INTEGROVANÁ STŘEDNÍ ŠKOLA
CENTRUM ODBORNÉ PŘÍPRAVY
VALAŠSKÉ MEZIŘÍČÍ

Mikrořadiče

Učebnice
pro odborný výcvik

–
2008

Učebnice obsahuje nejdůležitější informace o problematice mikrořadičů se zaměřením na typovou řadu PIC18FXX2 a a stručný popis vývojového systému MPLAB ICD2. Učebnice je zaměřen spíše na praktickou stránku tak, aby absolventi byli schopni použít mikrořadič s několika jednoduchými periferními obvody, napsat a odladit několik jednodušších programů a porozumět práci s vývojovým prostředím MPLAB ICD2.

Učebnice je určen pro třetí a čtvrtý ročník čtyřletých učebních oborů elektrotechnických. Rozsah učiva této učebnice je dán počtem hodin, které jsou pro výuku mikrořadičů v těchto ročnících k dispozici. V praxi to znamená, že budou popsány pouze základní funkce mikrořadiče řady PIC18FXX2, potřebné pro zvládnutí učiva v předepsaném rozsahu. Ostatní funkce (A/D převodníky, sériové porty, analogové komparátory, práce s pamětí EEPROM …) budou popsány pouze okrajově a nebudou k nim k dispozici cvičení a ukázkové programy.

Obsah: **Strana**

1. Co je to mikrořadič?
   Srovnání mikrořadiče s mikroprocesorem

2. Vnitřní struktura mikrořadiče
   Blokové schéma PIC16F873
   Paměť programu (FLASH)
   Datová paměť (RAM), banky
      Oblast speciálních funkčních registrů, pracovní registr W
      Oblast všeobecně použitelných registrů
      Paměť EEPROM
      Porty
         PORTA, TRISA, LATA
         PORTB, TRISB, LATB
         PORTC, TRISC, LATC
      Sériové porty
         Modul MSSP
         Modul USART
   Čítače, časovače
   Moduly CAPTURE, COMPARE, PWM
   A/D převodník
   Hardwarová násobička
   Systém přerušení
      Princip
      Zdroje přerušení, priority
      Příklady využití

Mikrořadič jako součástka
   Základní zapojení mikrořadiče, základní podmínky pro správnou funkci
      Připojení krystalu
      Připojení jednoduchých periferií, zatížitelnost portů
         Připojení tlačítka
         Připojení LED diody
         Připojení spínacího tranzistoru a relé
         Připojení LED displeje (přímý a multipexní režim, výhody a nevýhody)

Software a vývojové prostředí
   Co je to program?
   Jak začít s programováním?
   Vývojový diagram
   Co je to vývojové prostředí?
   Vývojové prostředky a jejich možnosti
Srovnání jednotlivých vývojových prostředí

Assembler - jazyk symbolických adres
      Způsob zápisu programu
      Instrukce, parametry, symbolické adresy, návěští, poznámky
      Překladač, základní directiva

Co je to mikrořadič?

Mikrořadič patří do skupiny programovatelných logických obvodů. Je to tedy obvod, jehož činnost není pevně daná jeho vnitřní strukturou, nýbrž je řízena programem, uloženým v jeho vnitřní paměti. Jaké to má výhody?

Představme si následující situaci: kamarád nás požádá, abychom mu sestavili digitální hodiny. Žádný problém, stačí si sehnat patřičné schéma, nakoupit součástky, navrhnout a vyrobit plošný spoj, osadit jej součástkami, oživit, vše umístit do patřičné krabičky a je hotovo. Pak si však kamarád vzpomene, že by součástí těchto hodin mohl být i budík, na který v původním zadání zapomněl. Dobře tedy, dokreslíme do schématu obvod pro buzení, přikoupíme součástky, navrhneme znovu plošný spoj, celý jej znovu osadíme, oživíme a jsme hotovi. Jak snadno se to řekne! Je však třeba si uvědomit, že prakticky celou práci děláme znovu.

Pokud však realizujeme tytéž hodiny pomocí mikrořadiče, celá funkce je řízena programem. Pokud tedy vyvstane nutnost cokoliv dodělat, přidat novou funkci, popřípadě opravit chyby, které se občas při každé práci vyskytnou, stačí pouze dopsat nebo opravit několik řádků programu, bez nutnosti zásahu do hardware.

Možná, že v tuto chvíli bychom se měli alespoň krátce zmínit o tom, co je to vlastně program. Je to vlastně posloupnost jednotlivých povelů (instrukcí), které zadává mikrořadiči člověk (programátor), kterým mikrořadič rozumí a které postupně, jednu po druhé vykonává. Lze tedy říci, že vhodným sestavením jednotlivých instrukcí docílíme toho, že elektronické zařízení dělá přesně to, co po něm požadujeme. Je velmi pravděpodobné, že stejné zařízení by šlo sestavit z klasických obvodových prvků (rezistory, kondenzátory, tranzistory, logické obvody atd.), avšak u takového zařízení je jeho funkce pevně dána obvodovým zapojením, kdežto při řešení s mikrořadičem je většinou možno funkci zařízení velmi jednoduše měnit pouhým přepsáním části programu, aniž by bylo nutno zasahovat do elektrického schématu a následně do plošného spoje.

Další velmi pádný argument pro používání mikrořadičů si uvědomíme ve chvíli, kdy si vedle sebe položíme elektrická schémata zařízení navržených „klasickou“ technikou a zařízení řízených mikrořadičem. Na první pohled vidíme značný rozdíl ve složitosti, v počtu součástek potřebných pro realizaci. Zařízení s mikrořadičem si vystačí mnohdy doslova s několika málo součástkami, což znamená mnohem vyšší spolehlivost, nižší příkon a v neposlední řadě nižší cenu.

Tyto výhody měly za následek obrovské rozšíření mikrořadičů v elektronických zařízeních. Rozhlédneme-li se kolem sebe, nalezneme mikrořadiče např. v automatických pračkách, mikrovlnných troubách, televizorech, videorekordérech a jejich dálkových ovladačích, kamerách, fotoaparátech, herních automatech a v mnoha jiných přístrojích spotřební elektroniky. Uplatnění našly i v automobilech, alarmech a zabezpečovacích systémech, měřicích a diagnostických systémech. Obrovskou oblast využití představuje vojenská technika, kde se mikrořadiče uplatňují především v navigačních systémech letadel, vrtulníků a zbraňových systémů.

Dá se tedy říci, že si dnes moderní elektroniku bez mikrořadičů prakticky nedovedeme představit.

Srovnání mikrořadiče s mikroprocesorem

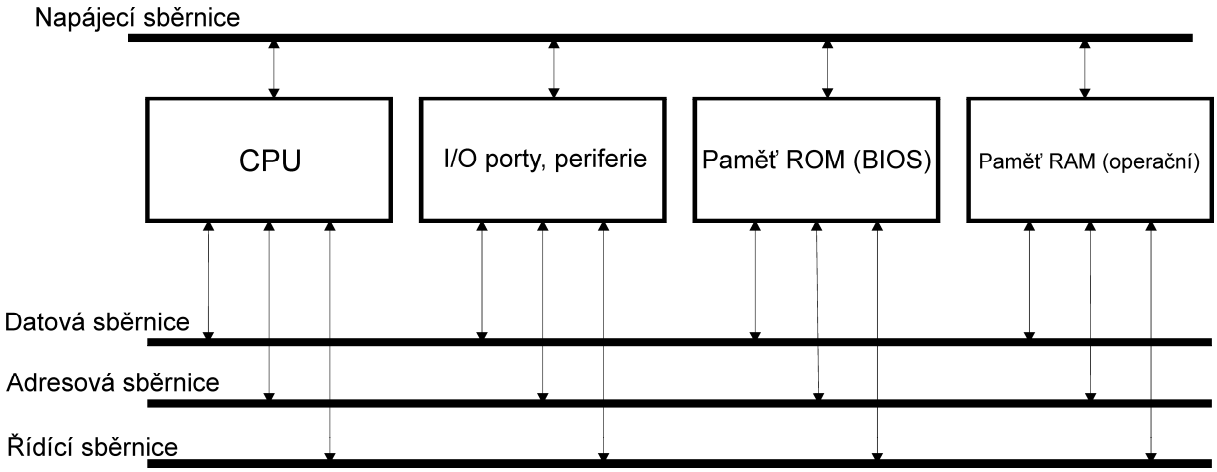
V úvodu jsme si řekli, že činnost mikrořadičů je řízena programem, uloženým v elektronické paměti. Nepřipomíná to něco? Samozřejmě mám na mysli věc, kterou dobře známe, mikroprocesor, srdce každého počítače, obvod, jehož činnost je rovněž řízena

programem. Je to tedy totéž? Odpověď zní ne, není to totéž. Oba obvody jsou si sice hodně podobné, princip činnosti je vlastně stejný, ale přece jen se od sebe značně liší. Protože se však srovnání opravdu nabízí, zkusme se podívat na to, čím se od sebe liší.

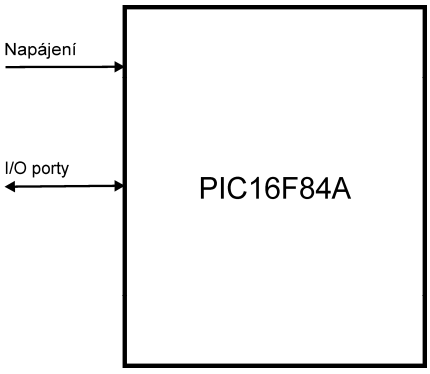
Činnost „klasického“ mikroprocesoru (CPU) se často definuje takto:

Mikroprocesor je logický automat, který provádí operace s daty podle určitého programu. Program i data jsou vždy uloženy v operační paměti typu RAM. Totéž by se však dalo říci i o běžném mikrořadiči. V čem se tedy liší?

Srovnejme si bloková schémata funkčních počítačů řešených s mikroprocesorem a mikrořadičem:



Obr. 1: Blokové schéma funkčního počítače s "klasickým" mikroprocesorem



Obr. 2: Blokové schéma funkčního počítače s mikrořadičem.

Na první pohled vidíme velký rozdíl ve složitosti obou zařízení. Tam, kde mikroprocesor potřebuje ke své činnosti spoustu dalších obvodů (paměť ROM, RAM, vstupní a výstupní porty, sběrnice a jejich řadiče, obvody hodinového kmitočtu atd.), mikrořadič si vystačí pouze s napájecím zdrojem, protože obvod hodinového kmitočtu, paměti, porty a všechny ostatní pomocné obvody již sám obsahuje.

Vždy je však „něco za něco“. Z následující tabulky jsou patrny rozdíly mezi oběma obvody:

	Mikroprocesor	Mikrořadič16F877A
Operační paměť	standard cca 512MB, společná pro program i data	32kB program 1536 byte data 256 byte konstanty (EEPROM)
Počet instrukcí	řádově stovky	77
Počet vývodů pouzdra	Cca 100 – 200	28 (6 – 84 u ostatních typů)
Periferie	omezeno pouze adresovacím prostorem pro I/O	3 porty = 23 I/O linek (některé typy této řady až 70 I/O linek)

Z toho vyplývají i značné rozdíly v provedení a využití těchto součástek.

Mikroprocesor je zaměřen co nejvíce univerzálně, z toho vyplývá:
- nutnost adresovat co největší adresový prostor (požadavek hardware i software)
- nutnost adresovat externí periferie (sběrnice PCI, PS-/2, USB, Fire-Wire)
- rozsáhlá instrukční sada (vyplývá z požadavku na co největší univerzálnost)
- spolupráce s výkonnými grafickými adaptéry (AGP sběrnice)
- externí paměť RAM i ROM (RAM paměť je společná pro program i data)
- používá společnou sběrnici pro instrukce i data
- externí I/O porty
- potřebuje externí zdroj hodinového kmitočtu a obvod pro reset
- jeden výrobce vyrábí v danou chvíli relativně málo typů procesorů
- vysoká cena

Výsledkem je pouzdro o velkém počtu vývodů (řádově stovky), s odběrem proudu několik ampérů.

Využití

Klasický mikroprocesor = z 90% osobní počítač pro obrovskou oblast využití (zpracování textů, účetnictví, kancelářské aplikace, technické konstrukce, grafika, řízení průmyslových procesů, vědecko-technické výpočty, multimedia (hudba, filmy, hry…).

Mikrořadič je zaměřen úzce účelově, z toho vyplývá:
- každý výrobce nabízí mnoho různých provedení v jedné typové řadě, s poměrně malými odlišnostmi u jednotlivých typů. Konstruktér si tak může zvolit typ, který nejlépe vyhovuje danému použití. Pro ilustraci - v květnu 2005 nabízela jen firma Microchip 273 typů mikrořadičů!
- oddělené sběrnice pro instrukce a data (vyšší výkonnost)
- interní oddělené paměti programu, dat a konstant. Většina typů má paměti typu FLASH a EEPROM, což umožňuje uchování programu a dat i po vypnutí napájecího napětí obvodu a usnadňuje programování paměti přímo v aplikaci.
- interní I/O porty
- interní zdroj hodinového kmitočtu
- interní několikaúrovňový reset opět zjednodušuje konstrukci

Vidíme tedy, že u mikrořadičů není důvod, aby byly interní sběrnice vyvedeny mimo pouzdro, z čehož vyplývá relativně malý počet vývodů pouzdra.Výsledkem je velmi nízká cena a tedy i dostupnost mikrořadičů.

Využití
- jednoúčelové automaty (zabezpečovací a hlídací systémy)
- řízení průmyslových robotů a jednoúčelových automatů
- ovládání přístrojů spotřební elektroniky
- elektronika v domácnosti (digitální hodiny, časovače, mikrovlnky, pračky, myčky nádobí, chladničky, programátory pro řízení topení, osvětlení, klimatizace)
- elektronika v autě (hlídací a indikační systém, alarm)
- mikročipy pro evidenci čehokoliv, čipové karty a jejich snímače
- vojenská technika

Pokud bychom si tedy chtěli na závěr této kapitoly odpovědět na otázku, kterou jsme si položili na jejím počátku, odpověď by nejspíše zněla takto:

Mikrořadič má všechny atributy jednoduchého počítače, je to tedy jednoduchý počítač. Je převážně určen k řízení jednoúčelových strojů a zařízení, kde díky své specializaci a programovému řízení podstatně zjednodušuje a tedy i zlevňuje řídící elektroniku, zvyšuje spolehlivost, snižuje nároky na napájení a umožňuje flexibilně reagovat na potřeby drobných úprav funkce zařízení. Protože výrobci nabízejí velké množství typů mikrořadičů, lišících se mnohdy jen detaily, má konstruktér možnost volit takový typ, který přesně vyhovuje jeho požadavkům a je tedy i maximálně využit.

Mikrořadič PIC16F873

V naší výuce se zaměříme na mikrořadič PIC16F873. Tento typ je jedním z mnoha zástupců výrobků americké firmy Microchip. Patří do rodiny obvodů PIC16F87x. Přehled jejich vlastností je v následující tabulce:

PIC16F87X

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

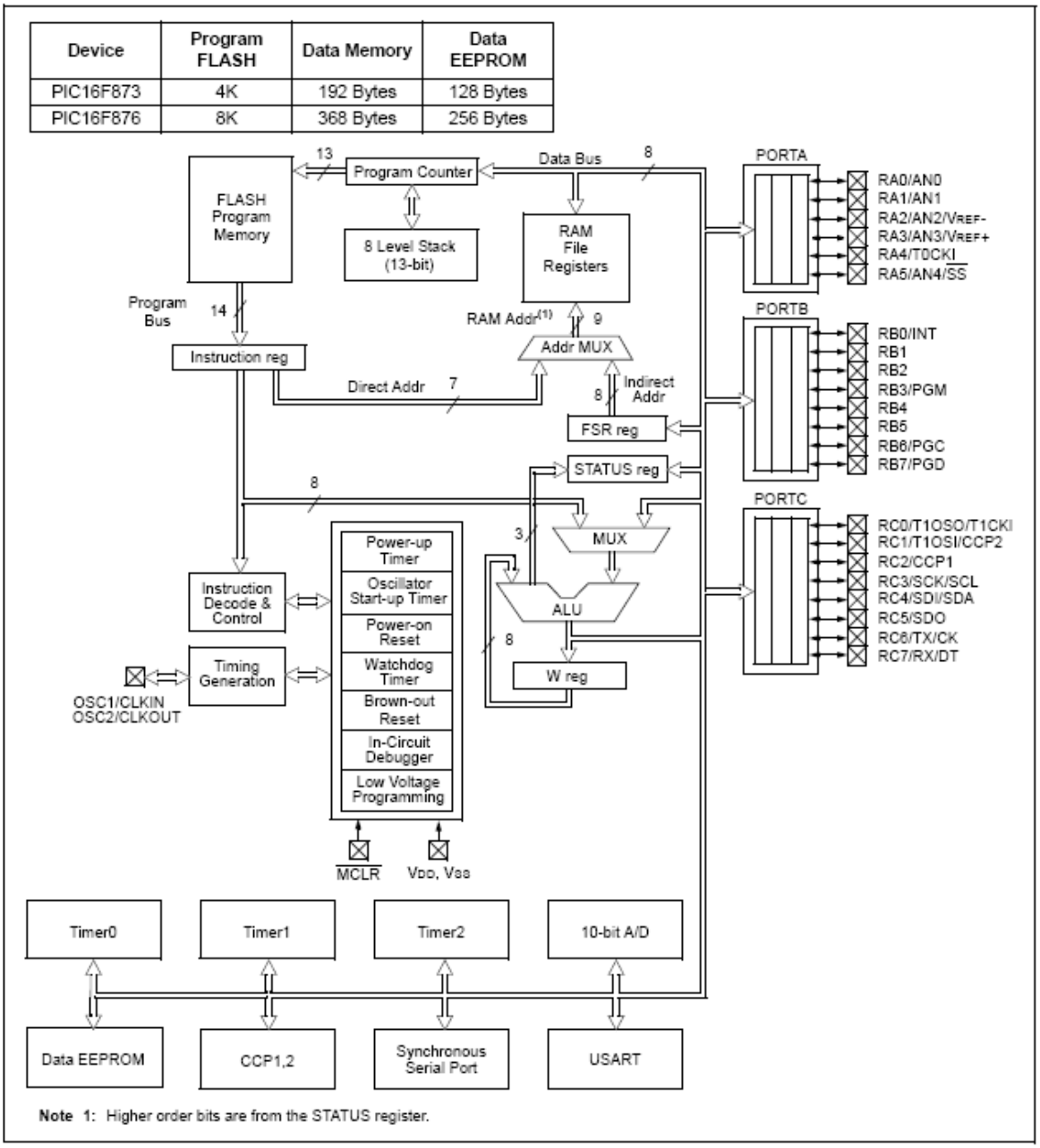
Abychom pochopili funkci tohoto obvodu, musíme se nejprve seznámit s jeho vnitřní strukturou.

