PIC krok za krokem

ASIX®

Komentované příklady programů pro PIC

1. Několiv slov úvodem

Příručka je určena k výukovému balíčku Optimum, pomocí ní si může každý osvojit základy pro práci s mikrokontroléry PIC16F84. Nesnaží se být technickou dokumentací ani popisem programovacího prostředí, ale snaží se vysvětlit na jednoduchých příkladech práci s jednotlivými částmi mikrokontroléru.

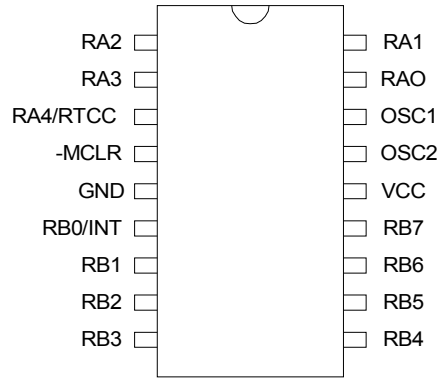
V popisu obvodu PIC16F84 jsou uvedeny pouze základní informace bez dalšího vysvětlení, přesto i tuto část je dobře si přečíst, můžete se k ní pak kdykoliv vrátit.

Vše potřebné bude dostatečně objasněno na konkrétních příkladech, které tvoří významově hlavní část této příručky.

2. Mikrokontrolér PIC16F84

Tento jednoduchý mikrokontrolér je postaven na RISC jádru. To znamená, že jeho instrukční sada je tvořena malým počtem instrukcí (čím méně instrukcí je, tím lépe se pamatují!), jejichž vykonání je však mnohem rychlejší než u mikrokontrolérů s CISC jádrem. PIC16F84 obsahuje ve své struktuře 1024 slov programové paměti typu Flash, 68 bytů paměti RAM a 64 bytů paměti EEPROM, určené k uchování konstant. Dále obvod obsahuje 8 bitový čítač TMR0 a 8 bitovou předdělička, hlídací obvod WDT, zpožděné zapnutí po náběhu napájecího napětí (PUT – Power Up Timer), ochranu Flash proti čtení (CP – Code Protect) a 13 vstup/výstupů.
Do PIC16F84 se vejde 1024 instrukcí. Je možné nadefinovat 68 proměnných v paměti. Až 64 hodnot je možné zálohovat do paměti, ve které zůstanou i po vypnutí napájení.

2.1 Zapojení PIC16F84 do obvodu



Vývod GND je zem a VCC je napájecí napětí. Na vývody OSC se připojuje krystalový rezonátor nebo jiný zdroj kmitočtu. TMR0 je vstup vnitřního čítače. INT je vstup vnějšího přerušení. Vývody RA jsou vstup/výstupy (V/V) portu A stejně jako vývody RB jsou vstupy/výstupy (V/V) portu B.

2.2 Obvod oscilátoru

PIC16F84 umožňuje připojení 4 typů oscilátoru:
- XT – připojení vnějšího krystalu do 4MHz
- HS – připojení vnějšího krystalu do 20MHz
- LP – připojení vnějšího krystalu do 200kHz
- RC – připojení vnějšího RC článku

U typů XT, HS, LP lze využít i nezávislý generátor připojený na vývod OSC1

U typu RC je RC článek připojen na OSC1 a vývod OSC2 je pak výstupem vnitřního generátoru s čtvrtinovou frekvencí.

1 instrukční cyklus (vykonání instrukce s počtem cyklů: 1) trvá 4 takty oscilátoru!

Nastavení požadovaného typu oscilátoru se provede zápisem do konfiguračního slova mikrokontroléru instrukcí:

__config konfigurační data

2.3 Obvod přerušení

Obvod PIC16F84 má několik zdrojů, které mohou vyvolat přerušení.
Hlavní rozdělení je na vnitřní a vnější.

- vnitřní: přetečení TMR0
- ukončení zápisu do EEPROM
- změna stavu na vývodech RB7-4
- vnější: vývod RB0/INT

Po přijetí přerušení mikrokontrolér skočí na adresu 0x0004 programové paměti. Protože po resetu nebo zapnutí napájení mikrokontrolér začíná na adrese 0x0000, zbývají 4 volná slova paměti mezi začátkem programu a vektorem přerušení. Toto volné místo se většinou využívá pro instrukci skoku na hlavní program a pro instrukce volání inicializačních podprogramů.

2.4 Obvod RESET

Reset nastane, přivedeme-li na vývod RESET logickou 0.
Při vyvolání resetu nebo po zapnutí napájení mikrokontrolér začíná na adrese 0x0000 programové paměti.

2.5 Paměť RAM a registry speciálních funkcí (SFR)

	Stránka 0	Stránka 1	
00	NEPŘÍMÁ ADRESA	NEPŘÍMÁ ADRESA	80
01	TMR0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORT A	TRIS A	85
06	PORT B	TRIS B	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C 4F	68 UNIVERZÁLNÍCH REGISTRŮ SRAM	MAPOVÁNO DO STRÁNKY0	8C AF

2.6 Pracovní registr W

W (work) je 8 bitový pracovní registr ALU. Je využíván téměř všemi instrukcemi.

3. Základy assembleru MPASM

Seznam zkratek:

k konstanta
f registr, se kterým instrukce pracuje
d určuje, kam je uložen výsledek operace. Je li d = 0,výsledek je uložen do pracovního registru W, je-li d = 1, výsledek je uložen do registru f. Výchozí nastavení tohoto parametru je 1.
b bit

3.1 Instrukce aritmetických a logických operací

ADDLW k
ADD Literal to W
Sečte obsah registru W s konstantou, výsledek se uloží do W.
Ovlivňuje stavové bity: C, DC, Z
Počet cyklů: 1

ADDWF f,d
ADD W to F
Sečte obsah W s registrem f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: C, DC, Z
Počet cyklů: 1

ANDLW k
AND Literal and W
Provede AND mezi registrem W a konstantou k. Výsledek je uložen do registru W.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

ANDWF f,d
AND W with F
Provede AND mezi registrem W a registrem f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

COMF f,d
COMplement F
Provede negaci (komplement) registru f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

DECF f,d
DECrement F
Zmenší obsah registru f o jedničku. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

INCF f,d
DECrement F
Zvětší obsah registru f o jedničku. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

IORLW k
Inclusive OR Literal with W
Provede OR mezi registrem W a konstantou k. Výsledek uloží do W.
Ovlivňuje stavové bity: Z

Počet cyklů: 1

IORWF f,d

Inclusive OR W with F

Provede OR mezi registrem W a registrem f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.

Ovlivňuje stavové bity: Z

Počet cyklů: 1

SUBLW k

SUBtract W from Literal

Odečte obsah registru W od konstanty k. Výsledek je uložen do registru W.

Ovlivňuje stavové bity: C, DC, Z

Počet cyklů: 1

SUBWF f,d

SUBtract W from F

Odečte obsah W od registru f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.

Ovlivňuje stavové bity: C, DC, Z

Počet cyklů: 1

XORLW k

eXclusive OR Literal with W

Provede XOR mezi registrem W a konstantou k. Výsledek uloží do W.

Ovlivňuje stavové bity: Z

Počet cyklů: 1

XORWF f,d

eXclusive OR W with F

Provede XOR mezi registrem W a registrem f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.

Ovlivňuje stavové bity: Z

Počet cyklů: 1

3.2 Instrukce nulování a nastavení

BCF f,b

Bit Clear F

Vynuluje bit b registru f.

Ovlivňuje stavové bity: -

Počet cyklů: 1

BSF f,b

Bit Set F

Nastaví bit b registru f do 1.

Ovlivňuje stavové bity: -

Počet cyklů: 1

CLRF f

CLeaR F

Vynuluje obsah registru f.

Ovlivňuje stavové bity: Z

Počet cyklů: 1

CLRW
CLeaR W register
Vynuluje obsah registru W.
Ovlivňuje stavové bity: Z
Počet cyklů: 1

CLRWDT
CleaR WatchDog Time
Vynuluje WDT a předděličku, když je k němu připojená.
Ovlivňuje stavové bity: TO = 1, PD = 1
Počet cyklů: 1

3.3 Instrukce přesunu dat

MOVF f,d
MOVe F
Přesune obsah registru f je-li d = 0, do registru W, je-li d = 1 zpět do registru F
Ovlivňuje stavové bity: Z
Počet cyklů: 1

MOVLW k
MOVe Literal to W
Přesune konstantu k do registru W
Ovlivňuje stavové bity: -
Počet cyklů: 1

MOVWF f
Move W to F
Přesune obsah registru W do registru f.
Ovlivňuje stavové bity: -
Počet cyklů: 1

RLF f,d
Rotate Left F through carry
Rotuje obsah registru f o jeden bit doleva přes C bit stavového registru. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: C
Počet cyklů: 1

RRF f,d
Rotate Right F through carry
Rotuje obsah registru f o jeden bit doprava přes C bit stavového registru. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: C
Počet cyklů: 1

SWAPF f,d
SWAP F
Prohodí horní a dolní půlbyte registru f. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f.
Ovlivňuje stavové bity: -
Počet cyklů: 1

3.4 Instrukce podprogramů a přerušení

CALL
subroutine CALL
Zavolá podprogram.
Ovlivňuje stavové bity: -
Počet cyklů: 2

RETLW k
RETurn Literal to W
Navrátí se z podprogramu. Registr W naplní konstantou k.
Ovlivňuje stavové bity: -
Počet cyklů: 2

RETURN
RETurn from subroutine
Navrátí se z podprogramu.
Ovlivňuje stavové bity: -
Počet cyklů: 2

RETFIE
RETurn From Interrupt
Navrátí se z podprogramu obsluhujícího přerušení.
Ovlivňuje stavové bity: GIE = 1
Počet cyklů: 2

3.5 Instrukce skoků

BTFSC f,b
Bit Test F, Skip if Clear
Je-li bit b registru f = 0, přeskočí následující instrukci (provede místo ní instrukci NOP)
Ovlivňuje stavové bity: -
Počet cyklů: 1(2)

BTFSS f,b
Bit Test F, Skip if Set
Je-li bit b registru f = 1, přeskočí následující instrukci (provede místo ní instrukci NOP).
Ovlivňuje stavové bity: -
Počet cyklů: 1(2)

DECFSZ f,d
DECrement F, Skip if Zero
Od obsahu registru f odečte jedničku. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f. Je-li výsledek po odečtení 0, přeskočí se následující instrukce (provede se místo ní instrukce NOP).
Ovlivňuje stavové bity: -
Počet cyklů: 1(2)

GOTO k

GO TO

Provede nepodmíněný skok na adresu k.

Ovlivňuje stavové bity: -

Počet cyklů: 1(2)

INCFSZ f,d

INCrement F, Skip if Zero

K obsahu registru f přičte jedničku. Je-li d = 0, výsledek se uloží do W, je-li d = 1, výsledek se uloží do f. Je-li výsledek po přičtení 0, přeskočí se následující instrukce (provede se místo ní instrukce NOP).

Ovlivňuje stavové bity: -

Počet cyklů: 1(2)

3.6 Zvláštní instrukce

NOP

No OPeration

Prázdná operace. Nic se neprovede.

Ovlivňuje stavové bity: -

Počet cyklů: 1

OPTION

load OPTION register

Přesune obsah registru W do registru OPTION.

Ovlivňuje stavové bity: -

Počet cyklů: 1

SLEEP

SLEEP

Mikrokontrolér přejde do stavu SLEEP. Vynuluje WDT a předděličku.

Ovlivňuje stavové bity: PD = 0, TO = 1

Počet cyklů: 1

TRIS f

load TRIS register

Přesune obsah registru W do registru TRIS portu f.

Ovlivňuje stavové bity: -

Počet cyklů: 1

3.7 Direktivy assembleru

INCLUDE

Syntaxe**:** **INCLUDE** „soubor"

Vloží soubor s defincemi, podprogramy, knihovnami

Příklad: `include „C:\MPLAB\PICPIC16F84.EQU`

EQU

Syntaxe**:** název konstanty **EQU** hodnota

Definice konstanty v programu

Příklad: `teplota equ 0x0D`

ORG

Syntaxe**:** (návěští) **ORG** hodnota
Nastaví adresu na kterou se uloží následující program

Příklad:
```
org    0x0004
call   I2C
```

#define

Syntaxe**:** **#define** název registr,bit
definuje název pro bit registru

Příklad:
```
#define   LED    porta, 0
```

MACRO
ENDM

Syntaxe**:** název **MACRO**
vytvoří makroinstrukci z instrukcí uzavřených mezi MACRO a ENDM.

Příklad:
```
bnk0   macro
bcf    RP0
endm
```

LIST P

Syntaxe**:** `LIST P= typ procesoru`
říká překladači, pro jaký procesor je program napsán

Příklad:
```
LIST P = PIC16F84
```

END

Konec programu

;

Oddělí komentář od vlastního programu od středníku do konce řádku.

3.8 Používané číselné formáty

Dekadické: D'100'

Dekadický formát je dobré používat, kvůli lepší čitelnosti, pro zadávání hodnot do čekacích smyček.

