-state is unstable and should always tip over to the Z- or G-state, since addition can yield a positive or negative number. For each P-input, the LZA should generate a shift signal (SHL).

Table 2 Two possible A/B combinations yielding the same

(a)	A	0 00000010001000010010001010000000
	B	0 00000000110100001001000100000100
	Z	*ZZZZZZZPZPPPPZZZPPZPPZZPPPZZZPZZ*
		$Carry = 0$
(b)	A	0 00000001001000010010001010000000
	B	0 00000001110100001001000100000100
	Z	*ZZZZZZZGPPPPZZZPPZPPZZPPPZZZPZZ*
		$Carry = 1$

Table 3 Illustrative examples for Case 2.

(a)	A	1 11111110000001000100000000101001
	B	1 11111111000010100000011000010010
	G	*GGGGGGGPZZZZPPPZZPZZPPZZZPPPZPP*
		$Carry = 0$
(b)	A	1 11111101100001000100000000101001
	B	1 11111111100010100000011000010010
	G	*GGGGGGPGGZZZPPPZZPZZPPZZZPPPZPP*
		$Carry = 1$

Table 4 Subtraction resulting in a positive number (Case 3).

	A	0 00100000101000010010001010000000
	B	1 11100010010100001001000100000100
	P	*PPGZZZPZPPPPZZZPPZPPZZPPPZZZPZZ*
		$Carry = 0$

Table 5 Illustrative examples for Case 4.

	A	0 00011111100001000100000000101001
	B	1 11011101100010100000011000010010
	P	*PPZGGGPGGZZZPPPZZPZZPPZZZPPPZPP*
		$Carry = 1$

- If the jth input signal is a G-signal, it is already known that the result is positive. The new state is the Z-state presented in Case 1. LZA creates a shift output and continues as if it had started with the Z-state.

Case 4: $A > 0$, $B < 0$, $A + B < 0$
On the basis of the above discussion, this case can easily be included in our finite-state machine (**Table 5**).

As described in Case 3, the subtraction starts with the P-state; therefore, the conditions described above are also valid here. The next input entered in the P-state,

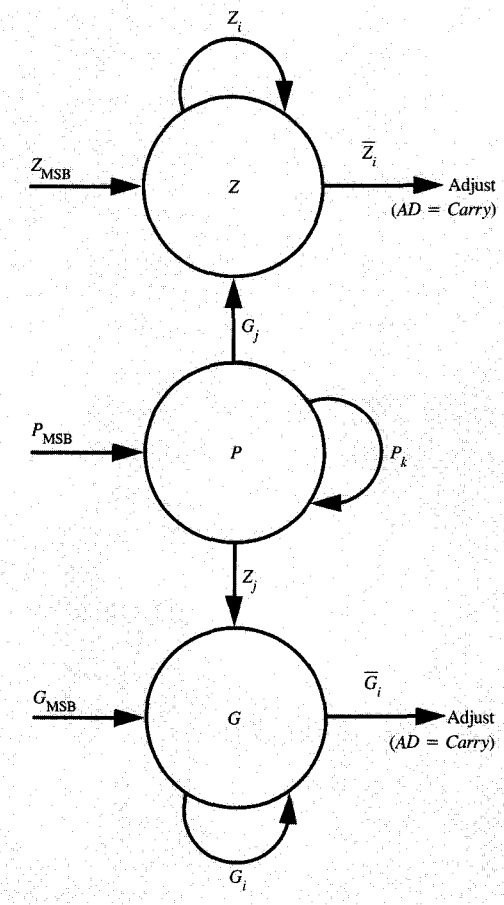

Figure 3

State diagram of the bit-serial leading-zero/one anticipator for addition and subtraction.

however, is not a G-input but rather a Z-input, indicating that the result is going to be negative. Consequently, the new state of the LZA must be the G-state.

If the jth input is a Z-signal, generate a shift output and continue as if the LZA had started with a G-state.

The general state diagram can be obtained as shown in **Figure 3**. Apart from the carry-dependent adjustment, the logical descriptions of the finite-state machine representation can be obtained as follows:

$$Z = Z_k + P_i G Z_{k-(i+1)} \text{ (positive result)}, \tag{6}$$

$$G = G_k + P_i Z G_{k-(i+1)} \text{ (negative result)}. \tag{7}$$

As shown, the Z-state can occur for the string of either (k) Z-inputs or (i) P-inputs followed by a single G and the string of (k–i–1) Z-inputs. Similar statements can be made for the G-state. Besides the total shift amount, the sequential model of the LZA also points out whether the final result of the addition is to be positive or negative, depending on the previous state before finishing leading-zero/one anticipation. If this is a Z-state, the final result is positive; otherwise, it is negative, since, as depicted in Figure 3, the P-state always leads to the Z- or G-state.

