Datawhale

Datawhale 开源社区

Datawhale open source community

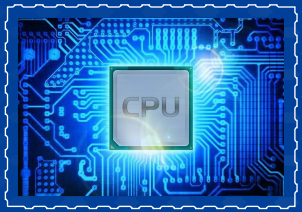
深入理解计算机系统（3）

Computer Systems A Programmer's Perspective CSAPP

李岳昆、易远哲

realyurk@gmail.com、yuanzhe.yi@outlook.com

2021 年 12 月 18 日



第 I 部分

信息的表示和处理-I

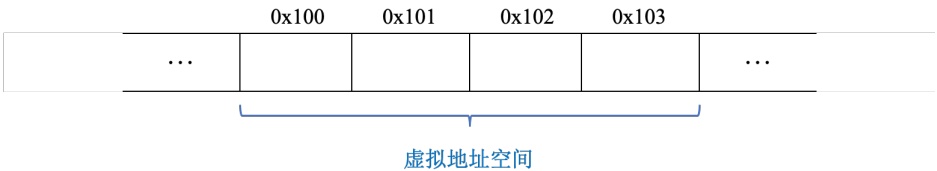
信息的存储

整数表示与编码

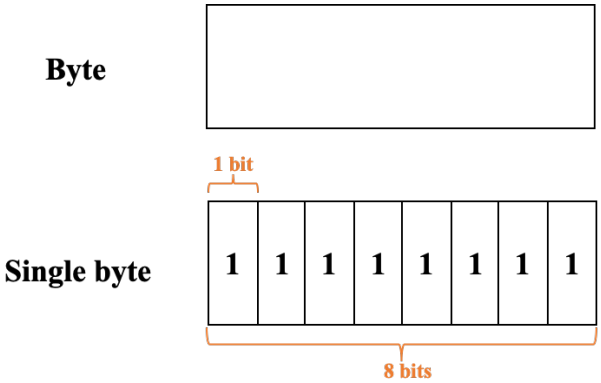
虚拟地址空间

Datawhale

通常情况下，程序将内存视为一个非常大的数组，数组的元素是由一个个的字节组成，每个字节都由一个唯一的数字来表示，我们称为**地址**（address），这些所有的地址的集合就称为**虚拟地址空间**（virtual address space）。



Byte 与 bit



二进制范围：00000000～11111111

十进制范围：0～255

字数据大小

Datawhale

字长决定了虚拟地址空间的最大的可以到多少，也就是说，对于一个字长为 w 位的机器，虚拟地址的范围是 0 到 2^w - 1。

表: 几种对比

字长	地址空间
w bit	$0\sim 2^w-1$
32 bit	$0\sim 2^{32}-1$, 4GB
64 bit	$0\sim 2^{64}-1$, 16EB

字数据大小

Datawhale

- 近些年，高性能服务器、个人电脑以及智能手机已经完成了从 32 位字长到 64 位字长迁移。不过在一些嵌入式的应用场景中，32 位的机器仍旧占有一席之地。对于 32 位的机器，虚拟地址空间最大为 4GB，而 64 位的机器，虚拟地址空间最大为 16EB。
- 在迁移的过程中，大多数 64 位的机器做了向后兼容，因此为 32 位机器编译的程序也可以运行在 64 位机器上。在 64 位的机器上，可以通过这条命令编译生成可以在 32 位机器上运行的程序。[1]
- linux> gcc -m32 -o hello32 hello.c
- 通过修改编译选项，就可以编译生成在 64 位机器上运行的程序。[2]
- linux> gcc -m64 -o hello64 hello.c

