

第3章 单片机程序结构和设计

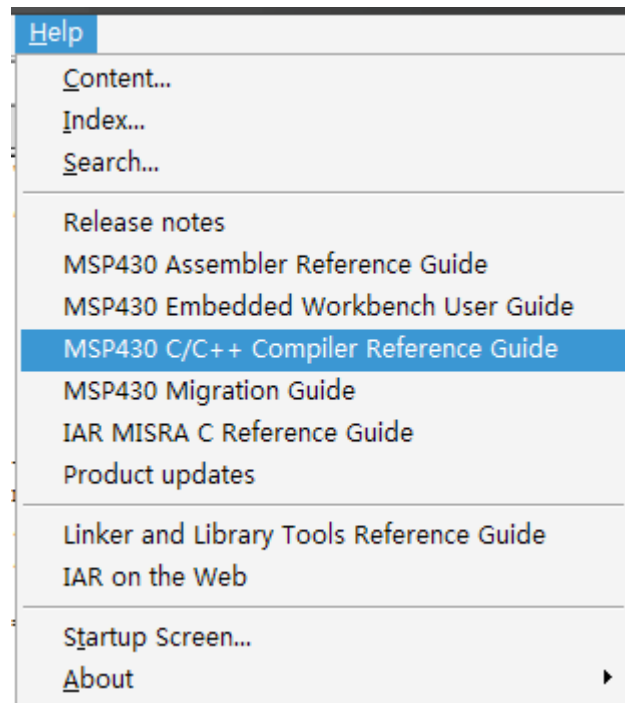
第1节 单片机 C语言程序结构

第2节 单片机 C语言

端口寄存器定义及编程控制

第3节 单片机 C语言程序设计举例

用EW430下的Help



或查看安装目录
\\430\doc\EW430_
CompilerReference.pdf

微机控制台方式下用C语言编写的“Hello, World!”

```
#include <stdio.h>  
  
main( )  
  
{  
  
    printf(“Hello, World !\n”);  
  
}
```

通过调用C语言的库函数实现屏幕输出

微机制台方式下用80x86汇编语言编写的“Hello, World!”

```
data    SEGMENT
string  DB 'Hello, World!', '$' ;定义显示的字符串
data    ENDS

code    SEGMENT
        ASSUME  CS:code, DS:data
start:  MOV     AX, data
        MOV     DS, ax
        MOV     AH, 9           ;调用字符串显示功能
        LEA    DX, string
        INT     21h
        MOV     AH, 4ch        ;返回DOS
        INT     21h
code    ENDS

        END     start
```

通过调用DOS系统功能实现屏幕输出

第一节 C语言程序结构

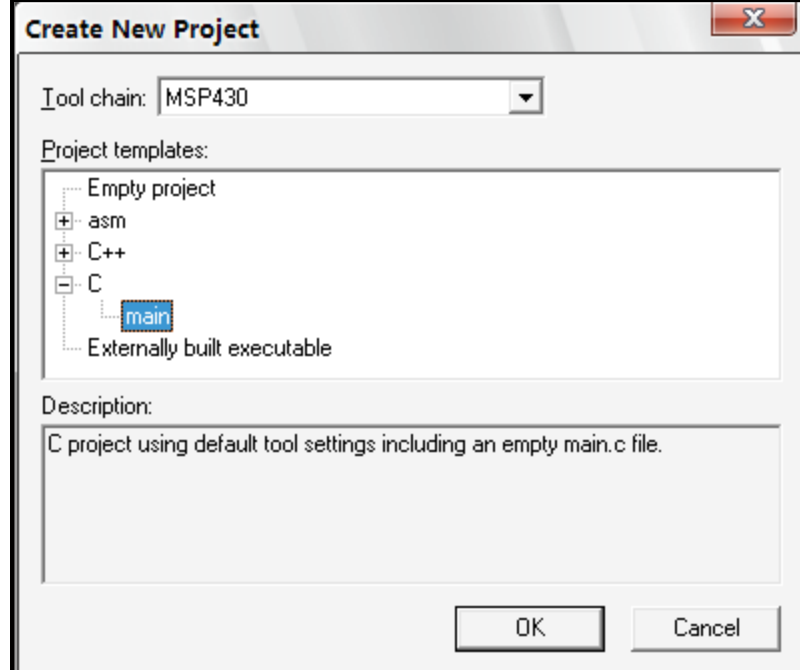
与汇编语言相比，C语言的优点：

1. 对单片机的指令系统不要求了解，对存储器结构简单了解即可；
2. CPU内寄存器的分配、不同存储器的寻址及数据类型等细节可由编译器管理；
3. 程序由函数构成，程序结构化；
4. 可调用系统提供的许多标准子函数；
5. 编程及调试时间缩短，效率提高；
6. 移植性比较好

C语言上机过程

- 一. 创建一个C语言项目
(Creat a New Project/C/main)
- 二. 设置项目属性(Options...)
(MCU类型、Debugger类型等)
- 三. 编写源程序并添加到项目中
- 四. 编译和连接
(compile、make 或Rebuild All)
- 五. 下载程序到目标MCU(Debug)
- 六. 调试和运行(Go、 Step Over、 View/Registers、 memory)

四、五、六中出现有问题， 返回二、三步骤



IAR EW430下msp430的C程序模板

```
#include "msp430F6638.h"

int main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;

    用户程序

    return 0;
}
```

IAR的C语言程序结构与一般C程序基本相同

```
#include "msp430F6638.h" //包含头文件
//常量型存储器定义(在ROM区)
const unsigned char LEDtab[8]={1,2,4,8,0x10,0x20,0x40,0x80};
//带初始化全局变量定义(在RAM区)
unsigned char STD[45]="Hi , this is MSP430.";
//未初始化全局变量定义(在RAM区)
int a, b, c;
//函数声明
void subN(形式参数);
//函数定义
void main ( void ) //用户程序入口函数
{
    数据说明部分; //局部变量定义
    执行语句部分;
    subN(); //函数调用
    .....
}
void subN (形式参数) //子函数
{
    数据说明部分; //局部变量定义
    执行语句部分;
    .....
}
```

程序格式

- 程序由函数构成, 函数由语句组成
- 语句以分号“;”作为结束符, 注意不是分隔符
- 程序由**主函数main()**作为程序入口,
用户程序从函数main() 的第一条语句开始执行,
程序执行完毕的标志是函数main()中的代码执行完毕
- 标识符区分大小写,
不能在变量名、函数名、关键字中插入空格和空行
- **关键字及编译预处理命令用小写字母书写**
- 程序用大括号{ }表示程序的层次范围
- 程序没有行的概念, 可任意书写, 但为了可读性,
书写一对大括号时根据层次采用缩格和列向对齐方式。
- 注释部分用/*.....*/表示
或用“//”表示其后的内容为注释

主程序含两部分：**初始化部分**和**执行部分**；
其中**执行部分**一般为一个无限循环。

test.c (采用**for(;;)**)

```
#include "msp430F6638.h"
int main ( void )
{
    unsigned int i;           //定义函数变量
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    P2OUT=0;                  //端口2输出初始值的设置
    P2DIR=0xFF;              //设置端口2为输出方向
    for ( ;; )                //无限循环
    {
        P2OUT=~P2OUT;        //将端口2的值取反后输出
        for (i=0;i<0xffff;i++); //延时
    };
}
```

主程序含两部分：**初始化部分**和**执行部分**；
其中**执行部分**一般为一个**无限循环**。

test_c.c (采用**while(1)**)

```
#include "msp430F6638.h"
int main ( void )
{
    unsigned int i;           //定义函数变量
    WDTCTL = WDTPW + WDTLHOLD; //关闭看门狗
    P2OUT=0;                  //端口2输出初始值的设置
    P2DIR=0xFF;              //设置端口2为输出方向
        while( 1 )           //无限循环
    {
        P2OUT = ~P2OUT; //将端口2的值取反后输出
        for ( i=0 ; i<0xffff ;i++); //延时
    };
}
```

子函数的声明、定义和调用

test_c.c (采用while(1)和延时函数)

```
#include "msp430F6638.h"
void delay();
int main ( void )
{
    WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
    P2OUT=0; //端口2输出初始值的设置
    P2DIR=0xFF; //设置端口2为输出方向
    while(1) //无限循环
    {
        P2OUT = ~P2OUT; //将端口2的值取反后输出
        delay(); //延时
    };
}

void delay()
{
    unsigned int i; //定义函数变量
    for ( i=0; i<0xffff; i++ ); //延时
}
```

第2节 单片机 C语言端口寄存器定义及编程控制

一、端口寄存器的定义

二、对端口寄存器Px编程控制

一、端口寄存器的定义

- 为方便操作MCU的外围功能模块中的I/O寄存器，
CCS对每一款msp430都提供了相应的头文件msp430Fxxxx.h，
其中msp430Fxxxx为单片机型号
- 在msp430Fxxxx.h头文件中对单片机内的各I/O端口及其各位
按照名称进行定义

在msp430x663x.h中定义了各端口寄存器，含符号和数据类型

```
/* External references resolved by a device-specific linker command file */
#define SFR_8BIT(address) extern volatile unsigned char address
#define SFR_16BIT(address) extern volatile unsigned int address

.....
SFR_8BIT(PAIN_L);          /* Port A Input */
SFR_8BIT(PAIN_H);          /* Port A Input */
SFR_8BIT(PAOUT_L);        /* Port A Output */
SFR_8BIT(PAOUT_H);        /* Port A Output */
SFR_8BIT(PADIR_L);        /* Port A Direction */
SFR_8BIT(PADIR_H);        /* Port A Direction */
SFR_8BIT(PAREN_L);        /* Port A Resistor Enable */
SFR_8BIT(PAREN_H);        /* Port A Resistor Enable */
SFR_8BIT(PADS_L);         /* Port A Drive Strenght */
SFR_8BIT(PADS_H);         /* Port A Drive Strenght */
SFR_8BIT(PASEL_L);        /* Port A Selection */
SFR_8BIT(PASEL_H);        /* Port A Selection */

.....
#define P1IN                (PAIN_L) /* Port 1 Input */
#define P1OUT                (PAOUT_L) /* Port 1 Output */
#define P1DIR                (PADIR_L) /* Port 1 Direction */
#define P1REN                (PAREN_L) /* Port 1 Resistor Enable */
#define P1DS                 (PADS_L) /* Port 1 Drive Strenght */
#define P1SEL                (PASEL_L) /* Port 1 Selection */
```

在msp430F6638.cmd中定义各端口寄存器的地址

```
/**
 *DIGITAL I/O
 ***/
PAIN_L      = 0x0200;
PAIN_H      = 0x0201;
PAOUT_L     = 0x0202;
PAOUT_H     = 0x0203;
PADIR_L     = 0x0204;
PADIR_H     = 0x0205;
PAREN_L     = 0x0206;
PAREN_H     = 0x0207;
PADS_L      = 0x0208;
PADS_H      = 0x0209;
PASEL_L     = 0x020A;
PASEL_H     = 0x020B;
```

二. 对端口寄存器的编程控制

对端口进行输出操作:

//置P1为i/o功能

P1SEL=0;

//置P1为输出方向

P1DIR=0xFF;

//置P1为高强度输出 (可选)

P1DR=0xFF;

//将变量tempout的值通过P1的引脚输出

P1OUT=tempout;

对端口进行输入操作:

//置P1为i/o功能

P1SEL=0;

//置P1为输入方向

P1DIR=0;

//使能P1的上/下拉电阻(可选)

P1REN=0xFF;

//置P1为上拉电阻(可选)

P1OUT=0xFF;

//从P1的引脚输入状态到变量tempin

tempin=P1IN;

为编程方便，
在msp430x663x.h中根据各寄存器各位的作用，
用符号对某位或多位进行了定义，
以便采用位与、位或等方式，对寄存器的各位进行复位或置位。

```
#define BIT0    (0x0001)
#define BIT1    (0x0002)
#define BIT2    (0x0004)
#define BIT3    (0x0008)
#define BIT4    (0x0010)
#define BIT5    (0x0020)
#define BIT6    (0x0040)
#define BIT7    (0x0080)
#define BIT8    (0x0100)
#define BIT9    (0x0200)
#define BITA    (0x0400)
#define BITB    (0x0800)
#define BITC    (0x1000)
#define BITD    (0x2000)
#define BITE    (0x4000)
#define BITF    (0x8000)
```

对端口的某位进行输出操作：

//置P1.1为i/o功能

```
P1SEL&=~BIT1;
```

//置P1为输出方向

```
P1DIR|=BIT1;
```

//置P1为高强度输出 (可选)

```
P1DR|=BIT1;
```

//置P1.1引脚输出高电平1

```
P1OUT|=BIT1;
```

//置P1.1引脚输出低电平0

```
P1OUT&=~BIT1;
```

```
#define BIT0    (0x0001)
#define BIT1    (0x0002)
#define BIT2    (0x0004)
#define BIT3    (0x0008)
#define BIT4    (0x0010)
#define BIT5    (0x0020)
#define BIT6    (0x0040)
#define BIT7    (0x0080)
#define BIT8    (0x0100)
#define BIT9    (0x0200)
#define BITA    (0x0400)
#define BITB    (0x0800)
#define BITC    (0x1000)
#define BITD    (0x2000)
#define BITE    (0x4000)
#define BITF    (0x8000)
```

对端口的多位进行输入操作:

