Pure and Applied Mathematics Journal
2017; 6(3): 89-100
http://www.sciencepublishinggroup.com/j/pamj
doi: 10.11648/j.pamj.20170603.11
ISSN: 2326-9790 (Print); ISSN: 2326-9812 (Online)

Design of ALU and Code Converter Using Matrix Calculation

Nirina Gilbert Rasolofoson, Raoelina Andriambololona

Theoretical Physics Department, Institut National des Sciences et Techniques Nucléaires (INSTN-Madagascar), Antananarivo, Madagascar

Email address:
nirizi.rasolofoson@gmail.com (N. G. Rasolofoson), raoelinasp@yahoo.fr (R. Andriambololona), raoelina.andriambololona@gmail.com (R. Andriambololona), instn@moov.mg (R. Andriambololona), jacquelineraoelina@hotmail.com (R. Andriambololona)

To cite this article:
Nirina Gilbert Rasolofoson, Raoelina Andriambololona. Design of ALU and Code Converter Using Matrix Calculation. *Pure and Applied Mathematics Journal.* Vol. 6, No. 3, 2017, pp. 89-100. doi: 10.11648/j.pamj.20170603.11

Received: April 3, 2017; **Accepted**: April 15, 2017; **Published**: May 27, 2017

Abstract: Arithmetic Logic Unit (ALU) is a fundamental building block of a central processing unit (CPU) in any computing system. The ALU is the hardware that performs logical (and, or, xor) and basic arithmetic (addition, subtraction, multiplication, division) operations. Thus, its construction requires techniques in which the treatment of operands should be consistent with operations rules. In this paper, ALU based on matrix calculation introduced and developed by Raoelina Andriambololona is proposed. These techniques aim to remove illogic and inconsistent appearing in the international writing numeration with the usual rules in arithmetic. We also propose the design of code converters which convert Binary to BCD (Binary Coded Decimal) code and vice versa using matrix calculation.

Keywords: ALU, Arithmetic, Numeration, Matrix Calculation, Code Converter

1. Introduction

There are incoherencies and illogical between usual written numeration and arithmetic rules. For instance, in English language, the number 216 is written as two one six but read as two hundred sixteen (i.e. "261" instead of 216). In many languages (English, French, German, Malagasy) the same inconsistency exists [2-3-4].

In order to avoid these problems, Raoelina Andriambololona has investigated and used matrix calculation and linear algebra tools [1-2-3-4-13]. This approach exhibits four and only four possibilities to write a number according to the disposition in row matrix or column matrix and in decreasing or increasing order. It has been shown by Raoelina Andriambololona that the writing in line from Left (L) handside to the Right (R) handside by increasing (i) order (LRi disposition) is more logical and coherent with the basic arithmetic operations rules than the international which starts from the left (L) handside to the right (R) handside by decreasing (d) order (or LRd). For instance, the number 216 written in usual one (LRd disposition) is written as 612 in LRi disposition. Thus, new rules for the addition, subtraction, multiplication and division operation which are homogeneous and consistent with the LRi disposition are established [2-3]. Raoelina Andriambololona's work has led him also to the study of the problem of numeral basis change by using the basis change in a linear space through a passage matrix.

The result thus obtained is simpler and more direct than the usual one utilizing the euclidian successive division [3-6]. These results are helpful for the construction of supercomputer system. This leads us to design new ALU and code converters which follow new rules established using matrix calculation. In this work, the components of the arithmetic unit such as addition, subtraction, multiplication and division of the proposed ALU are explicited and designed according to LRi disposition. As for the code converters, we only use full adders to perform calculations from the change of basis matrix.

In the present work, the vocabulary "radix" is used to express the basis of a number and any number will be written according to the LRi disposition. For instance, in the radix 10 the number spelt as one hundred twenty five will be written by 521_{10}. Besides, the direction of the basic arithmetic operations starts from the left handside to the right handside according the LRi writing and not always from the right handside to the left handside as in usual operation.

2. Utilization of Matrix Calculation in Numeration and Arithmetics

2.1. Matrix Representation for Writing Numeration and Numeral Basis Change

The matrix formalism consists to consider the numeral representation of a number as a matrix representation of an intrinsic number in a basis which represents the numeral system. The LRi disposition is shown to be more logical and consistent with the arithmetic operations rules than the usual one which is LRd. According to LRi writing, an intrinsic number N will be written as a row matrix $[a_{-m}a_{-m+1}\cdots a_{-1}a_0a_1\cdots a_{n-1}a_n]$ from left (L) handside to the right (R) handside by increasing (i) order in the basis numeral system with radix b with the vector basis $\overline{b}$ [1-3].

$$\overline{b} = \begin{bmatrix} b^{-m+0} \\ b^{-m+1} \\ \vdots \\ b^{-1+m} \\ b^{-m+0} \\ b^{-1+h} \\ \vdots \\ b^{+n-1} \\ b^{+n+0} \end{bmatrix}$$

$$N = [a_{-m}a_{-m+1}\cdots a_{-1}a_0a_1\cdots a_{n-1}a_n].\overline{b} \qquad (1)$$

In the change of radix b' represented by vector basis $\overline{b'}$

$$\overline{b'} = \begin{bmatrix} b'^{-m+0} \\ b'^{-m+1} \\ \vdots \\ b'^{-1+m} \\ b'^{-m+0} \\ b'^{-1+h} \\ \vdots \\ b'^{+n-1} \\ b'^{+n+0} \end{bmatrix}$$

is given by

$$\overline{b} = [P].\overline{b}' \qquad (2)$$

where the change of basis matrix is $[P]$. Therefore, the writing of the number N in the radix b' will be

$$[a_{-m}a_{-m+1}\cdots a_{-1}a_0a_1\cdots a_{n-1}a_n].[P] \qquad (3)$$

Example 1: the number usually spelt as ninety thousand two hundred and thirty one in the usual language is written in the numeral system with the radix 10

$$N = 13209_{10} = [1\ 3\ 2\ 0\ 9]\begin{bmatrix} 10^0 \\ 10^1 \\ 10^2 \\ 10^3 \\ 10^4 \end{bmatrix}$$

Example 2: Convert 74_{10} into binary.

