A Graphical Teaching Tool for Understanding Two's Complement

CARLOS L. LÜCK
Electrical Engineering Department
University of Southern Maine
Gorham, ME 04038-1088
luck@usm.maine.edu

Abstract

As part of the Electrical Engineering program, students are typically introduced to Two's Complement algebra and representation, a method to include negative numbers in the binary representation of integers that is widely used in microprocessors and related digital systems. The traditional, procedural method to generate and evaluate the binary pattern of negative numbers is often perceived to be non-intuitive and time-consuming when students perform it by hand.

A graphical method for the representation of binary numbers using Two's Complement algebra was presented to an introductory class on microprocessors with great success. The method is based on the principle of operation of the mechanical odometer and is particularly suited for such tasks as the quick and reliable generation/evaluation of "small" negative numbers (-1, -2, etc.), detecting typical microprocessor conditions such as carry and overflow, and performing the common arithmetic operations of adding, subtracting, incrementing and decrementing. Two important educational goals were achieved with this method. First, the graphical representation allowed the students to grasp the principles rather than to focus on the process. Second, the ability to quickly and reliably generate small negative numbers proved to be a valuable skill for assembly programming in the laboratory, since small address offsets are commonplace in branching instructions.

1 Introduction

Not unlike other subjects and concepts in engineering, Two's Complement algebra and representation are introduced in earlier courses and then fully explored later in the program. That in itself is not a problem. On the contrary, revisiting an earlier topic helps a student to approach it with a fresh perspective and solidify the concept. The problem we have experienced is in the way Two's Complement is normally presented.

Two's Complement is one among a variety of methods designed to represent negative integers in digital systems. As such, it is typically introduced in the first digital course of the Electrical Engineering program, along with logic gates, boolean algebra, flip-flops and combinatorial logic, to name a few. Over the years, students have rated this course as one of the heaviest in the program, not because of the complexity of the material, but rather due to the volume of content, the presence of a challenging laboratory component and it being so early in the program.

Consequently, relatively little time is spent in binary algebra involving negative numbers, and when it does the focus is often on the process rather than the concept, simply because the students are in the context of manipulating individual bits or short strings of bits. Students learn quickly that in order to "two's complement" a number, they must negate (or flip) each individual bit of the binary string and then add one to the result. But they often fail to distinguish between the algebra, the process and the representation, or worse, carry misconceptions about them. The process itself brings them no closer to understanding the concept. This problem is observed later in the program, when the students take a course on microprocessors, which makes extensive use of Two's Complement algebra in its assembly programming.

Having taught microprocessors for several years, I recognize the problem and this paper reports on my efforts to address it. The paper discusses an alternative method of presenting the concept and indicates how students benefit from this approach.

2 The Concept

Manipulation of binary numbers follows the same rules used in the manipulation of decimal numbers: to increment a number with the right-most digit having the highest symbol (in the decimal case 9), we return it to zero, index the digit immediately to its left and propagate the process to the left as needed. The only difference is that binary numbers have 2 symbols instead of 10. When dealing with larger numbers (thus long and tedious strings of 1's and 0's), we often cluster 4 consecutive bits to create the hexadecimal representation ($2^4 = 16$, *i.e.* 4 bits can represent 16 distinct, sequential symbols). In hexadecimal, we use 0-9 for the first 10 symbols and A-F for the remaining 6 symbols in the representation.