//置P2.3、P2.2为i/o功能

```
P2SEL&=~(BIT3+BIT2);
```

//置P2为输入方向

```
P2DIR&=~(BIT3+BIT2);
```

//使能P2.3、P2.2的上/下拉电阻(可选)

```
P2REN|=BIT3+BIT2;
```

//置P2.3、P2.2为上拉电阻(可选)

```
P2OUT|=BIT3+BIT2;
```

//判断引脚P2.3的状态是0:

```
if ( (P2IN&BIT3)==0) { .....}
```

//判断引脚P2.3的状态是1:

```
if ( (P2IN&BIT3)!=0) { .....}
```

三、C语言程序设计

一) 概述

二) 举例

例1. 节日彩灯的控制

例2. 数码管的工作原理和显示控制

例3. 用按键控制节日彩灯显示的变化

例4. 用按键控制数码管的显示

例5. 数据接收与处理

例6. 检测RAM

例7. 带初始化的变量实现

一) 概述

1. 编写C语言程序步骤
2. 判断程序质量的标准
3. 几种程序结构

1. 编写程序步骤

- 分析实际问题，抽象描述问题的模型
- 确定解决模型的算法
- 按算法画出程序流程图
- 按流程图编写程序
- 上机调试、运行程序

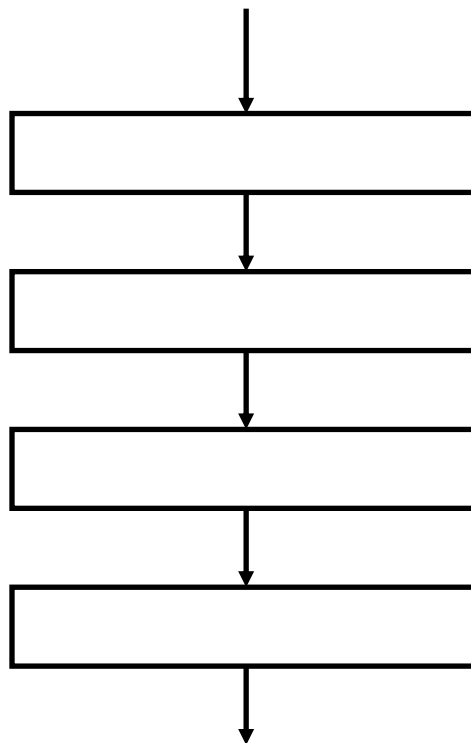
2. 判断程序质量的标准

- 程序的正确性
- 程序的可读性
- 程序的执行时间
- 程序所占内存大小

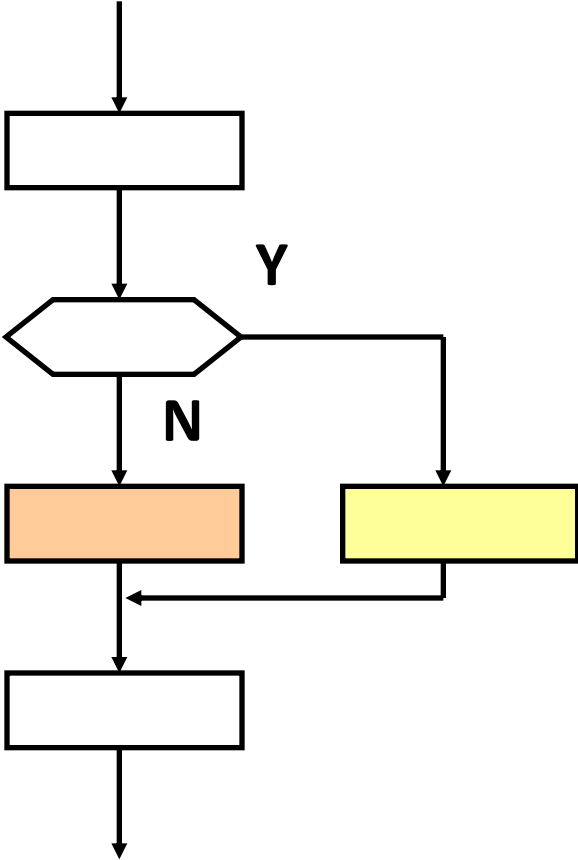
3. 几种程序结构

- 顺序结构
- 分支结构
- 循环结构
- 子程结构

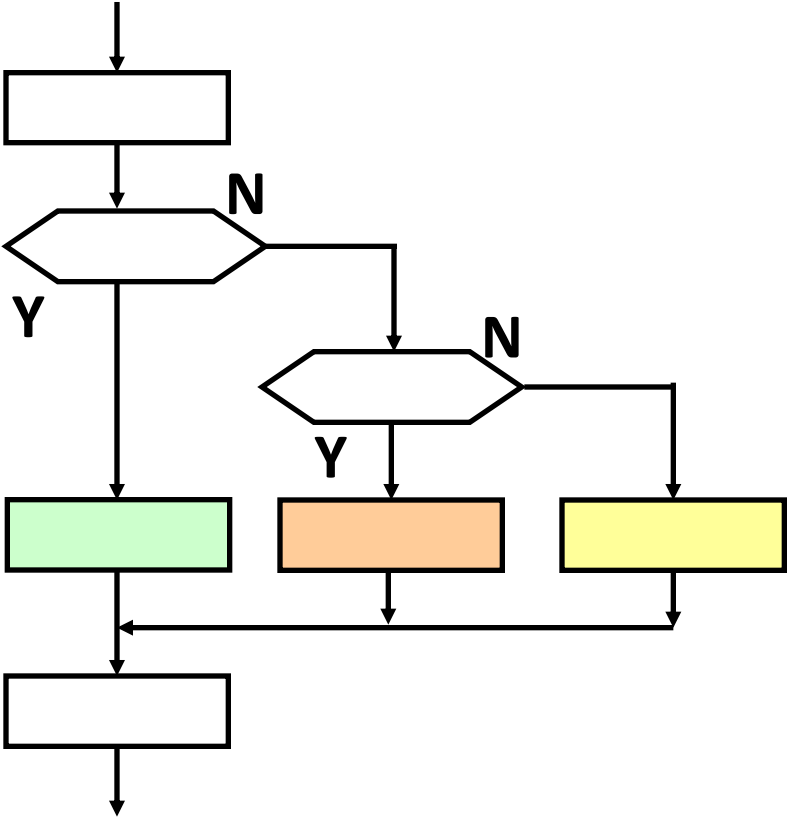
顺序结构



分支结构



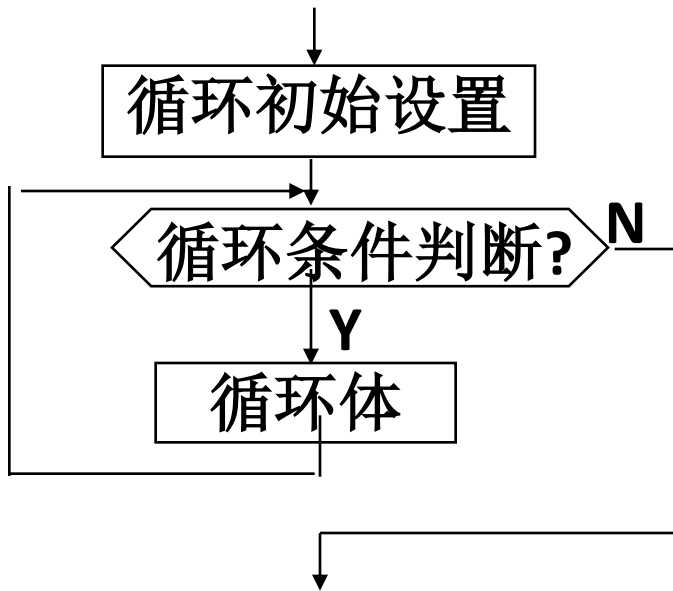
两个分支



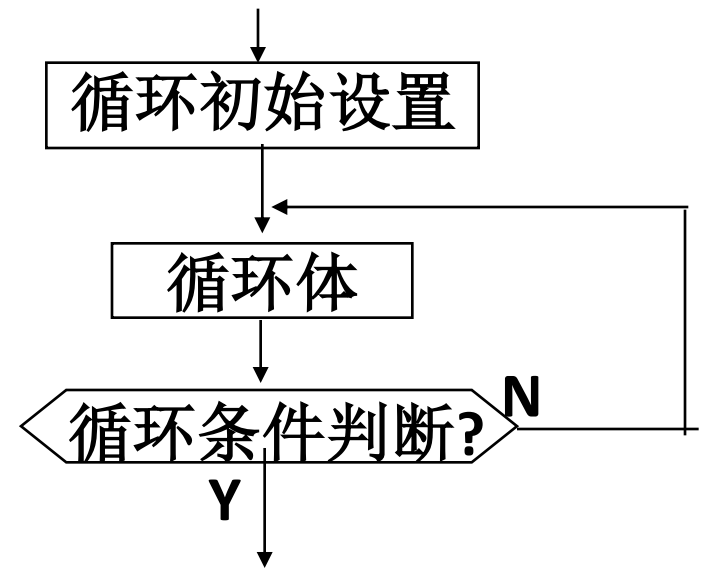
三个分支

if 语句、swich 语句等

循环结构



当型循环
(当条件成立进入循环)



直到型循环
(直到条件成立退出循环)

for 语句、while 语句、do... while语句等

函数调用

1. 多处调用完成同一功能的函数:

```
.....  
void main( )  
{  
    .....  
    delay();  
    .....  
    delay();  
    .....  
    delay();  
    .....  
}  
  
void delay( )  
{  
    .....  
    .....  
}
```

2. 利用函数进行模块化程序设计:

```
.....  
void main( )  
{  
    .....  
    IOinit();  
    TAINit();  
    ADCinit();  
    .....  
}  
void IOinit( )  
{  
    ..... }  
  
void TAINit( )  
{  
    ..... }  
  
void ADCinit( )  
{  
    ..... }
```

二) C语言程序设计举例

例1. 节日彩灯的控制

例2. 数码管的工作原理和显示控制

例3. 用按键控制节日彩灯显示的变化

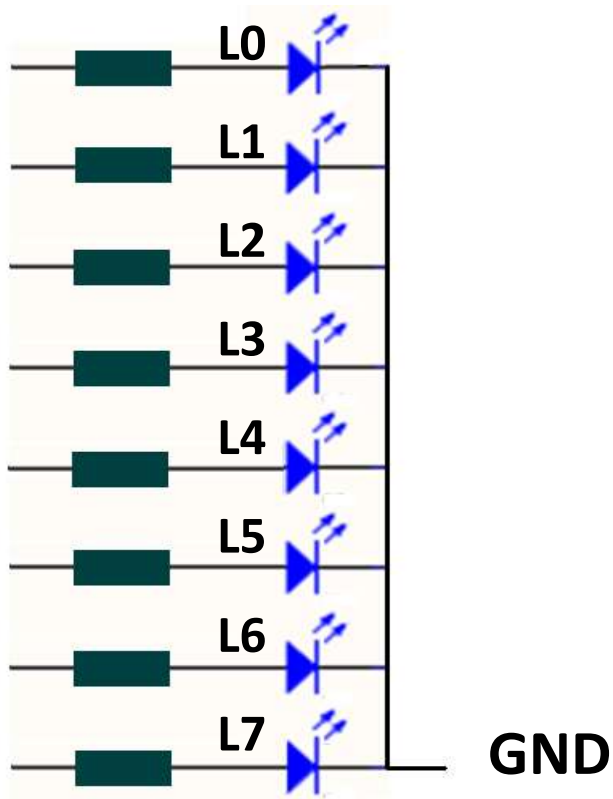
例4. 用按键控制数码管的显示

例5. 数据接收与处理

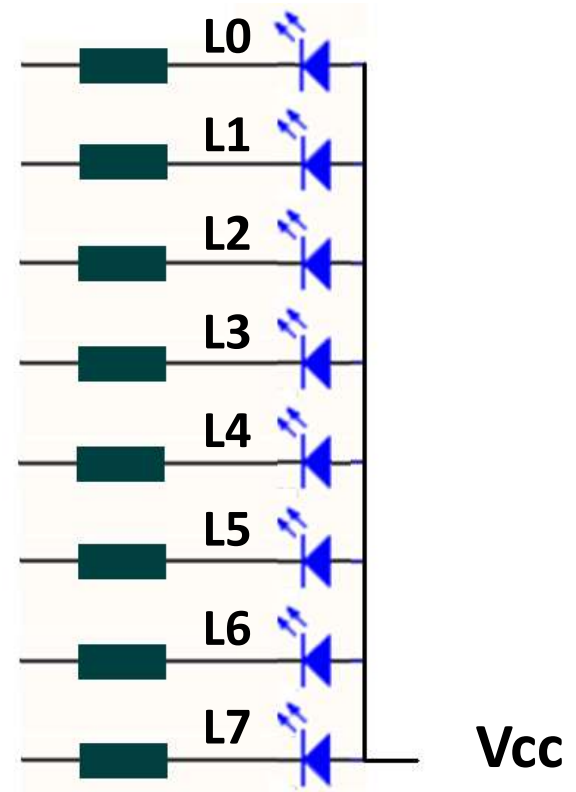
例6. 检测RAM

例7. 带初始化的变量实现

发光二极管电路图和工作原理



共阴连接



共阳连接

例1.

连接如右图，

编程控制发光二极管

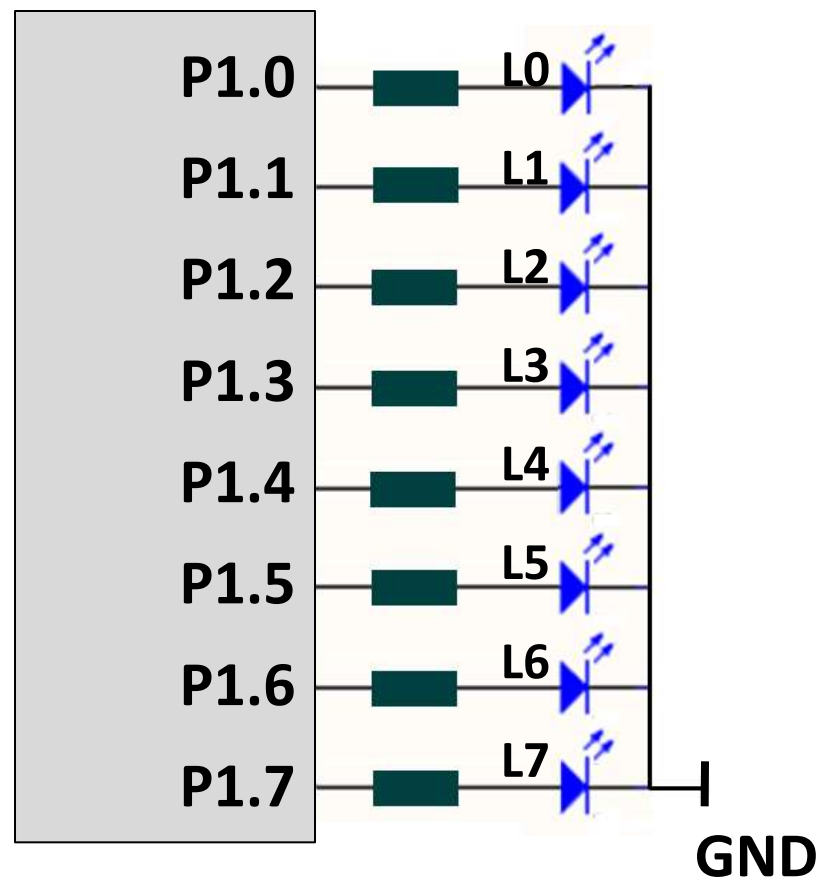
从 **L0** → **L1...** → **L7** → **L0...**

一盏一盏点亮，

每次只有一盏亮，

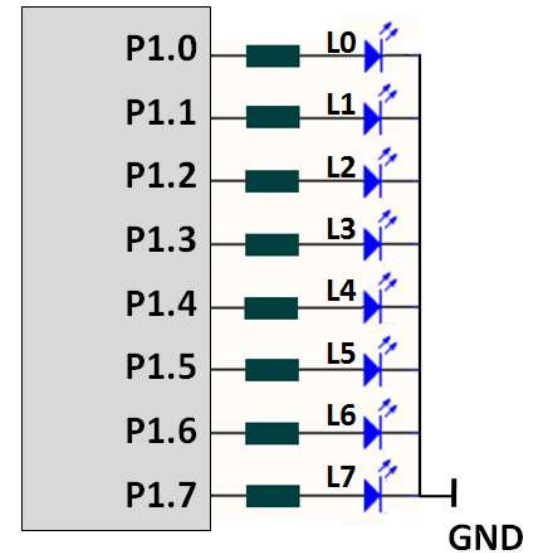
如此不断反复

MSP430F149



注意：硬件电路的连线设计与相关底层硬件的软件设置关系

8个发光二极管与单片机的端口1相连
通过控制与二极管相连接的引脚，
输出高/低不同的电平，可使二极管亮/灭
如P1.0连接L0，
P1.0输出1(高电平), 则L0亮；
P1.0输出0(低电平), 则L0灭



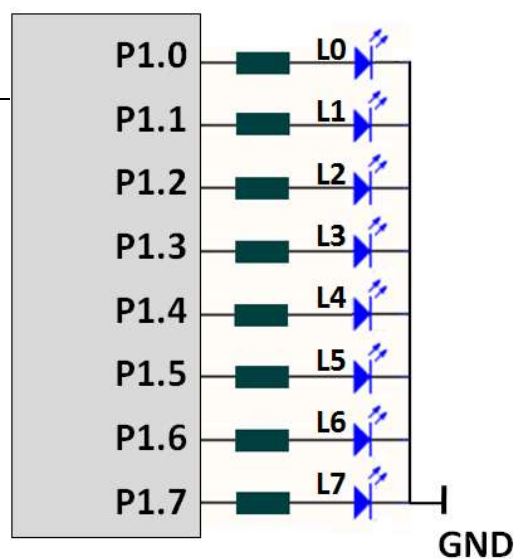
问题：

如何编程控制 端口引脚的输出？

有关的端口寄存器： P1SEL, P1DIR, P1OUT

方法1: 使用局部变量(Chap3_1_1.c)

```
#include "msp430F6638.h"
int main( void )
{
    unsigned char LED_0=0x01, LED_temp;
    unsigned int i, j;
    WDTCTL = WDTPW + WDTHOLD; //关闭看门狗
    P1SEL=0; //设置P1为基本I/O
    P1OUT=0; //使8个LED全灭
    P1DIR=0xFF; //设置P1为输出端口
    while(1) //无限循环
    {
        LED_temp=LED_0; //点亮L0的值
        for ( i=0; i<8; i++) //8个LED依次点亮
        { P1OUT=LED_temp; //输出到P1, 控制LED
            for ( j=0; j<0xffff; j++ ); //延时
            LED_temp=LED_temp<<1; //左移1位, 改变点亮值
        }
    }
};
```

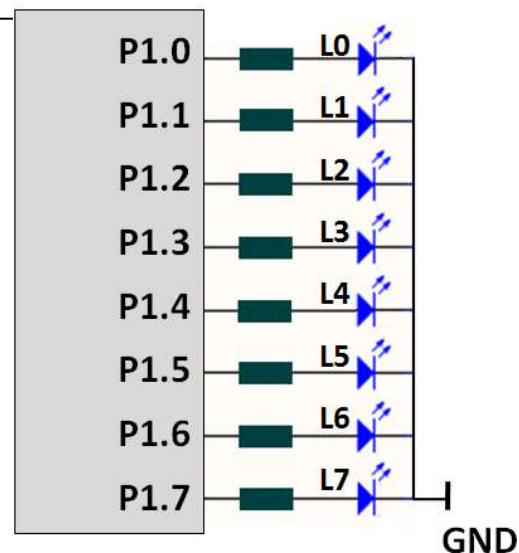


思考:

1. 若发光二极管不是连接在P1而是连接在P2上,如何修改程序?
2. 若要求是L7→L6... →L0 →L7循环点亮, 如何修改程序?
3. 若发光二极管不是共阴而是共阳连接,如何修改程序?