The matrix representing the basis change from radix 10 to radix 2 is

$$\begin{bmatrix} 10^0 \\ 10^1 \end{bmatrix} = \begin{bmatrix} 1\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix}_2 \begin{bmatrix} 2^0 \\ 2^1 \\ 2^2 \\ 2^3 \end{bmatrix}$$

Hence the writing in binary basis of the decimal number is

$$74_{10} = [74]\begin{bmatrix} 1\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \end{bmatrix}_2 = +\ \begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 0 \\ & & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 1 \\ \hline & & 1 & 1 & 1 & 0\ 1 \end{array}$$

$\hookrightarrow direction\ of\ operation$

$$= 111101_2$$

It may be checked easily

$$111101_2 = 1\times 2^0 + 1\times 2^1 + 1\times 2^2 + 1\times 2^3 + 0\times 2^4 + 1\times 2^5 = 74_{10}$$

Example 3: Convert 153_8 into hexadecimal.

Provided with the change of basis matrix of the two basis

$$[P]_{8\to 16} = \begin{bmatrix} 1\ 0 \\ 8\ 0 \\ 0\ 4 \end{bmatrix}_{16} \qquad (4)$$

we have

$$153_8 = [153]\begin{bmatrix} 1\ 0 \\ 8\ 0 \\ 0\ 4 \end{bmatrix}_{16} = +\ \begin{array}{cc} 9 & 2 \\ & \mathrm{C} \\ \hline 9 & \mathrm{E} \end{array} = 9\mathrm{E}_{16}$$

$\hookrightarrow$

where the hexadecimal numbers are $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$.

2.2. Utilization of Matrix Calculation in Arithmetics

2.2.1. Rules for Addition

These rules are obtained from [4]. We have for two positive integer A and A′

$$A = W_b(A)\,\overline{b} = \sum_{i=0}^{n} a_i b^i \qquad (5)$$

$$A' = W_b(A')\,\overline{b} = \sum_{i=0}^{n'} a'_i b^i \qquad (6)$$

We can suppose $n' > n$ with $a'_n \neq 0$ for $n+1 \leq i \leq n'$, without loosing in generality.

$$A + A' = W_b(A + A')\,\overline{b} = \sum_{i=0}^{n} a_i b^i + \sum_{i=0}^{n'} a'_i b^i$$

$$= \sum_{i=0}^{n"} (a_i + a'_i) b^i = \sum_{i=0}^{n"} a"_i b^i \qquad (7)$$

The integer $n"$ must be chosen so that $a"_i < b$ for all i and in particular for $i = n'$. In fact we know that the elements $a"_i$ of the row matrix $W_b(A + A')$ must be null or strictly less than b:

- If $a_i + a'_i < b$, then we have $a"_i = a_i + a'_i$ is the element at the $(i+1)^{th}$ column (from the left handside) of the matrix $W_b(A + A')$.
- If $a_i + a'_i > b$ we perform the decompositions

$$a"_i = a_{is}b^s + \cdots + a_{i1}b^1 + a_{i0}b^0 \text{ with } a_{is} < b$$

$$a"_i b^i = a_{is}b^{s+i} + \cdots + a_{i1}b^{1+i} + a_{i0}b^i$$

without summation on i. The element at the $(i+1)^{th}$ column from the left handside of $W_b(A + A')$ is $a"_i = a_{i0}$ while the remainders $a_{i1}, a_{i2}, \cdots$ are to be brought respectively at the $(i+2)^{th}, (i+3)^{th}, \cdots (i+s+1)^{th}$ column (from the left) of $W_b(A + A')$.

Example 4: perform the addition of 126_{10} and 93868_{10}. The intrinsic number A and A' are

$$A = [1\ 2\ 6\ .\ .] \begin{bmatrix} 10^0 \\ 10^1 \\ 10^2 \\ . \\ . \end{bmatrix}$$

$$A' = [9\ 3\ 8\ 6\ 8] \begin{bmatrix} 10^0 \\ 10^1 \\ 10^2 \\ 10^3 \\ 10^4 \end{bmatrix}$$

The dot. represents 0 (zero). The addition rules are easily obtained and may be compared with the usual one LRd.

Proposed disposition: LRi

carry →		*1*	*0*	*1*	*0*
+	1	2	6	.	.
	9	3	8	6	8
	0	6	4	7	8

↪ direction of operation
↪ direction of the writing

Usual disposition: LRd

	0	*1*	*0*	*1*		← carry
+	.	.	6	2	1	
	8	6	8	3	9	
	8	7	4	6	0	

direction of operation ↩
↪ direction of the writing

2.2.2. Rules for Subtraction

The subtraction of a number $A"$ from a number A is the research of the number A' such as $A" = A + A'$. It is the inverse operation of addition. We can also use the inverse of the addition operation and establish the LRi procedure. The rules can be obtained by analogy with the addition rules. [2]

Example 5: subtract 06478_{10} from 126_{10}

Proposed disposition: LRi

	0	6	4	7	8	
-	*0*	*1*	*0*	*1*	*0*	← carry
	1	2	6	.	.	
	9	3	8	6	8	

↪ direction of operation
↪ direction of the writing

Usual disposition: LRd

	8	7	4	6	0	
-	*0*	*1*	*0*	*1*	*0*	← carry
	.	.	6	2	1	
	8	6	8	3	9	

direction of operation ↩
↪ direction of the writing

2.2.3. Rules for Multiplication

Let be

$$A \times A' = W_b(AA')B = \sum_{i=0}^{n"} (aa')_i b^i \qquad (8)$$

For the explicit calculation, we proceed as for the case of the addition in respecting the places

$$A \times A' = \sum_{m=0}^{n"} \left(\sum_{p+q=0}^{m} a_p a_q{}' \right) b^m$$

$$= a_0 a'_0 b^0 + \sum_{i+k=0}^{1} (a_i a_k{}')b^1 + \sum_{i+k=0}^{2} (a_i a_k{}')b^2 + \cdots$$

$$A \times A' = P_0 b^0 + P_1 b^1 + P_2 b^2 \qquad (9)$$

In the calculation of the component $(A \times A')_i$, of the product $A \times A'$, which must be strictly less than b, we must bring the remainder in the partial sum. [2]

Example 6: multiply 518_{10} by 5216_{10}

Proposed disposition: LRi

x	5	1	8	.	
	5	2	1	6	
	5	7	0	4	
		0	3	6	1
			5	1	8

Usual disposition: LRd

	.	8	1	5	
x	6	1	2	5	
	4	0	7	5	
1	6	3	0		
1	8	1	5		

Proposed disposition: LRi

			0	9	8	4	
5	7	8	1	9	9	4	

↳ direction of operation
↳ direction of the writing

Usual disposition: LRd

4	8	9	0				
4	9	9	1	8	7	5	

direction of operation ↩
↳ direction of the writing

2.2.4. Rules for Division

The division is the inverse operation of multiplication. The division of a number A'' by a number A is the operation which gives the number A' such as $A'' = A \times A'$ [2]. According to the LRi procedure its rules differ radically from the usual one which starts from the left handside to the right handside by decreasing order LRd. The division starts from left handside by subtracting the least significant digit of the dividend from the divisor. Then we shift the divisor one place to the right and perform again the subtraction. Shifts end to the right where the most significant digit is. If the remainder is still greater than the divisor, we repeat the same operation from the left handside to the right handside until getting remainder less than the divisor. The final result is obtained from the addition of the partial quotients.

