

**Škola programování PIC**

**<http://www.pandatron.cz/>**

# Škola programování PIC 1

**V tomto seriálu se pokusím vysvětlit funkci procesorů PIC (především se zaměřením na oblíbený typ 16F84) a jejich programování tak, aby to pochopili i úplní začátečníci!**

Než začneme, je potřeba vysvětlit některé základní věci:

## 1) jaký je rozdíl mezi BITEM a BYTEM

1 BIT = jedna logická hodnota, tedy buď logická 0 (0V), nebo logická jednička 1 (5V)

1 BYT = [jeden bajt] 8 bitů, neboli osm logických hodnot. Například 10100101, nebo 00101101 atd.

128	64	32	16	8	4	2	1
0	1	0	0	1	0	1	1

 = 75  
nebo taky 4Bh

## 2) základní číselné soustavy a jejich převod

Před programováním potřebujeme znát alespoň tři základní soustavy: binární, hexadecimální a dekadickou.

Tak nejprve dekadická, neboli desítková. To je ta, co běžně používáme (1, 2, 3, 28, 247...) a jistě ji všichni dobře známe.

Jako další bych popsal binární (dvojkovou) soustavu. S ní jsme se dnes již také setkali. Je to ta, která obsahuje pouze jedničky a nuly (například číslo "10011101b" (b=binární)). Její převod na běžnou dekadickou soustavu je jednoduchý a je naznačen na obrázku. Vezme se binární číslo a od zadu (zprava) se nad každý jeho znak napíše jeho hodnota. Začíná se číslem 1 a pokračuje se vždy násobky dvěma: 1, 2, 4, 8, 16, 32... Poté se již jen sečtou hodnoty čísel, pod kterými je jednička (na obrázku jsou pro jistotu podtrženy).

Poslední je hexadecimální (šestnáctková) soustava. Ta obsahuje pouze tyto znaky: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Ano, obsahuje jen 16 znaků, načež dvouciferná čísla (10 a výše) jsou nahrazeny písmeny ze začátku abecedy. Tak například 9h, další je Ah atd. Jelikož jeden tento znak = 4 bity, používají se častěji dvouciferná čísla jako třeba B5h (h=hexa).

Převod z binární do hexadecimální soustavy se provádí podobně jako převod na dekadickou soustavu, jen s tím rozdílem, že se převádí binární čísla po čtyřech

bitech (0000b). Opět se zprava nad každý bit napíše jeho hodnota (1,2,4,8) a opět se sečtou ty hodnoty, pod nimiž je jednička. Pokud vyjde číslo jako 5, 8 a podobně, je vše v pořádku. Pokud ale vyjde číslo 10 až 15, musí se ještě převést na jedno písmeno ze začátku abecedy (10=A, 11=B, 12=C...15=F).

### 3) Registr

Registr, jako samostatná součástka, slouží jako malá jednobitová (případně i větší) paměť. Hodnota, kterou do ní vložíme v ní zůstane tak dlouho, dokud ji dalším zápisem nezměníme.

Zde se registrem nazývá místo (část paměti nebo i samostatný blok), který obsahuje 8 bitů. Takže když má například paměť RAM kapacitu 64 bytů (64×8bitů), znamená to, že má 64 registrů. Takže je to vlastně jiný název pro 1 byt.

Registry ale nejsou jen rozdělená paměť, ale jsou i jako samostatné bloky, které ale jako malá paměť slouží. Například obvody jako Výstupní registr (to je malá paměť, jejíž obsah je k dispozici na pinech procesoru).

### 4) Přerušení

Nejprve co a k čemu to vlastně je.

Přerušení je podobné resetu a vykoná se vždy, když se NĚCO stane. Tím něco se dá nastavit například přetečení čítače, změna hodnoty na nějakém vývodu a podobně.

Při zapnutí nebo resetu procesoru se tedy začne číst z jeho programové paměti program od adresy 00h. Přerušení se od resetu odlišuje tím, že procesor neskočí zpět na první řádek programu, ale až na čtvrtý (adresa 04h) !!!

Jak to využít? Jednoduše. Na první řádek programu se vloží příkaz, aby procesor skočil na nějaké jiné místo, kde začíná vlastní program a na čtvrtý řádek se vloží jiný odkaz, který donutí procesor, který se dostal sem, aby skočil na místo, kde je umístěn podprogram využívající toto přerušení.

Samozřejmě se neodpočítávají řádky, ale slouží k tomu přímo speciální příkazy. Vše bude ale později ještě podrobně popsáno.

### 5) Ostatní slova

Instrukce = jeden příkaz (řádek) v programu

Přetečení časovače = znamená jen to, že čítač dočítal do konce (celý se naplnil) a přetekl

Tak to by bylo pro začátek všechno, teď co vůbec budeme potřebovat.

Tak předně potřebujeme nějaké PC. Úplně stačí nějaké obyčejné na kterém běží DOS a má volný (a funkční) jeden sériový port pro připojení programátoru. Na něm budeme psát programy v běžném textovém editoru, výsledné soubory budeme poté programem MPASM převádět do souborů potřebných k programátoru a hlavně tedy ještě potřebujeme vlastní programátor. Dále je dobré mít na PC nějaký simulátor těchto procesorů, například jednoduchý, ale přehledný program PSIM. Všechny tyto programy, návod na instalaci a zapojení jednoduchého programátoru si můžete stáhnout jako jeden soubor v Download sekci.

## Škola programování PIC 2

**Nyní se podíváme na to, co je to vlastně procesor a co umí.**

Předně bych chtěl uvést, že procesor není žádná zázračná součástka a že toho prakticky mnoho neumí. V podstatě umí jen přesun dat (například mezi paměti a výstupy) a základní matematické operace. Ale má jednu velkou výhodu: Umí pracovat samostatně, nezávisle na okolí. A to už běžné číslicové obvody neumějí (vyjma časovačů). Co to znamená? Například to, že na jeden povel umí vykonat spoustu různých operací a na další jich vykoná jinou spoustu, což by se s běžnými klopnými obvody dělalo dost těžko.

Tak se na procesor podíváme trochu podrobněji.

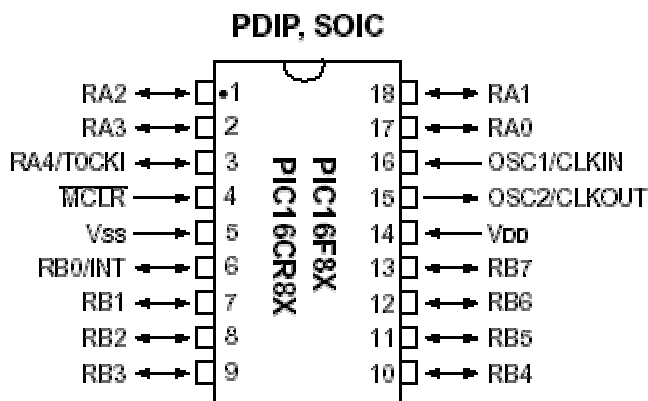
Všechny procesory PIC jsou typu RISC, tedy procesory s omezenou instrukční sadou. To znamená, že je k dispozici pouze několik málo příkazů (35), jejichž zpracování trvá pouze jeden (maximálně dva) strojové takty a v paměti zabírají vždy po jednom slově.

Jeden strojový takt je rychlost, se kterou procesor zpracovává příkazy a odvíjí se od kmitočtu oscilátoru jeho vydělením 4. Je-li například použit krystal 4MHz, můžete si lehce spočítat, že jeden takt trvá 1us (*jednu mikrosekundu*). Během tohoto jednoho taktu procesor provede jednu instrukci a zároveň si připraví tu následující. Většina instrukcí proto trvá přesně jeden takt. Pouze instrukce, které naruší plynulý běh

programu (skokové instrukce) trvají dva takty, protože procesor nevyužije připravenou instrukci a musí načítat další.

### Rozložení vývodů procesoru a základní technická data

Vlastní rozložení vývodů na procesoru je patrné z tohoto obrázku:



#### A teď s popisem:

RA0 - RA4 - vývody portu A (5)

RB0 - RB7 - vývody portu A (8)

MCLR - reset, stačí připojit přímo na +Ucc (procesor se resetuje sám)

Vss - zem napájecího napětí

Vdd - plus napájecího napětí (2 - 6 V)

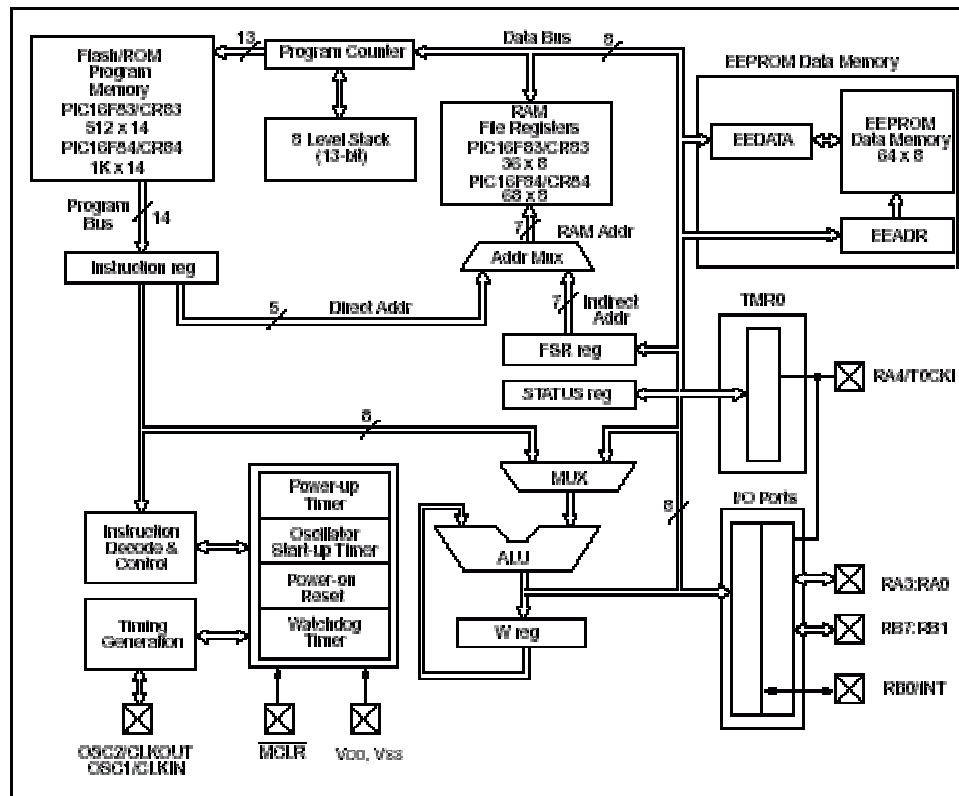
OSC1, OSC2 - vývody na připojení oscilátoru

Počet instrukcí	<b>35</b>
Velikost programové paměti	<b>1024 slov</b>
Velikost datové paměti RAM	<b>68 bytů</b>
Velikost paměti EEPROM	<b>64 bytů</b>
Počet I/O vývodů	<b>13</b>
Počet časovačů	<b>1</b>
Velikost zásobníku	<b>8 úrovní</b>
Typy oscilátorů	<b>RC, XT, HS, LP</b>
Napájecí napětí	<b>2 až 6 V</b>
Odebíraný proud	< 1uA – 2V standby 15uA – 2V, 32 kHz < 2mA – 5V, 4 MHz
Zatížitelnost portů	<b>20mA</b>
Další vybavení	Power-on Reset Power-up Timer

	Oscillator Start-up Timer
	Watchdog Timer
	Code-protection
	SLEEP mode

## Co obsahuje uvnitř

To je nejlépe vidět na obrázku jeho blokového zapojení. Není na první pohled moc přehledné a tak nejdůležitější části raději popíšu:



## ALU

[Aritmeticko Logická Jednotka]

Je to hlavní část každého procesoru. Zde se vykonávají všechny příkazy a matematické operace.