Hexadecimální:0FEH, 0xFE

Hexadecimální tvar se používá tam, kde zadáváme hodnotu celého bytu.

Binární: B'10001011'

Binární tvar je vhodné použít při nastavování bitů ve stavovém slově, registrech speciálních funkcí atd.

Toto je pouze doporučené použití různých číselných formátů, každému může vyhovovat jiný formát a je pouze na programátorovi, aby si zvolil ten nejpřijatelnější.

3.9 Adresování

K přístupu do paměti RAM je možné použít dva způsoby:

- Přímé adresování: Zapisujeme přímo na adresu (0 – 7F) registru dané stránky paměti.
- Nepřímé adresování: Adresu (0 – FF) zapíšeme do registru FSR, zápisem nebo čtením registru NEPŘÍMÁ ADRESA přistupujeme k námi požadovanému registru bez ohledu na stránku paměti.

Příklad přímého adresování

```
r1      equ    0xC0
        .
        .
        movwf  r1
```

Příklad nepřímého adresování

```
f0      equ    0x00    ;registr NEPŘÍMÁ ADRESA
r1      equ    0xC0    ;registr r1 leží na adrese 0xc0
        movlw  r1      ;adresu registru r1 přesune do W
        movwf  fsr     ;adresu registru r1 přesune do fsr
        movlw  0x56    ;do registru W zapíše číslo 0x56
        movwf  f0      ;zapíše číslo 0x56 do registru r1
```

Chceme-li se pomocí přímého adresování dostat k registrům stránky 1 paměti, musíme aktuální stránku přepnout pomocí nastavení registru Status.

3.10 Způsob psaní programu:

Aby byl odlišen kód programu od návěští a deklarace proměnných, platí toto pravidlo:
Deklarace proměnných a všechna návěští začínají od začátku řádky. Názvy instrukcí jsou odsazeny o tabelátor doprava. Parametry instrukcí jsou odsazeny o další tabelátor doprava.

Za návěštím MASM nepožaduje psaní dvojtečky. U velkých programů je však vhodné dvojtečky za návěštím psát. Při prohledávání souborů se zadá název s dvojtečkou a vyhledávač najde začátek podprogramu a ne místo, odkud se volá.

Všechny komentáře musí být od kódu programu odděleny středníkem.

4. Příklady programů

Každý program by kromě vlastních instrukcí měl obsahovat i další informace, tzv. hlavičku. Hlavičku programu obvykle tvoří název programu, jméno autora, datum poslední úpravy, použitý překladač. Vzhledem k velikosti ukázkových programů není hlavička v příkladech uvedena.

Ve všech programech budeme používat toto nastavení konfiguračního slova mikrokontroléru:

oscilátor: XT
WDT: OFF
PUT: ON
CP: OFF

Pro toto nastavení se konfigurační slovo rovná 0x3FF1.

Pro zvýšení přehlednosti doporučuji v programech označovat bity velkými písmeny, registry malými písmeny a náveští velkým počátečním písmenem a dále malými písmeny.
Je to ale pouze na Vás. Překladač dovoluje nastavení tzv. Case sensitivity, to znamená rozlišování malých a velkých písmen . Je-li překladač zavolán s parametrem /C-, nerozlišuje malá a velká písmena ve zdrojovém kódu. Výchozí nastavení překladače v prostředí emulátoru MU-Alpha je /C-. To znamená, že proměnné Pokus a pokus jsou totožné.

4.1 Úloha 1: Drát

Zadání úlohy

Při stisknutém tlačítku se rozsvítí LED.

Účel úlohy

Naučit se používat vstupy a výstupy mikrokontroléru.

Rozbor úlohy:

Určíme si V/V pin obvodu, na který připojíme tlačítko (vstup) a V/V pin, na který připojíme LED (výstup).

Poznámka

Připojení je již realizováno na vývojové desce EduKit84 a naše práce bude spočívat pouze v softwarovém výběru zvolených V/V.

Přečteme stav pinu, na kterém je připojeno tlačítko. Je-li tlačítko sepnuto (tj. na vstupu je logická 0 – viz zapojení EduKit84), rozsvítíme LED (na výstup pošleme logickou 0 – viz zapojení EduKit84).

Vytvoříme programovou smyčku, ve které se bude neustále opakovat čtení tlačítka a případné rozsvícení LED.

Trocha teorie:

Asi už Vás napadlo, jak asi určíme, který V/V bude v našem případě vstupem a který výstupem. SFR obsahují registry TRISA a TRISB, které v sobě mají uloženou informaci o tom, jaký V/V je vstupem a jaký výstupem. Zapíšeme-li do odpovídajícího bitu těchto registrů 1, V/V pin se

stane vstupem (což je také stav po zapnutí obvodu), zapíšeme-li 0, V/V pin se stane výstupem.

TRISA:

-	-	-	RA4	RA3	RA2	RA1	RA0

TRISB:

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Když je pin vstupní, můžeme ho číst, když je výstupní, můžeme na něj zapsat buď logickou 0 nebo 1. To uděláme zápisem do registru PORTA nebo PORTB, které jsou také mezi SFR.

PORTA:

-	-	-	RA4	RA3	RA2	RA1	RA0

PORTB:

RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

Máme tu ale další problém. Chceme-li totiž zapisovat do registru TRISA nebo TRISB, musíme zapisovat do stránky 1 paměti RAM.To lze, jak již víme, buď přímo nebo nepřímo. Zvolíme přímý zápis. Pak tedy musíme přepnout aktuální stránku paměti RAM na stránku 1. To lze provést pomocí bitu RP0 v registru STATUS. Je-li RP0 = 0, je nastavena stránka 0. Je-li RP0 = 1, je nastavena stránka 1.

SWR:

IRP	RP1	RP0	TO	PD	Z	DC	C

IRP a RP1 jsou pro nás univerzální bity, které můžeme použít, jak chceme. U nástupců mikrokontroléru PIC16F84 jsou to další bity pro výběr stránek pamětí !

- RP0 vybírá stránku paměti RAM: 0 - stránka 0
  1 - stránka 1

- TO a PD jsou stavové bity, které indikují události jako zapnutí napájení, reset, probuzení ze sleep atd.
- Z je nastaven do 1, je-li výsledek aritmetické nebo logické operace roven 0.
- DC indikuje přenos (výpůjčku) v operacích sčítání (odčítání) pro dolní 4 bity
- C indikuje přenos (výpůjčku) v operacích sčítání (odčítání), používá se také v operacích rotace.

Program

```
;*******************************************************
status   equ    0x03       ;status je na adrese 0x03
porta    equ    0x05
trisa    equ    0x05       ;použijeme přímé adresování
portb    equ    0x06
trisb    equ    0x06       ;použijeme přímé adresování
;*******************************************************
#define  TL     portb,0    ;tlačítko na pinu RB0
#define  LED    porta,4    ;LED je na pinu RA4
#define  RP0    status,5   ;RP0 je 5.bit registru status
;*******************************************************
         list       p = PIC16F84
         __config   0x3FF1         ;nastavení konfigurace
;*******************************************************
         org    0x0000        ;program je uložen od adresy ;
                              0x00
         bsf    RP0           ;stránka 1 paměti RAM
         movlw  B'11101111'
         movwf  trisa         ;pin RA4 je výstup
         movlw  B'11111111'
         movwf  trisb         ;piny portu RB jsou vstupy
         bcf    RP0           ;stránka 0 paměti RAM
Main     btfss  TL            ;je TL 0 nebo 1?
         goto   Main_A        ;je-li TL 0, skočí na Main_A
         bsf    LED           ;TL je 1, zhasne LED
         goto   Main          ;uzavírá smyčku
Main_A   bcf    LED           ;rozsvítí LED
         goto   Main          ;uzavírá smyčku
         end                   ;konec programu
```

4.2 Úloha 2: Blik

Zadání úlohy:
LED bude nezávisle blikat. Využijte nepřímé adresování při zápisu do registru trisa.

Účel úlohy
Použití vnořených čekacích smyček. Využití nepřímého adresování.

Rozbor úlohy
Vytvoříme si podprogram, který bude přesně definovanou dobu čekat. Rozsvítíme LED a zavoláme podprogram čekání, LED zhasneme a opět zavoláme podprogram čekání. Nakonec uzavřeme programovou smyčku skokem zpět, na instrukci rozsvěcující LED.

Trocha teorie
Časové smyčky tvoří důležité části programů, proto doporučuji věnovat zvláštní pozornost pochopení této kapitoly.
Můžeme rozlišit několik základních typů časových smyček:

- jednoduchá
- vnořená
- několikanásobně vnořená

Základní pravidlo výpočtu časových smyček
Podle počtu potřebných instrukčních cyklů vypočítáme čas, potřebný k vykonání jádra nejvnitřnější smyčky. Tento čas vynásobíme počtem opakování smyčky. Přičteme čas potřebný k naplnění registru, se kterým nejvnitřnější smyčka pracuje. Celé to vynásobíme počtem opakování nadřazené smyčky a opět připočteme čas potřebný k naplnění registru této smyčky, atd. Pokud jde o podprogram, přičteme nakonec ještě čas potřebný k jeho zavolání a k návratu zpět do programu.

Jednoduchá časová smyčka

```
;*******************************************************
x       equ    D'100'  ;konstanta časové smyčky
cnt     equ    0x0C    ;definice registru čítače

;*******************************************************
        movlw  x
        movwf  cnt     ;naplnění registru čítače hodnotou x
Loop    nop            ;tělo smyčky
        nop            ;tělo smyčky
        decfsz cnt, 1 ;odečte od cnt 1, pokud je výsledek 0,
        goto   Loop    ;tak je goto nahrazeno instrukcí nop
;*******************************************************
```

Na časové smyčce nás bude nejvíce zajímat, jak dlouho bude trvat. Při výpočtu je dobré předem uvažovat, zda je pro nás přesné trvání smyčky nezbytné nebo jestli potřebujeme pouze přibližnou hodnotu.

Pokud nám stačí přibližná hodnota, uvažujeme takto:

- sečteme počet instrukčních cyklů instrukcí v těle smyčky včetně rozhodovací instrukce a instrukce skoku na začátek smyčky, tj. nop(1 cyklus) + nop(1 cyklus) + decfsz(1 cyklus) + goto(2 cykly) = 5 cyklů
- počet cyklů, potřebných k jednomu průchodu smyčkou vynásobíme počtem opakování: 5 x 100 = 500
- naše smyčka bude tedy trvat přibližně 500 instrukčních cyklů, je-li jeden instrukční cyklus např. 1,22μs (krystal 3,2768 MHz), pak bude smyčka trvat přibližně 610 μs.

Požadujeme-li přesnou hodnotu, zpřesníme původní úvahu:

- k vypočítané hodnotě musíme přičíst i instrukce, které naplňují registr čítače movlw (1 cyklus) a movwf (1 cyklus)
- protože je při posledním průchodu smyčkou nahrazena instrukce goto (2 cykly) instrukcí nop (1 cyklus), musíme odečíst jeden cyklus.
- 500 + 2 - 1 = 501 tj. při instrukčním cyklu 1,22μs 611,22μs.
  Vidíte, že se přibližný výpočet od skutečnosti v tomto případě příliš neliší.

Vnořená časová smyčka:

Vnořená smyčka znamená, že jedna smyčka je umístěna ve druhé časové smyčce, pak dochází k násobení konstant jednotlivých smyček. Dočítá-li vnitřní smyčka do 0, je znovu naplněn její registr na původní hodnotu. To se děje tak dlouho, dokud není registr vnější smyčky roven 0. Pro zjednodušení použijeme jako vnitřní smyčku z minulého příkladu.

```
;*******************************************************
x       equ    D'100'    ;konstanta vnitřní smyčky
y       equ    D'10'     ;konstanta vnější smyčky
cnt1    equ    0x0C      ;registr vnitřní smyčky
cnt2    equ    0x0D      ;registr vnější smyčky
;*******************************************************
        movlw   y
        movwf   cnt2      ;naplnění registru vnější
                          smyčky ;hodnotou y
Loop_A  movlw   x
        movwf   cnt1      ;naplnění registru vnitřní
                          smyčky ;hodnotou x
Loop_B  nop               ;tělo vnitřní smyčky
        nop               ;tělo vnitřní smyčky
        decfsz  cnt1, 1 ;odečte od cnt1 1, je-li výsledek 0,
        goto    Loop_B  ;tak je goto nahrazeno instrukcí nop
        decfsz  cnt2, 1 ;odečte od cnt2 1, je-li výsledek 0,
        goto    Loop_A  ;tak je goto nahrazeno instrukcí nop
;*******************************************************
```

Pro výpočet platí stejné možnosti jako u minulého příkladu, pro ukázku použijeme pouze přesnější způsob:

- výpočet vnitřní smyčky je naprosto identický, tj. 501 instrukčních cyklů
- k počtu instrukčních cyklů vnitřní smyčky přičteme instrukce decfsz (1 cyklus), goto (2 cykly) a dostaneme tak počet instrukčních cyklů těla vnější smyčky: 501 + 2 + 1 = 504
- vynásobíme počet instrukčních cyklů těla vnější smyčky počtem jejích opakování, tj. konstantou vnější smyčky: 504 x 10 = 5040
- přičteme instrukce, které naplňují registr vnější smyčky, a odečteme jeden instrukční cyklus, protože goto je při posledním průchodu vnější smyčkou nahrazeno instrukcí nop.
- 5040 + 2 - 1 = 5041

- vnořená smyčka bude trvat 5041 instrukčních cyklů, je-li jeden instrukční cyklus např. 1,22μs (krystal 3,2768MHz), pak bude smyčka trvat 6,15 ms.

