Timer

实验要求

1. 编写程序，实现一个实时时钟；
   2. 从默认时间点开始计时；
   3. 必须使用定时器+中断响应程序来处理；
   4. 计时结果（年、月、日、时、分、秒）存放在RAM的数组中；
   5. 为了调试方便，可以缩短计时时间，实现短时间内计时数天、数月、数年，要在实验报告中说明当前的计时比例；
2. 实验报告中要附上调试结果截图

代码

```
#include <reg51.h>
#define start_year 4;  // 2004
#define start_month 2; // February
#define start_day 27;  // 27th
#define start_hour 23; // 23:58:00
#define start_min 58;
#define start_sec 0;

// no speed up
// #define counter_H 0x3c;
// #define counter_L 0xb0;

// speed up 1000x
#define counter_H 0xff;
#define counter_L 0xce;

struct TIME
{
    unsigned char year;
    unsigned char month;
    unsigned char day;
    unsigned char hour;
    unsigned char min;
    unsigned char sec;
} data time;

// not leap year
unsigned char MONTH1[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
// leap year
unsigned char MONTH2[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
unsigned int MonthDay;
unsigned char cnt_50ms = 0;

// judge whether the year is leap year
```

```c
unsigned char is_leap_year(unsigned char year)
{
    return ((year + 2000) % 4 == 0 && (year + 2000) % 100 != 0) || ((year + 2000) % 400 == 0);
}

// timer0 initialization
void timer0_init(void)
{
    TMOD = 0x01;     // set timer0 as mode1 (16-bit)
    TH0 = counter_H; // initial counter values
    TL0 = counter_L;
    TR0 = 1; // timer0 start run
    ET0 = 1; // enable timer0 interrupt
    EA = 1;  // open global interrupt switch
}

// timer0 interrupt service routine
void timer0_isr(void) interrupt 1
{
    TH0 = counter_H; // reload counter values
    TL0 = counter_L;
    cnt_50ms++;

    if (cnt_50ms == 20)
    {
        cnt_50ms = 0;
        time.sec++;
        if (time.sec == 60)
        {
            time.sec = 0;
            time.min++;
            if (time.min == 60)
            {
                time.min = 0;
                time.hour++;
                if (time.hour == 24)
                {
                    time.hour = 0;
                    time.day++;
                    if (is_leap_year(time.year))
                        MonthDay = MONTH2[time.month - 1];
                    else
                        MonthDay = MONTH1[time.month - 1];

                    if (time.day == MonthDay + 1)
                    {
                        time.day = 1;
                        time.month++;
                        if (time.month == 13)
                        {
                            time.month = 1;
                            time.year++;
                        }
                    }
                }
            }
        }
```

