

## 目录

第一章 字符与字符转换器.....	2
1.1 字符 .....	2
1.2 显示器.....	2
1.3 编辑器——光标移动逻辑.....	3
1.4 字符转换器.....	4
1.5 第一章练习.....	6
第二章 基本溢出.....	8
2.1 什么是基本溢出.....	8
2.2 基本溢出的方法.....	8
2.3 插入和删除.....	8
2.4 第二章例题.....	11
2.5 附：对回放区进行处理 .....	12
第三章 变量与历史记录 .....	14
3.1 变量格式 .....	14
3.2 历史记录格式 .....	15
3.3 变量和历史记录的重叠 .....	16

# 第一章 字符与字符转换器

从本章开始的各章都围绕着第一个主题：**输入区**。

本章将要讲解字符，显示器，光标的移动和字符转换器的基础知识。

## 1.1 字符

字符的概念相信大家都已明白。字符在内存中储存，对卡西欧的程序员来说，最自然的想法就是每个字符占 1 字节，按序储存。但是，一个字节的数只有  $16^2 = 256$  种可能。上代 fx-ES 型号里，就采取了每个字符 1 字节的办法，结果把控制字符、数字、变量、函数、算符，还有科学常数、单位换算全都挤在 256 个格子里，塞得满满当当。而现在的型号（ClassWiz X），有更多的科学常数和单位换算。（日版有 200 多个单位换算！）

那么单个字节是无论如何也塞不下了。是不是要全部改为双字节呢？那也不至于，因为双字节有 65536 种可能，用不了这么多，而且它会多占用一倍的空间。于是卡西欧的程序员采取了一种折中的办法，即同时采取单字节和双字节的字符。为了区分，规定双字节字符的第一个字节以 F 开头。以下是一些例子：

字符	1	Pa>atm	r <sub>e</sub>	M+
码位	31	FE1A	FD3A	FB1A

**注：**如果要查看完整的字符表，请参见附录一：一级字符表。

特殊地，还要有个特殊字符来标记字符串的结束。它就是字符 00。不过，为了方便，程序中许多操作都以字节为单位来进行，此时对程序来说，标记字符串末尾的就是字节 00，也就是 NUL，不同于字符 00。这就导致像 FE00 这样的字符在程序的不同部分含义不一样，它有时只是一个一般的字符，有时却能标记字符串的结束。一般来说这不要紧，因为这样的字符在正常情况下不会出现；反过来说，异常就出自这种特殊情况。但在探讨异常前，我们要先学习正常情况下各种操作的逻辑。

## 1.2 显示器

输入区的逻辑分为编辑器和显示器两块，编辑器负责处理增、删、移、插等操作，显示器则负责把内存中的字符串打印在屏幕上，两块逻辑之间相对独立。本章讲述显示器和编辑器中的“移”这一操作。

显示器的逻辑是：

1. 从字符串的开头开始；

2. 如果光标在这里，那么在屏幕上显示光标；
3. 取一个字节，如果它以 F 开头，就再取一个，组成一个字符；
4. 如果这个字符是 00，那么退出。（在线性模式中，只要这个字符包含 NUL 就退出，如 FE00）；
5. 否则，在屏幕上显示这个字符，并向右移动相应字节数；
6. 回到第 2 步。

阅读逻辑可以发现，如果光标插在双字节字符的中间，或者在字符 00 的右边（线性模式中，NUL 右边），那么显示器永远找不到光标。在数学模式，光标会停在之前的位置不动（光标默认位置在最左边，如果按第一个键的时候就没找到光标，它就会停在默认位置）；在线性模式，光标会移到最左边。此时我们说“光标卡住了”，即光标显示的位置不能反映它在内存中的实际位置。同学们，这时候千万别被它骗了！

## 1.3 编辑器——光标移动逻辑

左键：

- 如果在历史记录中或在数学模式且光标在开头，那么移到第一个 NUL 处；
- 否则，如果前面第二个字节以 F 开头，就左移 2 字节，反之移 1 字节。

右键：

- 如果在历史记录中或在数学模式且光标在 NUL 处，那么移到开头；
- 否则，如果当字节以 F 开头，就右移 2 字节，反之移 1 字节。

因此，一般来说，不可能把光标塞到双字节字符中间。有一种情况除外：如果某个双字节字符的两个字节都以 F 开头，而它右边有个单字节字符时：

```
F_ F_ _|
```

按 ◀，光标会以为它前面是个双字节字符，于是移动两个字节：

```
F_|F_ _
```

这时，光标就插在双字节字符中间了。但是，这样的字符不能直接打出，方法下面会说。

## 1.4 字符转换器

字符转换器需要用到字符 “1bf/in<sup>2</sup> ▶kPa” (SHIFT 8 ▼ 2 7)。输入

FE23

11bf/in<sup>2</sup> ▶kPa, 按 ☐ ▲。由于某些原因 (今后会讲), 在历史记录里, 字节 23 会被吃掉, 只剩下字节 FE, 和后面的 NUL 拼起来, 就是:

```
31 FE 00 00
```

