一、实验目的

1. 学习与理解 51 单片机的结构、工作原理;
2. 掌握单片机定时器和中断系统相关知识;
3. 进一步熟悉 51 单片机开发环境，掌握调试方法。

二、实验任务

在实验板上实现实时数字钟和万年历。要求

1) “hh-mm-ss”方式显示时钟;
2) 可以通过按键置入时间值;
3) 显示日期时可以从右向左滚动显示“yyyy-mm-dd”三次。

三、实验完成效果

完成数字钟和万年历，包含显示和设置功能，共使用三个按键（mode 键（三行一列），set 键（三行二列）和 increment（三行三列）键），共有四个模式:

1) 数字钟显示。按照“hh-mm-ss”(如 23-59-59)方式显示 24 小时制的时间。数字钟默认处于此模式。
2) 日期显示。按照“yyyy-mm-dd ”（如 2020-02-29 ）从左到右滚动显示三次。在此模式下,
   a) 如果不按下任何按键，将在循环显示三次后回到数字钟显示模式;
   b) 若在循环时按下 set 键或 increment 键也将退出此模式并回到“数字钟显示”模式;
   c) 若按下 mode 键，将进入“数字钟设置”模式。
3) 数字钟设置。七段数码管按照”hh-mm-ss”方式显示时间，可以通过 set 键调整要设置的位置（分钟位或小时位），要设置的位置会每隔 128 毫秒点亮 128 毫秒实现闪烁提示，按下 set 键可以切换闪烁位，按下 increment 键可以调整该位的数值，每按一次该位值+1。在此模式下，溢出不会导致进位。在此模式再按下 mode 键将进入“日期设置”模式。
4) 日期设置模式。七段数码管按”yyyymmdd”（如 2020-02-29）显示日期，其中有一位在闪烁，表示将调整该位。按下 set 键可以切换闪烁位，按下 increment 键可以调整该位数值，每按一次该位值+1。在此模式下按下 mode 键将进入数字钟显示模式。

四、实验原理

4.1 定时器

4.1.1 定时器结构

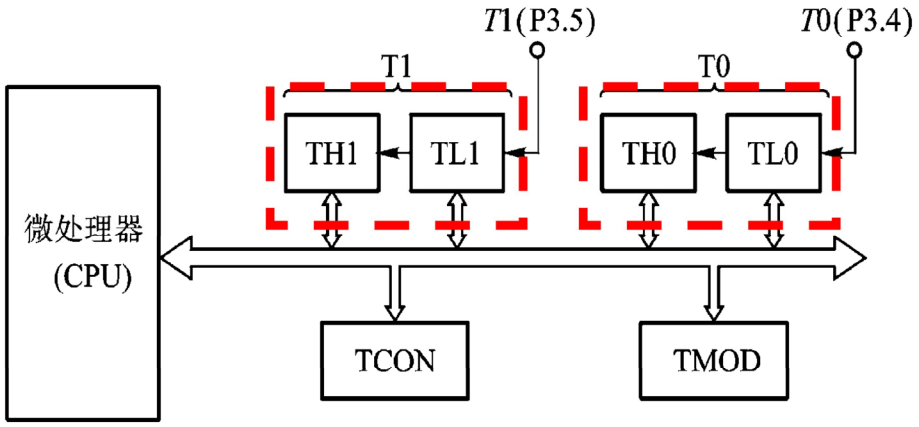

图 1 定时器结构

8051 单片机定时器主要有几个特殊功能寄存器组成：两个 16 位的定时器 T0 和 T1，以及他们的工作方式寄存器 TMOD 和控制寄存器 TCON 等。定时器 T0 和 T1 各由两个 8 位特殊功能寄存器 TH0、TL0、TH1、TL1 构成，它们用来存放定时初值或计数结果。工作方式寄存器 TMOD 用于设置定时器的工作模式和工作方式；控制寄存器 TCON 用于启动和停止定时器的计数，并控制定时器的状态。

4.1.2 工作方式寄存器 TMOD

工作方式寄存器 TMOD 用于设置定时/计数器的工作方式，低四位用于 T0，高四位用于 T1。其格式如下：

位	7	6	5	4	3	2	1	0	
字节地址：89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	TMOD

图 2 TMOD 寄存器

GATE 为门控位。GATE = 0 时，只要用软件使 TCON 中的 TR0 或 TR1 为 1，就可以启动定时/计数器工作，GATA = 1 时，要用软件使 TR0 或 TR1 为 1，同时外部中断引脚或也为高电平时，才能启动定时/计数器工作。即此时定时器的启动多了一条件。本实验中令 GATE=0 即可。