Vnitřní struktura PIC16F873

Mikrořadič PIC16F873 obsahuje tyto základní bloky:
Oscilátor (nastavitelný na 8 různých režimů činnosti)
Paměť (RAM, EEPROM - paměť pro uložení programu a zpracovávaných dat)
GPR (General Purpose Registers – všeobecně použitelné registry)
SFR (Special Functions Registers – registry pro speciální použití)
W registr (Work Register - pracovní registr)
PORTA, PORTB, PORTC (porty - zprostředkovávají styk s „okolním světem“)
ALU (Arithmetical/Logical Unit – aritmeticko/logická jednotka)
PC (Program Counter – čítač programu)
STACK (registr návratových adres přerušení)
Čítače/časovače
Moduly Capture/Compare/PWM (porovnávací moduly, generátor PWM signálu)
Dva sériové komunikační porty (MSSP, USART)
10-bitový, 8-kanálový A/D převodník
Watchdog Timer

FIGURE 1-1: PIC16F873 AND PIC16F876 BLOCK DIAGRAM

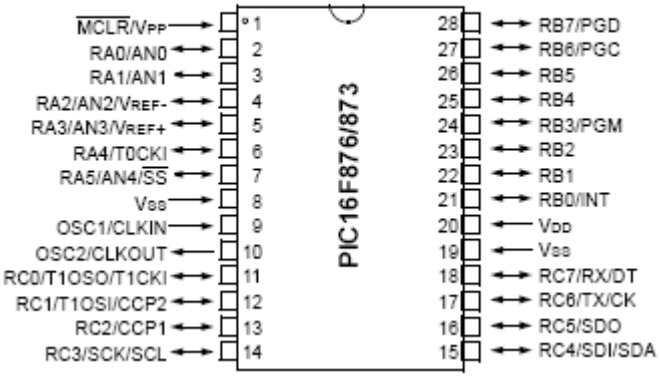
Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

Mikrořadič PIC16F873 se dodává v pouzdrech typu DIP a SOIC. Rozložení vývodů těchto pouzder je na následujícím obrázku:

PDIP, SOIC



Paměť

Paměť je jedním ze stěžejních prvků každého počítače. V paměti je uložen program a ukládají se do ní data, potřebná ke správné činnosti programu.

Mikrořadiče PIC využívají tzv. Harvardskou architekturu, která využívá oddělených pamětí pro program a data. Výhoda rozdělení paměti spočívá v možnosti použití různých typů pamětí ale především různé bitové šířky obou pamětí. Hlavní výhodou však je možnost nezávislého přenosu instrukce a datového slova po samostatných sběrnicích v jediném strojovém cyklu, což značně zjednodušuje řízení a urychluje běh programu.

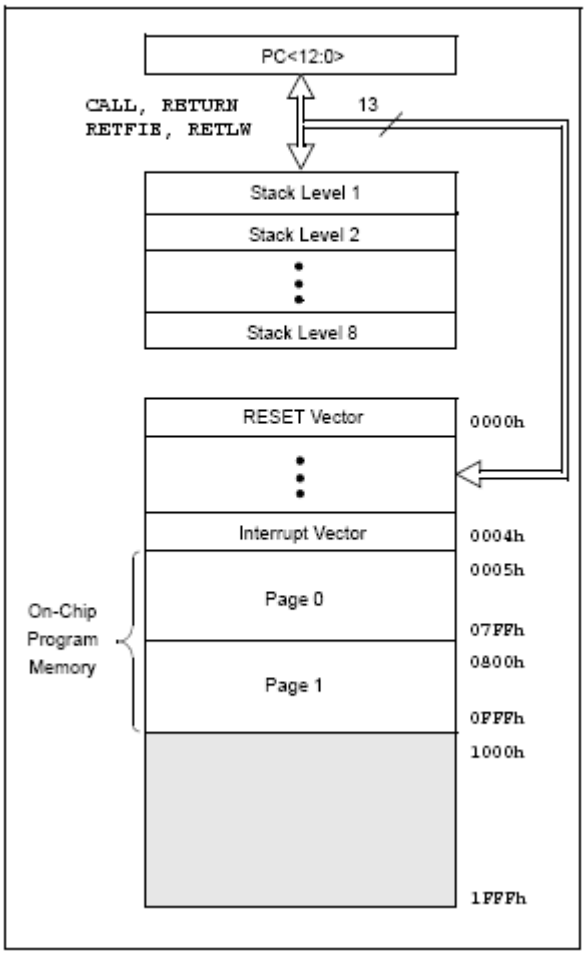
Paměť programu je typu FLASH, má kapacitu 4 kB a je tvořena polem paměťových buněk o šíři 14 bitů. Program se do ní ukládá pomocí speciálního programátoru (u vyšších typů mikrořadičů je do ní možný i zápis programem).

Paměť dat je typu RAM, má kapacitu 192 Byte a je tvořena polem paměťových buněk o šíři 8 bitů.

Paměť konstant je typu EEPROM, má kapacitu 128 Byte a je tvořena polem paměťových buněk o šíři 8 bitů. Slouží k trvalému uchování konstant (předvolby, poslední známý stav dat před vypnutím …).

Paměť programu.

FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



Paměť programu začíná na adrese 0000h (RESET Vector) a končí na adrese 0FFFh.

Adresa 0004h se nazývá Interrupt Vector a slouží k obsluze přerušení.

Oblast adres 1000h až 1FFFh leží mimo oblast fyzické paměti programu. Výsledkem čtení této oblasti paměti je vždy nula.

Čítač programu (PC – Program Counter) a registr STACK nepatří do paměti programu (i když jsou s ní zakresleny společně v jednom obrázku).

PC je speciální registr, který vždy obsahuje adresu paměti programu, ze které bude vzápětí přečtena následující instrukce. Je to tedy jakési „ukazovátko“ na aktuální pozici programové paměti. Obsah čítače programu je zvýšen o 1 v každém instrukčním cyklu. Výjimku tvoří instrukce podmíněných a nepodmíněných skoků, volání a návratů z podprogramu a instrukce PUSH a POP.

Registr STACK slouží k uchování tzv. návratových adres při volání podprogramu a rovněž pro dočasné uchování obsahu PC pomocí instrukcí PUSH a POP.

Paměť dat.

Datová paměť je typu SRAM (statická RAM). Jedna buňka této paměti se v terminologii mikrořadičů označuje jako **registr**. Toto označení se používá pro všechny buňky paměti dat, bez ohledu na to, zda se jedná o registry SFR nebo o uživatelskou oblast.

Organizace datové paměti je poněkud složitější, než tomu bylo u paměti programu. Podívejme se tedy na její strukturu:

FIGURE 2-4: PIC16F874/873 REGISTER FILE MAP

File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h		185h	
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h		187h	
08h	PORTD(1)	88h	TRISD(1)	108h		188h	
09h	PORTE(1)	89h	TRISE(1)	109h		189h	
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh		10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h		110h		190h	
11h	TMR2	91h	SSPCON2				
12h	T2CON	92h	PR2				
13h	SSPBUF	93h	SSPADD				
14h	SSPCON	94h	SSPSTAT				
15h	CCPR1L	95h					
16h	CCPR1H	96h					
17h	CCP1CON	97h					
18h	RCSTA	98h	TXSTA				
19h	TXREG	99h	SPBRG				
1Ah	RCREG	9Ah					
1Bh	CCPR2L	9Bh					
1Ch	CCPR2H	9Ch					
1Dh	CCP2CON	9Dh					
1Eh	ADRESH	9Eh	ADRESL				
1Fh	ADCON0	9Fh	ADCON1				
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 96 Bytes	120h	accesses 20h-7Fh	1A0h	accesses A0h - FFh
7Fh		FFh		16Fh 170h		1EFh 1F0h	
				17Fh		1FFh	

Unimplemented data memory locations, read as '0'.
* Not a physical register.
Note 1: These registers are not implemented on the PIC16F873.
2: These registers are reserved, maintain these registers clear.

Oblast datové paměti je rozdělena do 4 bank po 128 Byte. Volba konkrétní banky se děje zápisem do bitů RP0, R P1 registru ***STATUS*** (viz popis registru STATUS).

Datová paměť obsahuje dva druhy registrů: ***Special Function Registers (SFR)*** a

General Purpose Registers (GPR).

SFR jsou ***registry pro speciální využití*** a slouží pro řízení činnosti ***CPU*** mikrořadiče a jeho periferií. Zápisem do těchto registrů můžeme v jistých mezích činnost mikrořadiče ovlivňovat, jejich čtením naopak zjišťovat stav některých vnitřních zařízení a výsledky aritmetických a logických operací.

Přehled všech registrů mikrořadiče PIC16F873 je uveden následující tabulce: Jejich podrobný popis je však mimo rámec této učebnice a je třeba jej vyhledat v katalogových listech obvodu, nejlépe na stránkách výrobce: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2046&redirects=datasheets.

GPR jsou naopak ***registry pro obecné využití*** a slouží uživateli jako univerzální „odkládací schránky“ pro libovolná data během činnosti programu (je to vlastně paměť RAM, známá z běžných počítačů). U Banky 0 začínají na adrese 20h a končí na adrese 7Fh, u Banky 1 začínají na adrese A0h a končí na adrese FFh. Banky 2 a 3 kopírují banky 1 a 2, jak je patrno z tabulky.

Tato oblast datové paměti se využívá hlavně pro definici tzv. ***uživatelských registrů***, které uživatel (programátor) potřebuje k dočasnému ukládání svých proměnných nebo konstant. Typickým příkladem jsou proměnné pro čítače časových smyček, registry pro dočasné ukládání dat, načtených ze vstupních portů nebo naopak dat, určených pro porty výstupní, registry pro rotace při převodu dat z paralelního tvaru na sériový a naopak, ale taky nejrůznější pomocné proměnné, které je dobré mít po ruce kdykoliv, kdy je zapotřebí si „někde něco odložit a po chvíli to tam opět najít“.

Tyto ***uživatelské registry*** je zapotřebí na začátku programu nadefinovat pomocí directiva ***EQU*** (o tzv. ***directivech*** překladače si povíme něco málo v kapitole o programování).

Speciální funkční registry (SFR)

Podrobně popsat všechny SFR a jejich jednotlivé bity se všemi jejich funkcemi je nad rámec této učebnice. Množství SFR závisí u jednotlivých typů mikrořadičů na jejich vnitřní „vybavenosti“. Srovnáme-li např. typy PIC10F200 a PIC16F873, vidíme na první pohled velký rozdíl právě v počtu SFR. Je to dáno tím, že SFR slouží k podpoře hardwarových modulů uvnitř mikrořadiče. Obsahuje-li určitý mikrořadič např. modul A/D převodníku, je nutno, aby obsahoval rovněž SFR, jehož jednotlivé bity umožní nastavení parametrů převodu, ale současně dávají i přehled o tom, zda je již převod ukončen nebo zda stále probíhá. Je proto logické, že např. u typu PIC10F200, který A/D převodník neobsahuje, tento registr nenajdeme.

Pokud se někdo při pohledu na množství těchto registrů a jejich bitů zalekne (když si uvědomíme, že se navíc většinou pracuje s jejich jednotlivými bity, tak je to docela pochopitelné), rád bych jej ubezpečil, že si zpočátku pro pochopení práce vystačíme s několika málo registry. Jednou z výhod práce s mikrořadiči je mimo jiné i to, že lze začít jednoduchými úlohami, z nichž některé nevyžadují prakticky žádné SFR, postupně pak složitost zvyšovat a nové poznatky stavět na tom, co již známe a ovládáme.

Všechny SFR jsou osmibitové. U většiny z nich se však pracuje s jejich jednotlivými bity. Tyto bity většinou vyjadřují stav určité probíhající operace a proto se jim říká ***stavové bity***.

Jako příklad si uvedeme dva nejpoužívanější speciální funkční registry: registr STATUS a porty.

STATUS

Tento registr obsahuje informace o stavu aritmetických a logických operací ALU. Každý jeho bit má určitou specifickou funkci. Nás budou prozatím zajímat pouze tyto bity:

C: tento bit je nastaven při přetečení nebo podtečení obsahu registru.

Z: tento bit je nastaven, když se výsledek aritmetické nebo logické operace rovná nule.

RP0, RP1: slouží k volbě banky registrů.

Pozice jednotlivých stavových bitů v registru STATUS a jejich význam jsou patrné z tabulky:

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 Z: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 DC: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 C: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Pozn.: Řekneme-li, že je bit **nastaven**, znamená to, že má hodnotu 1. Naopak, je-li **nulován**, má hodnotu 0.
Přetečením obsahu registru se myslí stav, kdy obsah registru přechází ze stavu FFh do stavu 00h.
Podtečením obsahu registru se myslí stav, kdy obsah registru přechází ze stavu 00h do stavu FFh.

PORTY

Porty patří rovněž mezi SFR (jsou umístěny do stejné oblasti paměti). Jejich jednotlivé bity jsou však vyvedeny na vývody pouzdra a zprostředkovávají tak styk mikrořadiče s „okolním světem". Jsou to právě ony, které můžeme použít ke čtení stavu vstupních zařízení (tlačítka, čidla, snímače …) nebo k předávání dat na výstupní zařízení (LED diody, znakové a grafické displeje, spínací prvky, motory …).

Mikrořadič PIC16F873 má tři porty, označené **PORTA**, **PORTB** a **PORTC**.

PORTA má šířku 6 bitů, PORTB a PORTC 8 bitů. Celkem tedy má PIC16F873 dvacet dva samostatných vstupnních/výstupních „vodičů“.

TABLE 3-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2	bit2	TTL	Input/output or analog input.
RA3/AN3/VREF	bit3	TTL	Input/output or analog input or VREF.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/SS/AN4	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input.

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 3-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT	bit0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3/PGM(3)	bit3	TTL	Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6/PGC	bit6	TTL/ST(2)	Input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST(2)	Input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: Low Voltage ICSP Programming (LVP) is enabled by default, which disables the RB3 I/O function. LVP must be disabled to enable RB3 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

TABLE 3-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin or USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin or USART Asynchronous Receive or Synchronous Data.

Legend: ST = Schmitt Trigger input

Žádný počítač (a tedy ani mikrořadič) při práci nevystačí pouze s porty, ale potřebuje i jiné druhy signálů, např. vstupy pro přerušení, vstup pro interní čítač, pro připojení externího krystalu hodinového kmitočku apod. Protože prvořadou snahou výrobce mikrořadičů je malý počet vývodů pouzdra, jsou téměř všechny bity jednotlivých portů využity vícekrát – mají více funkcí. Tyto funkce se dají programově nastavovat a to právě zápisem do odpovídajících SFR.

Velmi důležitou vlastností portů je to, že jejich jednotlivé bity je možno kdykoliv, a tedy i za běhu programu, programovat tak, aby se chovaly buď jako vstupy nebo jako výstupy. Abychom byli schopni této vlastnosti využít, musíme mít možnost kdykoliv tento směr toku dat určit. K tomu slouží tři „pomocné“ registry TRISA, TRISB a TRISC.

Tyto registry jsou pevně „spjaty“ se svými porty a to tak, že registr TRISA patří PORTu A, TRISB PORTu B a TRISC PORTu C. Informaci o požadovaném směru toku dat jednotlivými bity portů nich ukládáme takto:

Vložení hodnoty „1“ do určitého bitu registru TRIS má za následek nastavení odpovídajícího bitu „spřaženého“ portu do funkce vstupu.
Vložení hodnoty „0“ do určitého bitu registru TRIS má za následek nastavení odpovídajícího bitu „spřaženého“ portu do funkce výstupu.

Příklad: Jestliže do registru TRISB vložíme hodnotu 01100100, pak se nastaví bity 2, 5 a 6 PORTu B do funkce vstupu a bity 0, 1, 3, 4, a 7 do funkce výstupu, jak je to znázorněno na následujícím obrázku:

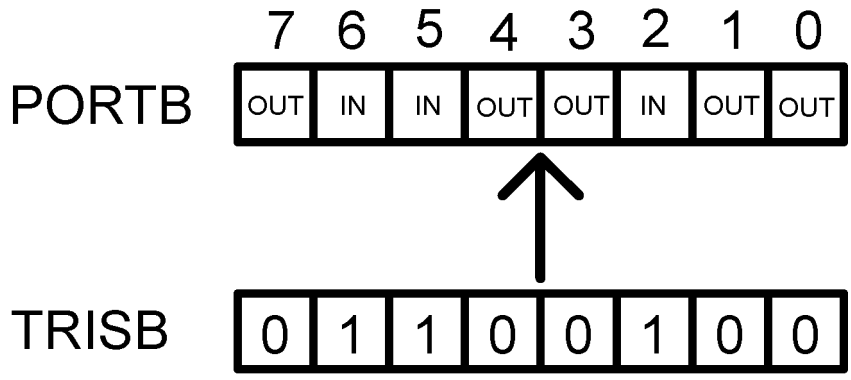
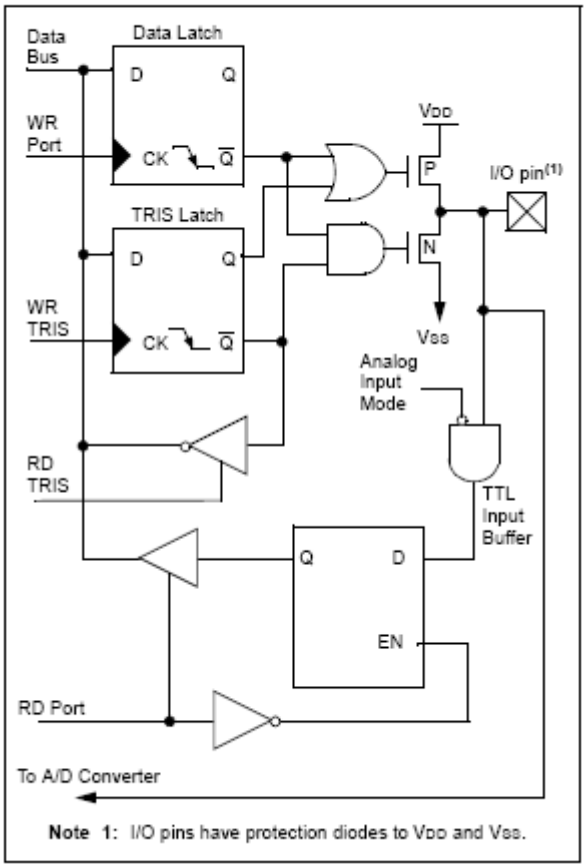


FIGURE 3-1: BLOCK DIAGRAM OF RA3:RA0 AND RA5 PINS



K lepšímu pochopení funkce jednotlivých portů slouží jeho bloková schémata, uvedená v katalogovém listu každého mikrořadiče. Jako příklad lze uvést blokové schéma pinů RA0 – RA3, z kterého je patrna funkce „zachytávacích“ latch klopných obvodů datových registrů a registrů TRIS a taky směr tok datu při určitých stavech výstupu klopného obvodu TRIS Latch:

Ostatními SFR se budeme zabývat postupně, tak jak je budeme potřebovat při programování jednotlivých úloh.

