

# 991CN X

Make 991CNX Great Again

fx-es(ms)编写组

# 前 言

# 目 录

前言	1
第一章 汇编语言	4
1.1 汇编语言简介	4
1.2 常用指令（nX-U16 指令集）	4
第二章 硬件与系统架构	4
2.1 nX-U16 处理器	4
2.2 内存与寄存器	4
2.3 自检与计算器版本号	4
2.4 ROM 概述	4
2.5 矩阵键盘	4
第三章 字符与表达式	5
3.1 单字节与双字节字符	8
3.2 数学显示字符	10
3.3 映射字符	10
第四章 内存分区	10
4.1 输入区，撤销区与回放区	10
4.2 变量区与变量存储格式	10
4.3 历史记录区	10
4.4 设置区	10
4.5 即时状态区	10
4.6 堆栈区	10
4.7 随机数种子	10
4.8 显示缓存区	10

4.9	空闲区 .....	10
4.10	杂项 .....	10
<b>第五章</b>	<b>输入的显示与编辑 .....</b>	<b>10</b>
5.1	输入的显示 .....	11
5.2	输入的编辑 .....	11
<b>第六章</b>	<b>特殊字符 .....</b>	<b>11</b>
6.1	字符转换器 .....	11
6.2	溢出 .....	11
6.3	@法刷字符 .....	12
6.4	字符的特殊性质 .....	12
<b>第七章</b>	<b>返回导向编程 .....</b>	<b>12</b>
7.1	an/@的异常性质 .....	12
7.2	ROP 进入方法 .....	12
7.3	简单 ROP .....	13
7.4	ROP 进阶 .....	13
7.5	ROP 实战 .....	13
<b>第八章</b>	<b>ROM 刷机 .....</b>	<b>14</b>
<b>附录一</b>	<b>字符表 .....</b>	<b>14</b>
<b>附录二</b>	<b>常用汇编指令 .....</b>	<b>14</b>
<b>附录三</b>	<b>按键内码对应表 .....</b>	<b>14</b>
<b>附录四</b>	<b>模拟器的配置与使用 .....</b>	<b>15</b>
<b>附录五</b>	<b>ROM 函数地址索引 .....</b>	<b>15</b>
<b>附录六</b>	<b>用 Ghidra 反编译 ROM .....</b>	<b>15</b>
<b>附录七</b>	<b>Github 上的数字资源仓库 .....</b>	<b>15</b>

结语.....	15
---------	----

## 第一章 汇编语言

### 1.1 汇编语言简介

### 1.2 常用指令（nX-U16 指令集）

## 第二章 硬件与系统架构

### 2.1 nX-U16 处理器

### 2.2 内存与寄存器

### 2.3 自检与计算器版本号

同时按下 **SHIFT** **7** **开机** 键（无论是否开机），你的 991CN X 都应显示为如图所示：

### 2.4 ROM 概述

### 2.5 矩阵键盘

矩阵键盘是一种电子线路，通常用于输入数据或控制电子设备。它由多个行和列的按键组成，形成一个二维排列的键盘结构。每个按键都位于特定的行和列交汇处，这种布局使得使用少量的引脚（通常为行和列的总数之和）就可以控制大量的按键。991CN X 中应用了一个 8 行 7 列的矩阵键盘。

2.5.1 结构

矩阵键盘由多行和多列的按键组成。每个按键都与一个行线（称为 KI）和一个列线（称为 KO）相连（O，I 分别为 Output 和 Input 的缩写）。

注：即使 991CN X 键盘排布外观不是矩形，其电路逻辑也是矩形。

以下是 991CN X 的键盘码对应表：（可以注意到有些节点未设按键）

<div>KO</div> <div>KI</div>	1	2	3	4	5	6	7
8	SHIFT	ALPHA	▲	▶	菜单		
7	OPTN	CALC	◀	▼		$x$	
6		$\sqrt{\square}$	$x^2$	$x^\square$	$\log_{\square}$	ln	
5	$(-)$	$\circ, \text{''}, \text{'''}$	$x^{-1}$	sin	cos	tan	0
4	STO	ENG	(	)	$S \div D$	M+	.
3	7	8	9	DEL	AC		$\times 10^x$
2	4	5	6	$\times$	$\div$		Ans
1	1	2	3	+	-		=

例如： $x^2$  键对应的行线，列线即为 KI6，KO3。

2.5.2 工作原理

1. 扫描行和列

控制器会逐行逐列地扫描整个矩阵。当某个按键被按下时，它所在的行和列之间形成电路连接，这会使得该行，列电平相等。

通常，控制器会通过以下步骤检测按键状态：

- 激活第一列（KO1），将这一列设置为低电平，其余列设置为高电平；
- 按 KI 值由大到小的顺序检查所有行，测试哪一行有信号返回（即低电平），这表示在当前行与该列交叉的按键被按下；
- 重复这个过程，依次激活每一列，直到所有行都被扫描完毕。再