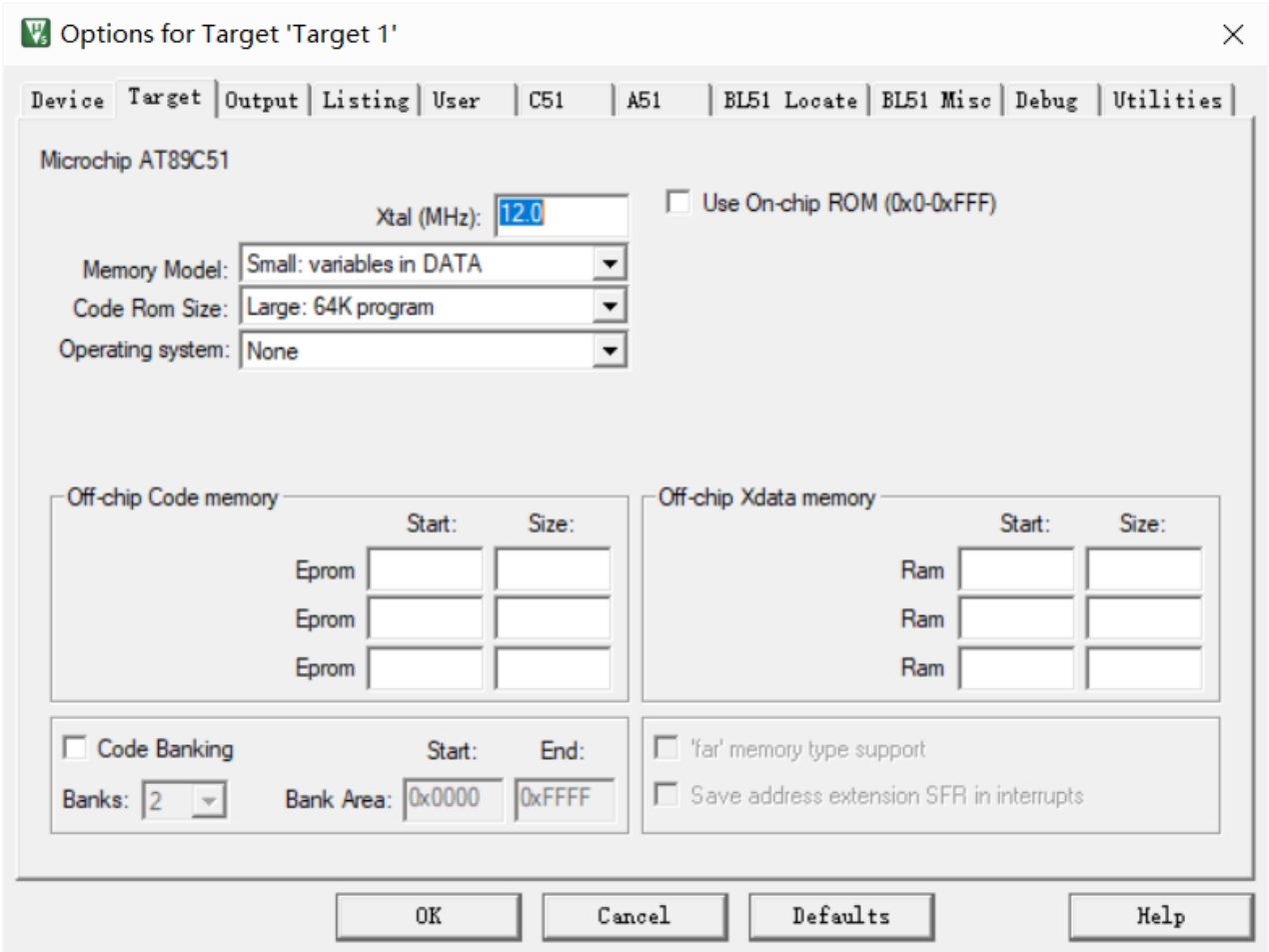
```
    }
}

void main(void)
{
    // initialize time
    time.year = start_year;
    time.month = start_month;
    time.day = start_day;
    time.hour = start_hour;
    time.min = start_min;
    time.sec = start_sec;

    timer0_init();
    while (1)
        ;
}
```

实验过程

- 关于时间参考
  首先我将晶振频率改为12MHz



  在写完代码之后，我先在无加速的条件下测试了代码的正确性，发现时间存在不匹配的现象，就是说Keil中的时间与现实世界的时间流逝不同。我所分析的时间倍数都是相对于Keil中的时间而言的（右下角时间t0）
- 关于倍速：通过改变TH0和TL0的初始值实现倍速操作。
  在代码中我注释掉了原来无加速（即时间流逝比为1：1）的情况：65536-50000=3CB0H

```
// no speed up
// #define counter_H 0x3c;
// #define counter_L 0xb0;
```

我希望加速**1000倍**，经过计算：65536-50=FFCEH

```
// speeding up 1000x
#define counter_H 0xff;
#define counter_L 0xce;
```

