High Speed Decimal Addition

MARTIN S. SCHMOOKLER, MEMBER, IEEE, AND ARNOLD WEINBERGER, MEMBER, IEEE

Abstract—Parallel decimal arithmetic capability is becoming increasingly attractive with new applications of computers in a multiprogramming environment. The direct production of decimal sums offers a significant improvement in addition over methods requiring decimal correction. These techniques are illustrated in the eight-digit adder which appears in the System/360 Model 195.

Index Terms—Adder, BCD code, carry look-ahead, decimal, subtractor.

INTRODUCTION

This paper describes several techniques useful in designing decimal adders which are both high speed and economical. One such technique is the direct production of decimal sums without the need of first producing the binary sums. Another technique is the refinement of carry look-ahead to directly produce the decimal carries. These techniques offer significant improvement over the well-known method of decimal correction. They also permit the design of a parallel decimal arithmetic unit which is competitive to a binary arithmetic unit in performance and cost.

A parallel decimal arithmetic unit is attractive for commercial applications where Cobol is most popular. The small number of calculations performed on each item of data makes it inefficient to convert between decimal and binary. In previous systems, such applications were frequently I/O limited so that conversion or inefficient decimal arithmetic operations were tolerable. In present systems, however, large main memories are often available. This permits a high degree of multiprogramming which results in greater concurrency between I/O and processor.

A parallel decimal arithmetic unit should be even more attractive for a future demand/response environment where there will be a greater percentage of language translation plus output with a lesser computation for both scientific and commercial applications.

There are many codes in use for decimal [1]. Some, such as the two out of five code, are used for their ability to be checked easily. The excess three code is used for its ability to indicate decimal carries when a binary adder is used. The resulting sums must be corrected by subtracting three if no carry was produced from the corresponding adder digits and by adding three if a carry was produced.

The standard four bit BCD code is assumed for the adder described in this paper. This choice was dictated by a requirement for System/360 compatibility. The well-known approach to decimal addition with this code usually consists of binary addition followed by decimal correction. Decimal carries are indicated by a binary carry from the corresponding digit position or by a sum digit of ten or more. Correction is carried out by adding six to each sum digit where a carry is produced. If one is designing an adder dedicated solely to decimal arithmetic, this approach would clearly result in an adder significantly slower and more costly than a corresponding binary adder. Therefore, we decided to investigate the possibility of obtaining the decimal sums and carries in a more direct manner.

ONE DIGIT DECIMAL ADD

First we demonstrate how addition for one digit may be accomplished. Consider the operands $A$ and $B$ which are to be added as follows. (Each subscript refers to the weight of the corresponding bit position.)

$$
A_8 \ A_4 \ A_2 \ A_1 \ C_{\text{in}}
+ \ B_8 \ B_4 \ B_2 \ B_1
$$

$$
C_{\text{out}} \ S_8 \ S_4 \ S_2 \ S_1
$$

If this were a four-bit binary addition, one could obtain the carry into each bit position from the lower order position to the right, and obtain the sum directly from

$$
S_i = (A_i \oplus B_i) \oplus C_{i-1}.
$$

The concept of bit carries is still retained in decimal addition when a binary adder is used with decimal correction. However, in the adders to be described in this paper, the concept of carries is only useful for the decimal carries produced from each digit, and for the carries produced from bit one of each digit.

For the input variables in columns 2, 4, and 8 of each digit, we define two new functions. Let $K$ be true only when the sum from these columns is ten or greater, and let $L$ be true only when the sum is eight or greater. Then if $C_1$ is the carry from column 1, the digit carry is

$$
C_{\text{out}} = K + LC_1
$$

and

$$
\bar{C}_{\text{out}} = L + \bar{K}\bar{C}_1.
$$

It is apparent (to those familiar with [2] or [3]) that $K$ represents a carry generate function and $L$ represents a carry propagate function. Thus, we have separated the digit into two groups, the higher order group consisting of columns 8, 4, and 2, and the lower order group consisting of column 1 alone.

Before exhibiting the Boolean expressions for $K$ and $L$, we define

$$
P_i = A_i + B_i, \ G_i = A_iB_i
$$

and

Manuscript received July 2, 1970; revised January 21, 1971.