**注:** 字符 FE00 没有宽度, 看不到, 所以显示效果是只有一个 1。

按 ◀, 光标移动到第一个 NUL 处, 也就是 FE 后面:

```
31 FE|00 00
```

为什么光标好像在 1 上? 仔细看, 这里它就位于双字节字符 FE00 中间, 所以卡住了。不要管它 (因为此时光标不是重点)。接下来, 输入字符 “1” :

31

```
31 FE 31|00 00
```

此时会显示 “1 l . y . ▶ml”。FE31 这个字符本来是打不出的, 但被我们通过字符转

31 FE31

换器得到了。这就是字符转换器。

如果在刚才输入第二个字符 “1” 的位置输入双字节字符会发生什么? 输入字符

“Pa ▶kgf/in<sup>2</sup>” (SHIFT 8 ▼ 2 6):

FE20

```
31 FE FE 20|00 00
```

此时会显示 “1 Unknow i”。按 ◀, 光标向左移动一个字符 (FE20), 看起来好像没有移动:

31 FEFE 20

```
31 FE|FE 20 00 00
```

这是因为它插在双字节字符中间, 卡住了。再按一下 ◀ 才会移动到字符 FEFE 上:

```
31|FE FE 20 00 00
```

按 ▶, 向右移动一个字符 FEFE, 到字符 i 上:

20

```
31 FE FE|20 00 00
```

然后就可以删掉字符 FEFE 了。你看，计算模式中本来不能打出 i，但是它也可以用字符转换器得到。回到上一步，在字符 FEFE 和字符 20 之间插入字符 Ran#，接下来

按 ：




```
31 FE FE FD 18|20 00 00
31 FE FE|FD 18 20 00 00
```

输入字符 “1”：

```
31 FE FE 31|FD 18 20 00 00
```

现在按  会删除双字节字符 FE31：

```
31 FE|FD 18 20 00 00
```

你看，输入了 31，却删除了 FE31。删除字符的方式和按  差不多，但是注意如果前一个字节以 F 开头，就会把这个字节以及后一个字节都删掉，比如 31 FE|FD 32 和 31 FE FD|32 删除都得到 31 32。此时显示 “1 1 t l & O & & s < l s < E @”。光标没有移动，是因为它在双字节字符 FEFD 中间。如果此时想删除单字节字符 @，那是行不通的，因为前面有 FD，会被识别成双字节字符 FD18。必须把它们隔开才能删除字符 18。按  再按 ，把光标移动到 18 上：



```
31 FE FD|18 20 00 20 00 00
```

然后输入任意双字节字符（以 Ran# 为例）：

```
31 FE FD FD 18|18 20 00 00
```

这样 18 和 FEFD 隔开了，就可以正常地删除了。把字符 @ 和前面的 Ran# 都删了，使 FEFD 和 20 接触：

```
31 FE FD|20 00 20 00 00
```

按  再按 ，光标移动到 FEFD 中间：









```
31 FE|FD 20 00 20 00 00
```

现在输入 1：






```
31 FE 31|FD 20 00 00
```

此时显示的是 “l l . y . m a n”，把其他字符都删掉，就获得了字符 an。这个

字符将来用处很大。

两个字节都是 F 开头的字符也可以被用做字符转换器，并可以多次使用。例如，要打出 3A 和 3B，可以用 FD3A (   ) 和 FD3B (   )。在字符转换器中先输入 FD3B：

```
31 FE FD 3B|00 00 00 00
```

  ，移到 FD 和 3B 之间，输入 FD3A，然后按 ，

```
31 FE FD FD 3A|3B 00 00
31 FE FD 31|FD 3A 3B 00 00
```

按  删除 FD31：

```
31 FE|FD 3A 3B 00 3B 00 00
```

此时，只要再删掉 FEFD，就获得了 A 和 B。用这种方法可以连续卡出多个字符。





## 1.5 第一章练习

**本教程所有练习请读者先尽力独立完成再查看解析以达到最佳学习效果。**





### 1. 尝试打出字符 FD31。

参考答案：nn。  
FD31

参考思路：

**法 1（不推荐）：**查字符表得 FD31 的按键为    。此法应用于能直接打出的字符较为方便，但对大量无法直接打出的字符无能为力。

**法 2：**已知 Ran# 可直接打出，为得到 FD31，需设法将光标移至 FD 与 18 之间以插

入 31。由上文可知，若能得到 FEFD，就可以将光标移至目标位置。由本节开头的操作可得 31 FE，请读者熟记这个操作，这是日后许多编辑操作的起点。在 31 FE 后输入 FD18，   即可使光标到达目标位置。输入 1，，再输入 1，得：

```
31 FE 31|FD 31 18 00 00
```

