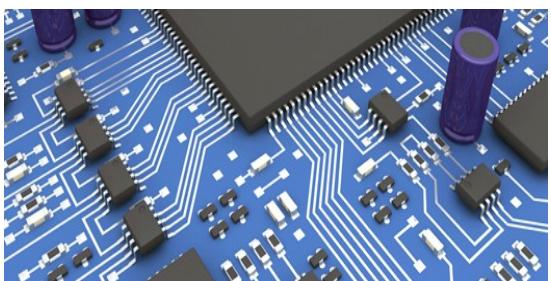


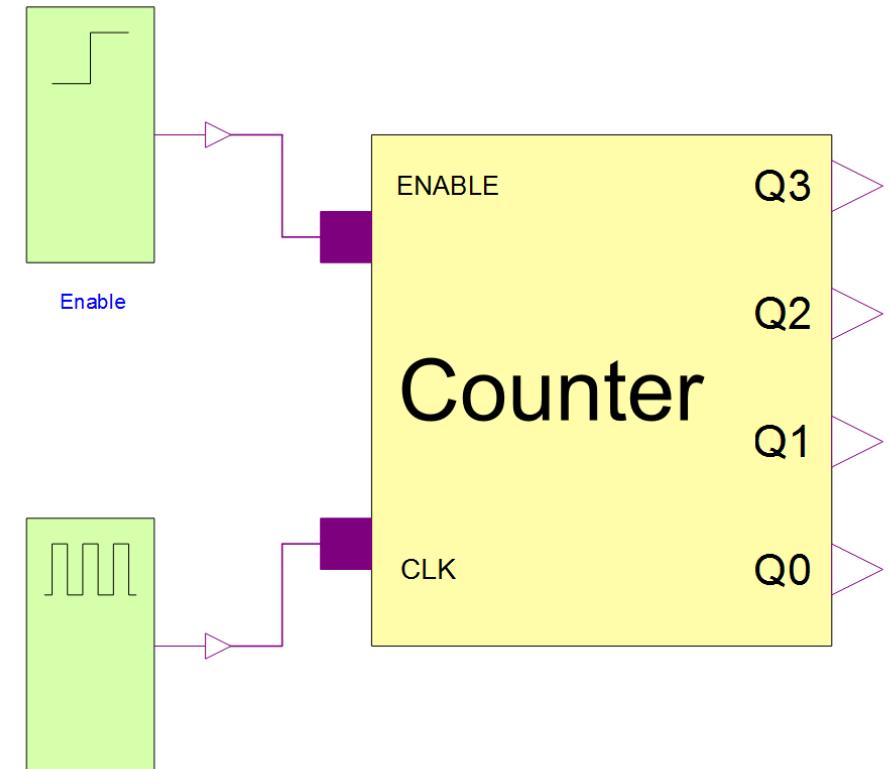
# Aplikace Embedded systémů v Mechatronice



Michal Bastl

# Timer

- Timer je periferie MCU. Jedná se o obvod k odměřování času. Jde v podstatě o binarní čítač, který načítá počet pulzů referenčního signálu.
- Zdroj signálu může být upraven děličkou
- Aktuální stav čítače je možné vyčíst z příslušných registrů
- PIC18: 16-bit 1/3/5, 8-bit 2/4/6
- čas se odvozuje od frekvence zdrojového signálu



# Nastavení oscilátoru

## Kapitola oscilátor module

### 2.3 Register Definitions: Oscillator Control

REGISTER 2-1: OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-0	R/W-1	R/W-1	R-q	R-0	R/W-0	R/W-0
IDLEN		IRCF<2:0>		OSTS <sup>(1)</sup>	HFOFS	SCS<1:0>	
bit 7							bit 0

### 2.7 Register Definitions: Oscillator Tuning

REGISTER 2-3: OSCTUNE: OSCILLATOR TUNING REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INTSRC	PLLEN <sup>(1)</sup>			TUN<5:0>			
bit 7							bit 0

#### Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

```
OSCCONbits.IRCF = 0b111;  
OSCTUNEbits.PLLEN = 1;  
while(!OSCCONbits.HFOFS){};
```

# Timer1/3/5

## 12.13 Register Definitions: Timer1/3/5 Control

## REGISTER 12-1: TXCON: TIMER1/3/5 CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/0	R/W-0/u
TMRxCS<1:0>		TxCKPS<1:0>		TxSOSCEN	TxSYNC	TxD16	TMRxON
bit 7							bit 0