方法2：使用全局变量(Chap3_1_2.c)

```
#include "msp430F6638.h"
unsigned char LED_0=0x01, LED_temp;
unsigned int i, j;
int main( void )
{
    WDTCTL = WDTPW + WDTHOLD;
    P1SEL=0;           //设置P1为基本I/O
    P1OUT=0;          //使8个LED全灭
    P1DIR=0xFF;       //设置P1为输出端口
    while(1)          //无限循环
    {
        LED_temp=LED_0;           //点亮L0的值
        for ( i=0; i<8; i++)      //8个LED依次点亮
        { P1OUT=LED_temp;         //
            for (j=0;j<0xffff;j++); //延时
            LED_temp=LED_temp<<1; //改变点亮值
        }
    }
}
```



- 思考:**
1. 变量定义放在函数体内，和放在函数体外有什么不同？
 2. 在EW430的DEBUG下，
用View/watch 查看函数体内和放在函数体外的变量有何不同？

方法3: 使用函数调用(Chap3_1_3.c)

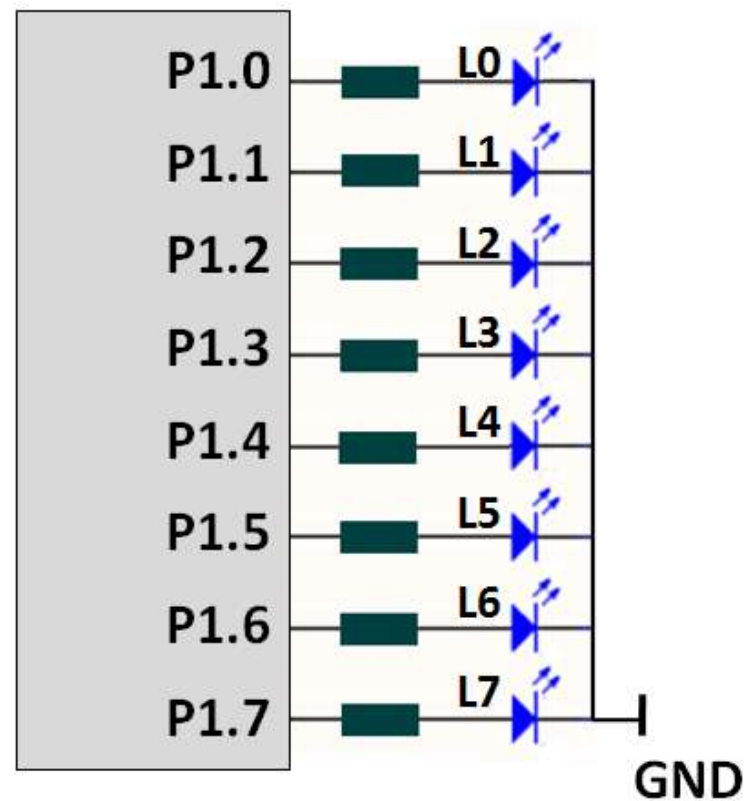
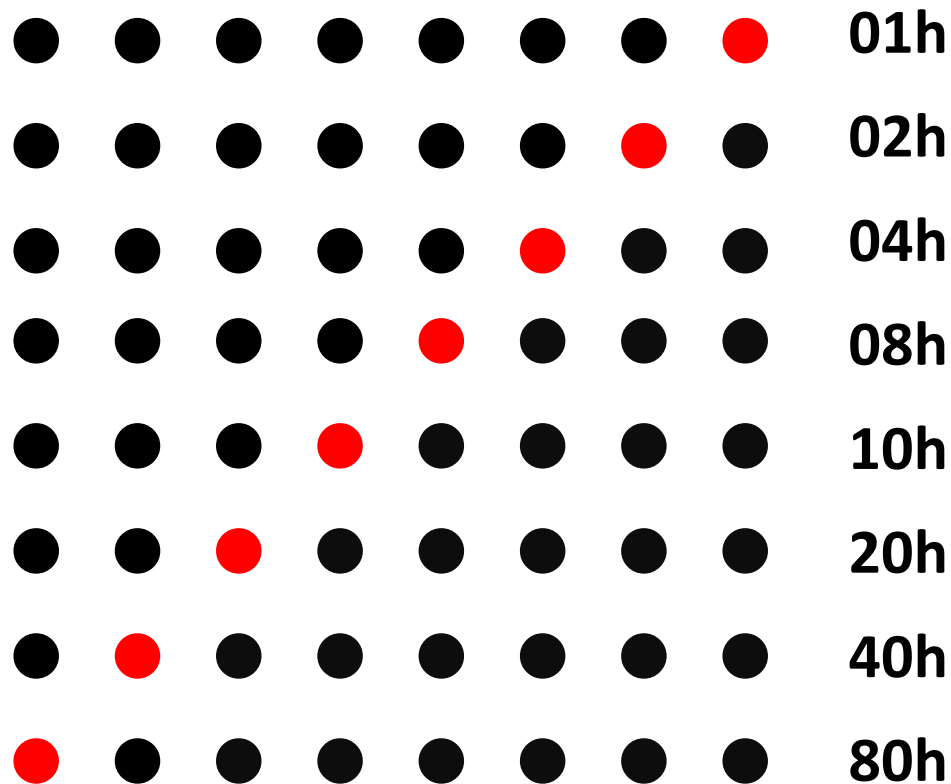
```
#include "msp430F6638.h"
void delay();
int main( void )
{
    unsigned char LED_0=0x01, LED_temp;
    unsigned int i;
    WDTCTL = WDTPW + WDTHOLD;           //关闭看门狗
    P1SEL=0;                             //设置P1为基本I/O
    P1OUT=0;                              //使8个LED全灭
    P1DIR=0xFF;                           //设置P1为输出端口
    while(1)                               //无限循环
    {
        LED_temp=LED_0;                   //点亮L0的值
        for ( i=0; i<8; i++)              //8个LED依次点亮
        { P1OUT=LED_temp;
          delay();                          //调用延时子程
          LED_temp=LED_temp<<1;           //改变点亮值
        }
    }
};

void delay()
{
    unsigned int i;
    for (i=0; i<0xffff; i++);
}
```

思考:

函数main()和delay()中的都有变量i;
两者是同一变量吗?

有无其他的编程方法?



用数组存放循环显示的显示值

```
unsigned char LEDdata[ ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
```

方法4：用数组存放显示表(Chap3_1_4.c)

```
#include "msp430F6638.h"
void delay( );
int main( void )
{
    unsigned char LEDdata[ ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
    unsigned int i;
    WDTCTL = WDTPW + WDTTHOLD;           //关闭看门狗
    P1SEL=0;           //设置P1为基本I/O
    P1OUT=0;           //使8个LED全灭
    P1DIR=0xFF;       //设置P1为输出端口
    while(1)           //无限循环
    {
        for ( i=0; i<8; i++ )           //8个LED依次点亮
        { P1OUT=LEDdata[i];           //
          delay();                     //调用延时子程
        }
    }
};

void delay( )
{
    unsigned int i;           //定义函数变量
    for ( i=0; i<0xffff; i++ ); //延时
}
```

比较数组定义成下面3种不同方式，会有何不同？

```
#include "msp430F6638.h"
void delay( );
int  main( void )
{   unsigned char LEDdata[8]={1, 2, 4,8,0x10,0x20,0x40,0x80};
    .....
}
```

```
#include "msp430F6638.h"
unsigned char LEDdata[8]={1, 2, 4,8,0x10,0x20,0x40,0x80};
void delay( );
int  main( void )
{   .....   }
```

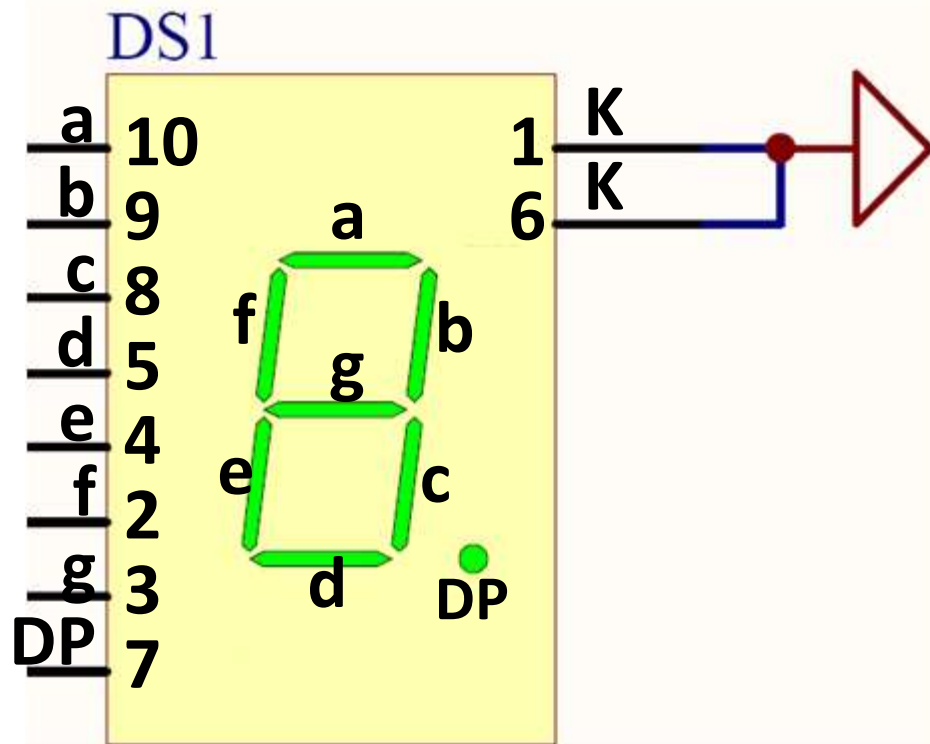
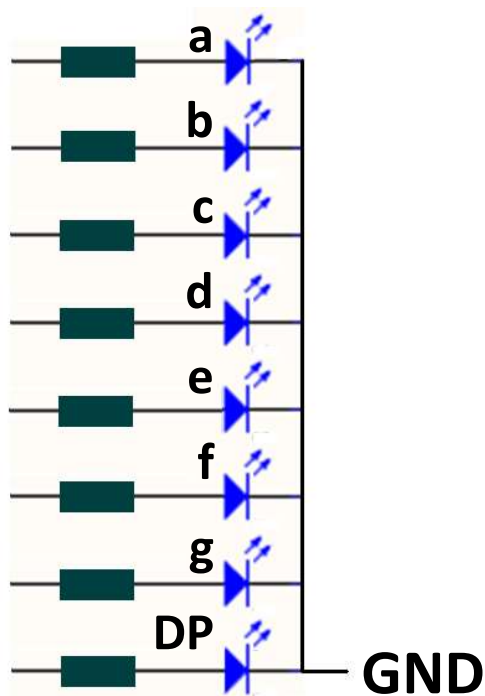
```
#include "msp430F6638.h"
const unsigned char LEDdata[8 ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
void delay( );
int  main( void )
{   .....   }
```

方法5：用指针取数组元素(Chap3_e1_5.c)

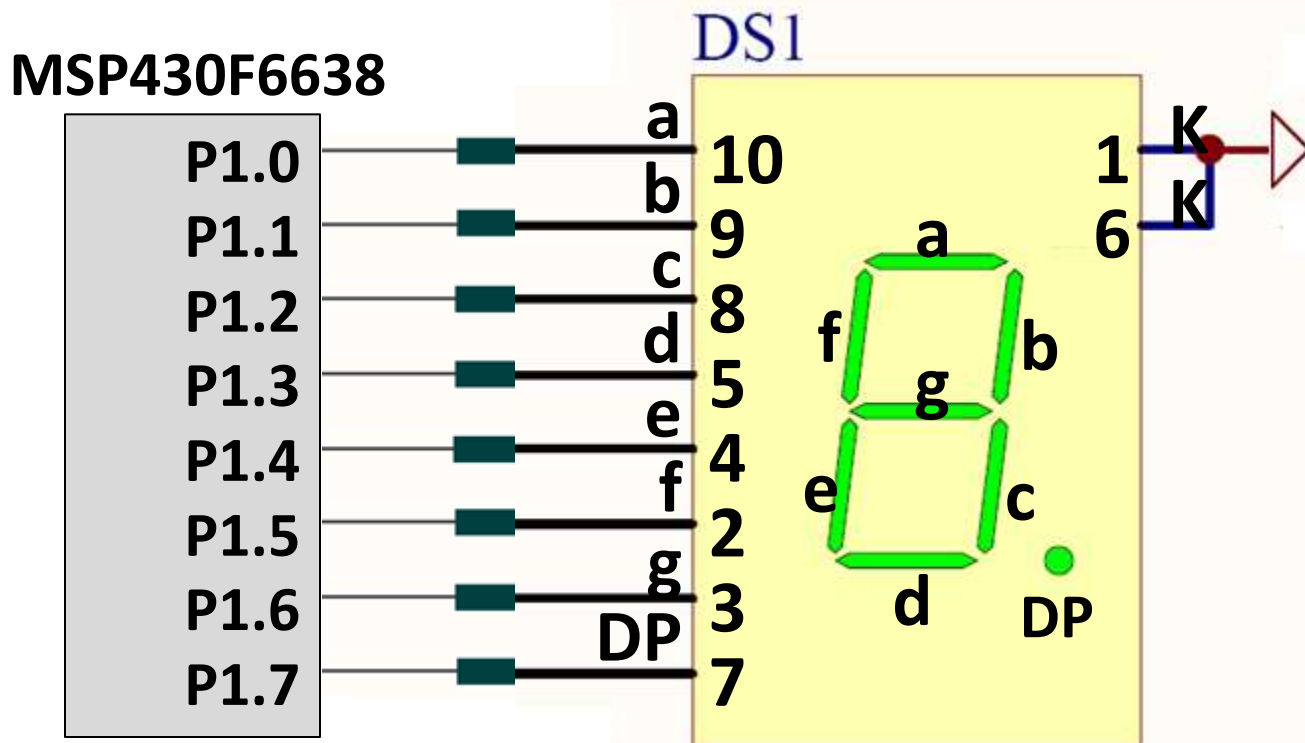
```
#include "msp430F6638.h"
void delay();
int main( void )
{
    unsigned char LEDdata[ ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
    unsigned int i;
    unsigned char *p;
    WDTCTL = WDTPW + WDTHOLD;           //关闭看门狗
    P1SEL=0;                            //设置P1为基本I/O
    P1OUT=0;                             //使8个LED全灭
    P1DIR=0xFF;                          //设置P1为输出端口
    while(1)                             //无限循环
    {
        p=&LEDdata[0];                  //指向数组首地址
        for ( i=0; i<8; i++)           //8个LED依次点亮
        { P1OUT=*p;
          p++;
          delay();                     //调用延时子程
        }
    };
}
void delay()
{
    unsigned int i;                    //定义函数变量
    for (i=0;i<0xffff;i++);           //延时
}
```