删去 FE31 即得 FD31。

## 2. 尝试打出字符 FB31。

提示：

- 出现语法错误不要惊慌，按 **AC** **◀** 即可回到最左侧。
- 此题完成过程中可能出现字符 FEFB，该字符不可见，请谨慎完成后续操作。
- 先按 **STO** 再按 **[X]** 即可输入 FB18。

参考答案：  $\frac{1}{\text{FB31}}$

**参考思路：**上文的法 1 已不可用，本题思路与上文法 2 一致。先输入 31 FE FB 18，此时会发生语法错误，但 **AC** **▶** 仍可继续编辑。如果你还记得说明书的话，你应该知道，已经算过一次的式子，不管有没有算出来，**AC** **◀** 或 **▶** 都能重新呼出。再按 **▶** **▶**，光标位于 FB|18 之间，依次按 **[1]** **◀** **[1]** 即得  $\frac{1}{\text{FB31}}$ 。

**总结：**类似上述两例的问题都可通过类似的思路得到结果：

- 使所需字符相邻，如例一中使 FD 与 31 相邻
- 拆除原有的双字节字符以得到目标字符，如例二中原本 FEFB 为一个双字节字符，我们通过在 FE 与 FB 之间插入 31 使 FB 与后面的 31 结合得到目标字符。

## 3. 尝试打出字符 3A 到 3F。（FD3A 到 FD3F 分别为 **SHIFT** **[7]** **[3]** **[7]**，...，**SHIFT** **[7]** **[3]** **[X]**）

**参考思路：**同上，操作完一次后不要删除 FEFD，而是 **◀** **▶** 移到 FD 的后面，此时即可进行下一步操作。多次重复就能获得所有需要的字符。

## 4. 综合运用上述内容尝试打出 3A FD20 3B（表现为 **A an B**）


**参考方法：**转换器，FE3B **◀** **◀** **▶** FE20 **◀** **[1]** **DEL** **◀** **▶** FD18 **◀** **[1]** **DEL** **◀** **▶** FD18 **▶** **DEL** **DEL** **▶** **◀** FE3A

**参考思路：**易知所有的字符转换操作都只能围绕字符转换器进行，为按顺序得到答案，我们需要依次得到 3B 20 FD 3A；为得到这四个字节，我们依次在 FE 右侧输入 FE3B (**SHIFT** **[7]** **[3]** **[8]**)，FE20 (**SHIFT** **[8]** **▼** **[2]** **[6]**)，FD18 (**SHIFT** **[◻]**)，FE3A (**SHIFT** **[7]** **[3]** **[7]**)；其余的操作则为输入保护字符，以防字符转换器或者所需字节和其他字节结合无法使用。




## 第二章 基本溢出

### 2.1 什么是基本溢出








字符串的长度是第一个 NUL 的位置（从 0 数起）。例如，31 00 00 的长度是 1，00 00 的长度是 0。光标位置>长度时，就叫基本溢出，此时光标在 NUL 的后面。

一般来说，光标不能移动到 NUL 后面，因为当下一个字符是 00 时，再按  在数学模式中会移动到最左边，在线性模式中没有操作。

### 2.2 基本溢出的方法



1. 字符转换器。在 31 FE 00 00 中，只要从最左边开始连接两下 ，就会到达 FE 00 右边，达成基本溢出。
2.  (框)。框和它右边的字符是“绑定”的，在框左边按  会连续跳过两个字符，19



包括字符 00。因此只要框在最后，就能用它达成基本溢出。那么如何刷出框呢？

**提示：**使用字符转换器，字符 atmPa 是     ，字符 xy 是   是








FE19


FB19


 。

刷框的办法除了字符转换器以外，还有一种：输入  $6 \div 2$ ，不断输入根号直到无法再输入，在根号内填入 1，移至最右端，输入“:” ( )，

   ，最后删去框右侧的括号即可。

3. 线性模式中的  $\Sigma$  (溢出法。需要使用 。
4. 先将 A 赋值为  $1 \times 10^{99}$ ，再输入  $x=AAin \rightarrow cm$  (   )，对  $x$  求解， 。

fx-991CN CW 中没有字符转换器和 ，这是该型号基本溢出的不多方法之一。

5. 当算式中存在连续两个 F 开头的字节时，报错后按  可能导致溢出。

**注意：**此方法不稳定，需谨慎使用。

### 2.3 插入和删除

在上一章中，我们并没有仔细学习插入和删除的逻辑，只是感性理解了一下。这一章要细讲这个问题。



## 2.3.1 插入

先来看看插入。假设现在字符串的总长度是  $L$ ，从左侧第一个字符开始计算，到右边的第一个 `00` 为止，不包括这个 `00`；记最左侧坐标为  $0$ ，每向右一个字节坐标加一，则光标坐标为  $P$ ，字节位置算法同理，最左侧的字节是第零个字节；插入的字符长度  $N$ 。（如果在框上插入，因为插完后框就没了，所以  $N$  要减 1）那么会进行三个操作：

