HIGH SPEED DIGITAL ARITHMETIC

Thesis submitted for the M.Sc. Degree

to the

Electrical Engineering Department

Faculty of Engineering

Ain Shams University

By

SOHEIR ABDEL HAY ABDEL HAMID

B.Sc. Electrical Engineering, Hons

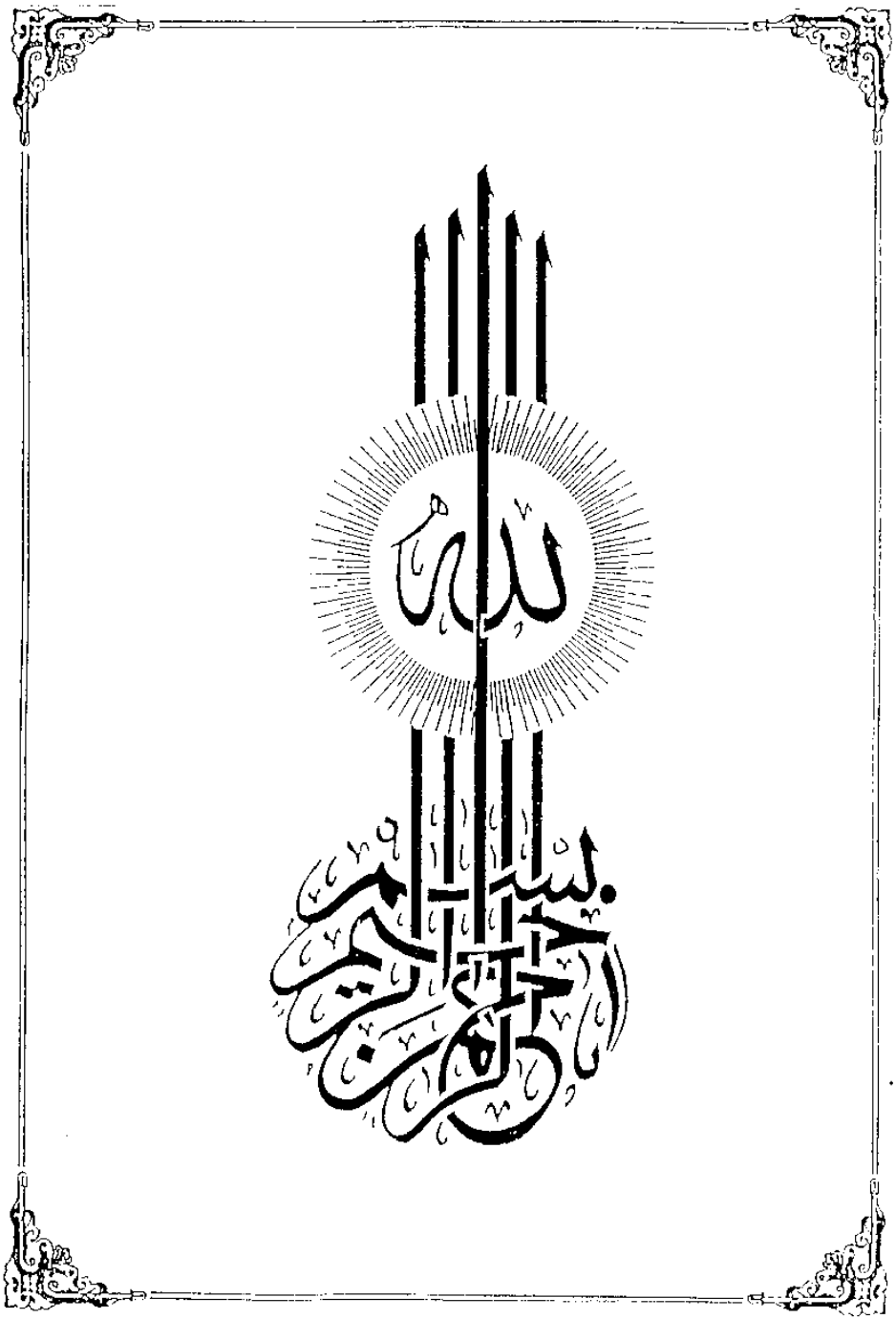
1973

Acknowledgement

The author wishes to thank Prof. Dr. S.M. Yusuf, Head of the Electrical Engineering Dept. for his encouragement during the execution of this research.

Thanks are also due to Dr. M.A.R. Ghonaimy and Dr. M.R. El-Karaksy for supervising this work and for their helpful guidance and constructive criticism.