**Legend:**

R = Readable bit

$u = \text{Bit is unchanged}$

'1' ≡ Bit is set

W = Writable bit

x = Bit is unknown

'0' ≡ Bit is cleared

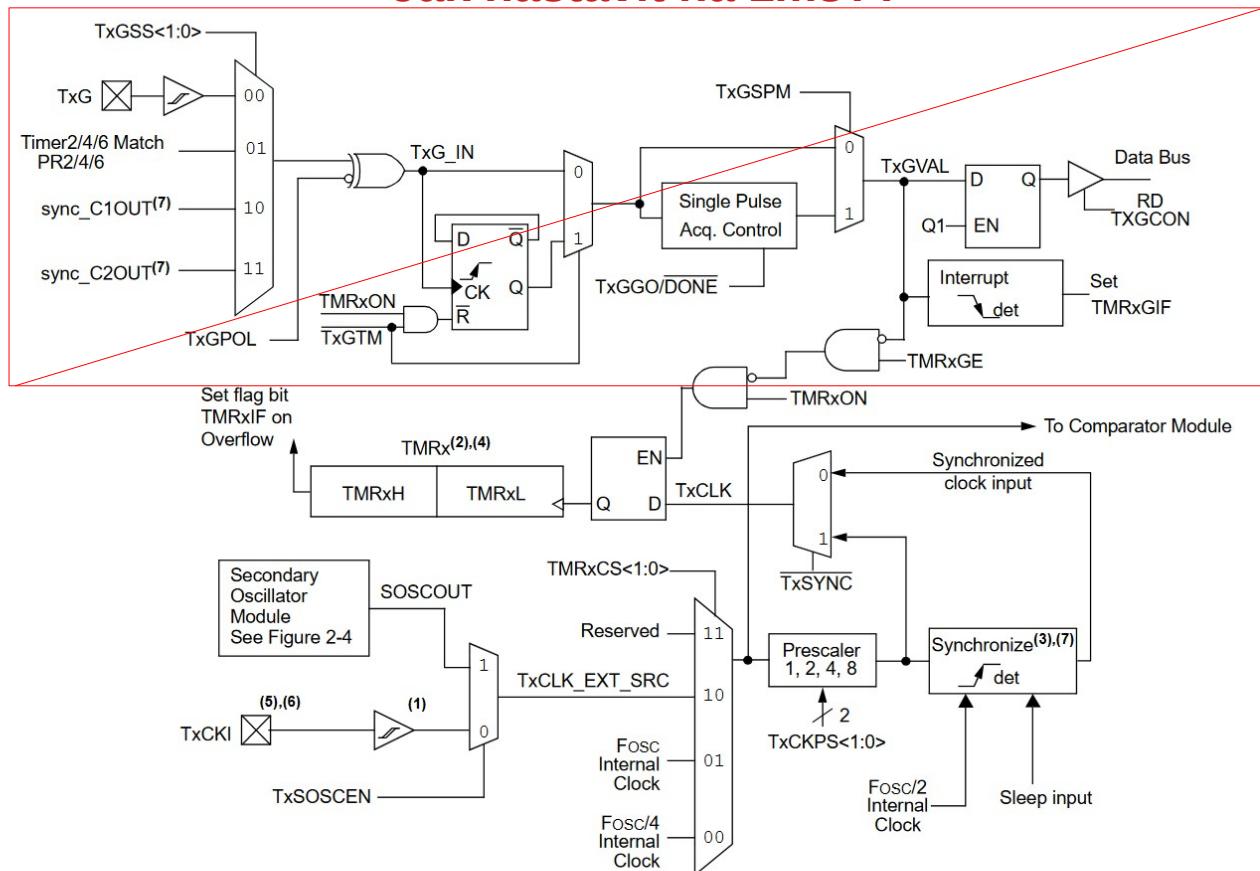
**U** = Unimplemented bit, read as '0'

-n/n = Value at POR and BOR/Value at all other Resets

- **TMRxCS** nastavuje zdroj signálu
  - **TxCCKPS** Nastavuje před-děličku
  - **TMRxON** Spouští timer

$$T_{period} = \frac{1}{f_{source}} \cdot DIV$$

## Jak nastavit na 1ms??



# Použití timeru 1/3/5

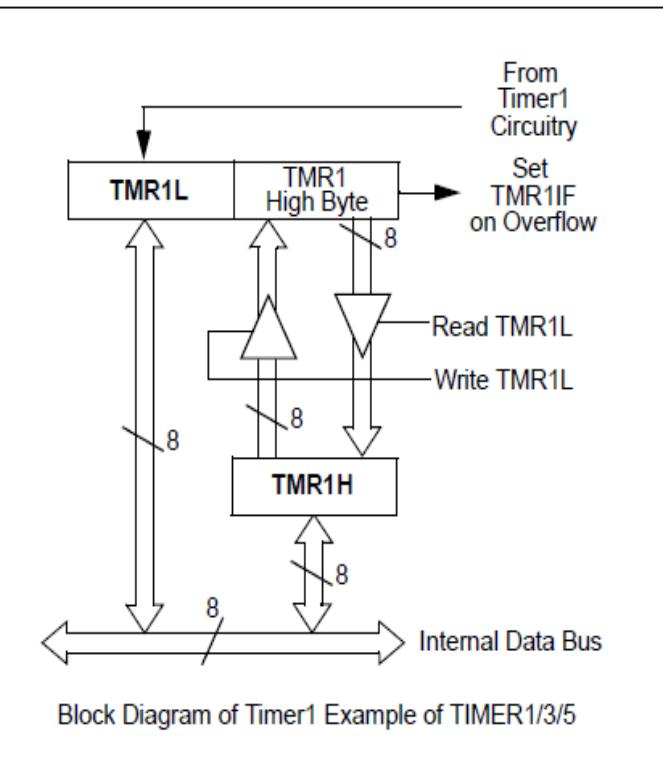
Nastavení timeru ( inicializace periferie):

```
void init(void){  
  
    // set pins as outputs  
    TRISDbits.TRISD2 = 0;  
  
    // Timer  
    T1CONbits.TMR1CS = 0b00; // TMR1 source (FOSC/4)  
    T1CONbits.T1CKPS = 0b11; // TMR1 prescaler (1:8)  
    T1CONbits.TMR1ON = 1;    // TMR1 on  
}
```