The authors are with the Systems Development Division, IBM Corporation, Poughkeepsie, N. Y. 12602.
ahead is any moved, always could be P8G2C1. For the input operands, the method of the digit position which cause a carry to be generated from the digit position P is true for the conditions in which a carry could propagate through from a lower order digit. If A_i and B_i are corresponding operand digits in the i-th column from the right and R is the radix, then

\[ G_i \equiv (A_i, B_i) \text{sum of } A_i \text{ and } B_i \geq R \]

and

\[ P_i \equiv (A_i, B_i) \text{sum of } A_i \text{ and } B_i = R - 1 \].

Since \( C_i = G_i + P_iC_{i-1} \), we may also include in \( P_i \) any terms contained in \( G_i \).

Using these functions, the carry from the i-th digit may be written as

\[ C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \cdots + P_{i-1}P_{i-2} \cdots P_1C_{i-1}. \]

Where there is a very large number of digits, \( generate \) and \( propagate \) functions may be used for groups of digits. With this technique, which is described in [2], the respective functions for four digits may be written as

- Group carry \( generate = G_2 + P_2G_3 + P_2P_3P_1G_2 + P_2P_1P_2G_1 \)
- Group carry \( propagate = P_2P_3P_2P_1 \).

Quite often, the digits themselves may be separated, as we already demonstrated for BCD, permitting other variations in the group carry functions. One such variation will be exhibited in the eight digit decimal adder described in the next section.

**Carry Look-Ahead for a Decimal Adder**

The generalized carry look-ahead principle can be applied to a multidigit decimal adder in a number of ways. The group carry \( generate \) and \( propagate \) signals may be formed for individual digits (radix 10) or for two-digit groups (corresponding to radix 100), etc. For the BCD decimal adder, the \( K \) and \( L \) functions previously defined permit other groupings as well. The reason is that \( K \) and \( L \) are, respectively, the carry \( generate \) and \( propagate \) functions of the three high-order bits of a decimal digit.

Table I illustrates two methods of forming an eight-bit group. In the first method, the eight bits comprise a byte; i.e., two adjacent decimal digits denoted as the high- and low-order digits. In the other method, the eight bits comprise a three-digit straddle to include a digit, the low-order bit of the next higher order digit, and the three high-order bits of the next lower order digit. The byte carry \( generate \) and \( propagate \) are

\[ G_B = G_H + P_HG_L = K_H + L_HG_{1H} + L_HP_{1H}K_L \]
\[ + L_HP_{1H}L_GK_L \] (12a)
\[ P_B = P_HP_L = L_HP_{1H}L_GP_{1L} \] (12b)

while the straddle carry \( generate \) and \( propagate \) are

\[ G_S = G_{1NH} + P_{1NH}K_H + P_{1NH}L_HG_{1H} + P_{1NH}L_HP_{1H}K_L \] (13a)
\[ P_S = P_{1NH}L_HP_{1H}L_G. \] (13b)
TABLE I
EIGHT-BIT GROUPINGS FOR BCD DECIMAL GROUP CARRY GENERATE AND PROPAGATE SIGNALS

<table>
<thead>
<tr>
<th>Bit 1</th>
<th>Bit 8</th>
<th>Bit 4</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 8</th>
<th>Bit 4</th>
<th>Bit 2</th>
<th>Bit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_{IH}</td>
<td>K_{H}</td>
<td>L_{H}</td>
<td>G_{IH}</td>
<td>K_{H}</td>
<td>L_{H}</td>
<td>G_{IH}</td>
<td>K_{H}</td>
<td>L_{H}</td>
</tr>
<tr>
<td>P_{IH}</td>
<td>G_{IH}</td>
<td>K_{H}</td>
<td>P_{IH}</td>
<td>G_{IH}</td>
<td>K_{H}</td>
<td>P_{IH}</td>
<td>G_{IH}</td>
<td>K_{H}</td>
</tr>
</tbody>
</table>

3RD BYTE (GROUP OF 2 DECIMAL DIGITS)

Fig. 1. Block diagram of third byte (group of two decimal digits) of a four byte adder.