Ještě však je třeba se zmínit o několika dalších, velmi důležitých funkčních blocích, bez kterých by mikrořadič nemohl pracovat:

W - REGISTR

Pracovní registr W (Work Register) má mezi ostatními registry poněkud výsadní postavení. Není totiž umístěn do oblasti datové paměti a není tedy uživateli přístupný pomocí konkrétní adresy, jako ostatní registry. Je využíván k aritmetickým a logickým operacím s daty a k přesunům dat mezi registry. Podle výsledku těchto operací se pak nastavují jednotlivé stavové bity registru ***STATUS***. Toto ovlivnění jednotlivých stavových bitů je uvedeno u jednotlivých instrukcí ve výpisu instrukční sady v příloze **xxx**.

PC (Program Counter)

13 - bitový ***Program Counter*** (čítač programu) obsahuje adresu instrukce, která má být vzápětí provedena. Je to tedy jakési „ukazovátko" uvnitř paměti programu, které neustále ukazuje za instrukci, která bude vzápětí provedena. Po provedení instrukce se jeho hodnota zvýší o 1 - ukáže na následující instrukci (vzpomeňme si, že program je sled jednotlivých instrukcí, postupně čtených z paměti programu a vzápětí vykonávaných) . Výjimkou jsou instrukce skoků a volání podprogramu, v těchto případech je hodnota PC dána hodnotou obsaženou v instrukci skoku (novou adresou, na kterou se má přemístit vykonávání programu). Do ***PC*** je možno zapisovat a takto je možno přímo ovlivňovat běh programu. Této techniky se využívá při práci s tabulkami.

STACK

13 - bitový registr STACK se používá pro uložení a opětovné vyvolání návratových adres při volání podprogramů nebo při vyvolání přerušení (interrupt). Tento registr není umístěn v adresovém prostoru programové ani datové paměti a jeho ukazatel (Stack Pointer) nelze uživatelsky žádným způsobem ovlivnit.

Má 8 úrovní, je do něj tedy možno uložit maximálně 8 návratových adres. Informace uložená do STACK registru jako první je při čtení vyvolána jako poslední (představme si, že vkládáme věci do papírového pytle a pak je znovu vytahujeme ven. Předmět, který jsme vložili do pytle jako poslední je nahoře a jako první jej tedy vyjmeme ven). Z toho vyplývá jedna nepříjemná skutečnost – devátý zápis v řadě má za následek nenávratnou ztrátu („vypadnutí") zápisu prvního. Na tuto skutečnost je třeba myslet při vkládání (vnořování) více podprogramů do sebe. Naneštěstí není situace, kdy dojde k „přetečení" registru STACK indikována žádným stavovým bitem, je tedy třeba dát pozor na to, aby celkový počet „vnořených" podprogramů nepřesáhl číslo 8.

ALU (aritmeticko-logická jednotka) - provádí matematické a logické operace s daty. Je to „výpočetní centrum" každého mikroprocesoru, a tedy i našeho mikrořadiče. Zajímavostí je, že tato „výpočetní jednotka" umí u levnějších typů mikrořadičů pouze sčítat (přestože existuje instrukce pro odečítání, je to pouze jistý speciální případ součtu)! Všechny ostatní matematické operace je nutno řešit pomocí speciálních podprogramů. Do ***ALU*** nemáme přímý přístup, výsledky operací jsou předávány ostatním registrům (především registru ***W***).

Speciální funkční registry, které jsme si doposud uvedli, nalezneme v každém mikrořadiči. Žádný mikrořadič se neobejde bez aritmetické jednotky, speciálních funkčních registrů, portů, čítače programu nebo zásobníku návratových adres. Co se týká ostatních funkčních bloků, prakticky všechny mikrořadiče obsahují ještě minimálně jeden čítač/časovač (o kterém si povíme později) a jeden tzv. ***Watchdog*** (doslovně „hlídací pes", který nám občas pomůže dostat se z bezvýchodné situace, která může nastat např. tím, že se program díky nějakému nečekanému vnějšímu vlivu dostane do nekonečné smyčky) , avšak pak už začíná být situace značně nepřehledná.

Jak již bylo zmíněno v úvodu, nabízejí výrobci velké množství typů mikrořadičů, lišících se právě „tím ostatním“ – tedy periferiemi, o kterých jsme dosud nemluvili. Jsou to především čítače/časovače různých modifikací, ***Capture/Compare*** registry, které dokáží porovnat dvě šestnáctibitová slova a v případě jejich rovnosti vyvolat určitou akci, vícekanálové A/D převodníky, analogové komparátory s vlastním (a většinou programovatelným) zdrojem referenčního napětí, sériové porty typu SPI, I^2C nebo USART, paralelní „Slave“ port pro přenos dat mezi mikrořadiči, obvody pro generování signálu s PWM modulací atd. Významných rozdílem mezi jednotlivými typy mikrořadičů je rovněž počet portů, který přímo ovlivňuje množství pinů a tím i velikost pouzdra.

Vzhledem k omezenému počtu hodin, které máme pro výuku k dispozici, se v této učebnici o těchto periferiích zmíníme jen okrajově a nebudeme je ani využívat v cvičeních. Zájemci o podrobnější studium si mohou potřebné informace vyhledat v katalogových listech výrobce.

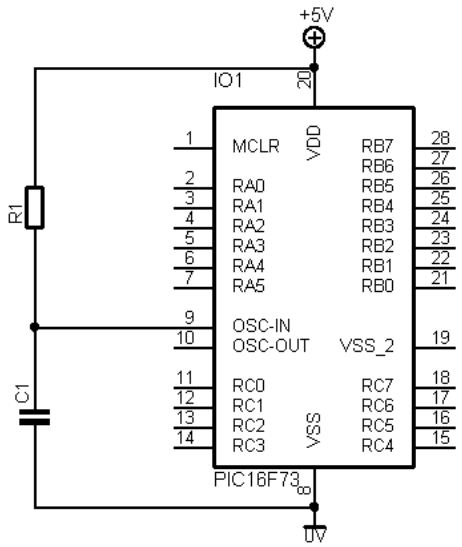
Mikrořadič jako součástka

Základní zapojení mikrořadiče

Podívejme se nyní na praktickou stránku věci. Máme mikrořadič a chceme jej nějak využít. Co potřebujeme k tomu, aby mikrořadič pracoval?

- jako každý integrovaný obvod potřebuje mikrořadič napájecí napětí
- protože je to sekvenční logický obvod, potřebuje ke své činnosti zdroj hodinového kmitočtu
- a konečně, programovatelné obvody bývá většinou nutno před započetím práce uvést do nějakého základního stavu – tzv. resetovat

Zapojení skutečně fungujícího mikrořadiče může vypadat např. takto:

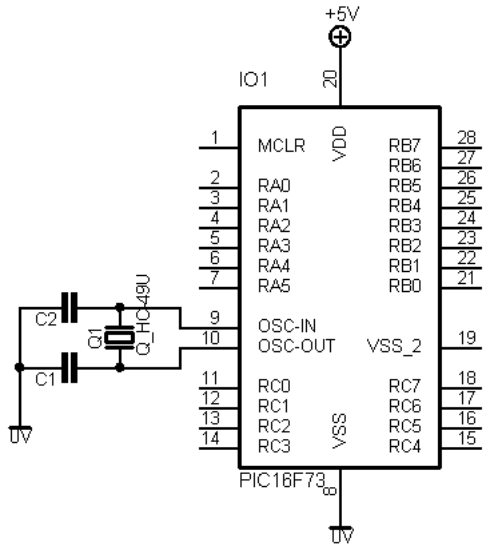


Obr. 3: Mikrořadič s RC oscilátorem.

V_{dd} je vstup kladného napájecího napětí +5V
V_{ss} je společný vývod 0V
OSC1, OSC2 slouží pro připojení obvodů oscilátoru (krystal nebo keramický rezonátor nebo RC člen)
MCLR slouží k nastavení mikrořadiču do výchozího stavu (Reset)

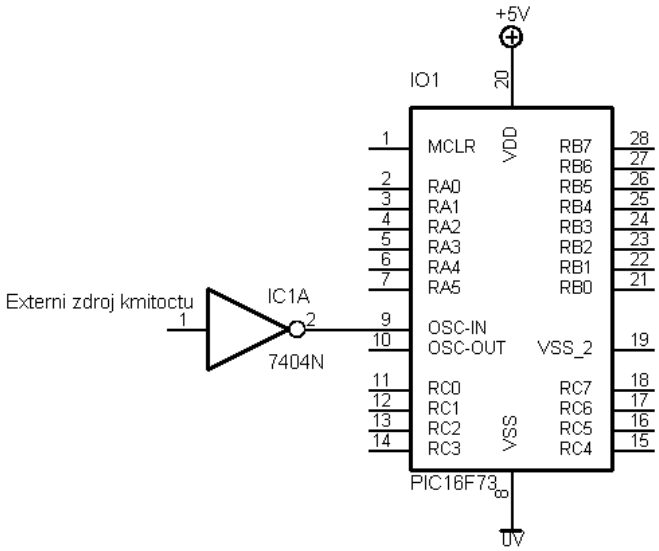
U tohoto zapojení je v obvodu oscilátoru použit levnější, avšak méně přesný a méně stabilní RC člen (R1, C1). S tímto typem oscilátoru vystačíme v aplikacích nenáročných na přesné časování programu.

Pokud potřebujeme přesné časování, je nutno použít zapojení s oscilátorem řízeným krystalem, popř. keramickým rezonátorem:



Obr. 4: Mikrořadič s krystalovým oscilátorem.

V některých případech (např. kvůli synchronizaci více mikrořadičů) potřebujeme řídit takt mikrořadiče externími hodinovými pulsy, např. takto:

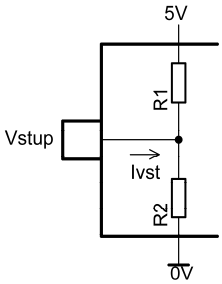


3. 2. Porty, připojení periferních zařízení

Již jsme se zmínili o tom, že s „okolním světem“ mikrořadič PIC16F873 komunikuje pomocí registrů ***PORTA***, ***PORTB*** a ***PORTC.*** Podívejme se tedy, jak se vstupy a výstupy mikrořadiče chovají při připojení externí součástky.

Připojení vstupních zařízení

Je-li pin portu nastaven jako **vstup**, říkáme, že je ve vysoké impedanci (nezatěžuje zdroj signálu k němu připojený). Můžeme si jej představit takto:

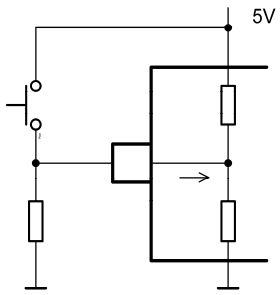


Obr. 5: Náhradní schéma vstupního obvodu mikrořadiče.

Proud, tekoucí vstupním obvodem, bude v tomto případě maximálně 1 uA (tato hodnota je daná vnitřním zapojením vstupních obvodů). Je tedy tak malý, že jej obvykle můžeme ve výpočtech zanedbat a nemusíme se jím při návrhu systému zabývat.

Přivedeme-li na vstup napětí v rozsahu 0V až 0,8V, mikrořadič tuto hodnotu chápe jako nízkou úroveň - logickou nulu (L). Napětí na vstupu v rozsahu 2,4V až 5V vyhodnotí jako vysokou úroveň – logickou jedničku (H).

Připojíme-li na vstup mikrořadiču spínač, můžeme jej zapojit např. tak, aby v rozepnutém stavu byla na vstupu log. 0 a v sepnutém stavu log. 1. Pokud mikrořadič zjistí, že je na vstupu úroveň H, znamená to, že je spínač sepnutý, zjistí-li úroveň L, považuje spínač za rozepnutý.



Obr. 6: Připojení tlačítka ke vstupnímu pinu mikrořadiče.

Vše, co připojujeme ke vstupu mikrořadiče, musí být navrženo tak, aby logická jednička či nula vždy vyjadřovaly určitý stav nebo situaci (někdo je ve střeženém prostoru = log. 1, nikdo tam není = log. 0).

Důležité je správně navrhnout hodnotu rezistoru spojujícího vstup se zemí.

Je-li spínač sepnut, je napětí + 5V přivedeno přímo na vstup mikrořadiče. Do vstupu teče proud maximálně 1uA. Přes rezistor tedy poteče proud I = 5/R. Z důvodu omezení pronikání rušivých napětí na vstup mikrořadiče volíme hodnotu rezistoru takovou, aby rezistorem tekl proud nejméně 10krát větší, než teče vstupním obvodem. Jednoduchým výpočtem (Ohmův zákon) tedy můžeme zjistit, že v našem případě by hodnota rezistoru neměla být větší než 500kΩ.

Je-li spínač rozepnut, teče proud ze vstupu mikrořadiče přes rezistor a opět nepřesáhne 1uA. Vlivem tohoto proudu vzniká na rezistoru úbytek napětí. Nyní je důležité, aby tento úbytek napětí nepřesáhl hodnotu 0,8V (úroveň L), což omezuje jeho maximální hodnotu na 800kΩ.

Vidíme tedy, že z hlediska vstupních obvodů mikrořadiče je omezena pouze maximální hodnota tohoto rezistoru. Je však třeba myslet i na to, že je-li spínač sepnut, teče rezistorem proud, který zatěžuje napájecí zdroj. Nejmenší hodnotu rezistoru tedy volíme takovou, aby nebylo nutno dodávat ze zdroje příliš velký proud. Zvolíme-li např. 10kΩ, odpovídá tato hodnota oběma požadavkům a zdroj přitom zatěžujeme proudem 0,5 mA, což je přijatelné.

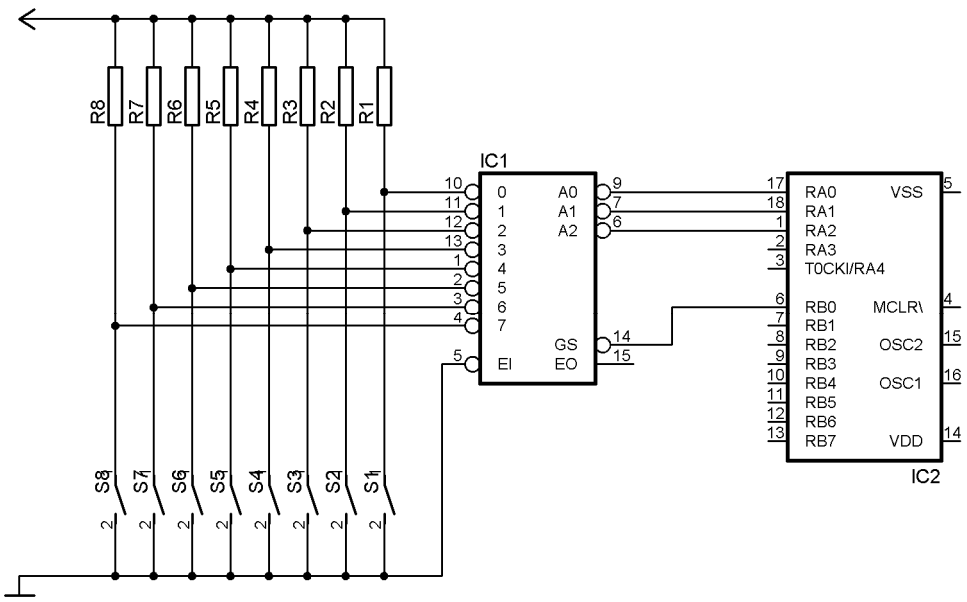
Připojení klávesnice

Klávesnice počítače nebo mikrořadiče je vlastně sada jednoduchých, vhodně zapojených spínacích tlačítek, která jsou při stlačení buďto připojována na napájecí napětí (úroveň H) nebo jsou uzemňována (úroveň L).

Vystačíme-li si s jednoduchou klávesnicí s několika málo tlačítky (např. pro dálkový ovladač), můžeme jednotlivá tlačítka připojit přímo na některý z portů mikrořadiče tak, jak bylo naznačeno na obr. …xx.

Většinou však potřebujeme tlačítek více než si můžeme dovolit připojit přímo k portům mikrořadiče. V takových případech můžeme použít některého z následujících zapojení:

Prioritní kodér



IC1 = 74148
IC2 = mikrořadič

FUNCTION TABLE – '148, 'LS148

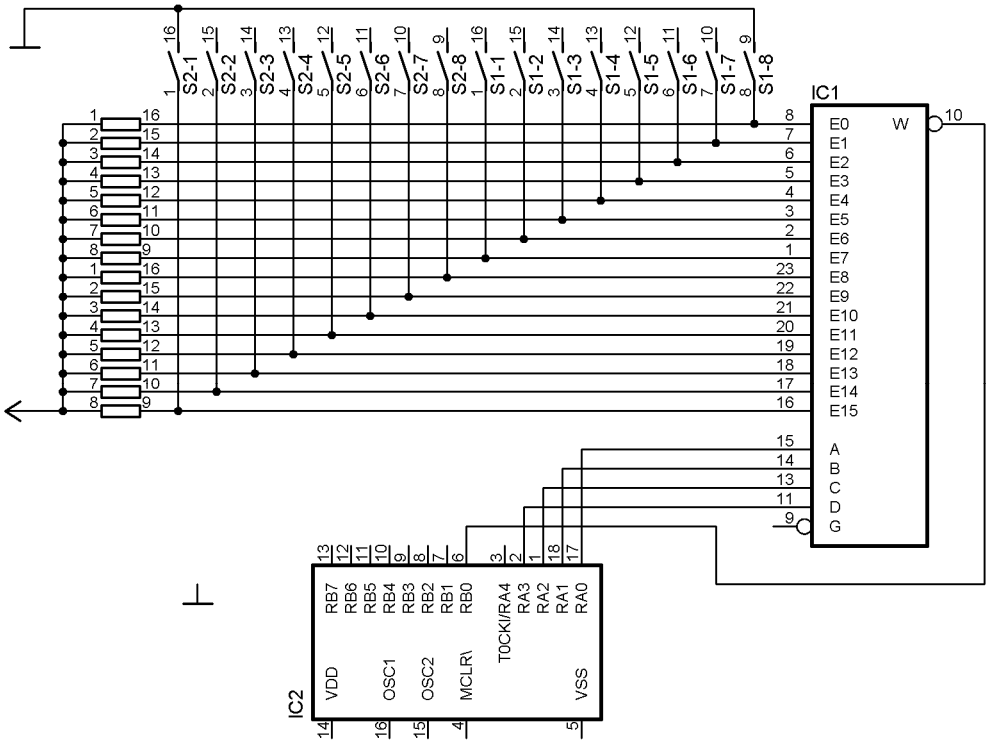
INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

H = high logic level, L = low logic level, X = irrelevant

Na výstupech A0 – A2 je stav, odpovídající stisknutému tlačítku s nejvyšší prioritou. Jak je vidět z tabulky, nejvyšší prioritu má vstup 7 a nejnižší vstup 0.

GS se použije pro vyvolání externího přerušení programu mikrořadiče (čtení klávesnice).

