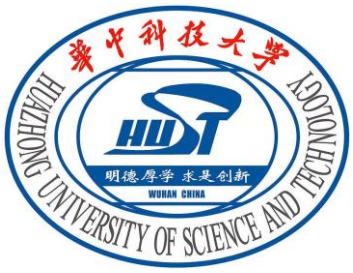

华中科技大学
实验报告
电子器件与电路（二）
嵌入式部分 1 2 3
闪灯实验

学院	工程科学学院	
班级	工程科学学院（生医）1701 班	
小组成员	汪能志	U201713082
	周政宏	U201714460
指导老师	钟国辉	

2020 年 9 月 18 日

目录

1. 实验要求

1.1. 闪灯实验

完成嵌入式系统的实验，验证实验系统的可用性，建立软件到硬件的反馈途径，使用汇编语言控制 LED 亮灭，使用指令控制闪烁的时间控制。

1.2. 采用查询定时器的闪灯实验

使用可重定位汇编代码完成 1s 为周期的闪灯实验，画出程序流程图，验证程序的正确性，描述周期的计算过程。

1.2.1. 采用定时器中断的闪灯实验

使用定时器中断方式完成闪灯试验，采用 C 语言实现，实现对周期的精确控制。

2. 实验原理与流程

2.1. 定时任务的基本方法

前三次实验的重点内容是在程序中实现延时或定时的操作。对于单片机来说，实现计时功能的方法大致有以下两种：

1. 让处理器执行特点数量的指令，通过指令占用处理器周期来实现延时的功能；
2. 通过定时器和时钟信号进行计时。

2.1.1. 软件延时

由于 PIC 单片机的处理器设计较为简单，指令的执行周期是固定的。因此通过在汇编中设置循环，在给定初值的情况下，循环的执行数量和每次循环的指令周期都是可以完全可控的。这样就可以通过循环嵌套的方法，占用给定的处理器指令周期以此达到延时的目的。

我们决定使用三个嵌套循环来实现这个功能，每一个语句时间约为 1us，如果需要 0.5s 的 delay，则需要 500000 次运行语句，我们使用 DECFSZ 指令来实现功能，DECFSZ 指令会减 1，为 0 则跳过，这提供了一个判断语句，我们只需初始化一个比较大的值，然后不断的减至 0，再进行下一个，如此三个嵌套循环，即可以实现数十万乃至百万的语句运行。

通过三个嵌套循环，我们实现了闪灯的效果。在此基础上，我们希望可以自由控制闪灯的周期，目前已知一个语句的执行时间为 1us，于是我们只需要知道三个循环数 n0、n1、n2 与总执行语句数的关系即可以完成定量预测闪灯周期。在经过许多实验与代码分析后，我们实践出了一个公式：

$$T=(3\times n_0\times n_1\times n_2+4\times n_0\times n_1+4\times n_0+5)\times 1us$$

为了验证这个公式，我们首先设置 n0=117、n1=31、n2=123，再加上循环内的语句会消耗 4 个指令周期，的出来 500000us，符合实验的闪灯周期 500ms；然后我们设置 n0=1、n1=1、n2=1，此时应该是人类视觉上的常亮，因为频率太大，实验也证明如此。

```
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
Target halted. Stopwatch cycle count = 500000 (500 ms)
```

图 1 循环嵌套闪灯调试结果

虽然可以通过循环嵌套实现延时，但是对于更先进的处理器，由于流水线、分支预测、超标量执行的原因，每条指令占用的处理器周期是不确定的，如果继续通过循环嵌套的方法，就会难以准确占用指定数量的指令周期，无法获得准确的延时（甚至可能直接被编译器优化掉）。因此需要使用到硬件定时器来获得准确的时间（时钟周期）。更为严重的问题是，在通过执行空循环进行延时时，处理器无法同时进行任何操作。因此考虑采用查询定时器控制闪灯实验。

2.1.2. 定时器延时

本次使用的 PIC 单片机的 TIMER0 定时器，通过对其进行设置，计数器会在输入指定个时钟信号后，产生溢出信号。处理器通过该溢出信号，就可以进行精确的定时操作。

