



# CMS8S6990

## 变量寻址方式注意事项

Rev. 1.00

请注意以下有关CMS知识产权政策

\* 中微半导体（深圳）股份有限公司（以下简称本公司）已申请了专利，享有绝对的合法权益。与本公司MCU或其他产品有关的专利权并未被同意授权使用，任何经由不当手段侵害本公司专利权的公司、组织或个人，本公司将采取一切可能的法律行动，遏止侵权者不当的侵权行为，并追讨本公司因侵权行为所受的损失、或侵权者所得的不法利益。

\* 中微半导体（深圳）股份有限公司的名称和标识都是本公司的注册商标。

\* 本公司保留对规格书中产品在可靠性、功能和设计方面的改进作进一步说明的权利。然而本公司对于规格内容的使用不负责任。文中提到的应用其目的仅仅是用来做说明，本公司不保证和不表示这些应用没有更深入的修改就能适用，也不推荐它的产品使用在会由于故障或其它原因可能会对人身造成危害的地方。本公司的产品不授权适用于救生、维生器件或系统中作为关键器件。本公司拥有不事先通知而修改产品的权利，对于最新的信息，请参考官方网站 [www.mcu.com.cn](http://www.mcu.com.cn)

---

## 目录

<b>CMS8S6990</b> .....	<b>1</b>
<b>1. 概述</b> .....	<b>3</b>
<b>2. 变量存储</b> .....	<b>4</b>
<b>3. KEIL 编译</b> .....	<b>5</b>
<b>4. 样例</b> .....	<b>6</b>
4.1 Data 类型与 Small 模式 .....	6
4.2 Xdata 类型与 Large 模式 .....	6
4.3 Idata 类型 .....	6
4.4 Pdata 类型与 Compact 模式 .....	7
<b>5. 应用注意事项</b> .....	<b>8</b>
5.1 实例说明 .....	8
5.2 实例运行结果 .....	8
5.3 原因分析 .....	10
5.3.1 异常出现的规律 .....	10
5.3.2 汇编代码分析 .....	10
5.3.3 中断功能分析 .....	10
5.3.4 实例应用总结 .....	10
<b>6. 总结</b> .....	<b>12</b>
<b>7. 版本修订说明</b> .....	<b>13</b>

## 1. 概述

CMS8S6990 芯片属于经典 8051 系列，程序中定义的变量可放在内部 RAM 区、外部 RAM (XRAM) 区，具体可分为 data、xdata、idata 以及 pdata，寻址方式分为直接寻址和间接寻址。在编写程序时，需要指定好变量所属于的存储类型，在 Keil 编译环境中 分为了 Small、Compcat、Large 模式，不同的模式编译出的结果也不相同，需对应不同的硬件系统选择合适的编译方式。

## 2. 变量存储

根据 8051 内核等资料说明，以下简单介绍 data、idata、xdata、pdata 的区别。

### 1. data 类型变量

当使用 data 定义变量时，变量会自动存储到内部 RAM（0x00~0x7F）中，此区域还包含了 4 个工作寄存器组（R0~R7）。变量访问模式为直接寻址。

### 2. idata 类型变量

当使用 idata 定义变量时，变量会自动存储到内部 RAM（0x00~0xFF）中。变量访问模式为使用变量地址进行间接访问。

### 3.xdata 类型变量

当使用 xdata 定义变量时，变量会自动存储到外部 RAM（0x00~0xFFFF）中。变量访问模式为使用数据指针寄存器 DPTR 进行间接访问。

### 4.pdata 类型变量

此变量类型暂不推荐使用。

### 3. Keil 编译

Keil 编译选项可分为 3 种模式 Small 模式、Compact 模式、Large 模式，可在 Option->Target 中的 Memory Model 中选择，如下图 1-1 所示。