Poznámka

Můžeme samozřejmě konstruovat i smyčky s vícenásobným vnořením. V čím vnořenější smyčce měníme počet cyklů těla smyčky, tím větší změna je díky násobení konstant ve výsledku.

Potřebujeme-li naprosto přesnou časovou smyčku a její hodnotu se nám nepodaří zkonstruovat, sestavíme smyčku s menším počtem instrukčních cyklů a doplníme ji vhodně instrukcemi nop. Můžeme také použít pomocnou smyčku, kterou sečteme s hlavní smyčkou.

Nepřímé adresování

- adresu registru, do kterého chceme zapisovat nebo z něho číst, zapíšeme do registru FSR
- chceme-li nyní pracovat s námi vybraným registrem, pracujeme s registrem f0 (nepřímá adresa)
- nemusíme dbát na to, v jaké stránce paměti RAM se adresovaný registr nachází

Doporučení

Používáme-li v programu instrukce, jejichž druhým parametrem je číslo, které určuje, kam se má ukládat výsledek operace, např. decfsz, movf, addwf atd., je dobré na začátek našeho programu psát tuto definici:

```
w       equ     0
f       equ     1
```

Chceme-li pak, aby se výsledek operace uložil do registru (registr - obecně f), píšeme instrukci ve tvaru

```
decfsz  cnt,f
```

Má-li se výsledek operace uložit do w, píšeme instrukci ve tvaru

```
addwf   disp,w
```

Program se pak stane čitelnějším a o to bychom se měli snažit.

Program

```
;*****************************************************
w        equ   0        ;parametr instrukcí
f        equ   1        ;parametr instrukcí
f0       equ   0x00     ;registr nepřímého adresování
status   equ   0x03     ;status je na adrese 0x03
fsr      equ   0x04     ;registr adresy nepřímého adresování
porta    equ   0x05
trisa    equ   0x85     ;použijeme nepřímé adresování
cnt1     equ   0x0C     ;definice registru cnt1
cnt2     equ   0x0D     ;definice registru cnt2
cnt3     equ   0x0E
;*****************************************************
```

```
#define  LED  porta,4 ;LED je na pinu RA4
;*******************************************************
list   p = PIC16F84
       __config     0x3FF1   ;nastavení konfigurace
;*******************************************************
org       0x0000     ;adresa začátku programu
goto      Main       ;skočí na začtek hlavního programu
;*******************************************************
Cekej     movlw   D'250'
          movwf   cnt3      ;naplnění registru 3.smyčky
Cekej_A   movlw   D'100'
          movwf   cnt2      ;naplnění registru 2.smyčky
Cekej_B   movlw   D'10'
          movwf   cnt1      ;naplnění registru 1.smyčky
Cekej_C   decfsz  cnt1,f    ;odečte od registru 1.smyčky 1
          goto    Cekej_C   ;skočí, je-li registr roven 0
          decfsz  cnt2,f    ;odečte od registru 2.smyčky 1
          goto    Cekej_B   ;skočí, je-li registr roven 0
          decfsz  cnt3,f    ;odečte od registru 3.smyčky 1
          goto    Cekej_A   ;skočí, je-li registr roven 0
          return            ;návrat do hlavního programu
;*******************************************************
Main      movlw   trisa
          movwf   fsr           ;registr trisa bude
                                nepřímo ;adresován
          movlw   B'11101111'
          movwf   f0            ;zápis do registru trisa
Main_A    bcf     LED           ;rozsvítí LED
          call    Cekej         ;zavolá podprogram
                                s čekací ;smyčkou
          bsf     LED           ;zhasne LED
          call    Cekej         ;zavolá podprogram
                                s čekací ;smyčkou
          goto    Main_A        ;skočí na začátek smyčky
          end
```

4.3 Úloha 3: Blikání

Zadání úlohy
Rychlost blikání LED se mění podle toho, zda je stisknuté tlačítko. Pro časování použijte vnitřní hardwarový časovač.

Účel úlohy
Naučit se pracovat s vnitřním hardwarovým časovačem.

Rozbor úlohy
Vytvoříme podprogram, kterému budeme přes universální bit předávat délku časového zpoždění. Tento podprogram bude obsahovat smyčku s proměnnou konstantou, ve které se bude čekat na přetečení vnitřního časovače. Hlavní program bude tvořen testováním tlačítka, nastavením příslušného universálního bitu, rozsvícením LED, zavoláním podprogramu

čekání, zhasnutím LED, zavoláním podprogramu čekání a skokem na začátek smyčky hlavního programu.

Trocha teorie

Vnitřní čítač

Mikrokontrolér PIC16F84 má ve své struktuře integrovaný 8-bitový čítač s 8-bitovou nastavitelnou předděličkou. Vnitřní čítač je nazýván TMR0. Zvyšovat obsah TMR0 lze pomocí externího vstupu RA4/TMR0 nebo vnitřním oscilátorem. Externí vstup je vybaven Schnittovým klopným obvodem a můžeme si vybrat, zda se má čítač inkrementovat náběžnou nebo sestupnou hranou vstupního signálu. Používáme-li k inkrementování čítače vnitřní oscilátor, pak čítač zaznamenává každý instrukční cyklus (frekvence oscilátoru / 4). Když dojde k přetečení TMR0 je nastaven příslušný bit v registru INTCON a je spuštěn systém přerušení (pokud je přerušení povoleno). S registrem TMR0 můžeme pracovat jako s kterýmkoliv jiným registrem, zápis do tohoto registru ale vyvolá vynulování předděličky (pokud je připojena k TMR0).

OPTION:

-RBPU	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0

-RBPU povoluje vnitřní PULL-UP odpory zapojené na port B:
0 - povoleny **1** - zakázány

INTEDG určuje aktivní hranu vnějšího signálu pro aktivaci přerušení :
0 - spádová hrana **1** - náběžná hrana

RTS určuje zdroj signálu pro TMR0:
0 - vnitřní oscilátor / 4 **1** - vstup RA4-TMR0

RTE určuje aktivní hranu pro práci s TMR0:
0 - náběžná hrana **1** - sestupná hrana

PSA určuje připojení předděličky:
0 - před TMR0 **1** - za WDT (WatchDog Timer)

PS2, PS1, PS0 určují dělící poměr předděličky dle následující tabulky

PS2	PS1	PS0	před TMR0	za WDT
0	0	0	1 : 2	1 : 1
0	0	1	1 : 4	1 : 2
0	1	0	1 : 8	1 : 4
0	1	1	1 : 16	1 : 8
1	0	0	1 : 32	1 : 16
1	0	1	1 : 64	1 : 32
1	1	0	1 : 128	1 : 64
1	1	1	1 : 256	1 : 128

INTCON:

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

GIE znamená globální povolení přerušení (Global Interrupt Enable):
1 - přerušení povoleno **0** - přerušení zakázáno
EEIE povoluje přerušení od ukončení zápisu do EEPROM:
1 - přerušení povoleno **0** - přerušení zakázáno
T0IE povoluje přerušení od TMR0:
1 - přerušení povoleno **0** - přerušení zakázáno
INTE povoluje vnější přerušení:
1 - přerušení povoleno **0** - přerušení zakázáno
RBIE povoluje přerušení od změny na pinech 4-7 portu B:
1 - přerušení povoleno **0** - přerušení zakázáno
T0IF indikuje přetečení TMR0:
0 - k přetečení nedošlo **1** - došlo k přetečení
INTF indikuje požadavek vnějšího přerušení:
0 - požadavek nenastal **1** - požadavek nastal
RBIF indikuje změnu na pinech 4-7 portu B:
0 - změna nenastala **1** - změna nastala

Řešení
Máme-li v aplikaci zapojen krystal 3,2768MHz, můžeme jeho frekvenci jednoduše vydělit číslem 2^{15}. Vzniknou nám tak intervaly 0,01s. Oscilátor je dělen 4 (2^2). Rozsah TMR0 je 8 bitů tj. 2^8, na předděličku nám tedy zbývá 2^5 tj. musíme nastavit dělící poměr 1 : 32.

Na přetečení budeme čekat ve smyčce, ve které budeme testovat bit T0IF, dokud nebude 1. Po přetečení TMR0 nesmíme zapomenout bit T0IF vynulovat.

Poznámka

Při pojmenovávání registrů si musíme dát pozor, abychom nepoužili klíčová slova. Např. option nemůže být jméno registru, protože je to existující název instrukce.

Program

```
;*******************************************************
w         equ     0         ;parametr instrukcí
f         equ     1         ;parametr instrukcí
TMR0      equ     0x01
optreg    equ     0x01      ;použijeme přímé adresování
status    equ     0x03      ;status je na adrese 0x03
porta     equ     0x05
trisa     equ     0x05      ;použijeme přímé adresování
portb     equ     0x06
trisb     equ     0x06      ;použijeme přímé adresování
intcon    equ     0x0B
cnt       equ     0x0C      ;definice registru cnt1
```

```
;*******************************************************
#define   LED     porta,4    ;LED je na pinu RA4
#define   TL      portb,0    ;tlačítko je na pinu RB0
#define   T0IF    intcon,2   ;indikace přetečení TMR0
#define   RP0     status,5
#define   BIT     status,7   ;universální bit
;*******************************************************
          list      p = PIC16F84
          __config  0x3FF1     ;nastavení konfigurace
;*******************************************************
          org     0x0000       ;adresa začátku programu
          goto    Main
;*******************************************************
Cekej     bcf     T0IF         ;vynulování bitu T0IF
clrf      tmr0                 ;vynulování TMR0 a předděličky
          movlw   D'10'
          btfsc   BIT
          movlw   D'5'
          movwf   cnt
Cekej_A   btfss   T0IF         ;testování přetečení TMR0
          goto    Cekej_A      ;nepřetekl-li TMR0,
                               testuje ;znovu
          bcf     T0IF         ;vynulování bitu T0IF
          decfsz  cnt,f
          goto    Cekej_A      ;vnější smyčka
          return
;*******************************************************
Main      bsf     RP0          ;stránka 1 paměti RAM
          movlw   B'11101111'
          movwf   trisa        ;pin RA4 je výstup
          movlw   B'11010100'
          movwf   optreg       ;konfigurace TMR0
                               a ;předděličky
          bcf     RP0          ;stránka 0 paměti RAM
Main_A    bcf     BIT          ;tlačítko vypnuto -
                               blikání ;po 1s
          btfss   TL
          bsf     BIT          ;tlačítko sepnuto - blikání
                               ;po 0,5s
          bcf     LED          ;rozsvítí LED
          call    Cekej
          bsf     LED          ;zhasne LED
          call    Cekej
          goto    Main_A
          end
```

4.4 Úloha 4: Displej

Zadání úlohy
Při stisku jednoho z osmi tlačítek se na displeji objeví číslo 1 až 8.

Účel úlohy
Seznámení se základy práce se sedmisegmentovým displejem.

Rozbor úlohy
Vytvoříme podprogram, který podle parametru v registru W (1 - 8) se kterým je zavolán, vrátí číslo v registru W, které odpovídá kombinaci rozsvícených segmentů sedmisegmentového zobrazovače požadovaného znaku (1 - 8). V hlavním programu testujeme tlačítka (port B je vstupní). Je-li některé tlačítko stisknuté, uložíme jeho pořadové číslo (1 - 8) do registru W a zavoláme náš podprogram. Jeho výsledek pošleme na port B (port B je výstupní). Zapneme sedmisegmentový zobrazovač vynulováním příslušného bitu registru A. Zavoláme podprogram čekání (asi 0,4s - téměř libovolná hodnota). Poslední instrukce zajistí skok na začátek hlavní smyčky programu.

Trocha teorie
Mikrokontrolér PIC16F84 používá tzv. čítač instrukcí PC (Program Counter), podle kterého adresuje svojí vnitřní paměť Flash. V čítači instrukcí je adresa (pořadí) následující instrukce, která bude vykonána. Během provádění instrukce se čítač instrukcí inkrementuje. U PIC16F84 je použit 13-bitový PC. Neboť má paměť Flash kapacitu jenom 1024 slov, využívá se z PC pouze jeho dolních 10 bitů. Instrukce goto, call, return, retlw naplňují všech 10 bitů, to znamená, že volání podprogramů a návraty z nich mohou být umístěny v kterékoliv části paměti Flash. Nižších 8 bitů z PC je plně přístupných pomocí registru PCL, chceme-li zapsat i do vyšších bitů, můžeme k tomu použít registr PCLATH.

Tabulka v programu:
Tabulku, která vrací číslo, uložené na požadované pozici, můžeme vytvořit právě pomocí čítače instrukcí. Podprogram, který bude tabulku představovat, nejdříve přičte k registru PCL hodnotu z registru W (požadovaná pozice v tabulce). Následovat budou instrukce návratu retlw, jejichž parametry budou konstanty, které budou představovat položky v tabulce.

