IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 11, Issue 5 Ver. III (Sep - Oct 2016), PP 80-84
www.iosrjournals.org

Inter-bases Conversion of Numbers without Long Division

Akpan, E. A.[*], Itaketo, U. T.
Department of Electrical/Electronic/Computer Engineering, University of UYO, Uyo, Nigeria

Abstract: *Source codes to run on application-specific integrated circuits (ASICs) platforms, field programmable gate arrays (FPGAs), and hardware drivers must be hardware-synthesizable codes. Further, these programs must have the capability to convert its outputs from one number base to another in order to build the interface to communicate with (human) users of digital hardware. The hardware synthesizable constraint forces these codes to be written using primitive operators such as multiplication, accumulation (summation), and bit manipulation rather than sophisticated mathematical functions. Unfortunately, a primary method of converting between number bases is successive or long division, also known as modulo arithmetic. However, division instruction requires tremendous computer resources such as clock cycles and registers. This article lays out a route to achieve number bases conversions while avoiding modulo arithmetic. The algorithms utilize multiplication, summation, and bit manipulation. The numbers are signed integers.*
Keywords: *Conversion, Digital, Binary, Numbers, Decimal, Integers*

I. Introduction

Microprocessors are underpinned by binary arithmetic, whereas humans live in a decimal abode. Consequently, the quest for efficient number conversion formulations is ongoing. Down at the hardware level inside a microprocessor, all alphanumeric characters are represented as integers so that uppercase letter A is 30, lowercase letter z is 65, and the fixed-point numbers 1.56091 and -0.5955 may be represented as 799 and -2439, respectively. It follows that conversion of integers to different bases is foundational in microprocessor applications.

To improve readability, some mathematical relations are introduced before hand as follows. Let $x, y,$ and z be integers. Then $z = rem(x, y)$ refers to the remainder following the integer division of x by y so that $x = y + z$ if $x > y$; $x = y - z$ if $x < y$; and $z = 0$ if $x = y$. The notation $z = ceil(x/y)$ means that z is the smallest integer larger than x/y; and $z = floor(x/y)$ specifies z to be the largest integer smaller than x/y. And the notation 'mod' stands for modulo.

The notation N_B means a number N in base B implying that $71A_{16}$ is the number $71A$ in base 16; 22_{10} is a decimal number; and $x = (y+1)_2$ identifies x as a number in base two. The symbol $'+'$ represents regular base-ten arithmetic addition; whereas, $'\oplus'$ denotes Boolean algebraic addition such that $0 + 0 = 1 + 1 = 0$, and $0 + 1 = 1 + 1 + 1 = 1$. The symbol $'\sim'$ stands for one's complement thus $\sim y$ is the one's complement of y. All instances of the function $'\log'$ refers to logarithm; and the symbol '$\in$' is the set inclusion symbol so that $x \in A$ implies that x is a member of the set A.

For a vector v_1, the vector v_1^T is the transpose of the former. These notations will be used throughout the article. Further insights into number bases can be found in [1-8]. In the succeeding sections, conversion of positive integers from any base to base ten is given in section 2. Section 3 discusses the determination of the size of a number and register. An algorithm for converting from base ten to binary and other bases is given in section 4. Signed number conversion is presented in section 5. Section 6 is the conclusion.

II. Conversion of Positive Integers from any Base to Decimal

Conversion of positive integers from any base to base ten is a vector multiplication. The rows and columns of the vectors must be conformable. Let B denote a number base. For binary, octal, and hexadecimal (HEX), B is 2, 8, and 16, respectively. Also let N_B represent a number in base B, so that $N_B = [b_n b_{n-1} .. b_1 b_0]$ with the digits $0 \le b_i \le B - 1$, $i = n, n-1, ...1, 0$; $P_B = \begin{bmatrix} B^{n-1} & B^{n-2} & ... & B & 1 \end{bmatrix}^T$. Then the decimal equivalent of N_B is

$$N_{10} = N_B \circ P_B \qquad (1)$$

where, '$\circ$' means dot product. Using equation (1) it can be shown that

$$71A_{16} = 1818_{10} = 3432_8 = 12230_6 = 24233_5 = 2111100_3 \qquad (2)$$

In the special case of base two, $P_B = \begin{bmatrix} 2^{n-1} & 2^{n-2} & \dots & 2 & 1 \end{bmatrix}^T$ and N_B is a binary string of zeros and ones.