[1]注意，hello32 既可以运行在 32 位机器上，也可以运行在 64 位机器上，但是 hello64 只能运行在 64 位的机器上。

[2]对于 32 位程序和 64 位程序，主要的区别还是在于程序是如何编译的，而不是运行机器的类型。

寻址和字节顺序

Datawhale

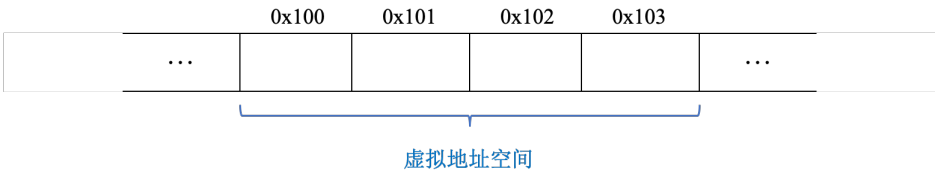
C 语言中，支持整数和浮点数多种数据格式，下表列式了不同数据类型在 32 位机器与 64 位机器上所占字节数的大小。

C 语言声明		Bytes	
有符号	无符号	32-bit	64-bit
[有符号]char	无符号 char	1	1
short	无符号 short	2	2
int	无符号	4	4
long	无符号 long	4	8
int32_t	uint_32t	4	4
int64_t	uint64_t	8	8
char*		4	8
float		4	4
double		8	8

寻址和字节顺序

Datawhale

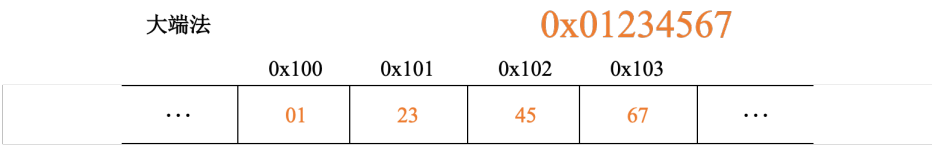
- 从这个表中，我们可以看到很多数据类型都是占用了多个字节空间。对于我们需要存储的数据，我们需要搞清楚该数据的地址是什么，以及数据在内存中是如何排布的。
- 例如：一个 int 类型的变量 x（0x01234567），假设地址位于 0x100 处，由于 int 类型占 4 个字节，因此变量 x 被存储在地址为 0x100,0x101,0x102,0x103 的内存处。



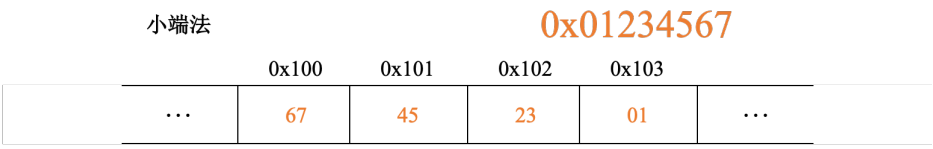
寻址和字节顺序

Datawhale

- 首先我们看一下大端法，最高有效字节[3]存储在最前面，也就是低地址处.



- 另外一种规则就是小端法，最低有效字节存储在最前面.



[3]注意对于变量 x，最高有效字节是 0x01，最低有效字节是 0x67

寻址和字节顺序

大多数 Intel 兼容机采用小端模式，IBM 和 Sun 公司的机器大多数机器采用大端法。对于很多新的处理器，支持双端法，可以配置成大端或者小端运行。例如基于 ARM 架构的处理器，支持双端法，但是 Android 系统和 iOS 系统却只能运行在小端模式

《深入理解计算机系统》的原书中，分别在以下 4 种不同的机器进行了程序测试：

1. 运行 linux 系统，字长为 32 位的机器；
2. 运行 windows 系统，字长为 32 位的机器；
3. SUM，大端法的机器；
4. 运行 linux 系统，字长为 64 位的机器.