一位数码管电路图和工作原理

将8个发光二级管按下图方式摆放，构成一位8段数码管，可通过不同段的显示组合，显示出不同字符

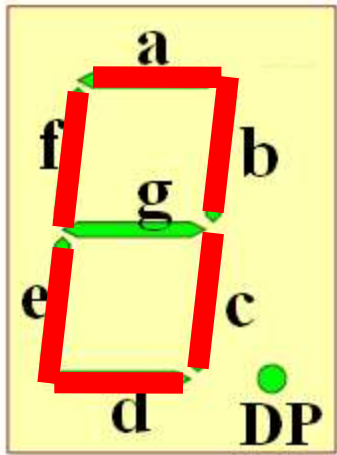


例2 连线如下，
编程顺序在数码管上显示0~9, A~F

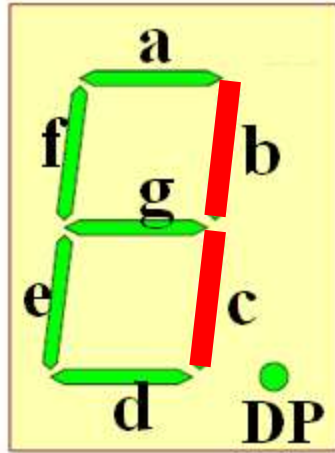


注意：硬件电路的连线设计与控制数码管的关系

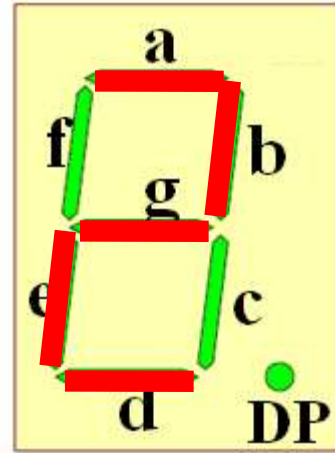
DP g f e d c b a : 字符0~7的段码值



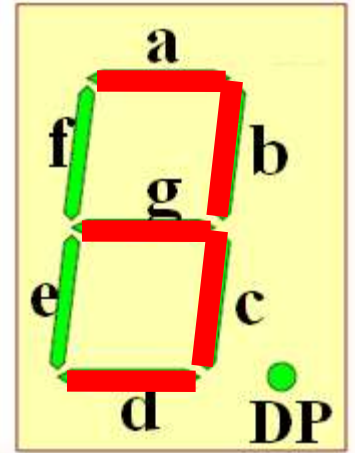
00111111



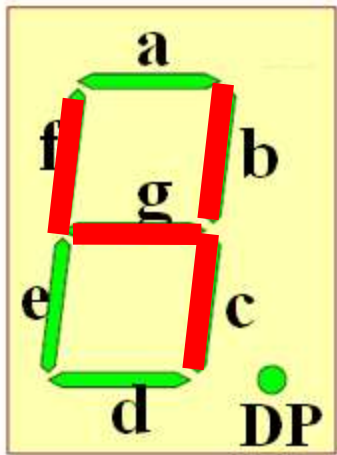
00000110



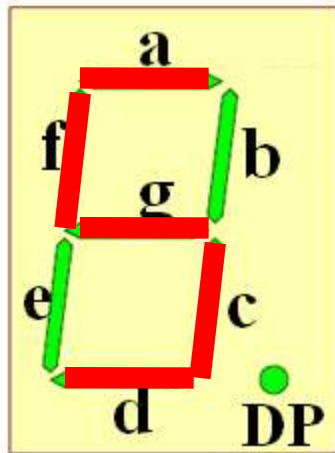
01011011



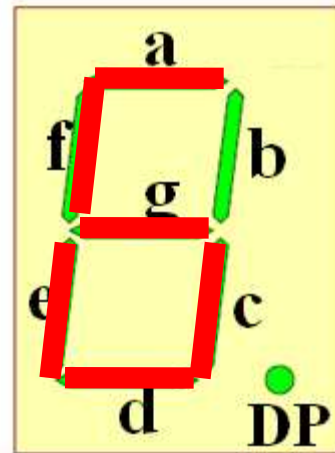
01001111



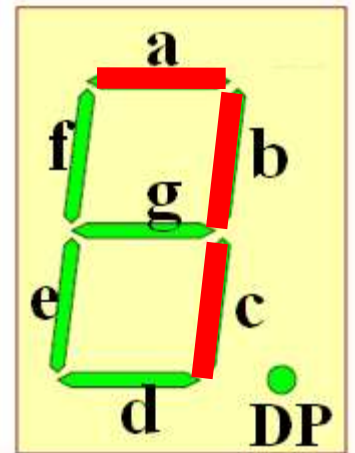
01100110



01101101

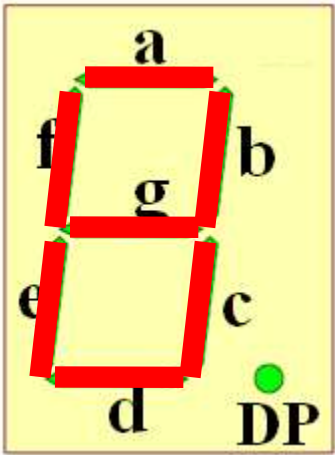


01111101

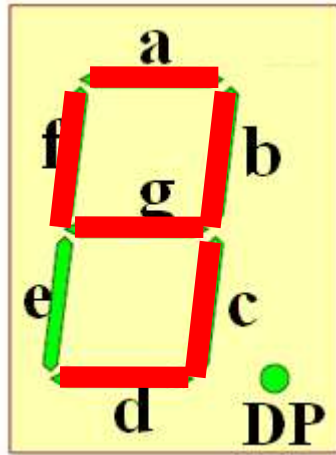


00000111

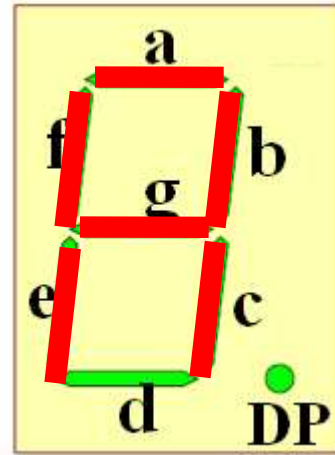
DP g f e d c b a : 字符8~F的段码值



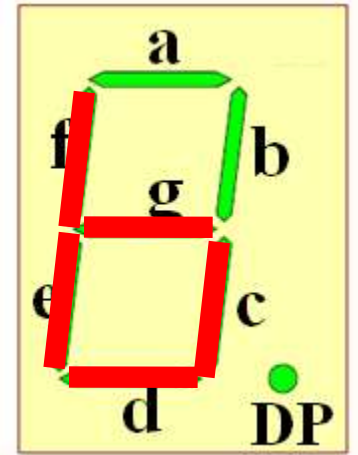
01111111



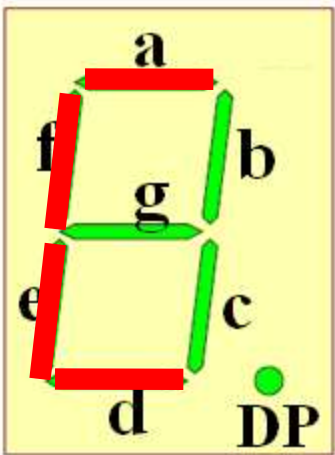
01101111



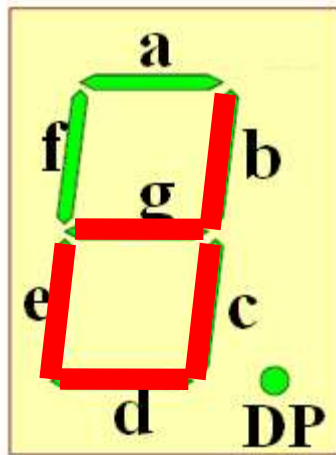
01110111



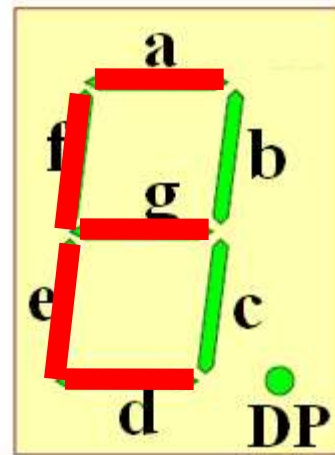
01111100



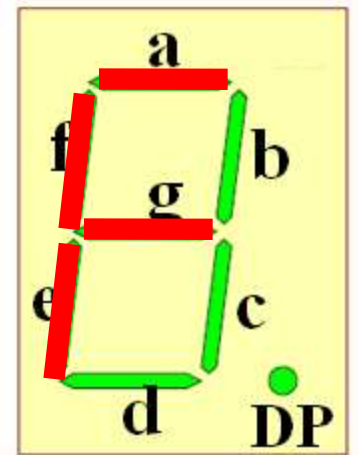
00111001



01011110



01111001



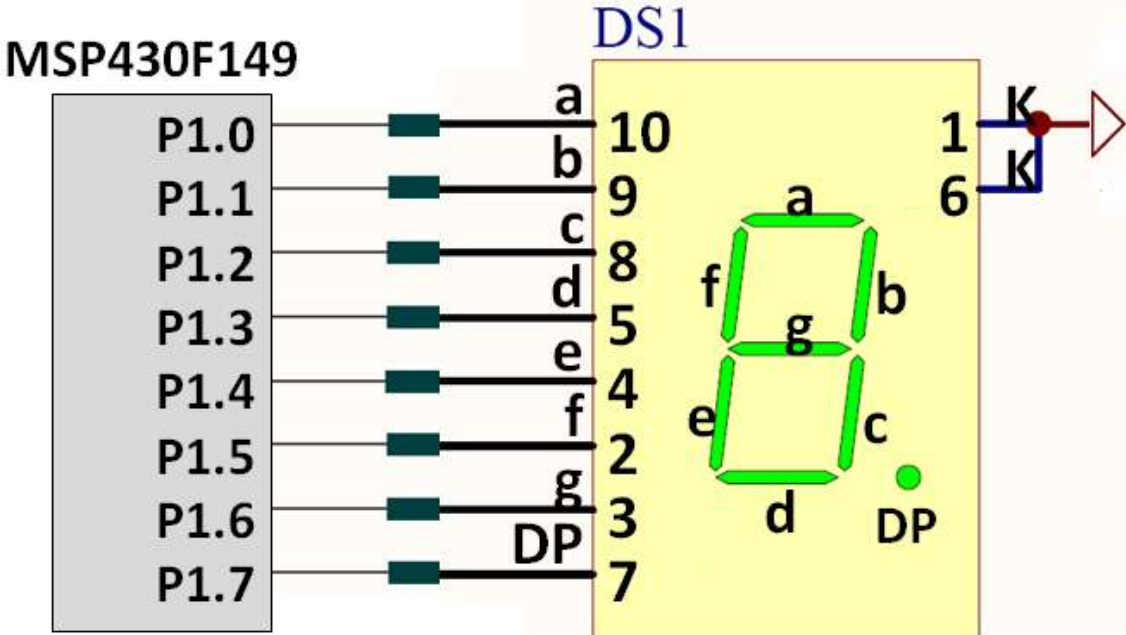
01110001

```
const char LEDtab[16]={0x3F, 0x06, 0x5B, 0x 4F, 0x66, 0x6D,  
0x7D, 0x07,0x7F, 0x 6F, 0x77, 0x7C, 0x39, 0x 5E, 0x79, 0x71}  
//在ROM中存放0~F的显示段码表
```

思考:

如果每个字符都点亮右下角的小数点，上表该如何修改？

例2编程：顺序在数码管上显示0~9,A~F



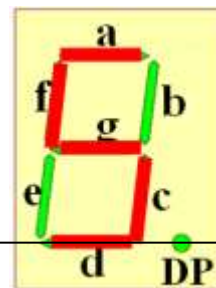
将数码管表格值顺序输出到端口1即可显示0~F

```
.....  
const char LEDtab[16]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,  
    0x7D, 0x07,0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};  
.....  
int main(void )  
{  
    unsigned int i;  
    WDTCTL = WDTPW + WDTTHOLD;    //关闭看门狗  
    P1SEL=0;                        //设置P1为基本I/O  
    P1OUT=0;                        //使8个LED全灭  
    P1DIR=0xFF;                    //设置P1为输出端口  
    while(1)  
    {  
        for( i=0; i<8; i++ )  
        { P1OUT=LEDtab[i];    //取表中的某一种状态输出  
          delay();          //延时  
        };  
    }  
}
```

例2 程序清单

```
#include "msp430F6638.h"
void delay();
const char LEDtab[16]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
                      0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
int main(void )
{  unsigned int i;
   WDTCTL = WDTPW + WDTLHOLD;      //关闭看门狗
   P1SEL=0;                        //设置P1为基本I/O
   P1OUT=0;                        //使8个LED全灭
   P1DIR=0xFF;                    //设置P1为输出端口
   while(1)
   {   for( i=0; i<8; i++ )
       {   P1OUT=LEDtab[i];        //取表中的某一种状态输出
           delay();                //延时
       };
   }
}
void delay()
{  unsigned int i;                //定义函数变量
   for (i=0;i<0xffff;i++);        //延时
}
```

思考: 如何编程让数码管上显示变量number中的数值?
 假设number中的内容在0000~000Fh之间?
 如 number = 5 在数码管上显示出“5”

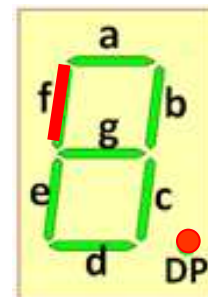


```
.....
const char LEDtab[16]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
    0x7D, 0x07,0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
char number;
```

.....

```
P1OUT=number;
delay(); //延时
```

.....

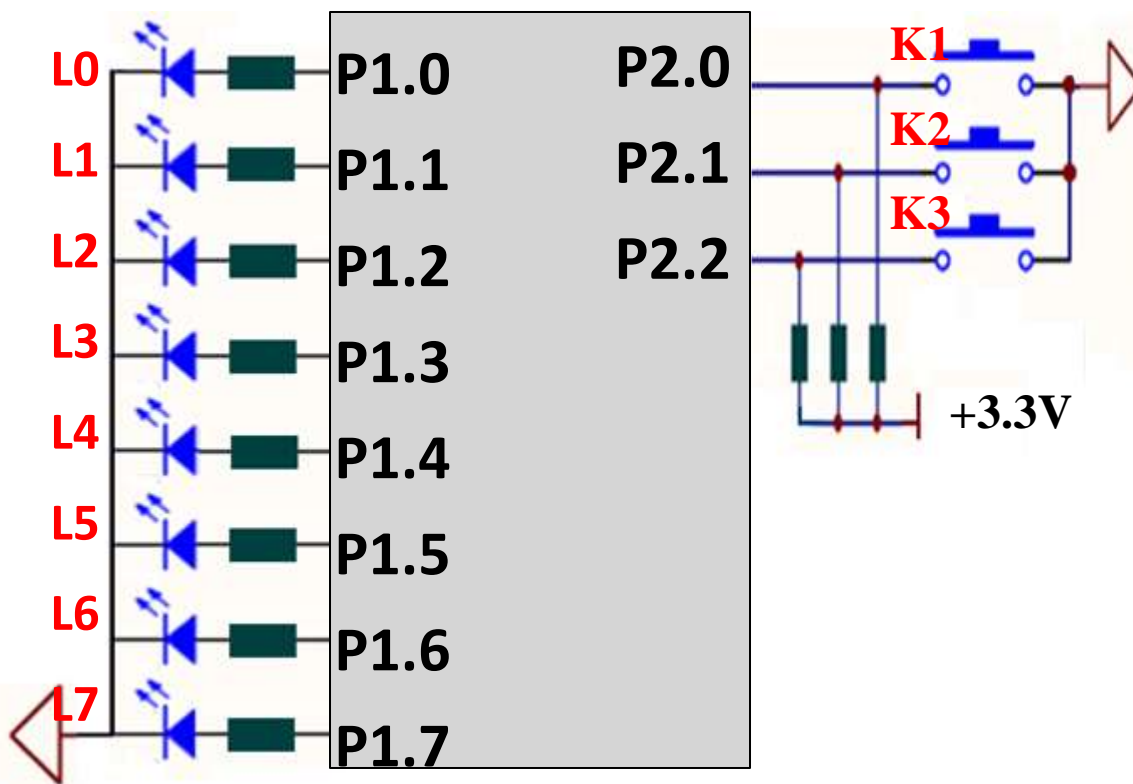


应在显示码表中查得显示码后写入P1OUT

```
P1OUT=LEDtab[number];
```

number=	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
LEDtab[16]=	3F	06	5B	4F	66	6D	7D	07	7F	6F	77	7C	39	5E	79	71
数码管显示:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

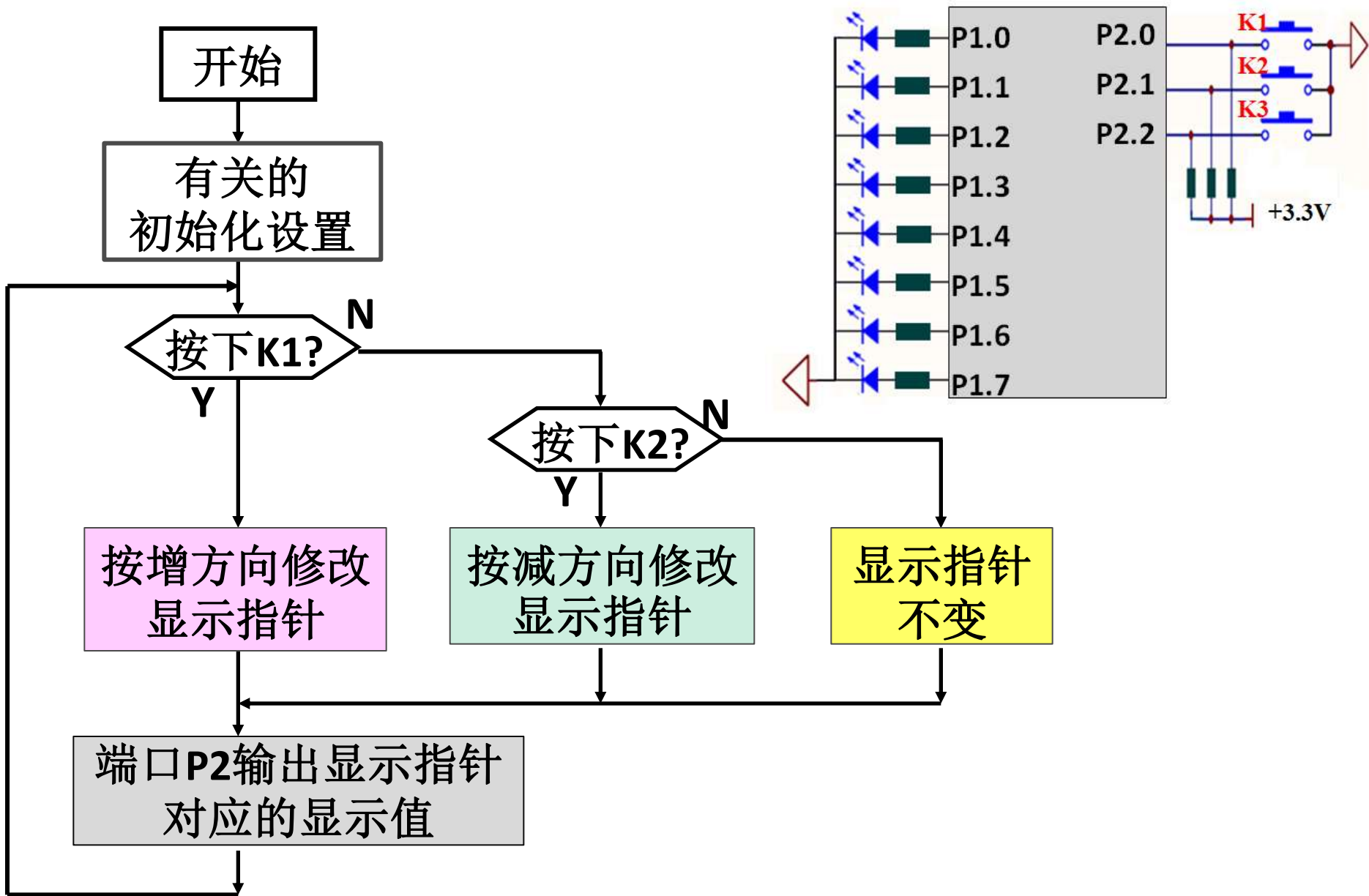
例3. 用按键控制节日彩灯显示的变化



当按下K1键时, 灯从L0→L1...→L7→L0...循环点亮,
当按下K2键时, 灯从L7→L6...→L0→L7...循环点亮,
当按下K3键时, 灯的状态保持不变, 停止变化
每次只有一个灯点亮

根据硬件连线，确定显示表格

`LEDtab[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80}`



方法1: 端口2使用字节操作(Chap3_3_1.c)

```
#include "msp430F6638.h"
const unsigned char LEDdata[ ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
int main( void )
{
    unsigned int j;                //延时控制变量
    int i=0;                       //置显示码数组初始下标
    WDTCTL = WDTPW + WDTTHOLD;    //关闭看门狗
    P1SEL=0;                       //设置P1为基本I/O
    P1OUT=0;                       //使8个LED全灭
    P1DIR=0xFF;                   //设置P1为输出端口
    P2SEL &= 0xFC;               //设置P2.0~1为基本I/O
    P2DIR &= 0xFC;               //设置P2.0~1为输入
    while(1)                      //无限循环
    {
        if ( (P2IN&BIT0) ==0)    //判断K1是否按下
        {
            i=i+1;
            if ( i==8 ) i=0; }
        else if ( (P2IN&BIT1) ==0) //判断K2是否按下
        {
            i=i-1;
            if ( i==-1 ) i=7; }
        P1OUT=LEDdata[i];        //取显示码输出
        for ( j=0; j<0xffff; j++); //延时
    };
}
```

注意关系运算符==的优先级高于逻辑运算符&, 比较:
(P2IN & BIT1) ==0 与 (P2IN BIT1 ==0) 一样吗?