如果将定时器的溢出视为一个特殊的 I/O 信号，那么检测这个信号的方法有轮询和中断两种。使用轮询的方法时，需要反复地去检测溢出标志位来判断定时器是否溢出。在 PIC 单片机中，一次标志位检测需要消耗至少三个指令周期（在部分情况下还需要进行 BANKSEL 操作，以免读取了错误的内存地址，此时需要占用四个指令周期）。而且轮询和其他的操作不能同时在处理器上进行。因此，在采用轮询操作时，处理器既要消耗大量的指令周期去检测定时器溢出，在定时器溢出发生时，又不能保证第一时间发现。是一种时效性差，处理器效率低的操作。

在使用中断时，中断事件可以抢占正常的处理器工作流程，并通过中断向量进入中断服务程序。在使用中断时，处理器不需要反复检测信号，这样就可以处理其他事务。如下图所示：

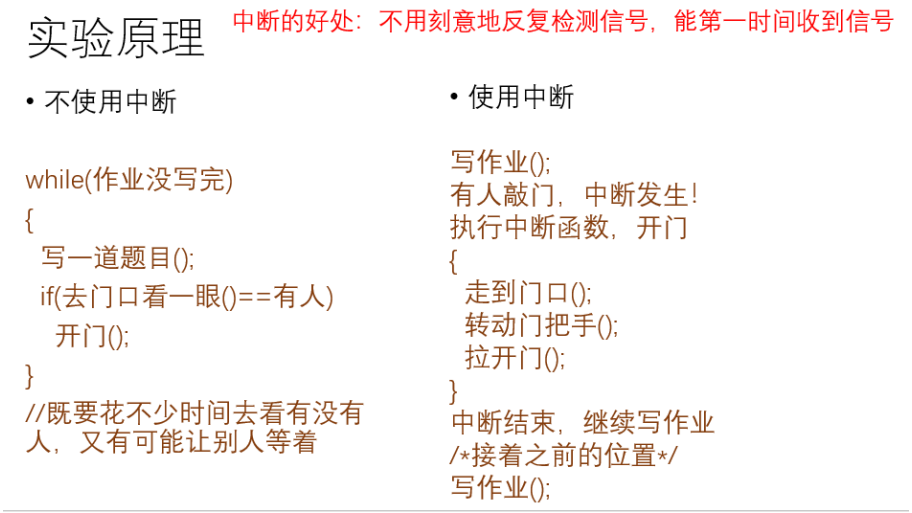

图 2 轮询与中断对比

2.2. 定时器查询

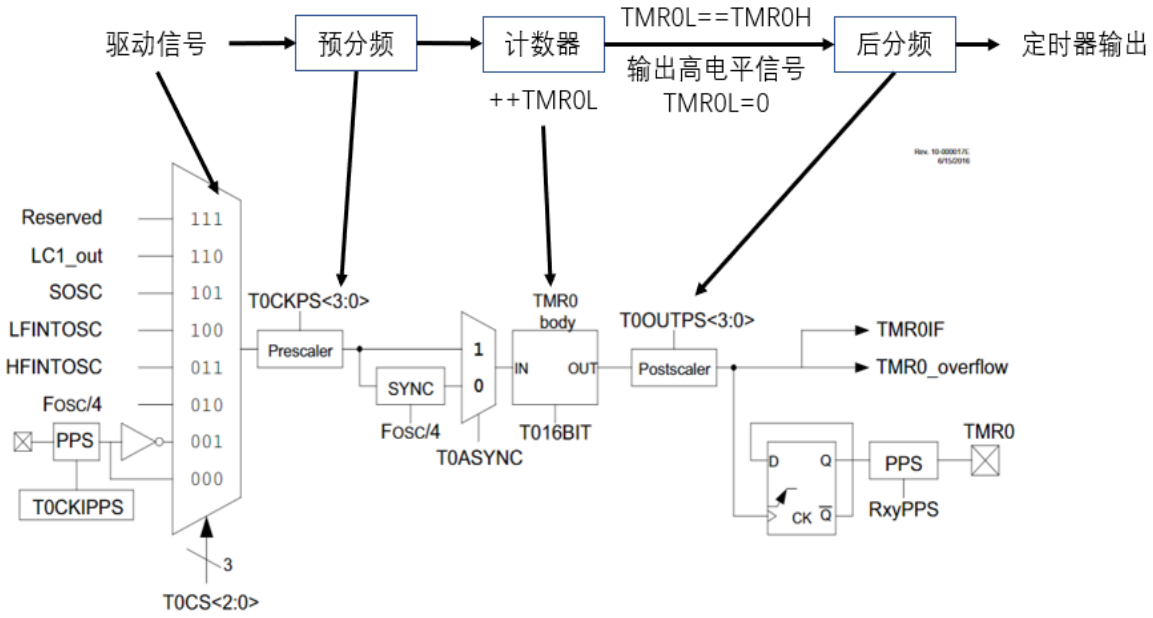

图 3 8bits 定时器原理与流程图

经过我们对代码的分析与多次实践，得出定时器的周期为：

$$T=prescaler \times postscaler \times (TMR0H+1)+1$$

当预分频为 128，后分频为 16，TMROH 的值取 243 时(考虑到 0，则计算应加 1)，则其理论结果为：

$$128 \times (243+1) \times 16 = 499{,}712$$

在 8 bits 的 TIMER0 定时器上取翻转端口电位语句中断进行调试，结果为：

```
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499709 (499.709 ms)
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499713 (499.713 ms)
Target halted. Stopwatch cycle count = 499709 (499.709 ms)
```