- Při čtení TMRxL dojde k přesunu hodnoty v TMRxH
- To umožňuje získat správnou hodnotu „najednou“

Použití v kódu:

```
void main(void)  
{  
    init();  
    while(1){  
        if(TMR1 >= 50000){  
            LATDbits.LATD2 ^= 1;  
            TMR1 = 0;  
        }  
    }  
}
```



# Timer2/4/6

## 13.6 Register Definitions: Timer2/4/6 Control

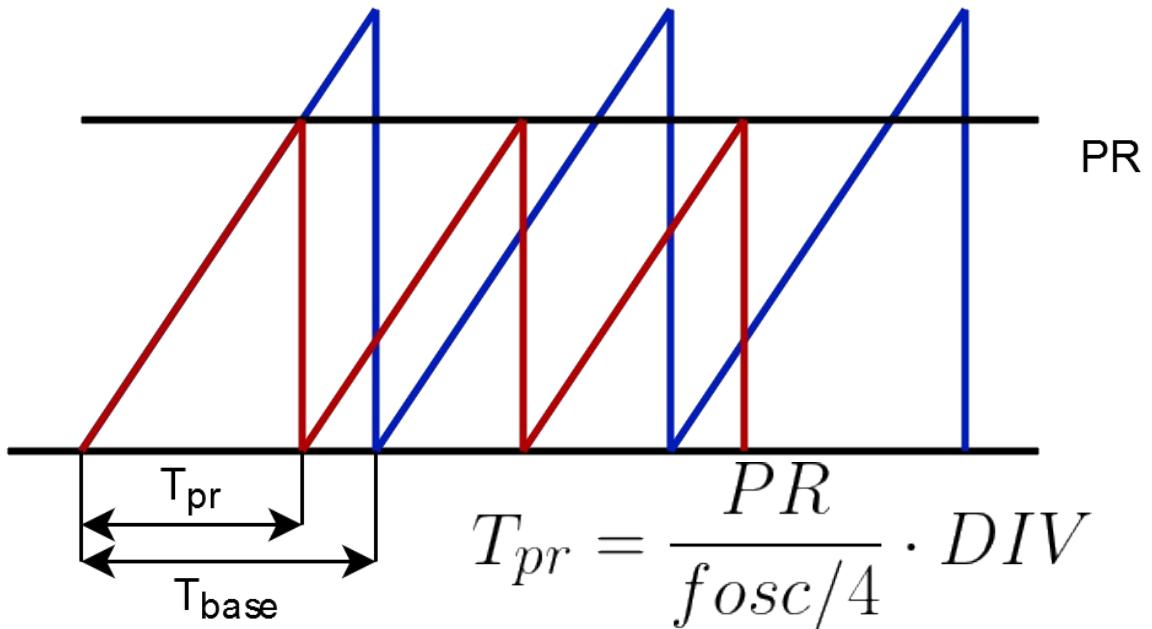
REGISTER 13-1: TCON: TIMER2/TIMER4/TIMER6 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TxOUTPS<3:0>			TMRxON		TxCKPS<1:0>	
bit 7							bit 0

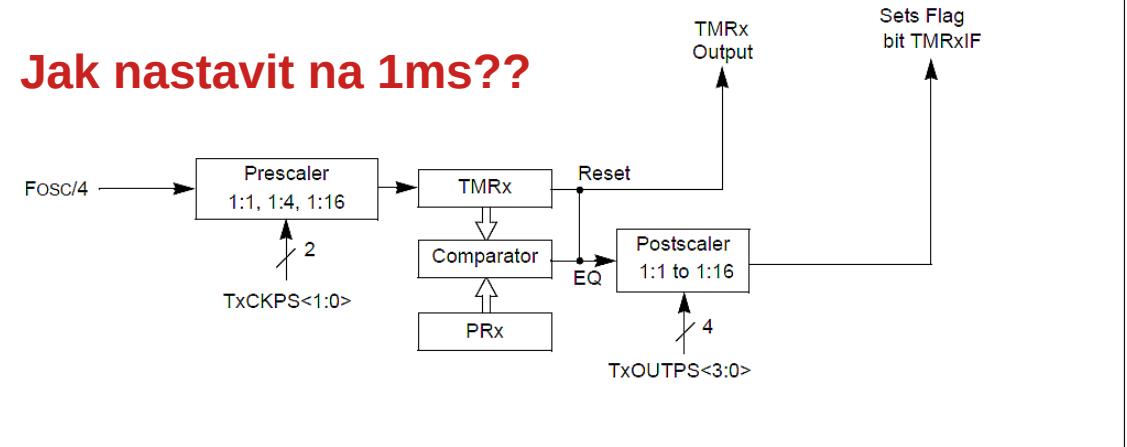
Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7	Unimplemented: Read as '0'
bit 6-3	TxOUTPS<3:0>: TimerX Output Postscaler Select bits
	0000 = 1:1 Postscaler
	0001 = 1:2 Postscaler
	0010 = 1:3 Postscaler
	0011 = 1:4 Postscaler
	0100 = 1:5 Postscaler
	0101 = 1:6 Postscaler
	0110 = 1:7 Postscaler
	0111 = 1:8 Postscaler
	1000 = 1:9 Postscaler
	1001 = 1:10 Postscaler
	1010 = 1:11 Postscaler
	1011 = 1:12 Postscaler
	1100 = 1:13 Postscaler
	1101 = 1:14 Postscaler
	1110 = 1:15 Postscaler
	1111 = 1:16 Postscaler
bit 2	TMRxON: TimerX On bit
	1 = TimerX is on
	0 = TimerX is off
bit 1-0	TxCKPS<1:0>: Timer2-type Clock Prescale Select bits
	00 = Prescaler is 1
	01 = Prescaler is 4
	1x = Prescaler is 16