C/T 为定时/计数模式选择位。C/T = 0 为定时模式，C/T=1 为计数模式。

M1M0：工作方式设置位。定时/计数器有四种工作方式，

定时/计数器工作方式设置表

M1M0	工作方式	说　　明
00	方式 0	13 位定时/计数器
01	方式 1	16 位定时/计数器
10	方式 2	8 位自动重装定时/计数器
11	方式 3	T0 分成两个独立的 8 位定时/计数器；T1 此方式停止计数

图 3 定时器工作方式

4.1.3 控制寄存器 TCON

TCON（88 H）

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

图 4 控制寄存器 TCON

1) TF1、TF0 是计数溢出标志位、中断请求标志位。定时器 T0 或 T1 计数溢出时，由硬件自动将此位置“1”；TFx 可以由程序查询，也是定时中断的请求源。
2) TR1、TR0 为计数运行控制位 TRx=1: 启动定时器/计数器工作；TRx=0: 停止定时器/计数器工作。
3) IT0（TCON.0）：外部中断 0 触发方式控制位。当 IT0=0 时，为电平触发方式；当 IT0=1 时，为边沿触发方式（下降沿有效）。
4) IE0（TCON.1）：外部中断 0 中断请求标志位
5) IT1（TCON.2）：外部中断 1 触发方式控制位
6) IE1（TCON.3）：外部中断 1 中断请求标志位

4.1.4 定时器工作模式

1) 计数器工作模式 对外部事件进行计数。计数脉冲来自相应的外部输入引脚 T0（P3.4)或 T1(P3.5)。当输入信号发生由 1 至 0 的负跳变(下降沿)时，计数器（TH0，TL0 或 TH1，TL1）的值增 1。
2) 定时器工作模式。通过计数实现，计数脉冲来自内部时钟脉冲，每个机器周期计数值增 1，每个机器周期=12 个振荡周期，因此计数频率为振荡频率的 1/12。所以定时时间=计数值×机器周期。

