Creative Education, 2021, 12, 2954-2958
https://www.scirp.org/journal/ce
ISSN Online: 2151-4771
ISSN Print: 2151-4755

Explanation of Several Problems with Complementary Code

Wenbing Wu, Yupeng Wu, Jinquan Xiong

Nanchang Normal University, Nanchang, China
Email: wwbysq@fjnu.edu.cn

How to cite this paper: Wu, W. B., Wu, Y. P., & Xiong, J. Q. (2021). Explanation of Several Problems with Complementary Code. *Creative Education, 12,* 2954-2958.
https://doi.org/10.4236/ce.2021.1212221

Received: November 18, 2021
Accepted: December 28, 2021
Published: December 31, 2021

Abstract

In order to improve the teaching effect of the course of Principles of Computer Composition, this article has conducted a more in-depth discussion on the teaching of complements. Complementary code is a difficult point in computer science. In the process of teaching, students feel that there are various difficulties in truly understanding this concept. Through the explanation of various forms of complements, including integers, decimals, and deformed complements, this article strives to make this concept concrete and visualized, so as to provide some help to the teaching of this concept in the teaching process. The teaching effect shows that the methods discussed in the article are helpful to improve the teaching effect.

Keywords

Complement Code, Decimal Complementary Code, Deformation Complementary Code, Teaching

1. Introduction

Computer complement has always been a difficult point in computer science teaching. The difficulty lies in the fact that complements require more mathematical knowledge (Kurabayashi et al., 2018; Sarkar, Majhi, & Liu, 2019; Holubnychyi & Konakhovych, 2018; Alexeev, 2021), and computer students are relatively lacking in this area. In the course of teaching this content, I have tried many methods in order to improve the teaching effect. This article writes down these tried methods, hoping to promote the teaching of complement codes.

2. The Definition of Complement

$$[x]_{com} = \begin{cases} 0, x, & \text{when } 2^n > x \geq 0 \\ 2^{n+1} + x, & \text{when } 0 > x \geq -2^n \pmod 2 \end{cases} \quad (1)$$

where $[x]_{com}$ represents the Complementary code of *x, and x is* the true value and *n* is the number of digits in the integer As can be seen from the above definition, for negative numbers,

$$[x]_{com} = 2^{n+1} + x \tag{2}$$

Here *x* represents a negative number, for example, $x = -1011$, *n* represents the number of digits of the negative number of *x* that does not consider the value of the sign bit, here it is equal to 4. Equation (2) can become

$$[x]_{com} + |x| = 2^{n+1} \tag{3}$$

This means that the complement of a negative number plus the absolute value of the negative number will return to zero, such as

$$\begin{array}{r} 1011 \\ +\ 0101 \\ \hline 10000 \end{array}$$

Here 1011 is the complement of −5, and the highest bit is the sign bit. According to the definition in Equation (3), n here is equal to 3, so the complement of −5 1011 plus the absolute value of −5, that is, 0101, the answer is 2 to the 4th power, and the result at this time has exceeded the number of digits (including the sign bit 4), so the highest bit is Abandon, that is, the phenomenon of returning to zero has appeared.

3. Representation Range of Complement

A byte is 8 bits, if the original code is used to represent a positive integer (including 0), it can express 0 - 255, that is, $2^8 = 256$, a total of 256 states, various permutations and combinations from all 0 to all 1. If you want to represent a negative number, the sign bit needs to occupy one bit (the highest bit, 1 represents a negative number, 0 represents a positive number), so the maximum absolute value range is 0 - 127, that is, $2^7 = 128$, a total of 128 positive and negative Status, if no special processing is used, at this time 0 occupies 2 codes, 10000000 is negative 0, and 00000000 is positive 0, the data representation range is −127 to −0 and +0 to 127, so that overall a byte is only 255 A state, because 0 has positive 0 and negative 0 points, which does not conform to the mathematical meaning and wastes a code. Therefore, people think of using negative 0, that is, when encountering a negative number, using the complement code to represent this problem can be solved, and when encountering a positive number or 0, the original code representation is still retained. Therefore, this negative 0 is naturally used after being processed by the complement algorithm to represent −128.

In order to prove that the negative 0 of 10000000 represents the correctness of −128's complement, consider

$$\begin{array}{r} 10000000 \\ +01111111 \\ \hline 11111111 \end{array}$$

Because 01111111 is the original code of +127, the above calculation is equivalent to calculating the result of (−128 + 127). Obviously, the result should be −1, and 11111111 is the complement of −1. Others can be deduced by analogy, so 10000000. It means −128 complement is correct. The above calculation also shows that the important function of the complement is to turn the subtraction (127 - 128) operation into the addition (−128 + 127) operation. In summary: Complement code: 8-bit complement code can represent the range of −128 - 127.

Because 01111111 is the original code of +127, the above calculation is equivalent to calculating the result of (−128 + 127). Obviously, the result should be -1, and 11111111 is the complement of −1. Others can be deduced by analogy, so 10000000. It means −128 complement is correct.