... ...



Abstract

This thesis is concerned with the development of methods for speeding arithmetic operations in digital computers. Residue number representation was used since it has the major advantage of having no inter-digit carries.

Different algorithms in existence were reviewed and a proposed one using signed - complement residue number representation was used. This representation resulted in improved algorithms for performing the basic arithmetic operations. It had also the advantage of providing simple means of determining the sign and the detection of overflow.

Implementation schemes using R.O.Ms were suggested as a feasible means of mechanizing the different proposed algorithms. This will result in simplicity of construction together with the possibility of achieving very high speeds.

... ...

I N T R O D U C T I O N

This thesis deals with methods of performing the basic arithmetic operations using residue representation of number.

The basic attraction of using residue representation is due to the fact that additive and multiplicative operations are inherently inter digit carry-free.

Chapter 1 reviews digital arithmetic when using the binary representation of numbers. Both fixed - point and floating - point arithmetic are considered. Speeding - up procedures are considered in great detail.

Chapter 2 introduces the residue number system notations and definitions. Basic theorems concerning the manipulation of residue arithmetic are also presented. Modulo adder schemes are introduced and the use of R.O.Ms is suggested as a practical implementation scheme. Conversion algorithms from the residue form to other forms of number representations or vice-versa are given.

Chapter 3 deals with additive operations when numbers are represented in their residue form. First , the available procedures in the literature are surveyed. Next , a proposed scheme is presented in which numbers are represented by their signed - complement residue form. Implementation schemes using R.O.Ms are given.

Chapter 4 considers multiplication and division in residue number systems. Algorithms are proposed using the above number representation. An implementation scheme for performing multiplication is suggested using R.O.Ms.

... ...

CHAPTER 1.

Review of Digital Computer Arithmetic

Introduction :

In this chapter different methods for performing the basic arithmetic operations in both fixed and floating point forms will be presented.

Section A deals basically with speeding-up procedures for fixed point arithmetic including addition , multiplication , and division.

Section B deals with the basic methods of performing floating point arithmetic operations. In this case operands are normally expressed by their normalized mantissa and their exponents.

A. Fixed Point Arithmetic

An adder is a basic part in the arithmetic and logic unit (A.L.U.) of a computer. All arithmetic operations ; namely addition , subtraction , multiplication and division make use of this adder unit. Thus , the speed of any A.L.U. will be limited by the speed of its adder. In this part we expose fixed point binary addition , and algorithms for speeding-up the multiplication and division processes.

1.1. Binary Addition :

Addition algorithms depend on the particular way of representing the associated binary numbers (signed magnitude, signed 1's complement , signed 2's complement forms). Addition can also be classified as serial or parallel. In what follows we shall survey different adders and discuss the speeding-up techniques adopted.

1.1.1. Serial Adders :

The simple serial adder consists of a "full" or "3 input-2 output" adder whose carry output is fedback to the carry input terminal through a unit delay. This is a direct implementation of the standard equation for binary addition, namely :

$$\left.\begin{aligned} \text{Sum} \quad &\equiv S = A_i \oplus B_i \oplus C_i \\ \text{Output Carry} \quad &\equiv C_o = (A_i \oplus B_i)\, C_i + A_i\, B_i \end{aligned}\right\} \quad \ldots\ldots (1.1)$$

This is shown schematically in Fig. (1.1), where ;

A ≡ augend number

B ≡ addend number.

1.1.2. Parallel Adders :

Parallel adders could be implemented by a set of full - adders arrayed as shown in Fig. (1.2). This adder can also be envisaged as a synchronous adder if it produces a usable sum in fixed time regardless of the numbers being added ; this

time is equal to 2 n levels where n is the number of adder bits , where a one-level delay corresponds to the propagation delay of an AND or OR gate.

1.1.3. Assynchronous Adders :

Speeding up can be achieved when using assynchronous adders, [1,2,3] In this adder , shown in Fig. (1.3) , extra speed is achieved in two ways :

(a) The simultaneous initiation of carries at all stages where the carries are independent of preceding carries.

(b) The use of extra fast carry transmission gate to propagate carries [4] .