寻址和字节顺序

Datawhale

测试代码

```
#include<stdio.h>
typedef unsigned char *byte_pointer;
void show_bytes(byte_pointer start, int len){
    int i;
    for(i = 0; i < len; i++){
        printf(" %.2x", start[i]);
    }
    printf("\n");
}

void show_int(int x){
    show_bytes((byte pointer) &x,sizeof(×));
}
```

具体的运行结果下表所示，12345 的十六进制表示为 0x00003039。

机器	值	类型	Byte（十六进制）
Linux	12,345	int	39 30 00 00
Windows	12,345	int	39 30 00 00
Sun	12,345	int	00 00 30 39
Linux 64	12,345	int	39 30 00 00

除了字节顺序之外，在所有机器上都得到了相同的结果。

机器	值	类型	Byte（十六进制）
Linux	12,345.0	float	00 e4 40 46
Windows	12,345.0	float	00 e4 40 46
Sun	12,345.0	float	46 40 e4 00
Linux 64	12,345.0	float	00 e4 40 46

由于不同操作系统使用不同的存储分配规则，指针的值是完全不同的。

机器	值	类型	Byte（十六进制）
Linux	& ival	int*	e4 f9 ff bf
Windows	& ival	int*	b4 cc 22 00
Sun	& ival	int*	ef ff fa 0c
Linux 64	& ival	int*	b8 11 e5 ff ff 7f 00 00

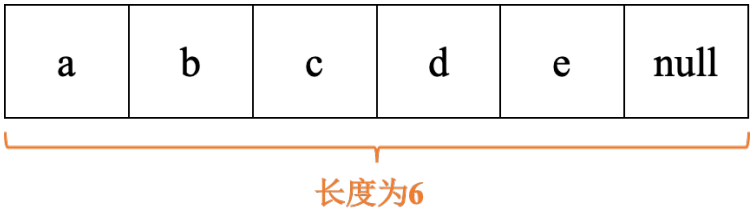
32 位的机器，使用 4 字节的地址，64 位的机器使用 8 字节的地址。虽然整型和浮点数都是对数值 12345 进行编码，但是它们却有着完全不同的字节模式。

类型	值	十六进制
int	12,345	0x00003039
float	12,345.0	0x4640E400

表示字符串

Datawhale

C 语言中的字符串被编码为以 NULL 字符结尾的字符数组，例如字符串“abcde”，这个字符串虽然只有 5 个字符，但是长度却为 6，就是因为结尾字符的存在。



通过以下程序可以得到每个字符在内存中对应的存储信息。

测试代码

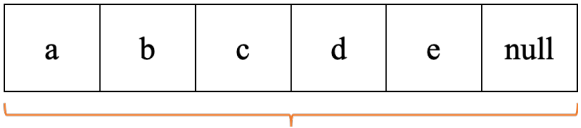
```
#include<stdio.h>
typedef unsigned char *byte_pointer;
void show_bytes(byte_pointer start, int len){
    int i;
    for(i = 0; i < len; i++){
        printf(" %.2x", start[i]);
    }
    printf("\n");
}

void show_int(int x){
    show_bytes((byte pointer) &x,sizeof(×));
}
```

字符串

Datawhale

const char *s="abcde";



show_bytes((byte_pointer)s,strlen(s));

61 62 63 64 65 00

其中结尾字符的十六进制表示为 0x00，使用 ASCII 码来表示字符，在任何系统上都会得到相同的结果。因此，文本数据比二进制数据具有更强的平台独立性。

布尔运算

Datawhale

表: 非（NOT）

~	
0	1
1	0

表: 或（OR）

|	0	1
0	0	1
1	1	1

表: 与（AND）

&	0	1
0	0	0
1	0	1

表: 异或（EOR）

^	0	1
0	0	1
1	1	0

C 语言中的位级运算

C 语言中的一个特性就是支持按位进行布尔运算，确定一个位级表达式结果的最好方法，就是将十六进制扩展成二进制表示，然后按位进行相应的运算，最后再转换回十六进制。

C 表达式	二进制表达	二进制结果	十六进制
~ 0x41	~ [0100 0001]	[1011 1110]	0xBE
~ 0x00	~ [0000 0000]	[1111 1111]	0xFF
0x69 & 0x55	[0110 1001]&[0101 0101]	[0100 0001]	0x41
0x69 | 0x55	[0110 1001]|[0101 0101]	[0111 1101]	0x7D