图 4 8 bits 查询定时器闪灯调试结果

由于我们适当轮询程序占用了四个指令周期，因此定时器的时间也以 4 次一循环。调试结果与我们的理论分析完全吻合，但是还并不是 500ms。

但是，在使用 c 语言复现这一代码时，跑表时间稳定在 499712，没有出现抖动。可能是在编译时，编译器会通过空指令控制程序执行周期，保证每次恰好在轮询时溢出。

2.3. 定时器中断

2.3.1. 8 bits 定时器

对于 8 bits 的 TIMER0 定时器，其每个延时周期的计算公式为：

$$PRE \times TMROH \times POST$$

当预分频为 128，后分频为 16，TMROH 的值取 243 时(考虑到 0，则计算应加 1)，则其理论结果为：

$$128 \times (243 + 1) \times 16 = 499{,}712$$

在 8 bits 的 TIMER0 定时器上取翻转端口电位语句中断进行调试，结果为：

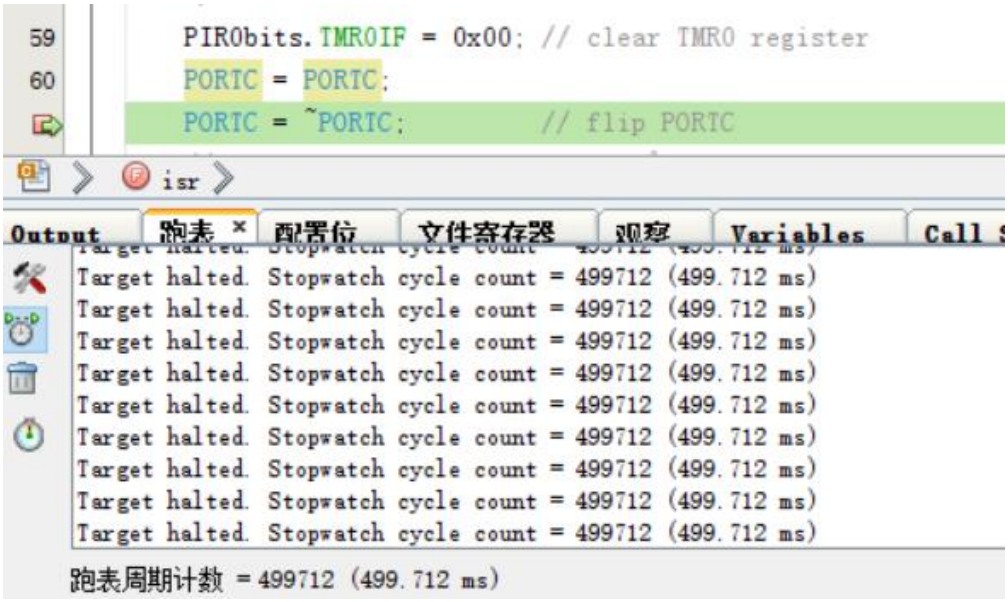

图 5 8 bits 定时器中断闪灯调试结果

可以看出，调试结果与我们的理论分析完全吻合，但是还并不是 500ms，接下来我们采用了 16 bits 的 TIMER0 定时器。

2.3.2. 16 bits 定时器

对于 16 bits 的 TIMER0 定时器，其每个延时周期的计算公式为：

$$T = pre \times (65536 \times (post - 1) + 65536 - tmr)$$

当预分频为 2，后分频为 4，TMR 的值取 12149 时，则其理论结果为：

$$2 \times (65536 \times 3 + 65536 - 12149) = 499990$$

每次定时器溢出，进入中断服务程序后，需要重置定时器，由于重置时，会清空分频器，因此实际的溢出时间会略长于理论值，调试表明，进入中断的周期正好是 500ms。

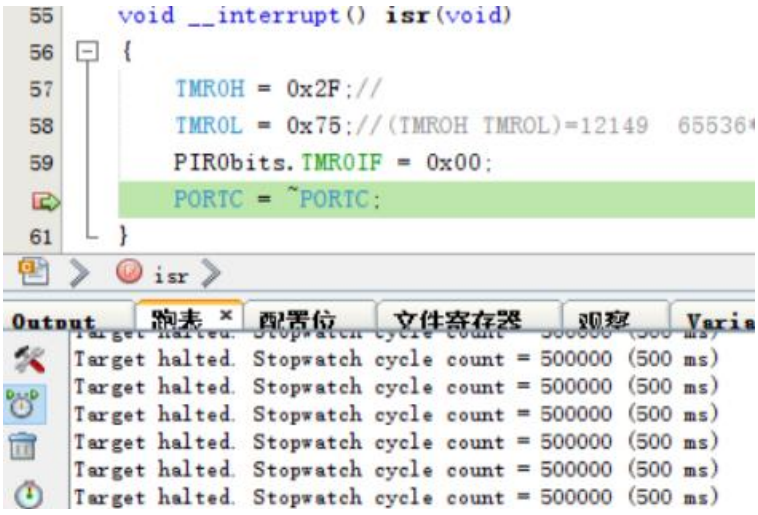

图 6 16 bits 定时器中断调试结果