Multiplex



IC1 = 74150

54150/74150

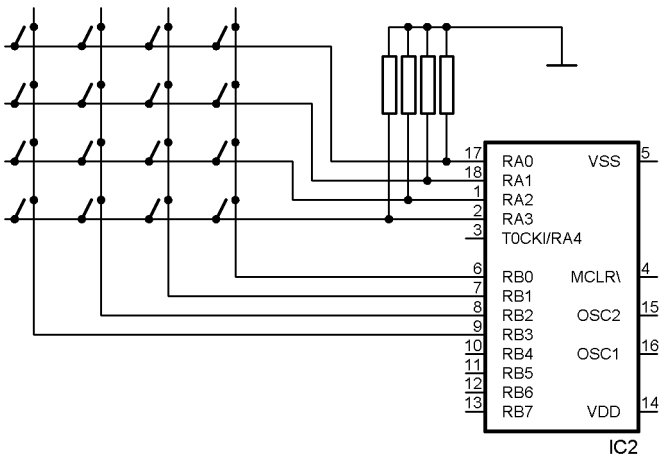
Inputs					Outputs
Select				Strobe	W
D	C	B	A	S	
X	X	X	X	H	H
L	L	L	L	L	$\overline{E0}$
L	L	L	H	L	$\overline{E1}$
L	L	H	L	L	$\overline{E2}$
L	L	H	H	L	$\overline{E3}$
L	H	L	L	L	$\overline{E4}$
L	H	L	H	L	$\overline{E5}$
L	H	H	L	L	$\overline{E6}$
L	H	H	H	L	$\overline{E7}$
H	L	L	L	L	$\overline{E8}$
H	L	L	H	L	$\overline{E9}$
H	L	H	L	L	$\overline{E10}$
H	L	H	H	L	$\overline{E11}$
H	H	L	L	L	$\overline{E12}$
H	H	L	H	L	$\overline{E13}$
H	H	H	L	L	$\overline{E14}$
H	H	H	H	L	$\overline{E15}$

H = High Level, L = Low Level, X = Don't Care

$\overline{E0}$, $\overline{E1}$ . . . $\overline{E15}$ = the complement of the level of the respective E input

Multiplexer pracuje tak, že na výstupu W se nachází negovaný stav vstupu E0 – E15, adresovaného vstupy ABCD.
Vstup G je povolení výstupu a musí být v úrovni L, aby se data ze vstupů objevila na výstupu.

Pro zjištění, které tlačítko je stlačeno, mikrořadič postupně adresuje všechny vstupy a zjišťuje stav výstupu. Zjistí-li stav L, je dané tlačítko stlačeno, zjistí-li stav H, tlačítko stlačeno není.

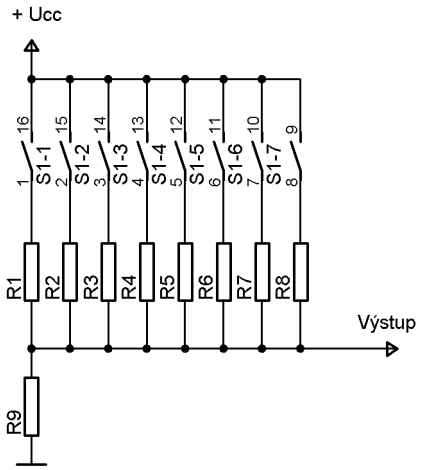


Matice

RB0 – RB3 = výstupy (nastavování sloupců)
RA0 – RA3 = vstupy (čtení řádků)

Mikrořadič postupně nastavuje na jednotlivých sloupcích (port B) úroveň H a vzápětí čte řádky (port A). Je-li některé z tlačítek stlačeno, objeví se na patřičném řádku úroveň H. Z vyslané a přečtené informace pak mikrořadič dekóduje, které tlačítko je v tu chvíli stlačeno.

Dělič napětí



Pro mikrořadiče, vybavené A/D převodníkem lze použít následující zapojení.
Rezistory R1 – R8 tvoří spolu s rezistorem R9 dělič napětí. Výstupní napětí se přivádí na vstup A/D převodníku mikrořadiče, který podle velikosti tohoto napětí vyhodnotí, které tlačítko je stlačeno.

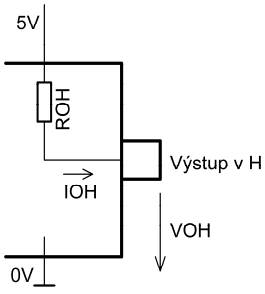
Pozn.: A/D převodník bývá většinou osmibitový, proto je tímto způsobem teoreticky možnost rozlišit až 255 tlačítek. K tomu by však bylo nutno dokonale stabilizovat napájecí napětí a přesně vybírat rezistory. Krok napětí pro dvě sousední tlačítka by pak byl 5/256 = 0,0195V a údaj převodníku by představoval dvojkové číslo stlačené klávesy.

Všechna tato zapojení mají jednu společnou vlastnost: podstatně redukují počet vodičů, potřebných pro připojení k mikrořadiči. Nejvýraznější úspora je u verze s děličem napětí a A/D převodníkem, kde je klávesnice připojena k mikrořadiči pouze jediným vodičem.

Podobně postupujeme, potřebujeme-li ke vstupnímu pinu připojit jiné druhy výstupních zařízení (LCD displeje, vysílací IR diody, optoizolační členy apod.). Vždy je třeba vycházet z vlastností a možností jednotlivých zařízení a tomu pak podřídit jejich pripojení k mikrořadiči.

Připojení výstupních zařízení

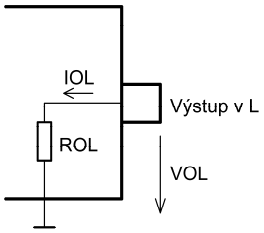
Je-li pin portu nastaven jako **výstup** a je v úrovni H, pak si jeho zapojení můžeme představit takto:



Obr. 7: Náhradní schéma výstupního obvodu mikrořadiče v úrovni H.

R_{OH} má hodnotu přibližně 90Ω. Je-li k pinu připojen nějaký obvod, může v úrovni H proud téci pouze z pinu směrem ven.

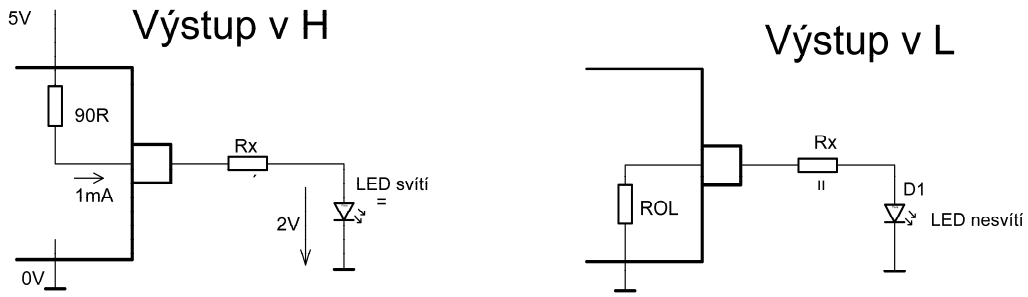
Je-li pin portu nastaven jako **výstup** a je v úrovni L, pak zapojení vypadá takto:



Obr. 8: Náhradní schéma výstupního obvodu mikrořadiču v úrovni L.

R_{OL} má hodnotu přibližně 26Ω. Je-li k pinu připojen nějaký obvod, může v úrovni L proud téci pouze směrem dovnitř.

Jako příklad si uvedeme připojení LED diody:



Obr. 9: Připojení LED diody k výstupnímu pinu proti zemi.

Jak je ze schématu patrné, bude-li výstup v úrovni L, pinem nepoteče proud a LED dioda nebude svítit.

Nastaví-li mikrořadič výstup do úrovně H, LED dioda se rozsvítí. Použijeme-li nízkopříkonovou LED diodu, bude dostačující proud 1mA. Navrhneme omezovací odpor:

Napětí na pinu bude sníženo oproti napájecímu napětí o úbytek napětí na rezistoru 90Ω, kterým poteče rovněž proud 1mA: 90 x 0,001 = 0,09V. Na LED diodě je úbytek napětí přibližně 2V. Na rezistoru R_x tedy vznikne úbytek napětí: 5 – 0,09 – 2 = 2,91V.

Z Ohmova zákonu snadno spočítáme, že R_x = 2,91 / 0,001 = 2910Ω. Zvolíme nejbližší hodnotu z běžně vyráběné řady E12, tedy 2k7 nebo 3k3.

LED diodu můžeme samozřejmě zapojit i proti kladnému napájecímu napětí. V takovém případě bude svítit tehdy, bude-li na výstupním pinu úroveň L a naopak. LED diodu je třeba v tomto případě přepólovat, výpočet omezovacích rezistorů bude obdobný:

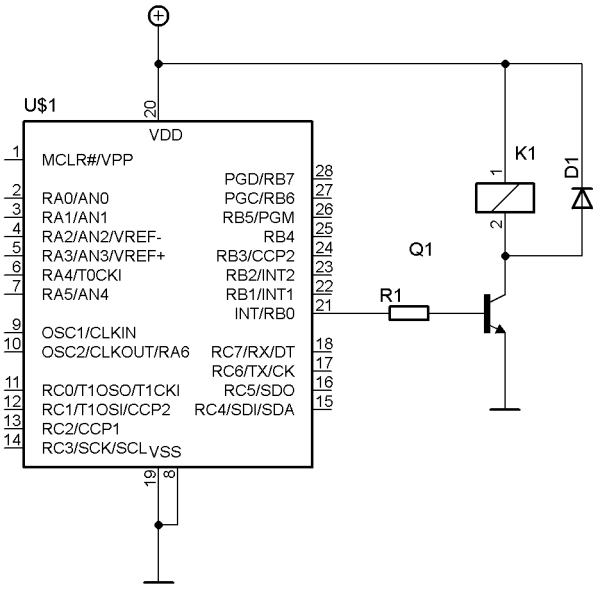
Podobným způsobem bychom při výpočtu omezovacího rezistoru postupovali, kdybychom měli k výstupnímu pinu připojeny jiné druhy zařízení (např. spínací tranzistor, relé, optoizolační člen apod.).

Při připojování výstupních zařízení k mikrořadiči je nutno mít na paměti, že nesmí být překročen maximální povolený proud jedním výstupním pinem a rovněž maximální povolený proud pro celý port. U mikrořadiče PIC16F873 je to 25mA, resp. 200mA.

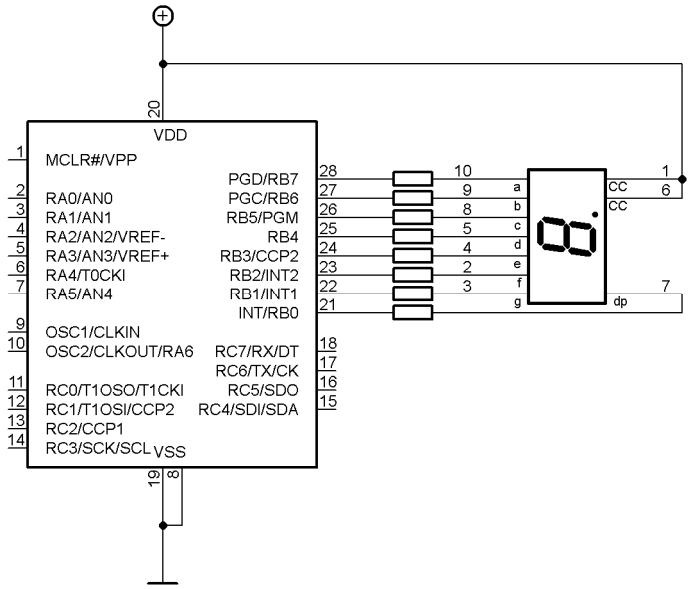
Pokud potřebujeme k výstupnímu portu připojit zařízení s vyšším proudovým odběrem, je nutno výstupní pin „posílit“ vhodným zesilovacím prvkem (např. tranzistorem).

Při připojování relé nesmíme nikdy zapomenout připojit k její cívce antiparalelně zapojenou ochrannou diodu, jako ochranu proti průrazu portu impulzním napětím při rozepnutí cívky relé.

Následující obrázek ilustruje připojení relé s ochrannou diodou a tranzistorem pro proudové posílení výstupního obvodu.

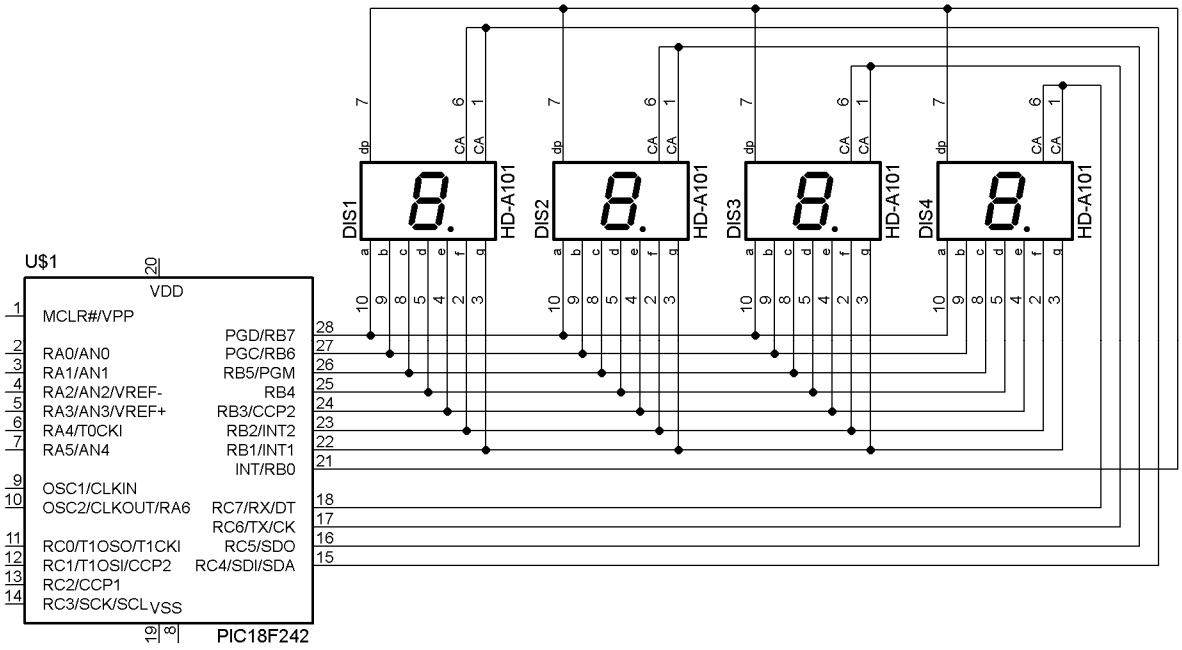


Často používanou součástkou je u mikrořadičů sedmisegmentová LED zobrazovací jednotka („číslovka“). Protože jednotlivé segmenty nejsou nic jiného, než obyčejné LED diody, můžeme jednu takovou číslovku zapojit např. takto (je použitá číslovka se společnou anodou):



Hodnotu jednotlivých rezistorů je nutno vypočítat tak, aby bylo dosaženo přijatelného jasu segmentů a přitom nebyl překročen maximální povolený proud jednotlivých pinů a výstupního portu. Pokud nelze přijatelného jasu bez nebezpečí proudového přetížení portu dosáhnout, je nutno buďto zakoupit číslovku s vyšší hodnotou svítivosti nebo v krajním případě proudově posílit každý jednotlivý segment tranzistorem, samozřejmě za cenu složitějšího zapojení.

Výhodou uvedeného zapojení je bezesporu jeho jednoduchost. Pokud bychom však chtěli tímto způsobem zapojit více číslovek, například čtyři, narazili bychom u tohoto typu mikrořadiče na nedostatek výstupních portů. Řešení, které se nabídne jako první, je pořídit si mikrořadič s dostatečným počtem portů. To však znamená zaplatit za něj podstatně vyšší cenu. Naštěstí existuje řešení, jak zapojit více číslovek k mikrořadiči, který disponuje pouze dvěma porty:



Vtip spočívá v tom, že všechny stejné segmenty číslovek (v případě číslovky se společnou anodou všechny stejné katody) jsou vzájemně spojeny a připojeny na jeden pin výstupního portu (např. spojené segmenty „a“ jsou připojeny na RB7). Společné anody jednotlivých číslovek jsou pak samostatně připojeny na čtyři piny jiného výstupního portu (např. port C).
Jakmile mikrořadič bude chtít zobrazit např. číslo 1 2 3 4, vyšle nejprve na port B (a tedy na katody všech číslovek) první číslici – tedy jedničku. Vzhledem k tomu, že všechny stejné segmenty všech čtyř číslovek jsou navzájem propojeny, zobrazila by se naše jednička na všech číslovkách současně, pokud bychom ovšem připojili všechny čtyři jejich anody na kladné napájecí napětí.
V tuto chvíli však mikrořadič připojí kladné napájecí napětí (log. úroveň H) pouze na první číslovku a naše jednička se tedy zobrazí pouze na ní.
Po malé časové prodlevě (v řádu milisekund) mikrořadič odpojí napájení první anody a vyšle na port B (a tedy opět na segmenty všech čtyř číslovek) druhou číslici – tedy dvojku a vzápětí sepne opět na několik milisekund napájení anody druhé číslovky. Dvojka se tedy na malou chvilku objeví na druhé číslovce. Totéž se provede s třetím a čtvrtým číslem a celý děj se pak neustále opakuje.
Při tomto způsobu zobrazení je tedy každá jednotlivá číslice zobrazena samostatně – nelze zobrazit více číslic najednou. Protože se však celý cyklus opakuje mnohokrát za sekundu, vnímáme tuto posloupnost zobrazení jednotlivých číslic jako zobrazení celého čísla najednou (známá setrvačnost lidského oka, díky které můžeme např. vnímat plynule filmový záznam, který je rovněž složen z jednotlivých statických záběrů).

Pro jednoduchost je v tomto schématu vynechán obvod oscilátoru a předpokládá se, že budou použity zobrazovače s velmi malým proudovým odběrem, jinak bychom museli posílit výstupy RC4 – RC7 tranzistory (nesmíme zapomenout, že společnou anodou každé číslovky teče součet proudů jednotlivých segmentů).