The advantages of this method are that it respects the writing numeration and the inverse operation of the multiplication according to the LRi disposition. In addition, we can usually get from the beginning the remainder.

Example 7: division of 9932_{10} by 4_{10}

Proposed disposition: LRi

9	9	3	2		4			
6	3				9			
3	6	3	2					
↓	6	3		+		9		
↓	0	0	2					
↓		0	2				5	
3	.	.	.		9	9	5	

↳ direction of operation
↳ direction of the writing

Usual disposition: LRd

2	3	9	9		4			
2	0				5			
	3	9	9					
	3	6		+		9		
		3	9					
		3	6				9	
.	.	.	3		5	9	9	

↳ direction of operation
↳ direction of the writing

Example 8: division of 4738_{10} by 21_{10}

Proposed disposition: LRi

4	7	3	8		2	1		
8	0	1			9			
6	6	2	8					
	8	0	1			9		
6	8	1	7					
		0	6				5	
6	8	1	1	+				
8	0	1			9			
8	7	0	1					
	6	9				8		
8	1	1	.					
8	0	1			9			
0	1				7	9	6	

↳ direction of operation
↳ direction of the writing

Usual disposition: LRd

8	3	7	4		1	2		
7	2				6			
1	1	7	4					
1	0	8		+		9		
.	.	9	4					
		8	4				7	
		1	0		6	9	7	

↳ direction of operation
↳ direction of the writing

Example 9: division of 1109_{10} by 154_{10}

Proposed disposition: LRi

1	1	0	9		1	5	4	
9	5	0	4		9			
2	5	9	4	+				
	1	5	4			1		

Usual disposition: LRd

9	0	1	1		4	5	1	
4	5	1			1			
4	5	0	1	+				
4	0	5	9			9		

Proposed disposition: LRi	Usual disposition: LRd
2 4 4 . | 9 1	. 4 4 2 | 1 9
↳ direction of operation	↳ direction of operation
↳ direction of the writing	↳ direction of the writing

Example 10: division of 5063_{10} by 1402_{10}

Proposed disposition: LRi	Usual disposition: LRd
5 0 6 3 | 2 0 4 1	3 6 0 5 | 1 4 0 2
4 0 8 2 | 2	2 8 0 4 | 2
1 0 8 .	8 0 1
↳ direction of operation	↳ direction of operation
↳ direction of the writing	↳ direction of the writing

3. Application of Matrix Calculation in Arithmetic Circuits

3.1. Design of Arithmetic Logic Unit (ALU)

An ALU typically has three input words which are the two input operands, denoted by A and B, and the carry-in input c_{in}. The output words are the result S and the carry-out c_{out}. Besides data inputs and outputs, an ALU must have control inputs to specify the operations to be performed. All these data are coded in Binary.

The major parts of ALU are [7-10]:

- Arithmetic block: This block is used to perform four arithmetic operations such as addition, subtraction, multiplication and division;
- Logic block: This block is used to perform simple bitwise logic operations such as AND, OR, XOR, XNOR, NAND, NOT and etc;
- Multiplexers: These blocks are used to select the appropriate inputs for the arithmetic and logic blocks

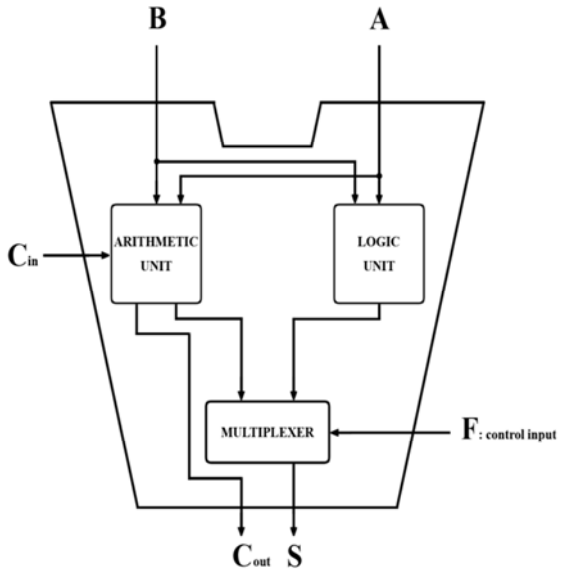

Figure 1. *Block Diagram of one-bit ALU.*

Here we apply the new rules established for arithmetic using matrix calculation and LRi writing to construct the components of the ALU's arithmetic unit which are addition, subtraction, multiplication and division.

3.1.1. Binary Addition - Subtraction

Let $A = (a_0, a_1 \cdots, a_{n-1})_2$ and $B = (b_0, b_1 \cdots, b_{n-1})_2$ be two n-bit numbers respectively. As the rules for arithmetic operations are similar for any basis binary, the addition is performed by adding bit by bit the components located at the same row of the matrix representing and bringing an eventual carry at the next row. [2-8].

Example 11: addition of 1001_2 and 1101_2.

carry →		*0*	*1*	*1*	*0*	*1*
	+	1	0	0	1	
		1	1	0	1	
		0	0	1	0	1

↳ direction of operation

The structure performing the addition of two n-bits numbers is realized from 1-bit full adder.

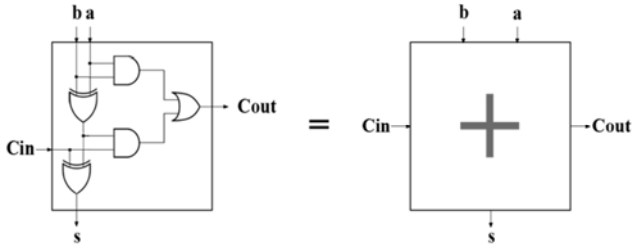

Figure 2. *One-bit Full Adder.*

The results are expressed as

$$s = a\ xor\ b\ xor\ c_{in} \tag{10}$$

$$c_{out} = c_{in}\ and\ (a\ xor\ b)\ or\ a\ and\ b \tag{11}$$

These relationships can be seen when considering the full adder's truth table, shown below:

Table 1. *One-bit Full Adder Truth Table.*

b	a	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

We can have multi-bit Full Adder from the one-bit Full Adder by chaining the carry bits, such that $c_{in_{i+1}} = c_{out_i}$, as shown in Figure 3.