若选择其他的加速倍数，只需要计算之后调整`counter_H`与`counter_L`的值即可

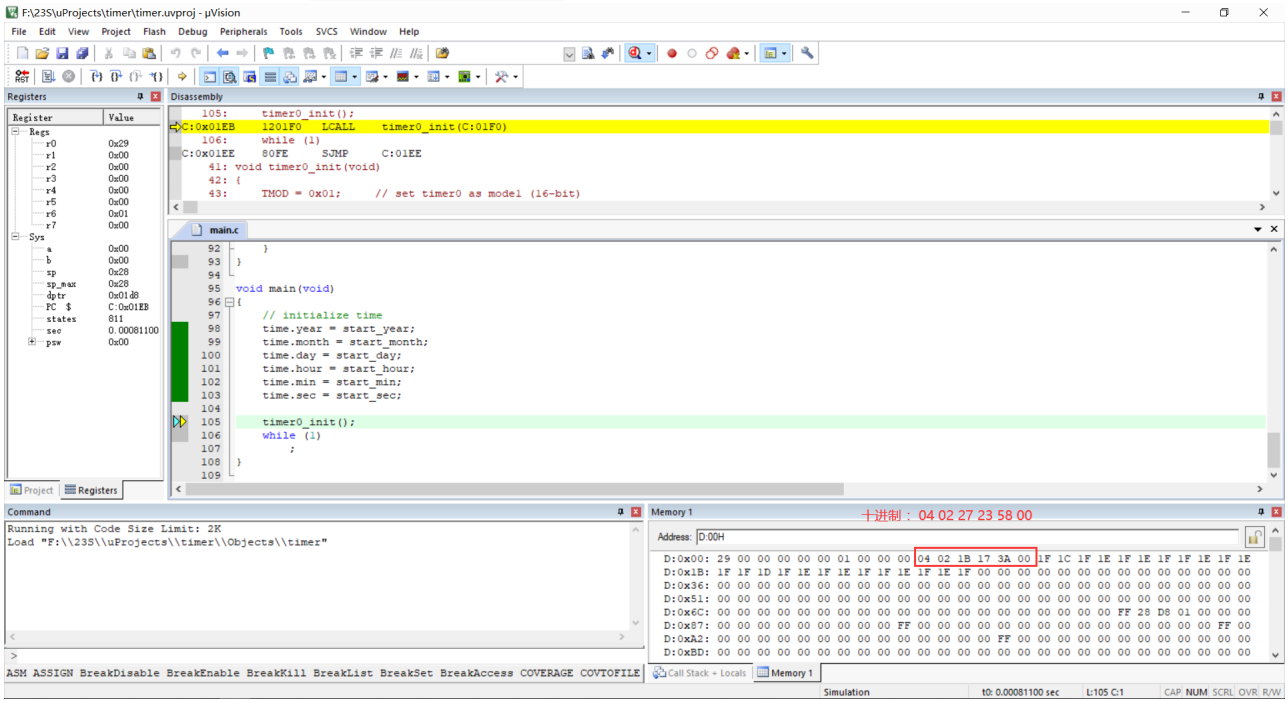
- 关于起始时间
  我选择的实验起始时间为`2004.2.27, 23:58:00`
  （对于年份，默认是从2000年开始，所以2004年对应04）

```
#define start_year 4;  // 2004
#define start_month 2; // February
#define start_day 27;  // 27th
#define start_hour 23; // 23:58:00
#define start_min 58;
#define start_sec 0;
```

