Data Conversions

A C program at machine-level is an assembly language program.  Assembly language uses hexadecimal representation for data.  The hardware itself processes information in bits.  When a program outputs data in hexadecimal or binary form, we may prefer to convert it into decimal form.

This chapter describes how to convert across binary, hexadecimal and decimal representations and shows what a trivially simple program looks like in binary and hexadecimal representations.

Binary - Hexadecimal

The most convenient base for storing byte-wise information is hexadecimal (base 16).  Two hexadecimal (base 16) digits can represent one byte of information.  Each hexadecimal digit represents 4 bits of binary information.

For example, the hexadecimal value 0x5C is equivalent to the binary 010111002.  The 0x prefix identifies the number as hexadecimal notation.  The digits in the hexadecimal number system are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.  The characters A through F denote decimal values 10 through 15 respectively.

Binary to Hexadecimal

To convert a binary number to its hexadecimal equivalent, we:

1. group the bits into nibbles,
2. assign powers of 2 to the different bits in each nibble,
3. multiply each bit value by the corresponding power of 2,
4. add the products together for each nibble, and
5. concatenate the nibble results

Consider the 8-bit number 010111002:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Nibble #** | 1 | | | | 0 | | | |
| **Bit #** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Multiplier** | 8 | 4 | 2 | 1 | 8 | 4 | 2 | 1 |
| **Contents** | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| **Nibble Values** | 0\*8 + 1\*4 + 0\*2 + 1\*1 = 0x5 | | | | 1\*8 + 1\*4 + 0\*2 + 1\*0 = 0xC | | | |
| **Byte Value** | 0x5C | | | | | | | |

Hexadecimal to Binary

To convert a hexadecimal number into its binary equivalent, we work from the lowest order bit to the highest.  We identify the lowest order bit as the first target bit, then

* divide by 2,
* put the remainder into the target bit,
* change the target to the next higher order bit, and

repeat the above.  Consider the hexadecimal number 0x5C:

* Identify the first target bit as bit 0
* Divide the number (0x5C) into left and right hexadecimal digits
* Take the right digit (0xC), divide it by 2 and put the remainder (0) in bit 0
* Take the result (0x6), divide it by 2 and put the remainder (0) in bit 1
* Take the result (0x3), divide it by 2 and put the remainder (1) in bit 2
* Take the result (0x1), divide it by 2 and put the remainder (1) in bit 3
* Take the left hexadecimal digit (0x5), divide it by 2 and put the remainder (1) in bit 4
* Take the result (0x2), divide it by 2 and put the remainder (0) in bit 5
* Take the result (0x1), divide it by 2 and put the remainder (1) in bit 6
* Take the result (0x0), divide it by 2 and put the remainder (0) in bit 7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit #** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Byte Value** | 0x5C | | | | | | | |
| **Nibble Values** | 0x5 | | | | 0xC | | | |
| **Divide by 2** | 0 | 0 | 1 | 2 | 0 | 1 | 3 | 6 |
| **Bit Values** | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

Decimal - Binary

To convert a non-negative integer into its binary equivalent, we start with the value and an empty container that consists of target bits.  We take the integer value, identify the lowest order bit in the container as our target bit, and then

* divide the value by 2,
* store the remainder in the target bit,
* take the result as the new integer value,
* identify the next higher-order bit our new target bit, and
* repeat this set of instructions until no value is left

Consider the value 92:

* Identify the target bit as bit numbered 0
* Take 92, divide it by 2 and put the remainder (0) in bit 0
* Take the result (46), divide it by 2 and store the remainder (0) in bit 1
* Take the result (23), divide it by 2 and store the remainder (1) in bit 2
* Take the result (11), divide it by 2 and store the remainder (1) in bit 3
* Take the result (5), divide it by 2 and store the remainder (1) in bit 4
* Take the result (2), divide it by 2 and store the remainder (0) in bit 5
* Take the result (1), divide it by 2 and store the remainder (1) in bit 6
* Take the result (0), divide it by 2 and store the remainder (0) in bit 7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit #** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Value** | 0 | 1 | 2 | 5 | 11 | 23 | 46 | 92 |
| **Bit Values** | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

(Eight bits and right to left bit numbering are for brevity and illustrative purposes only.)

To convert a binary number into its decimal equivalent, we multiply the value in each bit by its corresponding power of 2 and add the products together.

Consider the 8-bit binary number 010111002:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit #** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Power of 2** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Bit Values** | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| **Multiplier** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| **Byte Value** | 0\*128 + 1\*64 + 0\*32 + 1\*16 + 1\*8 + 1\*4 + 0\*2 + 0\*1 = 92 | | | | | | | |