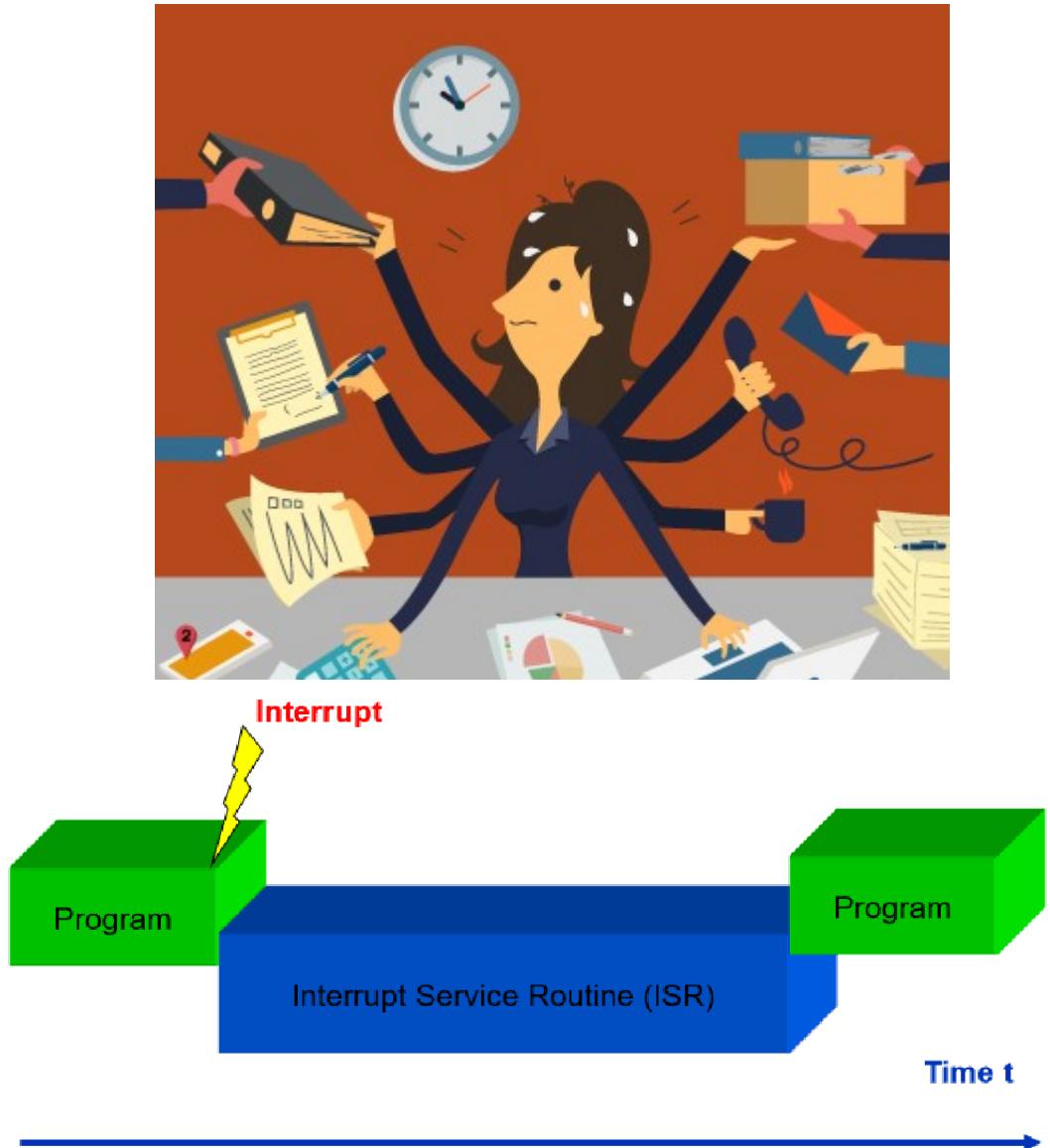


Jak nastavit na 1ms??