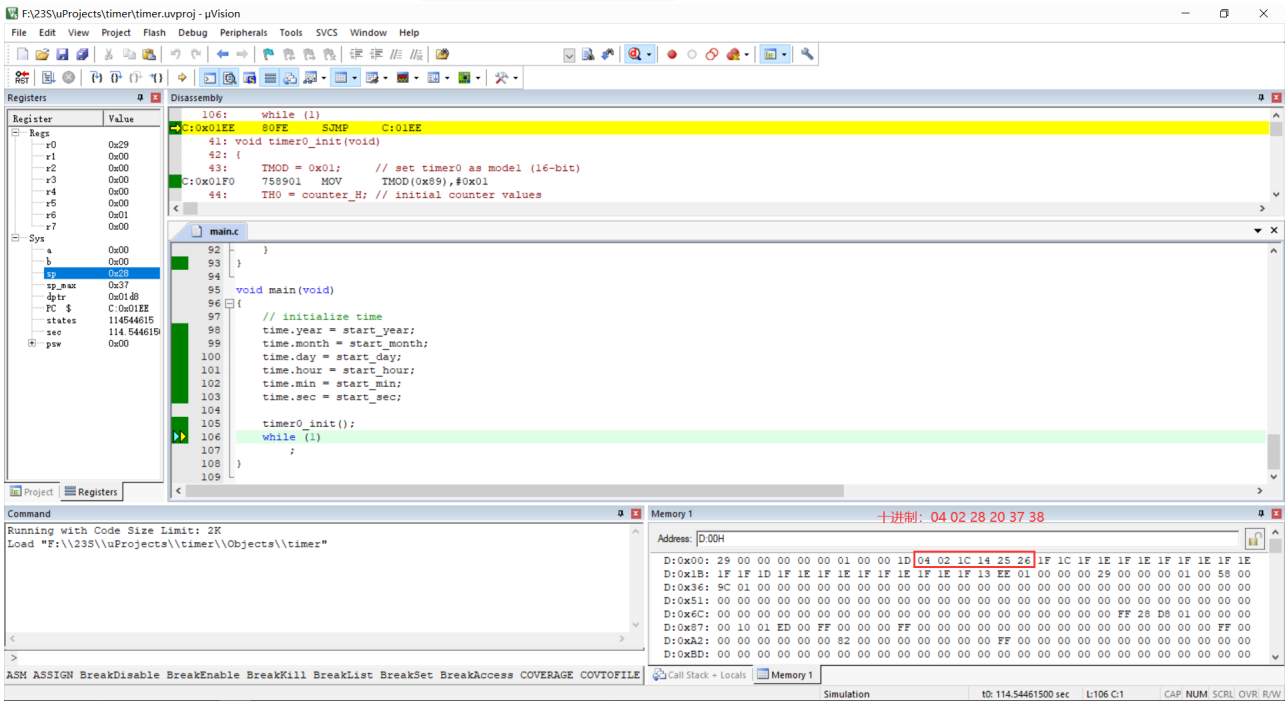
实验结果

在RAM中如图所示的6个单元分别储存了年、月、日、时、分、秒

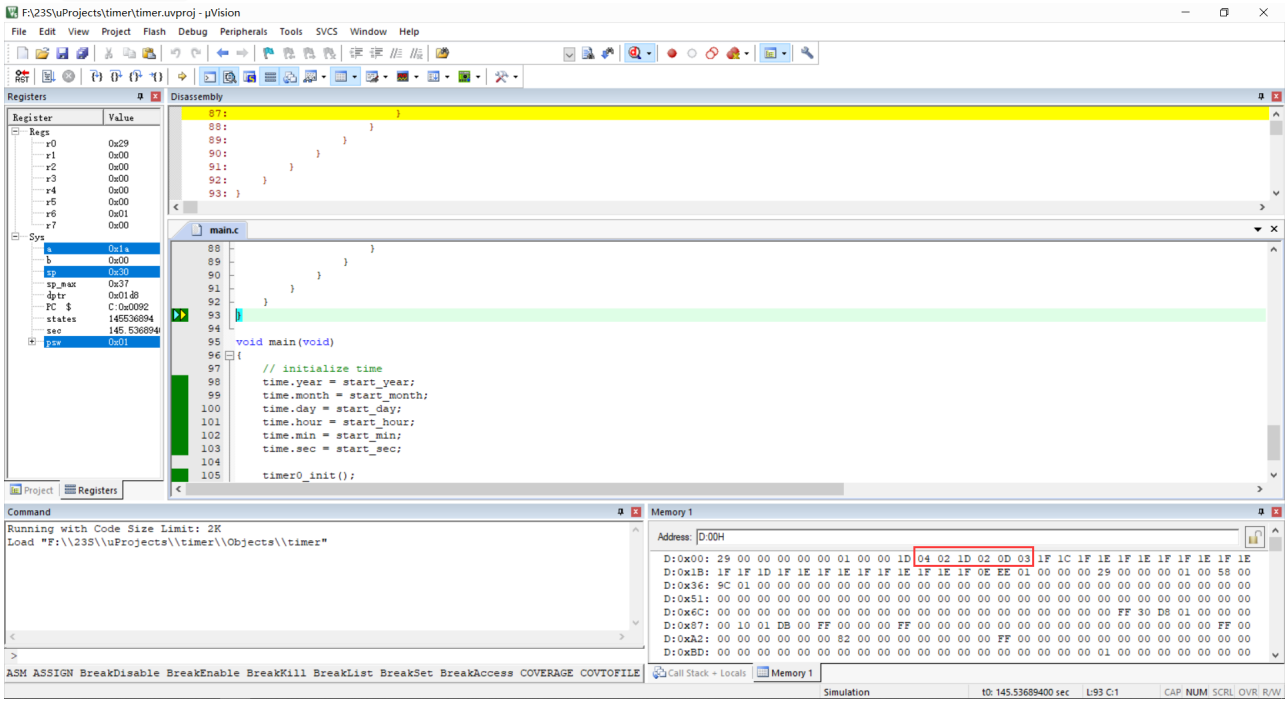
- 完成初始化后的起始时间`2004.2.27, 23:58:00`



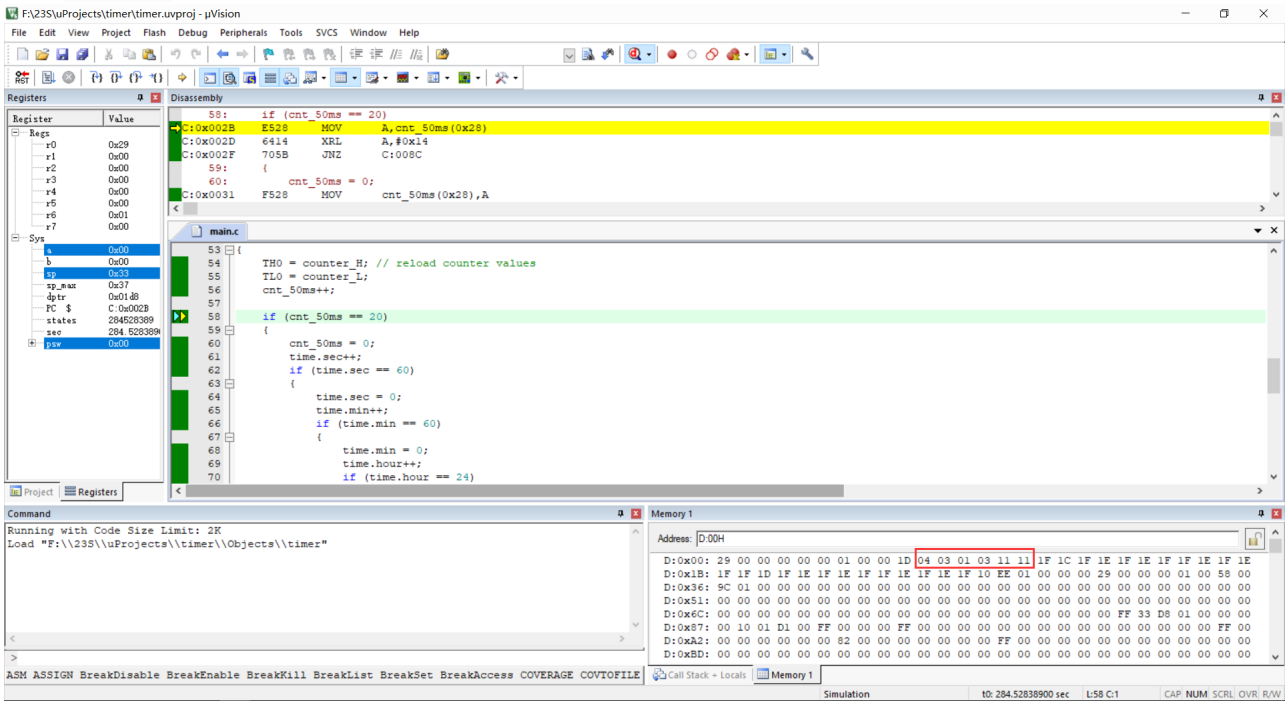