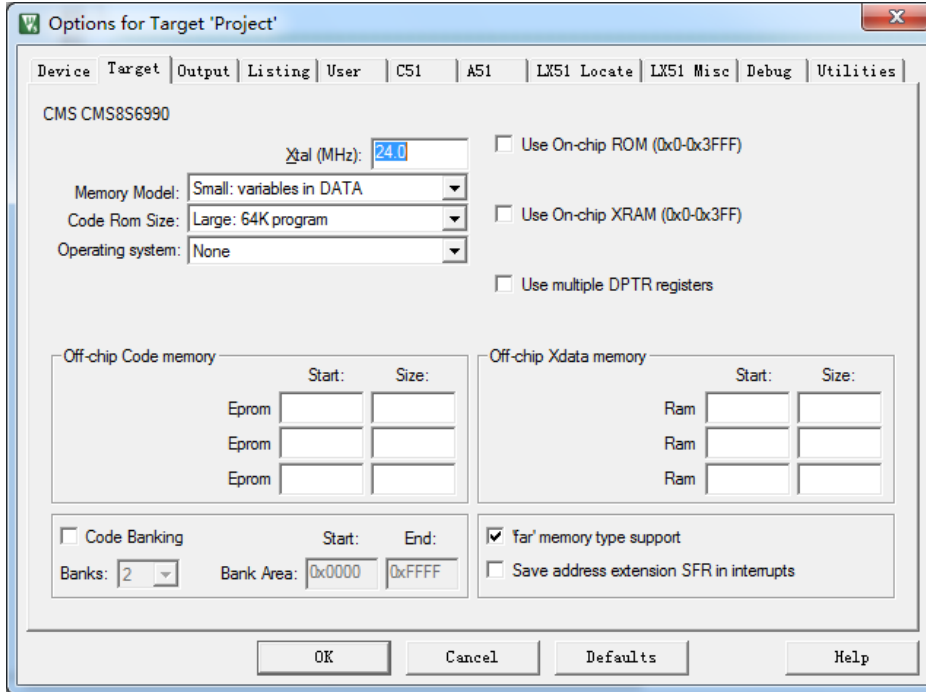


图 1-1： Memory Model 选项

## 4. 样例

为了分析 变量类型、Keil 不同的编译模式对变量访问模式的影响，以下将通过举例的方式加以分析。

### 4.1 Data 类型与 Small 模式

Data 类型与 Small 模式编译时，变量存储在内部 RAM 中。

如：

1. 定义一个 16 位全局变量 Count。
2. 编译后的汇编代码：

```

78: {
79:
80:      Count = Count+1;
⇒C:0x0056  0509  INC      0x09
C:0x0058  E509  MOV      A,0x09
C:0x005A  7002  JNZ      C:005E
C:0x005C  0508  INC      Count(0x08)
81:      while(1);
    
```

可看出 Count 加 1 的代码 实现 方式为直接寻址的方式。

### 4.2 Xdata 类型与 Large 模式

XData 类型与 Large 模式编译时，变量存储在外部 RAM 中。

如：

1. 定义一个 16 位全局变量 Count。
2. 编译后的汇编代码：

```

79:
80:      Count = Count+1;
⇒C:0x01A8  900001  MOV      DPTR,#0x0001
C:0x01AB  E0      MOVX     A,@DPTR
C:0x01AC  04      INC      A
C:0x01AD  F0      MOVX     @DPTR,A
C:0x01AE  7006  JNZ      C:01B6
C:0x01B0  900000  MOV      DPTR,#C_STARTUP(0x0000)
C:0x01B3  E0      MOVX     A,@DPTR
C:0x01B4  04      INC      A
C:0x01B5  F0      MOVX     @DPTR,A
81:      while(1);
    
```

可看出 Count 加 1 的代码 实现 方式为使用 DPTR 寄存器 间接寻址的方式。

### 4.3 Idata 类型

Idata 类型编译后变量存储在内部 RAM 中。

如：

1. 定义一个 16 位全局变量 Count。
2. 编译后的汇编代码：

```

80:
81:      Count = Count+1;
⇒C:0x0086  7809  MOV      R0,#0x09
C:0x0088  06      INC      @R0
C:0x0089  E6      MOV      A,@R0
C:0x008A  18      DEC      R0
C:0x008B  7001  JNZ      C:008E
C:0x008D  06      INC      @R0
82:      while(1);
    
```

