Computer Architecture

Arithmetic

Number bases

Three number bases are of interest: Binary, Octal and Hexadecimal. We look briefly at conversions among them and between each of them and decimal.

Binary

Base-two, or binary, contains the digits 0 and 1.

To convert a decimal number to binary, divide repeatedly by 2 until a quotient of zero is produced, keeping track of the remainders and the order in which they occur. Then arrange the remainders in reverse order of production, so that the last remainder becomes the most significant binary digit (bit) and the first remainder becomes the least significant bit.

Example

Convert Decimal 79 to Binary:

79/2 = 39 Remainder 1 Least Significant Bit (LSB)

39/2 = 19 Remainder 1

19/2 = 9 Remainder 1

9/2 = 4 Remainder 1

4/2 = 2 Remainder 0

2/2 = 1 Remainder 0

1/2 = 0 Remainder 1 Most Significant Bit (MSB)

Thus 79D = 1001111B

To convert a binary number to decimal, note that every bit position has a weighting. If we number the bit positions from the right, labeling the rightmost (least significant) bit position as position zero, then the weighting of each position is 2 to the power of the bit position. The value of each bit position is the digit in that position multiplied by the corresponding weighting. The decimal value of the binary number is the sum of all the bit position values.

Example

Convert the Binary number 1001111B to Decimal:

1 * 2^0 = 1 * 1 = 1

1 * 2^1 = 1 * 2 = 2

1 * 2^2 = 1 * 4 = 4

1 * 2^3 = 1 * 8 = 8

0 * 2^4 = 0 * 16 = 0

0 * 2^5 = 0 * 32 = 0

1 * 2^6 = 1 * 64 = 64

Add 1 + 2 + 4 + 8 + 64 = 79

Thus 1001111B = 79D

Octal

Base-eight, or octal, contains the digits 0, 1, 2, 3, 4, 5, 6 and 7.

To convert a decimal number to octal, divide repeatedly by 8 until a quotient of zero is produced, keeping track of the remainders and the order in which they occur. Then arrange the remainders in reverse order of production, so that the last remainder becomes the most significant octal digit and the first remainder becomes the least significant octal digit.

Example

Convert Decimal 119 to Octal:

119/8 = 14 Remainder 7 Least Significant Digit

14/8 = 1 Remainder 6

1/8 = 0 Remainder 1 Most Significant Digit

Thus 119 D = 167Q

To convert an octal number to decimal, note that every octal digit position has a weighting. If we number the digit positions from the right, labeling the rightmost (least significant) digit position as position zero, then the weighting of each position is 8 to the power of the digit position. The value of each digit position is the digit in that position multiplied by the corresponding weighting. The decimal value of the octal number is the sum of all the octal digit position values.

Example

Convert the Octal number 167Q to Decimal:

7 * 8^0 = 7 * 1 = 7

6 * 8^1 = 6 * 8 = 48

1 * 8^2 = 1 * 64 = 64

Add 64 + 48 + 7 = 119

Thus 167Q = 119D

Hexadecimal

Base-sixteen, or hexadecimal, contains the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

To convert a decimal number to hexadecimal, divide repeatedly by sixteen until a quotient of zero is produced, keeping track of the remainders and the order in which they occur. Then arrange the remainders in reverse order of production, so that the last remainder becomes the most significant hexadecimal digit and the first remainder becomes the least significant bit.

Example

Convert Decimal 1015 to Hexadecimal:

1015/16 = 63 Remainder 7 Least Significant Digit

63/16 = 3 Remainder 15 = hex digit F

3/16 = 0 Remainder 3 Most Significant Digit

Thus 1015 D = 3F7H

To convert a hexadecimal number to decimal, note that every bit position has a weighting. If we number the hexadecimal digit positions from the right, labeling the rightmost (least significant) digit position as position zero, then the weighting of each position is sixteen (16D) to the power of the digit position. The value of each digit position is the digit in that position multiplied by the corresponding weighting. The decimal value of the hexadecimal number is the sum of all the digit position values.