Logarithmic leading-zero/one anticipator (LZA)

The finite-state model of the LZA allows us to enter a string of serial inputs which, depending on the bit length (N), is not always as fast as a carry-lookahead adder [5]. It is therefore necessary to process the string of P-, G- and Z-inputs using a parallel algorithm similar to the lookahead structure. The final construction will process the input data in discrete blocks of length D. This approach can be interpreted as a parallel implementation of our finite-state machine, considering its combinatorial equivalents for different state and input combinations.

In the following, the leading-zero/one anticipation is carried out digitwise; i.e., the block length is 4 bits. The results of this study can easily be extended to arbitrary block lengths. We assume that the beginning of a block is the kth bit position. The possible input combinations at this point are given by

digit
| |
k

.... Z... ....
.... P... ....
.... G... ....

According to the string between the kth and $(k-3)$th bit positions, the state outputs of the LZA are defined as follows:

$$ZZ_{1k} = Z_k Z_{(k-1)} Z_{(k-2)} Z_{(k-3)}, \tag{8a}$$

$$PP_{1k} = P_k P_{(k-1)} P_{(k-2)} P_{(k-3)}, \tag{8b}$$

$$PZ_{1k} = P_k P_{(k-1)} [P_{(k-2)} G_{(k-3)} + G_{(k-2)} Z_{(k-3)}] + [P_k G_{(k-1)} + G_k Z_{(k-1)}] Z_{(k-2)} Z_{(k-3)}, \tag{8c}$$

$$PG_{1k} = P_k P_{(k-1)} [P_{(k-2)} Z_{(k-3)} + Z_{(k-2)} G_{(k-3)}] + [P_k Z_{(k-1)} + Z_k G_{(k-1)}] G_{(k-2)} G_{(k-3)}, \tag{8d}$$

$$GG_{1k} = G_k G_{(k-1)} G_{(k-2)} G_{(k-3)}. \tag{8e}$$

Notice that the names of the intermediate-state outputs correspond to their beginning and ending states. For a block-length D (3^D possible input combinations), the number of input combinations resulting in an intermediate LZA block output is $(2D + 3)$.

Looking back to the finite-state machine, the new set of state equations designated in terms of the beginning and ending inputs is slightly different. Two new terms,

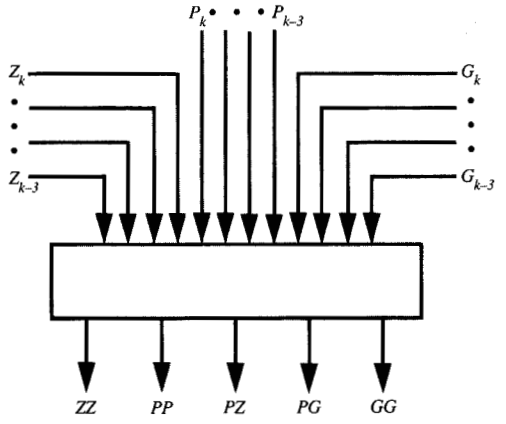

Figure 4

Block diagram for computing the initial states.

$$[G_k Z_{(k-1)} Z_{(k-2)} Z_{(k-3)}] \tag{9}$$

and

$$[Z_k G_{(k-1)} G_{(k-2)} G_{(k-3)}], \tag{10}$$

are included in the *PZ*- and *PG*-states that would not occur in the LZA model given in Figure 3. These expansions are due to the fact that each block handles the data without being informed about the results of the adjacent block. Hence, if the output state of the previous block is *PP*, the consecutive state should be a *PZ*- or *PG*-output. The resulting basic building block of the LZA is shown in **Figure 4**. This circuit is combinatorial and can be implemented using the logic equations given in Equations (8a–e). Thus, the implementation of the sequential machine is converted into the problem of propagating the different state outputs for the iterative combinatorial network. A possible anticipation scheme for the state iteration is depicted in **Figure 5**. The logical equations necessary for the state iteration can be obtained as follows:

(a)

1	$m-1$	m
$ZZZZ$	$ZZZZ$	$ZZZZ$
ZZ_{11} ...	$ZZ_{1(m-1)}$	ZZ_{1m}

$$ZZ_{(j+1)m} = ZZ_{j(m-1)} ZZ_{jm}. \tag{11}$$

(b)