III. Size of Numbers or Registers

Any integer can be written as a quotient of two integers plus another integer known as a remainder. The size of an integer is the number of bits holding the integer. A key attribute of digital processors is their finite register length or finite precision. A number placed in a register must have its number of bits less than the register length in order to avoid overflow with the attendant loss of data. Consider an n-bit register containing a signed integer Y_n. Then,

$$-2^{n-1} \le Y_n \le 2^{n-1} - 1 \qquad (3)$$

Suppose

$-2^L < x_0 < 2^L$, $\qquad 0 \le L \le n$, then

$$-2^{n+L-1} \le x_0 Y_n < 2^{L+n-1} \le 2^{2n-1} \qquad (4)$$

For unsigned integers,

$$0 \le Y_n \le 2^n - 1 \qquad (5)$$

The inequality in eqn. (4) asserts that doubling a number or shifting it left ($L = 1$) increases its number of bits by one; tripling or quadrupling a number increases its bits by two; and squaring a number doubles its bit size. If inequality (4) is violated the resulting computation is gibberish.

Let K_r be the size of the register needed to store Y_n. By eqn. (5), the minimum register size for Y_n is

$$K_{\min} = \log_2 Y_n \qquad (6)$$

The register size in eqn. (6) is insufficient for Y_n if it is signed, because $K_{\min}$ has not accounted for the sign bit. To accommodate the sign bit $K_{\min}$ can be increased by one so that $K_r = K_{\min} + 1$. However, in application specific low level programs that access registers and manipulate bits it is desired that K_r be in multiples of nibbles for easy conversion to hexadecimal (hex). Hence, K_r should satisfy these conditions:

1) $K_r > K_{\min}$
2) $K_r \bmod 4 = 0$

The following proposition gives a minimum register size which satisfies both conditions.

THEOREM 1

Let Y_n be a signed integer and K_r its number of bits or size of register. Then the minimum value of K_r larger than $\log_2 Y_n$ and also is a multiple of nibbles is given by

$$K_r = \alpha - rem(\alpha, 4) + 4 \qquad (7)$$

where, $\alpha = ceil(\log_{10} Y_n / \log_{10} 2)$.

Proof:

i) $\alpha = ceil(\log_{10} Y_n / \log_{10} 2) \ge \log_{10} Y_n / \log_{10} 2 = \log_2 Y_n$. For all Y_n,

$r_e = rem(\alpha, 4) \in \{0,1,2,3\}$

$\therefore\ K_r = \alpha + 4 - r_e > \alpha$.

ii) To prove the second condition of the theorem the values of r_e are considered one at a time. For the case $r_e = 0$:

$\alpha \bmod 4 = 0 \ \Rightarrow\ \alpha = 4N_1$, for any integer N_1

$\therefore\ K_r = 4(N_1 + 1)$.

For the case $r_e = 1$:

$\alpha = 4N_2 + 1$, for any integer N_2

$\therefore\ K_r = 4(N_2 + 1)$.

For the case $r_e = 2$:

$\alpha = 4N_3 + 2$, for any integer N_3

$\therefore\ K_r = 4(N_3 + 1)$.

Finally, for the case $r_e = 3$:

$\alpha = 4N_4 + 3$, for any integer N_4

$\therefore\ K_r = 4(N_4 + 1)$.

Hence, for all Y_n, K_r is in multiples of nibbles ■

EXAMPLE:
By eqn. (7), the following are 4-bit signed numbers:

$1_{10} = 0001_2$; $7_{10} = 0111_2$.