本次实验中主要使用定时器工作模式中的工作方式 1。

4.2 中断

4.2.1 中断的概念

中断是指计算机运行过程中，出现某些意外情况需主机干预时，机器能自动停止正在运行的程序并转入处理新情况的程序，处理完毕后又返回原被暂停的程序继续运行。

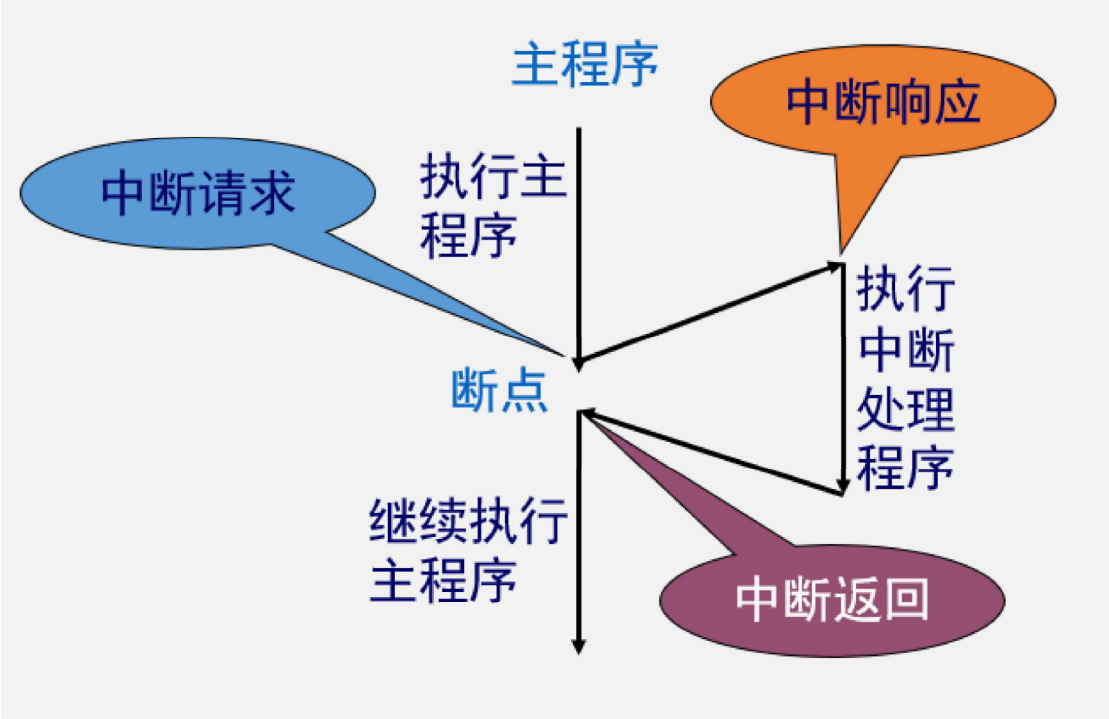

图 5 中断的概念

4.2.2 80C51 的中断系统

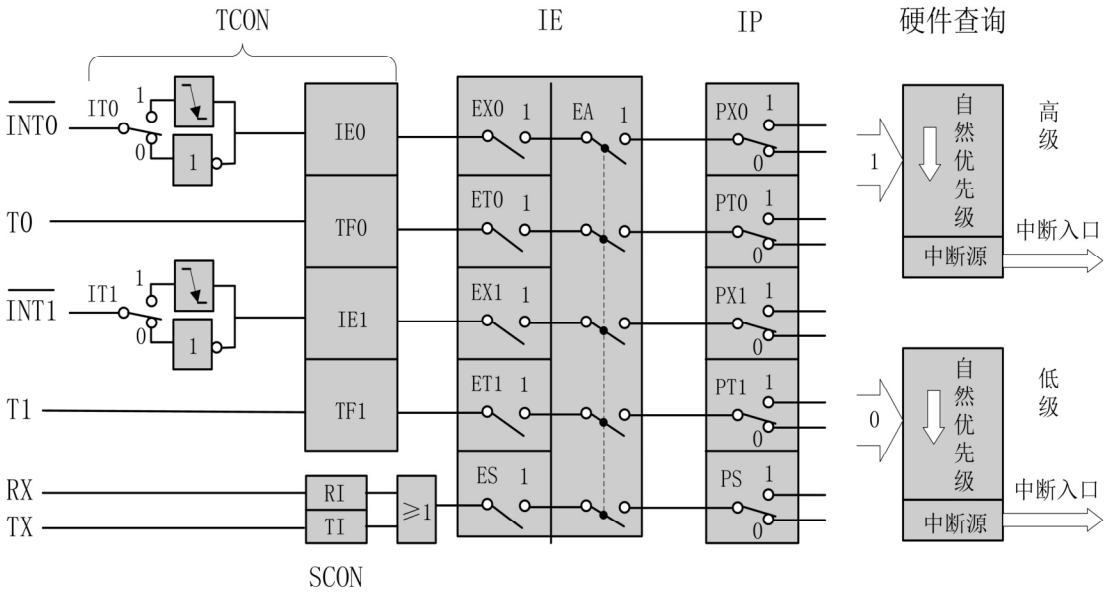

图 6 80C51 的中断系统

CPU 对中断系统所有中断以及某个中断源的开放和屏蔽是由中断允许寄存器 IE 控制的。

位	7	6	5	4	3	2	1	0	
字节地址：A8H	EA			ES	ET1	EX1	ET0	EX0	IE

图 7 中断允许寄存器 IE

① EX0(IE.0)，外部中断 0 允许位；

② ET0(IE.1)，定时/计数器 T0 中断允许位；

③ EX1(IE.2)，外部中断 0 允许位；

④ ET1(IE.3)，定时/计数器 T1 中断允许位；

⑤ ES(IE.4)，串行口中断允许位；

⑥ EA(IE.7)，CPU 中断允许（总允许）位

80C51 单片机有两个中断优先级，即可实现二级中断服务嵌套。每个中断源的中断优先级都是由中断优先级寄存器 IP 中的相应位的状态来规定的 。

位	7	6	5	4	3	2	1	0	
字节地址：B8H			PT2	PS	PT1	PX1	PT0	PX0	IP

图 8 中断优先级寄存器 IP

a) PX0（IP.0）外部中断 0 优先级设定位；

b) PT0（IP.1），定时/计数器 T0 优先级设定位；

c) PX1（IP.2），外部中断 0 优先级设定位；

d) PT1（IP.3），定时/计数器 T1 优先级设定位；

e) PS（IP.4），串行口优先级设定位；

f) PT2 (IP.5)，定时/计数器 T2 优先级设定位。

五、程序设计及流程图：

5.1 需求及利用资源分析：

总共需要四种模式：数字钟显示，日期显示，数字钟设置，日期设置。每种模式需要不同的 LED 显示方式和按键处理方式。

定时器每次定时最高实现约 65ms，因此设置定时器 0 计时 50ms，计 20 个 50ms 即为 1s。

矩阵键盘的定时扫描、LED 动态显示时各位的切换、设置模式下设置位的闪烁、日期显示模式下的滚屏速率均需要定时或延时，考虑到通过 for 延时记不准确也占用 CPU，因此全部用定时器+中断方式实现。矩阵键盘每隔 50ms 扫描一次按键，使用定时器 0；动态扫描时各位的切换、设置位的闪烁和滚屏等待时间均由定时器 1 计时若干个 2ms 实现。