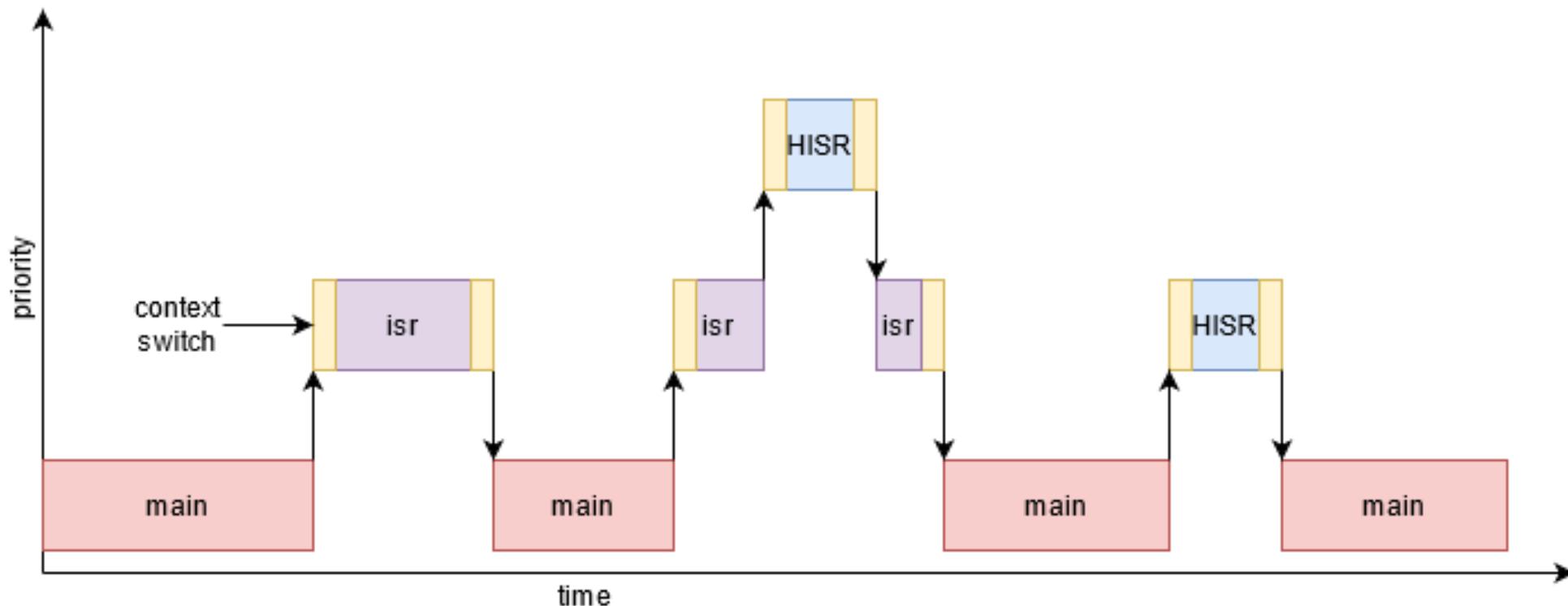


# Interrupt(přerušení)

- Jedná se o techniku, která umožňuje procesoru zpracovávat Asynchronní události.
- Kontrolér obsahuje řadič přerušení, který po vyvolání události přeruší program a obslouží přerušení. Poté se vrací zpět.
- Poslední instrukce je dokončena. Adresa následující je uložena do zásobníku a po návratu z obsluhy přerušení se pokračuje.
- PIC18 má pouze dvě lokace paměti pro vektor přerušení 0x0008 a 0x0018
- Má jen dvě priority přerušení