The above calculation also shows that the important function of the complement is to turn the subtraction (127 - 128) operation into the addition (−128 + 127) operation. In summary: Complement code: 8-bit complement code can represent the range of −128 - 127.

4. Decimal Complementary Code

For decimals, its complement is defined as

$$[x]_{com} = \begin{cases} x, & \text{when } 1 > x \geq 0 \\ 2 + x, & \text{when } 0 > x \geq -1 \end{cases} \tag{4}$$

For example:

$$x = +0.1110, \quad [x]_{com} = 0.1110$$

$$x = -0.1100, \quad [x]_{com} == 2 + (-0.1100) = 10.0000 - 0.1100 = 1.0100$$

That is, in the definition of the decimal's complement, the one in the integer part is used to represent the sign bit.

5. Deformation Complementary Code

$$[x]_{defom} = \begin{cases} x, & \text{when } 1 \geq x \geq 0 \\ 100 + x, & \text{when } 0 \geq x > -1 \end{cases} \tag{5}$$

The definition of Equation (5) is also for decimals, where 100 is binary, which represents the number 4. This definition means that each decimal has two sign bits.

Deformed complement is represented by "00" for positive and "11" for negative, which is also called modulo 4's complement. When adding and subtracting operations with deformed complement, when the sign bit of the operation result appears "01" or "10", it means that an overflow has occurred. The most significant bit (the first sign bit) of the anamorphic complement always represents the correct sign. For example, "00" and "01" represent positive numbers and positive overflow (overflow) respectively, and "11" and "10" represent negative numbers, Negative overflow (underflow).

$$\left[\frac{1}{2}X\right]_{defom} = 00.01011,\ \left[\frac{1}{2}Y\right]_{defom} = 00.01001$$

$$\left[\frac{1}{2}(X+Y)\right]_{defom} = \left[\frac{1}{2}X\right]_{defom} + \left[\frac{1}{2}Y\right]_{defom} = 00.01011 + 00.01001 = 00.10100$$

Here is the addition of two positive decimals. There is no overflow in this case. This is because the two decimals are less than 0.5, so their sum is less than 1. If it is 00.11 + 00.10, the result is 01.01, in this case An overflow occurs because the sum of the two decimals has exceeded 1. Another example

$$\left[\frac{1}{2}X\right]_{defom} = 11.10101,\ \left[\frac{1}{2}Y\right]_{defom} = 11.10111$$

$$\left[\frac{1}{2}(X+Y)\right]_{defom} = \left[\frac{1}{2}X\right]_{defom} + \left[\frac{1}{2}Y\right]_{defom} = 11.10101 + 11.10111 = 11.01100$$

This is the addition of the two's complement of two negative decimals, and the result does not overflow. But if you look closely, you can see that during the addition process, a carry from the most significant bit of the decimal to the sign bit occurs, so why is there no overflow? This is because the numerical parts of the two decimals' complements 11.10101 and 11.10111 are both greater than 0.5, which means that the absolute value of two negative decimals is less than 0.5, and the two decimals whose absolute value is less than 0.5 are added together. The absolute value cannot be greater than 1, and if the addition is used, the absolute value of the addition will exceed 1, so a carry will occur, but because it is the addition of two negative numbers, that is

$$\begin{array}{r} 11.11 = -0.75 \\ +\ 11.10 = -0.5 \\ \hline 111.01 = -1.25 \end{array}$$

Keep the decimal part = −0.25

The highest 1 is removed, leaving 11.01, that is, the carry generated by the addition of the two complements of the above two negative decimals, which is just offset by the addition of their sign bits.

6. Conclusion

Through the above explanation of the various situations of complements, it is hoped that in teaching, teachers and students who find it difficult to learn this aspect can provide a little help.

Funding

This paper is supported by Research Foundation of the Nanchang Normal University for Doctors (NSBSJJ2018014). Key R&D Project of Jiangxi Provincial Department of Science and Technology (20192BBEL50040, 20192BBHL80002).

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

Alexeev, B. (2021). Nonlocal Physics in the Wave Function Terminology. *Journal of Applied Mathematics and Physics, 9,* 2889-2908. https://doi.org/10.4236/jamp.2021.911183

Holubnychyi, A. H., & Konakhovych, G. F. (2018). Multiplicative Complementary Binary Signal-Code Constructions. *Radioelectronics and Communications Systems, 61,* 431-443. https://doi.org/10.3103/S0735272718100011

Kurabayashi, M., Tsuchimochi, H., Komuro, I. et al. (2018). Molecular Cloning and Characterization of Human Cardiac Alpha- and Beta-Form Myosin Heavy Chain Complementary DNA Clones. Regulation of Expression during Development and Pressure Overload in Human Atrium. *Journal of Clinical Investigation, 82,* 524-531. https://doi.org/10.1172/JCI113627

Sarkar, P., Majhi, S., & Liu, Z. (2019). Optimal Z-Complementary Code Set from Generalized Reed-Muller Codes. *IEEE Transactions on Communications, 67,* 1783-1796.