## W reg

Další neméně důležitou částí procesoru je registr W. Je to taková malá mezipaměť, přes kterou se provádí většina operací. Například, nemůžeme z paměti přesunout nějakou hodnotu přímo na výstupy. Nejprve si ji uložíme do tohoto registru a až dalším příkazem ji přesuneme dál (na výstupní registr).

## Program Memory

Jak už název napovídá, je to paměť ve které je uložen vlastní program. U tohoto procesoru, jakož i u ostatních které mají v názvu písmeno "F", je typu FLASH. To znamená, že je možné do ní program nahrát a vždy ho jedním povelům smazat a nahrát jiný. U levnějších procesorů je tato paměť typu EPROM, takže ji lze smazat pouze UV zářením, nebo u verzí bez mazacího okénka Rentgenovým zářením. Tato paměť má kapacitu 1024 slov (1 slovo = 14 bitů) a jeden příkaz v ní uložený zabírá pouze jedno slovo.

## RAM

Druhou pamětí v procesoru je paměť RAM se 68 byty. Ta je rozdělena do dvou tzv. BANK (jakoby na dvě paměti) a celkem obsahuje 68 registrů (bajtů). Z toho v první bance (adresy 00h - 4Fh) je prvních 12 registrů systémových (do adresy 0Bh) a zbytek jsou uživatelské registry. V druhé bance (adresy 80h - CFh) je opět na začátku 12 registrů systémových (do 8Bh) a zbytek uživatelských. Ovšem tyto uživatelské registry již nejsou fyzické. Je to jen kopie registrů z první banky. Takže ze začátku paměti jsou vždy uloženy různá nastavení a informace o běhu procesoru a zbytek paměti je plně k dispozici programátorovi na odkládání různých mezivýsledků. Registrům a rozdělení paměti se ještě budeme podrobně věnovat později.

## EEPROM

Je to poslední větší paměť v procesoru (64 bytů). Komunikace s ní není tak jednoduchá jako s předchozí pamětí a při zápisu je relativně pomalá (desítky ms). Zato si svůj obsah uchová i po odpojení napájecího napětí, proto se hlavně hodí k ukládání různých nastavení, hesel a podobně. Výrobce udává její životnost minimálně 10.000.000 zápisů a svůj obsah si bez napájení uchová více než 40 let.

## I/O Ports

Další částí procesoru jsou vstupně-výstupní porty. Tento procesor jich obsahuje celkem 13 a jelikož se pracuje s byty, má je rozděleny do dvou portů. Celý port B (RB0 - RB7) a zbytek port A (RA0 - RA4).

Každý z těchto vývodů se dá zvlášť nastavit buď jako vstupní, nebo výstupní a to i kdykoliv při běhu procesoru.

## TMR

Tento procesor také obsahuje jeden 8 bitový čítač / časovač.

A další obvody, ty ale pro nás v tuto chvíli nejsou podstatné.

# Škola programování PIC 3

## Registry a jejich popis.

Rozdělení registrů je patrné na následujícím obrázku:

adresa	banka 0	banka 1
00h	INDF	
01h	TMR0	OPTION_REG
02h	PCL	
03h	STATUS	
04h	FSR	
05h	PORTA	TRISA
06h	PORTB	TRISB
07h	-	
08h	EEDATA	EECON1
09h	EEADR	EECON2
0Ah	PCLATH	
0Bh	INTCON	
0Ch až 4Fh	uživatelské reg.	

Jak bylo uvedeno v úvodu, registrem se nazývají 8 bitové části až už paměti nebo samostatné celky, které jak je patrné z tabulky mají svou adresu. Takže pokud například chceme něco zapsat na výstupy portu B procesoru, provedeme to uložením informace na adresu 06h, nebo slovem do registru PORTB. Všechny tyto registry je tedy možné adresovat buď přímo číslem, nebo je možné je "volat" jejich názvem, který mají přiřazený (PORTB).

Registry jsou dvojího druhu. První tzv. systémové jsou všechny do adresy 0Bh (nebo 8Bh). Ty uchovávají různá nastavení a informace o běhu procesoru a budou



podrobně popsány v zápětí.

Druhý typ registrů jsou tzv. uživatelské (adresy 0Ch až 4Fh). Ty jsou plně k dispozici programátorovi, který je využívá na průběžné odkládání různých mezivýsledků a informací. Jejich adresování je samozřejmě také možné přímo pomocí adres, ale pro usnadnění a zpřehlednění programu se vždy na začátku každého programu všem registrům (samozřejmě jen těm které se používají) přiřazují nějaké názvy a později se pracuje jen s nimi. Tyto názvy většinou vystihují použití toho kterého registru a přeci jen je jednodušší pamatovat si, že jsem si nějakou hodnotu uložil do registru s názvem například CISLO, než jeho konkrétní adresu. To samé platí i o systémových registrech, ale ty své názvy již mají a tak je není potřeba znovu určovat.

### Nyní se podíváme na nejpoužívanější systémové registry:

#### TMR0

Jedná se o obyčejný 8 bitový čítač. V registru OPTION\_REG je kromě jiných funkcí možné nastavit i zdroj signálu pro tento čítač. Je možné vybírat mezi vývodem RA4 a impulsy z vnitřních instrukčních cyklů (oscilátor / 4). K tomuto čítači je v případě potřeby možné ještě přiřadit předděličku s nastavitelným dělicím poměrem viz. následující registr.

#### OPTION\_REG

Slouží především k různým nastavením:

<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>
<b>RBPU</b>	<b>INTEDG</b>	<b>T0CS</b>	<b>T0SE</b>	<b>PSA</b>	<b>PS2</b>	<b>PS1</b>	<b>PS0</b>

**RBPU** - zdvihací rezistory na portu B. Velké odpory zapojené mezi všechny vývody portu B a + napájení.

0 - připojeny

1 - odpojeny

**INTG** - řídicí bit hrany přerušení

0 - přerušení nastane při sestupné (1-0) hraně na vývodu RB0

1 - při vzestupné (0-1)

**T0CS** - zdroj signálu pro čítač TMR0

0 - vnitřní instrukční cyklus

1 - hrana na vývodu RA4

**T0SE** - volba hrany pro čítání TMR0

0 - přičte při vzestupné hraně

1 - při sestupné

**PSA** - přiřazení předděliče

0 - předdělič u TMR0

1 - předdělič u WDT

**PS2, PS1, PS0** - dělicí poměr předděliče dle následující tabulky:

PS	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

## PCL

V tomto registru je dostupných posledních osm bitů čítače programu. Je možné jen nejen číst, ale i zapisovat a tím procesor přesunout na jiné místo programu.

Toho se využívá například při tvorbě tabulek. Vezme se nějaké číslo a to se přičte k hodnotě v tomto registru. Tím se dosáhne přesunu programového čítače o počet kroků rovný přičtenému číslu.

## STATUS

Tento registr obsahuje především informace o běhu procesoru (procesor je tedy nastavuje sám) a také je zde bit k přepínání mezi bankou 0 a 1 s registry (proto je také k dispozici v obou).

D7	D6	D5	D4	D3	D2	D1	D0
IRP	RP1	RP0	TO	PD	Z	DC	C

**IRP, RP1** - tyto dva bity se zde nepoužívají a při čtení se jeví jako 0

**RPO** - výběr banky registrů

0 - banka 0

1 - banka 1

**TO** - dává informaci o prošlém časovém limitu, nastaví se po přetečení WATCH DOG časovače

0 - byl

1 - nebyl

**PD** - indikuje použití SLEEP (úsporného režimu)

0 - byl

1 - nebyl

**Z** - výsledek operace byl roven nule. Při probírání dostupných příkazů bude u všech označeno co ovlivňuje.

0 - nebyl

1 - byl

**DC** - jako C s tím rozdílem, že indikuje přenos přes 16

0 - nebyl

1 - byl

**C** - byl přenos do vyššího řádu, tedy přes číslo 256 (registr přetekl)

0 - nebyl

1 - byl

## **PORTA, PORTB**

Jak už název napovídá, jedná se o registry přímo napojené na výstupní porty (které jsou výstupní a které vstupní se nastavuje následujícími registry). Takže, pokud chceme zapsat nějakou hodnotu na výstupy, zapíšeme ji do jednoho z těchto registrů a ona tam zůstane do doby, než ji dalším zápisem změníme.

Popis vývodů u procesoru plně odpovídá bitům v těchto registrech. Tak například vývod RA0 je "nulový" bit v registru PORTA, viz následující obrázek:

**PORTA**

<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>
-	-	-	<b>RA4</b>	<b>RA3</b>	<b>RA2</b>	<b>RA1</b>	<b>RA0</b>

## PORTB

<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>
<b>RB7</b>	<b>RB6</b>	<b>RB5</b>	<b>RB4</b>	<b>RB3</b>	<b>RB2</b>	<b>RB1</b>	<b>RB0</b>

## TRISA, TRISB

Jak bylo naznačeno v předchozím odstavci, tyto registry slouží k nastavení vlastnosti toho kterého vývodu. Tedy, nastavuje se zde, který vývod chceme mít jako vstup a který naopak výstup a to i kdykoliv při běhu programu.