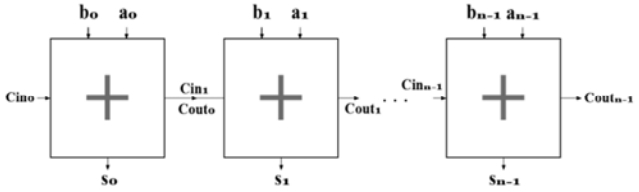

Figure 3. *Block diagram of n-bits Full Adder according to LRi disposition.*

The multi-bit Full Adder can be represented simply as follows

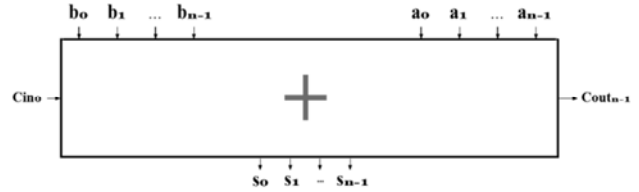

Figure 4. *Block diagram of n-bits Full Adder according to LRi disposition.*

Having an Adder which operates $A + B$, then subtractor performing $B - A$, using 2's complement theory, is obtained by inverting the subtrahend A and adding one: $B - A = B + \overline{A} + 1$. Besides the A and B inputs, we have introduced a control input Op. When this signal is 0, the circuit performs addition. When it is 1, the circuit becomes a subtractor. [10-11]

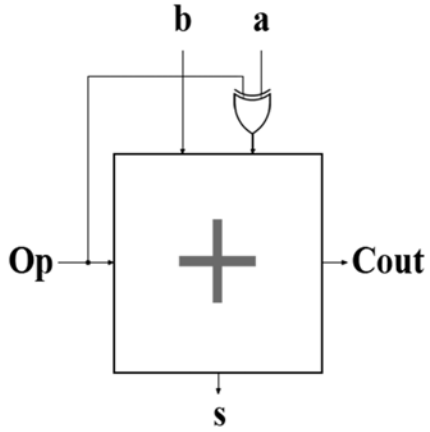

Figure 5. *One-bit Full Adder / Subtractor.*

Its truth table is shown in the table below

Table 2. *One-bit Full Adder - Subtractor Truth Table.*

b	a	Op	s	c_{out}
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

The same way as the n-bits Full Adder we can have the following structure

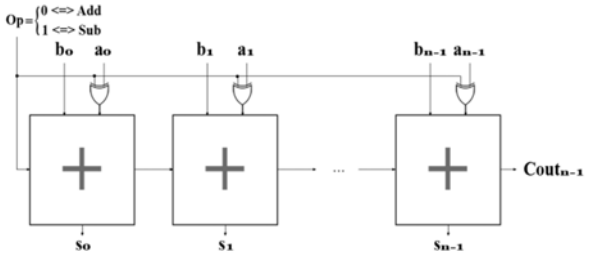

Figure 6. *Block diagram of n-bits Full Adder – Subtractor according to LRi disposition.*

3.1.2. Binary Multiplication

The multiplication is realized by performing the addition of the partial product with itself shifted 1 bit to the right in each step until n times. Partial product is equal to the multiplicand when the concerned multiplier is 1 if not it is 0 for multiplier set 0. [2-8]

Example 12: 0101_2 by 101_2

multiplicand →		0	1	0	1		
multiplier →	x	1	0	1			
		0	1	0	1		
			0	0	0	0	
				0	1	0	1
		0	1	0	0	1	1

↪ direction of operation

As the partial products can be performed with AND logical and their sum by n-bit full adders multiplier circuit performing $A \times B$ is shown in figure 7 [12]

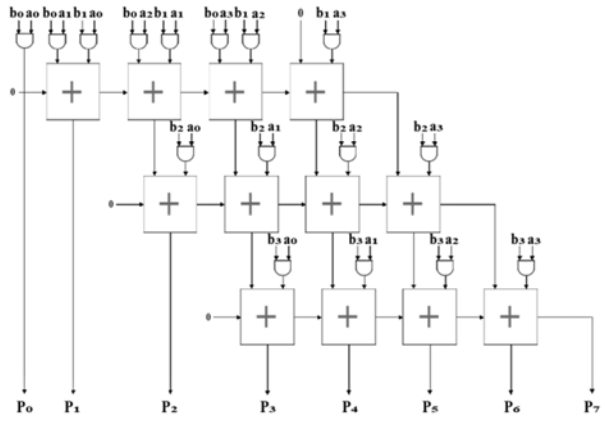

Figure 7. *Block diagram of 2n-bits Multiplier according to LRi disposition.*

In order to make this technique straightforward to implement in programmable logic the algorithm below is helpful.

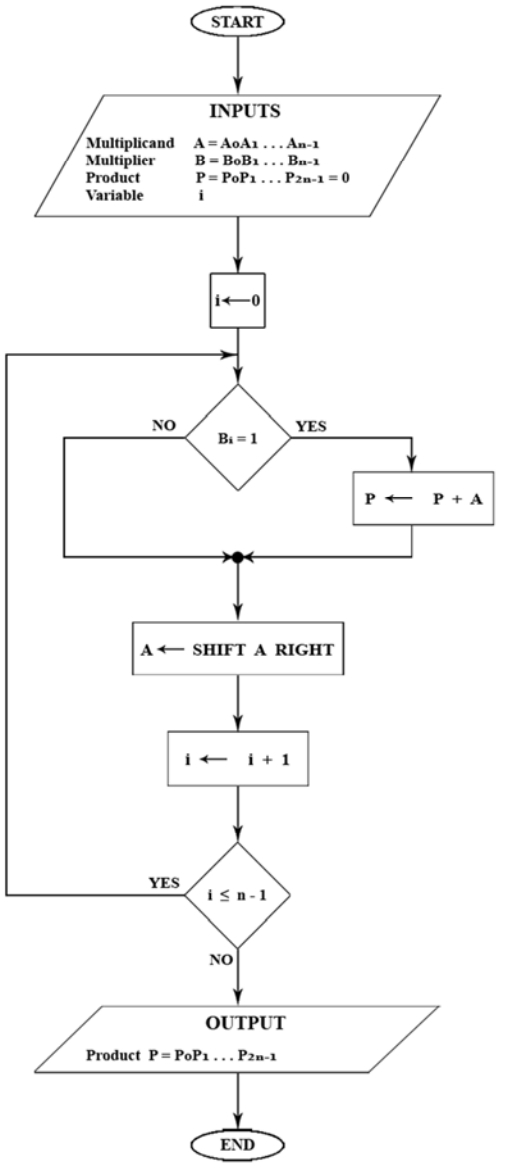

Figure 8. *Binary Multiplication Algorithm.*

3.1.3. Binary Division

The binary division follows the same rules as the decimal division by starting the subtraction of the dividend by the divisor from the left handside and shifting 1 bit to the right the divisor in every step. The final result is obtained by the sum of the partial quotients. [2]