1. 第  $L+N$  个字节设为 `00`；
2. 把  $[P, L)$  移动到  $[P+N, L+N)$ ，空出  $[P, P+N)$ ；
3.  $[P, P+N)$  设为要插入的字符，光标移动到  $P+N$ 。

这个算法一般来说工作良好，可以完美实现需要的操作。但在基本溢出时却达不到应有的效果。例如，在 `31|33 34 00` 中插入 32，此例中  $L=3$ ， $P=1$ ， $N=1$ ：

```
31|33 34 00 ?? ?? ?? ... //?表示此字节值不确定
31|33 34 00 00 ?? ?? ...
31|00 33 34 00 ?? ?? ...
31 32|33 34 00 ?? ?? ...
```

但是在 `31 00 32|34 35 00` 中插入 33， $L=1$ ， $P=3$ ， $N=1$ ：









```
31 00 32|34 35 00 ...
31 00 00|34 35 00 ... //由于 P>L，并不会移动字符
31 00 00 33|35 00 ...
```

可见有两个效果：第一，产生了一个 NUL，第二，新字符直接替换了旧字符。第二点可以用来便捷快速地把双字节字符变成单字节字符，第一点可以用来去除双字节字符的最后一个字节，得到字符转换器。

例如，在框上按 ，然后       ，就可得到字符转换器：


```
|19 00 00 00 00 00 00 ...
19 00|00 00 00 00 00 ...
19 00 FD 18|00 00 00 ...

19 00 FD 00 FD 18|00 ... //注意前一个 FD18 的 18 变成 00 了
19 00|FD 00 FD 18 00 ...
19|FD 00 FD 18 00 00 ... //删除了单字节字符 00
```

还记得吗，数学模式中在最左边处按 ，会移动到第一个 NUL 处。按  直至移至最左端，再按 ，然后输入 Pa<sup>2</sup>kgf/in<sup>2</sup> (    ):

FE20

```
19 FD|00 FD 18 00 00 ...
19 FD FE 20|00 00 00 ...
```

◀◀▶移动到 FE 和 20 之间，输入 1，：

```
19 FD FE|20 00 00 00 ...
19 FD FE 31|20 00 00 ...
19 FD|20 00 20 00 00 ...
```

这就是通过基本溢出得到字符 an 的办法。

FD20

**注意：**数学模式中字符 00 后面的东西都不可见，线性模式中 NUL 后面的东西不可见。由于此处的操作都在字符 00 后面，因此输入的字符和光标都是看不到的，需要同学们做好规划。另外，数学模式下的“撤销”功能只会恢复到第一个 NUL 为止，后面的东西不能恢复，所以同学们一定要小心，按错了是没有补救机会的。

## 2.3.2 删除

接下来讲删除。删除比较简单，从当前字符到下一个字符 00 所有的字符都向左复制即可。有两点需要注意：

第一，如果删除的字符长度>1，删除就会有残留（自己想一想为什么）：

```
31 FD 18|32 33 00 ...
31|32 33 00 33 00 ... //复制[32 33 00]并覆盖了 32，但 33 没覆盖
```

一般来说这不要紧，但它却是后面数学模式字符复制的基础。

第二，和插入不同，删除是以光标右侧第一个字符为起点，一直向右到第一个字符 00 为止，将这一串字符向左复制。因此字符转换器的基本溢出和其他基本溢出在这里体现出区别：

```
31|FE 00 32 33 00 ...
|FE 00 32 33 00 00 ... //FE00 不包含字符[00 FE00 32 33 00]被复制
```

而

```
31|19 00 32 33 00 ...
|19 00 00 32 33 00 ... //19 后面是字符 00，只有 19 和 00 被复制
```

## 2.4 第二章例题

### 1. 用框得到字符@。

18

**参考思路：**我们知道通过基本溢出可以覆盖掉双字节字符的前一个字节。在框上按  
 ➤进入基本溢出，输入字符Ran#。

FD18

```
19 00 FD 18|00 ...
```

◀，输入字符“1”：

```
19 00 31|18 00 ...
```

这样 FD 就被覆盖了。然后两次 DEL，删掉 31 和 00，离开基本溢出，就得到了字符  
 @。  
 18

### 2. 用框得到A到F。

3A 3F

**参考思路：**思路和上一个一样，只要覆盖就行。在框上按 ➤进入基本溢出，输入 5  
 个 1：

```
19 00 00 31 31 31 31|00 00 00 ...
```

输入 R (SHIFT 7 3 X)，然后 ◀ ◀：  
 FD3F

```
19 00 00 00 31 31 31 FD 3F|00 ...
19 00 00 00 31 31|31 FD 3F 00 ...
```