Rozložení bitů plně odpovídá předchozímu obrázku a teď to nejdůležitější. Bit, který má hodnotu 0 je výstupní a naopak, bit s log. 1 je nastaven jako vstupní. Uložíme-li například do registru TRISB tuto binární hodnotu: 00001111, nastavíme tím vývody RB0-RB3 jako vstupní a RB4-RB7 jako výstupní. A ještě taková pomůcka, můžete si to pamatovat podle začátečních písmen anglických názvů (1-In, 0-Out).

Pokud některý z vývodů nastavíme jako výstupní, má z počátku hodnotu Log. 1 a ta se dá změnit zápisem do registrů PORTA a PORTB. Pokud ale některý z vývodů nastavíme jako vstupní, má vysokou impedanci. To znamená, že má velký odpor a jakoby se odpojí. Této vlastnosti se hojně využívá.

## EEDATA, EEADR, EECON1, EECON2

Všechny tyto registry slouží ke práci s pamětí EEPROM, která bude podrobně popsána později.

Teď jen tolik, že první registr slouží pro data a druhý pro adresu do které se bude číst, nebo zapisovat. Zbývající dva registry představují něco jako zámek. Protože se tato paměť často používá na důležité údaje (hesla a podobně), tak aby se, například při poruše procesoru, zabránilo nechtěnému přepsání této paměti, musí se před každým zápisem provést přesně stanované kroky, které do těchto registrů uloží dva klíče. Ale o tom všem až později.

## PCLATH

Zde je zbylých 5 bitů programového čítače, pokračování z registru PCL.

## INTCON

Poslední systémový registr je opět nastavovací. Slouží k nastavení přerušení (viz. první díl) případně indikuje typ vzniklého přerušení.

<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>
<b>GIE</b>	<b>EEIE</b>	<b>TOIE</b>	<b>INTE</b>	<b>RBIE</b>	<b>TOIF</b>	<b>INTF</b>	<b>RBIF</b>

**GIE** - povolení jakéhokoli přerušení

0 - zakázat

1 - povolit

**EEIE** - přerušení po dokončení zápisu do EEPROM

0 - zakázat

1 - povolit

**TOIE** - přerušení po přetečení časovače TMR0

0 - zakázat

1 - povolit

**INTE** - přerušení od vývodu INT - RB0

0 - zakázat

1 - povolit

**RBIE** - přerušení od změny na portu B

0 - zakázat

1 - povolit

**TOIF** - příznak přetečení TMR0 (ručně nulovat)

0 - nebyl

1 - byl

**INTF** - příznak vnějšího přerušení (ručně nulovat)

0 - nebyl

1 - byl

**RBIF** - příznak přerušení od portu B (ručně nulovat)

0 - nebyl

1 - byl

## Škola programování PIC 4

**Seznam a popis dostupných příkazů.**

Před začátkem programování je nutné ještě znát dostupné příkazy. Není nutné je znát nazpaměť (při používání se je naučíte sami), ale je dobré alespoň vědět co máme k dispozici.

Dostupné příkazy spolu s podrobným popisem jsou v následující tabulce. V úvodu jsem sice zmínil že jich je 35, ale příkaz CLRW (maže registr W) se nepoužívá a v simulátoru nefunguje, tak jsem ho ani do této tabulky nezařazoval.

<b>BYTOVÉ INSTRUKCE</b>			
<b>kód</b>	<b>popis</b>	<b>cykly</b>	<b>ovlivňuje</b>
<b>ADDWF f,d</b>	algebraický součet <b>W</b> a <b>f</b> uložit podle <b>d</b> $136 + 152 = 288$	1	C,DC,Z
<b>ANDWF f,d</b>	logický součin <b>W</b> a <b>f</b> podle <b>d</b> $11010$ $\underline{1100}$ $1000$	1	Z
<b>CLRF f</b>	nulování <b>f</b>	1	Z
<b>COMF f,d</b>	komplement <b>f</b> podle <b>d</b> vytvoří opačné bity z <b>f</b> a uloží podle <b>d</b> $11010 \rightarrow 00101$	1	Z
<b>DECF f,d</b>	dekrement <b>f</b> [-1] odečte jedničku	1	Z
<b>DECFSZ f,d</b>	dekrement <b>f</b> [-1], když je výsledek nula, následující instrukce se přeskočí	1 (2)	-
<b>INCF f,d</b>	inkrement <b>f</b> [+1] přičte jedničku	1	Z
<b>INCFSZ f,d</b>	inkrement <b>f</b> [+1], když je výsledek nula, následující instrukce se přeskočí	1 (2)	-
<b>IORWF f</b>	logický součet <b>W</b> a <b>f</b> $W - 11010$ $f - 01100$ $100110 - W$	1	Z
<b>MOVF f,d</b>	přesun registru <b>f</b> podle <b>d</b>	1	Z
<b>MOVWF f</b>	přesun <b>W</b> do <b>f</b>	1	-
<b>MOVLW k</b>	přesun čísla <b>k</b> do <b>W</b>	1	-
<b>NOP</b>	bezvýznamná instrukce (čeká 1 cyklus)	1	
	používá se na zpoždění procesoru		
<b>RLF f,d</b>	rotace <b>f</b> vlevo přes <b>C</b> $C \rightarrow D7,D6,D5,D4,D3,D2,D1,D0 \rightarrow C$	1	C
<b>RRF f,d</b>	rotace <b>f</b> vpravo přes <b>C</b> $C \rightarrow D7,D6,D5,D4,D3,D2,D1,D0 \rightarrow C$	1	C
<b>SUBWF f,d</b>	rozdíl <b>f</b> a <b>W</b> $f - W = d$ <b>C = 1</b> - výsledek byl kladný <b>C = 0</b> - výsledek byl záporný	1	C,DC,Z
<b>SWAPF f,d</b>	výměna čtveřice bitů v <b>f</b> $11110000 \rightarrow 00001111$	1	-

<b>XORWF f,d</b>	exklusivní součet <b>W</b> a <b>f</b> podle <b>d</b> $1010$ $1100$ $1001$	1	Z
<b>BITOVÉ INSTRUKCE</b>			
<b>kód</b>	<b>popis</b>	<b>cykly</b>	<b>ovlivňuje</b>
<b>BCF f,b</b>	nulování bitu <b>b</b> registru <b>f</b> - 0	1	-
<b>BSF f,b</b>	nastavení bitu <b>b</b> registru <b>f</b> - 1	1	-
<b>BTFSC f,b</b>	test bitu <b>b</b> v <b>f</b> , skok když je 0	1 (2)	-
<b>BTFSS f,b</b>	test bitu <b>b</b> v <b>f</b> , skok když je 1	1 (2)	-
<b>OSTATNÍ INSTRUKCE</b>			
<b>kód</b>	<b>popis</b>	<b>cykly</b>	<b>ovlivňuje</b>
<b>ADDLW k</b>	algebraický součet <b>k</b> a <b>W</b> , výsledek do <b>W</b> $136 + 152 = 288 -- W$	1	C,DC,Z
<b>ANDLW k</b>	logický součin <b>k</b> a <b>W</b> , výsledek do <b>W</b> $11010$ $1100$ $1000$	1	Z
<b>CALL k</b>	volání podprogramu	2	-
<b>CLRWDT</b>	nulování <b>WDT</b>	1	TO,PD
<b>GOTO k</b>	skok na návěští <b>k</b>	2	-
<b>IORLW k</b>	logický součet <b>k</b> a <b>W</b> , výsledek do <b>W</b> $11010$ $1100$ $100110$	1	Z
<b>RETFIE</b>	návrat z přerušení	2	-
<b>RETLW k</b>	návrat z podprogramu, <b>k</b> se uloží do <b>W</b>	2	-
<b>RETURN</b>	návrat z podprogramu	2	-
<b>SLEEP</b>	uspí procesor návrat přerušením nebo resetem	1	TO,PD
<b>SUBLW k</b>	rozdíl <b>k</b> a <b>W</b> , výsledek do <b>W</b> $k - W = W$ <b>C = 1</b> - výsledek byl kladný <b>C = 0</b> - výsledek byl záporný	1	C,DC,Z
<b>XORLW k</b>	exklusivní součet <b>k</b> a <b>W</b> , výsledek do <b>W</b> $11010$ $1100$ $1001$	1	Z

#### Vysvětlivky:

**f** - název (adresa) registru

**k** - číslo, název návěští

**d** - kam uložit výsledek

**1** - výsledek do **f**

**0** - výsledek do **W**

**C , DC , Z , TO , PD** - bity v registru **STATUS**

**b** - pozice bitu v registru (počítáno zprava)

**7 , 6 , 5 , 4 , 3 , 2 , 1 , 0**

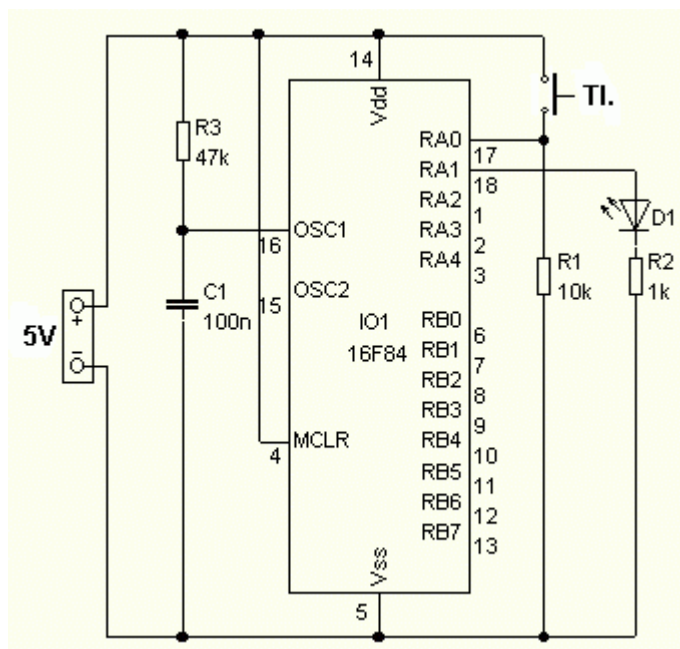
Jak je také vidět, většina příkazů potřebuje jen jeden strojový cyklus ( $OSC / 4$ ). 2 strojové cykly potřebují ty příkazy, které způsobí skok v programu. Je to logické, procesor má již připraven následující příkaz, ale pokud má skočit někam jinam, musí načíst nový. Stejně tak je to u příkazů, které mají uvedeno 1(2), například BTFSC. Pokud po něm program pokračuje dál, stačí mu jeden, pokud ale má někam skočit, potřebuje dva.