Example 13: division of 1011000_2 by 0100_2

1	0	1	1	0	0	0		0	1	0	0
0	1	0	0					1			
1	1	0	1	0	0	0					
	0	1	0	0					1		
1	1	1	0	0	0	0					
		0	0	0	0					0	
1	1	1	0	0	0	0					
			0	0	0	0					0
1	1	1	0	0	0	0	+				
0	1	0	0					1			
1	0	1	0	0	0	0					
	0	1	0	0					1		
1	0	0	0	0	0	0					
		0	0	0	0					0	
1	0	0	0	0	0	0					
			0	0	0	0					0
1	.	.	.	.	.	.		0	1	1	0

↪ direction of operation

Example 14: division of 1111000_2 by 1100_2

1	1	1	1	0	0	0		1	1	0	0
1	1	0	0					1			
0	0	1	1	0	0	0					
	1	1	0	0					1		
0	1	1	0	0	0	0					
		0	0	0	0					0	
0	1	1	0	0	0	0					
			0	0	0	0					0
0	1	1	0	0	0	0	+				
1	1	0	0					1			
1	1	0	0	0	0	0					
	0	0	0	0					0		
1	1	0	0	0	0	0					
		0	0	0	0					0	
1	1	0	0	0	0	0					
			0	0	0	0					0
1	1	0	0	0	0	0					
1	1	0	0					1			
.	.	.	.	.	.	.		1	0	1	0

↪ direction of operation

These examples show that

- Binary division is a cascade of subtraction and shift operations;
- The quotient is equal to the inverse of the last borrow of the subtractions.

The divisor circuit is thus constructed using subtractors in which multiplexers are integrated to select the output of the subtraction. Adders are used to sum the partial quotients.

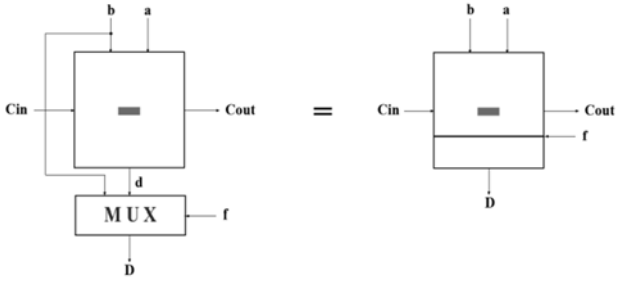

Figure 9. *One-bit Subtractor-Multiplexer.*

The multiplexer is shown in figure 10

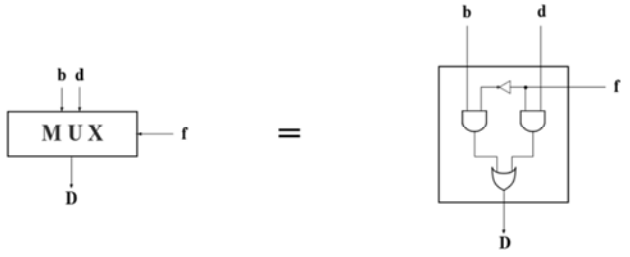

Figure 10. *2: 1 Multiplexer.*

Table 3. *2: 1 Multiplexer Truth Table.*

d	***a***	***c***	***D***
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The output D is expressed as

$$D = d\ and\ \overline{c}\ or\ a\ and\ c \qquad (12)$$

Cascading the subtractor-multiplexer modules permits the subtraction of the dividend from the divisor and the generation of the quotient of the division:

- The modules first subtract the dividend from the divisor from the least significant bit on the left handside. If the subtraction output borrowing r_{n-1} is zero, that is, the dividend is greater than the divisor; setting the input control bits c of each module to this value allows the calculation of the division remainders which are the new dividend. Otherwise, if $r_{n-1} = 1$, the subtraction result will not be considered and the new dividend will be the same as the previous. After each subtraction operation the divisor is shifted 1 bit to right until the subtraction of the most significant bit. If the dividend is still greater than the divisor we repeat the same process which starts from left until we get dividend less than the divisor.
- Each subtraction matches a quotient bit which is the inverse of the output borrowing $\overline{r}_{n-1}$.
- The final quotient of the division is the sum of the partial quotients $\sum \overline{r}_{n-1}$

Considering these factors and the new rules established for division, the figure 11 shows the divisor circuit according to the LRi disposition.