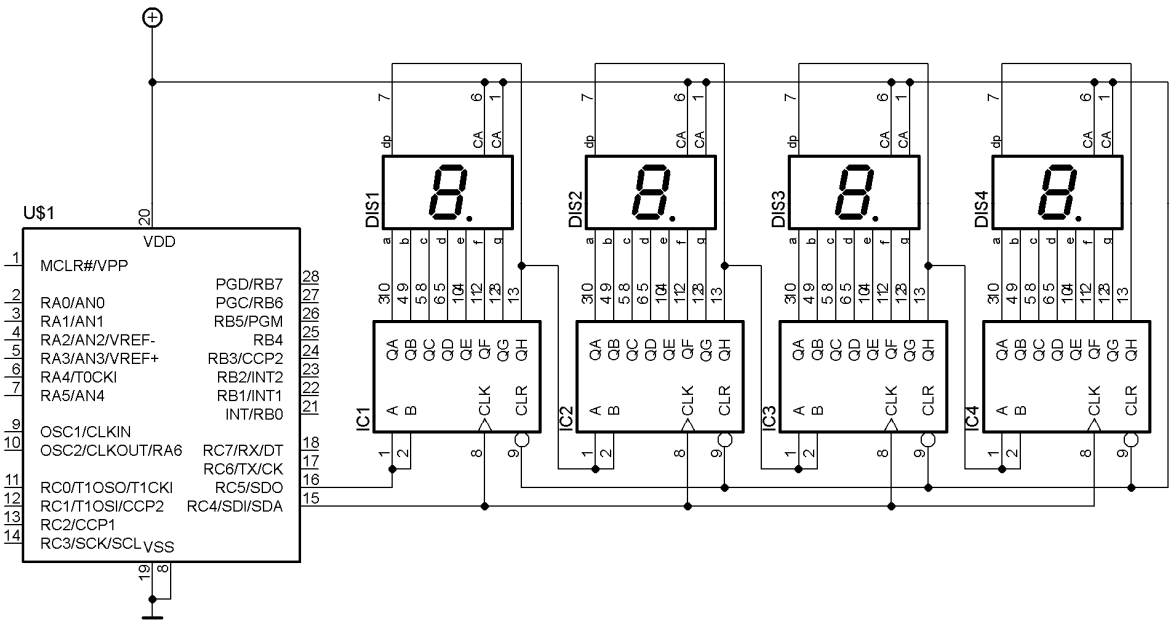
Je zřejmé, že přepínání zobrazování jednotlivých číslic musí být řízeno programem mikrořadiče, který musí zajišťovat vysílání jednotlivých číslic i periodické přepínání anod jednotlivých číslovek ve správný okamžik a správnou frekvencí.

Je vidět, že při tomto způsobu zobrazení si při čtyřech číslovkách vystačíme pouze s dvanácti vodiči.

Tento způsob řízení displeje se nazývá **multiplexním řízením.**

Největší výhodou multiplexního způsobu řízení je jednoduchost zapojení. Naopak nevýhodou je nutnost programové obsluhy displeje a taky skutečnost, že segmenty svítí vždy jen určitou dobu a po zbytek periody jsou zhasnuty. To má za následek snížení jasu segmentů. Tento nedostatek je možno řešit zmenšením hodnoty sériových rezistorů jednotlivých segmentů, protože LED dioda snese v impulsním režimu větší proud, než při trvalé zátěži. Pak ale zase hrozí jiné nebezpečí – pokud necháme LED diody nějakou dobu trvale svítit (což se při testování programu určitě stane), nemusí se jim to příliš zamlouvat, protože jimi v tu chvíli poteče značný trvalý proud a navíc to nemusí vydržet ani porty mikrořadiče.

Určitým kompromisem mezi přímým a multiplexním zapojením zobrazovače je zapojení na následujícím obrázku:



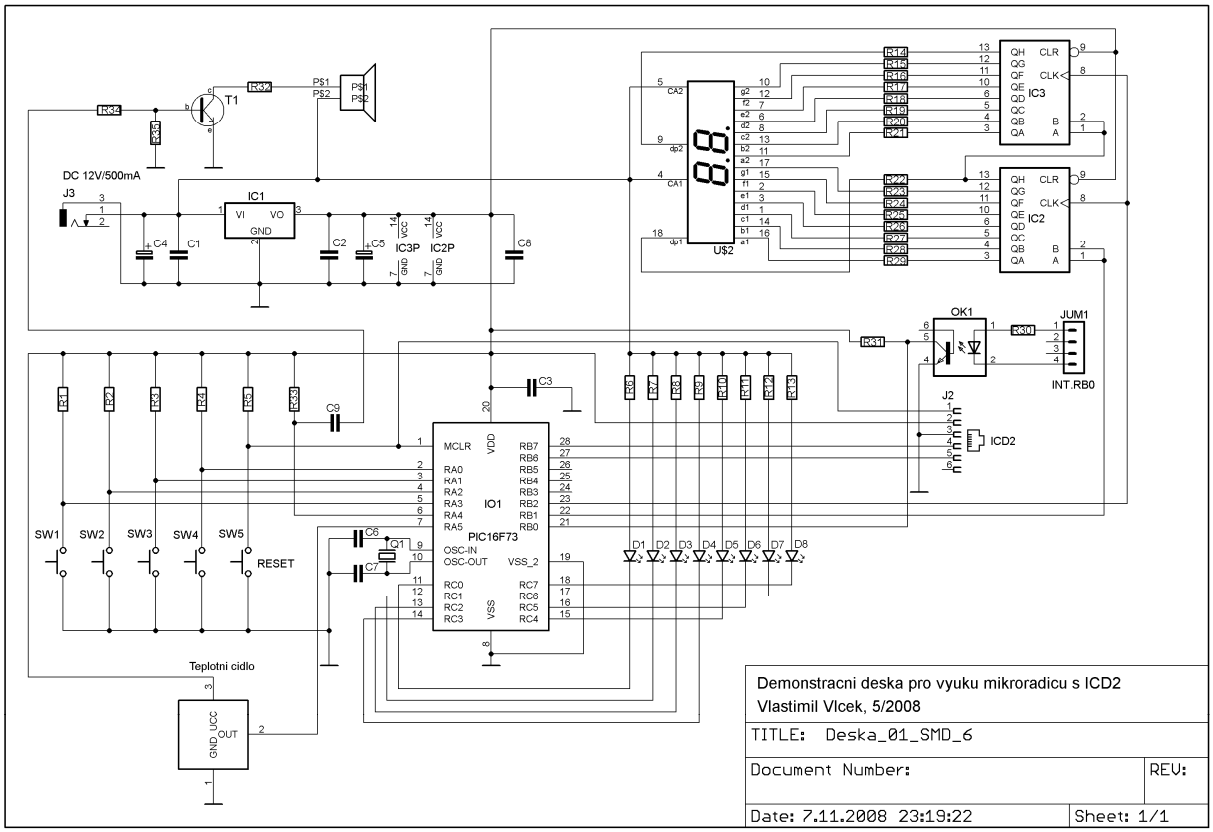
Zobrazovaná data jsou nejprve převedena na sériový tvar a poté vyslána sériovou datovou linkou (pin RC5/SDO) do čtveřice posuvných registrů. Paralelní výstupy těchto registrů jsou připojeny na jednotlivé segmenty zobrazovačů. Data jsou do posuvných registrů zapisována

(posouvána) hodinovými impulsy, které mikrořadič generuje na pinu RCA/SDI/SDA. V našem případě, kdy máme zapojeny čtyři zobrazovače po osmi segmentech, je tedy k přenosu zobrazované informace zapotřebí 32 hodinových impulzů.

Toto řešení má jednu podstatnou výhodu. Data jsou na výstupech posuvných registrů držena trvale tak dlouho, než jsou přepsána dalšími daty. V praxi to znamená, že jas všech segmentů je rovněž trvalý (není periodicky přerušován multiplexem) a zobrazení je tedy kvalitní a stabilní.
Další výhodou je menší náročnost na software, který zobrazení obsluhuje. Vše, o co se nyní musí program postarat, je převod dat na sériový tvar a vyslání jednotlivých bitů na patřičný výstupní port.

A jaký závěr bychom mohli učinit z toho, co bylo řečeno? Asi ten, že je na každém konstruktérovi, kterou variantu zvolí – bude to zřejmě taková, kterou považuje v dané situaci za nejvhodnější.

Nyní se již jistě dokážeme vyznat ve schématu výukové desky, kterou budeme používat:



Je zřejmé, že výuková deska obsahuje tyto části:
- stabilizátor napětí +5V pro napájení mikrořadiče a posuvných registrů
- čtyři tlačítka = vstupní zařízení
- konektor pro připojení externího signálu přerušení = vstupní zařízení
- osm LED diod = výstupní zařízení
- reproduktor = výstupní zařízení
- segmentový LED displej = výstupní zařízení
- tlačítko RESET = vstupní zařízení (reset mikrořadiče)

Tlačítka a LED diody jsou zapojeny na samostatné vstupní/výstupní piny mikrořadiče a mohou být tedy ovládány přímým zápisem potřebných logických úrovní do jednotlivých pinů portu A a portu C.
Data do LED displeje jsou posílána v sériovém tvaru a převod do potřebného paralelního tvaru zajišťuje dvojice posuvných registrů IC2, IC3. Převod dat do sériového tvaru je nutno zajistit programově.

4. Software - co je to program?

Již jsme se zmínili o tom, že každý mikroprocesor a mikrořadič potřebuje ke své činnosti program. Bez něj je mikrořadič pouze elektronická součástka, která je sice funkční, ale v podstatě nic pořádného neumí. To, co má umět, ji musíme nejprve naučit.

Každý mikroprocesor a mikrořadič disponuje určitým počtem tzv. **instrukcí**, tedy povelů, kterým rozumí a umí je provádět. Tyto instrukce jsou až triviálně jednoduché, avšak jejich vhodným seřazením a postupným prováděním lze docílit i velmi složitých operací.

Posloupnost instrukcí tvořících určitý funkční celek a vykonávající tedy určitou konkrétní činnost, tvoří **program**.

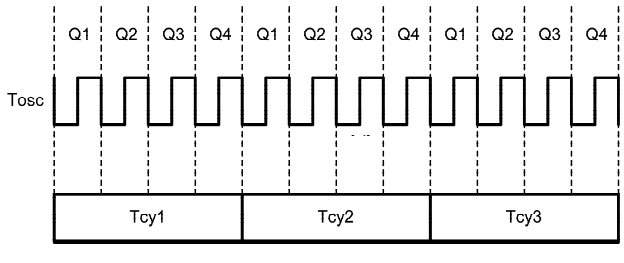
Program je uložen v paměti mikrořadiču (v jemu vyhrazené oblasti). Ve chvíli, kdy je spuštěn, přečte mikrořadič první instrukci, dekóduje ji a provede. Současně se čítač programu (PC – Program Counter) posune o jednu adresu programové paměti dále, je přečtena, dekódována a provedena další instrukce, PC se opět posune o jedno místo dále atd.

V úvodu jsme si řekli, že mikrořadič potřebuje ke své činnosti zdroj hodinového kmitočtu. Nyní již asi tušíme proč. Vidíme totiž, že celá činnost obvodu, a tedy i programu, je přesně časována, každý úkon má pro sebe vyhrazen ten správný okamžik, ve kterém má být proveden. Zde je nutno se zmínit o jedné důležité věci. Každá instrukce zabírá určitý počet tzv**. instrukčních cyklů**. Kolik instrukčních cyklů instrukce zabere, je uvedeno v instrukčním souboru mikrořadiču u každé instrukce. Ve většině případů zabírají instrukce jeden instrukčních cyklus, výjimku tvoří instrukce skoků a volání podprogramu, které spotřebují dva instrukční cykly.

A aby to nebylo tak jednoduché, platí, že **jeden instrukční cyklus trvá čtyři periody hodinového kmitočtu mikrořadiču**. Během jednoho instrukčního cyklu musí totiž mikrořadič provést několik operací. Pro ty, které by zajímalo, jak to funguje, uvádím ukázku:

Q – cykly

Každý instrukční cyklus je rozložen do čtyř tzv. Q – cyklů. Q – cyklus odpovídá jednomu hodinovému taktu oscilátoru. Q –cykly určují přesné časování pro operace čtení, dekódování, zpracování a zápisu dat v každém instrukčním cyklu. Následující diagram znázorňuje vztah mezi Q – cykly a instrukčním cyklem:



Obr. 11: Q – cykly instrukčního cyklu.

Q1: dekódování instrukce nebo provedení instrukce NOP
Q2: čtení dat nebo provedení instrukce NOP
Q3: zpracování dat
Q4: zápis instrukce nebo provedení instrukce NOP

Znalost počtu instrukčních cyklů, které instrukce potřebuje ke svému provedení, je velmi důležitá ve chvíli, kdy potřebujeme vytvořit přesně definované zpoždění nebo časovou smyčku. Později si výpočet takovéto časové smyčky vyzkoušíme.

4. 1. Jak začít s programováním?

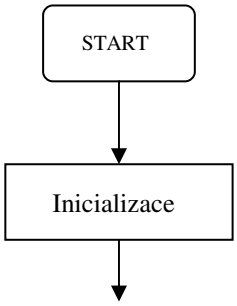
Vytváření jakéhokoliv programu je velmi dobré začít tzv. ***vývojovým diagramem***. Vývojový diagram je grafické znázornění činnosti základních funkcí programu, je to tedy obdoba blokového schématu elektronického zařízení. Smysl jeho použití je v tom, že podává dobrý přehled o celkové funkci programu a o jeho hlavních programových blocích. Tyto jednotlivé bloky je pak možno rozkreslit do dalších, dílčích vývojových diagramů, atd. Je však třeba důkladně zvážit, kdy je ještě rozumné používat toto grafické znázornění, a kdy už je vhodnější začít psát vlastní program pomocí jednotlivých instrukcí.

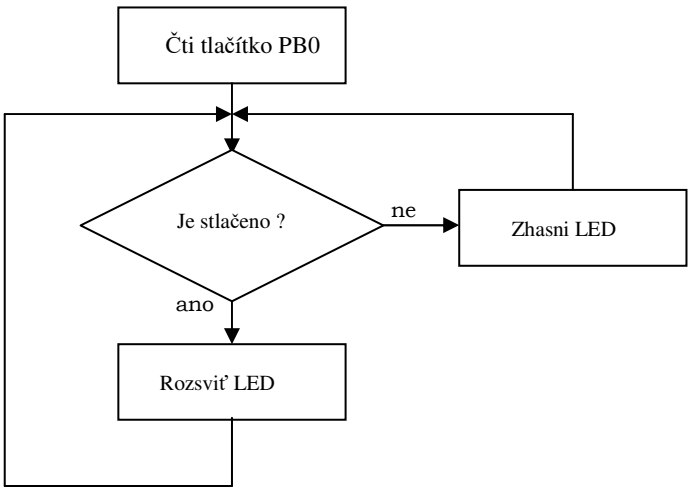
Zde je nutno se zmínit o jedné důležité vlastnosti vývojového diagramu. Protože se jedná o grafické vyjádření řešeného úkolu, v tuto chvíli nezáleží vůbec na tom, v jakém programovacím jazyce bude následně zapsán! Vývojový diagram tedy může teoreticky vypracovat kdokoliv, kdo má představu o tom, jak daný úkol řešit a přitom ani nemusí být programátor. Teprve když je takto zhruba nastíněno řešení, převezme úkol programátor, ovládající patřičný programovací jazyk a jednotlivé bloky vývojového diagramu přepíše do podoby, srozumitelné konkrétnímu mikrořadiči.

V této souvislosti se často používá pojem ***algoritmus.*** Bývá definován jako posloupnost konečného počtu elementárních kroků, vedoucí k vyřešení dané úlohy. Není to docela výstižná definice počítačového programu?

4. 1. 1. Vývojový diagram

Vývojový diagram velmi jednoduchého programu může vypadat například takto:





Obr. 12: Příklad vývojového diagramu.

Činnost tohoto programu lze z vývojového diagramu snadno vysledovat. Po úvodní inicializaci mikrořadič čte tlačítko PB0 a testuje, zda je, či není stlačeno. Pokud stlačeno není, zhasne LED diodu, pokud ano, LED diodu rozsvítí. Je rovněž patrné, že se tento cyklus stále opakuje v nekonečné smyčce.

Jakmile máme činnost programu dostatečně popsanou pomocí vývojového diagramu, můžeme přistoupit k zápisu programu pomocí jednotlivých instrukcí.

K vlastnímu zápisu programu je možno použít jakýkoliv textový editor, který do textu nepřidává žádné formátovací znaky, kromě znaku tabelátoru a znaku Enter. Takovýmto podmínkám vyhovuje např. Poznámkový blok ve Windows XP. Je však mnohem výhodnější použít specializovaný textový editor, který najdeme v každém vývojovým systému a tedy i v programu MPLAB IDE. Vše se nejlépe vyzkoušíme na příkladu:

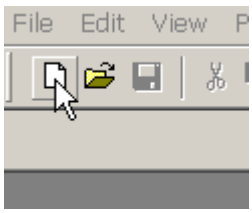
První seznámení s programem MPLAB IDE.

Začneme tím, že si napíšeme krátký ukázkový program.

Zápis zdrojového textu programu.

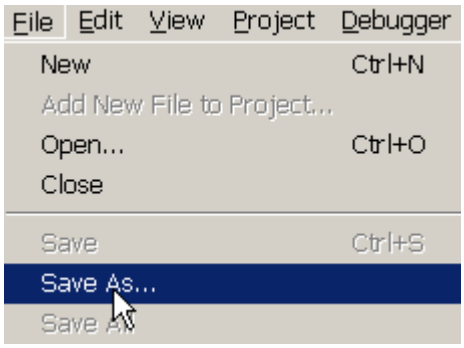
1. Spusťte program MPLAB IDE.

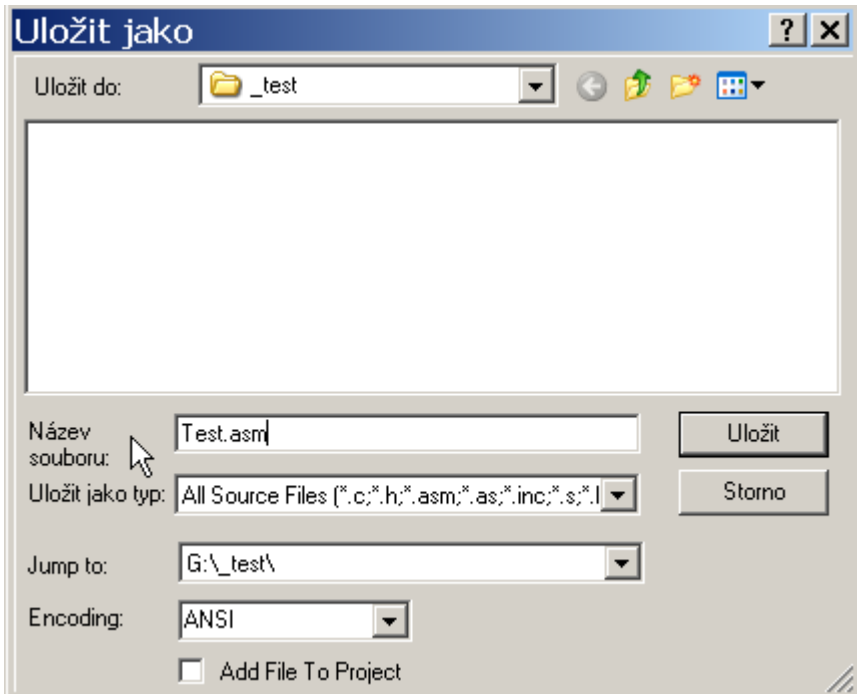
2. Klikněte na ikonku New File.



Otevře se textové okno, pojmenované Untitled.

3. Uložte jej příkazem Save As do vaší složky na disku, pod názvem Test.asm (doporučuji vytvořit další složku např. s názvem **test**).