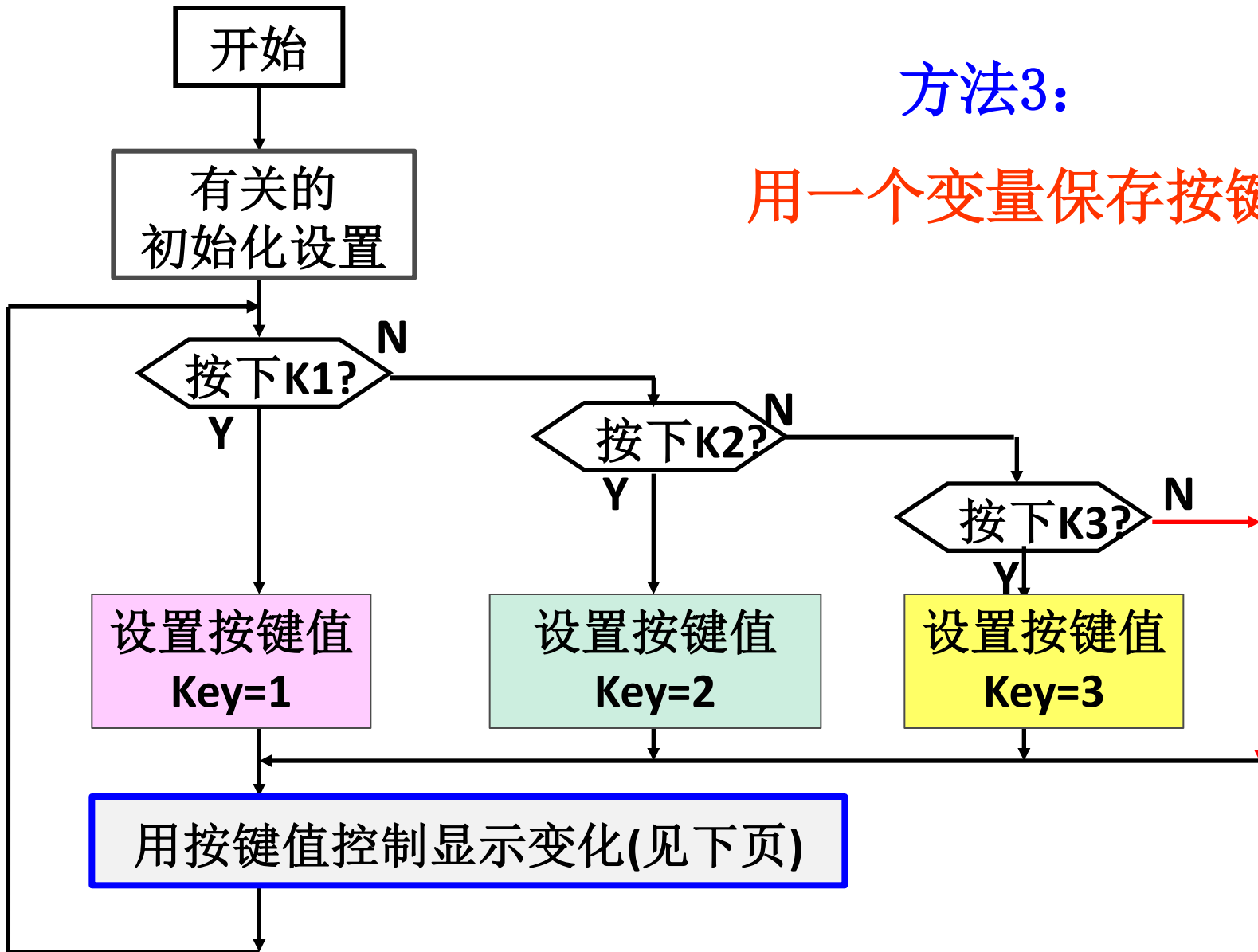
思考:

前面编写的程序，需要一直按着键，灯才显示变化，
可否修改程序，不需一直按键，
灯按最近一次的按键进行显示？

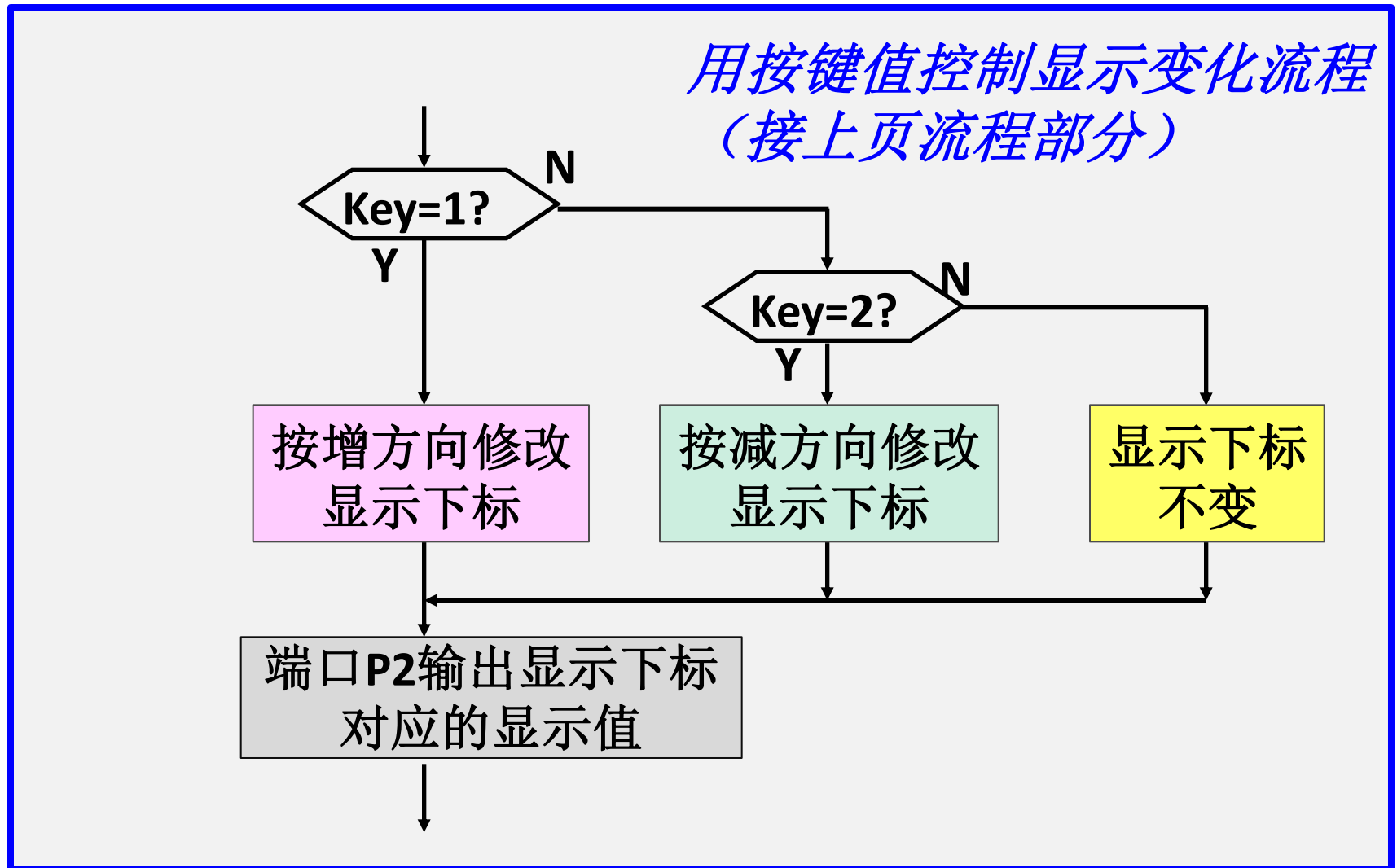
```
while(1) //无限循环
{
    if ((P2IN&BIT0)==0) //判断K1是否按下
    {
        i=i+1;
        if ( i==8 ) i=0; }
    else if ((P2IN&BIT1)==0) //判断K2是否按下
    {
        i=i-1;
        if ( i== -1 ) i=7; }
    P1OUT=LEDdata[i]; //取显示码输出
    delay(); //延时
};
```

方法3:

用一个变量保存按键动作



显示表格 LEDdata[8]={1, 2, 4,8,0x10,0x20,0x40,0x80}



思考：方法3与前两种实现方法有何不同？

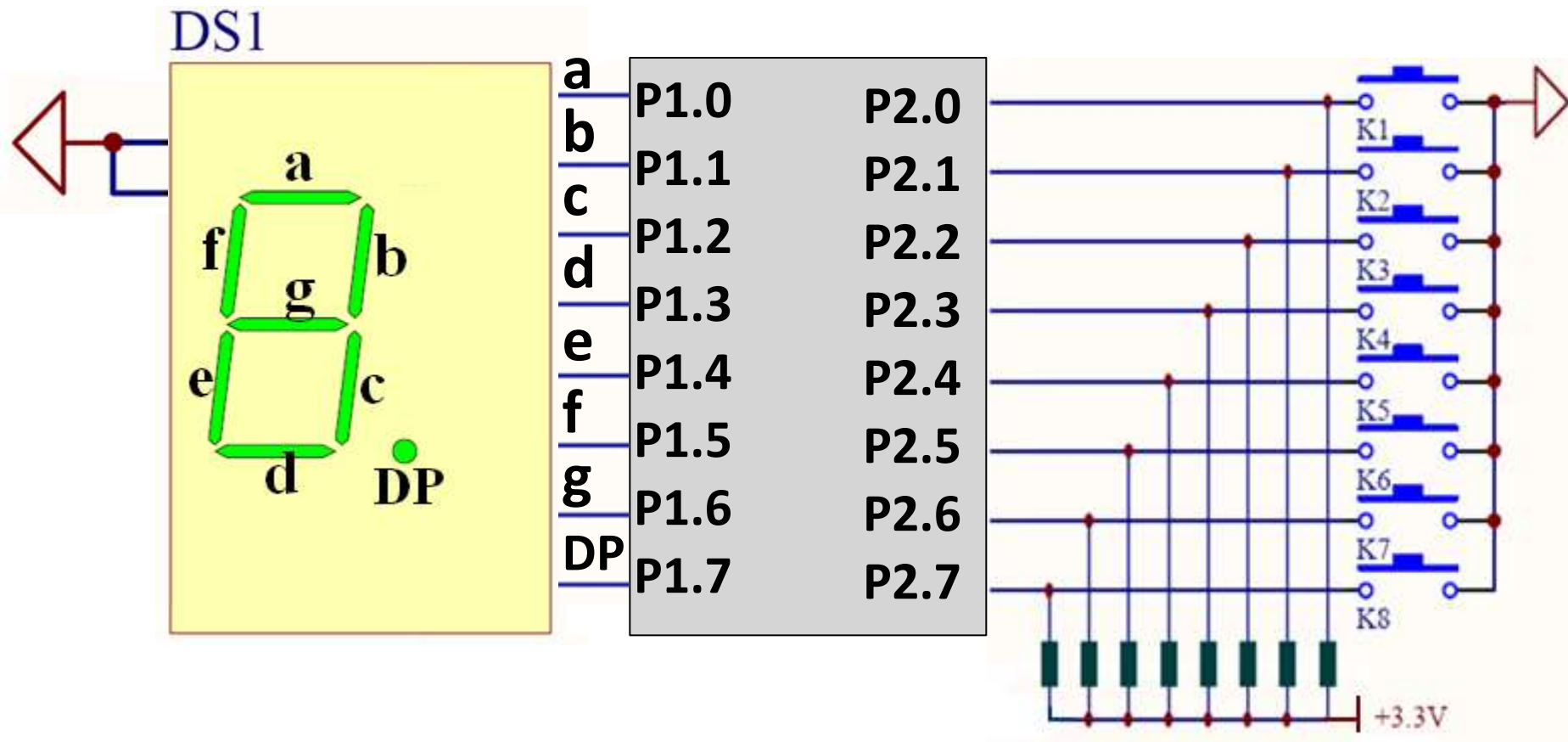
方法2: 用变量保存按键值(Chap3_3_3.c)

```
#include "io430.h"
int main( void )
{
    unsigned char LEDdata[ ]={1, 2, 4,8,0x10,0x20,0x40,0x80};
    unsigned int j, Key ;
    int i=0; //置显示码数组初始下标
    WDTCTL = WDTPW + WDTCTL; //关闭看门狗
    ..... //初始化端口1和端口2相关引脚 (略)
    while(1) //无限循环
    {
        if ((P2IN&BIT0)==0) { Key=1; } //判断K1是否按下
        else if ((P2IN&BIT1)==0) { Key=2; } //判断K2是否按下
        else if ((P2IN&BIT2)==0) { Key=3; } //判断K3是否按下

        if (Key==1) //据Key值控制显示码数组下标
        {
            i=i+1;
            if ( i==8 ) i=0; }
        else if (Key==2)
        {
            i=i-1;
            if ( i==-1 ) i=7; }

        P1OUT=LEDdata[i]; //取显示码输出
        for ( j=0; j<0xffff; j++); //延时
    };
}
```

例4. 用按键控制控制数码管的显示，
当按下K1显示0，按下K2显示1，...，按下K8显示7

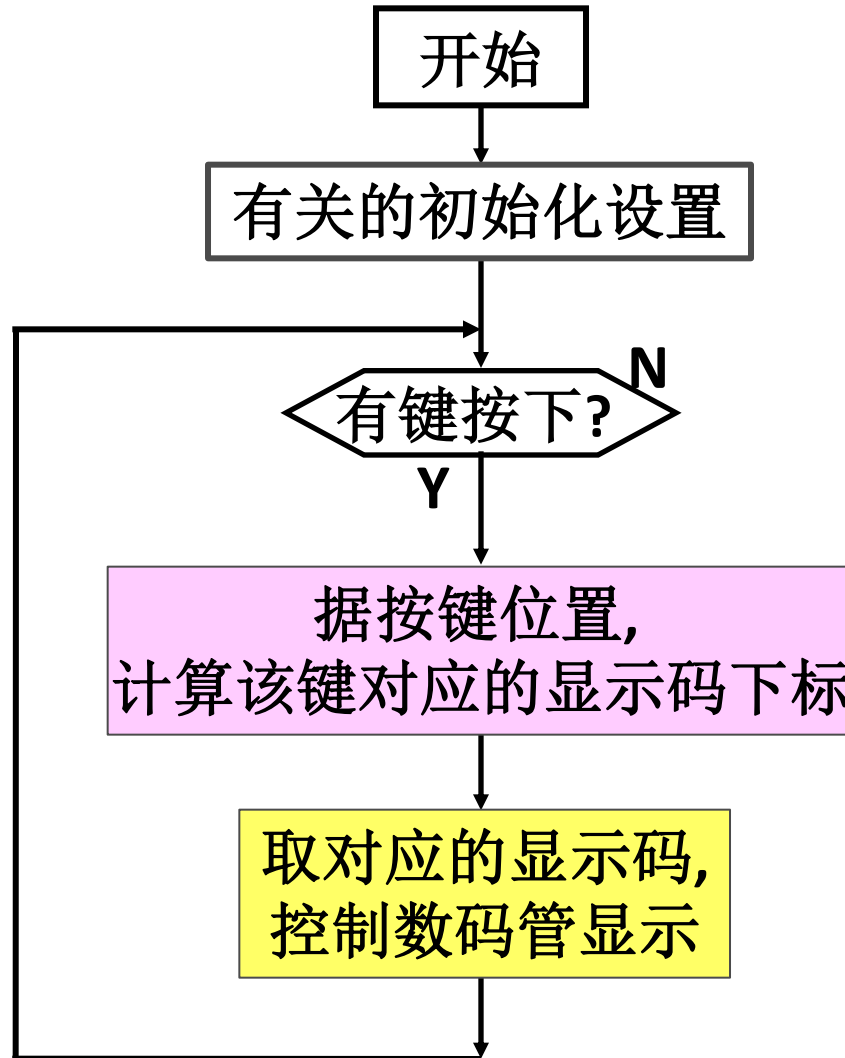


数码管显示

0 1 2 3 4 5 6 7

LEDtab[16]={0x3F, 0x06, 0x5D, 0x4F, 0x66, 0x6D, 0x7D, 0x07}

K1 K2 K3 K4 K5 K6 K7 K8



例4 用按键控制控制数码管的显示C语言程序段

```
while(1)
{
    if      ((P2IN&BIT0)==0) {Key=0;}      //判断K1是否按下
    else if ((P2IN&BIT1)==0) {Key=1;}      //判断K2是否按下
    else if ((P2IN&BIT2)==0) {Key=2;}      //判断K3是否按下
    else if ((P2IN&BIT3)==0) {Key=3;}      //判断K4是否按下
    else if ((P2IN&BIT4)==0) {Key=4;}      //判断K5是否按下
    else if ((P2IN&BIT5)==0) {Key=5;}      //判断K6是否按下
    else if ((P2IN&BIT6)==0) {Key=6;}      //判断K7是否按下
    else if ((P2IN&BIT7)==0) {Key=7;}      //判断K8是否按下

    P1OUT = LEDdata[Key];                  //取显示码输出
    delay( );                              //延时
};
```