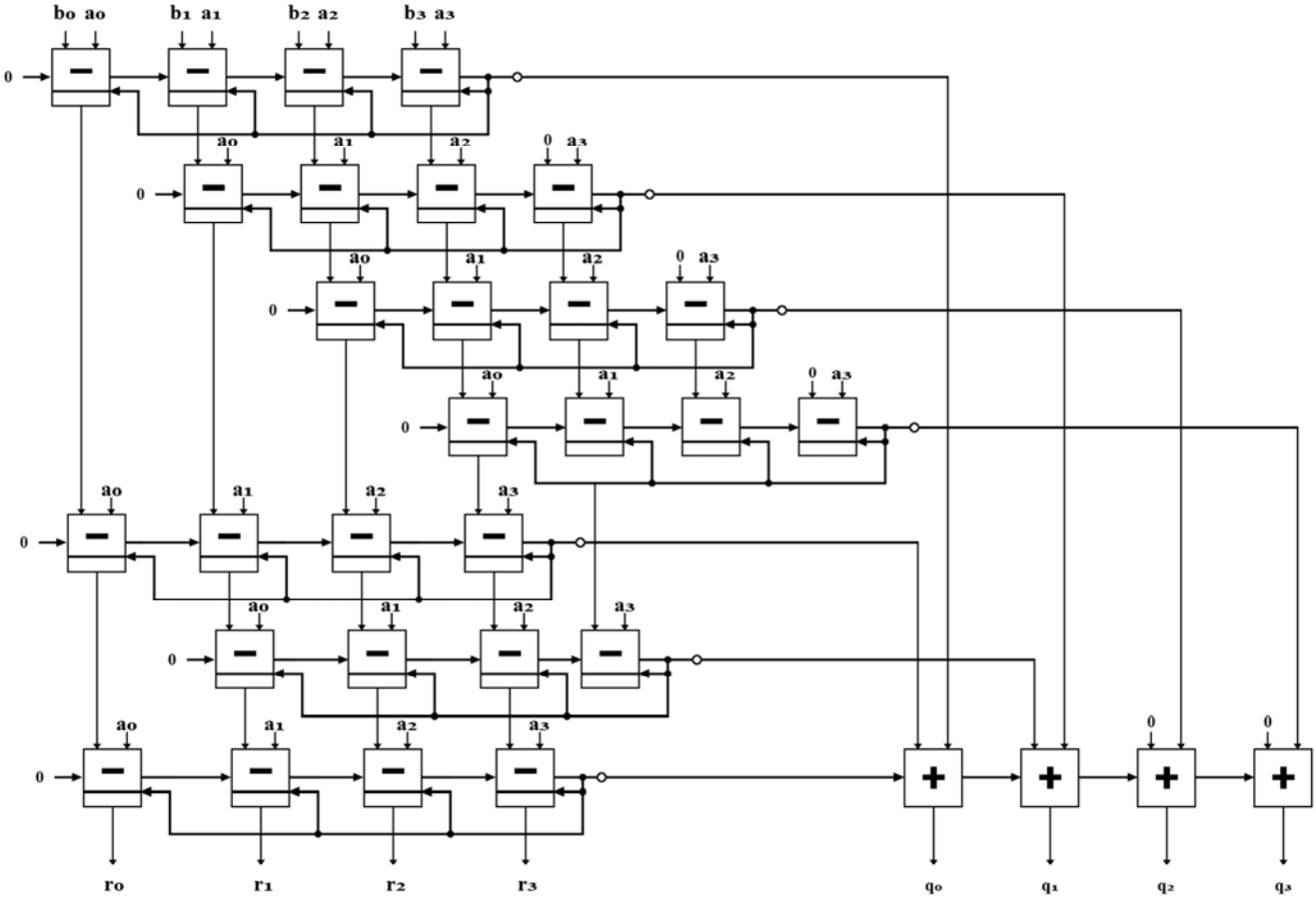

Figure 11. *Block diagram of four-bits divisor according to LRi disposition.*

The following algorithm helps illustrate this method

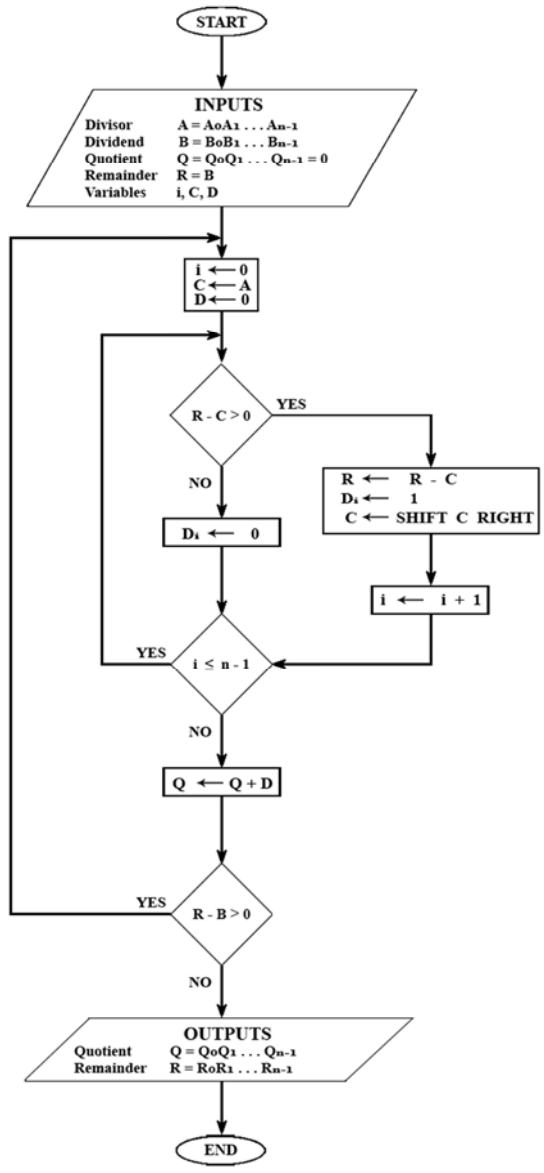

Figure 12. *Binary Division Algorithm.*

The components of the arithmetic unit are done according to the new rules established for the basic arithmetic operations.

3.2. Design of Code Converters

After operation results from the ALU outputs are usually coded in binary. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit should be inserted between the two systems if each uses different codes for the same information. Thus, a code converter is a circuit that makes the two systems compatible even though each uses a different code. This particular section deals with the design of binary to BCD (Binary Coded Decimal) code converter and vice versa using matrix calculation.

For practical purposes decimal system can be represented in BCD or Binary Coded Decimal which represents the 10 decimal digits in terms of binary numbers in arithmetic circuits.

Table 4. *Binary and BCD encoding according to LRi writing.*

Binary Code				Decimal Code	BCD Code							
0	0	0	0	00	0	0	0	0	0	0	0	0
1	0	0	0	10	1	0	0	0	0	0	0	0
0	1	0	0	20	0	1	0	0	0	0	0	0
1	1	0	0	30	1	1	0	0	0	0	0	0
0	0	1	0	40	0	0	1	0	0	0	0	0
1	0	1	0	50	1	0	1	0	0	0	0	0
0	1	1	0	60	0	1	1	0	0	0	0	0
1	1	1	0	70	1	1	1	0	0	0	0	0
0	0	0	1	80	0	0	0	1	0	0	0	0
1	0	0	1	90	1	0	0	1	0	0	0	0
0	1	0	1	01	0	0	0	0	1	0	0	0
1	1	0	1	11	1	0	0	0	1	0	0	0
0	0	1	1	21	0	1	0	0	1	0	0	0
1	0	1	1	31	1	1	0	0	1	0	0	0
0	1	1	1	41	0	0	1	0	1	0	0	0
1	1	1	1	51	1	0	1	0	1	0	0	0

The number 857_{10} will be represented, for instance, by $0001\ 1010\ 1110_{\mathrm{BCD}}$. In BCD, counting is always performed in radix $\mathrm{b} = 10$, that is the greatest digit of the 4-bit binary string is $9_{10} = 1001_2$. Consequently we have to make correction when exceeding 9 by adding 6 (0110) to generate carry brought to next row. [9]

