# Replacing the Hardware Addition Operation by Software algorithm without Carry

**Imad Matti Bakkow**
**M.Sc. Computer Science**

## Abstract:

**This paper presents a new method to perform the arithmetic addition operation on numbers in a faster way in comparison with the existing one on computers.**

**The proposed method builds a new architecture for the adder circuit in the CPU, So that there is no need for a waiting time to perform carrying bits from low order position to high order position when adding two numbers.**

**The details of the new method are successfully tested with many different examples.**

**Keywords: Adder circuit, Carry concept, Add operation, Algorithm.**

# استبدال عملية الجمع بالبوابات بخوارزمية بدون استخدام تحميل

**م.عماد متي بكو**
**كلية المأمون الجامعة / قسم علوم الحاسوب**

**المستخلص :**

**يقدم هذا البحث طريقة جديدة لتنفيذ عملية الجمع الحسابية على الاعداد بصورة أسرع مقارنة بما هو معتمد عليه في الحاسبات الالكترونية . تبني الطريقة المقترحة معمارية لدائرة الجامع (adder circuit) في المعالج المركزي بحيث لا يعد هنالك حاجة الى وقت للانتظار (waiting time)، عند تنفيذ عملية التحميل (carry bit) من المرتبة السابقة للعدد الى المرتبة اللاحقة له عند جمع عددين .وقد تم اختبار تفاصيل الطريقة الجديدة بنجاح على امثلة عديدة مختلفة.**

## 1. Introduction

**To implement an add micro operation in a computer, we need registers that hold data, and digital components that perform the arithmetic addition. Figure 1, shows block diagram of add micro operation, which accepts two binary digits on it's inputs, and produces two binary digits on it's outputs, a sum bit, and a carry bit [1], [2].**

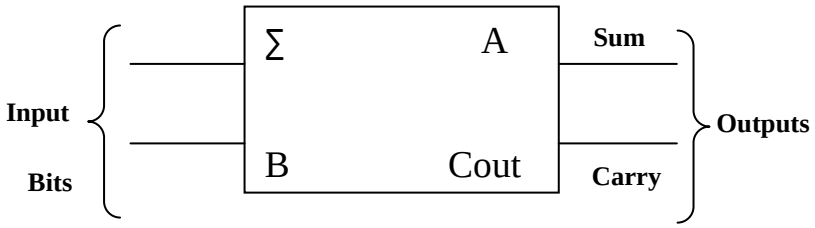


***Figure 1: Block Diagram of the Add Micro Operation***

**The digital circuit that generates the arithmetic sum of two binary numbers of any length is called binary adder.**
**The binary adder is constructed with full-adder circuits connected in a cascade, with the output carry from one full-adder connected to the input carry of the next full-adder. Figure 2, shows the inter connections of four full-adders (FA) to provide a 4-bit binary adder [2], [3], [4].**

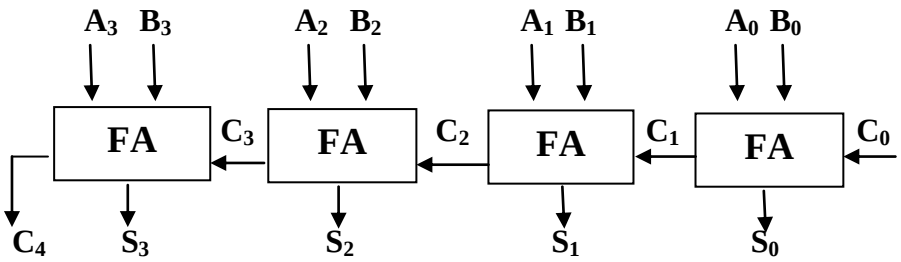


***Figure 2: 4–Bit Binary Adder.***

**The augends bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the**

**low-order bit. The carries are connected in a chain through the full-adders .The input carry to the binary adder is $C_0$ and the output carry is $C_4$. The S outputs of the full-adders generate the required sum bits [2],[3].**

**Since the output carry from each full-adder (FA) is the input carry of the next-high-order full-adder, hence to generate the sum S1 for example it depends on the carry $C_1$ generated from the previous full-adder (FA) and so forth. This situation slows the add microoperation, since there is a waiting time to generate a carry bit as an input to the next full-adder [5], [6], [7].**