Fig. (1.3) shows a logic circuit realizing this adder. The equations for the 1-carry C^1 and 0-carry C^0 are :

$$\left.\begin{aligned} {}^1C_{i+1} &= {}^1G_i + {}^1C_i\, P_i \\ {}^0C_{i+1} &= {}^0G_i + {}^0C_i\, P_i \end{aligned}\right\} \quad \ldots\ldots (1.2)$$

where , ${}^1G_i = A_i\, B_i$

${}^0G_i = \bar{A}_i\, \bar{B}_i$

$P_i = A_i \oplus P_i$

This adder is an assynchronous adder , since its addition time depends upon the longest path of dependent carries generated by any particular pair of summands. Thus , the circuit has a carry completion gate which transmits a signal indicating that all the carries have been generated. Statistical analysis [2] has shown that on the average the time required for addition by this method is approximately the transit time for seven stages for a 48 bit-adder.

1.1.4. Carry Lookahead Adders :

The principle of carry lookahead [3,5] , or skip technique as named by Lehan and Burla [6] , is to examine a number of stages of the adder to produce simultaneously the carries for each of these stages. The carries are then applied to the adder block for each stage which produces the proper sum. The number of stages examined is limited by the maximum fan-in of the gates used. By separating the carry out of a particular stage into two parts , that produced internally ; the generate carry $G_k = A_k B_k$, and that produced externally and passed through ; the propagated carry $P_k C_{k-1} = (A_k + B_k) C_{k-1}$. The equation of the carry may then be expanded as follows :

$$C_k = G_k + P_k G_{k-1} + P_k P_{k-1} G_{k-2} + \cdots + P_k P_{k-1} \cdots P_1 C_0 \qquad (1.3)$$

where C_c is the carry input to the least significant stage of the adder.

As a result of the limitation imposed by the maximum fan-in of the gate used as well as the complexity of realizing carry lookahead arrangement for a considerable number of stages , several arrangements are usually adopted. Assume that five stages represent a reasonable number of adder stages to be connected together and designate such an arrangement as a " group ". The group containing the five low order positions of the adder will be group 1, ... etc. A carry out of group n will be C_{gn}. If these five-bit groups are now connected in series , this arrangement is known as zero level carry lookahead. A carry will require four levels to be produced and reach the output of the first group , two levels to go through each intermediate group , and four levels to reach and be assimilated into the sum in the final group. Thus , for five-bit groups the maximum carry path length is $4 \times (\frac{2n}{5})$ levels , where n is the number of adder bits. The arrangements to generate carry lookahead for any group is shown in Fig. (1.4).

For each group , a propagate signal , $P_{gn} = P_1 P_2 P_3 P_4 P_5$, is generated to take a carry across the group , where the numbers 1, 2, ... etc. refer to positions within the group. A generate signal , $G_{gn} = G_5 + P_5 G_4 + P_5 P_4 G_3 + \ldots\ldots\ldots + P_5 P_4 P_3 P_2 G_1$, which includes only carries originating within

the group is also generated. Now each five groups will be designated as a section , the carry-out of section m is C_{sm}. Within the section , carry lookahead arrangement is used between the groups. The sections are connected in series , with the resulting arrangement referred to as first level carry lookahead. The maximum path length for a carry to be generated , within a section and reach the output C_{sm} is six levels , two levels to go through each intermediate section , and six levels to produce the sum in the last section. The maximum carry path length is therefore $\left[12 + 2\,(M - 2)\right]$, where M is the numbers of sections.

A second level lookahead arrangement can be thought of if carry lookahead is used between a group of sections.

1.1.5 Conditional Sum Adder :

The conditional sum adder is one of the fastest adders available [1,7] . It is of special interest because of the small fan-in of the circuits used. The conditional sum method of performing addition requires that , for each pair of bits to be added , two results are created. The first result , one sum and one carry bit per stage , is prepared as though the carry into a stage were a 0. The second result is prepared as though the carry into a stage were a 1.

The second level result for a pair of stages is determined by the results of the first stage of the pair. This procedure is continued , examining the results in quartets , octets , ... n-tets. Fig. (1.5) illustrates the conditional sum adder procedure.

0	0	0	1	1	1	1	0	Carry in	Level
1	0	1	1	0	0	0	1		
0 1	0 0	0 1	1 0	0 1	0 1	0 1	0 1	0	1
1 0	0 1	1 0	1 1	1 0	1 0	1 0	1 0	1	
0 1	0	1 0	0	0 1	1	0 1	1	0	2
0 1	1	1 0	1	1 0	0			1	
0 1	1	0	0	0 1	1	1	1	0	3
0 1	1	0	1					1	
0 1	1	0	0	1	1	1	1	0	4
								1	

Fig. (1.5) An addition by the conditional sum method.