Zavoláme-li tento podprogram, bude do PC přičten offset položky, jejíž hodnotu budeme požadovat. Další instrukce, která bude vykonána, bude příslušná instrukce retlw. Ta způsobí návrat z podprogramu s obsahem položky tabulky v registru W.

Poznámka
Protože ovlivňujeme pouze dolních 8 bitů, musí být celá tabulka v jednom bloku o velikosti 256 slov. Registr PCL nám tedy nesmí v žádném případě přetéci.

```
Tabulka  addwf   pcl,f     ;přičte obsah W k čítači instrukcí
         retlw   0x05      ;0. položka tabulky
         retlw   0x07      ;1. položka tabulky
         retlw   0x04      ;2. položka tabulky
         retlw   0x08      ;3. položka tabulky
;*****************************************************
         movlw   0x03
         call    Tabulka   ;Vrátí v registru W 3.
                           položku ;tabulky
```

Jestli Vám nejsou předcházející řádky úplně jasné, doporučuji následující program pečlivě odkrokovat pomocí emulátoru.

Poznámka

Časové zpoždění po zobrazení je potřeba, aby byla zajištěna mnohonásobně delší doba, kdy displej svítí, oproti době, kdy je zhasnut (port B je vstupní a testujeme tlačítka).

Program

```
;*****************************************************
w          equ    0         ;parametr instrukcí
f          equ    1         ;parametr instrukcí
tmr0       equ    0x01
optreg     equ    0x01      ;použijeme přímé adresování
pcl        equ    0x02
status     equ    0x03      ;status je na adrese 0x03
porta      equ    0x05
trisa      equ    0x05      ;použijeme přímé adresování
portb      equ    0x06
trisb      equ    0x06      ;použijeme přímé adresování
intcon     equ    0x0B
disp       equ    0x0C      ;pamět znaku
;*****************************************************
#define   RP0    status,5
#define   T0IF   intcon,2   ;indikace přetečení TMR0
;*****************************************************
       list       p = PIC16F84
        __config    0x3FF1        ;nastavení konfigurace
;*****************************************************
         org       0x0000
         goto      Main
;*****************************************************
Cekej    bcf       T0IF      ;vynulování bitu T0IF
         clrf      tmr0      ;vynulování TMR0 a předděličky
Cekej_A  btfss     T0IF      ;testování přetečení TMR0
         goto      Cekej_A   ;nepřetekl-li TMR0,
                             testuje ;znovu
         bcf       T0IF      ;vynulování bitu T0IF
         return
```

```
;*******************************************************
Tabulka addwf    pcl,f         ;přičte W k čítači instrukcí
        retlw    B'11000000'   ;zobrazí 0
        retlw    B'11111001'   ;zobrazí 1
        retlw    B'10100100'   ;zobrazí 2
        retlw    B'10110000'   ;zobrazí 3
        retlw    B'10011001'   ;zobrazí 4
        retlw    B'10010010'   ;zobrazí 5
        retlw    B'10000010'   ;zobrazí 6
        retlw    B'11111000'   ;zobrazí 7
        retlw    B'10000000'   ;zobrazí 8
        retlw    B'10010000'   ;zobrazí 9
;*******************************************************
Main    bsf      RP0           ;stránka 1 paměti RAM
        movlw    B'11010111'
        movwf    optreg        ;konfigurace TMR0 a
                               ;předděličky
        bcf      RP0           ;stránka 0 paměti RAM
;*******************************************************
Main_A  bsf      RP0           ;stránka 1 paměti RAM
        movlw    0xFF
        movwf    portb         ;port B je vstupní
        movlw    0xFF          ;port A je vstupní
        movwf    porta
        bcf      RP0           ;stránka 0 paměti RAM
        movlw    0x01
        call     Tabulka       ;zakóduje 1 jako znak 1
        btfss    portb,0
        goto     Main_B        ;je-li tlačítko sepnuto,
                               tak ;zobraz 1
        movlw    0x02
        call     Tabulka       ;zakóduje 2 jako znak 2
        btfss    portb,1
        goto     Main_B        ;je-li tlačítko sepnuto,
                               tak ;zobraz 2
        movlw    0x03
        call     Tabulka       ;zakóduje 3 jako znak 3
        btfss    portb,2
        goto     Main_B        ;je-li tlačítko sepnuto,
                               tak ;zobraz 3
        movlw    0x04
        call     Tabulka       ;zakóduje 4 jako znak 4
        btfss    portb,3
        goto     Main_B        ;je-li tlačítko sepnuto,
                               tak ;zobraz 4
        movlw    0x05
        call     Tabulka       ;zakóduje 5 jako znak 5
        btfss    portb,4
        goto     Main_B        ;je-li tlačítko sepnuto,
                               tak ;zobraz 5
```

```
        movlw   0x06
        call    Tabulka         ;zakóduje 6 jako znak 6
        btfss   portb,5
        goto    Main_B          ;je-li tlačítko sepnuto,
                                tak ;zobraz 6
        movlw   0x07
        call    Tabulka         ;zakóduje 7 jako znak 7
        btfss   portb,6
        goto    Main_B          ;je-li tlačítko sepnuto,
                                tak ;zobraz 7
        movlw   0x08
        call    Tabulka         ;zakóduje 8 jako znak 8
        btfss   portb,7
        goto    Main_B          ;je-li tlačítko sepnuto,
                                tak ;zobraz 8
        goto    Main_A          ;není-li stisknuto nic, skočí
                                ;na začátek
Main_B  movwf   portb           ;zapíše znak na portb
        bsf     RP0             ;stránka 1 paměti RAM
        movlw   0x00
        movwf   portb           ;port B je výstupní
        movlw   B'11111110'     ;aktivuje 1. sedmisegmentovku
        movwf   porta
        bcf     RP0             ;stránka 0 paměti RAM
        bcf     porta,0         ;rozsvítí 1. sedmisegmentovku
        call    Cekej
        goto    Main_A          ;skočí na začátek
        end
```

4.5 Úloha 5: Čítač

Zadání úlohy
Pomocí LED displeje vytvořte čítač 0 - 9999, který každou 1s přičte ke svému obsahu 1.

Účel úlohy
Seznámit se s multiplexním řízením displeje.

Rozbor úlohy
Pomocí TMR0 vytvoříme časové intervaly o periodě 0,005s. 200 těchto intervalů použijeme k vytvoření 1s. Při každém z 5ms intervalů rozsvítíme jinou sedmisegmentovou jednotku. Abychom nepotřebovali složitý postup při přepočítávání načítaného údaje, použijeme nehospodárný způsob: budeme hodnotu načítat do 4 registrů, přičemž zajistíme přetečení registru mezi číslem 9 a 10, tak budou údaje na jednotlivých sedmisegmentových jednotkách přímo odpovídat příslušným registrům.

Trocha teorie
Abychom mohli použít multiplexní řízení displeje, musíme zajistit přepínání jednotlivých sedmisegmentových jednotek určitou frekvencí. Tuto frekvenci lze odhadnout ze schopnosti lidského oka odlišit od sebe jednotlivé, rychle se měnící počitky. Z kinematografie víme, že již 24 snímků za sekundu by mělo stačit. Doporučuji však frekvenci minimálně 50Hz. Aby každý

zobrazovač mohl „poblikávat" frekvencí 50Hz, musí být přepínací frekvence tolikrát větší, kolik máme sedmisegmentových jednotek. V našem případě (4 jednotky) je řídící frekvence minimálně 200Hz. Maximální frekvence také není libovolná. Nesmíme překročit povolený frekvenční rozsah použitých součástek (LED, tranzistory) a musíme myslet na to, že čím větší frekvence, tím větší úbytky budeme mít na řídícím prvku (tranzistoru) a tím méně bude LED displej svítit. Ale to je kritické až od určité frekvence.

V našem případě použijeme periodu řídících impulsů 5ms, což odpovídá frekvenci 200Hz.

Abychom ještě více podpořili optický klam, budeme sedmisegmentovky rozsvěcet v pořadí 1, 3, 2, 4.

Poznámka

Všimněte si, v jakém pořadí jsou v paměti uloženy registry disp1, disp2, disp3 a disp4. Tak lze vyřešit požadované pořadí spínání sedmisegmentovek.

Program

```
;*******************************************************
w        equ    0           ;parametr instrukcí
f        equ    1           ;parametr instrukcí
f0       equ    0x00
tmr0     equ    0x01
optreg   equ    0x01        ;použijeme přímé adresování
pcl      equ    0x02
status   equ    0x03        ;status je na adrese 0x03
fsr      equ    0x04
porta    equ    0x05
trisa    equ    0x05        ;použijeme přímé adresování
portb    equ    0x06
trisb    equ    0x06        ;použijeme přímé adresování
intcon   equ    0x0B
c_disp   equ    0x0C        ;čítač multiplexeru
disp1    equ    0x0D        ;pamět pro 1 sedmisegmentovku
disp3    equ    0x0E        ;pamět pro 2 sedmisegmentovku
disp2    equ    0x0F        ;pamět pro 3 sedmisegmentovku
disp4    equ    0x10        ;pamět pro 4 sedmisegmentovku
sec      equ    0x11        ;čítač 1 sekundy
;*******************************************************
#define  RP0    status,5
#define  T0IF   intcon,2     ;indikace přetečení TMR0
#define  C      status,0
;*******************************************************
         list       p = PIC16F84
         __config   0x3FF1         ;nastavení konfigurace
;*******************************************************
         org         0x0000
         goto       Main
;*******************************************************
```

```
Tab1    addwf  pcl,f         ;přičte W k čítači instrukcí
        retlw  B'11000000'   ; zobrazí 0
        retlw  B'11111001'   ; zobrazí 1
        retlw  B'10100100'   ; zobrazí 2
        retlw  B'10110000'   ; zobrazí 3
        retlw  B'10011001'   ; zobrazí 4
        retlw  B'10010010'   ; zobrazí 5
        retlw  B'10000010'   ; zobrazí 6
        retlw  B'11111000'   ; zobrazí 7
        retlw  B'10000000'   ; zobrazí 8
        retlw  B'10010000'   ; zobrazí 9
;*****************************************************
Tab2    addwf  pcl,f         ;přičte W k čítači instrukcí
        retlw  B'11111110'   ;anoda 1. sedmisegmentovky
        retlw  B'11111011'   ;anoda 3. sedmisegmentovky
        retlw  B'11111101'   ;anoda 2. sedmisegmentovky
        retlw  B'11110111'   ;anoda 4. sedmisegmentovky
;*****************************************************
Counter clrf   sec          ;vynuluje registr sec
        incf   disp1,f      ;přičte 1 k registru Disp1
        movlw  0x0A
        subwf  disp1,w      ;odečte od Disp1 desítku a uloží
                            ;do W
        btfss  C            ;je-li C=1, Disp1 přetekl přes 9
        goto   Count_A
        clrf   disp1
        incf   disp2,f      ;přičte 1 k registru Disp2
        movlw  0x0A
        subwf  disp2,w      ;odečte od Disp2 desítku a uloží
                            ;do W
        btfss  C            ;je-li C=1, Disp2 přetekl přes 9
        goto   Count_A
        clrf   disp2
        incf   disp3,f      ;přičte 1 k registru Disp3
        movlw  0x0A
        subwf  disp3,w      ;odečte od Disp3 desítku a uloží
                            ;do W
        btfss  C            ;je-li C=1, Disp3 přetekl přes 9
        goto   Count_A
        clrf   disp3
        incf   disp4,f      ;přičte 1 k registru Disp4
        movlw  0x0A
        subwf  disp4,w      ;odečte od Disp4 desítku a uloží
                            ;do W
        btfss  C            ;je-li C=1, Disp4 přetekl přes 9
        goto   Count_A
        clrf   disp4
Count_A return
```

```
;*******************************************************
Displej  incf    c_disp,f ;přičte 1 k čítači multiplexeru
         movlw   0x04
         subwf   c_disp,w
         btfsc   C        ;je-li C=1, c_disp přetekl přes 3
         clrf    c_disp   ;přetekl-li c_disp, vynulujeme ho
         movlw   0xFF
         movwf   porta    ;zhasneme displej
         movlw   disp1
         movwf   fsr      ;nepřímá adresa disp1
         movf    c_disp,w ;c_disp překopírujeme do W
         addwf   fsr,f    ;přičteme c_disp k nepřímé
                          adrese ;disp1
                          ;vybereme tak jeden
                          z registrů ;disp1-4
         movf    f0,w     ;vybraný registr disp do registru W
         call    Tab1     ;zakódujeme obsah registru na znak
         movwf   portb    ;pošleme znak na port B
         movf    c_disp,w
         call    Tab2     ;vybereme příslušnou
                          anodu ;displeje
         movwf   porta    ;sepneme příslušný tranzistor
         return
;*******************************************************
Main     bsf     RP0          ;stránka 1 paměti RAM
         movlw   B'11010011'
         movwf   optreg       ;konfigurace TMR0 a předděličky
         movlw   0x00
         movwf   portb        ;port B je výstupní
         movlw   0xF0
         movwf   porta        ;bity 0-3 portu A jsou výstupní
         bcf     RP0          ;stránka 0 paměti RAM
         movlw   0xFF
         movwf   porta        ;zhasne
                              všechny ;sedmisegmentovky
         clrf    disp1        ;vynuluje registr Disp1
         clrf    disp2        ;vynuluje registr Disp2
         clrf    disp3        ;vynuluje registr Disp3
         clrf    disp4        ;vynuluje registr Disp4
         clrf    c_disp       ;vynuluje čítač multiplexeru
;*******************************************************
Main_A   bcf     T0IF         ;vynulování bitu T0IF
         clrf    tmr0         ;vynulování TMR0 a předděličky
Main_B   btfss   T0IF         ;testování bitu T0IF
         goto    Main_B
         call    DispIej      ;podprogram
                              multiplexování ;displeje
         incf    sec,f        ;přičte 1 k registru
                              čítače ;sekund
         movlw   0xC7
         subwf   sec,w        ;odečte od sec 199 a uloží do W
         btfsc   C            ;je li C=1, registr
                              sec ;nepřetekl
```

```
call    Counter     ;jestliže sec přetekl,
                    zavolá ;Counter
goto    Main_A      ;skočí na začátek smyčky
end
```

4.1 Úloha 6: Hrací automat

Zadání úlohy
Naprogramujte hrací automat. Na sedmisegmentových jednotkách budou rotovat čísla od 0 do 9. Po prvním stisku tlačítka se začnou čísla na první sedmisegmentovce zpomalovat až se zastaví úplně. Při dalším stisku se začnou zastavovat čísla na druhé sedmisegmentovce atd. Po zastavení poslední sedmisegmentovky bude proveden test. Pokud se číslice na displeji nebudou shodovat, počká automat na další stisk tlačítka a zahájí novou hru. Pokud se číslice shodovat budou, rozsvítí se LED a displej začne blikat. Automat opět vyčká na stisk tlačítka a pak zahájí novou hru.