可看出 Count 加 1 的代码 实现 方式为使用间接寻址的方式。

## 4.4 Pdata 类型与 Compact 模式

pdata 类型编译后变量存储在外部 RAM 中。

如：

1. 定义一个 16 位全局变量 Count。

2. 编译后的汇编代码：

```
80:
81:          Count = Count+1;
→C:0x01AF  7801      MOV      R0,#0x01
C:0x01B1  E2         MOVX     A,@R0
C:0x01B2  2401      ADD      A,#0x01
C:0x01B4  F2         MOVX     @R0,A
C:0x01B5  18         DEC      R0
C:0x01B6  E2         MOVX     A,@R0
C:0x01B7  3400      ADDC     A,#Count(0x00)
C:0x01B9  F2         MOVX     @R0,A
82:          while(1);
```

可看出 Count 加 1 的代码实现方式为使用间接寻址的方式。

## 5. 应用注意事项

在应用中会经常需要使用中断，并在中断中对变量进行处理。若没有正确选择变量的类型或者编译模式，将会导致一些意想不到的结果。

以下将通过一个实例举例说明变量类型与编译模式对程序的影响。

### 5.1 实例说明

使用 CMS8S6990 芯片编写代码实现以下功能，并使用 Small 模式编译。

- (1) 定义一个 16 位全局变量 Count，初始化为 0；
- (2) 开启定时器 1，设置定时器 1 的溢出时间为 1ms。不开启定时器 1 中断，通过在主程序中查询定时器 1 的溢出位标志实现对变量 Count 的计数，即每隔 1ms 的定时时间，变量 Count 加 1；
- (3) 开启 Timer2 的捕获模式，使用捕获通道 1 捕获信号（每隔 0.5ms 产生一次捕获），在捕获中断中将变量清除。
- (4) 在主程序中判断变量的值大于 10，则判断为异常。

程序如下所示：

```

TMR1_Config();
TMR2_Config();
while(1)
{
    if(TMR_GetOverflowIntFlag(TMR1))
    {
        TMR_ClearOverflowIntFlag(TMR1);
        TH1 =(65536-2000)>>8 ;
        TL1 = 65536-2000;
        P24 = 1;
        Count = Count+1;
        P24 = 0;
    }
    if(Count>10)
    {
        P26 =~P26;
    }
}

/** \brief Timer 2 interrupt service function
**
** \param [in] none
** \return none
**/
void Timer2_IRQHandler(void) interrupt TMR2_VECTOR using 0
{
    P25 = 1;
    Count =0;
    T2IF =0;
    P25 = 0;
}
    
```