1	$m-1$	m
$PPPP$	$PPPP$	$PPPP$
PP_{11} ...	$PP_{1(m-1)}$	PP_{1m}

$$PP_{(j+1)m} = PP_{j(m-1)} PP_{jm}. \tag{12}$$

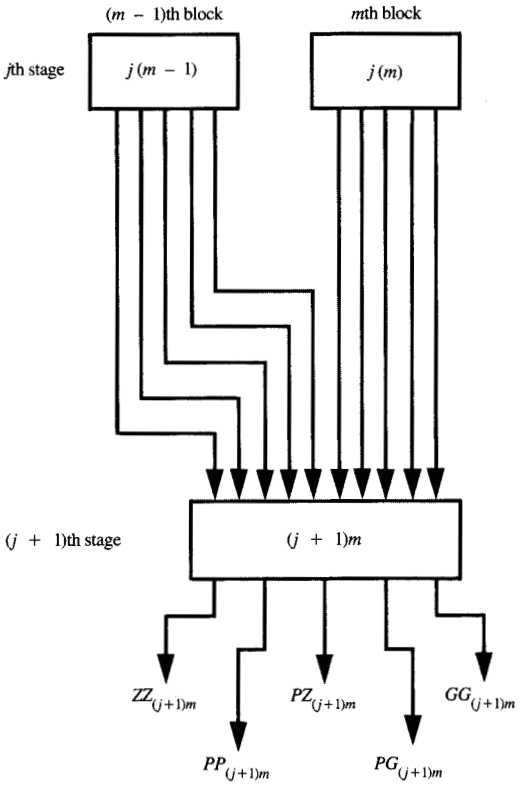

Figure 5

Building block for state iterations.

(c)

1	$m-1$	m
$PPPP$	$PPPP$	$PPGZ$
		[or $PGZZ$, $GZZZ$, $PPPG$]
PP_{11} ...	$PP_{1(m-1)}$	PZ_{1m}

1	$m-1$	m
$PPPP$	$PPGZ$	$ZZZZ$
PP_{11} ...	$PZ_{1(m-1)}$	ZZ_{1m}

$$PZ_{(j+1)m} = PP_{j(m-1)} PZ_{jm} + PZ_{j(m-1)} ZZ_{jm}. \tag{13}$$

(d)

1	$m-1$	m
$PPPP$	$PPPP$	$PPGZ$
		[or $PZGG$, $ZGGG$, $PPPZ$]
PP_{11} ...	$PP_{1(m-1)}$	PG_{1m}

1	$m-1$	m
$PPPP$	$PPZG$	$GGGG$
PP_{11} ...	$PG_{1(m-1)}$	GG_{1m}

$$PG_{(j+1)m} = PP_{j(m-1)} PG_{jm} + PG_{j(m-1)} GG_{jm}. \tag{14}$$

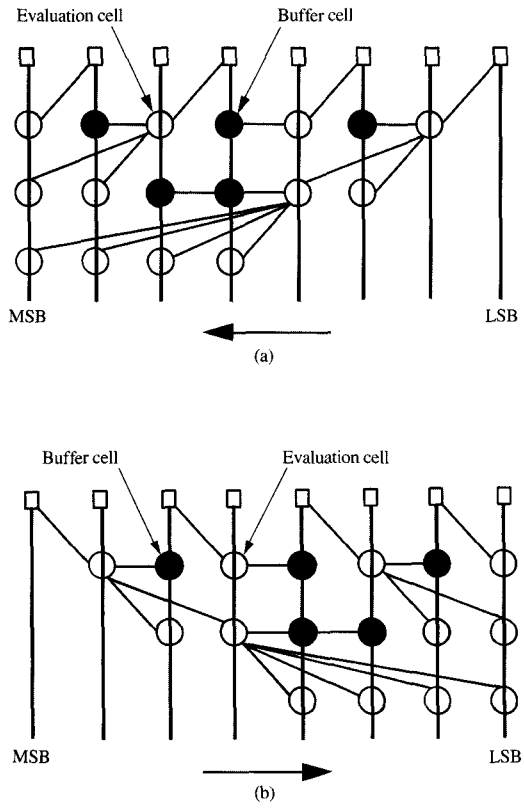

Figure 6

(a) Carry-lookahead; (b) LZA state anticipation.

(c)
$$\begin{array}{cccc} 1 & & m-1 & m \\ \hline GGGG & \cdots & GGGG & GGGG \\ GG_{11} & \cdots & GG_{1(m-1)} & GG_{1m} \end{array}$$

$$GG_{(j+1)m} = GG_{j(m-1)}GG_{jm}. \quad (15)$$

In the preceding, $j = 1, 2, \cdots, (J-1)$ and $m = (i^{(j-1)} + 1), \cdots, M$; $M = N/D$, $J = \log_i M$ (J, M integer, i = lookahead distance); N = total bit-length of the LZA, D = block-length, and J = number of LZA stages necessary.