输入 A (SHIFT 7 3 ' '), 覆盖掉 31FD，然后 ◀ ◀：  
 FD3E

```
19 00 00 00 31 31 FD 3E|3F 00 ...
19 00 00 00 31|31 FD 3E 3F 00 ...
```

如此重复，直到输入 r (SHIFT 7 3 7)，然后按一次 ◀：  
 FD3A

```
19 00 FD 3A|3B 3C 3D 3E 3F 00 ...
19 00|FD 3A 3B 3C 3D 3E 3F 00 ...
```

现在输入 1 覆盖掉 FD，然后 **DEL** 两次：

```
19 00 31|3A 3B 3C 3D 3E 3F 00 ...
19|3A 3B 3C 3D 3E 3F 00 00 00 ...
```

就得到了 **A** 到 **F**。回忆一下上一章讲的字符转换器，有一道例题也是这个，相比之下

下，基本溢出的按键数比字符转换器少一些。因此基本溢出适合用来打大量字符，而少量字符用字符转换器更方便。

在长 200 字节的输入区后面，紧接着还有一个同样长度的回放区，上一章提到过，按 **≡**, **CALC**, **SHIFT** **CALC** 或 **STO** 进行一次计算（即使报错了也没关系）会把输入区的内容复制到回放区（到 NUL 为止），**AC** 后按 **◀** 或 **▶** 会把回放区重新复制到输入区。一般来说回放区不能直接接触到，因为当输入区长度达到 199 字节时，就不能再输入字符。然而，基本溢出时输入的字符因为在 NUL 后面，不计入长度，所以可以通过基本溢出来使输入长度大于 199 字节，从而摸到回放区。这可以用来复制字符。例如，刷出框后，在框前输入任意式子，按 **≡** 把式子复制到回放区（为了方便，用冒号表示输入区的末尾），然后在框前打很多字直到打不下，然后移动到框后面，进入基本溢出：

```
... xx|xx 19 00:xx xx ...
... xx xx 19 00|xx xx ...
```

按 **DEL** 删掉 00：

```
... xx xx 19|xx:xx xx ...
```

把之前垫的字符都删掉，这样就把回放区的字符都“接”到了输入区，完成了一次复制。复制完以后，末尾依然有框，如有需要还能按=继续复制。这也是线性模式中复制字符的主要办法。

### 3. 尝试按上述方法复制字符串 1+1。

**参考结果：**1+1□1+1□（复制结束后最初的框依然存在，如需消除可在该框内输入任何字符然后删除）


**参考思路：**见上文，不再赘述。

## 2.5 附：对回放区进行处理


基本溢出还能对回放区的字符进行后期处理，这是字符转换器做不到的。例如，你有一个字符串 A6 FD 3A A6 FD 3B A6，假设这 **±** 是个不好打的字符，你希望在不干扰 **±** 的

A6

A6

前提下，把 FD3A 和 FD3B 变成 3A 和 3B。做法是，首先准备好字符转换器，放在历史记录里，然后输入那个字符串，按  把它存入回放区。接下来，找到字符转换器，用它刷出框，在它前面打满字（在数学模式，可以打一大堆分数线，会快一点），然后删掉一个字节：

```
19 00|00:A6 FD 3A A6 FD 3B A6 00 ...
```

删掉一个字节是因为：如果不删，字符串的长度就是 199，那就不能再输入新东西了。现在输入 1（不能只按 ，会回到最左边）：

```
19 00 31|A6 FD 3A A6 FD 3B A6 00 ...
```

这样就摸到了回放区，按  跳过需要保留的 A6，输入 1 把 FD 盖掉，再删除 1：

```
19 00 31:A6|FD 3A A6 FD 3B A6 00 ...  
19 00 00:A6 31|3A A6 FD 3B A6 00 ...  
19 00 00:A6|3A A6 FD 3B A6 00 00 ...
```

以此类推，把下一个 FD 也删掉：

```
19 00 00:A6 3A A6|3B A6 00 00 00 ...
```

按  ，回放区的内容就会复制到输入区，就得到了想要的 A6 3A A6 3B A6。

各位思考一下，同样是基本溢出，这里为什么绕了一步，先把字符转换器变成框才能做呢？

原因是，刚才讲删除的时候说过，如果直接用字符转换器的话，打满字以后删掉一个，会使得后面的字符全都往前移：

```
xx|FE 00:A6 FD 3A A6 FD 3B A6 00 ...  
|FE 00 A6:FD 3A A6 FD 3B A6 00 00 ...
```