### 5.2 实例运行结果

理想的代码运行结果为：永远不会出现变量值大于 10 的情况。但在实际运行中经常出现 Count 大于 10 的情况。

如下图所示：

- (1) 黄色线 代表需要捕获的信号，此信号频率为 3.4Khz。
- (2) 绿色线代表为 Timer1 的 1ms 定时信号（P24）。



(3) 蓝色线为代表 Timer2 的捕获中断信号（上升沿捕获）(P25)。

(4) 红色线为 变量值大于 10 时的输出信号（大于 10 后连续翻转）(P26)。

### 1.正常工作时:

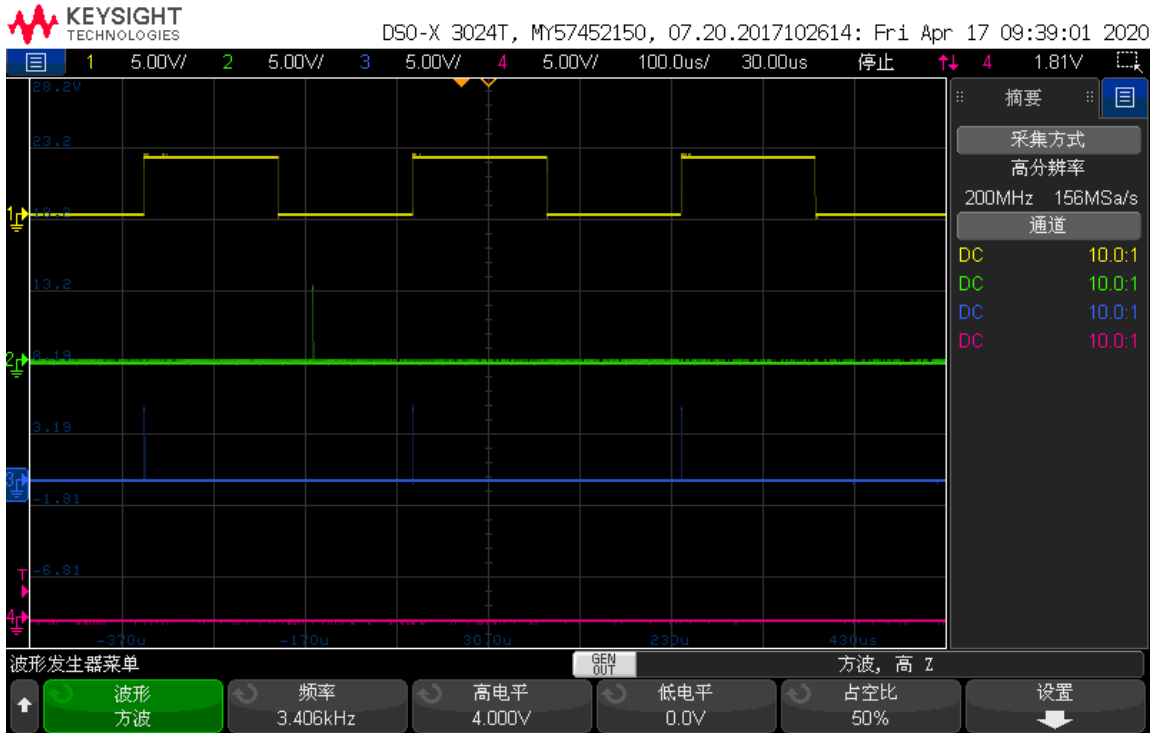


图 1-2: 工作正常

### 2.出现异常时:

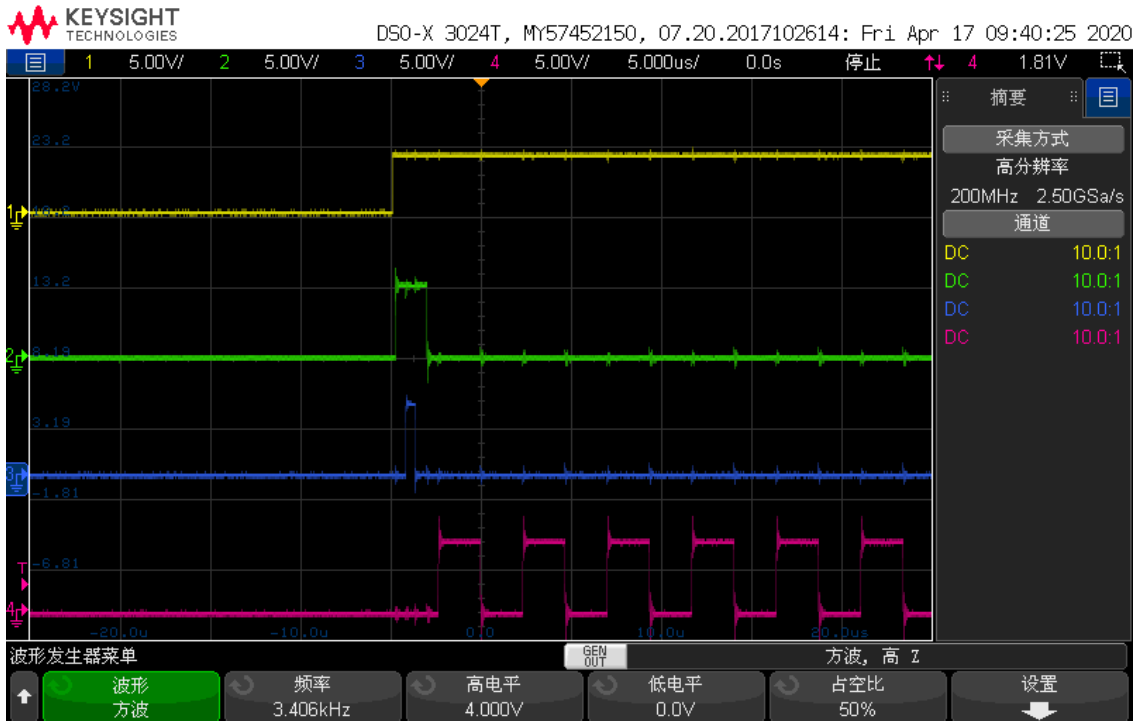


图 1-3: 工作异常

由以上可知 在 Timer1 定时时间内 对变量赋值时, 出现了 Timer2 的捕获中断, 此时出现了异常。通过仿真, 出现变量异常时, 变量的值总是为 0x100;