## Škola programování PIC 5

### První program

Teorie již bylo dost a tak se podíváme na nějaké jednoduché zapojení, na němž si vysvětlíme další funkce.

Jako první příklad si sestavíme zapojení podle následujícího obrázku a vytvoříme do procesoru program, aby se zařízení chovalo takto: Po zapnutí se navenek nic nestane, LED zůstane zhasnutá. Až po stisku tlačítka se LED rozsvítí a zůstane svítit i po jeho uvolnění. Po dalším stisku tlačítka LED opět zhasne a vše se bude opakovat. Prostě takové jednotlačítkové ON / OFF ovládání.



A tady je výsledný program:



```

LIST P=16F84
INCLUDE<P16F84.INC>
__CONFIG _PWRTE_ON & _WDT_OFF & _RC_OSC

#define TLAC    PORTA,0      ; Poznamky se pisi
#define LED     PORTA,1      ; za stredniky !!!

BSF    STATUS,RP0
MOVLW  B'00000001'
MOVWF  TRISA
BCF    STATUS,RP0

BCF    LED

START  BTFSS    TLAC
      GOTO     $-1
      BSF     LED
      BTFSC   TLAC
      GOTO     $-1

      BTFSS   TLAC
      GOTO     $-1
      BCF    LED
      BTFSC   TLAC
      GOTO     $-1

      GOTO    START

END

```

Docela malý program, že. Tak si ho postupně rozebereme a popíšeme.

***LIST P=16F84***

***INCLUDE<P16F84.INC>***

***\_\_CONFIG \_PWRTE\_ON & \_WDT\_OFF & \_RC\_OSC***

Tyto tři řádky slouží pro překladač a programátor. Na prvním řádku je určeno pro jaký typ procesoru je tento program určen a na dalším má napsáno v jakém souboru k němu najde potřebné údaje (je součástí překladače). Na třetím řádku je poté pro programátor určeno s jakými vlastnostmi se má procesor programovat. Když se to vezme popořadě, zpoždění po startu je zapnuto, Watchdog timer je vypnut a nakonec je určen typ oscilátoru. Tento poslední řádek není podmínkou, ale je vhodné ho přidávat, ušetří se později nastavování v programátoru.

***#DEFINE TLAC PORTA,0***

***#DEFINE LED PORTA,1***

Těmito příkazy se nastaví názvy používaným vývodům. Jak jsme si již vysvětlili, později se v programu může volat už jen tyto názvy a ne jejich adresy.

### ***BSF STATUS,RP0***

Protože ještě potřebujeme nastavit které vývody budou vstupní a které výstupní, musíme se přepnout do Banky 1 (viz. kapitola o registrech). To se provedlo tímto příkazem, který nám bit RP0 v registru STATUS nastavil na hodnotu log. 1 (BSF).

### ***MOVLW B'00000001'***

Ted' jsme si do pracovního registru (W) uložili tuto binární hodnotu.

### ***MOVWF TRISA***

A tímto jsme ji z pracovního registru zapsali do registru TRISA. Tím jsme nastavili vývod RA0 jako vstupní a zbývající vývody jako výstupní.

### ***BCF STATUS,RP0***

No a po nastavení se vrátíme zpět do Banky 0 opět změnou bitu RP0 tentokrátě však na log.0.

### ***BCF LED***

Na log. 0 jsme nastavili taky vývod s názvem LED.

### ***START BTFSS TLAC***

Slovo START slouží jako taková záložka. Až se později budeme chtít vrátit do tohoto místa, stačí napsat: GOTO START a jsme zde.

Příkaz za záložkou testuje vývod označený TLAC. Z tabulky příkazů je vidět, že je-li tento vývod v 0 pokračuje se normálně dalším příkazem. Má-li ovšem hodnotu 1 následující řádek se vynechá a pokračuje se až tím dalším (v našem případě je to BSF LED).

### ***GOTO \$-1***

Tento příkaz spolu s jeho hodnotou způsobí vrácení procesoru o jeden řádek více (\$-1).

Spolu s minulým příkazem jsme docílili toho, že procesor tu stále koluje a čeká, až bude mít vývod TLAC hodnotu 1. V tu chvíli se tento příkaz přeskočí a procesor se dostane na řádek:

***BSF LED***

Ted' jsme nastavili hodnotu 1 na vývod LED a tím rozsvítili připojenou LED diodu.

***BTFSC TLAC***

***GOTO \$-1***

Tyto dva příkazy jsou podobné těm co jsme tu již měli, jen s tím rozdílem, že pro přeskočení příkazu GOTO \$-1 je tentokrát potřeba přivést na vývod TLAC log. 0 (pustit tlačítko).

***BTFSS TLAC***

***GOTO \$-1***

***BCF LED***

***BTFSC TLAC***

***GOTO \$-1***

Program pokračuje téměř stejnou částí, jen s tím rozdílem, že po stisku tlačítka LED diodu zhasne.

***GOTO START***

Až se procesor dostane až sem, tímto příkazem ho vrátíme zpět na záložku START a tím se celý cyklus opakuje.

***END***

Poslední příkaz indikuje konec programu, ale díky předchozímu příkazu se sem procesor nikdy nedostane, ale musí se psát! Pokud by se sem procesor dostal, nic

horzného by se nestalo, prostě by proběhl zbytek volné programové paměti a začal by zase od začátku.

## Styl psaní programu

Program se píše v běžném Poznámkovém bloku a ukládá se s koncovkou **.ASM**. Proto ve Windows doporučuji nastavit pro tyto soubory automatické otevírání v Poznámkovém bloku (Notepad).

Jak je vidět z otištěného programu, program se píše do tří sloupců, které se oddělují klávesou **TAB**. Vyjma příkazů pro definování názvů se do prvního sloupce píše záložky, do druhého příkazy a do třetího sloupce jejich hodnoty. Vše pište velkými písmeny!

Do programu je také vhodné vkládat alespoň občas nějaké poznámky. Ty se píšou za středník ";" a již malými písmeny (pro přehlednost).

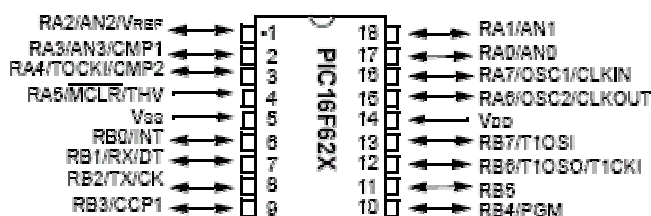
## A co teď s programem

Máme-li program napsaný a uložený v souboru s koncovkou ASM, spustíme si překladač ASM/HEX. V něm vybereme soubor který chceme přeložit a potvrdíme. Překladač nám z ASM souboru vygeneruje několik dalších s různými příponami. Soubor s příponou LST slouží k simulátoru. My ale pro programátor potřebujeme soubor s příponou HEX.

Teď již stačí k PC připojit programátor a vygenerovaný soubor naprogramovat do procesoru a celé zapojení odzkoušet.

# Škola programování PIC 6

## PIC 16F627 / 8



Jelikož je v dnešní době PIC 16F87 již poměrně zastaralý typ a jeho cena je značná, rozhodl jsem se v dalším díle přiblížit jeho náhradu - PIC 16F627 a 16F628. Tyto

procesory jsou téměř naprosto stejné - stejné pouzdro i rozmístění vývodů, stejné příkazy a programování (jako ostatně většina PICů)... Ale i když je jejich cena zhruba poloviční oproti tomu starému typu, mají spoustu nových funkcí, viz následující tabulka:

Počet instrukcí	<b>35</b>
Velikost programové paměti	<b>1024</b> slov - 16F627 <b>2048</b> slov - 16F628
Velikost datové paměti RAM	<b>224</b> bytů
Velikost paměti EEPROM	<b>128</b> bytů
Počet I/O vývodů	<b>16</b>
Počet časovačů	<b>3</b>
Velikost zásobníku	<b>8</b> úrovní
Typy oscilátorů	<b>RC, XT, HS, LP, IntRC</b>
Napájecí napětí	<b>3 až 5,5 V</b>
Odebíraný proud	< <b>1uA</b> – 3V standby <b>15uA</b> – 3V, 32 kHz < <b>2mA</b> – 5V, 4 MHz
Zatížitelnost portů	<b>20mA</b>
Další vybavení	Power-on Reset Power-up Timer Oscillator Start-up Timer Watchdog Timer Code-protection SLEEP mode
Nové funkce	2 x Komparátor Nastavitelný zdroj referenčního napětí PWM USART - sériová komunikace

Jak je vidět, zvětšili se všechny paměti. Mimochodem, zde je právě jediný rozdíl mezi 16F627 a 628 - ve velikosti programové Flash. No a dále přibyli nové funkce:

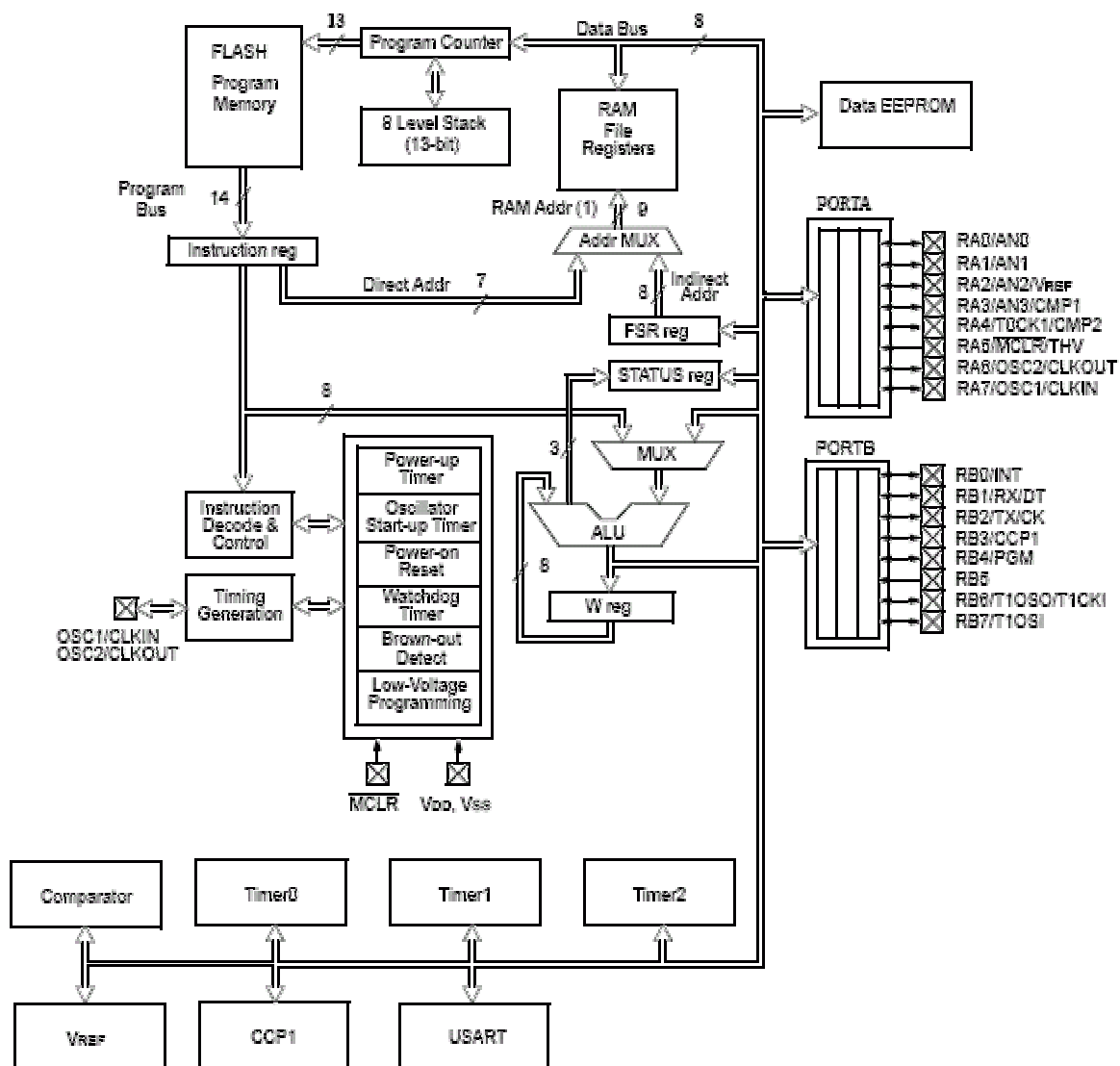
**2 x Komparátor** - snad netřeba dlouze vysvětlovat, 2 operační zesilovače u kterých si můžete nastavit kam připojit vstupy a kam výstup

**Vref** - interní zdroj referenčního napětí. Můžete si vytvořit jeden ze 16 dílů napájecího napětí (5V). Nelze vyvést ven, ale lze připojit k jednomu vstupu OZ.

**CCP** - kromě jiných funkcí nabízí i PWM = Vytvoření libovolného napětí s rozlišením 10b

**USART** - Univerzální Synchronní / Asynchronní Komunikace = sériový port na PC (RS232), nebo SCI

Na následujícím obrázku je vnitřní blokové schéma procesoru:



Ještě rozdělení registrů:

						File Address	
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
	07h		87h		107h		187h
	08h		88h		108h		188h
	09h		89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
	0Dh		8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh		8Fh		10Fh		18Fh
T1CON	10h		90h				
TMR2	11h		91h				
T2CON	12h	PR2	92h				
	13h		93h				
	14h		94h				
CCPR1L	15h		95h				
CCPR1H	16h		96h				
CCP1CON	17h		97h				
RCSTA	18h	TXSTA	98h				
TXREG	19h	SPBRG	99h				
RCREG	1Ah	EEDATA	9Ah				
	1Bh	EEADR	9Bh				
	1Ch	EECON1	9Ch				
	1Dh	EECON2*	9Dh				
	1Eh		9Eh				
CMCON	1Fh	VRCON	9Fh				
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 48 Bytes	11Fh 120h		
					14Fh 150h		
			EFh F0h		16Fh 170h		1EFh 1F0h
		accesses 70h-7Fh		accesses 70h-7Fh		accesses 70h - 7Fh	
			FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Zde je většina taky stejná až na změnu názvu `OPTION_REG` pouze na `OPTION`, ale pozor. Překladače jako MPASM berou stále jen ten starý název, takže používejte `OPTION REG`. Jinak zde však přibila spousta dalších speciálních registrů.

Uživatelské registry začínají až na adrese 20h (Banka 0).

### **I/O Piny**

Další velkou výhodou (nejen) u těchto dvou typů procesorů je možnost přepnutí funkce základních pinů jako jsou MCLR (reset) a OSC1, 2 na další I/O piny.

Resetovací pin se často nepoužívá (PIC má obvod resetu již v sobě ne jako Atmely), proto ho lze interně spojit trvale s kladným pólem napájení a tento pin využít jinak.

Zde je ale jedna nevýhoda, že tento pin lze využít pouze jako vstupní.

Často ani oscilátor není potřeba moc přesný, proto je i ten implementovaný (RC člen na 4MHz) a tyto dva piny lze již plně využít jako další In / Out.

V následujících dílech Školy programování PIC si popíšeme různé funkce právě těchto dvou procesorů. Bohužel však není v mých silách vše popisovat do detailu, proto doporučuji stáhnout si k procesorům katalogové listy, kde je vše popsáno mnohem podrobněji. Stáhnout si je můžete na stránkách výrobce:

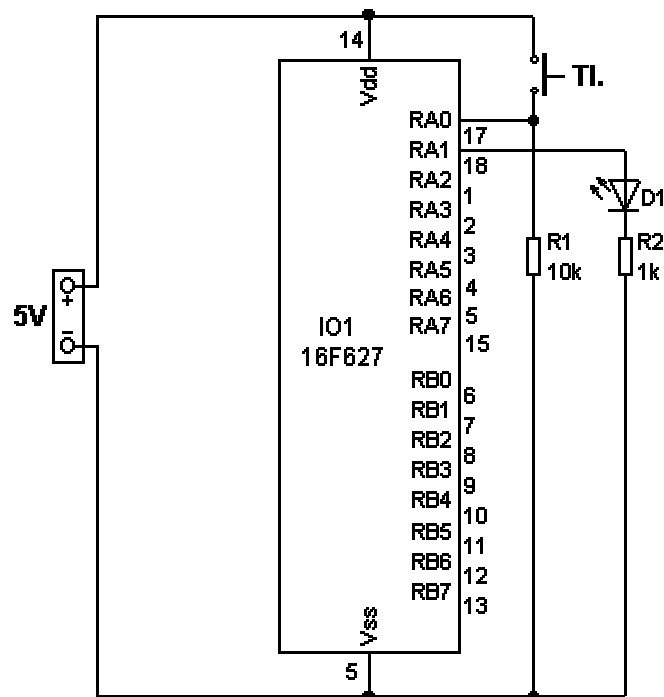
<http://microchip.com>.

## **Škola programování PIC 7**

### **První program - znovu pro 16F627-8**

Upravené zapojení. Již zde není potřeba nic kromě napájení a vlastních prvků. Reset i oscilátor je interní.





A upravený výsledný program:

```

LIST P=16F627
INCLUDE<P16F627.INC>
__CONFIG _PWRTE_ON & _WDT_OFF & _MCLRE_OFF & _BODEN_OFF & _LVP_OFF & _INTRC_OSC

#define TLAC    PORTA,0          ; Poznámky se pisi
#define LED     PORTA,1          ; za stredniky ???

        MOVLW   B'00000111'      ; typ komparatoru (off)
        MOVWF   CMCON
        BSF     STATUS,RP0
        MOVLW   B'00000001'
        MOVWF   TRISA
        BCF     STATUS,RP0

        BCF     LED

START   BTFSS   TLAC
        GOTO    $-1
        BSF     LED
        BTFSC   TLAC
        GOTO    $-1

        BTFSS   TLAC
        GOTO    $-1
        BCF     LED
        BTFSC   TLAC
        GOTO    $-1

        GOTO    START

END

```

Nebudu ho zde již tak podrobně popisovat, vše bylo již v 5 lekci, zde jen ty nové věci:

```
__CONFIG_PWRTE_ON & _WDT_OFF & _MCLRE_OFF & _BODEN_OFF &  
_LVP_OFF & _INTRC_OSC_NOCLKOUT
```

Tento řádek slouží k nastavení základních funkcí procesoru při programování. Když to vezmu zleva, tak PWRTE určuje že po zapnutí napájení bude procesor ještě asi 70ms čekat než se spustí, dále Watchdog je vypnut a Resetovací pin je vypnut. BODEN je ochrana při podpětí, klesne li napětí na asi 4,5V procesor se vypne, zde je to ale vypnuté (klidně si to však zapněte a vyzkoušejte). LVP je nízkonapěťové programování, doporučuji vždy vypínat a poslední nastavení se týká typu oscilátoru. Zde je nastaven na interní RC a kmitočet není vyveden ven (je možné si ho nechat vyvést na pin OSC2).

```
MOVLW B'00000111'
```

```
MOVWF CMCON
```

Nastavuje typ komparátoru. Zde je vypnut a piny jsou nastaveny na klasické I/O.

Jinak je vše při starém.

Zde si můžete uvedený program stáhnout:

[ASM](#)

[HEX](#)

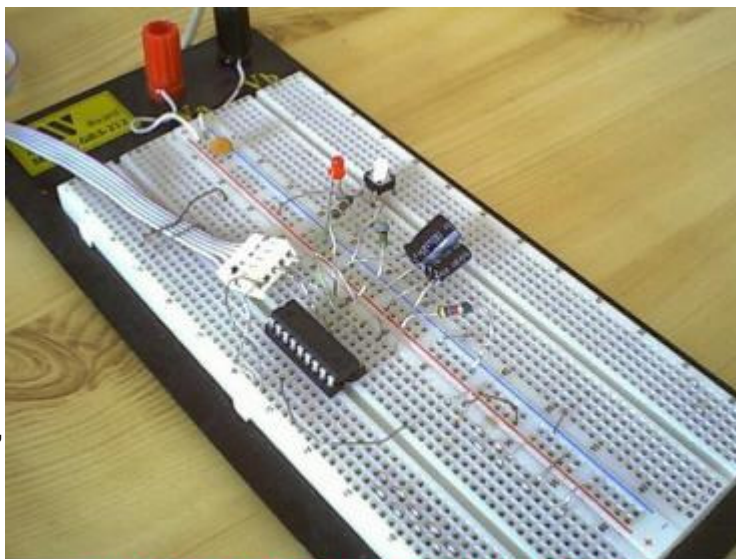
V následujících dílech této Školy programování PIC budu tedy všechny programy uvádět právě pro tento typ procesoru (16F627). Kdo má 16F628, stačí v úvodu změnit jméno procesoru, jinak vše zůstane stejné.

Kdo má 16F84, změni to samé, ale zároveň musí vždy změnit i dnes popisované nastavení na nastavení použité v 5 dílu školy PIC a smazat nastavení komparátoru. V budoucích dílech již však budu popisovat složitější věci, které tento typ procesoru ani neměl takže máte smůlu.