其实，打满字再删一个只是为了让字符串的长度正好为 198。仔细观察可以发现，长度达到 190 时，光标会变得很粗，提示你即将打满。如果用字符转换器，垫到光标变粗时开始数字符，正好数到 198，那也可以。（注：如果显示出来的字符串超过 200 字节很多，在数学模式中有时会直接崩溃死机，所以保险起见还是建议用框）

## 第三章 变量与历史记录

### 3.1 变量格式

#### 3.1.1 变量在内存中的位置

我们知道，0xD180 开始的 0xC8(200)个字节是输入区。紧接着，从 0xD248 开始的 0xC8(200)个字节为回放区。接下来，从 0xD310 开始有 10 个字节的随机数种子，0xD31A 开始就是变量存储区。

变量按照字符表的顺序连续存储，每个变量占 10 字节。比如，0xD31A~0xD324 为变量 **M**，0xD324~0xD32E 为变量 **Ans** 等，以此类推。

**注意：**字符表上一共有 16 个“变量”，但是其中只有前 12 个是真的变量（分别为：**M Ans A B C D E F x y PreAns @**），后面 4 个@占有的存储空间与即将讲到的历史记录相重合。

#### 3.1.2 变量的储存格式

一个变量 10 字节的储存空间中，第一个字节的前面一半表示了后面九个半字节的类型。在这里我们只需要掌握最常用的浮点数类型即可。对于浮点数类型，这半个字节为 0。从第一个字节的后面一半开始，有七个半字节，表示这个浮点数的 15 位有效数字，每半个字节表示一位，是按十进制直接存储的。最后还有两个字节。倒数第二个字节是指数，负指数用反码表示。最后一个字节的前一半固定为 0，后一半表示这个浮点数整体和指数部分的正负。0 表示整体正指数负，1 表示整体正指数正，5 表示整体负指数负，6 表示整体负指数正。

以这个数为例：

```
06 89 47 57 00 00 00 00 00 01
```

第一个字节的前面一半告诉我们，这是一个浮点数。接下来七个半字节是有效数字：

```
06 89 47 57 00 00 00 00 00 01
```

第九个字节 00 表示指数。最后一个字节 01 表示这个浮点数整体是正的，指数也是正的。所以这个数就是+6.894757000000000E+00，也就是 6.894757，大家对这个数一定很熟悉了。

再以这个数为例：



```
07 46 42 68 41 12 85 46 82 05
```


与刚才一样，有效数字为

07 46 42 68 41 12 85 46 82 05

不同的是，最后一个字节 05 表示整体是负的，指数也是负的。反码就是用 100 去减， $82-100=-18$ ，所以指数就是-18。这整个数是-7.46426841128546E-18。




### 3.1.3 合法变量与重置

在每次按下开机键  的时候，计算器会检查变量是否合法。如果某个变量第一个字节的后半在 0~9 之间，而且最后一个字节以 0 开头，那么它就是合法的。开机时，计算器会挨个检查前 10 个变量，如果遇到了不合法的变量，就会进行完全初始化（初始化的等级与自检相同，都不保留语言设置或对比度）。比如，如果我们卡出字符  并按下等于，

此时变量 **Ans** 就会设为 ，也就是

0A 00 00 00 00 00 00 00 00 01

按下开机，因为变量 **Ans** 的第一个字节的后半是 A，大于 9，不合法，所以就会重置。同样的，在位移较小的 **an** 模式中，你会发现有些变量的值是异常的，此时按开机，多半就会触发重置。

如果用  手动清空存储器，只有前 10 个变量会清空。后面的 **PreAns**、，还有后面 4 个，都不会受到影响。

## 3.2 历史记录格式

### 3.2.1 历史记录在内存中的位置

紧跟着 12 个变量就是历史记录区。因此历史记录区的起始与“变量”4C 重合。

### 3.2.2 历史记录的储存格式

每条历史记录分为 3 个部分。第一部分长 3 字节，主要是这条历史记录的储存模式（例如是否有虚部等等）。第二部分长 10 或 20 字节，包含了 1 或 2 个数值，表示这条历史记录的“答案”。如果答案是虚数，那就有 2 个数值，第一个是实部，第二个是虚部。类似的，计算余数、Pol / Rec 互化等，都会产生两个数值。如果答案是个一般的实数，那么数值就只有一个。第三部分是算式，长度不定。这一部分以字节 23（对应的单字节字符