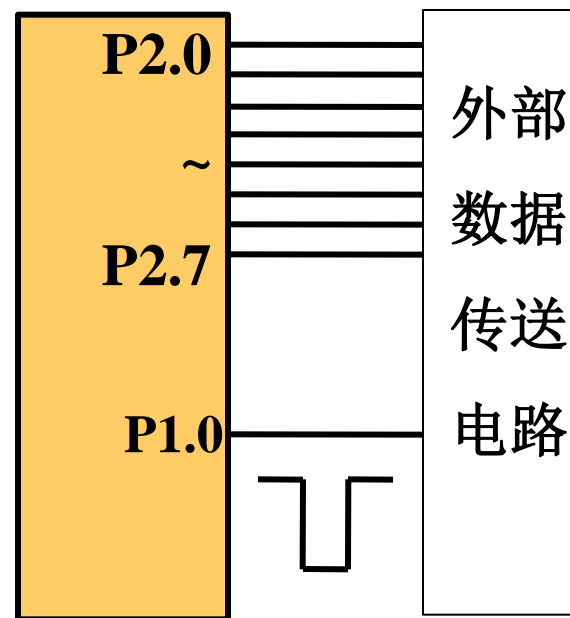
例5. 数据接收与处理

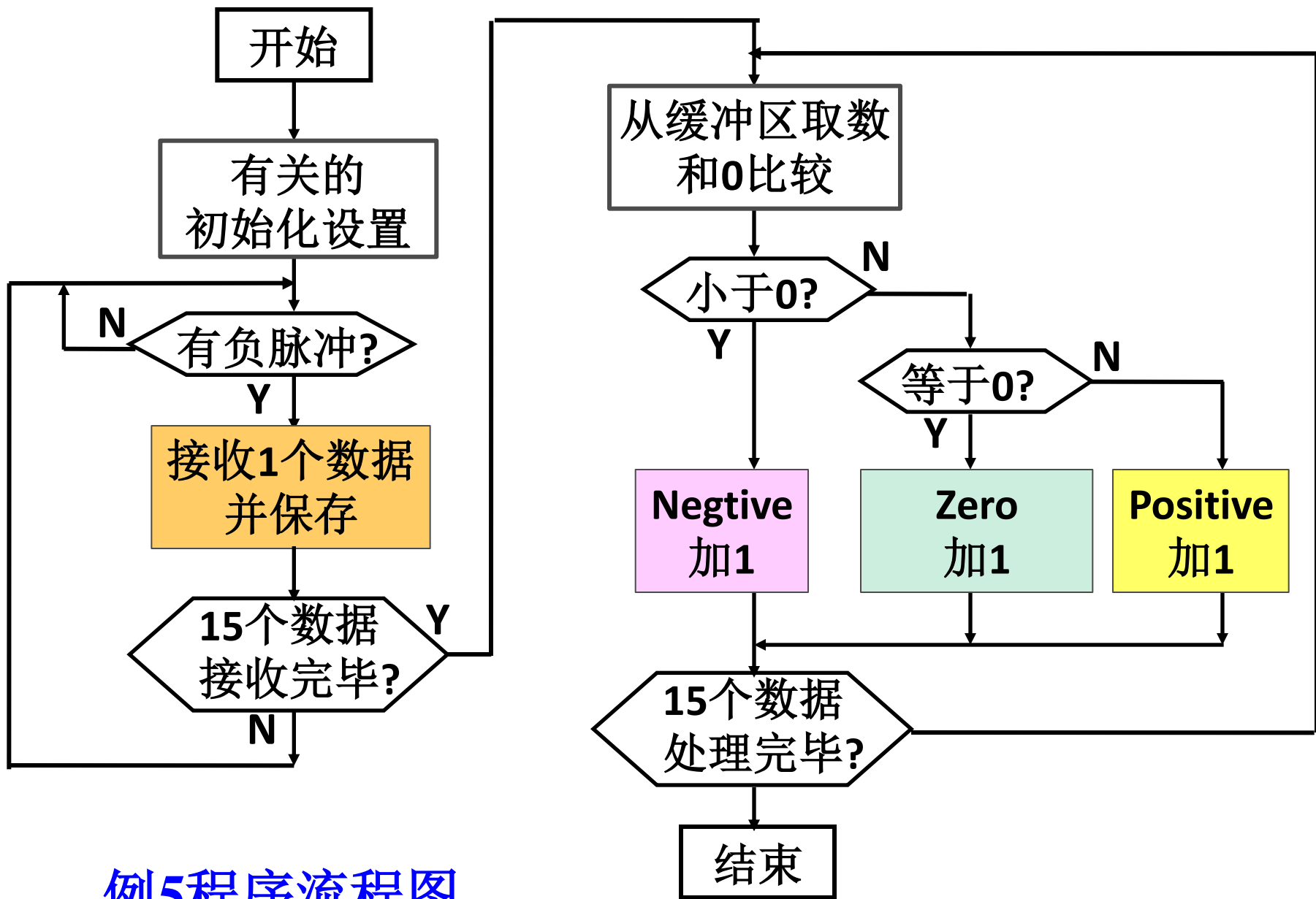
如图, 8位二进制数据由外部电路从端口P2传入, 每传一个数据, 外部电路向P1.0发出一个负脉冲信号, 即从高变低,再从低变回高。

单片机通过检测P1.0上有负脉冲信号, 知道有新数据到来,可从端口2读取该数据。

编程从端口P2接收15个数据, 保存在以buffer为首的RAM缓冲区中, 计数这15个数据中正数、负数、零的个数, 并将计数的结果顺序存放在定义的Positive、Negtive、Zero三个变量中。

MSP430F149





例5程序流程图

例5程序清单

```
#include "msp430f6638.h"
int main( void )
{
    unsigned char buffer[15 ];           //数据存放缓冲区
    unsigned int j;                     //循环控制变量
    int Negtive=0,Zero=0,Positve=0;     //计数结果存放
    WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗

    //初始化相关I/O引脚

    P2SEL=0;                            //设置P2为基本I/O
    P2DIR=0;                             //设置P2为输入端口
    P1SEL &= ~BIT0;                      //设置P1.0为基本I/O
    P1DIR &= ~BIT0;                      //设置P1.0为输入
```

例5程序清单(续)

//下面程序段功能是接收数据到缓冲区中

//每检测到一个负脉冲,接收一次数据,保存到缓冲区中

```
For(i=0;i<16;i++)
```

```
{ while((P1IN&BIT0)!=0); //查询P1.0上低电平?无,则等待
```

```
  while((P1IN&BIT0)==0); //查询P1.0上高电平?无,则等待
```

```
  buffer[i]=P2IN; //P1.0上检测到负脉冲后,接收一个数据
```

```
}
```

//下面程序段是处理数据

//从缓冲区取出数据,与0做比较,

//判断是正数,负数,还是0

```
For(i=0;i<16;i++)
```

```
{ if (buffer[i]<0) then { Negtive++;}
```

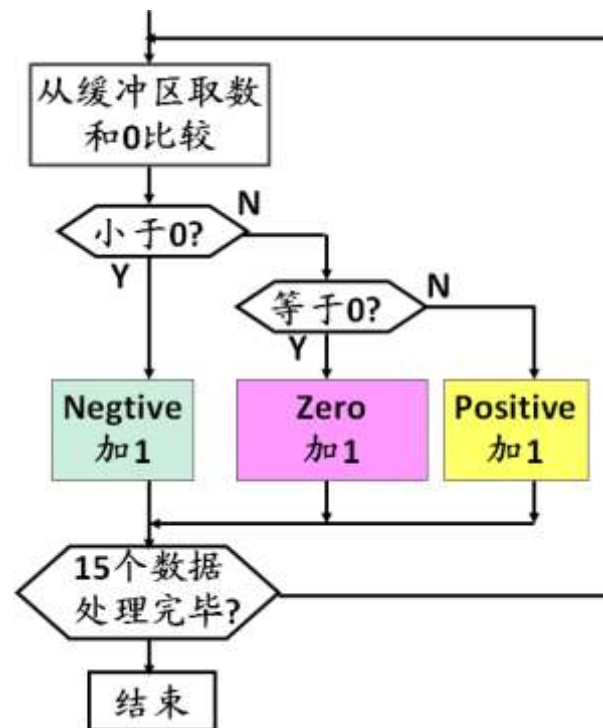
```
  else if (buffer[i]=0) then { Zero++;}
```

```
  else { Positve++;}
```

```
}
```

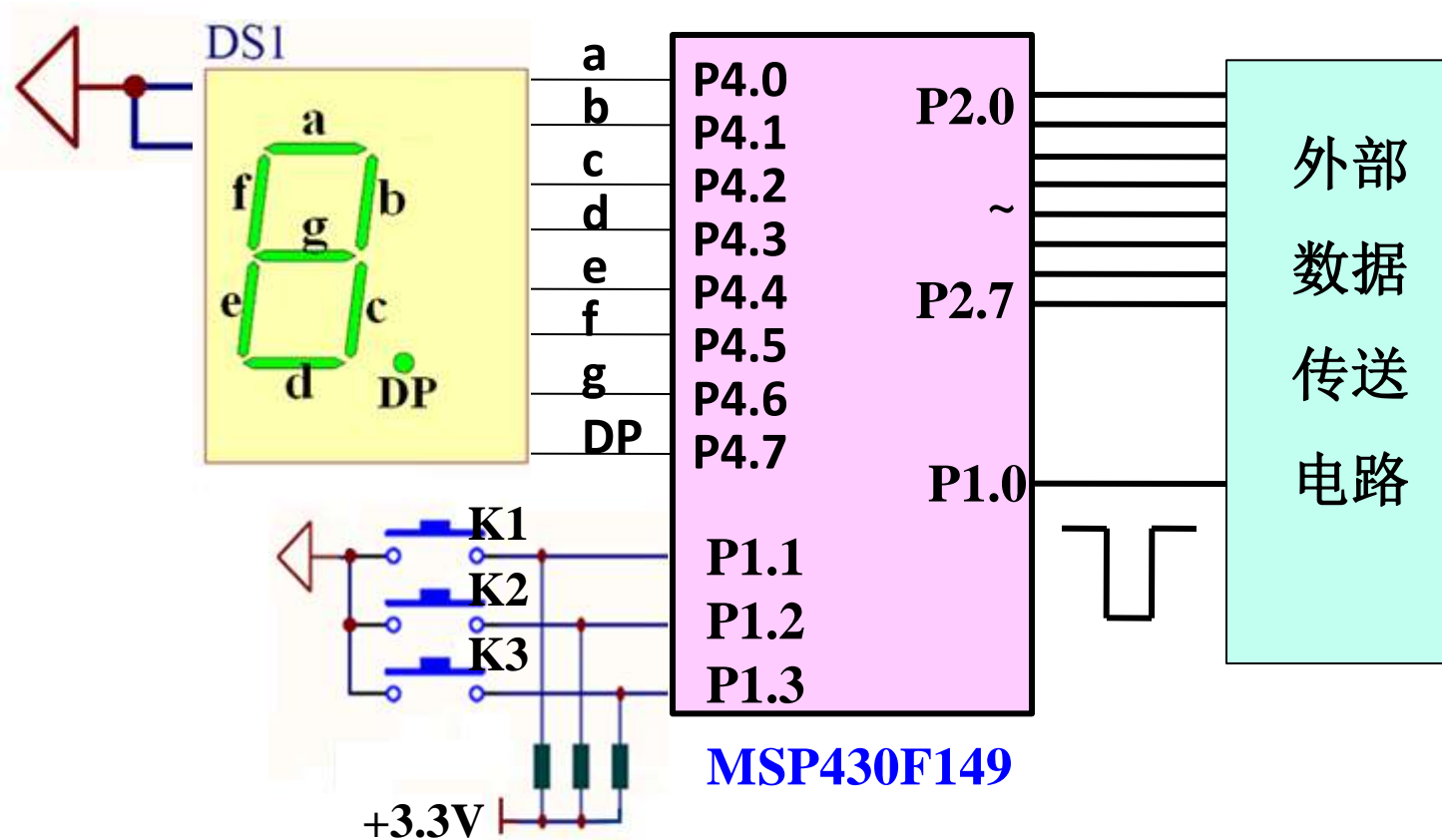
```
while(1); //无限循环
```

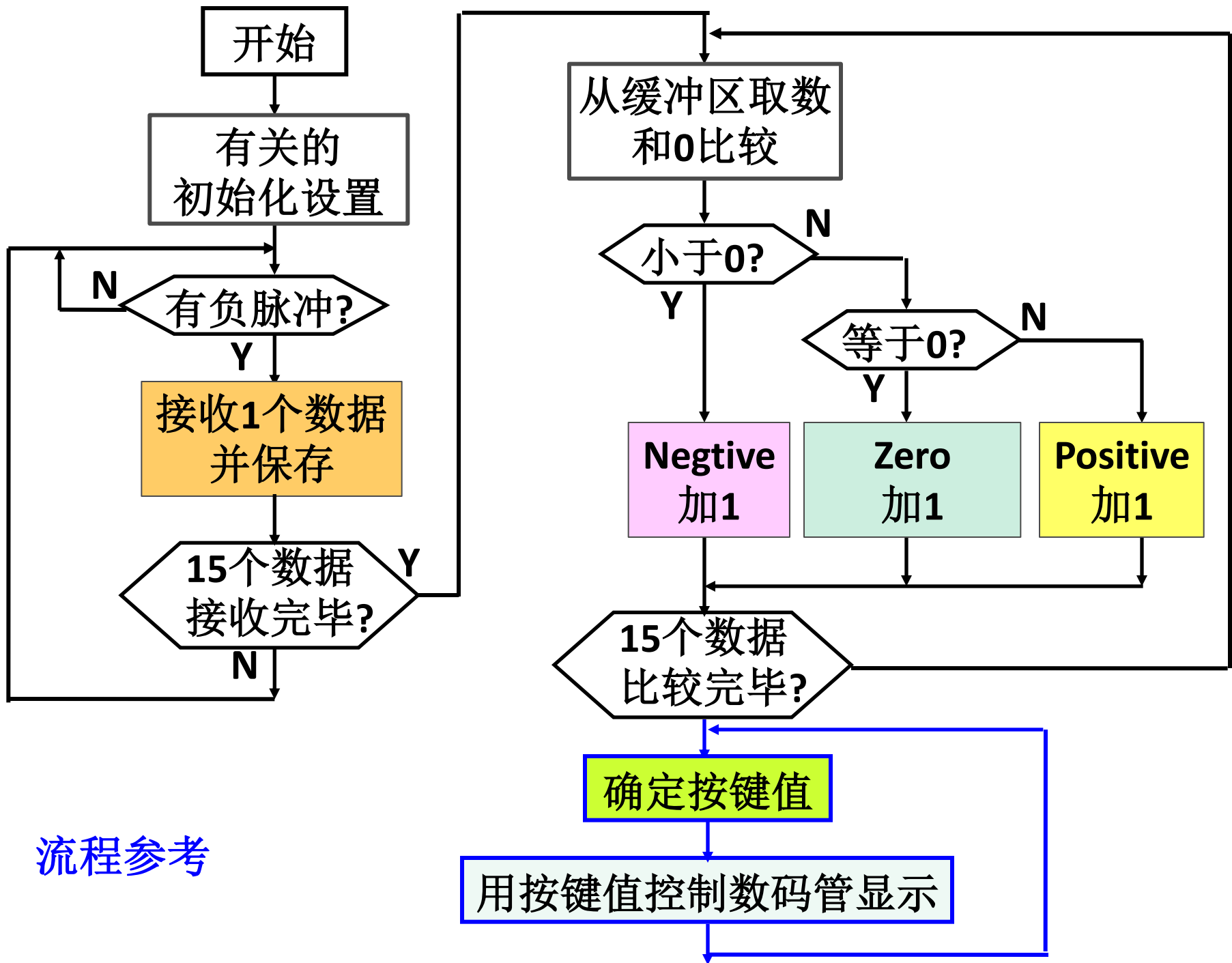
```
} //main函数结束
```



上例基础上，加上数码管显示电路和按键控制电路，如下图，
可实现一个简单的单片机应用系统。

思考：如何编程控制按下K1、K2、K3键，
分别将数据处理的结果通过数码管显示出来，
即按下K1键显示负数的个数，按下K2键显示正数的个数，
按下K3键显示零的个数





