

The Mechanism of The Arithmetic Logic Unit

Yuguo Liu¹, Chenyang Zhang^{2,*}, Haoyi Zhang³

¹ Ulink Shanghai, Shanghai, 200000, China

² The Affiliated International School of Shenzhen University, Shenzhen, 518000, China

³ Shanghai DTD Academy, Shanghai, 200000, China

* Corresponding Author Email: zhanghy@student.dtd-edu.cn

Abstract. The Arithmetic Logic Unit is widely used in electrical components nowadays such as the CPU in computers. The traditional research is based on the theory of a combination of multiple logic units, the results are not performed intuitional. This paper provides an introduction to the Arithmetic Logic Unit and its mechanisms, which consist of multiple logic gates to perform binary operations and send commands to the computer. The article discusses binary operations using logic gates, starting from the simplest one-bit half-adder to the more complex four-bit ALU, encompassing functions like addition, subtraction, division, and multiplication. The simulation module is used to justify the mechanism of the ALU. This research provides a new method in the process of developing the ALU, may largely increases the productivity. The approach may be used in the research and development of computer performance, providing visual, instant feedback and reducing the cost.

Keywords: Binary; Logic Gate; Instruction Set; ALU.

1. Introduction

The Arithmetic Logic Unit (ALU) is a critical component of modern computers responsible for performing both arithmetic and logical operations. It constitutes a fundamental building block of computer processors, handling essential mathematical operations such as addition, subtraction, multiplication, and division, as well as logical operations. The performance of the Arithmetic Logic Unit directly impacts the overall speed and efficiency of the computer's operations.[1]

The concept of the ALU was notably introduced by John von Neumann in his report titled "First Draft of a Report on the EDVAC." [2] Von Neumann recognized the necessity of an ALU within a computer system to execute basic mathematical calculations. [3] He envisioned computers equipped with specialized components dedicated to performing these essential operations efficiently.

A significant milestone in the evolution of ALUs occurred during the 1960s with the advent of integrated circuit technology. Integrated circuits facilitated the integration of multiple electronic components onto a single chip, leading to more compact and efficient ALUs. This technological advancement enhanced the performance of ALU. Meanwhile, it reduced manufacturing costs substantially. Moreover, it contributes to the widespread adoption and advancement of computers.

The 1960s witnessed a momentous milestone in the evolution of ALUs, marked by the advent of integrated circuit technology. The integration of multiple electronic components onto a single chip via integrated circuits resulted in more compact and efficient ALUs. This remarkable advancement not only bolstered the ALU's performance but also significantly reduced manufacturing costs, catalyzing the widespread adoption and advancement of computers.

The journey of the ALU towards modernity has been characterized by relentless improvement since the 1960s. The development of processors propelled the ALU's capabilities beyond basic integer arithmetic, gradually encompassing floating-point arithmetic, vector arithmetic, and other advanced functions. [4]

This article fulfills its purpose of offering a detailed and accessible resource for beginners seeking to understand ALU's internal structure and design. By leveraging established theoretical knowledge and a practical approach, newcomers are aimed to be equipped with a comprehensive understanding of ALU functionality. This essay mainly introduces the basic calculating system of ALU. Also, involve some simple calculation by using logic gates and the figures of the calculating process would

be shown. The insights gained from this article can serve as a stepping stone for beginners to embark on further explorations in computer architecture, empowering them to contribute to the exciting world of modern computing.

2. Method and Process

2.1. Binary Number System

2.1.1 Binary number system

The Binary Number System, invented by Gottfried Leibniz, is a base-2 numbering system consisting of only two digits: 0 and 1. This system serves as the foundation for all binary code, which is used to represent digital data in various computing devices, including computer processors. In the binary system, each digit's position represents a power of 2, allowing for efficient representation and manipulation of information within digital systems. Due to its simplicity and compatibility with electronic devices, the binary number system has become fundamental to modern computer technology and plays a vital role in processing instructions and data in everyday computing tasks. [5]

2.1.2 Methods to transform numbers from decimalism to binary.

To translate a number from decimal to binary, a straightforward method is employed. Take the example of the decimal number 18. 18 can be represented as the sum of powers of 2, which is $16 + 2$. In binary, 16 is represented as 10000, and 2 is represented as 00010. Combining these, 10010 is the result of the binary representation of 18. For negative numbers, a two's complement approach is used. Taking -17 as an example, a positive counterpart should be firstly presented, which is 17, as 010001. Then, all the bits are inverted, changing 0 to 1 and 1 to 0, resulting in 101110. Finally, 1 is added to this inverted binary representation, which gives us the binary representation of -17 as 101111.[6]

2.2. Adding, subtracting, multiplying and dividing

In the Arithmetic Logic Unit, there are logic calculations and four fundamental arithmetic calculations. They are adding, subtracting, multiplying, and dividing. In this section, methods of calculation consisting of a series of logic gates as well as more sophisticated electron components are provided.

