第二十三节：减法溢出与假想借位。

【23.1　减法溢出与假想借位。】

英文“unsigned”的中文意思就是”无符号的”，延伸含义是“无负号无负数”的意思，所以 unsigned char ,unsigned int ,unsigned long 这三种类型数据都是无负号无负数的，取值只能是 0 和正数，那么问题来了，当被减数小于减数的时候，运算结果会是什么样子，有什么规律？这就是本节要研究的减法溢出。

第一个例子:

```
unsigned char a;
a=0-1;
```

分析：

左边的“保存变量”a 的数据长度是 1 个字节 8 位，a=0-1 可以看成是十六进制的 a=0x00-0x01。由于 0x00 比 0x01 小，所以假想一下需要向高位借位，借位后成了 a=0x100-0x01。所以 a 的最终结果是 0xff(十进制是 255)，这个“假想一下需要向高位借位”的过程就是本节制造的新概念“假想借位”。根据“假想借位”这个规律，如果是 b 也是 unsigned char 类型，那么 b=2-5 自然就相当于 b=0x102-0x05，运算结果 b 等于 0xfd(十进制是 253)。

第二个例子:

```
unsigned int c;
c=0-1;
```

分析：

左边的“保存变量”c 的数据长度是 2 个字节 16 位，c=0-1 可以看成是十六进制的 c=0x0000-0x0001。由于 0x0000 比 0x0001 小，所以假想一下需要向高位借位，借位后成了 c=0x10000-0x0001。所以 c 的最终结果是 0xffff(十进制是 65535)。根据“假想借位”这个规律，如果是 d 也是 unsigned int 类型，那么 d=2-5 自然就相当于 d=0x10002-0x0005，运算结果 d 等于 0xfffd(十进制是 65533)。

综合分析：

为什么上述例子中会出现数据越减越大的奇葩现象？是因为减法溢出，是因为“假想借位”中的“借”是“光借不还”。一句话，根本问题就是溢出问题。

【23.2　因为减法溢出，所以加减顺序.....】

第三个例子：请分析下面例子中 e 和 f 因加减运算顺序不同而引发什么问题。

```
unsigned char e;
unsigned char f;
e=1-6+7;
f=1+7-6;
```

用两种思路分析：

第一种思路：只看过程不看结果。加减法的运算优先级是从左到右，e 先减法后加法，1 减去 6 就有溢出了，所以过程有问题。而 f 先加法后减法，整个过程没有问题。

第二种思路：先看结果再分析过程。e 的运算结果居然是 2，f 的运算结果也是 2。好奇怪，既然 e 的过程有问题，为什么运算结果却没有问题？其实 e 发生两次溢出，第一次是减法溢出，第二次是加法溢出，所以“溢溢得正”(这句话是开玩笑的)。1-6“假想借位”后相当于 0x101-0x06，运算结果等于 0xfb（十进制是 251），然后 0xfb 再加上 0x07 等于 0x102，因为 e 是 unsigned char 类型只有 1 个字节，根据加法溢出的规律，最后只保留了低 8 位的一个字节 0x02，所以运算结果就是十进制的 2。

结论：

虽然 e 的运算结果侥幸是对的，但是其运算过程发生了溢出是有问题的，当运算式子更复杂一些，比如有不同类型的变量时，就有可能导致运算结果也出错。所以得出的结论是：在加减法运中，为了减少出现减法溢出的现象，建议先加法后减法。在后续章节讲到的乘除法运算中，为了减小运算带来的误差也建议大家先乘法后除法。

【23.3　例程练习和分析。】

现在我们编写一个程序来验证上面讲到的例子：

程序代码如下：

```
/*---C 语言学习区域的开始。-----------------------------------------------*/

void main() //主函数
{
    unsigned char a;        //定义一个变量 a，并且分配了 1 个字节的 RAM 空间。
    unsigned char b;        //定义一个变量 b，并且分配了 1 个字节的 RAM 空间。
    unsigned int c;         //定义一个变量 c，并且分配了 2 个字节的 RAM 空间。
    unsigned int d;         //定义一个变量 d，并且分配了 2 个字节的 RAM 空间。
    unsigned char e;        //定义一个变量 e，并且分配了 1 个字节的 RAM 空间。
    unsigned char f;        //定义一个变量 f，并且分配了 1 个字节的 RAM 空间。

    //第一个例子,针对 a 与 b 都是 unsigned char 类型数据。
    a=0-1;
    b=2-5;

    //第二个例子,针对 c 与 d 都是 unsigned int 类型的数据。
    c=0-1;
    d=2-5;

    //第三个例子,e 与 f 的加减顺序不一样。
    e=1-6+7;
    f=1+7-6;

    View(a);            //把第 1 个数 a 发送到电脑端的串口助手软件上观察。
    View(b);            //把第 2 个数 b 发送到电脑端的串口助手软件上观察。
    View(c);            //把第 3 个数 c 发送到电脑端的串口助手软件上观察。
    View(d);            //把第 4 个数 d 发送到电脑端的串口助手软件上观察。
    View(e);            //把第 5 个数 e 发送到电脑端的串口助手软件上观察。
    View(f);            //把第 6 个数 f 发送到电脑端的串口助手软件上观察。

    while(1)
    {
    }
```

```
    }

    /*---C 语言学习区域的结束。-------------------------------------------*/
```

在电脑串口助手软件上观察到的程序执行现象如下：

```
开始...

第 1 个数
十进制:255
十六进制:FF
二进制:11111111

第 2 个数
十进制:253
十六进制:FD
二进制:11111101

第 3 个数
十进制:65535
十六进制:FFFF
二进制:1111111111111111

第 4 个数
十进制:65533
十六进制:FFFD
二进制:1111111111111101

第 5 个数
十进制:2
十六进制:2
二进制:10

第 6 个数
十进制:2
十六进制:2
二进制:10
```

分析：

通过实验结果，发现在单片机上的计算结果和我们的分析是一致的。

【23.4　如何在单片机上练习本章节 C 语言程序？】

直接复制前面章节中第十一节的模板程序，练习代码时只需要更改“C 语言学习区域”的代码就可以了，

其它部分的代码不要动。编译后，把程序下载进带串口的 51 学习板，通过电脑端的串口助手软件就可以观察到不同的变量数值，详细方法请看第十一节内容。