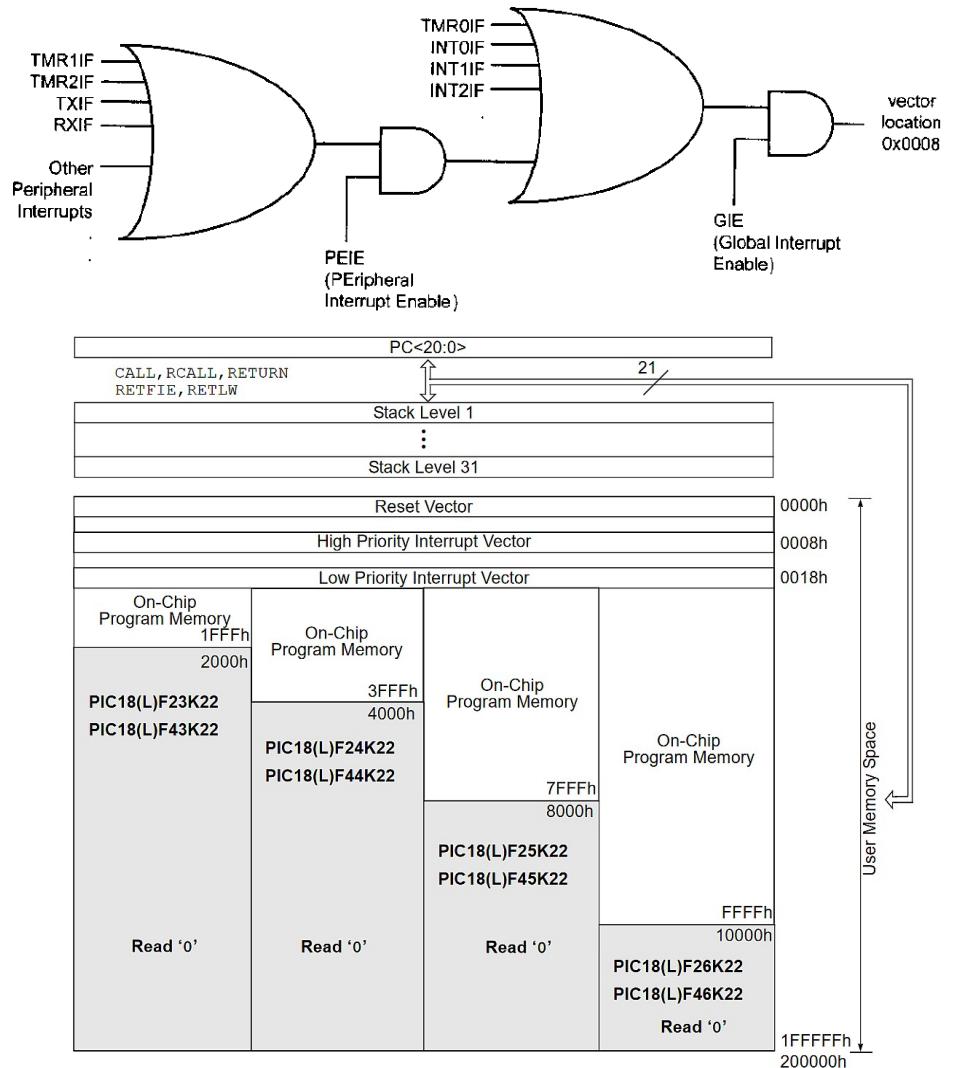


# Interrupt(přerušení)



# Interrupt(přerušení)

- Zdrojem přerušení můžou být různé události
- Kompletní seznam nalezne uživatel v datasheetu
- Lze zmínit např. přerušení od timeru, ADC převodníků, změna stavu pinu, komunikační sběrnice (příjem dat) apod.
- Ve cvičeních budeme pracovat s přerušením, které je vyvolané přetečením registru Timeru.
- Budeme tedy v přesných časových intervalech vykonávat program, který naprogramujeme do tzv. obsluhy přerušení.
- Tato obsluha se nazývá ISR tedy interrupt-service-routine.



# registry přerušení

## 9.8 Register Definitions: Interrupt Control

### REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

#### Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

bit 7

**GIE/GIEH:** Global Interrupt Enable bit

When IPEN = 0:

1 = Enables all unmasked interrupts

0 = Disables all interrupts including peripherals

When IPEN = 1:

1 = Enables all high priority interrupts

0 = Disables all interrupts including low priority

// interrupts

T1CONbits.TMR1CS = 0b00;

T1CONbits.T1CKPS = 0b11; // TMR1 prescaler

T1CONbits.TMR1ON = 1; // TMR1 on

bit 6

**PEIE/GIEL:** Peripheral Interrupt Enable bit

/\* init - interrupts \*/

PEIE = 1; // global interrupt enable

GIE = 1; // peripheral interrupt enable

TMR1IE = 1; // enable TMR1 interrupt

# registry přerušení

REGISTER 9-9: PIE1: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 1

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
bit 7							

**Legend:**

R = Readable bit

-n = Value at POR

W = Writable bit

'1' = Bit is set

U = Unimplemented bit, read as '0'

'0' = Bit is cleared

x = Bit is unknown

```
// interrupts
T1CONbits.TMR1CS = 0b00;
T1CONbits.T1CKPS = 0b11; // TMR1 prescaler
T1CONbits.TMR1ON = 1;    // TMR1 on

/* init - interrupts */
PEIE = 1;      // global interrupt enable
GIE = 1;       // peripheral interrupt enable
TMR1IE = 1;    // enable TMR1 interrupt
```

# Interrupt - proměnné

- Předání informací mezi hlavním programem a obsluhou přerušení se provádí pomocí globálních proměnných.
- Tyto proměnné musí být typu volatile, což zakazuje proměnnou optimalizovat.
- Změna této proměnné je vyvolána asynchronní událostí.
- Globalní proměnná se založí někde mimo funkci main() v globálním prostoru.
- Fakt, že se proměnná mění z více míst programu je častý zdroj problémů a chyb.

```
volatile unsigned char flag = 0; // globalni promenna

void __interrupt() T1_ISR_HANDLER(void){
    // staticka promenna nelze pouzit mimo ISR
    volatile static int i = 0
    if (TMR1IF && TMR1IE ){// kontrola priznaku
        if (i >= 500) {
            flag = 1;          // nastaveni vlajky
            i = 0;
        }
        i++;
        TMR1 = DELAY; // nastaveni registru TMR1
        TMR1IF = 0;
    }
}
```

# Zápis ISR

- Zápis se provádí vytvořením speciální funkce
- Používá se klíčové slovo `_interrupt`
- Pokud není dále specifikováno jedná se o vysokou prioritu
- proměnné, které používám v ISR by měly být deklarováný s užitím klíčového slova `volatile` (nebudou provedeny žádné optimalizace)
- Používají se globální proměnné, před funkcí `main`.

Tuto funkci ISR nelze volat z kódu.  
Provádí se s příznakem přerušení  
Interrupt flag!

```
//global variable
volatile char flag = 0;

void _interrupt ISR(void){
    if(TMR1IE & TMR1IF){
        TMR1 = 0xffff - ISR_PERIOD;
        flag = 1;
        TMR1IF = 0;
    }
}

while(1){
    if(flag){
        LED1 = ~LED1;
        flag = 0;
    }
}
```

# Zápis ISR

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:

R = Readable bit  
-n = Value at POR

W = Writable bit  
'1' = Bit is set

U = Unimplemented bit, read as '0'  
'0' = Bit is cleared  
x = Bit is unknown

Příznak přerušení je vyvolán přetečením TMR1 → začne se vykonávát ISR:

- kontrola zda je interrupt TIMER1 zapnutý a zda došlo ke změně příznaku (interrupt flag)
  - Nastavení hodnoty do TMR1 registru
  - přepsání proměnné, která se používá v hlavní smyčce
  - Smazání příznaku přerušení (interrupt flag)
- 
- Množství kódu v ISR se snažím omezovat
  - Pokud nedojde ke smazání příznaku, ISR se vyvolá okamžitě znovu!!!

```
//global variable
volatile char flag = 0;

void __interrupt ISR(void){
    if(TMR1IE & TMR1IF){
        TMR1 = 0xffff - ISR_PERIOD;
        flag = 1;
        TMR1IF = 0;
    }
}

while(1){
    if(flag){
        LED1 = ~LED1;
        flag = 0;
    }
}
```