## **Škola programování PIC 8**

## Zpoždění - delay.

Snad nejčastějšími rutinami používanými u procesorů jsou čekací smyčky. Procesor totiž běží většinou na 4MHz což je poměrně dost a často je potřeba ho chvíli zdržet. Klasickým použitím může být například odstranění zákmitů na tlačítkách, nebo i jen tvorba určitého kmitočtu či časové prodlevy.



*Zkušební zapojení s připojeným programátorem*

Problém je ale v tom, že procesor nezná žádné příkazy typu delay() a podobně, běží stále dál a dál. Jediný příkaz na zdržení je NOP, ten však čeká jen jeden instrukční cyklus což při 4MHz oscilátoru je 1us a při potřebě čekání 1s by jich bylo potřeba jeden milión!

U procesorů se tedy dělají již zmíněné čekací smyčky. To je část programu, kterou procesor běží několikrát za sebou a tím se zdrží:)

Konkrétně se to dělá tak, že do jednoho nebo více registrů se vloží určité číslo, od kterého se pak při každém průběhu smyčkou odečte jednička. Lze to však samozřejmě udělat i opačně, že se registr vynuluje a postupně se přičítá.

Tady je první příklad:

	<b>BCF</b>	<b>LED</b>
	<b>CLRF</b>	<b>CISLO</b>
<b>ZPET</b>	<b>INCF</b>	<b>CISLO,1</b>
	<b>GOTO</b>	<b>ZPET</b>
	<b>BSF</b>	<b>LED</b>

Nejedná se samozřejmě o celý program, ale jen o uvedenou rutinu.

Takže co program dělá:

Nejprve zhasne LED, poté vymaže registr s názvem CISLO. Pak k tomuto registru postupně přičítá jedničku a čeká na jeho naplnění, kdy znovu rozsvítí LED.

**INCFSZ CISLO,1** - tento příkaz funguje tak, že přičte jedničku k registru CISLO a novou hodnotu uloží zpět do tohoto registru (to ta jednička na konci, při nule zůstane výsledek jen v pracovním registru a CISLO se nezmění). Pokud se nic neděje, pokračuje procesor na následujícím řádku příkazem GOTO. Pokud se však již registr CISLO naplnil a přetekl (256), přeskočí se následující příkaz a skočí se až na ten v pořadí druhý. V našem případě tedy až na BSF LED.

Při počítání zpoždění vynecháme vše před a vše za a vezmeme pouze dva hlavní řádky: řádek s INCFSZ zabere jeden strojový cyklus a řádek s GOTO zabere dva = 3 strojový cykly \* maximální velikost registru 255 = **zpoždění 765 strojových cyklů**. To není mnoho, při 4MHz krystalu je to 765us.

Nic však nebrání použití dvou registrů, kdy od jednoho budeme stále odečítat (přičítat) jedničku a až přeteče, přičteme jedničku k tomu druhému. A zase budeme počítat s tím prvním:

	<b>BCF</b>	<b>LED</b>
	<b>CLRF</b>	<b>CISLO</b>
	<b>CLRF</b>	<b>CISLOA</b>
<b>ZPET</b>	<b>INCFSZ</b>	<b>CISLO,1</b>
	<b>GOTO</b>	<b>ZPET</b>
	<b>INCFSZ</b>	<b>CISLOA,1</b>
	<b>GOTO</b>	<b>ZPET</b>
	<b>BSF</b>	<b>LED</b>

Takhle jsme dosáhli zpoždění zhruba 196000 strojových cyklů.

Pokud by jste chtěli ještě víc, můžete přidat ještě další registry na počítání.

Jako třetí příklad tedy uvedu již celý program na zpoždění zhruba 1s. Zapojení bude stejné jako v minulém díle Školy PIC a LED zde bude blikat zhruba v 1s intervalech (1s zapnuta, 1s vypnuta).

Čítačem jsem měřil periodu blikání je 2 , 072 606 s.

Zde je ke stažení:

[ASM](#)

[HEX](#)

Uvedené rutiny lze použít i pro odstranění zákmitů tlačítek. V tom případě však je potřeba ještě u záložky ZPET příkaz na přičítání posunout o řádek níž a zde testovat zda je ještě tlačítko sepnuto. Pokud ano, vrátí se procesor zpět a tak čeká až na definitivní puštění tlačítka, po kterém jeste uvedenou dobu čeká zda se nejedná jen o zákmit a nemá se tedy znovu vrátit a počítat znovu.

Uvedená rutina může vypadat třeba takto:

START	BTFSS	TLAC	; ceka na stisk tlacitka
	GOTO	START	
			; tlacitko stisknuto, pokracuj
ZACNI	CLRF	CISLO	; smaz pocitaci registry
	CLRF	CISLOA	
ZPET	BTFSC	TLAC	; je jiz tlacitko pusteno?
	GOTO	ZACNI	; ne
	INCFSZ	CISLO,1	; ano, pocitej
	GOTO	ZPET	
	INCFSZ	CISLOA,1	
	GOTO	ZPET	
	BSF	LED	

A vložená do Prvního programu je zde ke stažení:

[ASM](#)

[HEX](#)

Můžete si vyzkoušet, že na další stisk reaguje až po chvíli klidu.

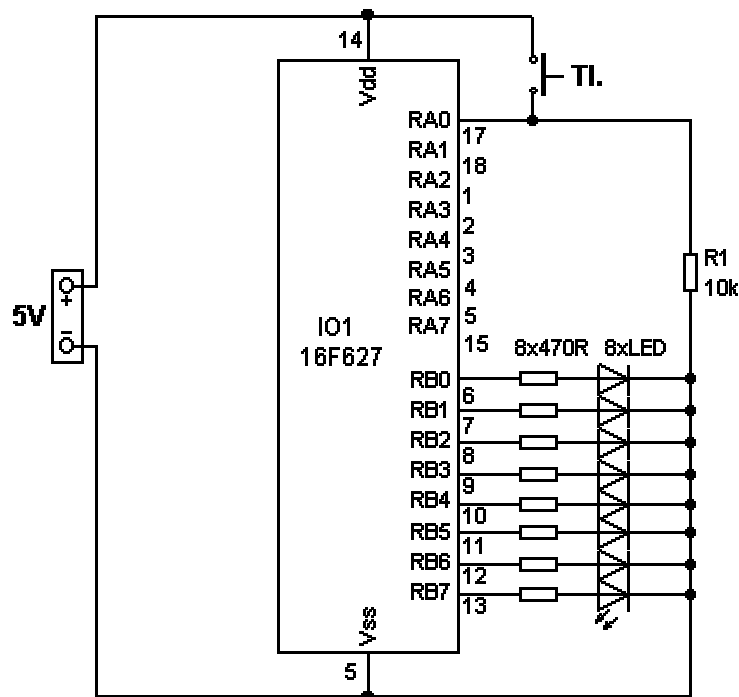
## Škola programování PIC 9

### Světelný had a ti další

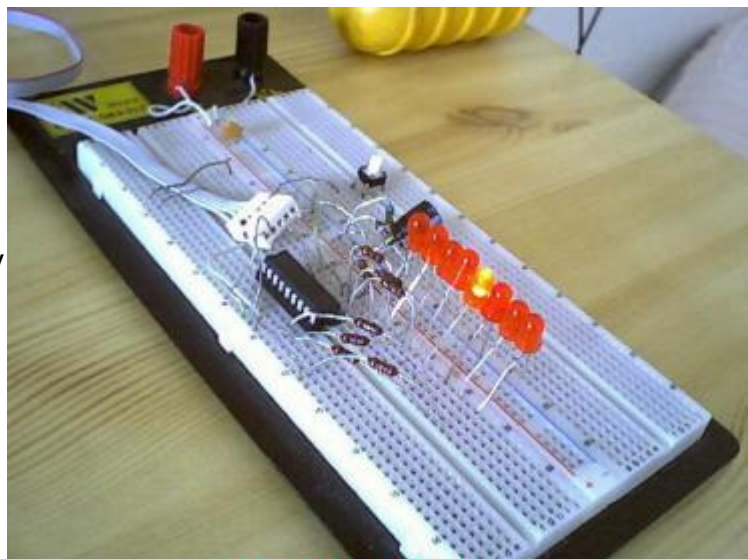
Ted', když už umíme čekací smyčky, můžeme se pustit do světelné efekty.

Jednoduchý blikáč byl popsán v minulém díle tak ted' padla volba na klasicky

Světelný had. Má 8 LED diod připojených přes rezistory na PORTB. Níže je schéma zapojení, na kterém je ještě z minula používané tlačítko. To jsem tam nechal záměrně, uvidíte později, ale ted' ho tam dávat nemusíte.



Program je již trochu větší, proto ho zde nebudu zobrazovat jako obrázek a jen text nemá smysl, protože by byl hrozně rozházený. Ale zrovna ho tu tedy rozeberu a podrobně popíšu.



*Světelný had v akci.*

***LIST P=16F627***

***INCLUDE<P16F627.INC>***

***\_\_CONFIG\_PWRTE\_ON &***

***\_WDT\_OFF & \_MCLRE\_OFF &***

***\_BODEN\_OFF & \_LVP\_OFF & \_INTRC\_OSC\_NOCLKOUT***

***CISLO equ 20h***

***CISLOA equ 21h***

***#DEFINE TLAC PORTA,0***

Takže začínáme více méně klasicky. Nejprve jsou informace pro programátor, poté se pojmenují používané registry a poté i samotné bity (TLAC zde sice není použito, ale později bude).

***org 00h***

***goto START***

V programu je použit takzvaný podprogram CEKEJ. Jde o to, že by jsme za každou LED museli dělat stejnou čekací smyčku. Tak se udělá jen jedna a ta se pak vždy zavolá příkazem *CALL CEKEJ*.

*Mimochodem: Z podprogramu se můžete dalším příkazem CALL dostat do dalšího podprogramu (hlouběji). Tento typ procesoru umožňuje až osminásobné vnoření. Z jednotlivých podprogramů se však taky musíte vrátit zpět příkazy RETURN.*

Podprogramy však musí být v programu uvedeny ze začátku, ještě před samotným programem, proto se tímto příkazem na první řádek programu (kde procesor začíná) vloží odkaz na záložku START, kde začíná vlastní program. Poté se uvedou všechny podprogramy a až po nich je vlastní program začínající právě záložkou START

***CEKEJ CLRF CISLO***

***MOVLW 150***

***MOVWF CISLOA***

***CEKEJ2 INCFSZ CISLO,1***

***GOTO CEKEJ2***

***DECFSZ CISLOA,1***

***GOTO CEKEJ2***

***RETURN***

Již zmiňovaný a v minulém díle popsáný podprogram na zpoždění mezi jednotlivými kroky. Změnou čísla 150 můžete změnit rychlost blikání.