2.2.1. Adding

i). 1 bit - half adder

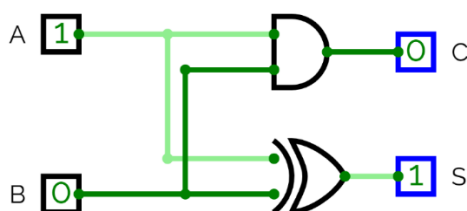


Fig. 1 Simulation module of one-bit half-adder

Table 1. The truth table of half-adder

A	B	Cin	S
0	0	0	0
0	1	0	1
1	1	0	1
1	0	1	0

This is a one-bit half-adder. It is called a one-bit half adder as the situation of adding the carry bit from behind is not involved. According to the truth table, the number of inputs '0' means the absence

of the following component, otherwise, '1' is the presence of the following one. Therefore, the equation of the S (self) as $S=A'B+AB'=A \text{ XOR } B$ (011&101), C (carry bit) as $C=A*B=A \text{ And } B$, can be concluded. Therefore, the equation can be summed as $S=A'B+AB'=A \text{ XOR } B$ (011&101), C (carry bit) as $C=A*B=A \text{ And } B$ (111 only). Therefore, two logic gates which are XOR Gate and AND Gate perform the process of adding.

ii). one-bit-full adder

In the one-bit full adder, the situation of adding the carry bit from behind is involved. So, there are two one-bit half adders are included in this particular full adder.

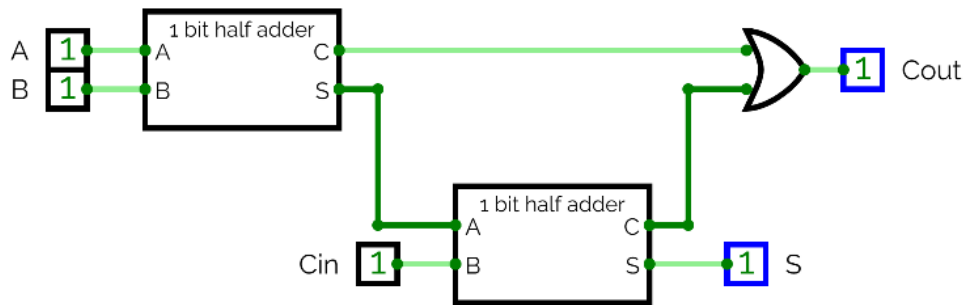


Fig. 2 Simulation module of one-bit full adder

Table 2. The truth table of full adder

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

In the simulation model shown above, A and B is the object that is added and C-in is the carry bit from behind. First, a one-bit half adder is used to add A1 and B1, and C1 and S1 are produced. S1 is the new A (marked as A2) and the C-in is the new B (marked as B2). Another half-adder is used to add A2 and B2 together. The final S is output directly, there are two carry bits to be exported, but another half adder is not used another half adder to add because there are only three possible situations (0,1/1,0/0,0) no carry bit generates. So Or Gate is used to connect these two Cs. Finally, the result is produced (S and C-out)

iii). Four-bit full adder

A split is a component (quite like a branch) splitting a four-bit number into four 1-bit numbers one by one, labeling them with 0, 1, 2, and 3, respectively. After that, four 1-bit half adders are used starting from the right side of each of the numbers (A0 and B0). As these two figures are on the right side of each number, there is no carry bit behind each of these, a one-bit half-adder for the addition of A0 and B0 is used. The one-bit full adder can be used here. In this situation, the value of C-in is constantly '0'. However, using a half adder may increase the efficiency of the whole adder, as well as minimize the wastage of the component. [7]

But in the rest of the process, one-bit full adders are needed to perform the addition of the carry bits from the right side of the figure. [8]

After adding A0 and B0, S0 (the first number of the answer counting from the right) and C-out1 (carry bit 1) are produced. Then the one-bit full adder is used to input three values which are A1, B1, and C-out1. After that, the cycle runs for another two times. But in the last cycle, the value that C-out

produces is S4 (the digit of the answer that lies on the left side). Therefore, the answer is (S4 S3 S2 S1 S0).