为“ $\frac{1}{2}$ ”) 结尾。注意，如果在读到字节 23 之前先读到了字节 00，那这条记录就是“错误”的，计算器也不会尝试读取下一条历史记录。

如果你还记得的话，在第一章我曾经提到，计算  $1 \text{ lbf/in}^2 \cdot \text{kPa}$  会导致字符

FE23

FE23 的 23 被吃掉。这是因为 23 被当成了这条历史记录的结尾。在  $1 \text{ lbf/in}^2 \cdot \text{kPa}$  后

FE23

面再多放几个字符，它们就会被当成下一条历史记录。如果这些字符的长度不到 13 字节，那么后面这条历史记录就是无效的，但是如果字符长度超过 13 字节，超过的部分就会进入下一条历史记录当中。

## 3.3 变量和历史记录的重叠

### 3.3.1 通过变量修改历史记录

如前所述，“变量” $\text{4C 4D 4E 4F}$  的位置和历史记录相重叠，可以通过给这些变量赋值

4C 4D 4E 4F

来修改历史记录。有一种广为人知的刷字符方法：刷出字符  $\text{4D}$ ，然后打  $\text{@=1.0000}$

4D

(编码)  $23:\text{x}$ ，其中编码是要刷的字符对应的粗体十六进制编码，要刷字符  $\text{81}$  就打

81

81，刷  $\text{B3}$  就打 B3 等等。最后  $\text{CALC AC}$ ，最上面一条历史记录就是想要卡的字符。

B3

现在我们来探究这种方法的原理。假设我们经济困难，要刷四个字符  $\text{24}$ 。那么算式即

24

为  $\text{@=1.00002424242423:\text{x}}$ 。一般来说，在刷字符之前，我们会用一下字符转换器，因此历史记录里已经有了一条：

```
xx xx xx:06 89 47 57 00 00 00.00 00 01:31 FE 23|23 ...
```

接下来的操作会给变量  $\text{4D}$  赋值。我们知道，变量  $\text{4C}$  与历史记录的开头对齐，那么变

4D

4C

量  $\text{4D}$  就位于历史记录开始 10 字节以后，也就是加了点的那个位置。变量  $\text{4D}$  会覆盖掉原有

4D

4D

的历史记录：

```
xx xx xx:06 89 47 57 00 00 00.01 00 00:24 24 24 24 23|00 01 ...
```

现在我们翻到最上面，计算器就会把第一条历史记录理解为，答案是

```
06 89 47 57 00 00 00 01 00 00
```

也就是 6.89475700000001（由于精度问题，只会显示成 6.894757），而算式是



24 24 24 24

也就是我们想要刷的四个  $\$$ 。注意，最后一个 23 是必须要加的，用来分隔历史记录。  
24

这种办法显然效率低下，一次只能刷 4 个字符，如果我需要更多的字符怎么办呢？回去看一下前面讲的浮点数格式，你会发现，浮点数的倒数第二个字节是指数，也可以利用一下。有一种显而易见的改进方法，就是把刚才打的 1.00002424242423 改为 1.00002424242424  $\times 10$  23。这样就利用了指数位，可以多打一个字。

单个  $\textcircled{D}$  基本上只能做到这样了。但是为什么只用  $\textcircled{D}$  呢？可以把  $\textcircled{D}$ 、 $\textcircled{E}$ 、 $\textcircled{F}$  三个变量拼起来一块用。有一个在虚数模式下刷 38 字的方法：

1. 每个浮点数可以放 8 字节的数据，格式为 1. (前 7 字节)  $\times 10$  (最后一个字节)；
2. 先按照顺序给变量赋值：D=1.0000 (6 字节)，M=1. (8 字节)，F=1. (8 字节)， $x=1. (8 \text{ 字节})+1. (8 \text{ 字节}) i$ ；
3. 切换模式以清空历史记录，刷出  $\textcircled{D}$   $\textcircled{E}$ ， $\text{[CALC]}$ ，把两个  $\textcircled{D}$  都赋值为 0，然后  
4D 4F

$\text{[AC]}$   $\text{[◀]}$ ，改为这个式子：

111:((M)):x:@=D:@=F

4.  $\text{[CALC]}$   $\text{[≡]}$   $\text{[≡]}$   $\text{[≡]}$   $\text{[≡]}$   $\text{[≡]}$ ，向上翻到第一条历史记录

(注：在线性复数 **an** 模式中刷字符的方法略有不同，见本章末)

这种方法的原理又是什么呢？首先为了刷  $\textcircled{D}$  和  $\textcircled{E}$ ，我们肯定用到了字符转换器，因  
4D 4F

此现在的历史记录区就是：

xx xx xx:06 89 47 57 00 00 00 00 00 01:31 FE 23|23 ...

然后变量  $\textcircled{D}$  被设成 0 了，这覆盖了唯一的一条历史记录，使其失效，现在就没有历史记录了：

xx xx xx 06 89 47 57 00 00 00 \*\*00 00 00 00 00 00 00 00 00\*\* ...