Do textového okna zapište následující text (odsazení textu proveďte tabelátorem):

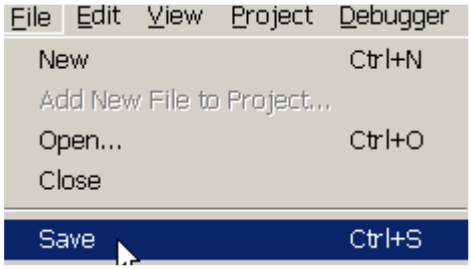
```
start   goto    sem     ;skok na návěští „sem“
        nop
sem     nop
        nip
        goto    start   ;skok na návěští „start“

        end
```

Jistě jste si ihned všimli, že zapsané výrazy jsou navzájem barevně odlišeny. Příkazy jsou označeny modrou barvou, návěští a parametry příkazů barvou fialovou a konečně poznámka (za středníkem) je označena barvou zelenou. Tento příklad rovněž ukazuje způsob zápisu, který by měl být vždy dodržen:

návěští	příkaz	parametr	poznámka
start	goto	sem	;skok na návěští „sem“

Návěští je symbolické označení adresy řádku a je definováno na začátku programu.
Příkazy a jejich parametry jsou definovány v instrukční sadě mikrořadiče.
Jakýkoliv text za středníkem v rámci jednoho řádku je považován za poznámku.
Všimněte si, že text **nip** je zvýrazněn fialově, přestože je umístěn ve sloupci pro zápis příkazů. Je to proto, že příkaz **nip** prostě neexistuje a editor nám jej proto neoznačil modře. Přepište jej na **nop** a vše bude v pořádku.

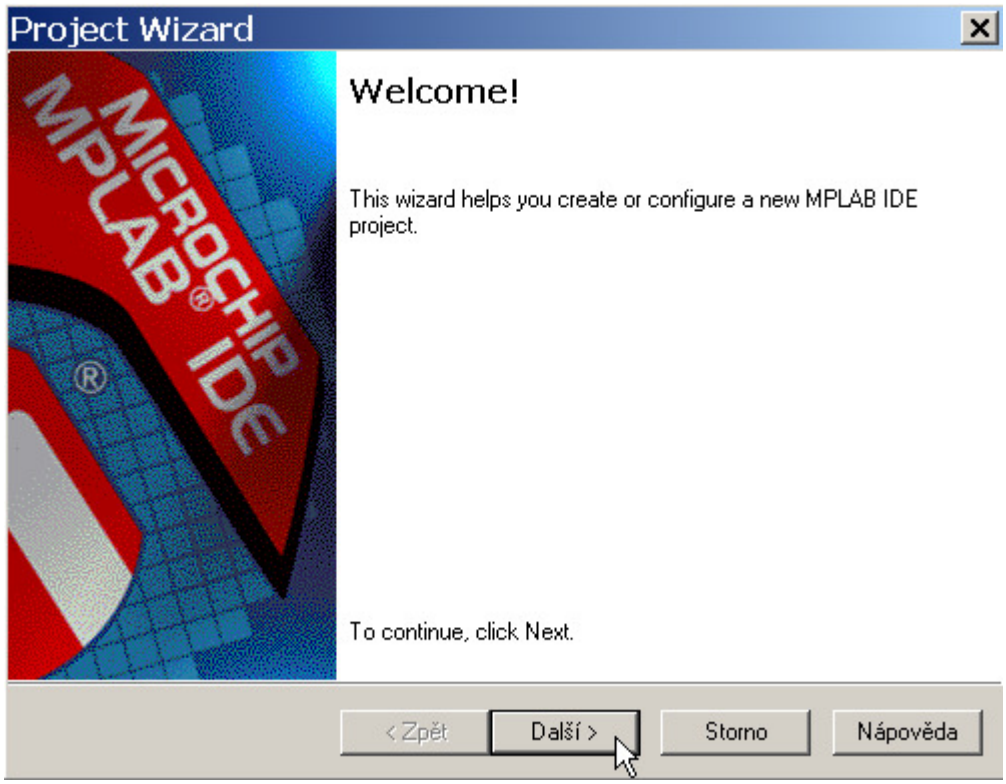
Znovu uložte soubor příkazem Save:



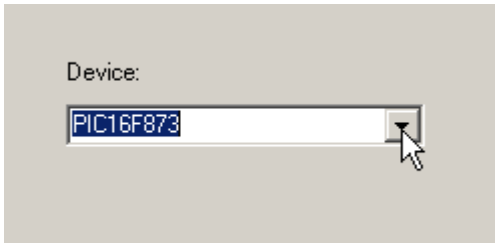
Gratuluji, právě jste zapsali svůj první funkční program.

Založení nového projektu.

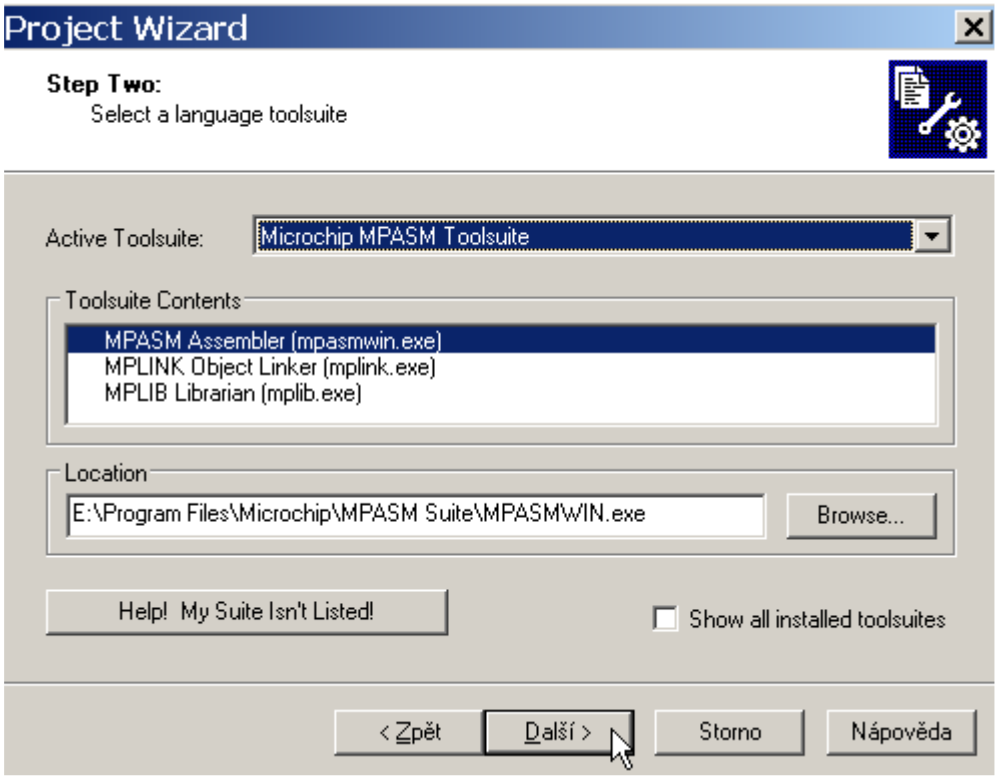
Abychom mohli využít všech možností programu MPLAB IDE, musíme nejprve založit nový projekt:



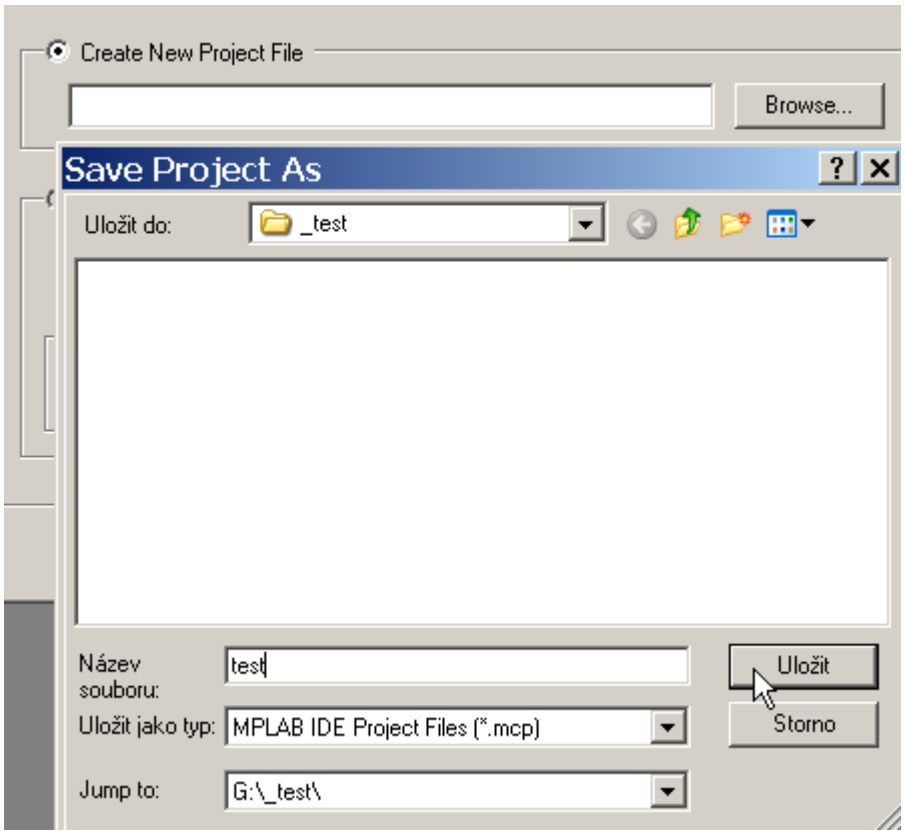
Zvolte typ mikrořadiče:



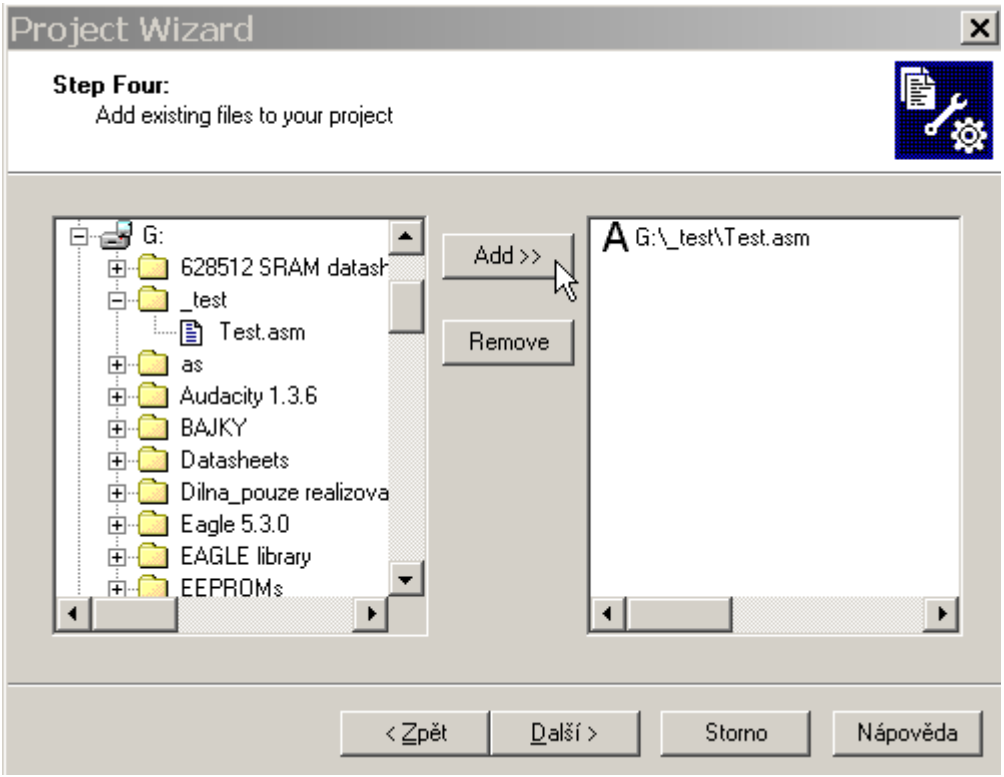
Zkontrolujte, zda následující nastavení odpovídají tomuto zobrazení. Pokud ne, najděte je v instalační složce Microchip a nastavte je.



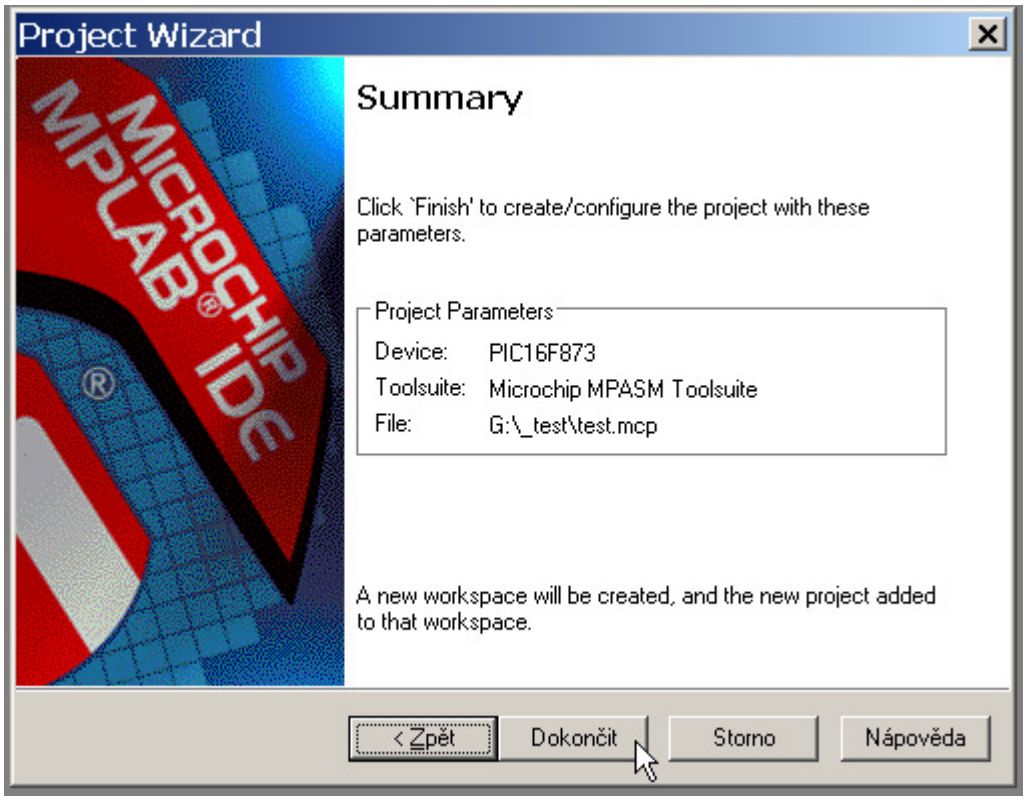
Klikněte na tlačítko Browse, nalezněte vaši složku **test** a zadejte název souboru **test.** Uložte, klikněte na tlačítko **Další.**



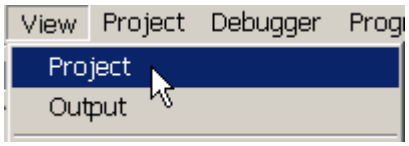
V následujícím okně označte zdrojový soubor **Test.asm** a pomocí tlačítka **Add** jej vložíte do pravé části okna:

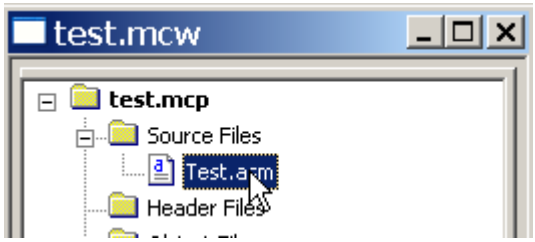


Poslední okno je už jen souhrnem všeho, co jsme doposud udělali. Zkontrolujte, zda je vše v pořádku a dokončete založení nového projektu tlačítkem **Dokončit**:



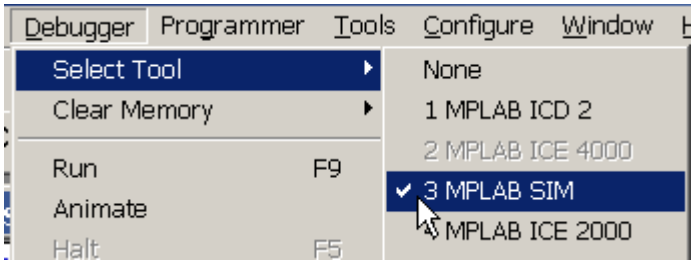
Otevřete zdrojový text programu dvojklikem na jeho název:



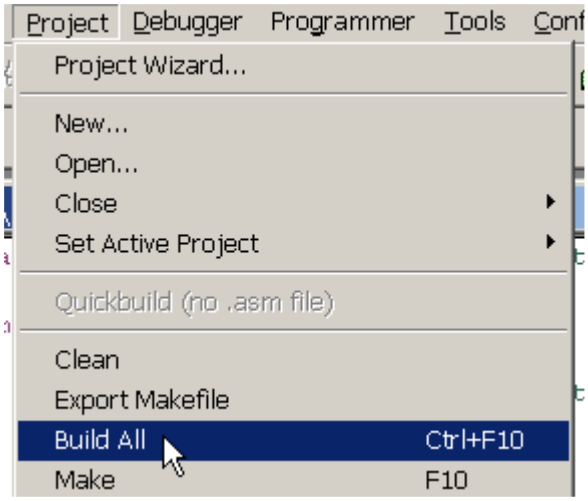


První otestování našeho programu provedeme v simulačním režimu programu MPLAB IDE (vzhledem k tomu, že nepotřebujeme komunikovat s žádným externím zařízením přes porty, nepotřebujeme ICD2).

Zvolte modul MPLAB SIM:



Aby mikrořadič, který veškeré informace umí zpracovávat pouze v dvojkové soustavě, „porozuměl“ našemu programu, zapsanému v textovém tvaru, je třeba jej nejprve přeložit do strojového kódu daného typu mikrořadiče. Tento překlad zajišťuje externí program MPASM, který je volán přímo z MPLAB IDE:



7. Přeložení zdrojového kódu do strojového kódu mikrořadiče

Během překladu se na krátkou chvíli zobrazí okno s indikačním proužkem, jehož barva určuje, zda překladač během překladu narazil na chybu (červený proužek), či nikoliv (zelený proužek). Výsledek překladu je navíc vypsán v okně OUTPUT.

8. Ladění programu

Zdárný průběh překladu ještě automaticky neznamená, že program bude pracovat podle předpokladů, protože již původní algoritmus nemusí být správný nebo zdrojový kód může být omylem napsán tak, že původnímu algoritmu neodpovídá. Postupu, při kterém se snažíme odhalit tento druh chyb, říkáme „ladění programu“.