为实现模式切换、设置位切换、设置为加一，共需要三个按键。

5.2 流程图：

5.2.1 主函数流程图

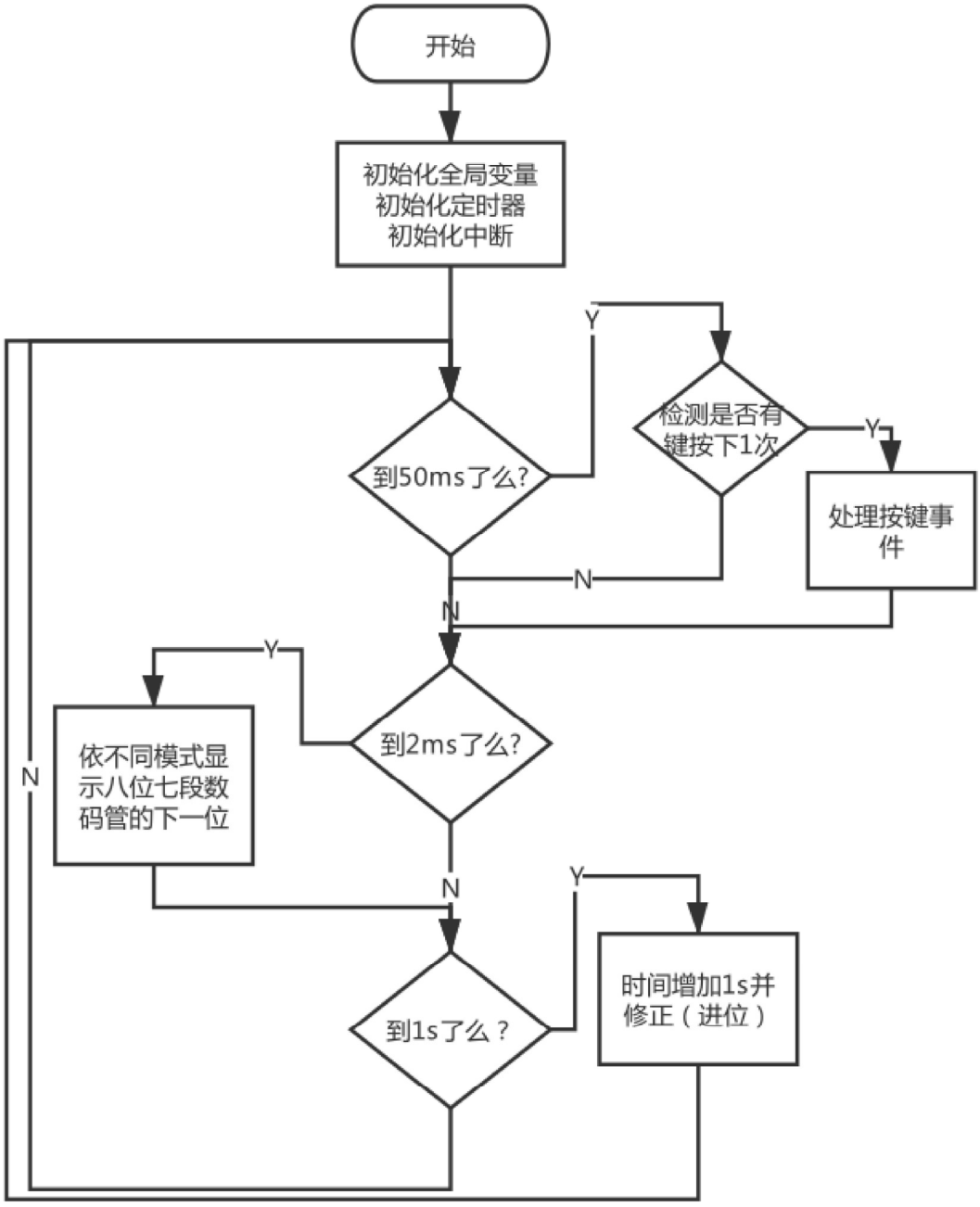

图 9 主函数流程图

5.2.1 处理按键事件流程图

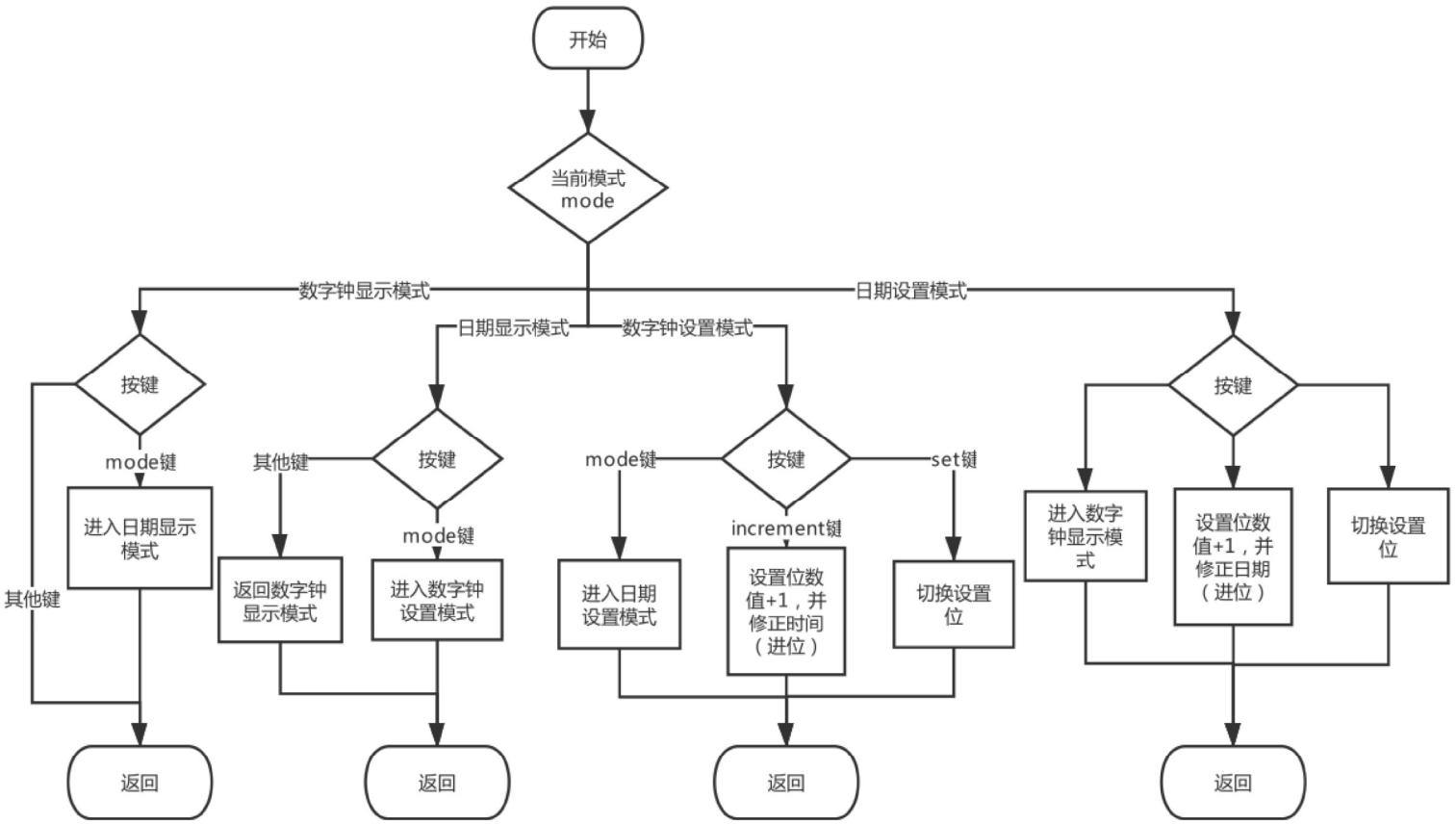

图 10 处理按键事件流程图

六、资源使用情况:

5.1 存储器使用情况:

Program Size: data=50.0 xdata=0 code=1912

5.2 按键使用情况:

使用三个按键分别用作：mode 键、set 键和 increment 键

5.3 定时器及中断使用情况:

使用 timer0、timer1 及对应的中断 1、中断 3。

七、源码:

```
#include <REG51.H>
#include <absacc.h>

#define TIMER0_SECOND_COUNT 20

typedef struct Time
{
    unsigned int year;
    unsigned char month;
    unsigned char day;
```

```
	unsigned char hour;
	unsigned char minute;
	unsigned char second;
	unsigned char isLeap;
} time;

// from 0 to 9, and blank, _, -, E
unsigned char code led_table[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x08,0x40,0x79 };

unsigned char G_timer0_count = 0;
unsigned char G_timer0_flag = 0;
unsigned char G_timer1_count = 0;
unsigned char G_timer1_flag = 0;

unsigned char G_keycode = 0;
unsigned char G_key_pressing_flag = 0;

unsigned char G_mode = 0;

unsigned char G_roling_shift = 0;

unsigned char G_setting_bit = 0;

unsigned char G_max_day[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

unsigned char G_time_array[8] = { 0,0,12,0,0,12,0,0 };
unsigned char G_date_array[12] = { 2,0,1,9,12,0,1,12,0,1,10,10 };

time G_current_time = { 2019,1,1,0,0,0};
time *G_pointer_current_time = &(G_current_time);

void timeToArray(time *ptr);

/********************/

unsigned char isLeap(time *ptr) {
	if ((ptr->year % 400 == 0) || ((ptr->year % 4 == 0) && (ptr->year % 100 != 0))) {
		return 1;
	}
	else {
		return 0;
	}
}

// this function has no use.
void delayByMs(unsigned char time) {
	static unsigned char i;
	for (; time > 0; time--);
	for (i = 0; i < 440; i++);
}

unsigned char getKeycode(void) {
	unsigned char line = 0x00;		/*行码*/
	unsigned char col = 0x00;		/*列码*/
	unsigned char scancode = 0x01;	/*行扫描码*/
	unsigned char keycode;		/*键号*/
	XBYTE[0x8000] = 0xff;
	col = XBYTE[0x8000] & 0x0f;		/*从列端口读入四位列码*/
	if (col == 0x00) {
		keycode = 0x00;
	}
	else
	{
```

```
        while ((scancode & 0x0f) != 0) /*取 scancode 的低四位，没变为全 0，循环*/
        {
            line = scancode;                    /*行号*/
            XBYTE[0x8000] = scancode;           /*给行赋扫描码，第一行为 0x01*/
            if ((XBYTE[0x8000] & 0x0f) == col) /*检测按键所在的行跳出循环*/
                break;
            scancode = scancode << 1;           /*行扫描码左移一位，转下一行*/
        }
        col = col << 4;                         /*把行码移到高四位*/
        keycode = col | line;
    }
    return keycode;
}

/*
* correct the time after time changed
*/
time *timeCorrection(time *ptr,unsigned char setting_mode) {
    if (!setting_mode) {
        if (ptr->second > 59) {
            ptr->second = 0;
            ptr->minute++;
        }

        if (ptr->minute >= 60) {
            ptr->minute = 0;
            ptr->hour++;
        }

        if (ptr->hour >= 24) {
            ptr->hour = 0;
            ptr->day++;
        }

        if (ptr->day > G_max_day[ptr->month - 1]) {
            ptr->day = 1;
            ptr->month++;
        }

        if (ptr->month > 12) {
            ptr->month = 1;
            ptr->year++;
            if (isLeap(ptr)) {
                ptr->isLeap = 1;
                G_max_day[1] = 29;
            }
            else {
                ptr->isLeap = 0;
                G_max_day[1] = 28;
            }
        }
    }
    else {
        if (ptr->second >= 60) {
            ptr->second -= 60;
        }

        if (ptr->minute >= 60) {
            ptr->minute -= 60;
        }

        if (ptr->hour >= 24) {
            ptr->hour -= 24;
```

```
            }

            if (ptr->day > G_max_day[ptr->month - 1]) {
                ptr->day = 1;
            }

            if (ptr->month > 12) {
                ptr->month = 1;
            }

            if (isLeap(ptr)) {
                ptr->isLeap = 1;
                G_max_day[1] = 29;
            }
            else {
                ptr->isLeap = 0;
                G_max_day[1] = 28;
            }
        }
        return ptr;
}

void timeToArray(time *ptr) {
        G_time_array[7] = ptr->second % 10;
        G_time_array[6] = ptr->second / 10;
        G_time_array[4] = ptr->minute % 10;
        G_time_array[3] = ptr->minute / 10;
        G_time_array[1] = ptr->hour % 10;
        G_time_array[0] = ptr->hour / 10;
}

void dateToArray(time *ptr) {
        unsigned int temp = 0;
        G_date_array[9] = ptr->day % 10;
        G_date_array[8] = ptr->day / 10;

        G_date_array[6] = ptr->month % 10;
        G_date_array[5] = ptr->month / 10;

        G_date_array[0] = ptr->year / 1000;
        G_date_array[1] = (ptr->year % 1000) / 100;
        G_date_array[2] = (ptr->year % 100) / 10;