可以看出，无论是 8 bits 的 TIMER0 定时器还是 16 bits 的 TIMER0 定时器，我们均实现了利用定时器中断进行延时闪灯的效果，并且理论分析与调试结果统一，给出了计算公式，显然 16 bits 的加法与乘法比 8 bits 的纯乘法更容易达到 500ms 的精确延时，在本实验中，16 bits 的结果为准确的 500ms 延时。

3. 遇到问题

3.1. 程序无法正常运行

首先是无法编译，然后和老师与同学们交流后换了一个编译器版本，调整了编译器版本后，在运行常亮的代码时，我们又遇到了一个问题，即我的电脑编译出来的 hex 文件始终有问题，后来在老师的指导下才发现是在打开项目的时候设置出现了问题，于是重新打开项目，问题解决。

3.2. 部分端口无法正常闪灯

经过我们的进一步调试发现，是我们的取反语句出现了问题，即我们并不是每次都对每个端口进行取反，后来发现了问题后，使所有端口都与 1 进行异或运算实现取反，最后解决了问题。

3.3. TMR0 定时器中陷入空循环无法跳出

在看过数据手册之后，发现是没有考虑分区，并且源设置也有问题导致，最后更正后解决了问题。

3.4. 延时周期呈现周期性的波动或者突然变大

我们的实验结果为周期以 4 为周期进行波动，但是其平均值与我们预期结果符合，这是因为不是每次循环都是正好在定时器询问是否溢出的时候溢出，有可能在其之前，也有可能在其之后，所以需要进行补偿，使得每次都正好在询问时溢出。

此外，仿真环境下周期突然变大的问题我们推测是因为在仿真环境下触发了看门狗，其变量清零，从头运行，定时发生的特性都非常类似于没有喂狗导致的看门狗作动。之后和老师讨论确认了在 boot loader 环境下，看门狗确实处于激活状态，会周期性触发，应该在 default 环境下进行调试。

3.5. 延时周期无法精确到 500ms

一开始我们使用 8 bits 的计数器，最好也只能达到 499712，离 500000 还有差距，于是后来我们改成了 16 bits 的计数器，并且花了很多时间调试，补偿，计算定时器参数，最后可以达到精确的 500ms。