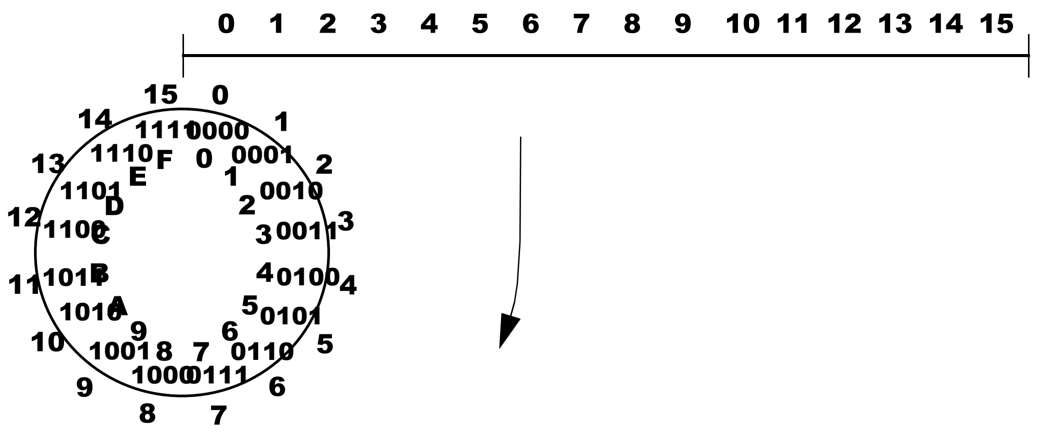

FIGURE 1. Unsigned Representation

But unlike our abstract view of numbers as points on an infinite straight line, the finite nature of the physical implementation of numbers in a digital system makes it more appropriate to conceive the topology of the representation as circular rather than linear. This characteristic is embedded in the design of digital systems through a feature referred to as “end-around,” *i.e.* when the value of a number has reached its maximum range, an increment causes it to return to the minimum value. It is a mechanism analogous to mechanical counters such as vehicle odometers (those that have not yet been replaced by digital counters), in which numbers are marked on the surface of a wheel and a full revolution (or a full sequence of indexed revolutions in case of multiple digits) brings it back to zero.

In fact, the mechanical analogy is the very basis for the graphical representation discussed in this paper. A simple way to conceive this notion is to take a straight line with a scale ranging from zero to the maximum number in the representation and wrap it around the wheel. For instance, in a 4-bit system, the 16 values are marked on a wheel equally spaced and in sequence, such that the highest value (15 in decimal or F in hexadecimal) becomes adjacent to zero. Incrementing and adding (decrementing and subtracting) are viewed as navigating along the wheel in the clockwise (counterclockwise) direction.

Figure 1 illustrates the concept above. Inside the wheel the binary pattern of each number is depicted, along with its hexadecimal equivalent. Outside the wheel the decimal number is shown, product of wrapping the scaled line around the wheel, starting at zero. Some of the most recent textbooks in digital logic present the concept of the number wheel, recognizing its power as an explanation tool [9,10]. This representation is suitable for all-positive (known as unsigned) numbers.

Negative numbers are represented by attributing negative values to half of the available binary patterns. Several methods are available:

2.1 Sign and Magnitude

The left-most bit represents the sign (0 for positive, 1 for negative) and the remaining bits represent the magnitude as in the unsigned case. Figure 2 illustrates the concept, with the decimal values (the meaning) on the outside and the binary/ hexadecimal values (the representation) on the inside.

This method is easy to be conceived and recognized by the students, but it has two disadvantages. First, the turning wheel concept is disrupted since negative numbers grow in the opposite directions as the positive numbers, making it harder to design an arithmetic unit to manipulate those numbers in additions and subtractions. Second, there is the inconvenience of double representation for the number zero. Graphically, the conversion from positive to negative (and vice-versa) can be obtained as the diametrically opposed pattern on the wheel.

2.2 One’s Complement

The representation of a negative number is obtained by complementing bit-by-bit the binary pattern of its corresponding positive number (*i.e.* the magnitude).

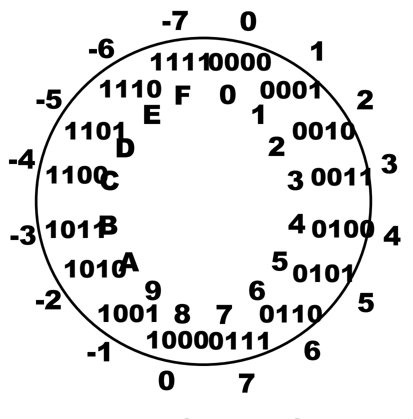

FIGURE 2. Sign and Magnitude Representation

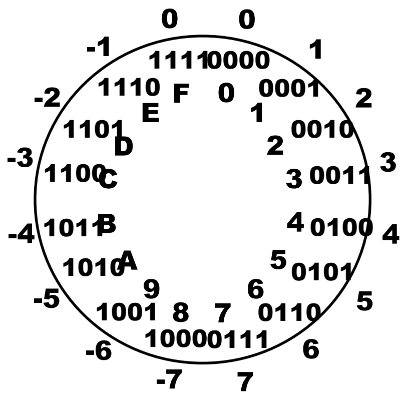

FIGURE 3. One's Complement Representation

As with all other methods of representation, the inverse process is identical, credit to the binary nature of the representation. In other words, the positive number is obtained by "one's complementing" the negative pattern. The leftmost bit functions as a flag to differentiate between positive and negative numbers just like in the Sign and Magnitude method.