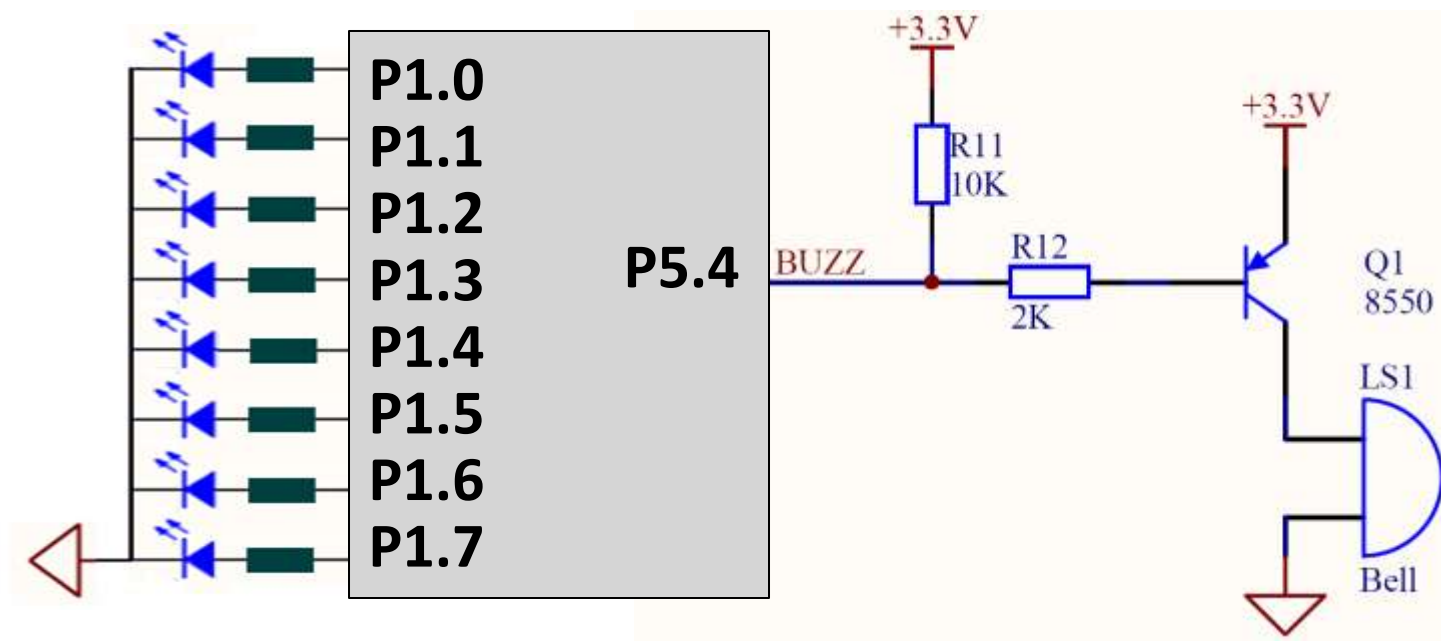
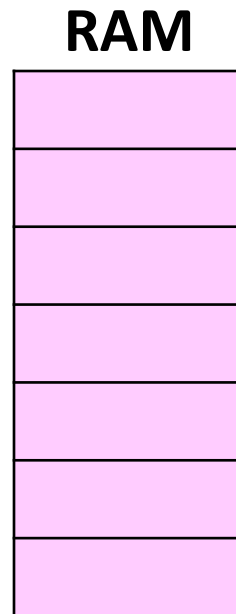
流程参考

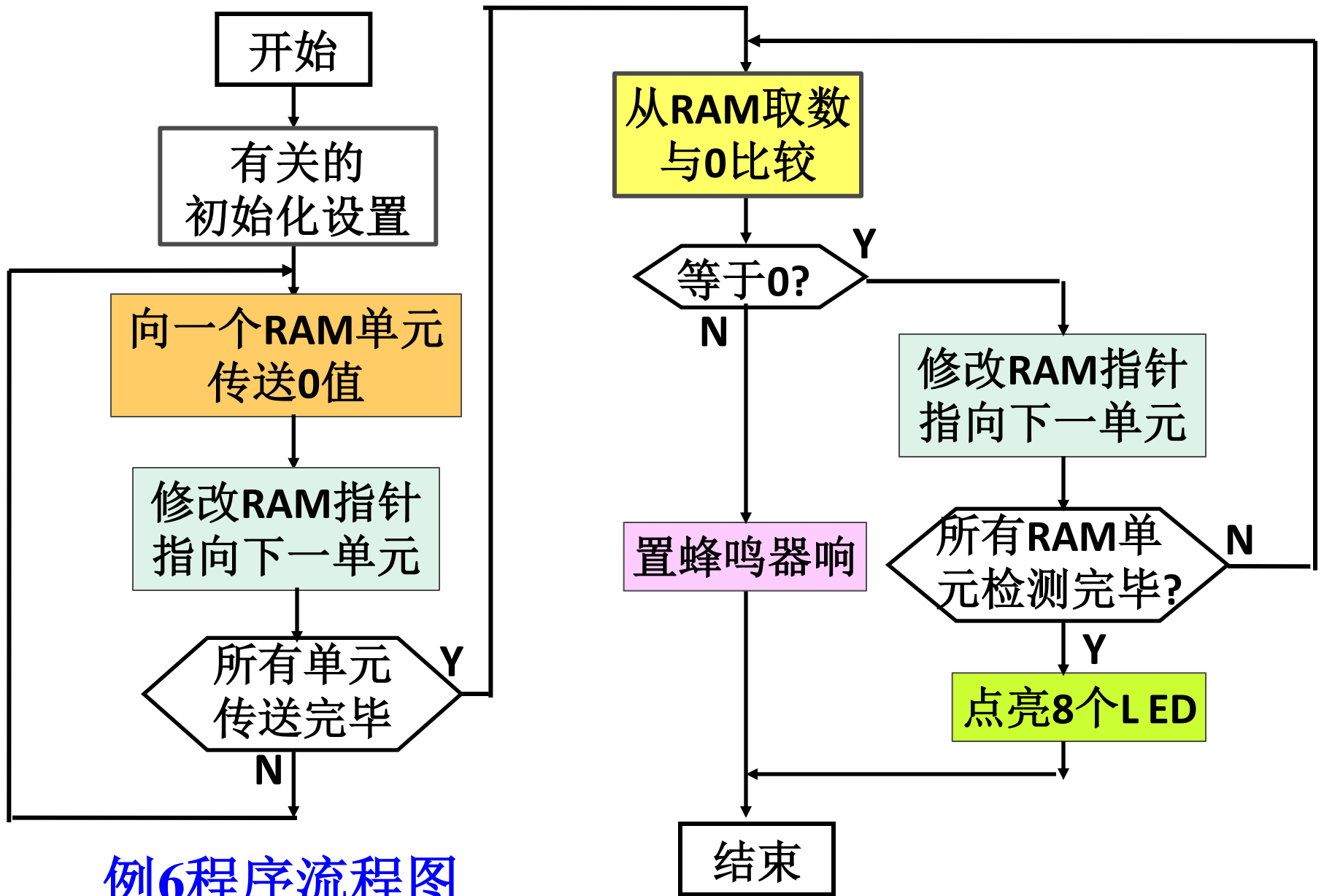
例6. 检测RAM

检测方法:

先向每个单元**填写0**，然后**回读**每个单元的值，
如果发现单元的内容**不为0**，发出**蜂鸣声**报警；
如果所有单元**均为0**，**点亮8个发光二极管**表示。

注意： RAM大小由具体的MCU类型决定

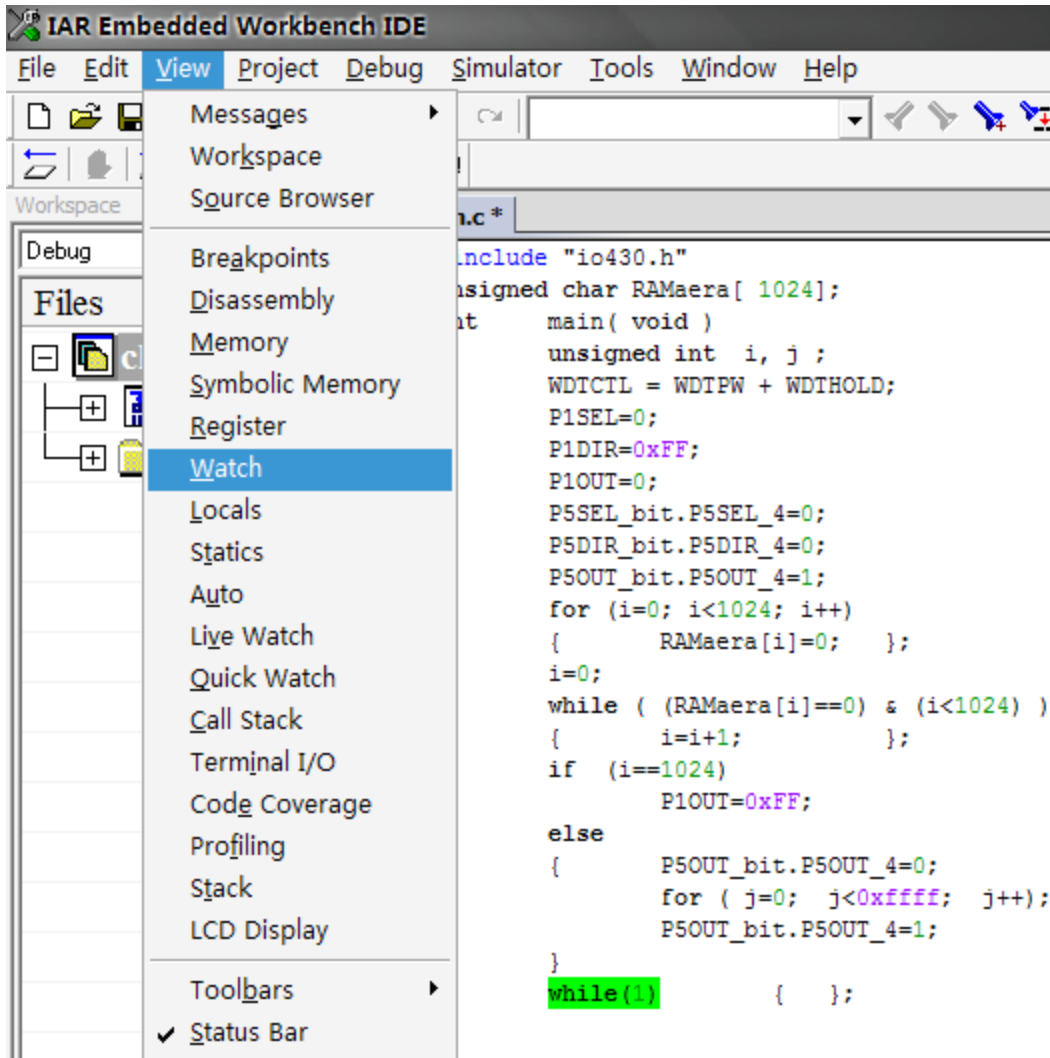




例6程序流程图

例6 检测RAM 程序清单(Chap3_e6.c)

```
#include "io430.h"
unsigned char RAMaera[ 1024];           //定义存储器单元
int  main( void )
{
    unsigned int i, j;
    WDTCTL = WDTPW + WDTTHOLD;         //关闭看门狗
    P1SEL=0;                            //设置P1为基本I/O
    P1OUT=0;                             //使8个LED全灭
    P1DIR=0xFF;                          //设置P1为输出端口
    P5SEL &= ~BIT4;                      //设置P5.4为基本I/O
    P5OUT |= BIT4;                        //设置P5.4为输出1, 停止蜂鸣
    P5DIR |= BIT4;                        //设置P5.4为输出方向
    for (i=0; i<1024; i++)
    {
        RAMaera[i]=0; };                //填充存储器单元为0
    i=0;
    while ( (RAMaera[i]==0) & (i<1024) ) { i=i+1; };
    //检查存储器单元是否为0
    //所有存储器单元全为0, 点亮LED
    //有存储器单元不为0
    if (i==1024) {P1OUT=0xFF;}
    else
    {
        P5OUT &= ~BIT4;                  //启动蜂鸣
        for ( j=0; j<0xffff; j++);      //延时
        P5OUT |= BIT4;                  //停止蜂鸣
    }
    while(1) { };                        //无限循环
}
```

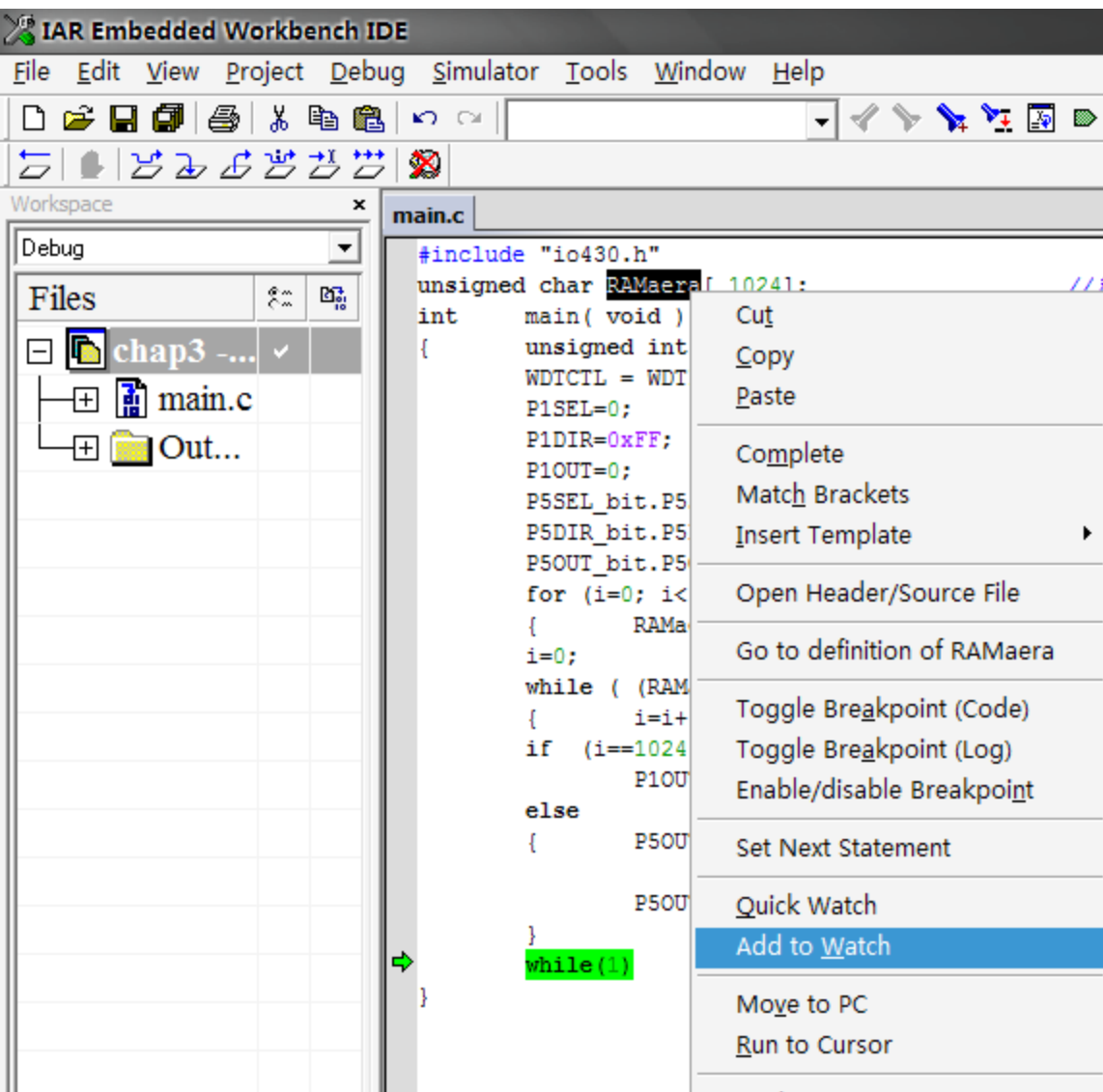


在DEBUG下查看变量 方法1:

1. 点击View/Watch
2. 在弹出的Watch窗口中输入变量的名字, 如RAMaera 即可查看变量的内容和地址

The Watch window is shown with a red arrow pointing to the 'Watch' title bar. The window contains a table with the following data:

Expression	Value	Location
RAMaera	""	Memory:0x200

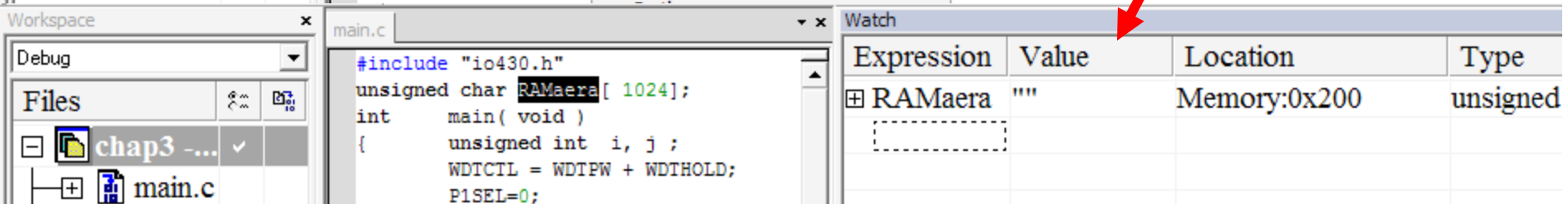


在DEBUG下查看变量(续)

方法2:

1. 选中要查看的变量名, 如RAMaera,
2. 点右键, 选择Add to Watch

即可添加到监视窗口, 查看到变量的内容和地址



在DEBUG下查看变量(续)