4. 实验总结

4.1. 周政宏

此次实验是数电课程的前三个实验，总体来说难度不大，但是对于刚接触到嵌入式系统的我们还有挑战，一开始我们并不熟悉工作原理、实验流程、如何编写代码等等，都是在做实验的过程中我们不断查阅资料去学习，去调试。不可避免的我们遇到了很多问题，在解决问题的过程中，我们也收获了很多经验，比如如何在写代码前一定要思路清晰，否则直接上手写代码会给调试带来很大的困难，而且往往写出来错误的代码；在遇到一些看似“玄学”的问题是不要轻易放弃，至少在我们接触到的实验中，结果都是可控的，所谓“玄学”，一定是还没有彻底搞清楚原理。

总的来说，通过这次实验，我们收获了很多，入门了嵌入式系统与汇编语言，还复习了 C 语言，还明白了面对问题时需要冷静分析、保持思路清晰等宝贵的经验。

4.2. 汪能志

这是数电最早的三次实验，在完成实验后再回看这三次实验的主要内容，其实并不复杂。虽然我之前也在其他场合接触过别的单片机系统（Arduino），但是刚刚开始实验时，由于对汇编语言以及 PIC 单片机的结构不太熟悉，一开始做实验时遇到了很多问题，单片机经常点不亮。通过实验过程，我越发明白，在初入一个陌生的领域时，一份编写良好的 datasheet 的重要作用。包括端口设置，定时器设置等一开始我觉得较为复杂的领域，在 datasheet 中都有不错的内容可以参考。另一点，虽然完成相似任务的代码在互联网上比比皆是，但是简单的 CTRL C +V 操作，只能在确定环境通用的情况下，快速验证系统可以完成需求。如果想真正对内容有所理解，还是需要自己一步一步地推导操作流程。

通过这三次实验，我对于嵌入式系统有了一个真正的入门，对于汇编语言和 C 语言都有了一定的认识与复习。同时也对于如何快速入门一个新的领域有了自己的了解。

本系列实验中所涉及的所有 PIC 单片机代码和相应的辅助计算代码均已上传到 GitHub：
https://github.com/NengzhiWang/Embedded_PIC_SES_2020

附录：实验一代码

```
org BLOFFSET
; pre define
; address in Common RAM, no need for `BANKSEL`
udata_shr
N0      res     1h
N1      res     1h
N2      res     1h

RST code    BLOFFSET
PAGESEL    MAIN
GOTO   MAIN

code
; ============================================
MAIN
; SETUP
; init ports, select port C, set as Digital Output
; select port C
BANKSEL     PORTC
CLRF       PORTC

; select digital signal
BANKSEL     ANSELC
CLRF       ANSELC
MOVLW      B'00000000'
MOVWF      ANSELC

; select output
BANKSEL     TRISC
MOVLW      B'00000000'
MOVWF      TRISC

; init output
MOVLW      B'00000000'
MOVWF      PORTC

; ============================================
; LOOP
; main loop
LOOP
    ; W is used as XOR
    MOVLW      B'11111111'
```

```
    ; XOR port c & reg f
    XORWF      PORTC,  1
    ; delay
    CALL       DELAY
GOTO    LOOP

; sub program for delay
DELAY
    ; Instruction Num = 3 * N0 * N1 * N2 + 4 * N0 * N1 + 4 * N0 + 5
    MOVLW      0x4F
    MOVWF      N0
    DELAY_LOOP_0

        MOVLW      0x19
        MOVWF      N1
        DELAY_LOOP_1

            MOVLW   0x53
            MOVWF   N2
            DELAY_LOOP_2
                DECFSZ  N2
            GOTO    DELAY_LOOP_2

            DECFSZ  N1
        GOTO    DELAY_LOOP_1

        DECFSZ  N0
    GOTO    DELAY_LOOP_0
RETURN


END
```

附录：实验二代码

```
org BLOFFSET

; ==========================================
pagesel    MAIN
GOTO       MAIN
; ==========================================

code

MAIN
; ==========================================
; init PORTC
BANKSEL    PORTC
CLRF       PORTC
BANKSEL    LATC
CLRF       LATC
BANKSEL    ANSELC
CLRF       ANSELC

BANKSEL    TRISC
MOVLW      B'00000000'
MOVWF      TRISC
; ==========================================
; init TMR0

BANKSEL    TMR0H
MOVLW      0xF3
MOVWF      TMR0H

BANKSEL    TMR0L
MOVLW      0x00
MOVWF      TMR0L

BANKSEL    T0CON0
MOVLW      B'11001111';
; bit7     enable timer0
; bit4     8-bit timer
; bit 3-0   postscaler  1:16
MOVWF      T0CON0

BANKSEL    T0CON1
MOVLW      B'01000111';
```

```
; bit 7-5   clk source  F_OSC / 4
; bit 4     sync
; bit 3-0   prescaler   1:128
MOVWF      T0CON1

; clear the output signal
BANKSEL    PIR0
BCF        PIR0,   5
; ==========================================

LOOP
    BANKSEL    PIR0
    BTFSS      PIR0,   5
    ; no output from TMR0
    GOTO       LOOP
    ; have output from TMR0
    ; BANKSEL PIR0
    BCF        PIR0,   5         ; clear TMR0IF
       ; set TMR0L as 0x00
    ; make sure TMR0IF overflow at same time during test



    BANKSEL    PORTC
    MOVLW      B'11111111'
    XORWF      PORTC,    f       ; flip PORTC



    GOTO       LOOP
    END

END
```

