**[浮点数在内存中的存储方式](http://www.cnblogs.com/dolphin0520/archive/2011/10/02/2198280.html)**

任何数据在内存中都是以二进制的形式存储的！无论是整数、实数（浮点数）、还是正负数等。

则在Intel CPU架构的系统中，存放方式以小端模式（Little Endian, 即低字节存在低地址中，每个存储单元存储一个字节）。但是对于浮点数在内存是如何存储的?目前所有的C/C++编译器都是采用IEEE所制定的标准浮点格式，即二进制科学表示法。

在二进制科学表示法中，S=M\*2^N 主要由三部分构成：符号位+阶码(N)+尾数(M)。对于float型数据，其二进制有32位，其中符号位1位，阶码8位，尾数23位；对于double型数据，其二进制为64位，符号位1位，阶码11位，尾数52位。

Float类型数据在内存中的存储格式：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 内存存储单元 | | | | | |
| 高地址4 | |  | |  | 低地址1 |
| 1bit符号位 | 阶码8bits | | 23bits尾数 | | |

Double类型数据在内存中的存储格式：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 内存存储单元 | | | | | | | | | |
| 高地址8 |  | |  |  | |  |  |  | 低地址1 |
| 1bit符号位 | | 阶码11bits | | | 52bits尾数 | | | | |

阶码：采用移码表示，对于float型数据其规定偏置量为127,阶码有正有负，对于8位二进制，则其表示范围为-128-127，double型规定为1023，其表示范围为-1024-1023。比如对于float型数据，l例如，阶码的真实值为2，则加上127后为129，其阶码表示形式为1000 0001。

尾数:有效数字位，即部分二进制位(小数点后面的二进制位)，因为规定M的整数部分恒为1，所以这个1就不进行存储了。

举例说明：float型数据125.5转换为标准浮点格式

将125.5转换为二进制形式，表示为1111101.1，由于规定尾数的整数部分恒为1，则表示为1.1111011\*2^6，阶码为6，加上127为133，则表示为10000101，而对于尾数将整数部分1去掉，为1111011，在其后面补0使其位数达到23位，则为11110110000000000000000

则其二进制表示形式为

0 10000101 11110110000000000000000，则在内存中以字节（1个存储单元）为单位的存储形式为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 高地址 | | 在内存存储单元中的存储形式 | | 低地址 | |
| 0100 0010 | 1111 1011 | | 0000 0000 | | 0000 0000 |
| 以标准C语言编程输出为： | | | | | |
| 66 | -5(负数，补码形式) | | 0 | | 0 |

由上分析可知，float型数据最大表示范围为

1.11111111111111111111111\*2^127=3.4\*10^38

对于double型数据情况类似，只不过其阶码为11位,偏置量为1023，尾数为52位。

测试程序：

/\*测试浮点型数据在内存中存放方式\*/

#include <stdio.h>

int main(int argc, char \*argv[])

{

float a=125.5;

char \*p=(char \*)&a;

printf("%d\n",\*p);

printf("%d\n",\*(p+1));

printf("%d\n",\*(p+2));

printf("%d\n",\*(p+3));

return 0;

}

运行输出结果为：

0

0

-5

66