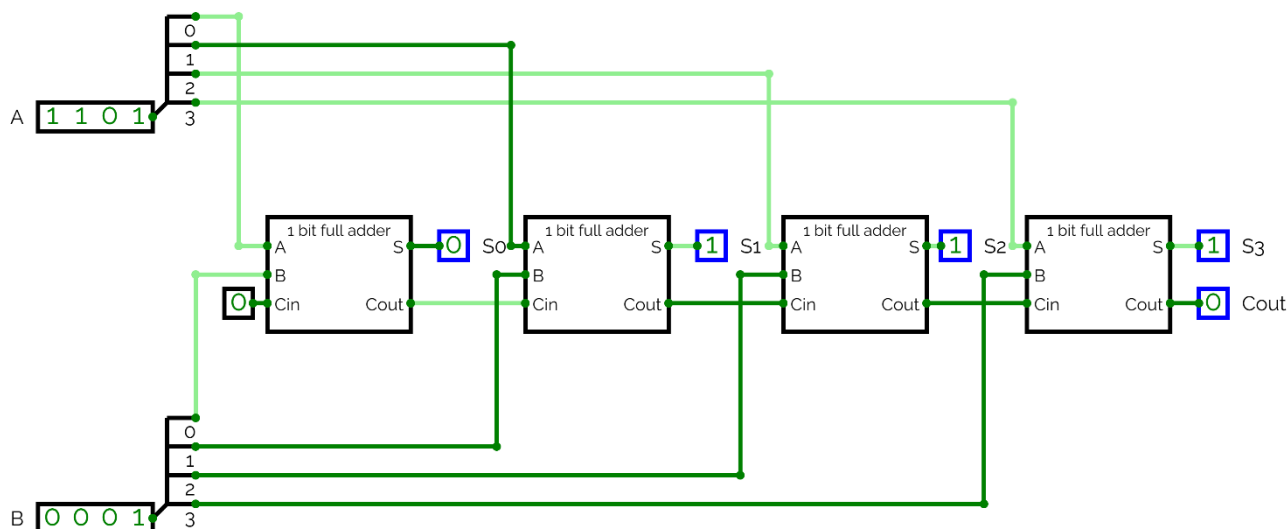


Fig. 3 Simulation module of the four-bit full adder (using splits)

2.2.2 Subtracting

i). one-bit half subtractor

The one-bit half subtractor, similar to the one-bit half adder, ignores the calculations behind the digits.

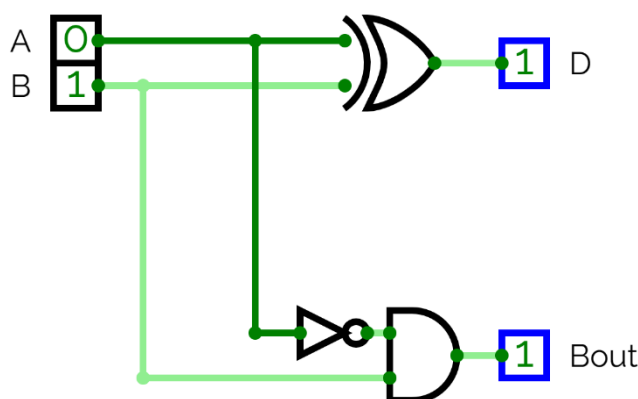


Fig. 4 Simulation module of one-bit half subtractor

Table 3. The truth table of half subtractor

A	B	Bout	D
0	0	0	0
0	1	1	1
1	1	0	0
1	0	1	0

According to the truth table provided in Table 3. The B-out (the situation when 0-1, 1 is borrowed from the digit in front of the current one) and D (the difference between the two figures) could be expressed as $B\text{-out} = A'B + AB' = A \text{ Xor } B$, $D = A'B = A' \text{ And } B$ (using a Not Gate and an And Gate). So, the series of logic gates should be assembled like Fig. 4.

ii). One-bit full subtractor

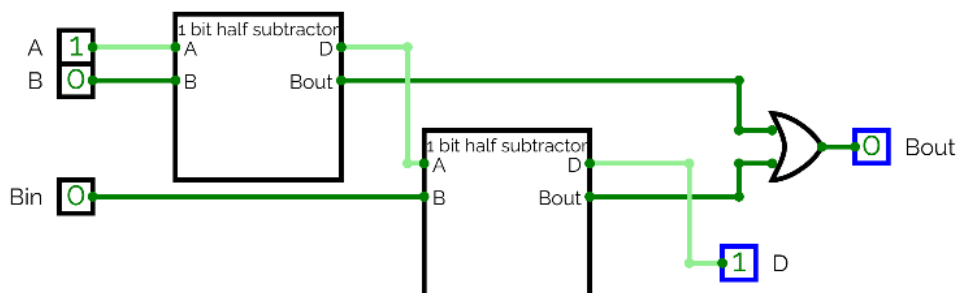


Fig. 5 Simulation module of one-bit subtractor

Table 4. The truth table of full subtractor

A	B	Bin	Bout	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The one-bit full subtractor consists of two one-bit half subtractors. After the first process of a one-bit half subtractor, the situation of the existence of borrowing from the digit behind it is considered. As shown above, after the first subtraction of A and B (marked as A₀ and B₀), D₀ and (B-out)₀ are produced, and then D₀ and B-out play the role of A₁ and B₁ respectively. The D₁ is the final result output is the final result. There are two borrowing bits to be exported, but another half adder is not used to add because there are only three possible situations (0,1/1,0/0,0) no carry bit is generated. An Or Gate is used to connect these two (B-out)s. Finally, the result (D and B-out) is generated.