        G_date_array[3] = ptr->year % 10;
}

/*
* when 1s passes, call this function to increase the second
*/
time *timeAutoIncrement(time *ptr) {
        ptr->second++;
        timeCorrection(ptr, 0);

        timeToArray(ptr);
        dateToArray(ptr);
        return ptr;
}

//call this function every timer1 break in mode 0
void displayTime() {
        unsigned char shift;
        shift = G_timer1_count % 8;
        XBYTE[0x8000] = 0x80 >> shift;
        XBYTE[0x9000] = led_table[G_time_array[shift]];
```

```
}

// mode 1
void displayDateRoling() {
	unsigned char bit_to_show_in_date_array;
	unsigned char bit_to_show_in_eight_leds;
	if (G_timer1_count % 350 == 0) {
		G_roling_shift++;
		if (G_roling_shift == 12 * 3) {
			G_roling_shift = 0;
			G_mode = 0;
		}
	}
	bit_to_show_in_eight_leds = G_timer1_count % 8;
	bit_to_show_in_date_array = (G_roling_shift + bit_to_show_in_eight_leds) % 12;
	XBYTE[0x8000] = 0x80 >> bit_to_show_in_eight_leds;
	XBYTE[0x9000] = led_table[G_date_array[bit_to_show_in_date_array]];
}

void displayTimeBlink(unsigned char shift_bit){
	unsigned char shift;
	unsigned char blink = 0;
	if ((G_timer1_count & 0x40) == 0x40) {
		blink ^= 0x01;
	}
	shift = G_timer1_count % 8;
	XBYTE[0x8000] = 0x80 >> shift;
	if (shift_bit == 0) {
		if (shift == 3 || shift == 4) {
			XBYTE[0x9000] = blink ? led_table[G_time_array[shift]] : 0x00;
		}
		else
		{
			XBYTE[0x9000] = led_table[G_time_array[shift]];
		}
	}
	else
	{
		if (shift_bit == 1) {
			if (shift == 0 || shift == 1) {
				XBYTE[0x9000] = blink ? led_table[G_time_array[shift]] : 0x00;
			}
			else
			{
				XBYTE[0x9000] = led_table[G_time_array[shift]];
			}
		}
	}

}

void displayDateBlink(unsigned char shift_bit) {
	//unsigned char G_date_array[12] = { 2,0,1,9,12,0,1,12,0,1,10,10 };
	unsigned char shift_in_leds;
	unsigned char shift_in_array;
	unsigned char blink = 0;
	if ((G_timer1_count & 0x40) == 0x40) {
		blink ^= 0x01;
	}
	shift_in_leds = G_timer1_count % 8;
	switch (shift_in_leds)
	{
	case 0:case 1:case 2: case 3:
```

```
            shift_in_array = shift_in_leds;
            break;
        case 4:case 5:
            shift_in_array = shift_in_leds + 1;
            break;
        case 6:case 7:
            shift_in_array = shift_in_leds + 2;
            break;
        default:
            break;
        }
        XBYTE[0x8000] = 0x80 >> shift_in_leds;
        if (shift_in_leds == 7 - shift_bit) {
            XBYTE[0x9000] = blink ? led_table[G_date_array[shift_in_array]] : 0x00;
        }
        else {
            XBYTE[0x9000] = led_table[G_date_array[shift_in_array]];
        }
}

unsigned char modeSwitch(unsigned char current_mode) {
        /*
            0 for displaying time, hh-mm-ss
            2 for setting time, hh-mm-ss , make the bit being changed blink
            3 for setting date, yyyymmdd,
            4 for setting clock.

            1 for showing date, yyyy-mm-dd,roling
        */

        if (current_mode == 0) {
            return 1;
        }
        else
        {
            if (current_mode == 1) {
                G_roling_shift = 0;
            }
            else {
                G_setting_bit = 0;
            }
            return (current_mode + 1) % 4 ;
            // todo, make this 5 later
        }
}

time *setTime(time *ptr, unsigned char shift_bit) {
        switch (shift_bit)
        {
        case 0:
            ptr->minute++;
            break;
        case 1:
            ptr->hour++;
            break;
        default:
            break;
        }
        ptr->second = 0;
        return timeCorrection(ptr, 1);
}

time *setDate(time *ptr, unsigned char shift_bit) {
```

```
        switch (shift_bit)
        {
        case 0:
                ptr->day++;
                break;
        case 1:
                ptr->day += 10;
                break;
        case 2:
                ptr->month++;
                break;
        case 3:
                ptr->month += 10;
                break;
        case 4:
                ptr->year++;
                break;
        case 5:
                ptr->year += 10;
                break;
        case 6:
                ptr->year += 100;
                break;
        case 7:
                ptr->year += 1000;
                break;
        default:
                break;
        }
        return timeCorrection(ptr, 1);
}