Účel úlohy
Seznámit se s použitím přerušení.

Rozbor úlohy
Pomocí TMR0 vytvoříme časové intervaly o periodě 0,005s. Každé přetečení TMR0 vyvolá přerušení. Přerušovací podprogram bude zajišťovat multiplexní řízení displeje a rozsvěcení/zhasínání LED. Hlavní část programu bude rotovat čísla od 0 do 9 v registru určeném nepřímou adresou. Rotovaný registr bude dále obsahovat informaci, zda má být rotován nebo zastaven, rotován rychle nebo postupně zastaven. Zbytek programu bude zajišťovat postupné přepínání nepřímých adres všech registrů příslušných sedmisegmentovek, pokud budou všechny sedmisegmentovky zastaveny (budou nastaveny příslušné stavové bity), provede testování. Frekvence rotování čísel na sedmisegmentovkách bude odvozena od vnořené časové smyčky s možností nastavení požadovaného časového zpoždění pomocí parametru v registru W.

Trocha teorie
Systém přerušení je ovládán pomocí registru INTCON, který byl použit v příkladu 3, proto ho již nebudu znova popisovat. Pokud je vyvoláno přerušení, ať od vnějšího vstupu RB0/INT, změny na vstupech RB4-7, nebo od přetečení TMR0, či od ukončení zápisu do EEPROM, mikrokontrolér provede skok na adresu 0x0004 paměti programu a do stacku uloží návratovou adresu. Po vyvolání přerušení je automaticky vynulován bit GIE. 16F84 nerozlišuje jaká periferie způsobila vyvolání přerušení. Je pouze na programátorovi, aby pomocí stavových bitů v registru INTCON rozpoznal o jaký typ přerušení se jedná a aby provedl buď skok na blok programu s obsluhou příslušného přerušení, zakončeného instrukcí retfie, nebo zavolání obsužného podprogramu a po návratu z něj vykonání instrukce retfie. Je také na programátorovi, aby zajistil vynulování příslušného stavového bitu periferie, která vyvolala přerušení. Po vykonání instrukce retfie, je totiž zpět nastaven bit GIE. Nevynulovaný flag by pak způsobil

další falešné vyvolání přerušení. Pokud potřebujeme používat přerušení v již spuštěném přerušení, musíme po vynulování příslušného flagu nastavit bit GIE softwarově. Neboť mikrokontrolér nerozlišuje jaké přerušení bylo vyvoláno, nejsou nikde pevně stanoveny priority přerušení. Různých priorit lze dosáhnout tím, že stavový bit přerušení s nejvyšší prioritou testujeme jako první, s nižší jako druhý atd. Přerušení od jednotlivých periferií lze povolit nebo zakázat příslušnými bity registru INTCON (EEIE, T0IE, INTE, RBIE). Tyto bity nejsou po vyvolání přerušení automaticky vynulovány.

Příklady řešení obsluhy přerušení:

```
;*********************************************************
        org     0x0004      ;adresa vektoru přerušení
        btfsc   EEIE        ;ukončení zápisu do EEPROM
        goto    EEWrite
        btfsc   TOIE        ;přetečení TMR0
        goto    Counter
        btfsc   INTE        ;vnější přerušení
        goto    Int
        btfsc   RBIE        ;změna na vstupech RB4 - RB7
        goto    Tlac
EEWrite bcf     EEIF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        retfie
Counter bcf     T0IF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        retfie
Int     bcf     INTF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        retfie
Tlac    bcf     RBIF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        retfie

;*********************************************************
        org     0x0004      ;adresa vektoru přerušení
        btfsc   EEIE        ;ukončení zápisu do EEPROM
        call    EEWrite
        btfsc   TOIE        ;přetečení TMR0
        call    Counter
        btfsc   INTE        ;vnější přerušení
        call    Int
        btfsc   RBIE        ;změna na vstupech RB4 - RB7
        call    Tlac
        retfie
EEWrite bcf     EEIF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        return
Counter bcf     T0IF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        return
```

```
Int     bcf     INTF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        return
Tlac    bcf     RBIF        ;vynulování flagu
        nop                 ;vlastní obsluha přerušení
        return
```

Při používání přerušení si však musíme dát dobrý pozor. Přerušení totiž může nastat všude tam, kde je nastaven bit GIE a alespoň jeden bit povolující přerušení nějaké periferie. To znamená, že přerušovací procedura musí striktně uložit všechny registry, které může sama měnit, včetně registru W. Před návratem do hlavního programu musí obnovit původní stav registrů. Musíme také počítat s tím, že procedura přerušení použije minimálně jednu úroveň zásobníku pro uložení návratové adresy. V programu pak nesmíme na žádném místě, kde je povoleno přerušení, použít všechny úrovně zásobníku, došlo by k jeho přetečení a následně k chybné funkci programu.
Pokud je vynulován bit GIE nebo některý z bitů povolujících přerušení od periferií a nastane událost, která by jinak vedla k vyvolání přerušení, je nastaven pouze příslušný stavový bit.