位运算一个常见的用法就是实现掩码运算，通俗点讲，通过位运算可以得到特定的位序列。例如对于操作数 0x89ABCDEF，我们想要得到该操作数的最低有效字节的值，可以通过 & 0xFF，这样我们就得到了最低有效字节 0x0000 00EF。

C 语言中的位级运算

除了位级运算之外，C 语言还提供了一组逻辑运算，注意逻辑运算的运算符与位级运算容易混淆。逻辑运算认为所有非零的参数都表示 true，只有参数 0 表示 false。

表达式	结果
!0x41	0x00
!0x00	0x01
!!0x41	0x01
0x69 && 0x55	0x01
0x69 || 0x55	0x01

对于 if(a && 5/a) 表达式，如果 a 等于 0，该逻辑运算的结果即为 false，不用再去计算 5 除以 a，这样就可以避免了出现 5 除以 0 的情况。

数值信息的表示

Datawhale

有符号的二进制数的表示

十进制数有正负之分，那么二进制数也有正负之分。带有正、负号的二进制数称为**真值**，例如 +1010110、-0110101 就是真值。为了方便运算，在计算机中约定: 在有符号数的前面增加 1 位符号位，用 0 表示正号，用 1 表示负号。这种在计算机中用 0 和 1 表示正负号的数称为机器数。目前常用的机器数编码方法有源码、反码和补码 3 种。

数值信息的表示

Datawhale

1. 原码
   - 正数的符号位用“0”表示，负数的符号位用“1”表示，其余数位表示数值本身[4]。
   - 例如:X=+1010110, $[X]_{原}$=01010110.
   - Y=-0110101, $[Y]_{原}$=10110101.

延拓

原码方法很简单，但是用原码表示的数在计算机中进行加减法运算很麻烦。比如遇到两个异号数相加或者两个同号数相减时，就要做减法。为了简化运算器的复杂性，提高运算速度，需要把减法做成加法运算，因此人们引入了反码和补码。

[4]对于 0，我们既可以认为是 +0，也可以认为是-0，因此它的原码并不唯一。$[+0]_{原}$=00000000 $[-0]_{原}$=10000000

Datawhale

❷ 反码

- 正数的反码与其原码相同; 负数的反码是在原码的基础上保持符号位不变，其余各位按位求反得到的。
- 例如:X=+1010110, $[X]_{反}=[X]_{原}=01010110$.
- Y=-0110101, $[Y]_{原}=10110101$ $[Y]_{反}=11001010$. .

❸ 补码

- 正数的补码与其原码相同; 负数的补码是在原码的基础上保持符号位不变，其它的数位 1 变为 0，0 变为 1，最后再加 1 运算。也就是说，负数的补码是它的反码加 1。在计算机中，有符号整数常常用补码形式存储[5]。
- 例如:X=+1010110 $[X]_{补}=[X]_{反}=[X]_{原}=01010110$.
- Y=-0110101 $[Y]_{原}=10110101$ $[Y]_{补}=11001011$.
- 对于任意一个数它的补码的补码是原码，即 $[[X]_{补}]_{补}=[X]_{原}$.

[5]补码中 0 无正负之分，即 $[+0]_{补}=[-0]_{补}=00000000$

补码意义

Datawhale

补码的计算规则为什么是这样的?

- 我们该如何理解补码的意义? 一般情况下我们有两种理解方法:
  1. 对于一个数:-5，其原码为 1101_2，此时最高位是符号位。若想找到其补码则需找到与 $+5$ 和为 10000_2 的二进制码，由于计算机中二进制是从右向左计算的，而我们只有 4 个 bit，因此多出来的 1 会溢出以达到效果。$+5$ 的原码为 0101_2，设-5 的补码为 1010_2，这两个相加是否可以实现上述结果? 答案是否定的，此时答案为 1111_2，还需要再加上 1，才能让答案归零。
  2. 仍以上文为例，我们可以知道 $[-5]_{补}=1011_2$，此时最高位 1 不仅仅代表了负号，也代表了 8，因此最高位代表-8，而其余两个 1 分别代表 $+2$，$+1$，这样我们仅通过补码就可以直接知道它对应的十进制数字，甚至不需要知道它的原码!