Example

Convert the Hexadecimal number 3F7H to Decimal:

7 * 16^0 = 7 * 1 = 7

F * 16^1 = 15 * 16 = 240

3 * 16^2 = 3 * 256 = 768

Add 768 + 240 + 7 = 1015

Thus 3F7H = 1015D

Natural numbers

An 8-bit value has 2^8 = 256 different states corresponding to the bit patterns 00000000 to 11111111. Several interpretations of the bit patterns are possible. If they are viewed as labels, then 256 distinct labels exist. If the patterns are viewed as natural numbers, then there are 256 of these, ranging from 0 to 255. The binary number corresponding to decimal 256 is the 9-bit value 100000000. The ninth bit represents an overflow from the eighth bit position, and in x86 processors it sets the carry flag so that this overflow can be detected within a program. Note that the 8 register bits are all set to zero.

A 16-bit value has 2^16 = 65536 different states. If viewed as labels, then 65536 labels exist, ranging from 0000 0000 0000 0000 to 1111 1111 1111 1111. Therefore a 16-bit memory address can distinguish between 2^16 = 2^6 * 2^10 = 64K memory locations. Viewed as natural numbers, 16-bit values range from 0 to 65535. Any number greater than this represents an overflow that must be handled by appropriate program actions.

Integers

A pattern of digits in any number base can be viewed as an integer, having possibly positive, negative or zero value.

The inclusion of negative numbers in a system gives rise to some interesting possibilities. Usually, the same quantity of positive and negative numbers is desired, at least approximately. In the decimal system, we are accustomed to using the “+” and “-“ signs to distinguish between positive and negative quantities. This is known as a signed magnitude system, and we have been conditioned into ignoring the resulting two zero values, +0 and -0.

We have been trained from an early age to perform decimal subtractions by referring to a set of tables committed to memory. Multiple-digit subtractions are dealt with by borrowing a

ten from the next-most-significant digit place if necessary and possible. If the minuend is greater than the subtrahend, a positive difference is generated naturally. If the subtrahend is greater than the minuend, we have been taught to swap the subtrahend and minuend and reverse the sign of the result.

We can, however, also perform subtraction by combining a complement operation with an addition. The nine's complement of a decimal number is formed by subtracting each digit of the number from 9 and placing the result in the same digit position.

Thus, to form the nine's complement of 256, we do 999 - 256 = 743.

The ten's complement of a decimal number is its nine's complement plus 1.

Thus, the ten's complement of 256 is 743 + 1 = 744.

The sum of 256 and ten's complement (256) = 256 + 744 = 1]000. If we restrict our operations to three decimal digits, the 1 can be ignored and we have formed the additive inverse of a decimal number by calculating its ten's complement.

So, to subtract 73 from 85, we can add: 85 + 27 = 1]12. Note that, since we have performed our operations on two digits, the third digit that is formed must be ignored (for the moment). If we want four digits in the answer (i.e. the difference), both the minuend and the subtrahend must be represented to four digits from the beginning.

For a four-digit representation of the difference between 85 and 73, we proceed as follows: 85 - 73 = 0085 - 0073 = 0085 + {ten's complement(0073)} = 0085 + {9926 + 1} = 0085 + 9927 = 1]0012.

The difference 73 - 85 can be formed in the same way: 73 - 85 = 0073 - 0085 = 0073 + {9914 + 1} = 0073 + 9915 = 9988. So what is this? Well, you notice that the negatives of 0073 and 0085 are 9927 and 9915 respectively. So we can deduce that the leading digits of a negative number in ten's complement form will be 9 before the significant digits of the number. Therefore, 9988 is a negative number in ten's complement form. Its signed-magnitude representation is -{ten's complement(9988)} = -(0011 + 1) = -12.

You may note that the ten's complement operation is itself based on subtraction, so that performing subtraction by ten's complement addition in decimal numbers does not yield any advantage. However, the binary number system is different in that respect.

In the binary system, a fixed number of bits corresponds to a fixed number of possible states. If we choose to use some states to represent negative numbers, then there will be fewer

positive numbers in our system as a consequence.