***START MOVLW B'00000111'***

***MOVWF CMCON***

***BSF STATUS,RP0***

***MOVLW B'00000001'***

***MOVWF TRISA***

***CLRF TRISB***  
***BCF STATUS,RP0***

Zde začíná vlastní program. Nastaví se zde vstupy a výstupy.

***ZNOVU MOVLW B'00000001'***  
***MOVWF PORTB***  
***CALL CEKEJ***

***MOVLW B'00000010'***  
***MOVWF PORTB***  
***CALL CEKEJ***

No a tohle je již část programu, který mění rozsvícené LED diody. Celý ho nejdete v souboru s tímto programem. Čísla, ukládaná do PORTB jsou pro lepší přehlednost v binárním tvaru. Požadovaná hodnota se uloží do PORTB a zavolá se podprogram CEKEJ.

Zde je celý program ***Světelný had*** ke stažení:

[ASM](#)  
[HEX](#)

Samozřejmě si můžete libovolně měnit vkládané binární hodnoty, přidávat nebo ubírat počet kroků a tím měnit světelný efekt. Nevýhodou je nemožnost plynulé změny rychlosti blikání přímo za běhu. Co ale při programování místo interního oscilátoru nastavit externí RC člen (jeho vzhled je v lekci 5) a místo pevného odporu dát potenciometr.

### **Světelný had s tlačítkem**

V úvodu jsem se zmínil o tlačítku co je ve schématu uvedeno, ale není použito. Co ale do programu dát několik světelných efektů najednou a tím tlačítkem je postupně přepínat. To není nic těžkého jen se tlačítko musí snímat za každým "světelným" krokem.



Je zde tedy ke stažení další ukázkový program, který má v sobě dva světelné programy přepínajícíse stiskem tlačítka.

[ASM](#)

[HEX](#)

# Škola programování PIC 10

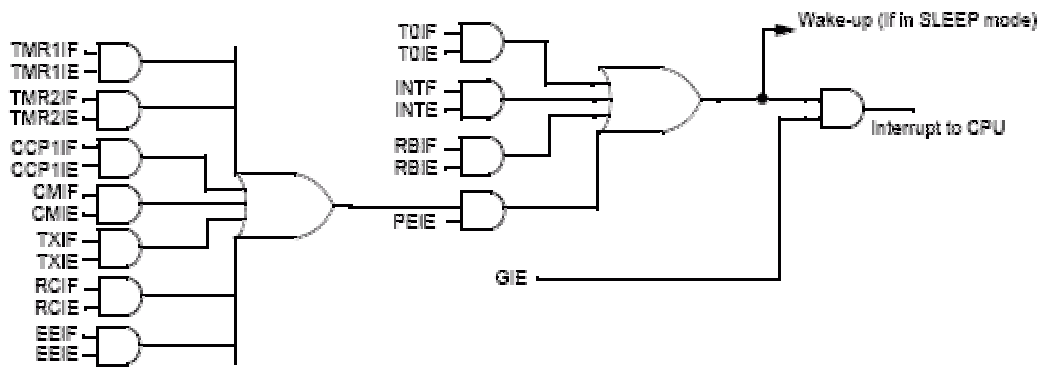
## Přerušení

Přerušení je funkce, která po předem nastavené akci přeruší aktuální činnost procesoru a vykoná danou práci. Poté se procesor vrátí zpět a pokračuje kde skončil. Jinými slovy procesor dokončí právě rozdělanou instrukci a skočí na 4 řádek programu! Zde se nachází rutina, která do pomocných registrů uloží pracovní registr W a další tři důležité registry - STATUS, PCLATH a FSR a skočí na podprogram obsluhy přerušení. Zde se obvykle zjistí jaký typ přerušení proběhl (pokud nepoužíváte jen jedno) a provede se příslušná akce. Při návratu z přerušení se opět obnoví uložené registry a příkazem RETFIE se procesor vrátí přesně do místa kde byl před přerušením a pokračuje zde dál.

Zdrojů přerušení je samozřejmě víc. Zapnou se jen ty, co jsou zrovna potřeba. Takže přerušení lze provést od:

- změny na pinu RB0/INT
- změny na pinech RB4 - RB7
- přetečení TMR0
- TMR1
- TMR2
- změny stavu komparátoru
- USART
- CCP

A vlastní systém přerušení vypadá takhle:



A k čemu je to vlastně dobrý? Nemusíte neustále hlídat stav těchto "věcí". Například máte na pinech RB4 až RB7 tlačítka, nemusíte se tedy neustále dotazovat na jejich stav. Jednoduše nastavíte příslušné přerušení a vše proběhne automaticky. Přerušení od přetečení některého z časovačů se používá na tvorbu přesných časů, atd.

Takže se podíváme na programovou obsluhu:

***W\_save equ 20h***

***Sta\_sav equ 21h***

***PCL\_sav equ 22h***

***FSR\_sav equ 23h***

V úvodu nastavíme názvy "záložních" registrů.

***org 00h***

***goto INIT***

***org 04h***

***movwf W\_save***

***movf STATUS,w***

***clrf STATUS***

***movwf Sta\_sav***

***movf PCLATH,w***

***movwf PCL\_sav***

***movf FSR,w***

***movwf FSR\_sav***

***GOTO PRERUS***

Těmito příkazy se na první řádek paměti s programem vloží skok na začátek vlastního programu a na 4 řádek se vloží rutina k přerušení. Touto rutinou, jak už bylo uvedeno v úvodu se do pomocných registrů uloží hodnoty nejdůležitějších registrů. Poté se provede skok na podprogram PRERUS.

***PRERUS BTFSC PIR1,0***

***GOTO PRET1***

***GOTO PRERKON***

***PRET1 BCF PIR1,0***

***NOP***

***GOTO PRERKON***

Zde se zjistí zdroj přerušení a vykonají se požadované příkazy (momentálně NOP).

***PRERKON BSF STATUS,RP0***

***MOVLW B'00000001'***

***MOVWF PIE1***

***MOVLW B'11000000'***

***MOVWF INTCON***

***BCF STATUS,RP0***

***clrf STATUS***

***movf FSR\_sav,w***

***movwf FSR***

***movf PCL\_sav,w***

***movwf PCLATH***

***movf Sta\_sav,w***

***movwf STATUS***

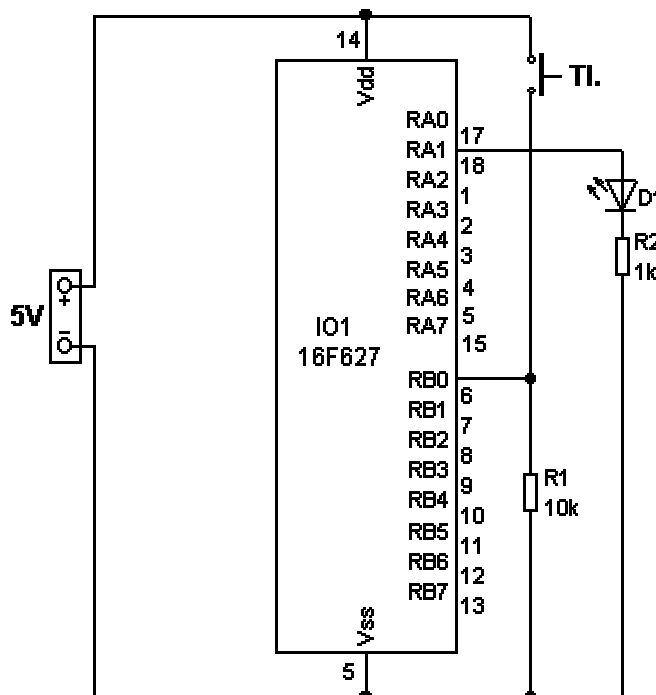
***swapf W\_save,f***

***swapf W\_save,w***

*retfie*

No a nakonec se musí znovu povolit požadovaná přerušení, tedy přesně těchto prvních sedm řádků je přítomno i v místě kde se nastavují I/O piny a podobně. Poté se obnoví zpět uložené registry a procesor se vrátí na původní adresu.

V následujícím jednoduchém příkladu je přerušení využito ke snímání stisku tlačítka. Tedy funkce jako v Prvním příkladu, jedním stiskem zapne, druhým vypne.



*Schéma zapojení*

Odkazy & Download:

[Preruseni.asm](#)

[Preruseni.hex](#)

## Škola programování PIC 11

### Časovače TMRx

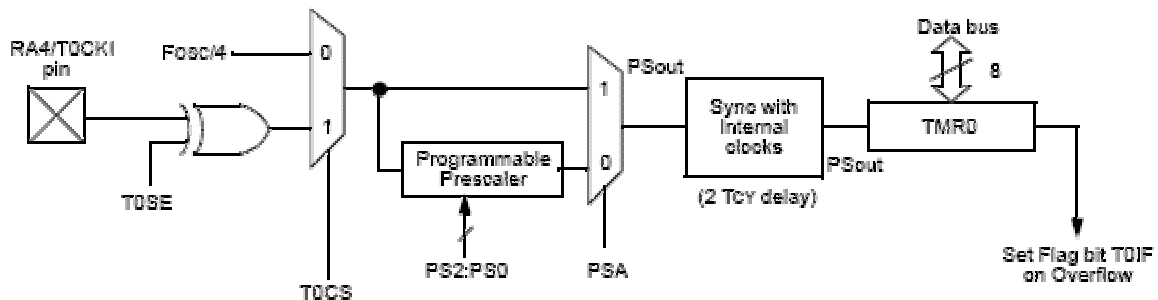
V předchozích dílech jsem se zmínil o třech implementovaných čítačích / časovačích, tak se na ně teď podíváme blíže.