Although the equations for the state iteration are expressed in terms of the state outputs of the two adjacent blocks, they can be extended to arbitrary lookahead distances. At any stage, an anticipated state can be generated by implementing the above equations and using auxiliary functions ZZ_{1k}, PP_{1k}, PZ_{1k}, PG_{1k}, and GG_{1k}. Since only one of these states can be true, the shift signal is defined by

$$SHL_{jm} = \text{OR}\,(ZZ_{jm}, PP_{jm}, PZ_{jm}, PG_{jm}, GG_{jm}). \quad (16)$$

The shift signals generated by the LZA consist of a 1-string followed by the 0s. The adjustment position is determined by the transition digit and can be defined as

$$ADP_m = SH_{im}\,\text{INV}\,[SH_{i(m+1)}], \quad (17)$$

which is true if and only if the two successive shift signals are different. Interestingly enough, if the finishing state is PP, $carry = 1$ and $carry = 0$ will determine whether the result is positive or negative, but it will not change the shift amount.

Implementation

As pointed out earlier, taking full advantage of leading-zero/one anticipation can be achieved by processing the string of P-, G-, and Z-inputs as fast as the adder, e.g., by using a parallel algorithm similar to the carry-lookahead structure. Building blocks of this computation were presented in the previous section. The LZA operates over a doubling/buffering process similar to the CLA of the MAF unit, but with the following differences:

- As abstracted in **Figures 6(a)** and **6(b)**, the LZA operates in the opposite direction from the CLA, i.e., from MSB to LSB.
- Its hexadecimal implementation makes it possible to build more complex logical functions at each step. After the initial states of the first stage are computed, each cell at the following iterations has the intermediate ZZ-, PP-, PZ-, PG-, and GG-signals as inputs (to receive the information from its neighbors on the left and the top) and outputs (to supply the anticipated states to its neighbors on the right and the bottom, as well as to the network outputs).
- The LZA's conclusion is to OR all intermediate states, which generally yields a 1-string followed by a 0-string. To accommodate the partial-decode scheme used in the shifters [4], the shift signals are further coded into shift positions $(0 \cdots 7) \times 4$ digits + $(0, 1, 2, 3) \times 1$ digit (total of 12 control signals), by connecting the SHL-signals at appropriate distances. Note that a generalized form of Equation (17) for an n-digit shift signal can be defined by

$$NSH_m = SH_m\,\text{INV}\,(SH_{m+n}). \quad (18)$$

The LZA implementation in the RS/6000 floating-point execution unit has been tuned to the carry-lookahead adder so that the control signals for the 4-digit and 1-digit shifts as well as the data output arrive at the shifter nearly simultaneously.

Summary

A new concept for normalizing the result of a floating-point addition has been presented. Implementation of the LZA has made it possible to reduce the latency of the

multiply-add-fused (MAF) unit without sacrificing performance.

References

1. S. Winograd, "On the Time Required for Binary Addition," *J. ACM* **12,** 277–285 (1965).
2. R. R. Oehler and R. D. Groves, "IBM RISC System/6000 Processor Architecture," *IBM J. Res. Develop.* **34,** 23–36 (1990, this issue).
3. G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. Res. Develop.* **34,** 37–58 (1990, this issue).
4. R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 Floating-Point Execution Unit," *IBM J. Res. Develop.* **34,** 59–70 (1990, this issue).
5. O. L. MacSorley, "High Speed Arithmetic for Binary Computers," *Proc. IRE* **49,** 67–91 (1961).

Received October 10, 1989; accepted for publication December 14, 1989

Erdem Hokenek *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.*

Dr. Hokenek received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Technical University of Istanbul, Turkey, and the Swiss Federal Institute of Technology, Zurich, Switzerland, in 1974, 1976, and 1985, respectively. As a postdoctoral World Trade Visiting Scientist, he was assigned in 1985 to the Thomas J. Watson Research Center, Yorktown Heights, New York, where he joined the VLSI Department as a Research Staff Member in 1986.

Robert K. Montoye *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.*

Dr. Montoye received his B.S. in 1977 in physics and his M.S. in 1981 and Ph.D. in 1983 in computer science from the University of Illinois. He joined IBM in 1983 and began research into high-performance CMOS design, including the MAF floating-point unit. Dr. Montoye is the author of numerous articles and holds patents in parallel processing, VLSI architectures, and design automation.