## 2. The Proposed Method for Adding Two 4-bit Numbers

**Let $A = (a_3 , a_2 , a_1 , a_0 )_2$ , and $B = (b_3 , b_2 , b_1 , b_0 )_2$ be two binary numbers.**

**Three registers are needed, say register R1 for the number A, register R2 for the number B , and register S for the sum of the numbers A and B. The digits of the number A are put in register R1, and the digits of the number B are put in register R2 as follows:-**

| | | | | | |
|---|---|---|---|---|---|
| $2^3$ | $2^2$ | $2^1$ | $2^0$ | | |
| $a_3$ | $a_2$ | $a_1$ | $a_0$ | ← | Register R1 |
| $b_3$ | $b_2$ | $b_1$ | $b_0$ | ← | Register R2 |

**i.e. The digit $a_0$ in the position $2^0$ of the number A will be in the position $2^0$ of register R1.**

**The digit $a_1$ in the position $2^1$ of the number A will be in the position $2^1$ of register R1.**

**The digit $a_2$ in the position $2^2$ of the number A will be in the position $2^2$ of register R1.**

**The digit $a_3$ in the position $2^3$ of the number A will be in position $2^3$ of register R1.**

**In the same way , we put the digits of the number B in register R2.**

**i.e. The digit $b_0$ in the position $2^0$ of the number B will be in the position $2^0$ of register R2.**

**The digit $b_1$ in the position $2^1$ of the number B will be in the position $2^1$ of register R2.**

**The digit $b_2$ in the position $2^2$ of the number B will be in the position $2^2$ of register R2.**

**The digit $b_3$ in the position $2^3$ of the number B will be in the position $2^3$ of register R2.**
**For the sum of the numbers A and B , we use a register say S , and we put 0 in all it's positions , and as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ← register S |

**Now , to add the registers R1 and R2 into the sum register S , the method suggests the truth table shown in Figure(**3).

| + | 0 | 1 |
|---|---|---|
| 0 | The same bit in register S | 1 if register S contains 0 in its position, else 0 if register S contains 1 in its position, with putting 1 in the next position of register S. |
| 1 | 1 if register S contains 0 in its position , Else 0 if register S contains 1 in its position , with putting 1 in the next position of register S . | 1 in the next position of register S the bit in the current position of register S stay the same. |

***Figure 3 : Suggested Truth Table for the Adder circuit.***

**From the preceding paragraphs and the truth table of the adder circuit suggested earlier, we can write an algorithm for adding two 4-bit binary numbers A and B, which are in registers R1, R2 respectively , and their sum register S , as follows:-**

## 2.1 The Algorithm for Adding Two 4-Bit Numbers

**1. Start.**
**2. i ← 0 .**
**3. REPEAT**
**if the digit in positions $2^i$ of registers R1 and R2 are equal to 0**
**then i = i + 1;**
**else**

**if the digit in positions $2^i$ of registers R1 and R2 are not equal to 0**
**then**
**begin**
**put 1 in position $2^{i+1}$ of register S;**
**i = i + 1;**
**end**
**else**
**If the digit in positions $2^i$ of registers R1 and R2 are different i.e ( 0 and 1) or ( 1 and 0 )**
**then**

**If the digit in position $2^i$ of register S is equal to 0**
**then**
**begin**
**put 1 in position $2^i$ of register S;**
**i = i + 1;**
**end;**
**else**
**begin**
**put 0 in position $2^i$ of register S;**
**put 1 in position $2^{i+1}$ of register S;**
**i = i + 1;**
**end;**
**UNTEL ( i = 4 );**
**4- stop.**

**Note 1:The number in the register S will be the result of the sum of adding the numbers A and B.**

## 2.2 Example for Adding Two 4-Bit Numbers according to the Proposed Algorithm

**Suppose we want to add the binary number A = ( 0111 )$_2$ , to the binary number B = ( 1011 )$_2$ , without using any carry. We follow the following steps:-**

**1.We put the digits of the number A in register R1, as follows:-**

| $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | register R1 |

**2. We put the digits of number B in register R2 , as follows:-**

| $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | register R2 |

**3. We put 0 in all positions of register S ( the sum ) , as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | register S |

**Now we follow :-**
**For i = 0.**

**4. Since the digit in position $2^0$ of register R1 is equal to 1 , and the digit in position $2^0$ of register R2 is equal to 1 , we put 1 in position $2^1$ of register S , as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | register S |

**We repeat for i = 1.**

**5. Since the digit in position $2^1$ of register R1 is equal to 1 , and the digit in position $2^1$ of register R2 is equal to 1, we put 1 in position $2^2$ of register S , as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | register S |

**We again , repeat for i = 2.**

**6. Since the digit in the position $2^2$ of register R1 is equal to 1 , the digit in position $2^2$ of register R2 is 0, and since the digit in the position $2^2$ of register S is equal to 1 , we put 0 in position $2^2$ of register S , and we put 1 in position $2^3$ of register S , as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | register S |