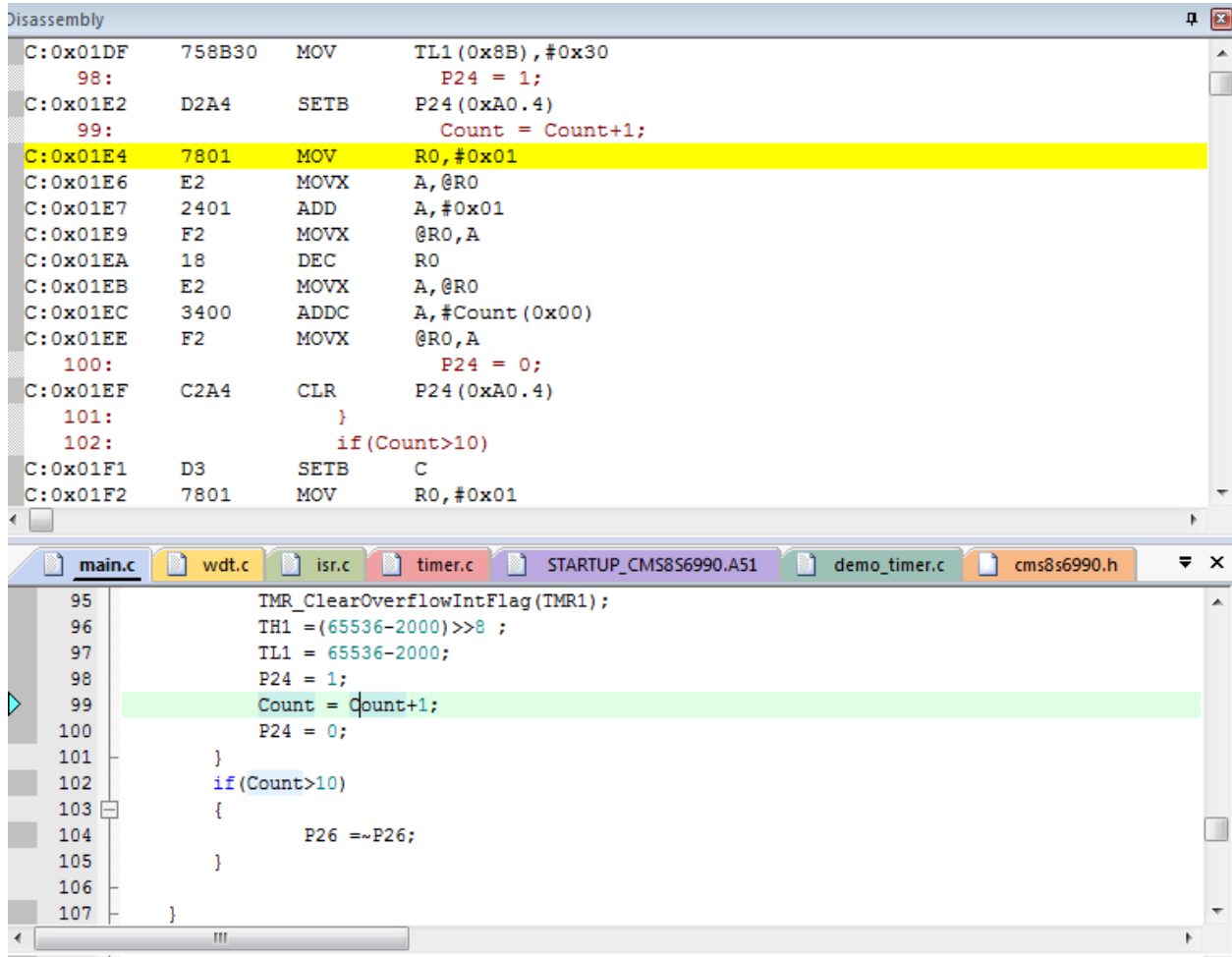
## 5.3 原因分析

### 5.3.1 异常出现的规律

- 1.当异常出现时，总是会出现 Timer2 中断打断了变量的赋值过程。
2. 每次变量出现异常时，变量的值总是等于 0x100。

### 5.3.2 汇编代码分析

汇编代码如下所示：



```

Disassembly
C:0x01DF 758B30 MOV TL1 (0x8B), #0x30
98: P24 = 1;
C:0x01E2 D2A4 SETB P24 (0xA0.4)
99: Count = Count+1;
C:0x01E4 7801 MOV R0, #0x01
C:0x01E6 E2 MOVX A, @R0
C:0x01E7 2401 ADD A, #0x01
C:0x01E9 F2 MOVX @R0, A
C:0x01EA 18 DEC R0
C:0x01EB E2 MOVX A, @R0
C:0x01EC 3400 ADDC A, #Count (0x00)
C:0x01EE F2 MOVX @R0, A
100: P24 = 0;
C:0x01EF C2A4 CLR P24 (0xA0.4)