从第一列开始。

## 2. 识别按键

通过识别当前激活的行和有信号的列，控制器就可以确定是哪一个按键被按下了。例如，如果当前激活的是 KO5，且在 KI3 检测到了低电平信号，那么按下的按键就是 KI3,KO5 相交处的键为 **AC** 键。

处理器以非常高的频率来扫描矩阵键盘，以保证我们按下按键时，总是能够及时地检测到。由于处理器的运算速度非常快，因此可以轻易地让我们认为是“实时”的。

### 2.5.3 优先级

在 991CN X 上，若**同时**<sup>1</sup>按下 **1** 键和 **2** 键，则屏幕上应显示 **1**；**同时**按下 **1** 键和 **4** 键，则屏幕上应显示 **4**。同学们不妨自己动手试一试。这是为什么呢？

先说结论：当一个 KI 同时被多个 KO 接通时，只有 KO 最小的有效；当多个 KI 同时被一个 KO 接通时，只有 KI 最大的有效，和 KO 相反。

**简记：先看左右，再看上下，左优于右，上优于下。**

举个例子：当同时有 KO1→KI1（即按下 **1**）和 KO2→KI1（即按下 **2**）时，只有前者有效，因为前者 KO 小。当同时有 KO1→KI1（即按下 **1**）和 KO1→KI2（即按下 **4**）时，只有后者有效，因为后者 KI 大。一定要注意 KO 和 KI 的优先级是相反的。

原因：为了保护电路，在两个完整的扫描周期之间，有一段相对较长（仍然很短<sup>2</sup>）的空闲时间。因此，我们可以认为，当我们按下按键时，处理器总是处在间歇时间当中。开始扫描后，根据 2.5.2 节中的工作原理，键位的扫描顺序为：**【(x,y)指(KIx,KOy)】**

(8,1) (7,1) (6,1) (5,1) (4,1) (3,1) (2,1) (1,1)

(8,2) (7,2) (6,2) (5,2) (4,2) (3,2) (2,2) (1,2)




(8,3) (7,3) (6,3) (5,3) (4,3) (3,3) (2,3) (1,3)

<sup>1</sup> 一般来说，同时不易做到，此时可以在关机状态下按住 **1** 键和 **2** 键（不分先后），再开机，则应显示 **1**。

<sup>2</sup> 一个完整的扫描周期大约为 1μs 的数量级，而间歇时间约为 1ms 的数量级。




(8,4) (7,4) (6,4) (5,4) (4,4) (3,4) (2,4) (1,4)  
(8,5) (7,5) (6,5) (5,5) (4,5) (3,5) (2,5) (1,5)  
(8,6) (7,6) (6,6) (5,6) (4,6) (3,6) (2,6) (1,6)  
(8,7) (7,7) (6,7) (5,7) (4,7) (3,7) (2,7) (1,7)

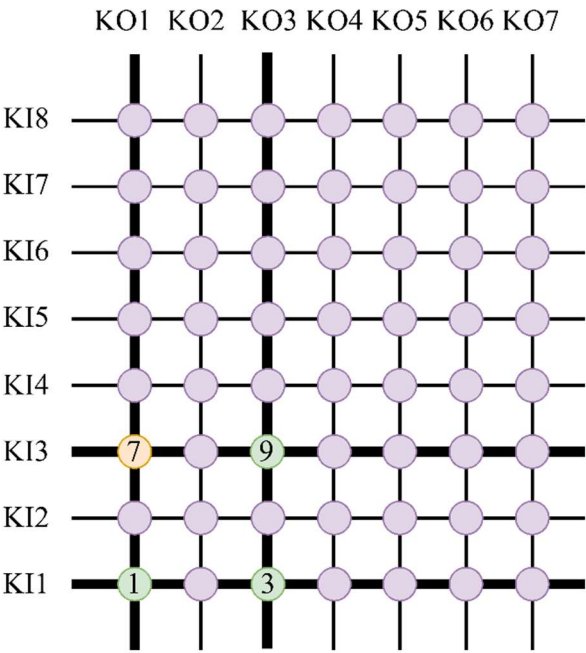
对应键位图，该扫描顺序可简单概括为：**逐列由上到下扫描。**











当扫描到有键被按下后，处理器会直接开始新一轮扫描而非继续。故在上表中，越在前的节点对应的按键优先级越高，比如(KI5,KO2)在(KI3,KO7)的前面。故当同时按下  键和  键时， 的优先级更高。

2.5.4 鬼键问题（四按一松）

在上述的矩阵键盘设计中，如果有多个按键同时被按下，我们有时可以正确检测，有时却会发生错误，将未按下的按键误判为按下。这种现象被称为“鬼键”（Ghost Key）问题。

例如：**同时按下 ，， 键时**（“同时”的概念参考上页注释），则应显示 **7**。



如图所示：当同时按下 ，， 键时， 键连通了 KI1 和 KO1， 键连通了 KI1 和 KO3， 键连通了 KI3 和 KO3，这最终使得 KI3 和 KO1 相连通，造成了  键被按下的假象。由于  键的优先级>，，