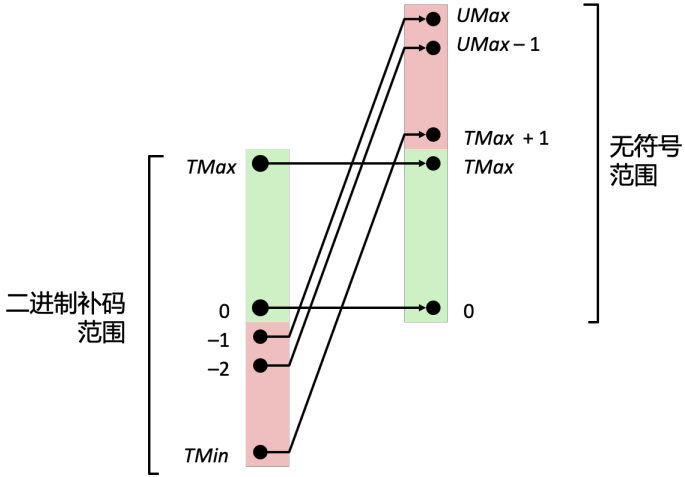
补码意义

Datawhale

4 bit 有符号数范围

对于一个 4 bit 的数据而言，补码所能表达的最小数字为:1000，即-8，在一个 8 bit 的数据中，最小的数字为 1000 0000_2，即:-2^{8-1}=-128，而最大的数为:0111 1111_2，即:2^{8-1}-1=127，故上限为 127，下限为-128，共 255 个数字，也即 1 个 Byte 的范围为:-128~127.

补码意义



TMin=1000_2， TMax=0111_2,UMax=1111_2.

有符号数和无符号数转换

Datawhale

C 语言允许数据类型之间做强制类型转换，例如代码示例，变量 a 是 short 类型，通过强制类型转换成无符号数，那么变量 b 的数值是多少呢?

强制转换

```
1 short int a =-12345;
2 unsigned short b = (unsigned short)a;
3 printf("a= %d, b = %u" , a, b);
```

- -12345 经过强制类型转换后得到的无符号数是 53191, 从十进制的表示来看，很难看出二者的关系，将十进制表示转换成二进制表示，可以发现二者的位模式是一样的。
- -12345: 1100 11111100 0111
- 53191: 1100 11111100 0111

相互转化

1. 有符号转无符号
   - 用 T2U 来表示有符号数到无符号数的函数映射，当最高位 X_{w-1} 等于 1 时，此时有符号数 x 表示一个负数，经过转换后，得到的无符号数等于该有符号数加上 2^w；当最高位 X_{w-1} 等于 0 时，此时有符号数 x 表示一个非负数，得到的无符号数与有符号数是相等的。

$$T2U_w(x) = \begin{cases} x + 2^w, & x < 0 \\ x, & x \geqslant 0 \end{cases}$$

❷ 无符号转有符号

- 用 U2T 来表示无符号数到有符号数的函数映射。当最高位等于 0 时，无符号数可以表示的数值小于有符号数的最大值，此时转换后的数值不变。当最高位等于 1 时，无符号数可以表示的数值大于有符号数的最大值，在这种情况下，转换后得到有符号数等于该无符号数减去 2^w 。

$$U2T_w(u) = \begin{cases} u, & u \leqslant TMax_w \\ u - 2^w, & x > TMax_w \end{cases}$$

为什么要了解这个转换

在 C 语言中，在执行一个运算时，如果一个运算数是有符号数，另外一个运算数是无符号数，那么 C 语言会隐式的将有符号数强制转换成无符号数来执行运算。

强制转换

```
int i = -1;
unsigned int b = 0;
if(a < b)
    printf("-1 < 0")
else
    printf("-1 > 0")
```

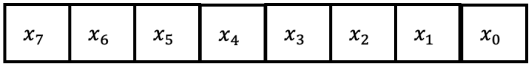
- 输出-1>0，由于第二个操作数 b 是无符号数，第一个操作数 a 就隐式的转换成无符号数，这个表达式实际上比较的是 $4294967295(=2^{32}-1) < 0$。
- C 语言中将一个较小数据类型转换成较大的类型时，保持数值不变是可以的；但大转小不行.