Poznámka

Všimněte si, že sedmisegmentovky jsou opět spínány v pořadí 1, 3, 2, 4. Je k tomu však použita jiná metoda, neboť se jeví výhodnější, aby registry disp1 až disp4 byly v paměti dat uloženy vzestupně. Při multiplexování je tedy použito překódování z pořadí 1, 2, 3, 4 na 1, 3, 2, 4 pomocí vhodné tabulky.

```
;*****************************************************
w         equ     0
f         equ     1
indf      equ     0x00      ;registr nepřím. adresování
optreg    equ     0x01      ;registr option
pcl       equ     0x02      ;čítač programu
status    equ     0x03
fsr       equ     0x04      ;nepřímá adresa
porta     equ     0x05
trisa     equ     0x05
portb     equ     0x06
trisb     equ     0x06
intcon    equ     0x0B      ;registr řízení přerušení
tlreg     equ     0x0C      ;záchytný registr tlačítek
cnttl     equ     0x0D      ;filtr zákmitů tlačítek
cntd      equ     0x0E      ;čítač multiplexeru
disp1     equ     0x0F      ;pamět pro 1 sedmisegmentovku
disp2     equ     0x10      ;pamět pro 2 sedmisegmentovku
disp3     equ     0x11      ;pamět pro 3 sedmisegmentovku
disp4     equ     0x12      ;pamět pro 4 sedmisegmentovku
cntd1     equ     0x13      ;čítač 1 sedmisegmentovky
cntd3     equ     0x14      ;čítač 2 sedmisegmentovky
```

```
cntd2     equ     0x15       ;čítač 3 sedmisegmentovky
cntd4     equ     0x16       ;čítač 4 sedmisegmentovky
s_w       equ     0x17       ;stack registru w
s_stat    equ     0x18       ;stack registru status
s_fsr     equ     0x19       ;stack registru fsr
cnt1      equ     0x1A       ;čítač časové smyčky
cnt2      equ     0x1B       ;čítač časové smyčky
cnt3      equ     0x1C       ;čítač časové smyčky
cntrot    equ     0x1D       ;čítač rotací
rel1      equ     0x1E       ;pointer na adresu
rel2      equ     0x1F       ;pointer na adresu
stack     equ     0x20       ;zásobník
cntloop   equ     0x21       ;čítač průchodů smyčkou
pointer   qu      0x22       ;pointer na adresu
cntbl     equ     0x23       ;čítač blikání displeje
led       equ     0x24       ;registr stavu led
s_trisa   equ     0x25       ;stack registru trisa
s_trisb   equ     0x26       ;stack registru trisb
;*****************************************************
#define   P0      status,5   ;stránka paměti dat
#define           status,0
#define           status,2
#define   IE      intcon,7   ;globální povolení přerušení
#define   0IE     intcon,5   ;povolení přerušení timeru0
#define   0IF     intcon,2   ;flag přetečeni timeru0
;*****************************************************
          ist        p = 16f84
          _config    0x3FF1      ;nastavení konfigurace
;*****************************************************
          org     0x0000
          goto    Init
;*****************************************************
        org     0x0004    ;adresa vektoru přerušení
Zobraz  bcf     T0IF      ;vynulování flagu timeru0
        movwf   s_w       ;uložení registru w
        movf    status,w
        movwf   s_stat    ;uložení registru status
        movf    fsr,w
        movwf   s_fsr     ;uložení registru fsr
        bsf     RP0       ;vyšší stránka paměti dat
        movf    trisa,w
        movwf   s_trisa   ;uložení registru trisa
        movf    trisb,w
        movwf   s_trisb   ;uložení registru trisb
        bcf     RP0       ;nastaveni nižší str.paměti
        incf    cntd,f    ;přičte 1 k čit. multiplexeru
        movlw   0x04
        subwf   cntd,w
        btfsc   C         ;je-li C=1,c_disp přetekl přes 3
        clrf    cntd      ;přetekl-li c_disp, vynulujeme ;ho
```

```
        movlw   0xFF
        movwf   porta       ;zhasneme displej
        movlw   disp1
        movwf   fsr         ;nepřímá adresa disp1
        movf    cntd,w      ;c_disp překopírujeme do W
        call    Tab4        ;překódujeme c_disp -
                            ;střídavý multiplex
        addwf   fsr,f       ;přičte W k nepřímé adrese disp1
                            ;vybere tak jeden z registrů
                            disp
        movf    indf,w      ;vybraný registr disp do
                            registru W
        call    Tab1        ;zakódujeme obsah registru na
                            znak
        movwf   portb       ;pošleme znak na port B
        movf    cntd,w
        call    Tab2        ;vybereme příslušnou anodu
        movf    led,f
        btfss   Z           ;testujeme registr led na 0
        addlw   0x10        ;je-li 0, LED zhasne (RA4=1)
        movwf   porta       ;sepneme příslušný tranzistor
        bsf     RP0         ;vyšší stránka paměti dat
        movf    s_trisa,w
        movwf   trisa       ;obnovení registru trisa
        movf    s_trisb,w
        movwf   trisb       ;obnovení registru trisb
        movf    s_stat,w
        movwf   status      ;obnovení registru status
        movf    s_fsr,w
        movwf   fsr         ;obnovení registru fsr
        movf    s_w,w       ;obnovení registru W
        retfie              ;návrat z přerušení
;*****************************************************
Tab1    addwf   pcl,f           ;přičte W k čítači instrukci
        retlw   B'11000000'     ;zobrazí 0
        retlw   B'11111001'     ;zobrazí 1
        retlw   B'10100100'     ;zobrazí 2
        retlw   B'10110000'     ;zobrazí 3
        retlw   B'10011001'     ;zobrazí 4
        retlw   B'10010010'     ;zobrazí 5
        retlw   B'10000010'     ;zobrazí 6
        retlw   B'11111000'     ;zobrazí 7
        retlw   B'10000000'     ;zobrazí 8
        retlw   B'10010000'     ;zobrazí 9
        retlw   B'11111111'     ;nezobrazí nic
;*****************************************************
Tab2    addwf   pcl,f          ;přičte W k čítači instrukcí
        retlw   B'11111110'    ;anoda 1. sedmisegmentovky
        retlw   B'11111011'    ;anoda 3. sedmisegmentovky
        retlw   B'11111101'    ;anoda 2. sedmisegmentovky
        retlw   B'11110111'    ;anoda 4. sedmisegmentovky
```

```
;*******************************************************
Tab3    addwf   pcl,f   ;tabulka koeficientů postupného
                        ;zpomalování rotace
                        ;sedmisegmentovek
        retlw   0x03
        retlw   0x05
        retlw   0x07
        retlw   0x09
        retlw   0x0C
        retlw   0x0F
        retlw   0x12
;*******************************************************
Tab4    addwf   pcl,f   ;tabulka překódování
                        ;pořadí registru
                        ;disp1-4 v paměti dat
        retlw   0x00
        retlw   0x02
        retlw   0x01
        retlw   0x03
;*******************************************************
Cekej   movwf   cnt1     ;naplnění registru 3.smyčky
Cekej_A movlw   0x3c
        movwf   cnt2     ;naplnění registru 2.smyčky
Cekej_B movlw   0x5B
        movwf   cnt3     ;naplnění registru 1.smyčky
        decfsz  cnt3,f   ;odečte od registru 1.smyčky 1
        goto    $-1      ;skočí, je-li registr roven 0
        decfsz  cnt2,f   ;odečte od registru 2.smyčky 1
        goto    Cekej_B  ;skočí, je-li registr roven 0
        decfsz  cnt1,f   ;odečte od registru 3.smyčky 1
        goto    Cekej_A  ;skočí, je-li registr roven 0
        return           ;návrat do hlavního programu
;*******************************************************
Cti_tl  bsf     RP0      ;vyšší stránka paměti dat
        movlw   0xFF
        movwf   trisb    ;port A vstupní
        movwf   trisa    ;port B vstupní
        bcf     RP0      ;nižší stránka paměti dat
        movlw   0x0A
        movwf   cnttl    ;nastavení filtru zákmitů
                         ;tlačítka
        movf    portb,w  ;přečtení stavu portu B
        andlw   0x01     ;zamaskování nepotřebných bitů
Cti_A   movwf   tlreg    ;uložení stavu tlačítka
        movf    portb,w
        andlw   0x01
        subwf   tlreg,f  ;odečtení nového stavu tlačítek
        btfss   Z        ;test shodnosti
        goto    Cti_tl   ;jsou-li rozdílné skočí na
                         ;začátek
        decfsz  cnttl,f
```

```
        goto    Cti_A   ;opakuje cnttl krát
        movwf   tlreg   ;uloží stav tlačítka
        bsf     RP0     ;vyšší stránka paměti dat
        movlw   0x00
        movwf   trisa   ;port A výstupní
        movwf   trisb   ;port A výstupní
        bcf     RP0     ;nižší stránka paměti dat
        return          ;návrat z podprogramu
;*****************************************************
Init    bsf     RP0         ;vyšší stránka paměti dat
        movlw   0x00
        movwf   trisb       ;port B výstupní
        movwf   trisa       ;port A výstupní
        movlw   B'11010011'
        movwf   optreg      ;konfigurace RTCC a
                            předděličky
        bcf     RP0         ;nižší stránka paměti dat
        clrf    led         ;vynulování stavu LED
        bcf     T0IF        ;vynulování flagu přerušení
                            ;timeru0
        bsf     T0IE        ;povolení přerušení od
                            timeru0
        bsf     GIE         ;globální povoleni přerušení
;*****************************************************
Main    movlw   0x00
        movwf   cntd1   ;přednastavení
                        ;1. sedmisegmentovky
        movlw   0x01
        movwf   cntd2   ;přednastavení
                        ;2. sedmisegmentovky
        movlw   0x02
        movwf   cntd3   ;přednastavení
                        ;3. sedmisegmentovky
        movlw   0x03
        movwf   cntd4   ;přednastavení
                        ;4. sedmisegmentovky
        clrf    cntrot  ;vynulování čítače rotací
        clrf    cntloop ;vynulování čítače průchodů
                        ;smyčkou
        movlw   cntd4   ;adresa cntd4
        movwf   pointer ;uložení adresy do ukazatele
Main_A  movlw   cntd1   ;adresa cntd1
        movwf   rel1    ;uložení adresy do ukazatele
        movlw   disp1   ;adresa cntd2
        movwf   rel2    ;uložení adresy do ukazatele
        movf     pointer,w
        movwf   fsr     ;nepřímá adresace aktuálního
                        ;cntd
        btfss   indf,6  ;test flagu uloženého v cntd
        goto    Main_I  ;je-li v pohybu, skočí na ;Main_I
        decf    pointer,f   ;jinak přeadresuje pointer
```

```
                        ;na další cntd
        movf    pointer,w
        sublw   (cntd1 - 1)
        btfss   Z               ;není-li adresa v pointeru
                                ;cntd1 menší než
        goto    Main_J          ;tak skočí na Main_J
        movf    disp1,w         jinak testuje registry
                                ;disp1-disp4
        subwf   disp2,w
        btfss   Z
        goto    Main_G          ;disp1 <> disp2 -> skočí na
                                ;Main_G
        movf    disp3,w
        subwf   disp4,w
        btfss   Z
        goto    Main_G          ;disp3 <> disp4 -> skočí na
                                ;Main_G
        movf    disp1,w
        subwf   disp3,w
        btfss   Z
        goto    Main_G          ;disp1 <> disp3 -> skočí na
                                ;Main_G
                                ;jsou li všechny cntd shodné
                                -;>
                                ;výhra!
        movlw   0x01            ;nastaví led tak, aby se po
                                ;proběhnutí
        movwf   led             ;přerušení rozsvítila LED
        movf    disp1,w
        movwf   stack           ;uložení vítězného čísla
Main_B  movlw   0x0C
        movwf   cntbl           ;perioda blikání displeje
        movlw   0x0A
        movwf   disp1           ;zhasnutí 1.sedmisegmentovky
        movwf   disp2           ;zhasnutí 2.sedmisegmentovky
        movwf   disp3           ;zhasnutí 3.sedmisegmentovky
        movwf   disp4           ;zhasnutí 4.sedmisegmentovky
Main_C  call    Cti_tl          ;testování tlačítka
        btfss   tlreg,0
        goto    Main_E          ;je-li stisknuté, skočí na
                                ;Main_E
        movlw   0x01
        call    Cekej           ;chvilku čeká
        decfsz  cntbl,f         ;dekrementuje a testuje cntbl
        goto    Main_C          ;je-li cntbl=0 -> nikam
                                neskáče
        movf    stack,w         ;obnovení stavu displeje
        movwf   disp1

       movwf   disp2
       movwf   disp3
```

```
        movwf   disp4
        movlw   0x0C        ;perioda blikání displeje
        movwf   cntbl
Main_D  call    Cti_tl      ;testování tlačítka
        btfss   tlreg,0
        goto    Main_E      ;je-li stisknuté, skočí na
                            ;Main_E
        movlw   0x01
        call    Cekej       ;chvilku čeká
        decfsz  cntbl,f     ;dekrementuje a testuje cntbl
        goto    Main_D      ;je-li cntbl=0 -> nikam neskáče
        goto    Main_B      ;návrat na Main_B
Main_E  clrf    led         ;nastaví led tak, aby se po
                            ;proběhnutí přerušení
                            ;rozsvítila LED
        movlw   0x0A        ;zhasnutí sedmisegmentovek
        movwf   disp1
        movwf   disp2
        movwf   disp3
        movwf   disp4
Main_F  call    Cti_tl      ;testování tlačítka
        btfsc   tlreg,0
        goto    Main        ;je-li puštěné,skočí na začátek
                            ;programu
        movlw   0x01
        call    Cekej       ;chvilku čeká
        goto    Main_F      ;návrat na začátek testovací
                            ;smyčky
Main_G  call    Cti_tl      ;testování tlačítka
        btfss   tlreg,0
        goto    Main_H      ;je-li stisknuté, skočí na
                            ;Main_H
        movlw   0x01
        call    Cekej       ;chvilku čeká
        goto    Main_G      ;návrat na začátek smyčky
Main_H  call    Cti_tl      ;testování tlačítka
        btfsc   tlreg,0
        goto    Main        ;je-li puštěné,skočí na začátek
                            ;programu
        movlw   0x01
        call    Cekej       ;chvilku čeká
        goto    Main_H      ;návrat na začátek testovací
                            ;smyčky
Main_I  btfsc   indf,7      ;test flagu uloženého v cntd
        goto    Main_J      ;je-li nastaven na rychlý
                            ; režim,neskáče
        call    Cti_tl      ;testování tlačítka
        btfsc   tlreg,0
        goto    Main_J      ;je-li stisknuté, nikam neskáče
        clrf    cntrot      ;vynulování čítače rotací
        clrf    cntloop     ;vynulování čítače průchodů
```

```
                                ;smyčkou
        bsf     indf,7          ;nastavení flagu pomalého běhu
Main_J  clrw                    ;vynuluje W
        call    Tab3            ;vrátí 1. řádek tabulky
        call    Cekej           ;chvilku čeká
Main_K  movf    rel1,w
        movwf   fsr             ;postupně nepřímo adresuje
                                ;cntd1-4
        btfsc   indf,6          ;podle stavu flagu skočí na
                                ;Main_O
        goto    Main_O          ;skok na Main_O -> bez rotace
        btfss   indf,7          ;podle stavu flagu skočí na
                                ;Main_M
        goto    Main_M          ;skok na Main_M->rychlá rotace
        movf    cntrot,w        ;podle obsahu registru cntrot
                                ;vybere konstantu
        call    Tab3            ;zpomalení rotace z Tab3
        subwf   cntloop,w       ;je konstanta shodná s obsahem
                                ;čítače
        btfsc   Z               ;průchodu smyčkou???
        goto    Main_L          ;jestli ano, skočí na Main_L
        incf    cntloop,f       ;jesli ne,inkrementuje čítač
                                ;průchodů
        goto    Main_O          ;smyčkou a skočí na Main_O
Main_L  clrf    cntloop         ;vynuluje čítač průchodů
                                ;smyčkou
        incf    cntrot,f        ;inkrementuje čítač rotací
        movf    cntrot,w
        sublw   0x07            ;testuje obsah čítače rotací
        btfss   Z
        goto    Main_M          ;nepřetekl-li 6, skočí na
                                ;Main_M
        clrf    cntrot          ;jinak jej vymaže
        bsf     indf,6          ;nastaví flag v aktuálním cntd
                                ; -> zastavení rotace
Main_M  incf    indf,f          ;inkrementuje aktuální cntd
        movf    indf,w
        andlw   B'00111111'     ;zamaskuje aktuální cntd, aby
                                ;se neporušily flagy
        sublw   0x0A
        btfss   Z               ;jestliže po inkrementování
                                ;nepřetekl
        goto    Main_N          ;cntd pres 9 skočí na Main_N
        movf    indf,w          ;jinak si zapamatuje flagy
        andlw   B'11000000'     ;a registr cntd vynuluje
        movwf   indf            ;flagy uloží nazpátek do cntd;
Main_N  movf    indf,w
        andlw   B'00111111'     ;zamaskuje aktuální cntd
        movwf   stack           ;a uloží jej do stacku
        movf    rel2,w          ;nastaví do fsr adresu
        movwf   fsr             ;aktuálního disp (1-4)
```

```
        movf    stack,w
        movwf   indf            ;přesune do aktuálního disp
                                ;obsah stacku
Main_O  incf    rel1,f          ;nastaví ukazatel na
                                ;následující cntd
        incf    rel2,f          ;nastaví ukazatel na
                                ;následující disp
        movf    rel1,w
        sublw   (cntd1 +        ;testuje, zda ukazatel
                0x04)
                                ;nepřetekl
        btfsc   Z               ;adresový prostor
                                ;vyhrazený pro cntd1-4
        goto    Main_A          ;přetekl-li -> skočí na Main_A
        goto    Main_K          ;jinak skočí na Main_K
;*************************************************
        end
```

4.8 Úloha 7: Kódový zámek

Zadání úlohy

Naprogramujte kódový zámek. Prvních šest tlačítek bude reprezentovat znaky 1 – 6, které budou přípustné v kódu. Sedmé tlačítko bude ESC a osmé ENTER. Po zapnutí budou na displeji svítit tečky. Po zadání znaku se na příslušné sedmisegmentovce rozsvítí - . Kdykoliv je možno přerušit zadávání stiskem ESC. Po zadání čtvrtého znaku bude program čekat buď na ESC, který zadávání zruší, nebo na ENTER, který potvrdí zadané heslo. Program provede kontrolu s heslem uloženým v EEPROM. Pokud se hesla budou shodovat, displej zhasne a na 3s se rozsvítí LED (aktivace zámku). Pokud budou hesla různá, vypíše se na displej: Err. a program opět počká 3s. Potom lze znova zadat heslo. Režim zadání nového hesla se spustí, pokud bude při zapnutí nepo stisku tlačítka RESET stisknuto tlačítko ENTER. Nejprve bude třeba zadat staré heslo stejným způsobem jako v normálním režimu. Pak se zadá nové heslo, které se bude zároveň pro kontrolu opisovat na displej. Během zadávání je opět možno heslo zrušit stiskem ESC a potvrdit stiskem ENTER. Před prvním spuštěním bude nastaveno heslo 1234.

Účel úlohy

Seznámit se s použitím integrované paměti EEPROM.