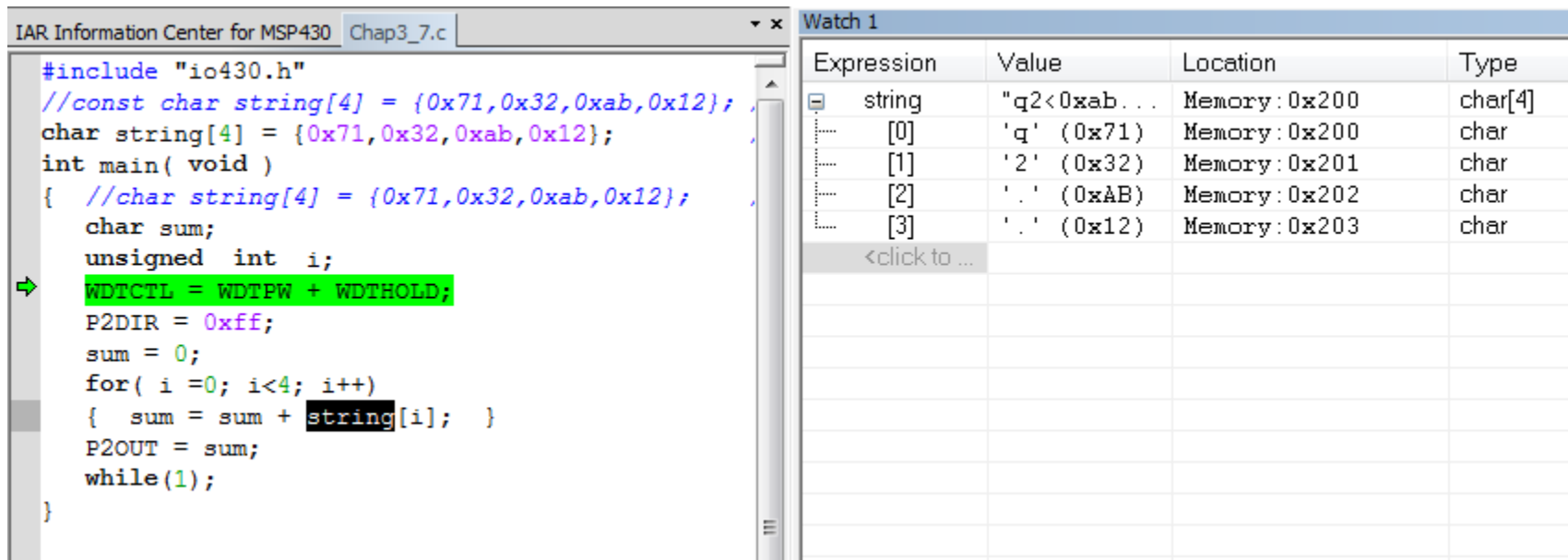
如果变量是数组，
点击变量名前的+，
即可查看都各个单元
的内容和地址

Watch		
Expression	Value	Location
<input checked="" type="checkbox"/> RAMaera	""	Memory:0x200
- [0]	'□' (0x00)	Memory:0x200
- [1]	'□' (0x00)	Memory:0x201
- [2]	'□' (0x00)	Memory:0x202
- [3]	'□' (0x00)	Memory:0x203
- [4]	'□' (0x00)	Memory:0x204
- [5]	'□' (0x00)	Memory:0x205
- [6]	'□' (0x00)	Memory:0x206
- [7]	'□' (0x00)	Memory:0x207
- [8]	'□' (0x00)	Memory:0x208
- [9]	'□' (0x00)	Memory:0x209
- [10]	'□' (0x00)	Memory:0x20A
- [11]	'□' (0x00)	Memory:0x20B
- [12]	'□' (0x00)	Memory:0x20C
- [13]	'□' (0x00)	Memory:0x20D
- [14]	'□' (0x00)	Memory:0x20E
- [15]	'□' (0x00)	Memory:0x20F
- [16]	'□' (0x00)	Memory:0x210
- [17]	'□' (0x00)	Memory:0x211
- [18]	'□' (0x00)	Memory:0x212
- [19]	'□' (0x00)	Memory:0x213

例7. 下面程序中 **string** 是一个带初始化的数组, 程序功能是对该数组求和, 将结果通过端口**P2**输出。(不考虑进位情况) 请在**DEBUG**下查看系统实现变量**string**的初始化方法。

```
#include "io430.h"
char string[4] = {0x71,0x32,0xab,0x12};           //第2种
int main( void )
{ char sum;
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD;
  P2DIR = 0xff;
  sum = 0;
  for( i =0; i<4; i++)
  { sum = sum + string[i]; }
  P2OUT = sum;
  while(1);
}
```

在EW430的DEBUG下，在程序执行用户main前，在Watch窗口下可以看到string的初始值



The screenshot shows the IAR Information Center for MSP430. The left pane displays the source code for 'Chap3_7.c'. The right pane shows the 'Watch 1' window with a table of variables.

```
#include "io430.h"
//const char string[4] = {0x71,0x32,0xab,0x12};
char string[4] = {0x71,0x32,0xab,0x12};
int main( void )
{ //char string[4] = {0x71,0x32,0xab,0x12};
  char sum;
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD;
  P2DIR = 0xff;
  sum = 0;
  for( i =0; i<4; i++)
  { sum = sum + string[i]; }
  P2OUT = sum;
  while(1);
}
```

Expression	Value	Location	Type
string	"q2<0xab...	Memory: 0x200	char[4]
[0]	'q' (0x71)	Memory: 0x200	char
[1]	'2' (0x32)	Memory: 0x201	char
[2]	'.' (0xAB)	Memory: 0x202	char
[3]	'.' (0x12)	Memory: 0x203	char
<click to ...			

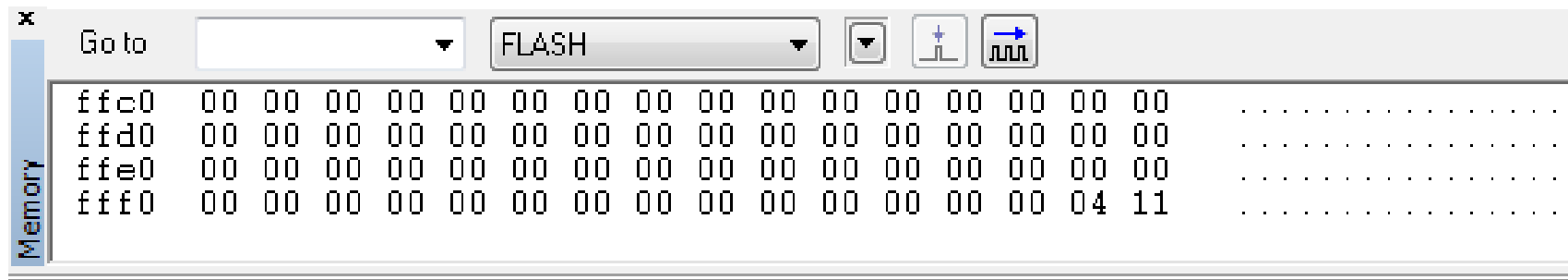
选中变量string, 点击鼠标右键中的add to watch

在DEBUG下,

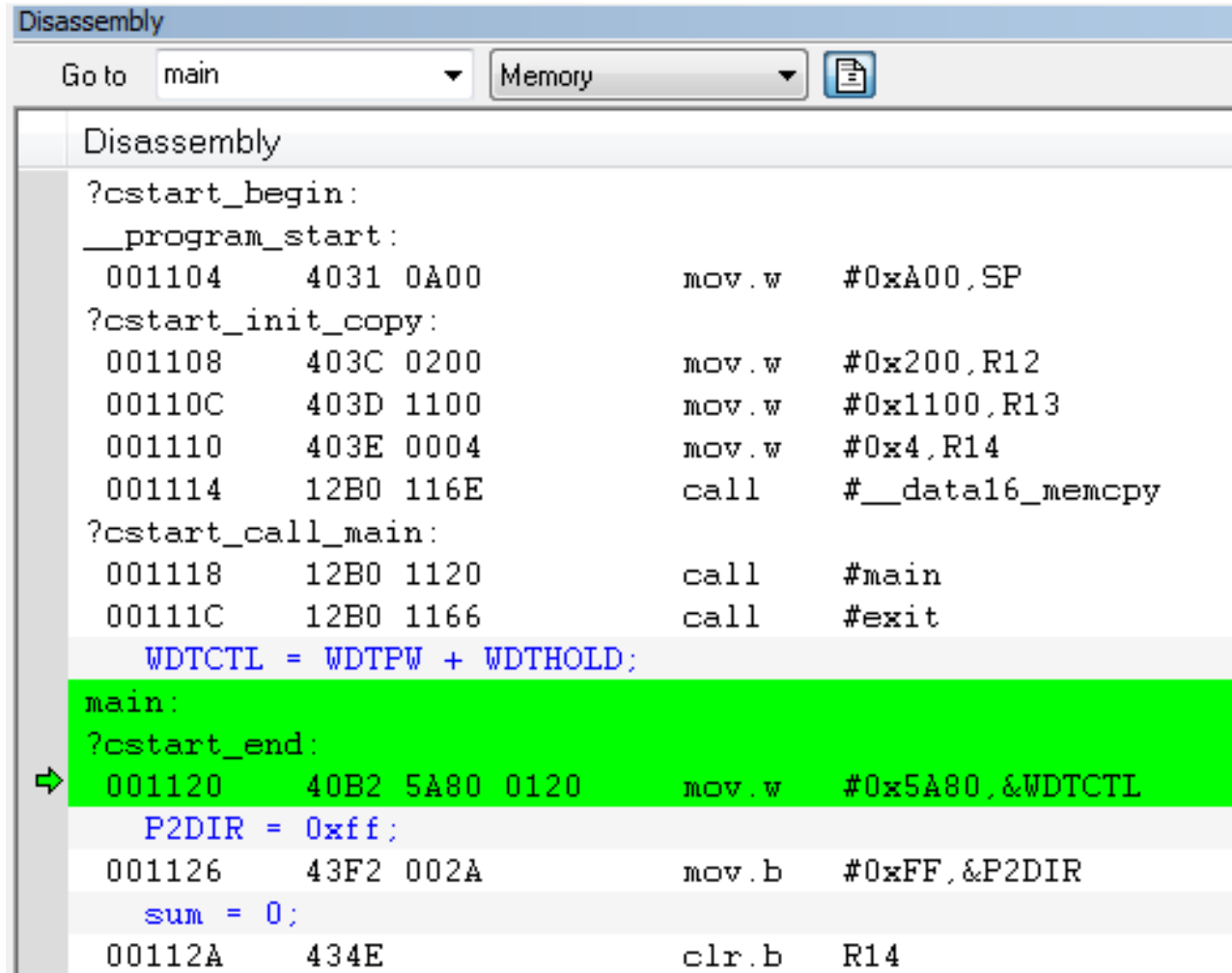
用view/memory 查看0xFFFE~FFFF存放的内容为0x1104,

即上电后的复位地址是0x1104,

单片机从此处开始取指执行程序

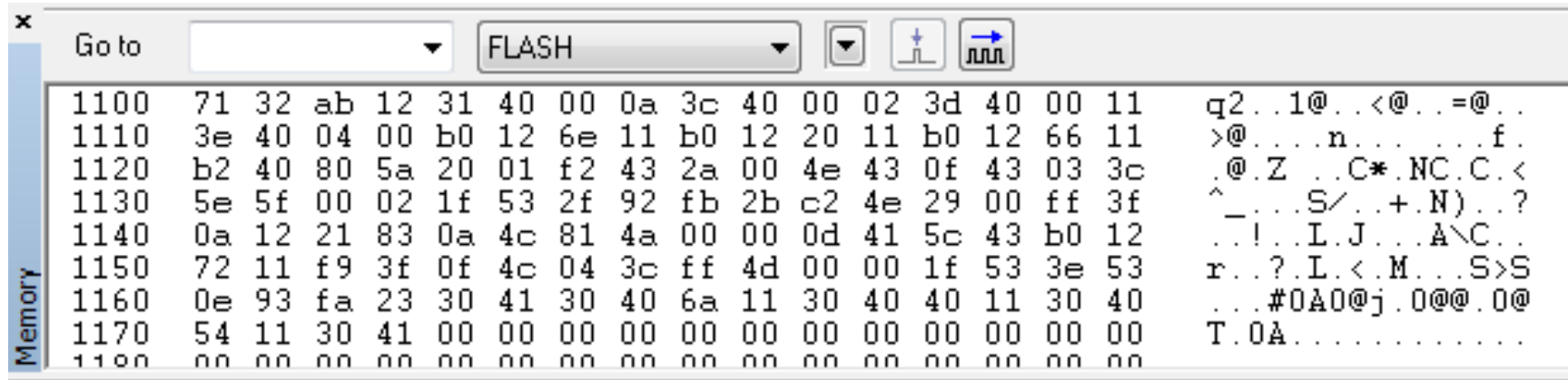


将PC值改为0x1104, 并反汇编该段存储单元内容,
可以看到系统编写的copy段程序,
功能是将0x1100开始的0x4个存储单元内容拷贝到
0x0200开始存储单元的中



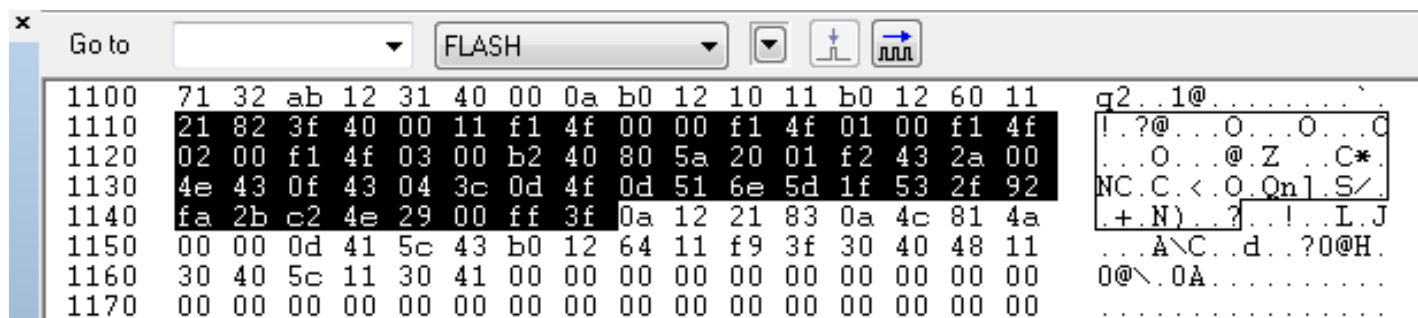
```
Disassembly
Go to main Memory
Disassembly
?cstart_begin:
__program_start:
001104 4031 0A00      mov.w    #0xA00, SP
?cstart_init_copy:
001108 403C 0200      mov.w    #0x200, R12
00110C 403D 1100      mov.w    #0x1100, R13
001110 403E 0004      mov.w    #0x4, R14
001114 12B0 116E      call    #__data16_memcpy
?cstart_call_main:
001118 12B0 1120      call    #main
00111C 12B0 1166      call    #exit
    WDTCTL = WDTPW + WDT HOLD;
main:
?cstart_end:
➔ 001120 40B2 5A80 0120  mov.w    #0x5A80, &WDTCTL
    P2DIR = 0xff;
001126 43F2 002A      mov.b    #0xFF, &P2DIR
    sum = 0;
00112A 434E          clr.b    R14
```

用view/memory 查看0x1100存放的内容，
可以看到存放有string的初始值“0x71,0x32,0xab,0x12”



```
Go to [ ] FLASH [ ] [ ] [ ] [ ]
Memory
1100 71 32 ab 12 31 40 00 0a 3c 40 00 02 3d 40 00 11 q2..1@..<@..=@..
1110 3e 40 04 00 b0 12 6e 11 b0 12 20 11 b0 12 66 11 >@...n...f.
1120 b2 40 80 5a 20 01 f2 43 2a 00 4e 43 0f 43 03 3c .@.Z ..C*.NC.C.<
1130 5e 5f 00 02 1f 53 2f 92 fb 2b c2 4e 29 00 ff 3f ^_...S/..+.N)..?
1140 0a 12 21 83 0a 4c 81 4a 00 00 0d 41 5c 43 b0 12 ..!.L.J...A\C..
1150 72 11 f9 3f 0f 4c 04 3c ff 4d 00 00 1f 53 3e 53 r..?.L.<.M...S>S
1160 0e 93 fa 23 30 41 30 40 6a 11 30 40 40 11 30 40 ...#0A0@j.0@@.0@
1170 54 11 30 41 00 00 00 00 00 00 00 00 00 00 00 00 T.0A.....
1180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

思考： 系统在执行用户程序前，
为何要添加这段copy程序段？
没有这段copy程序段，会有何不同？



```
Go to [ ] FLASH [ ] [ ] [ ] [ ]
Memory
1100 71 32 ab 12 31 40 00 0a b0 12 10 11 b0 12 60 11 q2..1@.....
1110 21 82 3f 40 00 11 f1 4f 00 00 f1 4f 01 00 f1 4f !.?.@...O...O...C
1120 02 00 f1 4f 03 00 b2 40 80 5a 20 01 f2 43 2a 00 ...O...@.Z ..C*.
1130 4e 43 0f 43 04 3c 0d 4f 0d 51 6e 5d 1f 53 2f 92 NC.C.<.O.Qn|S/
1140 fa 2b c2 4e 29 00 ff 3f 0a 12 21 83 0a 4c 81 4a .+.N)..?.!.L.J
1150 00 00 0d 41 5c 43 b0 12 64 11 f9 3f 30 40 48 11 ...A\C..d..?0@H.
1160 30 40 5c 11 30 41 00 00 00 00 00 00 00 00 00 00 00@.\.0A.....
1170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

由于**MCU**内部没有操作系统，
复位后即开始执行用户程序的话，
存放在**RAM**区中的变量的值是随机的，
不能确保每次上电后变量的初始值，
如何完成带初始值的变量设置呢？

解决办法：

编译系统将所有带初始化变量的初值放在**Flash ROM**中，
然后通过执行**copy**程序段，将其传送到**RAM**中定义的变量中，
后续程序可以引用**RAM**中的已有初值的变量。

思考：请分别在DEBUG下查看系统实现对下面2种定义初始化的方法。比较与第1种的实现有何不同？

```
#include "io430.h"
char string[4] = {0x71,0x32,0xab,0x12};           //第1种
//const char string[4] = {0x71,0x32,0xab,0x12}; //第2种
int main( void )
{ //char string[4] = {0x71,0x32,0xab,0x12};       //第3种
  char sum;
  unsigned int i;
  WDTCTL = WDTPW + WDTHOLD;
  P2DIR = 0xff;
  sum = 0;
  for( i =0; i<4; i++)
  { sum = sum + string[i]; }
  P2OUT = sum;
  while(1);
}
```