As a result of the procedure, however, the order of the negative numbers is reversed, making it more suitable for arithmetic operations in the sense of navigating the wheel as previously discussed. Yet it still contains the inconvenience of double representation for the number zero. Whenever there is a crossing between the positive and the negative sides, the wheel must be turned one extra position to compensate for the double zero. Graphically, the conversion from positive to negative (and vice-versa) can be obtained as a mirror image about the vertical diameter line (Figure 3).

2.3 N/2 Bias

As a transition to motivate the final solution, it is useful to analyze the following method. The idea stems from the desire to establish a natural sequence of numbers from the minimum value to the maximum value along the wheel, such that additions and subtractions can be performed irrespective of the sign of the number. Since there are $N=2^n$ numbers on the wheel and only one zero is to be present, the remaining patterns for positive and negative numbers total an odd quantity. We choose to have the positive side one value less than the negative side to compensate, meaning that the range of numbers will be from -N/2 to +N/2-1. Furthermore, the minimum number will match the minimum binary representation of all-zeros and the maximum number will match the maximum representation of all-ones.

The principle can be conceived graphically as taking a straight line with a scale spanning the representation from -N/2 to +N/2-1 and wrapping it around the wheel, analogous to the unsigned concept in Figure 1. As a result, the numerical value in this representation is obtained by subtracting N/2 from the unsigned binary pattern, thus the term N/2 Bias. Since the bias is carried equally in every value, arithmetic operations can be performed exactly as in the unsigned case, requiring no extra hardware beyond the simple arithmetic engine used to manipulate unsigned numbers. Negative numbers make up the lower half of the representation (identified by the leading bit equal to zero) and positive numbers take the upper half (leading bit equal to one), as seen in Figure 4.

However, this convenience comes at a price. Unlike all previous methods, positive numbers are represented differently between the signed and unsigned cases. The final method aims at fixing this problem.

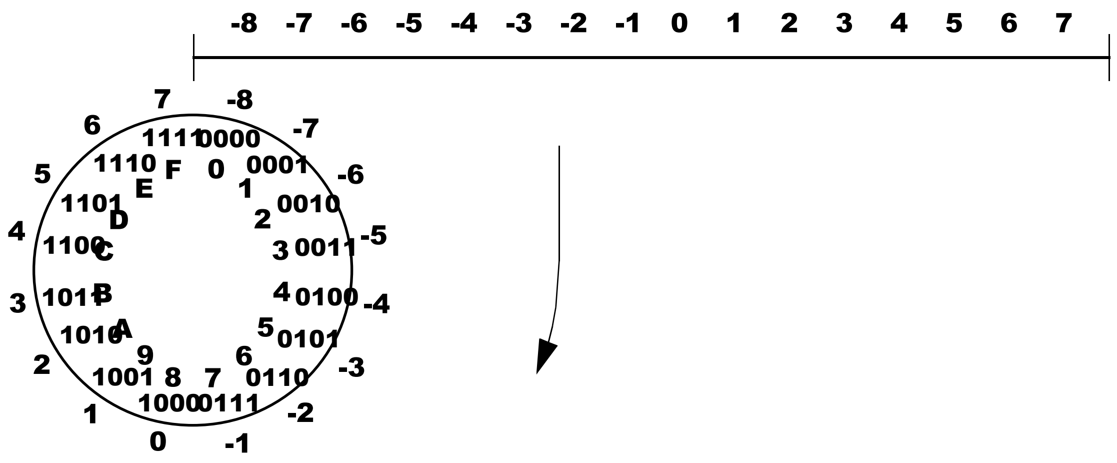

FIGURE 4. N/2 Bias Representation

2.4 Two's Complement

Textbooks present this method either procedurally (such as a bias of one from the one's complement method to eliminate the double zero) or as the principle that gives the representation its name: $x + \bar{x} = 2^n$, such that $\bar{x}$ is the complement of x with respect to 2^n and $\bar{x}$ defines the representation of the negative of x. Neither approach captures easily the essence of what makes the method work and therefore create confusion, and sometimes misconceptions, in the minds of students. They fail to recognize the distinction between the process of obtaining the negative of a number by "two's complementing" it and the representation that establishes the numerical value assigned to a binary pattern.