101: }
102: if (Count>10)
C:0x01F1 D3 SETB C
C:0x01F2 7801 MOV R0, #0x01
    
```

```

main.c wdt.c isr.c timer.c STARTUP_CMS8S6990.A51 demo_timer.c cms8s6990.h
95 TMR_ClearOverflowIntFlag(TMR1);
96 TH1 = (65536-2000)>>8;
97 TL1 = 65536-2000;
98 P24 = 1;
99 Count = Count+1;
100 P24 = 0;
101 }
102 if (Count>10)
103 {
104 P26 =~P26;
105 }
106
107 }
    
```

图 1-4：汇编分析

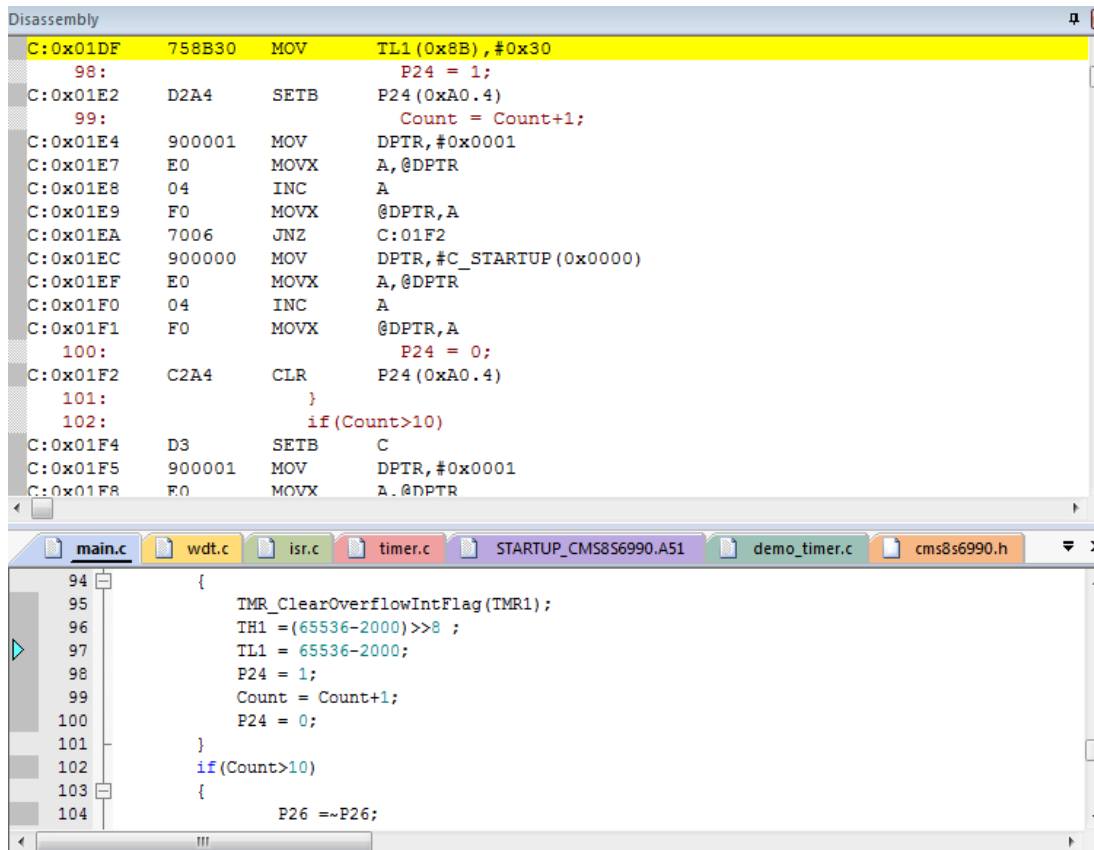
- 1.变量 Count 的 RAM 地址为 0x0C（高位）、0x0D（低位）。
- 2.当变量自加时，首先将 0x0D 地址中的值递增 1。然后将 0x0D 中的值赋值到 A 中，判断 A 的值是否为 0，若为 0，则 0xC（高位）递增 1。