K ladění programu máme k dispozici tyto prostředky:

- krokování programu po jednotlivých krocích (*Step Into*)
- krokování programu s rychlým přeběhem podprogramů (*Step Over*)
- rychlé opuštění podprogramu (*Step Out*)
- automatické krokování s nastavitelnou rychlostí kroků (*Animate*)
- běh programu normální rychlostí (*Run*)
- zastavení běhu programu z režimu *Run* nebo *Animate* (*Halt*)
- vložení bodů zastavení (*Breakpoint*s)
- stopky ke změření doby provádění úseků programu (*Stopwatch*)

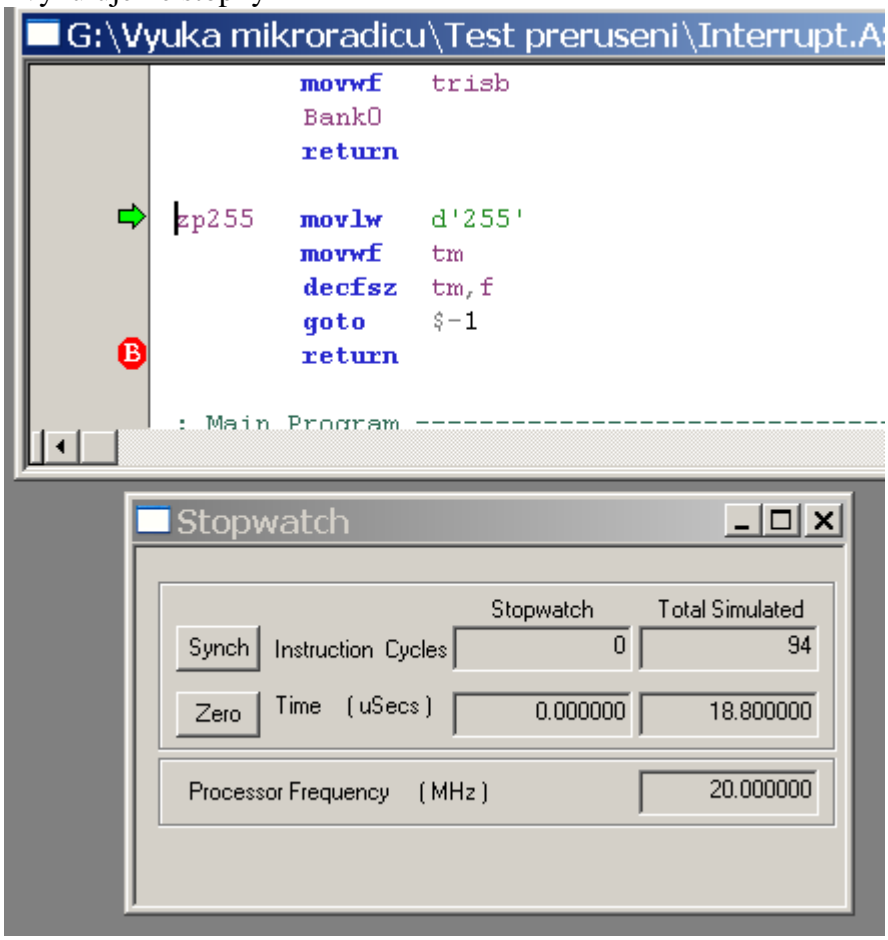
Všechny výše uvedené prostředky nalezneme v okně *Debugger:*

Použití stopek k měření doby trvání úseku programu:

- nastavíme PC kurzor na první instrukci měřeného úseku

- nastavíme bod zastavení programu (Break Point)

- vynulujeme stopky



4. 1. 2. Jazyk symbolických adres

Jistě si vzpomeneme, že každý mikroprocesor pracuje s daty výhradně ve dvojkové soustavě, pouze s hodnotami 0 a 1. Ve dvojkové soustavě musí být tedy zapsány jak instrukce programu, tak i zpracovávaná data. Avšak zadávat mikroprocesoru tyto údaje přímo ve dvojkové soustavě je pro člověka zvyklého na počítání v soustavě desítkové velmi nepohodlné. Zkusme například sečíst dvojkově dvě jednoduchá čísla: 20 + 10 = 30:

20 desítkově = 10100 dvojkově
10 desítkové = 01010 dvojkově
30 desítkově = 11110 dvojkově

Každému je na první pohled jasné, že tudy rozumná cesta nevede. Proto byly vyvinuty softwarové pomůcky, které převedly strohou řeč nul a jedniček do srozumitelnější podoby.

Jednotlivé instrukce jsou vyjádřeny výstižnou slovní zkratkou a data je možno zadávat v číselné soustavě, jaká je v tu chvíli pro programátora nejvýhodnější - tedy i v desítkové. Vznikl tak tzv. **programovací jazyk**, plnící funkci prostředníka mezi člověkem - programátorem a součástkou – mikrořadičem.

Zde je ukázka zápisu programu, sčítajícího výše uvedená čísla:

```
Start     MOVLW     b'00010100'     ;b = binární vyjádření čísla
          ADDLW     b'00001010'
          GOTO      ZOBRAZ
End
```

Jazyku, použitému pro tento zápis, se říká **jazyk symbolických adres** (nebo také assembler).

Jazyk symbolických adres popisuje každou instrukci mikrořadiču zkratkou vycházející z anglického popisu významu jednotlivých instrukcí.

Tak např. výše uvedená instrukce:

MOVLW má původ ve slově Move (přenos) a přenáší datovou konstantu L do registru W
ADDLW je od slova Add (sečíst) a opravdu sečte konstantu L s obsahem registru W
GOTO ZOBRAZ je zkratka slov Go To (jdi na adresu paměti, danou návěštím ZOBRAZ).

Přestože již mluvíme o určitém programovacím jazyce, je třeba si uvědomit, že se nejedná o žádný z tzv. vyšších programovacích jazyků - stále budeme pracovat na té nejnižší úrovni programování – na úrovni tzv. strojového kódu, kdy se pracuje s konkrétními místy programové a datové paměti a provádějí se operace s jednotlivými byty a bity. Tento způsob

programování má však obrovskou výhodu v tom, že máme nad jednotlivými částmi programu nejvyšší možnou kontrolu a výsledný program zabírá v operační paměti mikrořadiču nesrovnatelně méně místa, než by tomu bylo v případě naprogramování v jakémkoliv vyšším programovacím jazyce.

4. 1. 3. Instrukční soubor PIC16F873

Podívejme se nyní blíže na instrukční soubor mikrořadiču PIC16F873.

Instrukční soubor tohoto mikrořadiču obsahuje 33 instrukcí, které si kvůli přehlednosti rozdělíme do následujících skupin:

Aritmetické a logické operace
Instrukce nulování a nastavení
Instrukce přesunu dat
Instrukce podprogramů a přerušení
Instrukce skoků v programu
Zvláštní instrukce

Kompletní seznam všech instrukcí PIC16F873 je uveden v ***Příloze č. 1***. Zde si zatím na několika jednoduchých příkladech zkusíme vysvětlit základní principy.

NOP
Syntax: NOP
Popis: No Operation (žádná operace)

Příklad: NOP
Před instrukcí: PC = adresa
Po instrukci: PC = adresa + 1

Záměrně jsem začal s nejjednodušší instrukcí. NOP - znamená No Operation a přesně to opravdu dělá – tedy nic! K čemu tedy je?

Je třeba si uvědomit, že i když instrukce NOP zdánlivě nic nedělá, tak něco přece jen udělá. Spotřebuje jeden instrukční cyklus mikrořadiče. A o to zde hlavně jde. Zařazením několika instrukcí NOP na správném místě programu se vytvoří sice nepatrné, ale přece jen někdy potřebné časové zpoždění.

MOVWF
Syntax: MOVWF f
Popis: Přesune data z registru w do registru f

```
Příklad:        MOVWF   OPTION_REG
Před instrukcí: OPTION_REG = FFh          ;h = hexadecimální vyjádření čísla
                W = 4Fh
Po instrukci:   OPTION_REG = 4Fh
                W = 4Fh
```

Tato instrukce přesune data z pracovního registru W do registru, označeného v instrukci symbolem f. Namísto tohoto symbolu pak můžeme dosadit jakýkoliv existující registr nebo uživatelskou proměnnou, kterou jsme si předtím vytvořili v oblasti datové paměti.

Jedná se o jednu z nejčastěji používaných instrukcí. Mikrořadiče PIC nižších typových řad bohužel nedisponují instrukcí, která by dokázala přesunout obsah libovolného registru do jiného libovolného registru. Vše se proto musí „přeposlat“ přes pracovní registr W.

MOVF
Syntax: MOVFf,d d = 0,1
Popis: Obsah registru f je přesunut do jiného registru, určeného obsahem proměnné d:
Je-li d = 0, jsou data přesunuta do registru W.
Je-li d = 1, jsou data přesunuta zpět do registru f.
Ovlivňuje: Z

Příklad: MOVF FSR,0
Před instrukcí: W = 00h
FSR = C2h
Po instrukci: W = C2h
FSR = C2h
Z = 0

Příklad: MOVF FSR,1
Před instrukcí: FSR = C2h
Po instrukci: FSR = C2h
Z = 0

Příklad: MOVF FSR,1
Před instrukcí: FSR = 00h
Po instrukci: FSR = 00h
Z = 1

V případě, kdy d = 0, je vše jasné. Instrukce přesune obsah registru FSR do registru W, obsah registru FSR zůstává nezměněn.

Pokud je však d = 1, obsah registru FSR je přesunut do registru FSR – tedy do sebe samého. K čemu je nám to dobré?

Je třeba si všimnout, že nám v příkladu přibyl jeden řádek: Z = 0 (popř. Z = 1). Jedná se o jeden z tzv. stavových bitů, které jsou umístěny v registru STATUS a mají za úkol „hlídat“ výsledky určitých operací, právě prováděných v registrech.

V tomto případě nám stavový bit Z oznamuje, zda je - či není - výsledkem operace nulový obsah registru. Je-li obsah nulový, bit Z se nastaví (1), není-li obsah nulový, bit Z se nuluje (0).

Všimněme si, že v příkladu, kdy obsah registru FSR po provedení přesunu dat není nulový (C2h), je stavový bit Z = 0, avšak v příkladu, kdy je obsah registru FSR po provedené operaci nulový (00h), se bit Z nastavil na hodnotu 1.

Skutečnost, že tato instrukce dokáže při zdánlivě nesmyslném přesunu dat do stejného registru otestovat jeho obsah na nulu, se dá velmi dobře využít např. při zjišťování stavu tlačítek na vstupním portu.

Pokud jsou tlačítka připojena k portu mikrořadiče tak, že v uvolněném stavu je na vstupních pinech úroveň L, pak je jasné, že pokud není stlačeno žádné tlačítko, obsah portu musí být nulový.

ADDLW
Syntax: ADDLW k
Popis: Obsah registru W je přičten k osmibitové konstantě k. Výsledek je umístěn do registru W.
Ovlivňuje: C, DC, Z

Příklad: ADDLW 15h
Před instrukcí: W = 10h
Po instrukci: W = 25h

Tato instrukce provádí aritmetický součet obsahu registru W s konstantou, uvedenou v instrukci. Výsledek je vždy uložen do registru W. Ovlivňuje stavové bity C, DC, Z.

BCF (Bit Clear f)
Syntax: BCF f,b
Popis: Bit „b“ v registru „f“ je vynulován
Ovlivňuje: Neovlivňuje žádný ze stavových registrů

Příklad: BCF FLAG_REG,7 ;sedmý bit registru FLAG_REG bude nulován
;pozor - počítá se odzadu a od nuly – tedy 7 – 0
Před instrukcí: FLAG_REG = 11000111 (binárně)
Po instrukci: FLAG-REG = 01000111

Tato instrukce vynuluje konkrétní bit v konkrétním registru. Podobně pracuje instrukce BSF (Bit Set f), která naopak určený bit nastavuje (1).

GOTO (Go to ... adresa – nepodmíněný skok na určenou adresu)
Syntax: GOTO k ;k = adresa
Popis: Jedná se o nepodmíněný skok na určenou adresu. Program pokračuje od uvedené adresy.
Ovlivňuje: Neovlivňuje žádný ze stavových registrů

Příklad: GOTO 22h
Před instrukcí: PC = adresa
Po instrukci: PC = adresa + 22h

5. Vývojové prostředky

Při návrhu programu pro mikropočítače nebo mikrořadiče jsme postaveni před otázku, jaké vývojové prostředky použít pro zápis programového kódu, pro kontrolu správnosti zápisu a pro konečnou kontrolu funkčnosti - tzv. odladění programu. Pod pojmem vývojové prostředky máme většinou na mysli specializovaný počítačový program a určitý specializovaný hardware, který zajišťuje výše uvedené požadavky.

Existují tři hlavní druhy vývojového prostředí pro návrh programů pro mikroprocesory:
- softwarové simulátory
- hardwarové emulátory

- In Circuit Debuggery (umožňují ladění programu přímo na čipu mikrořadiče)

Softwarový simulátor umožňuje zápis programu, kontrolu syntaxe a odladění programu. Jeho nevýhodou je skutečnost, že nedokáže pracovat v rychlosti skutečného mikroprocesoru, jehož činnost simuluje, protože rychlost zpracování laděného programu velmi závisí na výkonnosti počítače, na kterém simulátor běží. Naopak výhodou softwarového simulátoru mohou být nulové pořizovací náklady; často bývá totiž distribuován jako free software.

Jako příklad kvalitního softwarového simulátoru je možno uvést MPLAB IDE, který zdarma nabízí firma Microchip pro vývoj aplikací s mikrořadiči PIC. Tento software však umí spolupracovat i s hardwarovým emulátorem (např. MPLAB ICE 4000) nebo s tzv. „In Circuit Debuggerem" MPLAB ICD 2, který budeme požívat při výuce.

Hardwarový emulátor je elektronické zařízení, které se připojí k hostitelskému počítači (běžné PC) a plně nahrazuje konkrétní typ mikroprocesoru nebo mikrořadiče. To mimo jiné znamená, že rychlost provádění jednotlivých instrukcí, a tedy i rychlost provádění samotného programu bude stejná jako při použití skutečného mikroprocesoru.

K hardwarovým emulátorům je navíc možno pomocí dodávaného kabelu přímo připojit vyvíjené zařízení, a to buďto již v konečné verzi osazeného plošného spoje, nebo ve formě univerzální desky tištěných spojů nebo kontaktního pole. Emulátor v tu chvíli nahrazuje fyzický mikrořadič (zapojí se přímo do jeho patice).

Hotový, odladěný program se pak ve speciálním programátoru, který většinou bývá součástí emulátoru, uloží do programové paměti mikrořadiču a ten se nakonec vloží do patice vyvíjeného zařízení.

Určitou nevýhodou hardwarových emulátorů je cena zařízení, pohybující se řádově v tisících korun. To je však bohatě vyváženo možností sledovat chování vyvíjeného programu v reálném čase a možností ověření funkce vyvíjené aplikace na zkušebních deskách.

Tato vlastnost se jistě dá dobře využít při vyvíjení nových aplikací v laboratorních nebo dílenských podmínkách. Pro praktickou práce v reálných provozních podmínkách však hardwarový emulátor příliš vhodný není, protože málokdy je možno vyjmout mikrořadič z patice a nahradit jej emulační hlavicí emulátoru – už jen proto, že málokdy se mikrořadiče do patic v průmyslových zařízeních umisťují pro jejich malou spolehlivost. Rovněž je třeba připomenout, že hardwarový emulátor nemůže nikdy napodobit skutečné fyzikální vlastnosti použitého mikrořadiče (vliv teploty, vlhkosti, vibrací, tolerance elektrických parametrů každého kusu atd.). Z těchto důvodů se od používání hardwarových emulátorů postupně upouští a jsou nahrazovány emulací přímo na čipu konkrétního mikrořadiče (In Circuit Debugger).

Typickým představitelem hardwarových emulátorů, dostupných na našem trhu je výrobek firmy ASIX, MU- Alpha, který emuluje tyto typy mikrořadičů Microchip:

PIC16F84/84A,
PIC12C508A,
PIC12C509A,
PIC16C54C,
PIC16C56A,
PIC16C58C

MU-Alpha se pomocí dodávaného kabelu připojí

k paralelnímu portu osobního počítače. Součástí dodávky je dále obslužný software IDEA, napájecí adaptér a kabel s emulační hlavicí pro propojení s vyvíjeným zařízením.

Obr. 13: Hardwarový emulátor MU-Alpha s emulačním kabelem.

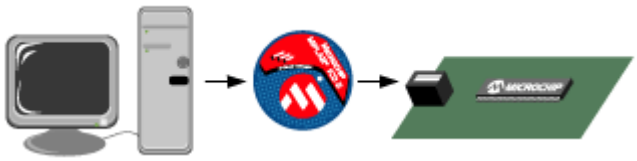
In Circuit Debugger MPLAB® ICD 2

MPLAB® ICD 2 je levný debugger a programátor pro práci s mikrořadiči firmy Microchip. K mikrořadiči se připojuje pomocí speciálního konektoru a k přenosu dat využívá dvou pinů portu. Jeho největší předností je možnost spouštět programy přímo na čipu konkrétního mikrořadiče, krokovat program, nastavovat body přerušení (breakpoint), zobrazit obsah vnitřních registrů mikrořadiče a měnit jejich obsah. Laděný aplikační program je možno pomocí něj ihned naprogramovat do paměti mikrořadiče, takže odpadá nutnost použití externího programátoru.
Protože laděná aplikace běží přímo na čipu mikrořadiče, jsou respektovány jeho fyzikální vlastnosti a vliv okolního prostředí (např. vliv teploty čipu i teploty okolí, rušení, kolísání napájecího napětí apod.).
Celý systém je řízen programem MPLAB od firmy Microchip.

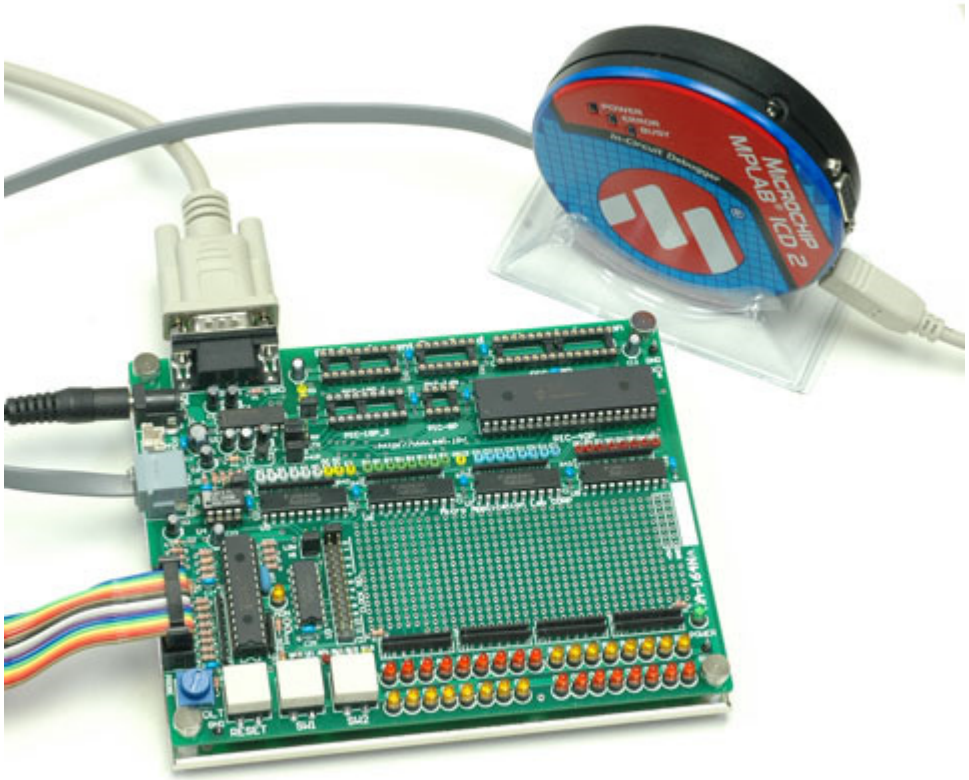
K práci potřebujeme:
- osobní počítač typu PC s operačním systémem Windows XP a nainstalovaným softwarem MPLAB v. 6.0 nebo vyšší
- ICD 2 modul a napájecí adaptér
- USB kabel pro propojení ICD 2 s počítačem
- kabel pro propojení ICD 2 s laděnou aplikací

Modul MPLAB ICD 2

V následující textu bude popsána práce s modulem MMPLAB ICD2 a řídícím software MPLAB IDE. Tuto sestavu budeme používat při výuce.