# Použití timeru 2/4/6

```
void init(void){  
  
    TRISDbits.TRISD2 = 0;          // RD2 jako výstup  
  
    T2CONbits.T2CKPS = 0b01;      // dělení /4  
    PR2 = 200;                   // hodnota periody registru  
    T2CONbits.T2OUTPS = 0b1001;    // jednou za 10 přetížení  
    TMR2IE = 1;                  // povolení prerušení pro TMR1  
    TMR2IF = 0;                  // smazání průkazu (pro jistotu)  
    PEIE = 1;                    // povolení prerušení od periferie  
    TMR2ON = 1;                  // spuštění TMR1  
    GIE = 1;                     // globální povolení prerušení  
}
```

```
void __interrupt() T2_ISR_HANDLER(void){  
    volatile static int count=0;  
    if (TMR2IF && TMR2IE ){  
        if (count >= 500){  
            LED ^= 1;  
            count = 0;  
        }  
        count++;  
        TMR2IF = 0;  
    }  
}
```

# Více zdrojů přerušení

```
T1CONbits.TMR1CS = 0b00;  
T1CONbits.T1CKPS = 0b11; // TMR1 prescaler  
T1CONbits.TMR1ON = 1;    // TMR1 on  
  
T5CONbits.TMR5CS = 0b00;  
T5CONbits.T5CKPS = 0b11; // TMR1 prescaler  
T5CONbits.TMR5ON = 1;    // TMR1 on  
  
/* init - interrupts */  
PEIE = 1;      // global interrupt enable  
GIE = 1;       // peripheral interrupt enable  
TMR1IE = 1;    // enable TMR1 interrupt  
TMR5IE = 1;    // enable TMR1 interrupt
```

```
void __interrupt ISR(void){  
    if(TMR1IE && TMR1IF){  
        TMR1 = 0x8000;  
        LED1 ^= 1;  
        TMR1IF = 0;  
    }  
  
    if(TMR5IE && TMR5IF){  
        TMR5 = 0;  
        LED2 ^= 1;  
        TMR5IF = 0;  
    }  
  
    while(1){  
        asm("NOP");  
    }  
}
```

# Priority přerušení

```
void init(void){  
  
    TRISDbits.TRISD2 = 0;          // RD2 jako výstup  
    TRISDbits.TRISD3 = 0;          // RD3 jako výstup  
  
    IPEN = 1;                     // rozložení priorit  
    // timer 2  
    T2CONbits.T2CKPS = 0b01;      // dělení /4  
    PR2 = 200;                    // hodnota periody registru  
    T2CONbits.T2OUTPS = 0b1001;    // jednou za 10 přetíkání  
    TMR2IP = 0;                   // low priority  
    TMR2IE = 1;                   // povolení přerušení pro TMR1  
    TMR2IF = 0;                   // smazání priznaku (pro jistotu)  
    // timer 1  
    T1CONbits.TMR1CS = 0b00;      // timer 1 zdroj  
    T1CONbits.T1CKPS = 0b11;      // timer1 delicka  
    TMR1IE = 1;                   // povolení přerušení  
    TMR1IF = 0;                   // smazání priznaku  
    TMR2ON = 1;                   // spuštění TMR1  
    TMR1ON = 1;  
    GIEL = 1;                     // povolení přerušení  
    GIEH = 1;                     // povolení přerušení  
}
```

```
void __interrupt(high_priority) T1_ISR(void){  
  
    if (TMR1IF && TMR1IE ){           // kontrola priznaku IF  
        TMR1 = DELAY;  
        LED1 ^= 1;  
        TMR1IF = 0;                  // smazání IF jinak nedojde k  
    }  
  
    void __interrupt(low_priority) T2_ISR(void){  
        volatile static int count=0;  
        if (TMR2IF && TMR2IE ){           // kontrola priznaku IF  
            if (count >= 500){  
                LED2 ^= 1;  
                count = 0;  
            }  
            count++;  
            TMR2IF = 0;                  // smazání IF  
        }  
    }  
}
```