The graphical approach we are using with great success to capture the essence of the method is to take the same desirable sequence from -N/2 to +N/2-1, as in the N/2 Bias method, and wrap it around the wheel in such a way that the representation of positive numbers will coincide between the unsigned and the signed cases, as shown in Figure 5.

The approach explores the "end-around" feature to allow a seamless transition between the positive and the negative side of the wheel, using the standard concept of navigating the wheel clockwise for addition and counterclockwise for subtraction. From the hardware point of view, further simplification is obtained by treating subtraction as the special case of addition with the negative of the second term, readily available by "two's complementing" it. All those features combined make two's complement the method of choice in current design of microprocessors.

3 The Experience

The presentation of Two's Complement in the context of the wheel has resulted in a better understanding of the concept, reflecting an improved performance in exam questions involving the manipulation of signed numbers. But in the microprocessor course, the concept helped in other ways too.

In an introductory course on microprocessors, it is desirable to base it upon a commercial product that is simple and inexpensive for the sake of experimentation by the students and at the same time contains the fundamental principles and operating features of modern, larger scale microprocessors used in computer systems. The usual choice is an 8-bit microprocessor, which is produced by several manufacturers with similar characteristics. The one adopted in our program is the Motorola 6800 series.

Since numbers are 8 bits wide and $2^8 = 256$, unsigned numbers are represented in the 0/255 range and signed numbers are represented in the -128/+127 range, following Two's complement representation. 8-bit patterns can be captured using 2 hexadecimal symbols, ranging from 00 to FF. The wheel concept helps conceive the representation and assists in interpreting the principles of carry and overflow. Carry occurs as a result of an arithmetic operation that causes the number to cross the FF-00 mark, or the top of the wheel. It represents the crossing of the boundary in the unsigned representation. Overflow, on the other hand, occurs as a result of an arithmetic operation that causes the number cross the 7F--80 mark, or the bottom of the wheel. It represents the crossing of the boundary in the signed representation. Figure 6 illustrates the notion.

But perhaps the most significant advantage of the wheel concept is in the quick and reliable generation and evaluation of negative numbers of small magnitude. It is obtained by mentally decrementing the number from zero until the desired value is reached, which means navigating the wheel in the counterclockwise direction. For instance, -1 is represented by FF and -4 is obtained by counting backwards FF, FE, FD, FC. Thus, the binary representation of -4 is FC.