**We again , repeat for i = 3.**

7. **Since the digit in position $2^3$ of register R1 is equal to 0, and the digit in position $2^3$ of register R2 is equal to 1, and since the digit in position $2^3$ of register S is equal to 1 , we put 0 in position $2^3$ of register S , and we put 1 in position $2^4$of register S , and as follows:-**

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | register S |

**Note 2 :The value in register S is the sum of the number A and B without using any carry.**

## 2.3 Theoretical of the New Add Method with Many other Test Data:-

**We introduce here many different samples of data for adding the numbers A and B.**

**The results of applying the new add method according to the algorithm mentioned earlier, are the same results of sum when we add them by using the carry concept.**

**Sample 1: A = $(0111)_2$ , and B = $(1111)_2$ .**

**Register R1 = 0 1 1 1**
**Register R2 = 1 1 1 1 +**
**---**
**Register S = 0 0 0 0 0 initial value**
**Register S = 0 0 0 1 0 step i = 0**
**Register S = 0 0 1 1 0 step i = 1**
**Register S = 0 1 1 1 0 step i = 2**
**Register S = 1 0 1 1 0 step i = 3, the sum.**

**Sample 2 : A = $(1110)_2$ and B = $(1011)_2$ .**

**Register R1 = 1 1 1 0**

**Register R2 = 1 0 1 1 +**
**Register S = 0 0 0 0 0 initial value**
**Register S = 0 0 0 0 1 step i = 0**
**Register S = 0 0 1 0 1 step i = 1**
**Register S = 0 1 0 0 1 step i = 2**
**Register S = 1 1 0 0 1 step i = 3 the sum .**

## 2.4 Implementation of The Proposed Add Method on Fractional Numbers

**The suggested add method presented earlier in the algorithm can also be applied perfectly on fractional binary numbers.**

**Example 1: A = $(0.101)_2$ and B = $(0.111)_2$,**
**Register R1 = 0 . 1 0 1**
**Register R2 = 0 . 1 1 1 +**
**---**
**Register S = 0 . 0 0 0 initial value**
**Register S = 0 . 0 1 0 step i = 0**
**Register S = 0 . 1 0 0 step i = 1**
**Register S = 1 . 1 0 0 step i = 2**
**Register S = 1 . 1 0 0 step i = 3**

**Example 2 : A = $(0.11111)_2$ and B = $(0.11101)_2$ .**

**Register R1 = 0 . 1 1 1 1**
**Register R2 = 0 . 1 1 0 1 +**

**Register S = 0 . 0 0 0 0 initial value**

**Register S = 0 . 0 0 1 0 step i = 0**
**Register S = 0 . 0 1 0 0 step i = 1**
**Register S = 0 . 1 1 0 0 step i = 2**
**Register S = 1 . 1 1 0 0 step i = 3**
**---**

## 2.5 Conclusions

**From figure 1 and figure 2 , several notes can be concluded:**

1- The add operation is built on the carry concept by hardware. The add operation in the proposed algorithm is built by software.
2- Since there is always a waiting time to add the carry bit from low-order position to high-order position , the add with carry operation will be slower in comparison with the proposed add operation.
3- The move (we use put ) operation takes in general less clock cycles than the add with carry operation.
4- It is recommended to develop a general algorithm that extends the numbers A and B as follows:-
   $A = ( a_7 , \ldots , a_2 , a_1 , a_0 )_2$ and $B = ( b_7 , \ldots , b_2 , b_1 , b_0 )_2$ .
5- It is recommended to test it practically and then evaluate the difference in processing time between the two methods.

## References


1.Morris Mano , "Digital Fundamentals ", Prentice Hall, International Education, Eighth Edition, 2003.
2. M. Morris Mano and Charles R. Kime, "LOGIC and COMPUTER DESIGN FUNDAMENTALS " , Pearson International Edition, Pearson Prentice Hall, fourth Edition, Printed in Singapore, 2008.
3.Morris Mano, " Digital Design ", Prentice Hall, Education International, Third Edition, Printed in the United States Of America, 2002.
4. Barry B. Brey, "The Intel Microprocessors , Architecture , Programming , and Interfacing ", Pearson International Edition , printed in the United States of America , Pearson Prentice-Hall , Eighth Edition , 2009.
5. www.google.com
CARRY [PDF] -LOOKAHEAD ADDERS
http://writphotec.com/mano4/Supplements/Carrylookahead_supp4.pdf
6. WWW.YAHOO.COM
" Carry-lookahead Adder ",
WIKIPEDIA the Free Encyclopedia.
7. www.yahoo.com
" Carry- save Adder ",
Wikipedia , the free Encyclopedia.