### 5.3.3 中断功能分析

在程序进行赋值的过程中，出现了 Timer2 中断，在中断中清除变量值。当中断返回后，变量值赋值到 A 中，则 A 的值为 0，则变量高位加 1。即变量值为 0x100，变量赋值出现异常。

### 5.3.4 实例应用总结

将上述的程序选择 Large 模式编译，变量使用的是间接寻址的方式。因此中断返回后不会对变量的赋值造成影响。



The screenshot shows a disassembler window at the top and a source code editor at the bottom. The disassembler window displays assembly instructions for a timer interrupt service routine, including setting P24, incrementing a counter, and clearing P24. The source code editor shows the corresponding C code, including comments and conditional logic for the counter.

```

Disassembly
C:0x01DF 758B30 MOV TL1(0x8B),#0x30
98: P24 = 1;
C:0x01E2 D2A4 SETB P24(0xA0.4)
99: Count = Count+1;
C:0x01E4 900001 MOV DPTR,#0x0001
C:0x01E7 E0 MOVX A,@DPTR
C:0x01E8 04 INC A
C:0x01E9 F0 MOVX @DPTR,A
C:0x01EA 7006 JNZ C:01F2
C:0x01EC 900000 MOV DPTR,#C_STARTUP(0x0000)
C:0x01EF E0 MOVX A,@DPTR
C:0x01F0 04 INC A
C:0x01F1 F0 MOVX @DPTR,A
100: P24 = 0;
C:0x01F2 C2A4 CLR P24(0xA0.4)
101: }
102: if(Count>10)
C:0x01F4 D3 SETB C
C:0x01F5 900001 MOV DPTR,#0x0001
C:0x01F8 F0 MOVX A.@DPTR

main.c wdt.c isr.c timer.c STARTUP_CMS8S6990.A51 demo_timer.c cms8s6990.h
94 {
95 TMR_ClearOverflowIntFlag(TMR1);
96 TH1 =(65536-2000)>>8 ;
97 TL1 = 65536-2000;
98 P24 = 1;
99 Count = Count+1;
100 P24 = 0;
101 }
102 if(Count>10)
103 {
104 P26 =~P26;
    
```

图 1-5: Large 模式下的汇编程序

因此，根据不同的程序应用选择 合适的编译模式是十分重要的。

## 6. 总结

在有中断的情况下使用全局变量时，建议使用间接寻址方式访问变量。

## 7. 版本修订说明

版本号	时间	修改内容
V1.00	2020 年 4 月	初始版本