EIGHT-DIGIT DECIMAL ADDER

The principles described in the previous sections are illustrated by the eight-digit (four byte) decimal adder used in the System/360 Model 195. The adder consists of four groups of one byte each. Fig. 1 shows the logical organization of one of the bytes, the third byte from the right. The K and L functions for each digit are obtained and used to form the generate and propagate functions for the bytes which are then fed to the higher order bytes. Similar signals are received from each of the lower order bytes to develop the carries needed within the byte. They are the C_l carries, which are labeled C_{IL} and C_{IH} for the lower and higher order digits respectively, and the byte carry C_B which is needed by the next higher byte as the input carry.

The detailed logic for the same byte is shown in Fig. 2. It shows the circuitry for the K and L functions, the byte generate and propagate functions G_B and P_B, and the carries. It also shows the circuitry for developing the sums of both the low- and high-order digits.

The circuits used are current-switch circuits with emitter follower outputs, as shown in Fig. 3. Each circuit may provide two possible outputs. The upper output of a logic block provides the OR function of the inputs, and the lower output provides the AND function of the inputs. The outputs of several logic circuits may be externally wired together to give the OR of the separate outputs. However, a wired OR is not considered a logic level since it does not contribute significantly to the delay. Up to four circuit outputs may be wired together.
Fig. 2. Detailed logic diagram of third byte of four byte adder.

Fig. 3. Circuit and logic diagrams of current switch gate.
For the design shown, all of the output sums are obtained in six logic levels or less. The $K$ and $L$ functions are obtained in two logic levels using (3a) and (3b). Each $L$ function actually consists of a pair of lines whose OR is the $L$ function. This method of ORing is used instead of a wired OR so that one of the lines $G_4$ may be used separately.

$G_B$ and $P_B$ are available after three levels and are implemented according to the following Boolean expressions:

$$G_B = L_H(K_H + P_{1H})(K_H + G_{1H} + L_I)(K_H + G_{1H} + K_L + G_{1L})$$

$$P_B = L_HL_1P_{1H}P_{1L}.$$  

(14a)

(14b)

In Fig. 2, $S_{2L}$, $S_{4L}$ and $S_{8L}$ are obtained using (7), (8), and (9), respectively and $S_{1L}$ is implemented using the Boolean equation

$$S_{1L} = H_{1L}C_B + P_{1L}C_{1L}.$$  

(15)

This form is used in preference to (5) because it does not require $C_B$.

In the high-order digit, $S_{2H}$, $S_{4H}$, and $S_{8H}$ are obtained in identical manner, while $S_{1H}$ is obtained from

$$S_{1H} = H_{1H}K_L + P_{1H}L_1C_{1L} + P_{1H}C_{1H}.$$  

(16)

### Decimal Subtraction

Subtraction is often performed in a binary system by adding the 2's complement of the subtrahend to the minuend. If the result is negative, as indicated by lack of a high-order output carry, then it is complemented during the following cycle to obtain the proper magnitude. The 2's complement is effectively obtained by using the 1's complement and forcing a carry into the low-order adder position. The 1's complement is obtained simply by inverting each input bit.

The corresponding method for decimal subtraction requires the 9's complement instead of the 1's complement. Fortunately, the 9's complement may be provided with little additional complexity. Let $A$ be a decimal digit encoded as $A_8A_4A_2A_1$. Then its complement $C$ consists of

$$C_1 = \overline{A_1}$$

$$C_2 = A_2$$

$$C_4 = A_4\overline{A_2} + \overline{A_4}A_2$$

$$C_8 = \overline{A_8}A_4\overline{A_2}.$$  

### Comparisons with Other Adders

The eight-digit decimal adder as described, requires fewer logic circuits than comparable adders which employ decimal correction. Furthermore, decimal correction incurs two additional logic levels of delay.

When compared with a 32-bit binary adder which has the same width, the decimal adder requires about 18 percent more circuitry for the same number of logic levels. On the other hand, a 32-bit binary adder having the same cost as the decimal adder can be implemented with only five logic levels of delay instead of six.

Other decimal adders have been designed based on the principles described in this paper. Among them is a two-digit adder requiring only three logic levels of delay.

### References