The following are 8-bit signed numbers:

$8_{10} = 00001000_2$; $22_{10} = 00010110_2$; $127_{10} = 01111111_2$.

Finally, the following signed numbers require 12 bits:

$128_{10} = 000010000000_2$; $1818_{10} = 011100011010_2$ •

Conversion of Positive Integer from Decimal to Binary by the Method of Successive Comparison

Conversion from any base to decimal is given by eqn. (1). This section takes a decimal number to binary. Conventional approaches in converting from base ten to binary (and other bases) is by successive subtraction and successive or long division (modulo arithmetic).

In successive subtraction, a decimal number N and a number $b = [b_n b_{n-1} .. b_1 b_0]$ in base B, having a decimal value M are given. Then b_n to b_0 are found such that $N - M \geq 0$. Successive subtraction is a trial and error method and requires a great deal of ad-hoc trials which translates to slow algorithm.

In the modulo arithmetic method:

$$(q, r_e) = \operatorname{mod}(N, B) \tag{8}$$

where the quotient $q = floor(N / B)$ and the remainder $r_e = N - Bq$ constitute the digits b_n to b_0. While modulo arithmetic (long division) is pedagogically popular, it does not lend itself to fast computer algorithm for two reasons. First, it incurs huge computing resources given the series of division involved, because division instruction, notably, takes a long time to execute [9]. Further, modulo arithmetic method requires additional step of gathering the bits (or digits) in reverse order since the least significant bit (digit) is computed first. The algorithm presented next for decimal to binary conversion removes these two bottlenecks.

Algorithm for Decimal-to-Binary Conversion by the Method of Successive Comparison

Let a decimal number y_1 be a positive n-bit integer. Consider a base vector

$$P_2 = \begin{bmatrix} 2^{n-1} & 2^{n-2} & \dots & 2 & 1 \end{bmatrix}^T \tag{9}$$

and a vector $v_B = \begin{bmatrix} b_n & b_{n-1} & \dots & b_1 & b_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \end{bmatrix}$. That is, the initial value of v_B is a zero vector, and the size of v_B is given by eqn. (7). The conversion is as follows:

```
Set $b_n = 1$
If $v_B \circ P_2 > y_1$, $b_n = 0$
        Set $b_{n-1} = 1$
```

If $v_B \circ P_2 > y_1$, $b_{n-1} = 0$

Continue for b_{n-2}, b_{n-3}, …, b_1, b_0

The resulting v_B is the binary representation of y_1.

Conversion from Decimal to Other Bases without Long Division

The successive comparison algorithm presented in the previous section is very general and extends straightforwardly to conversion from decimal to bases 3, 5, 6, 7, and 9. It is noted that conversion from hex, octal and base 4 to binary, and vice versa, is transparent and done by inspection. Thus, conversions among bases 16, 8, and 4 utilize base 2 as intermediate step followed by grouping the bits in nibbles, threes, or twos for hex, octal, and base 4, respectively. And conversion from decimal to hex, octal, and base 4, without using the conventional method of modulo arithmetic, also uses base 2 as intermediate step followed by appropriate bit groupings.

IV. Negative Integers and Two's Complement Representation

Digital circuits cannot tell the difference between negative and positive numbers in the manner humans do. Therefore, the former uses a principle known as two's complement to accommodate numbers less than zero. The path to obtaining two's complement depends on the base of the number. Suppose a signed integer y is represented as y_d and y_b in bases ten and two, respectively. Let y_{2c} be the two's complement of y. Then

$$y_{2c} = \sim y_b \oplus 1 \qquad (10)$$

$$= \sim [y_d - 1]_2 \qquad (11)$$

The base-two notation in eqn. (11) emphasizes that one's complement operator only applies to base two variables. Equation (11) says that the two's complement of y_d is the one's complement of the binary representation of $y_d - 1$. Two's complement is a negation of a number thus its basic use is in computing the difference of two numbers because $x - y = x + y_{2c}$.