接下来的 111 只是用来“垫”的，占了 17 个字节的空间：

xx xx xx:01 11 00 00 00 00 00 00 02 01:31 31 31 23|00 00 00 ...

((M)) 就是 M，它把变量 M 中存储的数值放在历史记录里面：

xx xx xx:01 11 00 00 00 00 00 00 02 01:31 31 31 23|xx xx xx:01 mm mm

```
mm mm mm mm mm mm 01:60 60 40 D0 D0 23| ...
```

又填充了 $x$ :

```
xx xx xx:01 11 00 00 00 00 00 02 01:31 31 31 23|xx xx xx:01 mm mm
mm mm mm mm mm mm 01:60 60 40 D0 D0 23|xx xx xx:01 xx xx xx xx xx
xx xx 01 01 xx xx xx xx xx xx xx 01:48 23| ...
```

注意 $x$ 是虚数，因此虚部也包含在里面，答案区占了 20 个字节。然后我们给  $\textcircled{D}$  和  $\textcircled{F}$   
4D 4F

赋值：

```
xx xx xx:01 11 00 00 00 00 00.01 00 00:dd dd dd dd dd dd 01.01 mm mm
mm mm mm mm mm mm 01.01 ff ff ff ff ff ff ff 01.01 xx xx xx xx xx
xx xx xx 01 01 xx xx xx xx xx xx xx 01 48 23| ...
```

此时你会发现， $D$ 、 $M$ 、 $F$ 、 $x$ 都连在一起了，而且都位于第一条历史记录的计算区。我们只要向上翻到第一条记录，就能看到要刷的字符，多达 38 个！

但是，这种方法还是有一些限制。第一，它只能在虚数模式用。（计算模式里可以去掉 $x$ 的虚部，可是那样就只能刷 30 个字符了。）第二，如果一个浮点数的最后几位有效数字中包含粗体十六进制字母，它可能会触发进位，导致刷字符失败。而且，指数位也不能包含粗体十六进制字母。（其实所有通过这个原理刷字符的方法都有这个缺点。）怎么办呢？

### 3.3.2 通过历史记录填充变量

我们发现，一个变量占了 10 字节，但是我们只用到了其中的 8 个字节。当然，这是因为我们把变量作为浮点数来赋值。有没有办法直接给变量“填充”值呢？当然也是有的。假设你要给一个变量填充 10 字节的值。首先你要做一个准备工作。刷出字符  $\textcircled{r}$ ，  
FD13

$\textcircled{=}$   $\textcircled{SD}$   $\textcircled{x}$ 。这是一个白值，显示不出来，计算它的零次方会得到 **ERROR**。这个 **ERROR** 和数学错误不一样，它会使计算器停止运算接下来的式子，不会报错。有了它，就能把任何东西放进历史记录了。

接下来，刷出你想要的 10 字节（可以通过  $\textcircled{D}$   $\textcircled{D}$  连续赋值的办法），记得刷的时候—  
4D 4E

并刷一个  $\textcircled{D}$ 。  
4D


翻到最上面一条历史记录，先直接  $\textcircled{CALC}$ ，把  $\textcircled{D}$  赋值为 0，这样就清掉了第一条历史记  
4D

录。然后输入如下算式：

$x^0++$ 要刷的 10 字节




为什么要这样做呢？其实，数学输入中的次方符号长 4 个字节，因此你输入的式子在内存里长这样：

```
48 C9 1A 30 1B A6 A6 十字节...
```

按 ，把它读入历史记录中，现在历史记录就是：






```
xx xx xx:F3 00 00 00 00 00 00.00 00 00:48 C9 1A 30 1B A6 A6.十字
节...
```

你会发现那十个字节刚好和变量  $\textcircled{4E}$  对齐了，因此现在变量  $\textcircled{4E}$  就是你想要填充的值！

刷出  $\textcircled{4E}$ ，然后  ，就可以了。（有些时候， 会导致报错，那就只好老老实实用浮点数了，浪费的 2 个字节我们也拿它没办法。）同理，其实可以一次刷 20 个字节，然后用和刚才一模一样的步骤，同时填充  $\textcircled{4E}$  和  $\textcircled{4F}$ ，效率直接翻倍。通过这种方法，在计算模式中理论上一次可以刷出 37 个字符，基本上是用够了。

### 3.3.3 附：线性复数 an 模式中刷字符的方法

1. 进入 an 模式之前，先按照顺序给变量赋值： $D=1.0000$  (6 字节)， $M=1.$  (8 字节)， $F=1.$  (8 字节)， $x=1.$  (8 字节)+ $1.$  (8 字节)  $i$ ；
2. 进入 an 模式，刷出  $\textcircled{4D} \textcircled{4F}$ ，输入这个式子：  

$$111:((M)):x:@=D:@=F;$$
3.     ，向上翻到第一条历史记录。