扩展一个数字的位表示

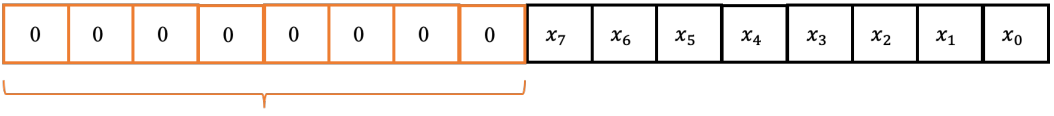
Datawhale

先来看一下把无符号数转换成一个更大的数据类型，例如，我们将一个 unsigned char 类型变量，转换成 unsigned short 类型。变量 a 占 8 个 bit 位，而变量 b 占 16 个 bit 位，对于无符号数的转换比较简单，只需要在扩展的数位进行补 0 即可，我们将这种运算称为零扩展，具体表示如图所示。

无符号**char a;**



无符号**short b;**

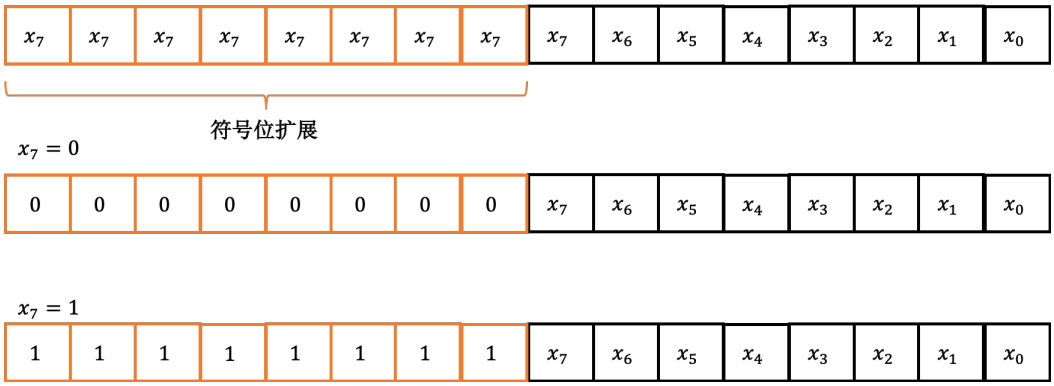


扩展一个数字的位表示

Datawhale

当有符号数表示非负数时，最高位是 0，此时扩展的数位进行补零即可；当有符号数表示负数时，最高位是 1，此时扩展的数位需要进行补 1。

short b;



扩展一个数字的位表示

Datawhale

转换定理

当有符号数从一个较小的数据类型转换成较大类型时，进行符号位扩展，可以保持数值不变。[6]

[6]可以通过两种转换映射函数作差进行证明，此处略.

截断

Datawhale

将 int 类型强制类型转换成 short 类型时，int 类型高 16 位数据被丢弃，留下低 16 位的数据，因此截断一个数字，可能会改变它原来的数值。

- 无符号数：
  1. 将一个 w 位的无符号数，截断成 k 位时，丢弃最高的 w-k 位，截断操作可以对应于取模运算，即除以 2 的 k 次方之后得到的余数。
- 有符号数：
  1. 我们用无符号数的函数映射来解释底层的二进制位，这样一来我们就可以使用与无符号数相同的截断方式，得到最低 K 位；
  2. 我们将第一步得到的无符号数转换成有符号数。