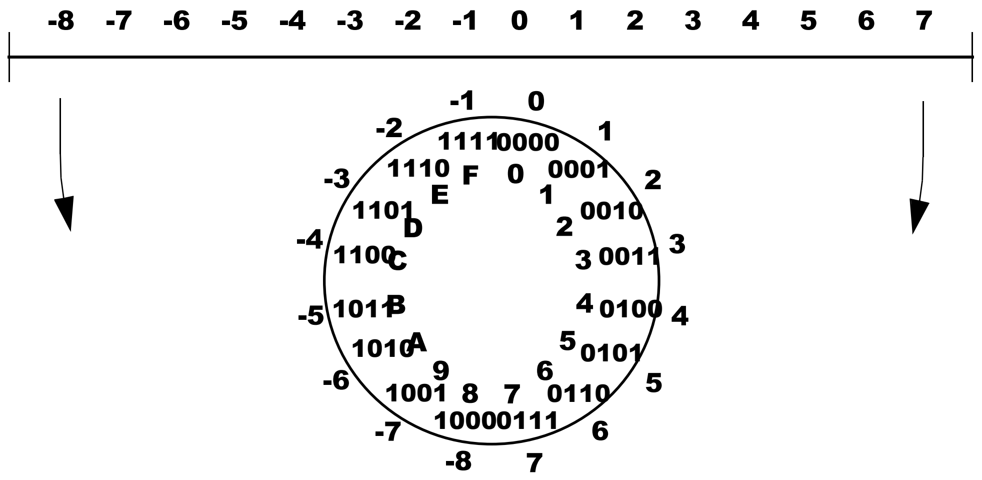

FIGURE 5. Two's Complement Representation

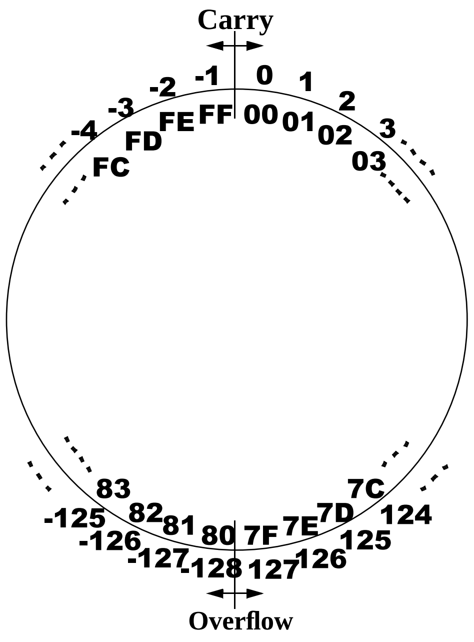

FIGURE 6. 8-bit Two's Complement Representation with Carry and Overflow

The representation FC is naturally obtained with the traditional method of "two's complementing" 04 as well, but students often take longer and lack confidence in the correctness of the outcome. Speed and confidence are greater with the help of the wheel.

But why would such a skill be desirable to begin with? As it turns out, in assembly language programming, notably in the 6800 architecture, the target of branching instructions is defined as an offset from the present position. In other words, the address of the next instruction to be executed in the case of a branch is obtained by adding the given offset to the current location (program counter). Typically, branching instructions are local and associated with small loops, such as adding vector elements or looping with countdown to produce a time delay. Consequently, the offset is often a positive number (for branching forward) or a negative number (for branching backwards) with a small magnitude. In the laboratory, students use the wheel concept to quickly assemble and disassemble programs that contain branching instructions, by mentally applying the procedure described in the previous paragraph.

4 Conclusion

In a world in which principles are becoming increasingly transparent to the students with access to sophisticated tools and software, it is important that concepts be presented with an ever increasing focus on the meaning rather than the process for learning to be more effective.

Assembly language programming as a skill is becoming nearly obsolete, credit to powerful compilers and smart assemblers that hide the details of the code generation. On the other hand, logic gates are not just the link from the hardware of electronic circuits to the abstraction of boolean algebra, but are also heavily involved in the design of digital systems, connecting the various subsystems in a computer, for example. Therefore, the two extremes of gate level and processor level are used in conjunction during system design. Two's Complement is caught in the middle and there is a risk that it will be relegated to a level of detail hidden by the programming software tools. However, since it is at the core of how microprocessors manipulate binary numbers, a good understanding of microprocessors can not be attained without mastering this concept.

In this paper, a method is presented to explain the mechanism behind Two's Complement algebra, without having to focus on the procedure for generating negative numbers. Over time, the details of the process (and there are many which have not been addressed in this paper for the sake of brevity) will fade, as the student moves on to different areas, but hopefully the concept of the wheel will help to keep Two's Complement as one of the key features bridging the gap between the transistor logic and the user level view of a computer.

References

1. Zvi Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 2nd Edition, 1978.
2. Fredrick Hill and Gerald Peterson, *Introduction to Switching Theory & Logical Design*, John Wiley, 3rd Edition, 1981.
3. Joseph Cavanagh, *Digital Computer Arithmetic*, McGraw-Hill, 1984.
4. Frederick F. Driscoll, *Introduction to 6800/68000 Microprocessors*, Delmar, 1987.
5. John P. Hayes, *Computer Architecture and Organization*, 2nd Edition, 1988.
6. Barry B. Brey, *Microprocessors and Peripherals*, Merrill, 2nd Edition, 1988.
7. Jack Quinn, *The 6800 Microprocessor*, Merrill, 1990.
8. John Uffenbeck, Microcomputers and Microprocessors, Prentice Hall, 2nd Edition, 1991.
9. Randy H. Katz, *Contemporary Logic Design*, Benjamin/ Cummings, 1994.
10. John F. Wakerly, *Digital Design: Principles and Practices*, Prentice-Hall, 2nd Edition, 1994.