- 运行一段时间之后，到达了第二天`2004.2.28, 20:37:38`



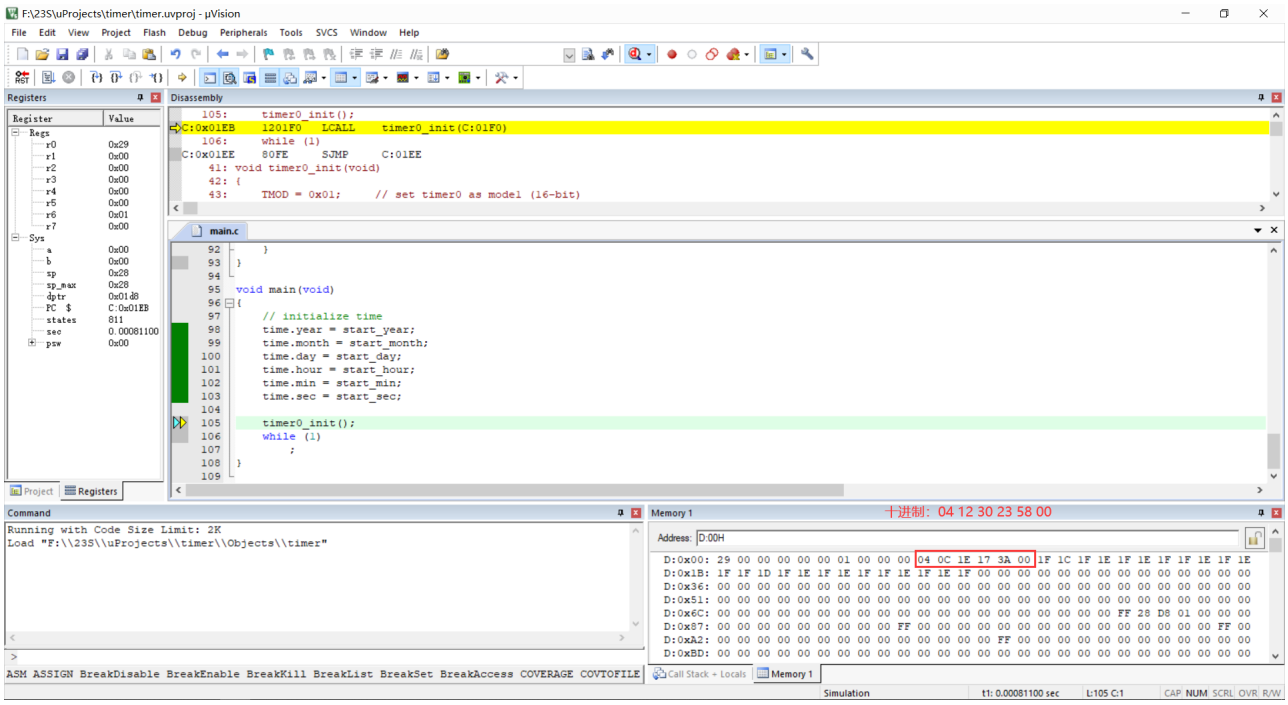
- 到达`2004.2.29` （2004是闰年）



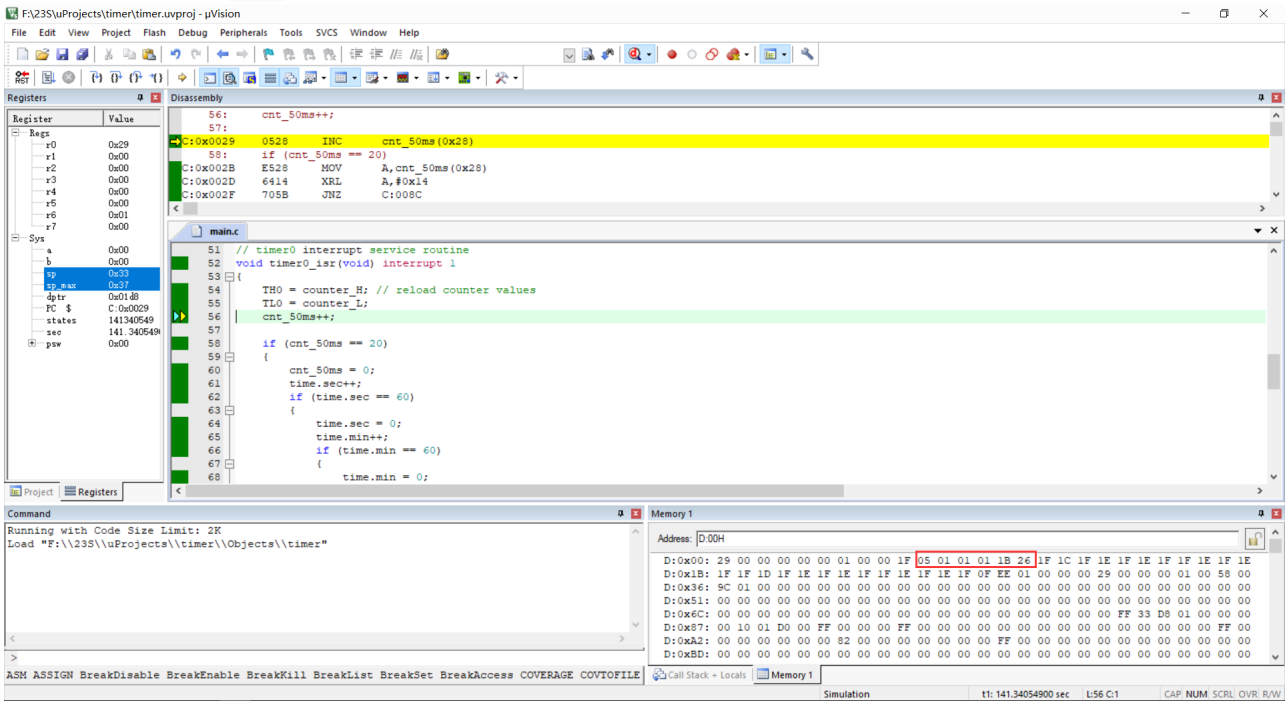
- 到达 `2004.3.1`



- 为了展示关于年份的变化，我没有改动其他，而仅仅将**初始时间**进行了改动，改为 `2004.12.30, 23:58:00` 初始时间：



一段时间之后到了 2005 年



实验分析

经过我的观察与计算，在没有加速的时候，按照右下角的时间作为参考，进行时间估算的结果时间是比较接近计时器的结果的。但是在加速倍数较大（1000）的时候，以右下角时间作为参考结合倍数所得到的时间估算结果和实际计时器的结果是存在差距的（估算结果经过的时间长于计时器经过的时间）。并且我尝试了加大倍数（到1000000倍时差距很大），这个差距显然是增大的。分析原因是加速之后其余指令执行的时间所带来的时间消耗被放大了，导致计时器结果与基于Keil中参考时间计算后的结果出现偏差，而并非代码逻辑错误带来的误差。