iii). four-bit full subtractor

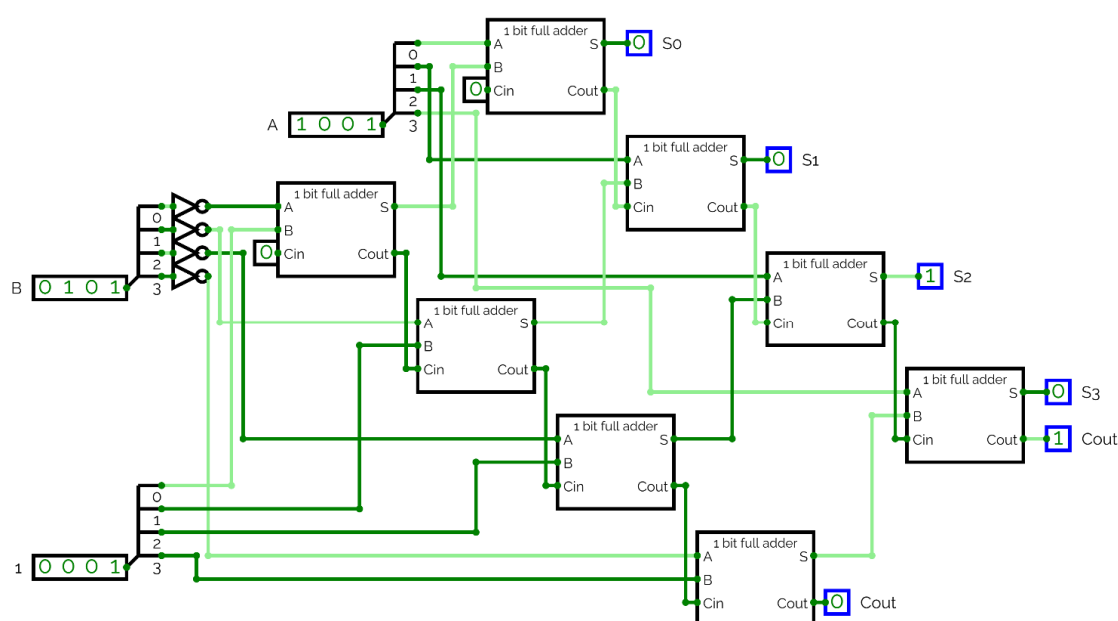


Fig. 6 Simulation module of four-bit full subtractor

This is the method of subtracting by adding the number's opposite number. [9] Generally speaking, there are two steps to transform the number to its opposite state. The two's complement approach is

utilized to transform. The first digit of the B in this subtractor must be '0' which means that B must be positive. This is the limitation of this method. B in this subtractor can only range from 0 to 7. Then the digit '0' is shifted to '1', and '1' is shifted to '0'. After that, the transformed number is added by 1. Then another four 1-bit full adders are used to add A and the new four digits gained before. Finally, the answer (S3 S2 S1 S0) is produced. {The two (C-out)s are excluded from the calculation, they do not have meaning in this method of subtractor}

2.2.3 Multiplying

i). multiplier using logic gates

Table 5. Four-bit vertical multiplication

				B3	B2	B1	B0
			x	A3	A2	A1	A0
				A0B3	A0B2	A0B1	A0B0
		+	A1B3	A1B2	A1B1	A1B0	
	+	A2B3	A2B2	A2B1	A2B0		
+	A3B3	A3B2	A3B1	A3B0			
+	(C-in)6	(C-in)5	(C-in)4	(C-in)3	(C-in)2	(C-in)1	
P7	P6	P5	P4	P3	P2	P1	P0

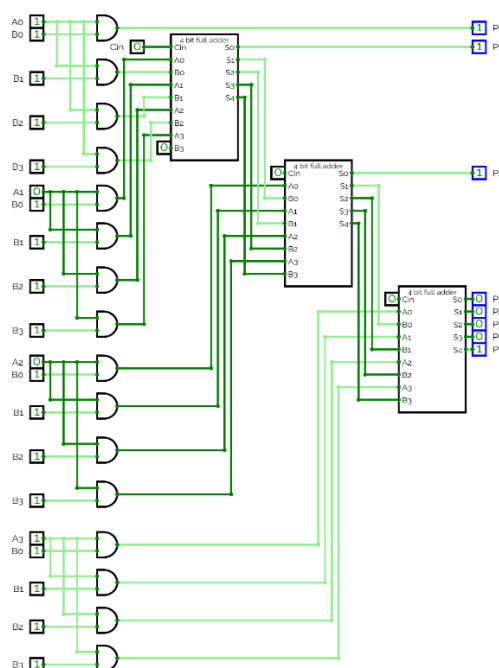


Fig. 7 Simulation module of four-bit multiplier

The four-bit multiplier operates with greater sophistication compared to basic calculators, as illustrated in Table 5. The multiplication process can be likened to a shifting adder. [10] Initially, P0 (the first digit of the product from the right) is obtained directly as the result of $A0 \cdot B0$, without considering any carry bit, as it represents the initial digit calculation. Subsequently, the addition of the first row and the second row takes place by using a four-bit full adder. In the first row, the last digit is ignored but adding a '0' in front of A0B3. The outcomes are as follows: $S0 = A0B1 + A1B0 + C\text{-in}$ (this becomes the value of P1), $S1 = A0B2 + A1B1 + C\text{-in}$, $S2 = A0B3 + A1B2 + C\text{-in}$, $S3 = 0 + A1B3 + C\text{-in}$, and $S4 = C\text{-in}$. The same process is repeated between the four-digit number (0 S4 S3 S2) and another four-bit number (A2B3 A2B2 A2B1 A2B0), with the value of S1 already exported as P2. Finally, one last cycle is performed before obtaining the final product, which is represented as (P7 P6 P5 P4 P3 P2 P1 P0).