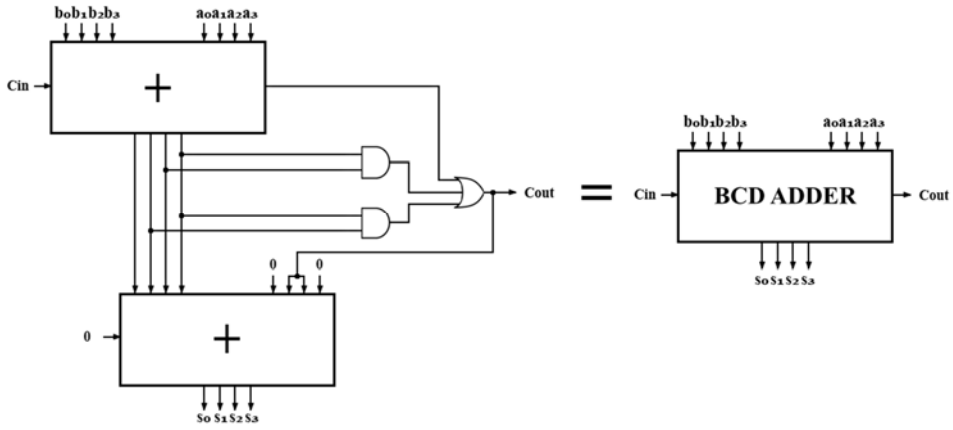

Figure 13. *Four-bits BCD Adder.*

Considering this last condition, we can apply the matrix calculation method to convert binary numbers into BCD number and vice versa.

Example 15: convert the binary digit 11111111_2 (which is 552_{10}) into BCD

$$N = [11111111] \otimes \begin{bmatrix} (1000) & (0000) & (0000) \\ (0100) & (0000) & (0000) \\ (0010) & (0000) & (0000) \\ (0001) & (0000) & (0000) \\ (0110) & (1000) & (0000) \\ (0100) & (1100) & (0000) \\ (0010) & (0110) & (0000) \\ (0001) & (0100) & (1000) \end{bmatrix} \otimes \begin{bmatrix} \overline{0000\ 1000}^{0} \\ \overline{0000\ 1000}^{1} \\ \overline{0000\ 1000}^{2} \\ \overline{0000\ 1000}^{3} \\ \overline{0000\ 1000}^{4} \\ \overline{0000\ 1000}^{5} \\ \overline{0000\ 1000}^{6} \\ \overline{0000\ 1000}^{7} \end{bmatrix}$$

where, in the row matrix,

$$1 = I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$N = + \begin{array}{lll} (1000) & & \\ (0100) & & \\ (0010) & & \\ (0001) & +0110 & \\ (0110) & +0110 & \\ (0100) & & \\ (0010) & & \\ (0001) & +0110 & \\ & (0000) & \\ & (0000) & \\ & (0000) & \\ & (0000) & \\ & (1000) & \\ & (1100) & \\ & (0110) & +0110 \\ & (0100) & \\ & & (0000) \\ & & (0000) \\ & & (0000) \\ & & (0000) \\ & & (0000) \\ & & (0000) \\ & & (0000) \\ & & (1000) \\ \hline \underbrace{(1010)}_{5} & \underbrace{(1010)}_{5} & \underbrace{(0100)}_{2}{}_{BCD} \end{array}$$

The examination of this example shows that a binary to BCD converter is realized with simple adders and BCD adders. As the digit 1 is less than 0 in the basis change, the binary numbers to be converted are connected to the digits 1 of the operands. We thus propose in figure 12 the design of a binary converter to BCD using matrix calculation.

Example 16: (Reverse calculation): convert into binary the BCD number $1010\ 1010\ 0100_{BCD}$

$$N = [(1010) \quad (1010) \quad (0100)] \otimes \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} \overline{2}^{0} \\ \overline{2}^{1} \\ \overline{2}^{2} \end{bmatrix}$$

where, in the passage matrix,

$$1 = I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } 0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

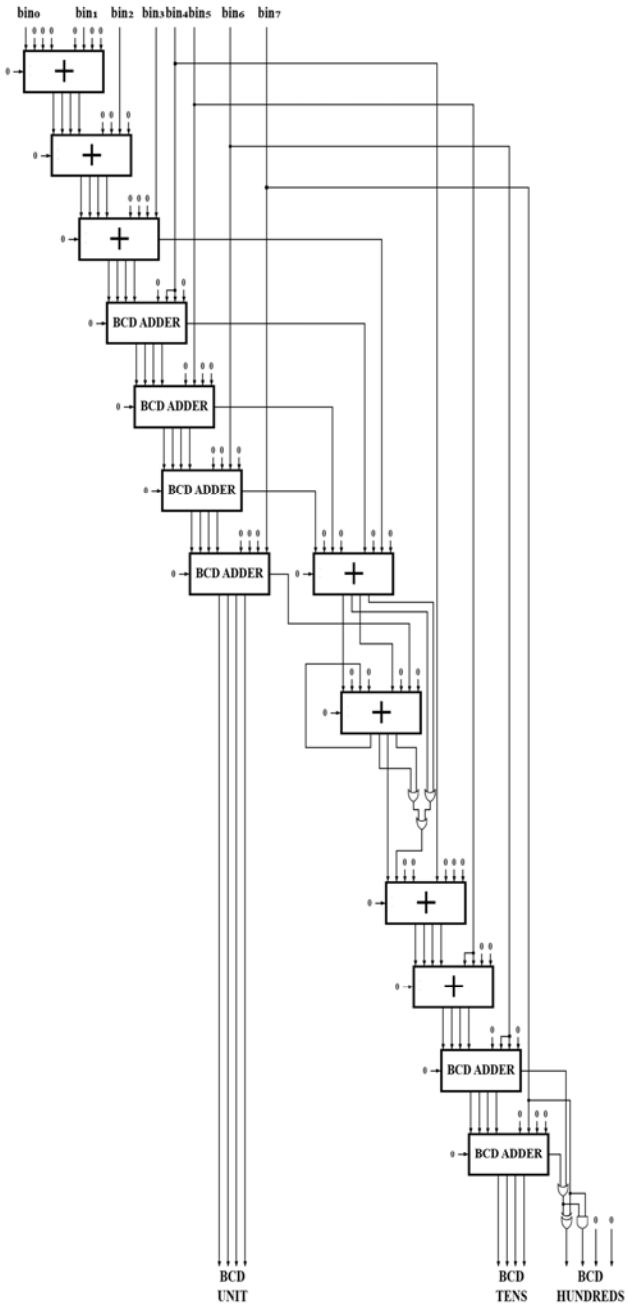

Figure 14. *Binary to BCD Code Converter.*

$$N = + \begin{array}{l}
1\,0\,1\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
1\,0\,1\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,1\,0\,0 \\
0\,0\,0\,0 \\
1\,0\,1\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,1\,0\,0 \\
0\,0\,0\,0 \\
0\,0\,0\,0 \\
0\,1\,0\,0 \\
\hline
1\,1\,1\,1\,1\,1\,1\,1\,0\,0_2
\end{array}$$

This example shows the design of a BCD to binary converter is only made of simple adders which sum the BCD digits activated by the digits 1 in the matrix of basis change.