**Základní vlastnosti:**

**TMR0**- 8 bitů

- možnost čtení i zápisu
- 8 bitů předdělička (společná pro Watchdog)
- zdroj signálu:

- a) interní OSC/4
- b) externí pin RA4

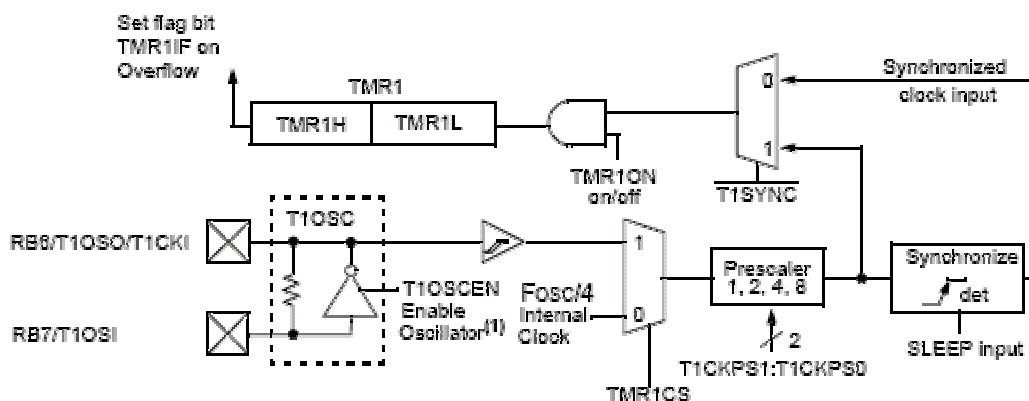


Note 1: Bits T0SE, T0CS, PS2, PS1, PS0 and PSA are located in the OPTION register.

2: The prescaler is shared with Watchdog Timer (Figure 6-8)

### TMR1- 16 bitů

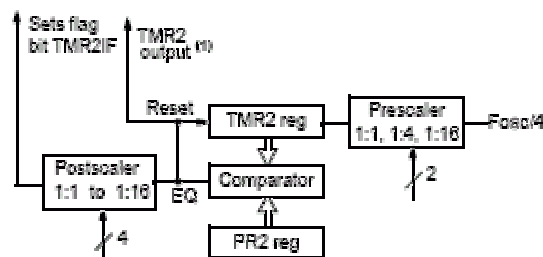
- možnost čtení i zápisu
- předdělička s dělicími poměry:
  - 1, 2, 4, 8
- zdroj signálu:
  - a) interní OSC/4
  - b) externí pin RB6
  - c) implementovaná oscilátor na pinech RB6, RB7 na max 200kHz



Note 1: When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

### TMR2- 8 bitů

- předdělička s dělicími poměry:
  - 1, 4, 16
- 4 bitová dělička na výstupu
- zdroj signálu pouze interní OSC/4
- možnost přednastavení
- není možné přímo číst ani zapisovat
- tato dělička je využívána i procesorem, při PWM



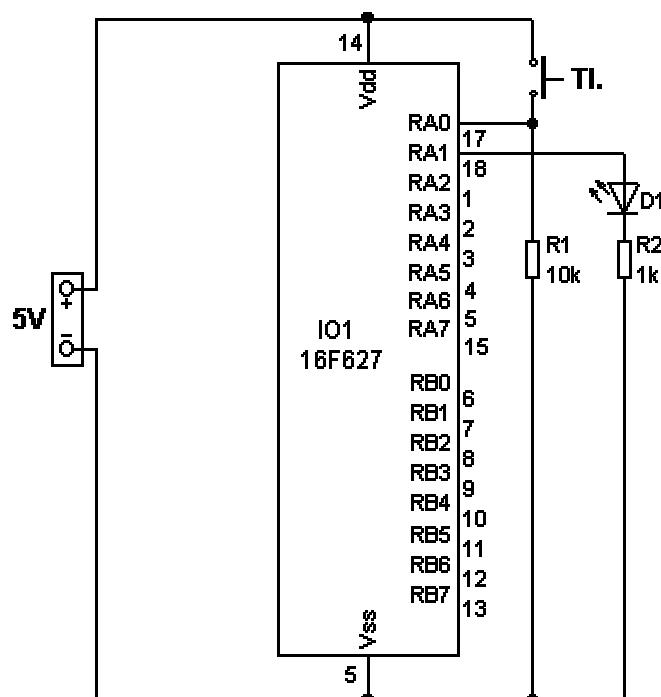
Note 1: TMR2 register output can be software selected by the SSP Module as a baud clock.

Všechny tři časovače nabízejí řadu různých nastavení, bohužel však není v mých silách zde vše do detailu popisovat, takže Vás opět odkazuji na příslušné katalogové listy (viz. 6. díl).

Zde ještě alespoň popíšu použití TMR1 jako zdroje 0,5 sekundových intervalů.

### Zdroj 0,5s

Opět zde využijeme naše klasické schéma zapojení, tentokrát nám však bude stačit jen ona LED.



Programem vytvoříme přerušení v intervalu 0,5s a při každém tomto přerušení změníme stav LED (on/off).

ASM

HEX

## Škola programování PIC 12

## Interní EEPROM

Paměť je možné libovolně číst i do ní zapisovat, nejprve si však ukážeme, jak do ní zapsat data už při programování procesoru.

[illegible][illegible]

***org 00h***

***goto INIT***

Takhle by tedy měl vypadat úvod programu, pokud chcete při programování procesoru již zapsat data do EEPROM. Řádky samozřejmě nemusí být všechny.

***CTIEE movf EADR,W***

***bsf STATUS,RP0***

***movwf EEADR***

***bsf EECON1,RD***

***movf EEDATA,W***

***bcf STATUS,RP0***

***return***

Tento podprogram CTIEE slouží ke čtení dat z EEPROM. V úvodu celého programu je ještě třeba vytvořit registr EADR.

Obsluha podprogramu je velice jednoduchá, do registru EADR vložíte adresu ze které chcete číst a poté příkazem ***CALL CTIEE*** zavoláte tento podprogram. Ten na určené adrese přečte hodnotu, uloží ji do pracovního registru a skončí.

***PISEE bsf STATUS,RP0***

***btfsc EECON1,WR***

***goto PISEE***

***bcf STATUS, RP0***

***movf EADR,0***

***bsf STATUS,RP0***

***movwf EEADR***

***bcf STATUS,RP0***

***movf CISLO,0***

***bsf STATUS,RP0***

***movwf EEDATA***

***bsf EECON1,WREN***

***bcf INTCON,GIE***

***movlw H'55'***



```
movwf EECON2  
movlw H'AA'  
movwf EECON2  
bsf EECON1,WR  
bcf EECON1,WREN  
bsf INTCON,GIE  
bcf STATUS,RP0  
return
```

Podprogram pro zápis dat je již trochu složitější. Nejprve je třeba si vysvětlit, že se jedná o paměť typu EEPROM, tedy paměť u které zápis trvá i několik ms. Proto je v úvodu podprogramu čekací smyčka, které testuje zda již skončil případný předchozí zápis. Za ní následuje přesun hodnot z registru EADR (adresa) a CISLO (zapisovaná hodnota) do systémových registrů. Poté se zakáže všechna přerušení a dvěma po sobě jdoucími "klíči" (55h a Hh) se odemkne zápis do paměti. Pak se již povolí zápis a je hotovo. Nakonec se ještě zpět povolí všechna přerušení a je konec podprogramu. Zápis dat pak již proběhne automaticky.

Po skončení zápisu se doporučuje provést kontrolní čtení, ale není to třeba.

Zde je ke stažení ukázkový program se všemi těmito ukázkami:

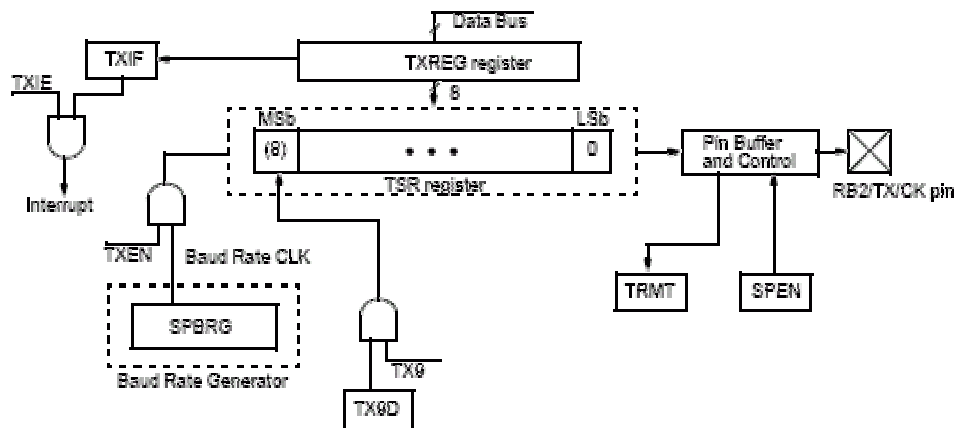
[ASM](#)

[HEX](#)

## **Škola programování PIC 13**

### **USART - sériový port**

V dnešním díle se podíváme na jednotku USART, konkrétně na její využití pro Asynchronní sériový přenos, používaný například u PC jako sériový port - RS232.



*Blokové zapojení vnitřní jednotky USART*

Na kompletní popis a nastavení vás opět odkazují do katalogových listů (viz. 6 lekce). Zde popíšu nastavení při použití interního oscilátoru, nebo i jiného zdroje 4MHz. Rychlost komunikace bude nastavena na 19,2 kbaud, 8 bitů a bez parity.

### **Piny jsou:**

RxD - RORTB,1

TxD - RORTB,2

***MOVLW B'00100110'***

***MOVWF TXSTA***

***MOVLW 12***

***MOVWF SPBRG***

***MOVLW B'00100000'***

***MOVWF PIE1***

***BCF STATUS,RP0***

***MOVLW B'10010000'***

***MOVWF RCSTA***

***MOVLW B'11000000'***

***MOVWF INTCON***

Nejprve je potřeba provést základní nastavení. Nastaví se způsob a rychlost přenosu a povolí se přerušení po přijmutí bytu. Je to výhodnější, protože jinak by jsme neustále museli testovat zda něco nepřišlo, což by při vysokých přenosových rychlostech bylo téměř nemožné.

***MOVLW 48***

***MOVWF TXREG***

Takhle vypadá odeslání jednoho bytu (znaku), konkrétně nuly (viz ASCII tabulka). Stačí ho uložit do registru TXREG a o nic víc se nemusíme starat, vše proběhne automaticky. Pokud by jste chtěli odesílat víc znaků najednou, je třeba ještě testovat zda byl již minulý byt odeslán. To se provádí testováním bitu TXSTA,1, pokud je vysílač v klidu, je v jedničce.

***MOVF RCREG,0***

***MOVWF CISLO***

No a takhle vypadá sekvence na přečtení přijmutého bytu. Samozřejmě ji musí předcházet zjištění že něco nepřišlo, viz. celý příklad. Přijmutý byte je uložen v registru RCREG, ze kterého ho přečteme a uložíme do registru CISLO, případně provedeme jinou operaci.

Zde je ke stažení ukázkový program se všemi těmito ukázkami:

[ASM](#)

[HEX](#)