And Gates are used to multiply. Because there are four times of multiplication in this component, four And Gates should be used in each cycle. And the first result is exported as the digit of the product, one by one. After three complete cycles, the result, as shown in Fig. 7, which is P0 to P7, ranges from right to left.

ii). sequential logic multiplier

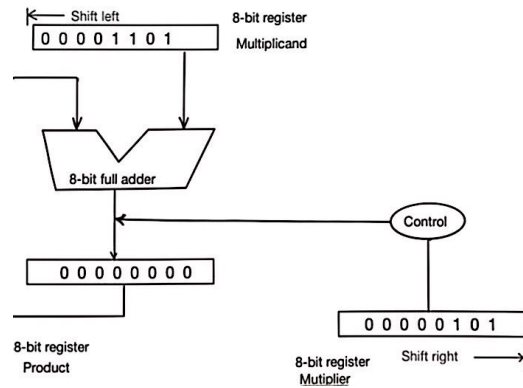


Fig. 8 Model of the sequential logic multiplier

The form of a multiplication circuit using loads of logic gates is not used in a real computer because it contains too many elements. A shift addition circuit which uses the sequential logic is used instead. As shown in Fig. 8, there are three 8-bit registers, each representing the Multiplicand, Multiplier, and Product, respectively. There's an 8-bit full adder linking the multiplicand and the product together, and this process is determined by the control. First, the two 8-bit numbers enter the Multiplicand and Multiplier, with eight zeroes in the Product. Then the control is used to detect whether the first digit (counting from the right side) of the Multiplier is '0' or '1'. If the digit is '1', the 8-bit number from the Multiplicand is copied to the Product. If the digit is '0', nothing happens. After the first addition, the digits of the Multiplier are shifted about one unit to the right (delete the first digit and add a '0' in front of the number) and one unit of the Multiplicand is shifted to the left (delete the eighth bit counting from the right and add a '0' on the right). Then the control is used again to perform the same function. [11] In the situation where the digit detected on the multiplier is '1', the new 8-bit number of the Multiplicand and the value of the product are added together. After about 8 cycles, the final output is the product.

2.2.4 Dividing

The divider containing a control is used to demonstrate the logic of division.

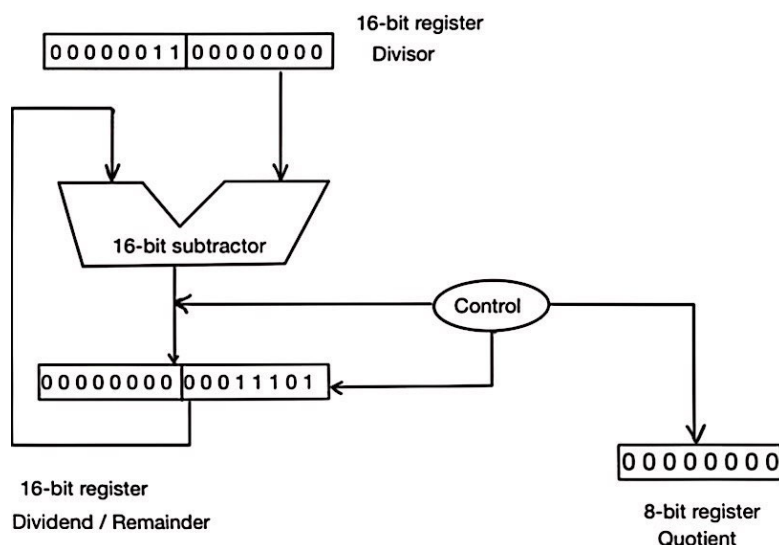


Fig. 9 Model of the sequential logic divider

In this sequential logic divider (used for an eight-bit number divided by an eight-bit number). We use three registers to store the values: two 16-bit registers for Divisor and Dividend (or Remainder) and an 8-bit register for Quotient. Divider has a control to determine the instruction of subtraction. First, the eight-digit dividend enters the right section of the register with other digits '0' and the eight-digit divisor is placed into the left part of the register, with other digits '0'. Because the divisor shifts to the right by one unit after each complete cycle (delete the first digit, which is '0', and add a '0' in front of the whole number). The control is used to compare the value of the divisor and the dividend. If the divisor is smaller than the dividend, then there is a '1' in the first digit of the present quotient. After this cycle, the digits of the quotient shift to the left (delete the eighth digit counting from the right and add a '0' at the right). [11]The same process is repeated until there's a '1' appearing at the first digit of the Divisor. At this time, the value of the Dividend is the Remainder. Finally, the quotient and the remainder are generated. s method is correct. [12]