void dealWithKeyPressed(unsigned char keycode) {
        if (keycode == 0x14) { // row 3 col 1
                G_mode = modeSwitch(G_mode);
        }
        else {
                if (keycode == 0x24) { //row 3 col 2
                        if (G_mode == 2) {
                                G_setting_bit = (G_setting_bit + 1) % 2;
                        }
                        else {
                                if (G_mode == 3) {
                                        G_setting_bit = (G_setting_bit + 1) % 8;    // will compiler optimize %2, %8 ? to
&0x01,0x07 ?
                                }
                        }
                }
                else {
                        if (keycode == 0x44) { //row 3 col 3
                                if (G_mode == 2) {
                                        setTime(G_pointer_current_time, G_setting_bit);
                                }
                                else {
                                        if (G_mode == 3) {
                                                setDate(G_pointer_current_time, G_setting_bit);
                                        }
                                }
                                timeToArray(G_pointer_current_time);
                                dateToArray(G_pointer_current_time);
                        }
                }
                if (G_mode == 1) {
```

```
                G_roling_shift = 0;
                G_mode = 0;
            }
        }
}

/*
*       excute different routine work in different mode
*/
void excute(unsigned char mode) {
        switch (mode)
        {
        case 0:
                displayTime();
                break;
        case 1:
                displayDateRoling();
                break;
        case 2:
                displayTimeBlink(G_setting_bit);
                break;
        case 3:
                displayDateBlink(G_setting_bit);
                break;
        default:
                XBYTE[0x8000] = 0x40 ;
                XBYTE[0x9000] = led_table[mode];
                break;
        }
}

void timerAndInterruptInit(){
        EA = 1;                 // enable global interrupt

        TMOD = 0x11;  // set timer0 and timer1 to mode 1

        // (65535-X)*12/11.0592us = 50ms; X=19455;
        TH0 = 0x4B;             // load 19455D to timer0
        TL0 = 0xFF;
        TR0 = 1;          // start timer0
        ET0 = 1;          // enable timer0 interrupt

        // (65535-X)*12/11.0592us=2ms; X=63691;
        TH1 = 0xF8;             // load 63691 to timer1  ,  2ms
        TL1 = 0xCB;
        TR1 = 1;          // start timer0
        ET1 = 1;          // enable timer0 interrupt
}

void main() {
        unsigned char keycode = 0;
        unsigned char key_input_break = 0;

        timerAndInterruptInit();

        while(1) {

                //1s has passed
                if (G_timer0_count == TIMER0_SECOND_COUNT) {
                        timeAutoIncrement(G_pointer_current_time);
                        G_timer0_count = 0;
                }
```

```
            //50ms passed, check if any key being pressed
            if (G_timer0_flag) {
                keycode = getKeycode();
                if (keycode) {
                    if (G_key_pressing_flag) {
                        G_keycode = 0;
                    }
                    else {
                        G_key_pressing_flag = 1;
                        G_keycode = keycode;
                    }
                }
                else {
                    G_key_pressing_flag = 0;
                }
                G_timer0_flag = 0;
            }
            // 2ms passed , excute routine work(display next 7-seg-led)
            if (G_timer1_flag) {
                excute(G_mode);
                G_timer1_flag = 0;
            }

            //if key pressed
            if (G_keycode) {
                dealWithKeyPressed(G_keycode);
                G_keycode = 0;
            }
        }
        // never reach here;
        return ;
}

void timer0() interrupt 1 {
        TH0 = 0x4B;             // load 19455D to timer0, 50ms.
        TL0 = 0xFF;
        G_timer0_count++;
        G_timer0_flag = 1;
}

void timer1() interrupt 3{
        TH1 = 0xF8;             // load 63691 to timer1 ,  2ms
        TL1 = 0xCB;
        G_timer1_count++;
        G_timer1_flag = 1;
}
```