9 键，故最终使得计算器认为按下了 7 键。

想一想：同时按下 1，0，=，=，再松开 1，屏幕上会出现什么？（“同时”的概念仍参考之前的注释）

### 2.5.5 自检

在 2.3 节中，我们讲过同时按下 SHIFT 7 开机 键可以进入自检。事实上，进入自检的方式并不止这一种。

进入自检的条件是：按下 开机 （或刚装电池等原因导致开机）时同时存在 KO1→KI3 和 KO1→KI8，且 KO1 的信号不再传入其它的 KI。

想一想：为什么同时按 菜单，AC，7， 开机 也可以进入自检？

## 第三章 字符与表达式

### 3.1 单字节与双字节字符

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，一个字节的数据只有  $16^2 = 256$  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 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。

### 3.2 数学显示字符

### 3.3 映射字符

## 第四章 内存分区

### 4.1 输入区，撤销区与回放区

### 4.2 变量区与变量存储格式

### 4.3 历史记录区

### 4.4 设置区

### 4.5 即时状态区

### 4.6 堆栈区

### 4.7 随机数种子

### 4.8 显示缓存区

### 4.9 空闲区

### 4.10 杂项

## 第五章 输入的显示与编辑

## 5.1 输入的显示

## 5.2 输入的编辑

### 5.2.1 光标移动

### 5.2.2 增删字符

### 5.2.3 撤销

### 5.2.4 插入

### 5.2.5 编辑数学表达式

# 第六章 特殊字符

## 6.1 字符转换器

### 6.1.1 字符转换器原理

### 6.1.2 连续刷字符

### 6.1.3 根号法复制字符

## 6.2 溢出

### 6.2.1 达成溢出的方法

#### 6.2.1.1 方程溢出

6.2.1.2 线性求和溢出

6.2.1.3 双字节溢出

6.2.1.4 框溢出

6.2.2 溢出法刷字符

6.2.3 溢出法复制字符

6.2.4 定向填充法

6.3 @法刷字符

6.3.1 @法刷字符原理

6.3.2 @法的优化

6.3.3 在 `x an` 模式下@法的变化

6.4 字符的特殊性质

## 第七章 返回导向编程

7.1 `an/@`的异常性质

7.2 ROP 进入方法

7.2.1 `x an/@`模式

7.2.2 `an+12char` 法与`@+2char` 法

## 7.3 简单 ROP

### 7.3.1 内存修改器

### 7.3.2 显示字符

### 7.3.3 读取按键

### 7.3.4 循环结构

### 7.3.5 分支结构

## 7.4 ROP 进阶

### 7.4.1 编辑内存

### 7.4.2 计时

### 7.4.3 ROM 提取

### 7.4.4 绘制屏幕

### 7.4.5 调用其他函数

### 7.4.6 quickcpy——快速注入内存

## 7.5 ROP 实战

### 7.5.1 内存注入可视化

7.5.2 像素编辑器

7.5.3 贪吃蛇

7.5.4 俄罗斯方块

7.5.5 1024

7.5.6 函数绘图

第八章 ROM 刷机


























附录一 字符表

此处有 fx-991CN X 硬件版本 Ver C 和 Ver F 的字符表。

附录二 常用汇编指令

附录三 按键，键盘码，内码表

<div>KI</div> <div>KO</div>	1	2	3	4	5	6	7
8	<div>SHIFT</div>	<div>ALPHA</div>	<div>▲</div>	<div>▶</div>	<div>菜单</div>		
7	<div>OPTN</div>	<div>CALC</div>	<div>◀</div>	<div>▼</div>	<div><div>∫</div><div>▢</div></div>	<div>x</div>	
6	<div><div>□</div><div>▢</div></div>	<div><div>√</div><div>▢</div></div>	<div>x<sup>2</sup></div>	<div>x<sup>■</sup></div>	<div>log<sub>▢</sub></div>	<div>ln</div>	
5	<div>(-)</div>	<div><div>0</div><div>999</div></div>	<div>x<sup>-1</sup></div>	<div>sin</div>	<div>cos</div>	<div>tan</div>	<div>0</div>

4							
3							
2							
1							

# 附录四 模拟器的配置与使用

为了更好地研究计算器，我们需要有模拟器。以下是由编写组人员@qiufuyu123制作的991CN X模拟器，源码位于Github上。  
Github地址：[github.com/qiufuyu123/CasioEmuNeo/releases/latest](https://github.com/qiufuyu123/CasioEmuNeo/releases/latest)

# 附录五 ROM 函数地址索引

# 附录六 用 Ghidra 反编译 ROM

# 附录七 Github 上的数字资源仓库

本书编写组在Github上存放了一个数字资源仓库，里面存储了所有本书所需的资源。  
网址：[github.com/Physics365/991CN-X-CW-Decompilation](https://github.com/Physics365/991CN-X-CW-Decompilation)

# 结 语