3. Experiment and analysis

3.1. 4-bit full adder

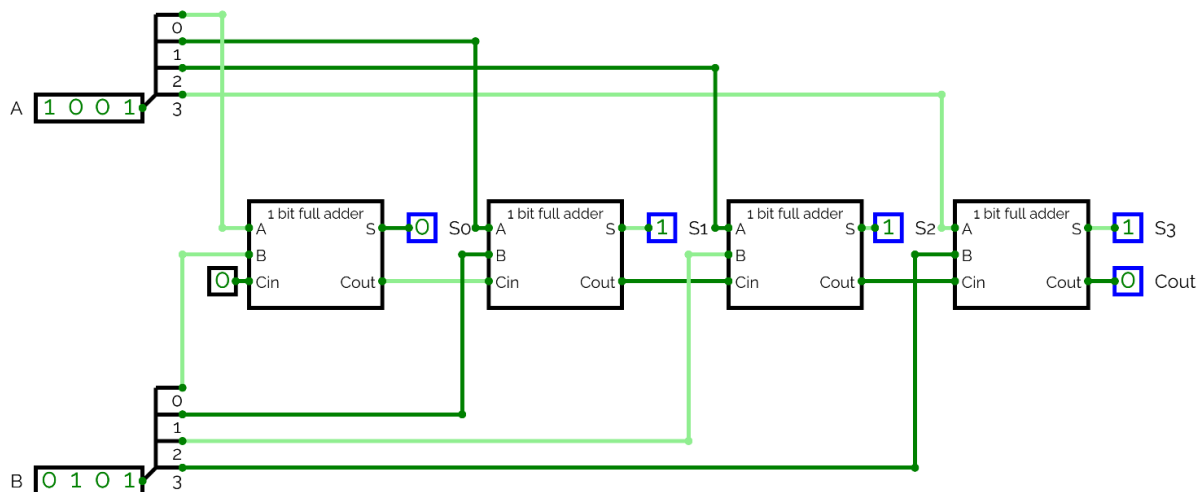


Fig. 10 Simulation module of four-bit full adder

The example of $1001 + 0101$ is provided. 1001 and 0101 are typed into the module. As shown in Fig.10, two splits separate a full 4-bit binary number into 4 single 1-bit binary numbers. In the first 1-bit half adder, 'A0' and 'B0' is added. Two outputs are produced 'S0' and 'C-out'. 'S0' is the answer of 'A0' + 'B0', which is $1 + 1 = 0$. Therefore, '1' is output from 'C-out' (carry-out bit). C-out' from the first 1-bit half adder is inputted into the 1-bit full adder's 'C-in' and repeat this step (input 'C-out' to the next 1-bit full adder's 'C-in') with the other two 1-bit full adders. 'A1' and 'B1' enter the first 1-bit full adder to output 'S1', then repeat this step with the other two 1-bit full adders using 'A2', 'B2', 'A3', and 'B3' to get 'S2' and 'S3'. The last 1-bit 4 adder is different from others, as the output of 'C-out' is considered as 'S4'. The answer of $1001 + 0101$ is equal to 1110. A check can be done by changing the binary number into decimal using two's complement. 1001 is the number 9 in decimal, and 0101 is 5 in decimal, so $9 + 5$ is equal to 14, and 1110 is also equal to 14.

3.2. 4-bit full subtractor (with 1-bit full subtractor)

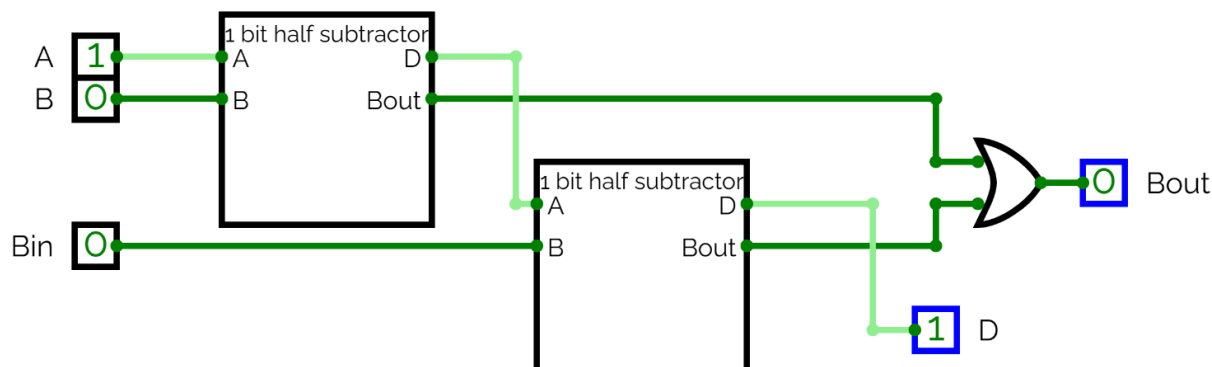


Fig. 11 Simulation module of four-bit full subtractor

The example of 1101-1010 is demonstrated. 1101 is considered to be A, and the digits in 1101 as 'A3', 'A2', 'A1', and 'A0'. Similarly, 1010 is B, and the digits in B are represented in the same way as A. Two splits are used to separate 1101 and 1010 into single 1-bit binary numbers. Firstly, 'A0' and 'B0' enter the first 1-bit half subtractor. Then the subtractor outputs 'D0' and 'B-out', which is the difference of 'A0' - 'B0' = 1 - 0 = 1. Since 1 - 0 doesn't require borrowing, 'B-out' = 0. 'B-out' is the input of 'B-in' in the next 1-bit full subtractor. Then 'A1' and 'B1' enter the 1-bit full subtractor, which gives 'A1' - 'B1' = 0 - 1 = 1. For 0 - 1, because 0 is less than 1, a digit '2' is borrowed from 'A2', performing the process of subtraction 2 - 1 = 1, so 'D1' = 1, and 'B-out' = 1. This step is repeated to gain output 'D2' and 'D3', so 0011 is the answer of 1101 - 1010. Transforming the binary number to decimal, a check can be done. When calculated, 1101 - 1010 = 0011. Binary number '1101' = '13', '1010' = '10', and '0011' = '3'. As 13 - 10 = 3, and calculation is correct.