# Debouncing s ISR

Doba kmitání: Max: 10 ms

Reakce softwaru: do 50 ms

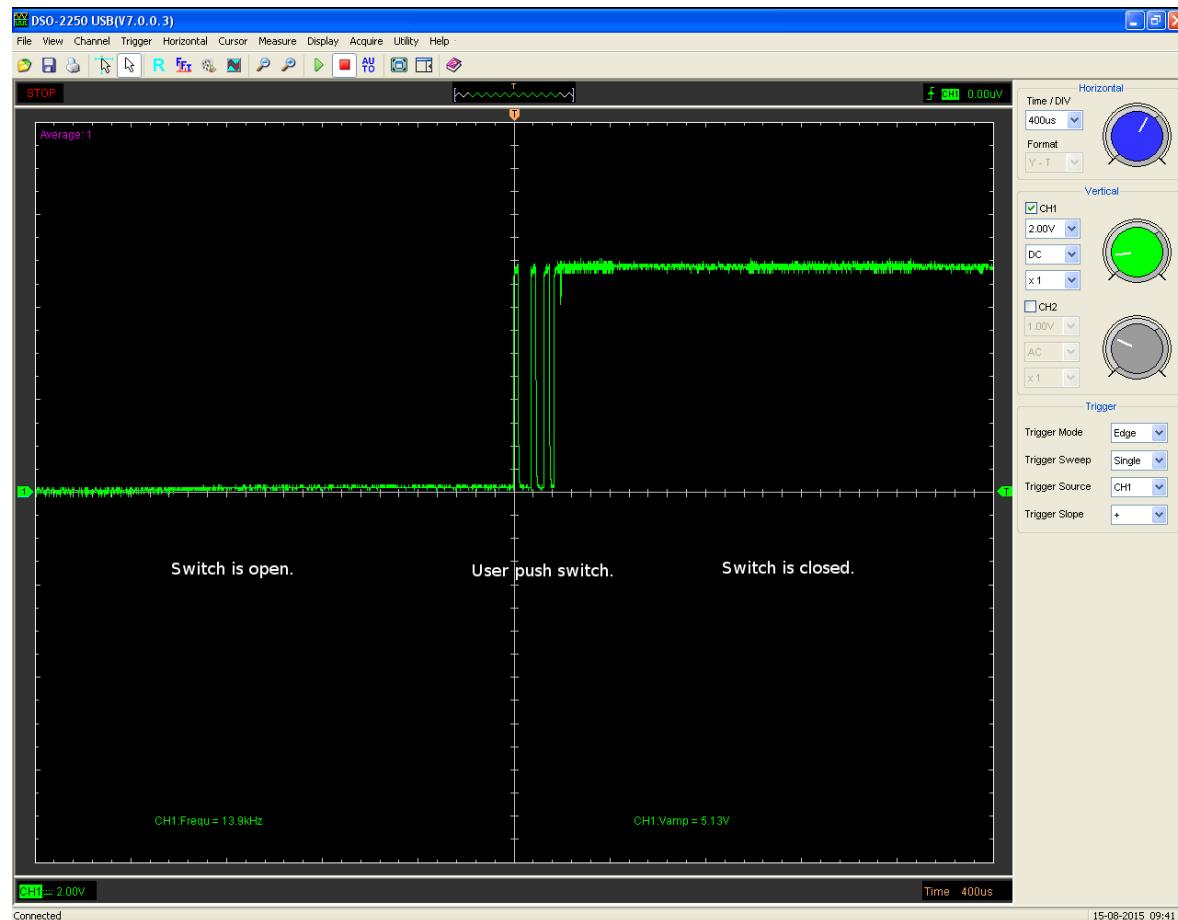
- Možné řešení je kontrolovat stisknutí v ISR.
- Perioda 3-6 ms
- Pokud je  $\text{BTN1} = 1$  tak plním akumulátor:

```
void ISR(void){  
    btnacc <= 1;  
    btnacc |= BTN1;  přidám na konec stav BTN1  
}
```

00000001  
00000011  
00000111  
....

01111111

detekce vzestupné hrany



# LCD

**RAYSTAR Optronics RX1602A3-BIW-TS:**

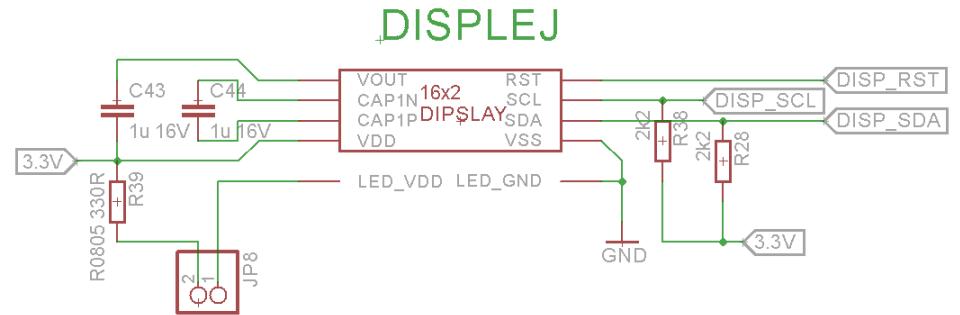
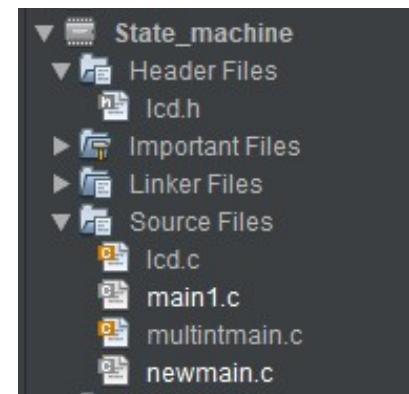
řadič: ST7032  
sběrnice: I2C

Použití:

```
#include <stdio.h>
#include "lcd.h,"

LCD_Init();
char text[16];
sprintf(text,"Mechlab je bozi!"); //funkce z stdio.h
LCD_ShowString(1,text);

if(BTN1){
    LCD_Clear(); //smazání
```



# LCD-funkce

- Umožňuje ovládat oba řádky displeje
- Uživateli slouží tyto funkce:

void LCD\_Init(void)

void LCD\_ShowString(char lineNum, char textData[])

void LCD\_Clear(void)

void LCD\_Reset(void)

Ukázka zahrnutí knihovny do projektu

```
void main(void) {  
    REV_init();  
    LCD_init();  
    bool flag = true;  
    char count = 0;  
    char text[17];  
    int32_t pot1;  
    int16_t pot2;  
  
    sprintf(text, "Mechlab je bozi!");  
    LCD_ShowString(1, text);  
    sprintf(text, "");
```