Rozbor úlohy

Pomocí TMR0 vytvoříme časové intervaly o periodě 0,005s. Každé přetečení TMR0 vyvolá přerušení. Přerušovací podprogram bude zajišťovat multiplexní řízení displeje a rozsvěcení/zhasínání LED. Hlavní část programu bude zajišťovat ukládání znaků do paměti RAM, zapisování do paměti EEPROM a porovnávání hesel v RAM a EEPROM. Dále bude v programu funkce, která čte klávesy, a funkce, která vyhodnocuje jejich stav.

Trocha teorie

Paměť EEPROM : S integrovanou pamětí EEPROM komunikujeme pomocí registrů eeadr – adresa bytu, z/do kterého chceme číst/zapisovat, eedata – data, která jsou vyčtena nebo která chceme zapisovat, eecon1 – řídící signály, eecon2 – virtuální registr, pomocí kterého definujeme zapisovací sekvenci.

EEIF – příznak přerušení od ukončení zápisu do EEPROM:
1 - zápis ukončen **0 -** zápis neukončen
WRERR – příznak RESETu mikrořadiče během zápisu do EEPROM:
1 - RESETem přerušen zápis **0 -** zápis ukončen

-	-	-	EEIF	WRERR	WREN	WR	RD

WREN – povolení zápisu do EEPROM:
1 - zápis povolen **0 -** zápis zakázán
WR - zahajuje zápis do EEPROM:
1 - čtení aktivováno **1 -** čtení neaktivní
RD - zahajuje čtení z EEPROM:
1 - zápis aktivován **0** – zápis neaktivní

Po ukončení čtení/zápisu je bit RD/WR automaticky vynulován. Po zahájení čtení nastavením bitu RD jsou platná data k dispozici již v následujícím instrukčním cyklu.

```
;*******************************************************
w       equ  0
f       equ  1
indf    equ  0x00     ;registr neprimeho adresovani
optreg  equ  0x01     ;registr option
pcl     equ  0x02     ;citac programu
status  equ  0x03
fsr     equ  0x04     ;neprima adresa
porta   equ  0x05
trisa   equ  0x05
portb   equ  0x06
trisb   equ  0x06
eedata  equ  0x08     ;data EEPROM
eecon1  equ  0x08     ;rizeni EEPROM
eeadr   equ  0x09     ;adresa EEPROM
eecon2  equ  0x09     ;rizeni EEPROM
intcon  equ  0x0B     ;registr rizeni preruseni
tlreg   equ  0x0C     ;zachytny registr tlacitek
cnttl   equ  0x0D     ;citac filtru zakmitu tlacitek
cntd    equ  0x0E     ;citac multiplexeru
disp1   equ  0x0F     ;pamet pro 1 sedmisegmentovku
disp2   equ  0x10     ;pamet pro 2 sedmisegmentovku
disp3   equ  0x11     ;pamet pro 3 sedmisegmentovku
disp4   equ  0x12     ;pamet pro 4 sedmisegmentovku
s_w     equ  0x13     ;stack registru w
s_stat       equ    0x14      ;stack registru status
s_fsr        equ    0x15      ;stack registru fsr
s_trisa equ         0x16      ;stack registru trisa
```

```
s_trisb equ         0x17      ;stack registru trisb
cnt1          equ   0x18      ;citac casove smycky
cnt2          equ   0x19      ;citac casove smycky
cnt3          equ   0x1A      ;citac casove smycky
cnts          equ   0x1B      ;citac stavu klaves
latch         equ   0x1C      ;zachytny registr
stav          equ   0x1D      ;stav klaves
kod4          equ   0x1E      ;pamet pro 4. znak kodu
kod3          equ   0x1F      ;pamet pro 3. znak kodu
kod2          equ   0x20      ;pamet pro 2. znak kodu
kod1          equ   0x21      ;pamet pro 1. znak kodu
cntk          equ   0x22      ;citac poctu zadanych znaku
flags         equ   0x23      ;stav klaves ENTER a ESC
cntt          equ   0x24      ;citac testovani
led           equ   0x25      ;stav LED
cntw          equ   0x26      ;citac zapisu do EEPROM
;*******************************************************
#define RP0   status,5   ;stranka pameti dat
#define C     status,0
#define Z     status,2
#define GIE   intcon,7   ;globalni povoleni preruseni
#define T0IE  intcon,5   ;povoleni preruseni timeru0
#define T0IF  intcon,2   ;flag preteceni timeru0
#define EEIF  eecon1,4   ;flag ukonceni zapisudo EEPROM
#define WREN  eecon1,2   ;povoleni zapisu do EEPROM
#define WR    eecon1,1   ;zahajeni zapisu do EEPROM
#define RD    eecon1,0   ;zahajeni cteni z EEPROM
#define ENTERflags,7     ;klavesa ENTER
#define ESC   flags,6    ;klavesa ESC
;*******************************************************
              list       p = 16f84
              __config   0x3FF1     ;nastaveni konfigurace
;*******************************************************
              org        0x0000
              goto       Init
;*******************************************************
       org    0x0004     ;adresa vektoru preruseni
Zobraz bcf    T0IF       ;vynulovani flagu timeru0
       movwf  s_w        ;ulozeni registru w
       movf   status,w
       movwf  s_stat     ;ulozeni registru status
       movf   fsr,w
       movwf  s_fsr      ;ulozeni registru fsr
       bsf    RP0        ;vyssi stranka pameti dat
       movf   trisa,w
       movwf  s_trisa    ;ulozeni registru trisa
       movf   trisb,w
       movwf  s_trisb    ;ulozeni registru trisb
       bcf    RP0        ;nastaveni nizsi stranky pameti dat
       incf   cntd,f     ;pricte 1 k citaci multiplexeru
       movlw  0x04
```

```
        subwf  cntd,w
        btfsc  C          ;je-li C=1, c_disp pretekl pres 3
        clrf   cntd       ;pretekl-li c_disp, vynulujeme ho
        movlw  0xFF
        movwf  porta      ;zhasneme displej
        movlw  disp1
        movwf  fsr        ;neprima adresa disp1
        movf   cntd,w     ;c_disp prekopirujeme do W
        call   Tab3       ;prekodujeme W
        addwf  fsr,f      ;pricteme W k neprime adrese disp1
                          ;vybereme tak jeden z
                          registrů ;disp1-4
        movf   indf,w     ;vybrany registr disp do registru W
        call   Tab1       ;zakodujeme obsah registru na znak
        movwf  portb      ;posleme znak na port B
        movf   cntd,w
        call   Tab2       ;vybereme prislusnou anodu displeje
        movf   led,f
        btfss  Z          ;testujeme registr led na 0
        addlw  0x10       ;je-li 0, LED nebude svitit (RA4=1)
        movwf  porta      ;sepneme prislusny tranzistor
        bsf    RP0        ;vyssi stranka pameti dat
        movf   s_trisa,w
        movwf  trisa      ;obnoveni registru trisa
        movf   s_trisb,w
        movwf  trisb      ;obnoveni registru trisb
        movf   s_stat,w
        movwf  status     ;obnoveni registru status
        movf   s_fsr,w
        movwf  fsr        ;obnoveni registru fsr
        movf   s_w,w      ;obnoveni registru W
        retfie            ;navrat z preruseni
;*****************************************************
Tab1    addwf   pcl,f       ;pricte W k citaci instrukci
        retlw   B'11000000' ;zobrazi 0
        retlw   B'11111001' ;zobrazi 1
        retlw   B'10100100' ;zobrazi 2
        retlw   B'10110000' ;zobrazi 3
        retlw   B'10011001' ;zobrazi 4
        retlw   B'10010010' ;zobrazi 5
        retlw   B'10000010' ;zobrazi 6
        retlw   B'11111000' ;zobrazi 7
        retlw   B'10000000' ;zobrazi 8
        retlw   B'10010000' ;zobrazi 9
        retlw   B'11111111' ;nezobrazi nic
        retlw   B'10111111' ;zobrazi -
        retlw   B'10000110' ;zobrazi E
        retlw   B'10101111' ;zobrazi r
        retlw   B'00101111' ;zobrazi r.
        retlw   B'01111111' ;zobrazi .
;*****************************************************
```

```
Tab2   addwf   pcl,f       ;pricte W k citaci instrukci
       retlw   B'11111110' ;anoda 1. sedmisegmentovky
       retlw   B'11111011' ;anoda 3. sedmisegmentovky
       retlw   B'11111101' ;anoda 2. sedmisegmentovky
       retlw   B'11110111' ;anoda 4. sedmisegmentovky
;*********************************************************
Tab3   addwf   pcl,f       ;prekoduje 0 1 2 3 na 0 2 1 3
       retlw   0x00
       retlw   0x02
       retlw   0x01
       retlw   0x03
;*********************************************************
Cekej   movwf   cnt1    ;naplneni registru 3.smycky
Cekej_A movlw   0x3c
        movwf   cnt2    ;naplneni registru 2.smycky
Cekej_B movlw   0x5B
        movwf   cnt3    ;naplneni registru 1.smycky
        decfsz  cnt3,f  ;odecte od registru 1.smycky 1
        goto    $-1     ;skoci, je-li registr roven 0
        decfsz  cnt2,f  ;odecte od registru 2.smycky 1
        goto    Cekej_B ;skoci, je-li registr roven 0
        decfsz  cnt1,f  ;odecte od registru 3.smycky 1
        goto    Cekej_A ;skoci, je-li registr roven 0
        return          ;navrat do hlavniho programu
;*********************************************************
Cti_tl  bsf     RP0     ;vyssi stranka pameti dat
        movlw   0xFF
        movwf   trisb   ;port A vstupni
        movwf   trisa   ;port B vstupni
        bcf     RP0     ;nizsi stranka pameti dat
        movlw   0x0A
        movwf   cnttl   ;nastaveni filtru zakmitu tlacitka
        movf    portb,w ;precteni stavu portu B
Cti_A   movwf   tlreg   ;ulozeni stavu tlacitka
        movf    portb,w
        subwf   tlreg,f ;odecteni noveho stavu tlacitek
        btfss   Z       ;test shodnosti
        goto    Cti_tl  ;jsou-li rozdilne skoci na zacatek
        decfsz  cnttl,f
        goto    Cti_A   ;opakuje cnttl krat
        movwf   tlreg   ;ulozi stav tlacitka
        bsf     RP0     ;vyssi stranka pameti dat
        movlw   0x00
        movwf   trisa   ;port A vystupni
        movwf   trisb   ;port B vystupni
        bcf     RP0     ;nizsi stranka pameti dat
        return          ;navrat z podprogramu
;*********************************************************
Stav_tl clrf    flags   ;zrusi stav klaves ENTER a ESC
        movf    tlreg,w
        movwf   latch   ;stav klaves do latch
```

```
        sublw   0xFF
        btfss   Z       ;neni-li stisknuta zadna klavesa
        goto    Stav_A  ;nikam neskace
        clrf    stav    ;smaze stav
        return          ;a ukonci podprogram
Stav_A  btfss   latch,6 ;otestuje klavesu ESC
        goto    Stav_D  ;je-li stisknuta tak skoci
        btfss   latch,7 ;otestuje klavesu ENTER
        goto    Stav_E  ;je-li stisknuta tak skoci
        clrf    cnts    ;jinak smaze stav
Stav_B  incf    cnts,f  ;inkrementuje stav
        movlw   0x07
        subwf   cnts,w  ;testuje, zda je stav roven 7 (tj
                        > ;6)
        btfsc   Z       ;pokud ano, ukonci podprogram
        return
        rrf     latch,f ;jinak rotuje zachytny registr
        btfsc   C       ;testuje klavesu
        goto    Stav_B  ;neni-li stisknuta, tak skoci zpet
        movf    cnts,w
        movwf   stav    ;je-li stisknuta, ulozi jeji index
Stav_C  movlw   0x0A
        call    Cekej   ;chvilku ceka
        call    Cti_tl  ;precte klavesy
        movf    tlreg,w
        sublw   0xFF
        btfss   Z       ;jsou-li vsechny uvolnene
        goto    Stav_C
        return          ;ukonci podprogram
Stav_D  clrf    flags   ;zrusi stav ENTER a ESC
        bsf     ESC     ;nastavi ESC
        goto    Stav_C  ;skoci na testovani uvolneni klavesy
Stav_E  clrf    flags   ;zrusi stav ENTER a ESC
        bsf     ENTER   ;nastavi ENTER
        goto    Stav_C  ;skoci na testovani uvolneni klavesy
;*********************************************************
Init    bsf     RP0         ;vyssi stranka pameti dat
        movlw   0x00
        movwf   trisb       ;port B vystupni
        movwf   trisa       ;port A vystupni
        movlw   B'11010011'
        movwf   optreg      ;konfigurace RTCC a preddelicky
        bcf     RP0         ;nizsi stranka pameti dat
        bcf     T0IF        ;vynulovani flagu preruseni
                            timeru0
        bsf     T0IE        ;povoleni preruseni od timeru0
        bsf     GIE         ;globalni povoleni preruseni
        clrf    led         ;zhasne LED
;*********************************************************
Setup   call   Cti_tl    ;precte klavesy
        call   Stav_tl   ;vyhodnoti stav
```

```
        btfss  ENTER      ;ne-li pri resetu stisknut ENTER
        goto   Main       ;skoci na Main
Setup_A movlw  0x0F
        movwf  disp1      ;zobrazi .
        movwf  disp2      ;zobrazi .
        movwf  disp3      ;zobrazi .
        movwf  disp4      ;zobrazi .
        movlw  0x04
        movwf  cntk       ;zapise 4 do citace zadanych znaku
Setup_B movlw  0x0A
        call   Cekej      ;chvilku pocka
        call   Cti_tl     ;precte klavesy
        call   Stav_tl    ;vyhodnoti stav
        btfsc  ESC
        goto   Setup_A    ;je-li stisknut ESC, skoci
                          na ;zacatek
        movf   stav,w
        btfsc  Z          ;neni-li stisknuta zadna klavesa
        goto   Setup_B    ;skoci zpet
        decf   cntk,w     ;dekrementuje citac zadanych znaku
        addlw  kod4       ;pricte adresu kod4
        movwf  fsr        ;a vznikne pointer na pamet kodu
        movf   stav,w
        movwf  indf       ;ulozi stav do pameti kodu
        decf   cntk,w     ;dekrementuje citac zadanych znaku
        addlw  disp1      ;pricte adresu disp1
        movwf  fsr        ;a vznikne pointer
                          na ;sedmisegmentovku
        movlw  0x0B       ;na sedmisegmentovce rozsviti -
        movwf  indf
        decfsz cntk,f     ;dekrementruje citac zadanych znaku
        goto   Setup_B    ;ne-li nulovy, skoci zpet
Setup_C movlw  0x0A       ;je-li nulovy, pak byly zadany
                          4 ;znaky
        call   Cekej      ;chvilku ceka
        call   Cti_tl     ;precte tlacitka
        call   Stav_tl    ;vyhodnoti stav
        btfsc  ESC        ;stisk ESC zrusi nastavene znaky
        goto   Setup_A
        btfss  ENTER
        goto   Setup_C    ;ceka na stisk ENTER
        movlw  0x04
        movwf  cntt       ;ulozi 4 do citace testovani
Setup_D movf   cntt,w
        movwf  eeadr      ;adresa znaku v pameti EEPROM
        bsf    RP0        ;vyssi stranka pameti dat
        bsf    RD         ;zahajeni cteni
        bcf    RP0        ;nizsi stranka pameti dat
        addlw  (kod4 - 1);pricte k citaci testovani adresu
        movwf  fsr        ;kod4 zmensenou o jednicku
        movf   indf,w     ;precte z pameti dat znak
```

```
        subwf eedata,w  ;porovna ho s kodem ulozenym
                        v ;EEPROM
        btfss Z
        goto  $         ;je-li ruzny, procesor se zablokuje
        decfsz cntt,f   ;je-li stejny, testuje se dalsi znak
        goto  Setup_D   ;probehlo-li testovani 4x uspesne
Setup_E movlw 0x0F      ;tak pokracuje dal
        movwf disp1     ;zobrazi .
        movwf disp2     ;zobrazi .
        movwf disp3     ;zobrazi .
        movwf disp4     ;zobrazi .
        movlw 0x04
        movwf cntk      ;ulozi 4 do citace znaku
Setup_F movlw 0x0A
        call  Cekej     ;chvilku ceka
        call  Cti_tl    ;precte tlacitka
        call  Stav_tl   ;vyhodnoti stav
        btfsc ESC
        goto  Setup_E   ;je-li stisknute ESC, vrati se zpet
        movf  stav,w
        btfsc Z         ;neni-li stisknute zadne tlacitko
        goto  Setup_F   ;tak se vrati na cteni tlacitek
        decf  cntk,w    ;dekrementuje citac znaku a ulozi do
                        W
        addlw kod4      ;pricte adresu kod4
        movwf fsr
        movf  stav,w
        movwf indf      ;ulozi znak do pameti kodu
        decf  cntk,w    ;dekrementuje citac znaku a ulozi do
                        ;W
        addlw disp1     ;pricte adresu disp1
        movwf fsr
        movf  stav,w
        movwf indf      ;ulozi znak do pameti
                        sedmisegmentovky
        decfsz cntk,f
        goto  Setup_F   ;po 4 pruchodech smyckou
                        pokracuje ;dal
Setup_G movlw 0x0A
        call  Cekej     ;chvilku ceka
        call  Cti_tl    ;precte tlacitka
        call  Stav_tl   ;vyhodnoti jejich stav
        btfsc ESC       ;je-li stisknut ESC, skoci zpet
        goto  Setup_F   ;na zadavani znaku
        btfss ENTER
        goto  Setup_G   ;ceka na stisk ENTER
        movlw 0x04
        movwf cntt      ;ulozi 4 do citace testovani
        movlw 0x04
        movwf cntw      ;ulozi 4 do citace zapisu do EEPROM
        bsf   RP0
```

```
         bsf    WREN       ;povoli zapis do EEPROM
         bcf    EEIF       ;vynuluje flag ukonceni zapisu
                           do ;EEPROM
         bcf    RP0
Setup_H  movf   cntw,w     ;obsah citace zapisu do EEPROM
         movwf  eeadr      ;je pouzit jako adresa do EEPROM
         addlw  (kod4 - 1) ;do W ulozi adresu kod4 - 1
         movwf  fsr        ;vznikne ukazatel na heslo
         movf   indf,w
         movwf  eedata     ;ulozi znak do eedata
         bcf    GIE        ;zakazani preruseni
         bsf    RP0
         movlw  0x55       ;sekvence zapisu do EEPROM
         movwf  eecon2     ;-II-
         movlw  0xAA       ;-II-
         movwf  eecon2     ;-II-
         bsf    WR         ;-II-
         bsf    GIE        ;povoleni preruseni
         btfss  EEIF
         goto   $-1        ;ceka na ukonceni zapisu do EEPROM
         bcf    EEIF       ;vymaze flag ukonceni zapisu
                           do ;EEPROM
         bcf    RP0
         decfsz cntw,f
         goto   Setup_H    ;po 4 zapisovych cyklech
                           pokracuje ;dal
         bsf    RP0
         bcf    WREN       ;zakazani zapisu do EEPROM
         bcf    RP0
;*********************************************************
Main     movlw  0x0F
         movwf  disp1      ;zobrazi .
         movwf  disp2      ;zobrazi .
         movwf  disp3      ;zobrazi .
         movwf  disp4      ;zobrazi .
         clrf   kod1
         clrf   kod2
         clrf   kod3
         clrf   kod4       ;vymaze pameti hesla
         clrf   flags      ;zrusi stav ENTER a ESC
         movlw  0x04
         movwf  cntk
Main_A   movlw  0x0A
         call   Cekej      ;chvilku pocka
         call   Cti_tl     ;precte klavesy
         call   Stav_tl    ;vyhodnoti stav
         btfsc  ESC
         goto   Main       ;je-li stisknut ESC, skoci na
                           zacatek

         movf   stav,w
```

```
        btfsc  Z           ;neni-li stisknuta zadna klavesa
        goto   Main_A      ;skoci zpet
        decf   cntk,w      ;dekrementuje citac zadanych znaku
        addlw  kod4        ;pricte adresu kod4
        movwf  fsr         ;a vznikne pointer na pamet hesla
        movf   stav,w
        movwf  indf        ;ulozi stav do pameti hesla
        decf   cntk,w      ;dekrementuje citac zadanych znaku
        addlw  disp1       ;pricte adresu disp1
        movwf  fsr         ;a vznikne pointer
                           na ;sedmisegmentovku
        movlw  0x0B        ;na sedmisegmentovce rozsviti -
        movwf  indf
        decfsz cntk,f      ;dekrementruje citac zadanych
                           znaku
        goto   Main_A      ;ne-li nulovy, skoci zpet
Main_B  movlw  0x0A        ;je-li nulovy, pak byly zadany
                           4 ;znaky
        call   Cekej       ;chvilku ceka
        call   Cti_tl      ;precte tlacitka
        call   Stav_tl     ;vyhodnoti stav
        btfsc  ESC         ;stisk ESC zrusi nastavene znaky
        goto   Main
        btfss  ENTER
        goto   Main_B      ;ceka na stisk ERNTER
        movlw  0x04
        movwf  cntt        ;ulozi 4 do citace testovani
Main_C  movf   cntt,w
        movwf  eeadr       ;adresa znaku v pameti EEPROM
        bsf    RP0
        bsf    RD          ;zahajeni cteni
        bcf    RP0
        addlw  (kod4 - 1)  ;pricte k citaci testovani adresu
        movwf  fsr         ;kod4 zmensenou o jednicku
        movf   indf,w      ;precte z pameti dat znak
        subwf  eedata,w    ;porovna ho se znakem ulozenym v
                           EEPROM
        btfsc  Z
        goto   Main_D      ;je-li shodny, skoci na Main_D

        movlw  0x0C
        movwf  disp4       ;zobrazi E
        movlw  0x0D
        movwf  disp3       ;zobrazi r
        movlw  0x0E
        movwf  disp2       ;zobrazi r.
        movlw  0x0A
        movwf  disp1       ;nezobrazi nic
        movlw  0x48
        call   Cekej       ;pocka asi 3s
        goto   Main        ;skoci na zacatek zadavani hesla
```

```
Main_D  decfsz cntt,f
        goto   Main_C     ;testuji se 4 znaky
        movlw  0x0A
        movwf  disp1      ;zhasne displej
        movwf  disp2      ;zhasne displej
        movwf  disp3      ;zhasne displej
        movwf  disp4      ;zhasne displej
        movlw  0x01
        movwf  led        ;rozsviti se LED
        movlw  0x48
        call   Cekej      ;ceka asi 3s
        clrf   led        ;zhasne LED
        goto   Main       ;skoci na zacatek zadavani hesla
;*******************************************************
        org    0x2100
        de     0x00
        de     0x04
        de     0x03
        de     0x02
        de     0x01
;*******************************************************
        end
```