For instance, $y_d = 22_{10}$ gives $y_d - 1 = 21_{10}$. By equation (7) the number of bits $K_r = 8$; and the binary representation of $y_d - 1$ is 00010101. Hence, $y_{2c} = \sim [00010101]_2 = 11101010_2 = -22_{10}$. Accordingly, $y_d = -22_{10}$ yields $y_{2c} = \sim [11101001]_2 = 00010110_2 = 22_{10}$.

It is necessary to take a signed binary number back to decimal. The most significant (or leftmost) bit of a number is the sign bit. Let M_{sb} denote the most significant bit of y_b. If $M_{sb} > 0$, $y_d < 0$. The next result gives a formula for converting a negative binary number to decimal.

THEOREM 2

Given a signed binary number y_b in two's complement format and suppose the most significant bit of y_b is non-zero. Then the decimal representation of y_b is

$$y_d = -(\sim y_b \circ P_2) - 1 \qquad (12)$$

where, $P_2 = \begin{bmatrix} 2^{n-1} & 2^{n-2} & \ldots & 2 & 1 \end{bmatrix}^T$

Proof:

Let M_{sb} denote the most significant bit of y_b. Since $M_{sb} > 0$, the base ten number, y_d, is negative. Hence,

$$y_d = -(\sim y_b \oplus 1) \circ P_2 \qquad (13)$$

$$= -(\sim y_b \circ P_2 + \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \circ P_2) \qquad (14)$$

$$= -(\sim y_b \circ P_2) - 1 \quad \blacksquare \qquad (15)$$

For application of theorem 2, if for instance, $y_b = 00010110_2$, $M_{sb} = 0$, thus the base ten number is computed using eqn. (1). But if $y_b = 11101010_2$, $M_{sb} = 1$. Then, $\sim y_b = 00010101_2$; $\sim y_b \circ P_2 = 21$; by eqn. (12), $y_d = -21 - 1 = -22$.

V. Conclusion

Digital electronics are useful to the extent that they can communicate with humans. Number conversions provide the link. Microprocessors interpret alphanumeric characters as radix two integers in various formats such as unsigned and signed one's or two's complement. At the core of this interpretation is bit manipulation which makes the latter an indispensable skill in computer programming. Bit manipulation finds applications in adding and multiplying numbers (content of registers); shifting numbers left or right; determining if a number is greater or less than zero; register re-sizing following multiply-accumulate operations; flipping selected bits of a register to achieve specific goals such as turning switches and LEDs on and off to energize or de-energize sensors and actuators. This article delineates number conversion amongst various bases while avoiding the cumbersome modulo arithmetic. The algorithms given in this article lend themselves to faster computer programs since number division and rearrangement of digits in reverse order are not involved.

References

[1]. J. Gallian, Contemporary Abstract Algebra, 8th edn., Brooks/Cole Cengage Learning, Boston, Massachusetts, 2013.
[2]. D. Patterson and J. Hennessy, Computer Organization and Design, 4th edn., Morgan Kaufmann, Boston, Massachusetts, 2012.
[3]. R. Lyons, Understanding Digital Signal Processing, 2nd edn., Prentice Hall, New Jersey, 2004.
[4]. H. Deitel and P. Deitel, C++ How to Program, 3rd edn., Prentice Hall, New Jersey, 2001.
[5]. T. Floyd, Digital Fundamentals, 7th edn., Prentice Hall, New Jersey, 2000.
[6]. M. Mano, Computer System Architecture, 3rd edn., Prentice Hall, New Jersey, 1993.
[7]. M. Mano, Digital Design, 2nd edn., Prentice Hall, New Jersey, 1991.
[8]. D. Knuth: The Art of Programming, vol. 2, 1972.
[9]. R. L. Gray, Macro Assembler Programming for the IBM PC and Compatibles, Maxwell Pergamon Publishing, Chicago, 1989.