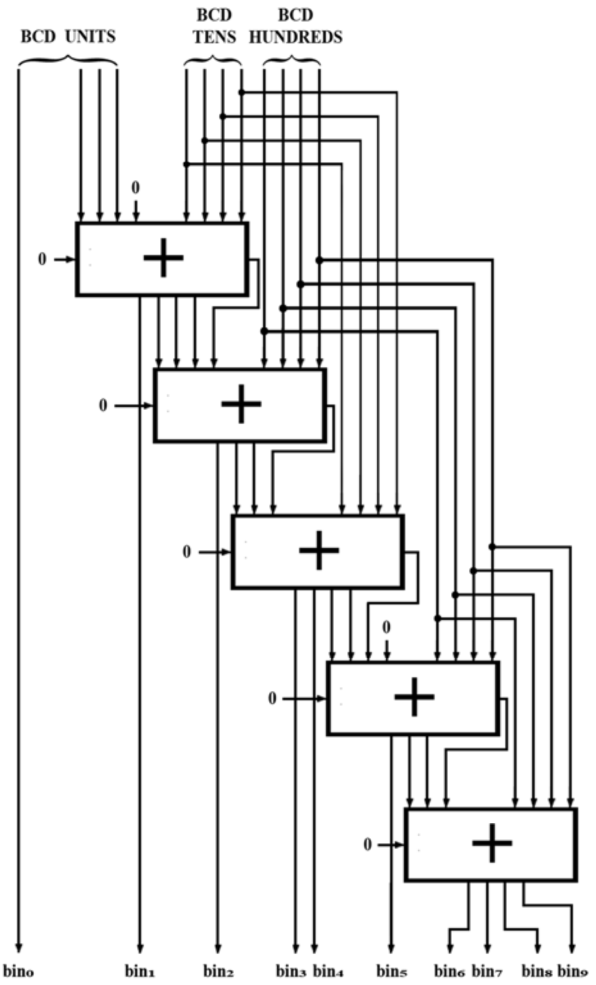

Figure 15. *BCD to Binary Code Converter.*

4. Conclusion

The utilization of matrix calculation illuminates the writing and the disposition of operations on numbers. In addition, as spoken numeration must be consistent with the writing numeration, the writing and reading of numbers have to be performed by increasing order. It supposes changes in the design of computer system. In this paper, circuits for basic arithmetic operations such as adder, subtractor, multiplier and divisor are designed and implemented in the arithmetic unit of the ALU according to the LRi writing and using matrix calculation. We propose design of code converters based on matrix calculation to convert the output code to another code such as binary to BCD code and inversely too. As the quantum computations are consistent with matrix calculations, we think that the application of these algorithms to quantum circuits is also promising.

References

[1] Raoelina Andriamblololona, "Théorie générale des numérations écrite et parlée". Bull. Acad. Malg. LXIV./1-2, Antananarivo, Madagascar, 1986.

[2] Raoelina Andriambololona, "Théorie générale des numérations écrite et parlée. II Utilisation du calcul matriciel en arithmétique. Nouvelle proposition d'écriture, d'énoncé des règles d'addition et de multiplication des nombres.".Bull. Acad.Malg LXV/1-2, Antananarivo, Madagascar, 1987.

[3] Raoelina Andriambololona, "Théorie générale des numérations écrite et parlée. II- Utilisation du calcul matriciel en arithmétique. Application au changement de bases de numération. Bull. Acad. Malg. LXV./1-2, Antananarivo, Madagascar ", 1987 (1989).

[4] Raoelina Andriambololona, Ravo Tokiniaina Ranaivoson, Wilfrid Chrysante Solofoarisina. Arithmetic and Matricial Calculation. Pure and Applied Mathematics Journal. Vol. 5, No. 3, 2016, pp. 82-86. doi: 10.11648/j.pamj.20160503.14.

[5] Raoelina Andriambololona, Hanitriarivo Rakotoson "Mpikajy elekronika sy siantifika mampiasa ny fomba fanisana Malagasy (Electronic and scientific calculator based on malagasy counting method)", communication at the Academie Malgache, Antananarivo Madagascar, 05 June 2008.

[6] Raoelina Andriambololona, "Algèbre linéaire et multilinéaire", Collection LIRA, INSTN-Madagascar, Antananarivo, Madagascar, 1986.

[7] Priyanka Yadav, Gaurav Kumar, Sumita Gupta, "Design and Implementation of 4-Bit Arithmetic and Logic Unit Chip with the Constraint of Power Consumption", IOSR Journal of Electronics and Communication Engineering, 2014.

[8] M. Morris Mano, Charles Kime. "Logic and computer design fundamentals." (4th ed.). Pearson, 2014.

[9] Osama Al-Khaleel, Mohammad Al-Khaleel, Zakaria Al-QudahJ, Christos A. Papachristou, Khaldoon Mhaidat, Francis G. Wolff, "Fast binary/decimal adder/subtractor with a novel correction-free BCD addition", Electronics, Circuits and Systems (ICECS), 2011.

[10] Deshpande Akshay, Sanidhya Mohan Sharma, Lochan Anil Vyas and K. Sivasankaran. Design of Low Power and Area Efficient 4-bit Arithmetic and Logic Unit using Nanoscale FinFET. Indian Journal of Science and Technology, Vol 8(S2), 250-256, Jan 2015. doi:10.17485/ijst/2015/v8iS2/70759.

[11] Sneh Lata Murotiya, Anu Gupta. Design of CNTFET-based 2-bit ternary ALU for nanoelectronics. International Journal of Electronics. Volume 101, 2014, Pages 1244-1257. http://dx.doi.org/10.1080/00207217.2013.828191.

[12] Garima Rawat, Khyati Rathore, Siddharth Goyal. Design and analysis of ALU: Vedic mathematics approach. International Conference on Computing, Communication & Automation (ICCCA), 2015. doi: 10.1109/CCAA.2015.7148593.

[13] Simge Öztunç, Ali Mutlu, Necdet Bildik, Computing Hypercrossed Complex Pairings in Digital Images, Abstract and Applied Analysis, Volume 2013 (2013), Article ID 675373, http://dx.doi.org/10.1155/2013/675373.