název instrukce	popis	počet cyklů	ovlivňuje
ADDLW k	Sečte obsah registru W s konstantou	1	C, DC, Z
ADDWF f,d	Sečte obsah W s registrem f	1	C, DC, Z
ANDLW k	Provede AND mezi registrem W a konstantou k	1	Z
ANDWF f,d	Provede AND mezi registrem W a registrem f	1	Z
BCF f,b	Vynuluje bit b registru f	1	-
BSF f,b	Nastaví bit b registru f do 1	1	-
BTFSC f,b	Je-li bit b registru f = 0, přeskočí následující instrukci (provede místo ní instrukci NOP)	1(2)	-
BTFSS f,b	Je-li bit b registru f = 1, přeskočí následující instrukci (provede místo ní instrukci NOP)	1(2)	-
CALL k	Zavolá podprogram	2	-
CLRF f	Vynuluje obsah registru f	1	Z
CLRW	Vynuluje obsah registru W	1	Z
CLRWDT	Vynuluje WDT a předdělička, když je k němu připojená	1	TO, PD
COMF f,d	Provede negaci (komplement) registru f	1	Z
DECF f,d	Zmenší obsah registru f o jedničku	1	Z
DECFSZ f,d	Od obsahu registru f odečte jedničku, je-li výsledek po odečtení 0, přeskočí se následující instrukce (provede se místo ní instrukce NOP)	1(2)	-
GOTO k	Provede nepodmíněný skok na adresu k	2	-
INCF f,d	Zvětší obsah registru f o jedničku	1	Z
INCFSZ f,d	K obsahu registru f přičte jedničku, je-li výsledek do odečtení 0, přeskočí se následující instrukce (provede se místo ní instrukce NOP)	1(2)	Z
IORLW k	Provede OR mezi registrem W a registrem f	1	Z
IORWF f,d	Provede OR mezi registrem W a registrem f	1	Z
MOVF f,d	Přesune obsah registru f	1	Z
MOVLW k	Přesune konstantu k do registru W	1	-
MOVWF f	Přesune obsah registru W do registru f	1	-
NOP	Prázdná operace. Nic se neprovede	1	-
OPTION	Přesune obsah registru W do registru OPTION	1	-
RETLW k	Navrátí se z podprogramu, registr W naplní konstantou k	2	-
RETURN	Navrátí se z podprogramu	2	-
RETFIE	Navrátí se z podprogramu obsluhujícího přerušení	2	-
RLF f,d	Rotuje obsah registru f o jeden bit doleva přes C bit stavového registru	1	C
RRF f,d	Rotuje obsah registru f o jeden bit doprava přes C bit stavového registru	1	C
SLEEP	Mikrokontrolér přejde do stavu SLEEP. Vynuluje WDT a předdělička	1	TO, PD
SUBLW k	Odečte obsah registru W od konstanty k	1	C, DC, Z
SUBWF f,d	Odečte obsah W od registru f	1	C, DC, Z
SWAPF f,d	Prohodí horní a dolní půlbyte registru f	1	-
TRIS f	Přesune obsah registru W do registru TRIS portu f	1	-
XORLW k	Provede XOR mezi registrem W a konstantou k	1	Z
XORWF f,d	Provede XOR mezi registrem W a registrem f	1	Z