附录：实验三代码 8bits 定时器

```
/*
 * ISR definition
 */

void __interrupt() isr(void)
{
    /*
       interrupt sub program
     */
    PIR0bits.TMR0IF = 0x00;
    // clear TMR0 register
    PORTC = ~PORTC;
    // flip PORTC
}

void setup(void)
{
    /************************************************************/
    // init PORTC
    ANSELC = 0x00;
    LATC = 0x00;
    TRISC = 0x00;
    PORTC = 0x00;
    /************************************************************/
    // init interrupt
    INTCONbits.GIE = 1;
    // global interrupt    enable
    INTCONbits.PEIE = 0;
    // peripheral interrupt disable
    INTCONbits.INTEDG = 1;
    // interrupt          rising edge
    PIE0bits.TMR0IE = 1;
    // Timer0 interrupt    enable

    /************************************************************/
    // init TMR0
    T0CON0 = 0xCF;
    /*
    B'1100 1111'
    bit_7      T0EN    1      enable     TMR0
    bit_4      T016BIT 0      select     8bit
    bit_3-0    T0OUTPS 1111   postscaler  1:16
```

```
     */
    T0CON1 = 0x47;
    /*
    B'0100 0111'
    bit_7-5    T0CS    010    clk_source  F_OSC / 4
    bit_4      T0ASYNC 0      sync
    bit_3-0    T0CKPS  0111   prescaler   1:128
     */

    TMR0H = 0xF3;
    TMR0L = 0x00;
    PIR0bits.TMR0IF = 0x00;
}

void loop(void)
{
}

void main(void)
{
    setup();
    while (1)
    {
       loop();
    }
    return;
}
```

附录：实验三代码 16bits 定时器

```
#define TMR_rst TMR0H = TMR0H_rst, TMR0L = TMR0L_rst, PIR0bits.TMR0IF =
0x00

void __interrupt() isr(void)
{

    // reset TMR0
    // TMR0H = TMR0H_rst;
    // TMR0L = TMR0L_rst;
    // PIR0bits.TMR0IF = 0x00;
    TMR_rst;
    // interrupt task
    PORTC = ~PORTC;
    // flip PORTC
}

void setup(void)
{
    /************************************************************/
    // init PORTC
    ANSELC = 0x00;
    LATC = 0x00;
    TRISC = 0x00;
    PORTC = 0x00;
    /************************************************************/
    // init interrupt
    INTCONbits.GIE = 1;
    // global interrupt    enable
    INTCONbits.PEIE = 0;
    // peripheral interrupt disable
    INTCONbits.INTEDG = 1;
    // interrupt          rising edge
    PIE0bits.TMR0IE = 1;
    // Timer0 interrupt    enable

    /************************************************************/
    // init TMR0
    T0CON0 = 0xD3;
    /*
    B'1101 0011'
    bit_7      T0EN    1      enable     TMR0
    bit_4      T016BIT 1      select     16bit
```

```
    bit_3-0    T0OUTPS 0011    postscaler  1:4
    */
    T0CON1 = 0x41;
    /*
    B'0100 0001'
    bit_7-5    T0CS    010    clk_source  F_OSC / 4
    bit_4      T0ASYNC 0      sync
    bit_3-0    T0CKPS  0001    prescaler   1:2
    */
    TMR0H = TMR0H_rst;
    TMR0L = TMR0L_rst;
    PIR0bits.TMR0IF = 0x00;
}
void loop(void)
{
}
void main(void)
{
    setup();
    while (1)
    {
        loop();
    }
    return;
}
```