3.3. 4-bit full multiplier

Take 01 as an example. Let's set the first digit of 1101 as 'A0', which is '1', the second as 'A1', the third as 'A2', and the last as 'A3', which is the far left digit in 1101, and set 1001 as 'B0', 'B1', 'B2', and 'B3' with the same way as 1001. As Fig.12 shows, '1' multiplied with '1', '0', '0', and '1' the same process is taken on figures '0', '1', and '1' respectively. The way of operating this multiplier has been illustrated carefully in the content above(II/C/3a)

Table 6. Calculation of 1101x1001

				1	1	0	1
			x	1	0	0	1
				1	1	0	1
		+	0	0	0	0	
	+	0	0	0	0		
+	1	1	0	1			
	1	1	1	0	1	0	1

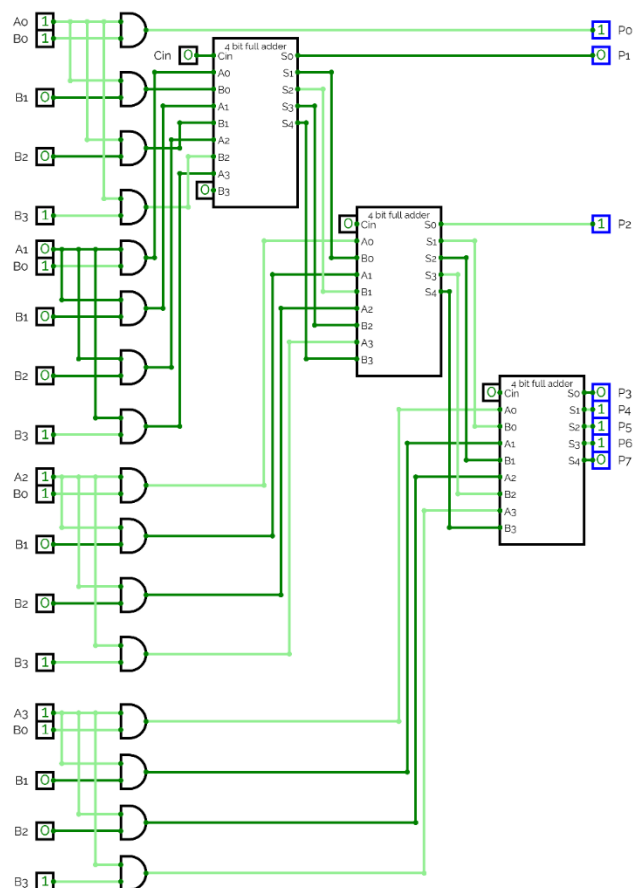


Fig. 12 Simulation module of four-bit multiplier

3.4. 4-bit full divider

4-bit divider stimulation borrowed form: CircuitVerse – 4-bit nonrestoring divider with a remainder

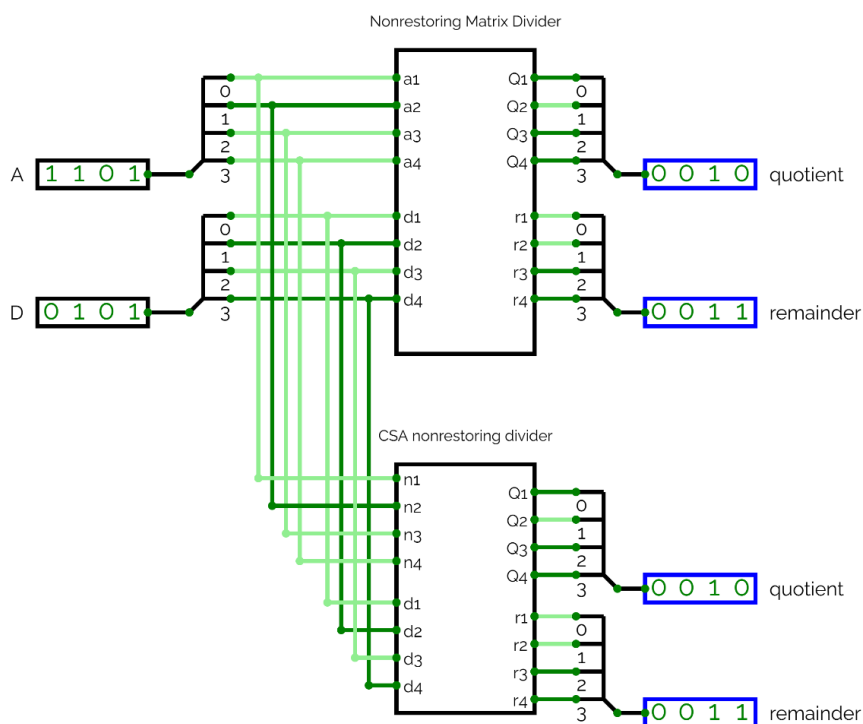


Fig. 13 Simulation module of four-bit divider

Table 7. Calculation of $1101 \div 0101$

					0	0	1	0
0	1	0	1	/	1	1	0	1
				-	1	0	1	
					0	0	1	1
				-	0	0	0	0
					0	0	1	1

Check:

Dividend = $1101 = 13$

Divisor = $0101 = 5$

Quotient = $0010 = 2$

Remainder = $0101 = 5$

$1101/0101 = 0010 \dots 0011 \quad 13/5 = 2 \dots 3$

3.5. divided completely

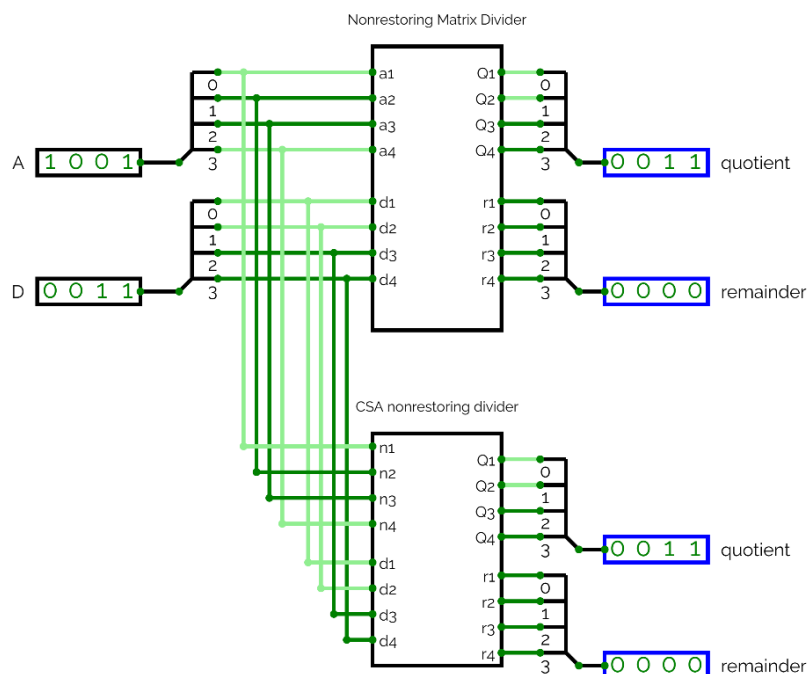


Fig. 14 Simulation module of four-bit divider

Table 8. Calculation of $1001 \div 0011$

					0	0	1	1
0	0	1	1	/	1	0	0	1
				-		1	1	
					0	0	1	1
				-	0	0	1	1
								0

Check

Dividend = $1001 = 9$

Divisor = $0011 = 3$

Quotient = $0011 = 3$

Remainder = 0

$1001/0011 = 0011 \quad 9/3 = 3$

4. Conclusion

In conclusion, ALU plays a vital role in performing mathematical calculations and logical calculations as a fundamental component in modern computer structure. This article based on the use of the simulation module, researches the basic logic of the four calculators and proves the feasibility of the sequential logic multiplier and divider. The result of the research implicates: 1, All the mathematical calculations used in the computer can be expressed using a series of logic gates. 2, Sequential logic calculators have much higher automaticity than the normal combination of logic gates and the efficiency of the ALU is much higher using the control. The result of the research is of great value in improving the performance of the ALU, especially in fields such as the integration of parallel processing and optimized circuitry design, revealing the utilization potentiality of the sequential logic calculator. However, the research, which is based on the simulation module, lacks the consideration of the practical application conditions. Therefore, in the future, we will base on the practical appliances, analyze the actual operating mechanism of ALU under different practical conditions, and promote the development of the ALU.

Author Contribution

All the authors contributed equally and their names were listed in alphabetical order

References

- [1] Burks A W, Goldstine H H and von Neumann. Preliminary discussion of the logical design of an electronic computing instrument, Institute for Advanced Study, Princeton, NJ, USA, 1946: 399-413.
- [2] Von Neumann J. First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing, 1993, 15(4): 27-75.
- [3] T. Rhyne and N.R. Strader. A signed bit-sequential multiplier. IEEE Transaction on Computers, 1989, 38(9): 1328-1330.
- [4] Bennett C. IBM J. Res. Develop, 1973, 17(6): 525-532.
- [5] Jamil T. The complex binary number system. IEEE potentials, 2002, 20(5): 39-41.
- [6] Bocker R P, Clayton S R, Bromley K. Electrooptical matrix multiplication using the twos complement arithmetic for improved accuracy. Applied Optics, 1983, 22(13): 2019-2021.
- [7] Cuccaro S A, Draper T G, Kutin S A, et al. A new quantum ripple-carry addition circuit. arXiv preprint quant-ph/0410184, 2004.
- [8] Thomsen M K, Axelsen H B. Parallelization of reversible ripple-carry adders. Parallel Processing Letters, 2009, 19(02): 205-222.
- [9] Vieri C J. Reversible computer engineering and architecture. Massachusetts Institute of Technology, 1999, pp. 96-100.
- [10] Booth A D. A signed binary multiplication technique. The Quarterly Journal of Mechanics and Applied Mathematics, 1951, 4(2): 236-240.
- [11] Tocher K D. Techniques of multiplication and division for automatic binary computers. The Quarterly Journal of Mechanics and Applied Mathematics, 1958, 11(3): 364-384.
- [12] Freiman C V. Statistical analysis of certain binary division algorithms. Proceedings of the IRE, 1961, 49(1): 91-103.