For instance, with 8-bit numbers, the leftmost bit (MSB) can be used as a sign bit, with 0 as a plus sign and 1 as a minus sign. This leaves 7 bits for the magnitude. The range of numbers that can be represented will then be -127 .. -0 +0 .. +127, a completely symmetric representation with 256 states and two zeroes. Then, adding -5 to +5 gives 10000101 + 000000101 = 100001010 which is -10 in decimal. Another example is adding -127 to +127, which either gives overflow or the number +126, depending on how it is handled.

Another approach is to use the complement of a number as its negative form, so that

+0 = 00000000; -0 = 11111111

+1 = 00000001; -1 = 11111110

+126 = 01111110; -126 = 10000001

+127 = 01111111; -127 = 10000000

Again, this is a symmetric system. The 256 states include two values for zero. This system has the advantage that adding +n to -n results in a zero value. The system is known as the one's complement system. Note that the negative numbers all have a 1 in the MSB position.

In an attempt to eliminate the duplicate zero, consider an extension of the one's complement representation. In this third way of treating negative numbers, we modify the one's complement approach by adding 1 to the one's complement of a number. The resulting system is known as the two's complement representation. Using this approach, the positive integers are the same as in the one's complement system

+0 = 00000000; -0 = 11111111 + 1 = 1 0000 0000, the same as + 0 if we ignore the carry.

+1 = 00000001; -1 = 11111110 + 1 = 1111 1111

+126 = 01111110; -126 = 10000001 + 1 = 1000 0010

+127 = 01111111; -127 = 10000000 + 1 = 1000 0001

The one negative code that does not appear above is 1000 0000.

-127 = 1000 0001 - 1 = -128 = 1000 0000.

So the two's complement system represents numbers -128 .. -1 0 1 .. 127.

Therefore it is an asymmetrical system, having 1 zero in its 256 states.

In general, n bits can represent 2^n states.

If natural numbers are represented, the range is 0 .. 2^n - 1

For signed integers, the range is -2^(n-1) .. +2^(n-1) - 1

Converting between negative and positive numbers in the two's complement system involves the same operation in both directions.

Example

Express -5 in its 8-bit two's complement form

+5 = 00000101; -5 = 11111010 + 1 = 11111011

What is the magnitude of the 8-bit two's complement number 11111011?

one's complement(11111011) = 00000100. Add 1 –> 00000101

Answer: the magnitude of the number is 5.

Subtraction can be implemented by two's complement addition. Instead of a hardware subtractor, the number being subtracted can be complemented, incremented and added to the other number.

Example

Perform the calculation 119D -79D by two's complement addition

119D = 167Q from the decimal-to-octal conversion example above.

167Q = 001 110 111 = 01110111

79D = 0100 1111B from the decimal-to-binary conversion example above.

-79D = 1011 0000 + 1 = 1011 0001

119D + (-79D) = 0111 0111 + 1011 0001 = [1] 0010 1000 = 2^5 + 2^3 = 40

This is the correct result, but note that there is a carry out of bit 7, indicating (incorrectly) a borrow from the next more significant byte position.

Perform the calculation 79D -119D by two's complement addition.

-119D = 1's complement(0111 0111) + 1 = 1000 1000 + 1 = 1000 1001

79D + (-119D) = 0100 1111 + 1000 1001 = [0] 1101 1000

What is this number? It is negative (1 in MSB position), so its magnitude is obtained by getting its 2's complement: 0010 0111 + 1 = 0010 1000 = 2^5 + 2^3 = 40.

Thus the answer is correct, but no borrow is being generated for use by higher byte positions if desired.

Thus it can be seen that, if subtraction is performed by two's complement addition, an incorrect borrow will result. Some processors (for example, the Intel 8085) compensate for this by complementing the carry flag internally after a subtraction (two's complement) operation.

Examples

397 - 46, 16 bits

397/8 = 49 R 5

49/8 = 6 R 1

=> 397D = 615Q = 1 1000 1101B = 018DH

46/8 = 5R6 => 46D = 56Q = 10 1110B = 002EH

-46 = 2's complement (0000 0000 0010 1110) = 1111 1111 1101 0010

397 - 46 = 0000 0001 1000 1101 + 1111 1111 1101 0010 = 0000 0001 0101 1111 = 15FH = 256 + 5*16 + 15 = 256 + 80 + 15 = 351 decimal.