

第二十九节：“先余后商”和“先商后余”提取数据某位，哪家强？

【29.1 先余后商。】

求商求余除了数学运算外，在实际单片机项目中还有一个很常用的功能，就是提取某个数的个十百千位。提取这些位有什么用呢？用途可大了，几乎凡是涉及界面显示的项目都要用到，比如数码管的显示，液晶屏的显示。提取某个数的个十百千位是什么意思呢？比如 8562 这个数，提取处理后，就可以得到千位的 8，百位的 5，十位的 6，个位的 2。这里提到的“个，十，百，千”位只是一个虚数，具体是多少应该根据实际项目而定，也有可能是“个，十，百，千，万，十万，百万...”等位，总之，提取的思路和方法都是一致的。下面以 8562 这个数为例开始介绍提取的思路和方法。

第一步：先把 8562 拆分成 8562, 562, 62, 2 这四个数。怎么拆分呢？用求余的算法。比如：

```
8562 等于 8562%10000;  
562 等于 8562%1000;  
62 等于 8562%100;  
2 等于 8562%10;
```

第二步：再从 8562, 562, 62, 2 这四个数中分别提取 8, 5, 6, 2 这四个数。怎么提取呢？用求商的算法。比如：

```
8 等于 8562/1000;  
5 等于 562/100;  
6 等于 62/10;  
2 等于 2/1;
```

第三步：最后，把第一步和第二步的处理思路连写在一起如下：

```
8 等于 8562%10000/1000;  
5 等于 8562%1000/100;  
6 等于 8562%100/10;  
2 等于 8562%10/1;
```

仔细观察，上述处理思路的规律感特别清晰，我们很容易发现其中的规律和原因，如果要提取“万，十万，百万...”的位数，也是用一样的思路。另外，多说一句，根据我的经验，有一些单片机的 C 编译器可能不支持 long 类型数据的求余求商连写在一起，那么就要分两步走“先求余，再求商”，分开来操作。比如：

```
unsigned char a;  
a=8562%10000/1000; //提取千位。
```

分成两步走之后如下：

```
unsigned char a;  
a=8562%10000;  
a=a/1000;           //提取千位。
```

提取其它位分两步走的思路也是一样，不多说。

【29.2 先商后余。】

刚才讲到了“先余后商”的提取思路，其实也可以倒过来“先商后余”，也就是先求商再求余数。下面还是以 8562 这个数为例。

第一步：先把 8562 拆分成 8, 85, 856, 8562 这四个数。怎么拆分呢？用求商的算法。比如：

```
8 等于 8562/1000;  
85 等于 8562/100;  
856 等于 8562/10;  
8562 等于 8562/1;
```

第二步：再从 8, 85, 856, 8562 这四个数中分别提取 8, 5, 6, 2 这四个数。怎么提取呢？用求余的算法。比如：

```
8 等于 8%10;  
5 等于 85%10;  
6 等于 856%10;  
2 等于 8562%10;
```

第三步：最后，把第一步和第二步的处理思路连写在一起如下：

```
8 等于 8562/1000%10;  
5 等于 8562/100%10;  
6 等于 8562/10%10;  
2 等于 8562/1%10;
```

上述的规律感也是特别清晰的。

【29.3 “先余后商”和“先商后余”哪家强？】

上面讲了“先余后商”和“先商后余”这两种思路，到底哪种思路在实际项目中更好呢？其实我个人倾向于后者的“先商后余”，为什么呢？请看这个例子，以 3100000000 这个数为例，要提取该数的“十亿”位 3。

第一种：用“先余后商”的套路如下：

```
3 等于 3100000000%1000000000/1000000000;
```

这里出现了一个问题，我们知道，unsigned long 类型最大的数据是 0xffffffff，转换成十进制后最大的数是 4294967295，但是上面出现的 1000000000 这个数比 unsigned long 类型最大的数据 4294967295 还要大，这个就会引来我个人的担忧，C 编译器到底会怎么处理，很有可能会出现意想不到的错误，至少会让我感到心里不踏实。当然，也许会有一些朋友说，这个是多虑的，最高位完全可以把求余这一步省略，这个说法也对，但是作为一种“套路”，我还是喜欢“套路”的对称感，“套路”之所以成为“套路”，是因为有一种对称感。下面再看看如果用“先商后余”的思路来处理，会不会出现这个担忧。

第二种：用“先商后余”的套路如下：

```
3 等于 3100000000/1000000000%10;
```

这一次，上面出现的 1000000000 这个数比 unsigned long 类型最大的数据 4294967295 小，所以没有刚

才那种担忧，也维护了“套路”的对称感。所以我在实际项目中喜欢用这种方法。

【29.4 例程练习和分析。】

现在编写一个程序来验证刚才讲到的两种思路：

程序代码如下：

```
/*---C 语言学习区域的开始。-----*/

void main() //主函数
{
    unsigned char a; //千位
    unsigned char b; //百位
    unsigned char c; //十位
    unsigned char d; //个位

    unsigned char e; //千位
    unsigned char f; //百位
    unsigned char g; //十位
    unsigned char h; //个位

    //x 初始化为 8562，必须是 unsigned int 类型以上，不能是 char 类型，char 最大范围是 255。
    unsigned int x=8562; //被提取的数

    //第一种：先余后商。
    a=x%10000/1000; //提取千位
    b=x%1000/100;    //提取百位
    c=x%100/10;      //提取十位
    d=x%10/1;        //提取个位

    //第二种：先商后余。
    e=x/1000%10;     //提取千位
    f=x/100%10;      //提取百位
    g=x/10%10;       //提取十位
    h=x/1%10;        //提取个位

    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。
    View(g);          //把第 7 个数 g 发送到电脑端的串口助手软件上观察。
    View(h);          //把第 8 个数 h 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:8

十六进制:8

二进制:1000

第 2 个数

十进制:5

十六进制:5

二进制:101

第 3 个数

十进制:6

十六进制:6

二进制:110

第 4 个数

十进制:2

十六进制:2

二进制:10

第 5 个数

十进制:8

十六进制:8

二进制:1000

第 6 个数

十进制:5

十六进制:5

二进制:101

第 7 个数

十进制:6

十六进制:6

二进制:110

第 8 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。

【29.5 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。