MODUL MPLAB ICD-2 je USB kabelem propojen s osobním počítačem typu PC a speciálním kabelem, zakončeným koncovkou typu RJ12, s laděnou aplikací. Tou může být např. prototyp zařízení, sestavený na desce nepájivého kontaktního pole, na univerzální desce plošných spojů nebo již hotové elektronické zařízení tak, jak je znázorněno na následujícím obrázku.



Podmínkou pro správnou funkci emulátoru je, aby použitý mikrořadič emulaci uvnitř čipu podporoval (tuto podmínku naštěstí splňuje mnoho novějších typů mikrořadičů).

Protože mikrořadič musí být schopen nějakým způsobem komunikovat s řídícím software, jsou dva z jeho pinů vyhrazeny právě pro tuto komunikaci (obvykle jsou to piny RB6 a RB7). Tyto dva piny nemohou být při ladění programu využity k ničemu jinému. Tato jistá nevýhoda je však do značné míry vyvážena možností volby mikrořadiče s větším počtem portů – výrobce naštěstí nabízí dostatečný počet typů za rozumné ceny.

Abychom byli schopni s mikrořadičem komunikovat, zapisovat do něj program a data, prohlížet a upravovat obsah registrů apod., musíme mít k dispozici vhodný software, který nám tyto operace umožní. Pro tento účel je výhodné využít program MPLAB IDE, který

zdarma poskytuje firma Microchip a v době psaní této učebnice byl k dispozici ve verzi 8.0. Je možno jej stáhnout z www stránek výrobce:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDoc

Před samotným laděním programu nejprve řídící software nahraje do části registrů mikrořadiče krátký pomocný program, který zajistí potřebné funkce pro spouštění a krokování uživatelského programu. Tento pomocný program se po úspěšném odladění uživatelského programu před konečným naprogramování mikrořadiče z jeho paměti automaticky odstraní.

Při první instalaci ICD2 a taky při změně typu mikrořadiče je nutno do paměti ICD2 nahrát operační systém (tato operace může trvat i několik minut). ICD2 si v případě potřeby změnu operačního systému vyžádá.

D Ů L E Ž I T Á I N F O R M A C E !

Při propojování jednotlivých hardwarových komponent je třeba dodržet určité zásady pro připojování napájecího napětí:

- při USB připojení by může být MPLAB ICD 2 napájen z PC, ale cílová aplikace musí mít vlastní zdroj.
- při připojení přes RS-232 musí mít MPLAB ICD 2 vlastní zdroj.
- má-li MPLAB ICD 2 připojen vlastní zdroj, může z něj být napájena cílová aplikace napětím 5V a proudem max. 200mA.
- MPLAB ICD 2 nesmí být napájen z cílové aplikace
- napájecí zdroj by měl být připojen nejdříve k MPLAB ICD 2 a pak teprve k cílové aplikaci.

Podle toho, zda má cílová aplikace vlastní napájecí zdroj anebo zda používá napájení z MPLAB ICD 2, se liší i postup připojování napájecího napětí k těmto komponentům:

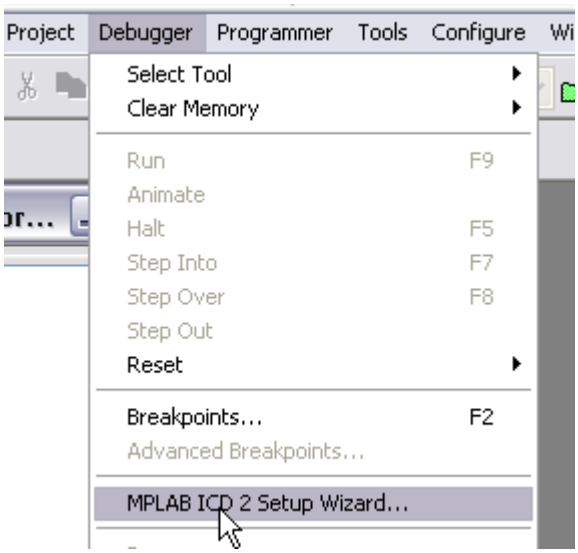
Pokud cílová aplikace nemá vlastní napájecí zdroj (je napájena z MPLAB ICD 2):
- připojte napájecí napětí k MPLAB ICD 2. NEPŘIPOJUJTE NAPÁJENÍ K CÍLOVÉ APLIKACI!
- spusťte MPLAB-IDE
- z menu Debugger zvolte Connect
- po navázání komunikace s MPLAB ICD 2 zvolte Debugger>Settings
- klikněte na záložku Power a ujistěte se, že je zatrženo políčko „Power target circuit from MPLAB ICD 2“.
- klikněte na OK

Pokud má být cílová aplikace napájena z vlastního zdroje, je třeba dodržet tento postup:
- - připojte napájecí napětí k MPLAB ICD 2. NEPŘIPOJUJTE NAPÁJENÍ K CÍLOVÉ APLIKACI!
- spusťte MPLAB-IDE
- z menu Debugger zvolte Connect
- po navázání komunikace s MPLAB ICD 2 zvolte Debugger>Settings

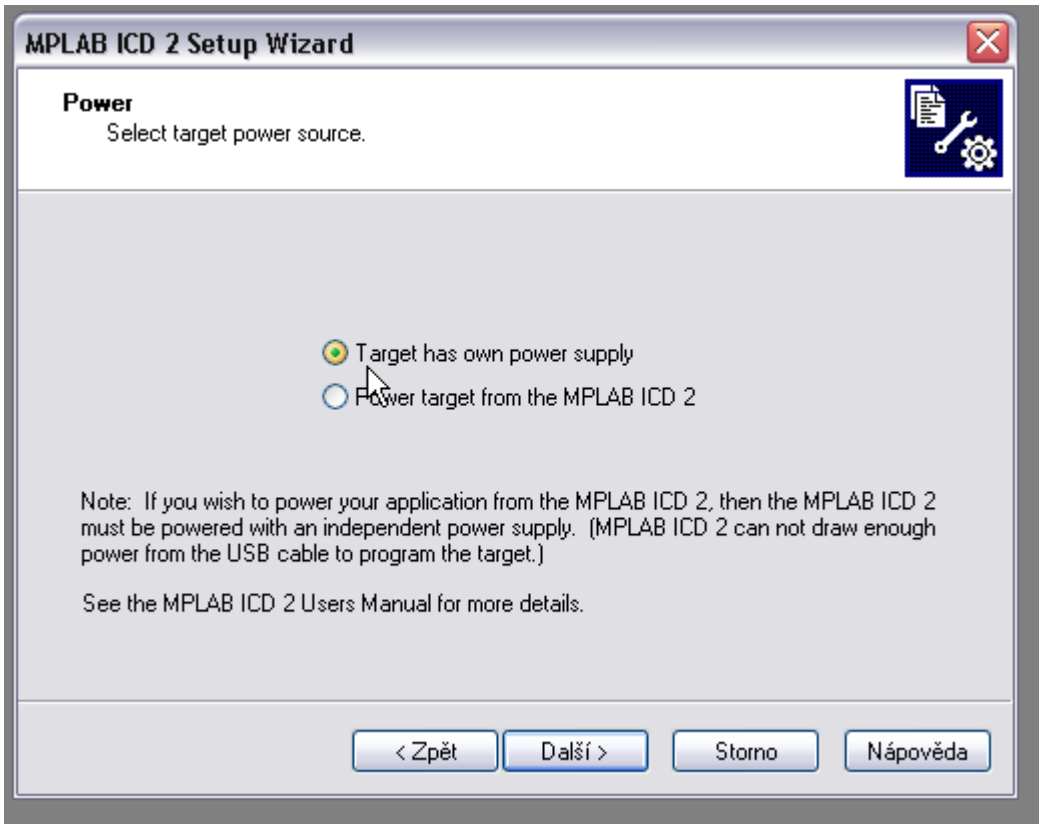
- klikněte na záložku Power a ujistěte se, že není zatrženo políčko „Power target circuit from MPLAB ICD 2“.
- klikněte na OK
- připojte napájení cílové aplikace a pak zvolte Debugger>Connect

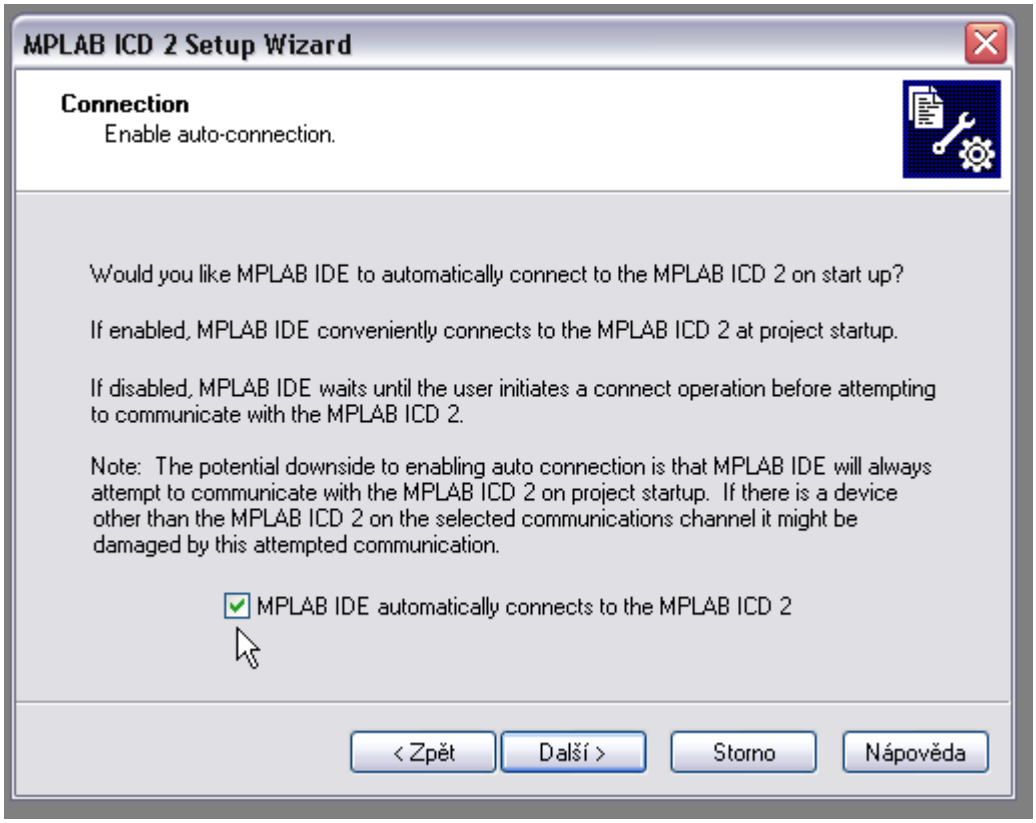
6. Konfigurace MPLAB ICD2

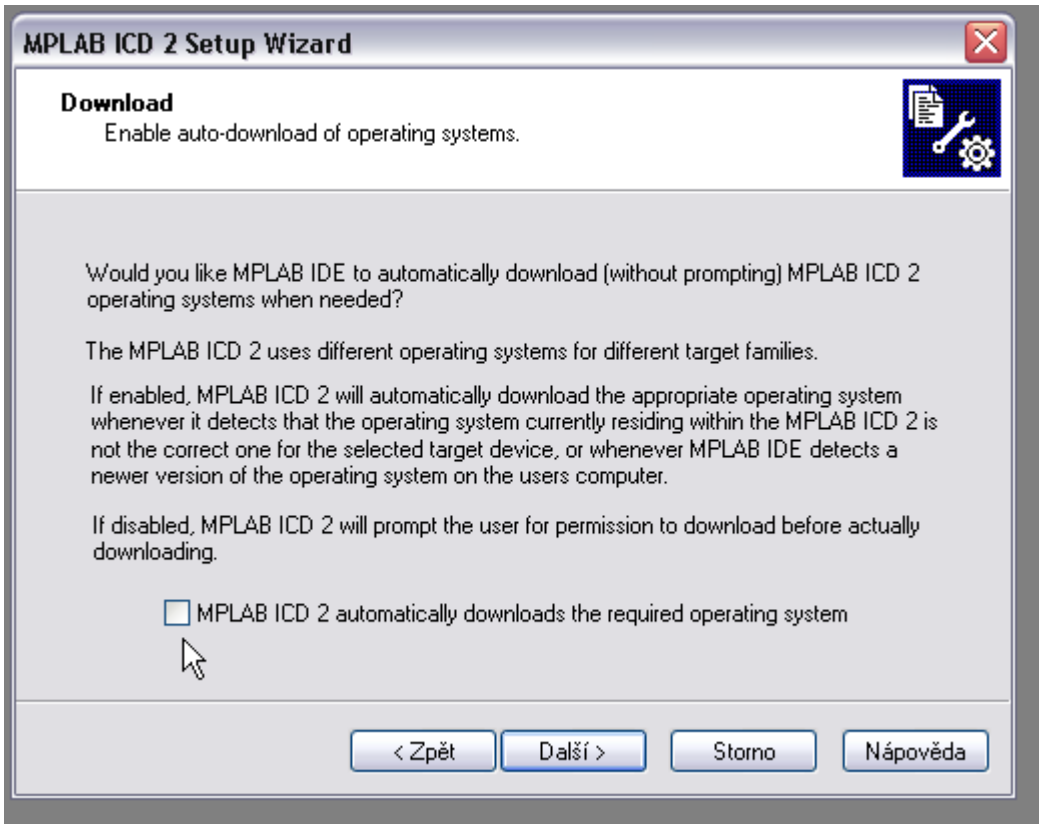
Následující postup je nutný při prvním připojení MPLAB ICD2 nebo při změně jeho konfigurace:

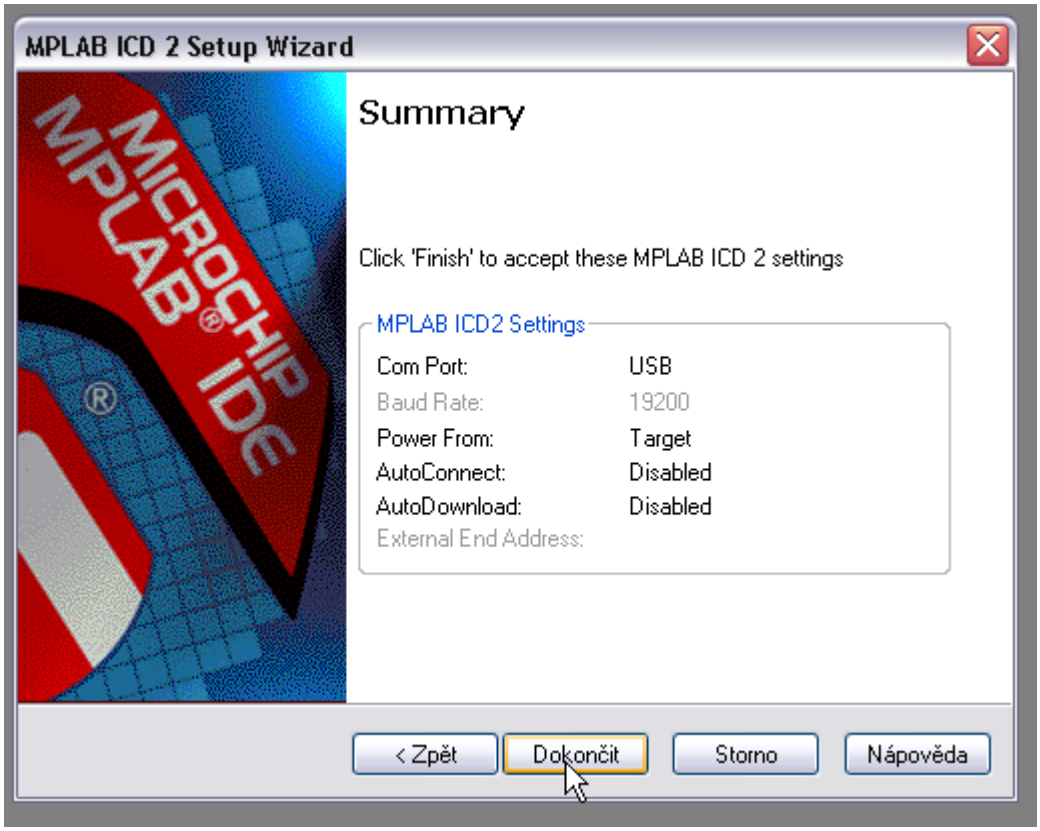


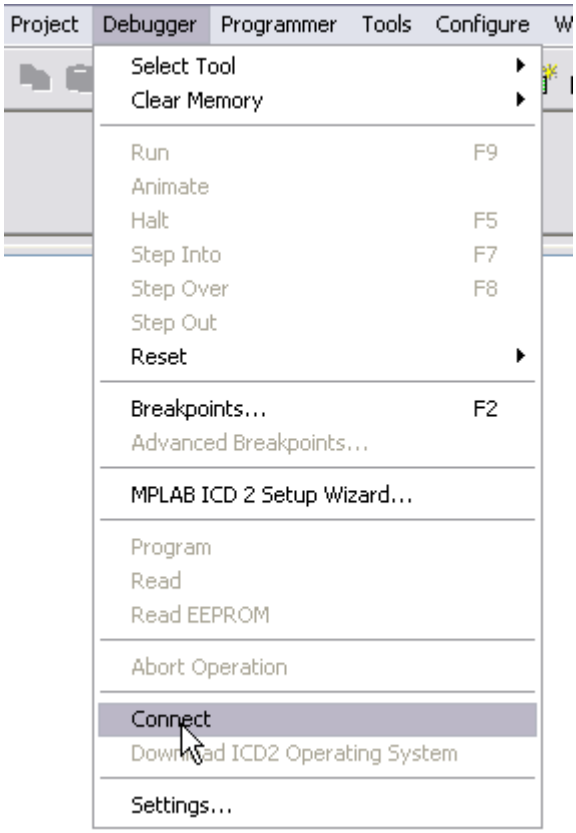

MPLAB ICD 2 Setup Wizard
Communications
Select a communications method.
Com Port
USB
Baud Rate
19200
< Zpět
Další >
Storno
Nápověda


MPLAB ICD 2 Setup Wizard
Power
Select target power source.
Target has own power supply
Power target from the MPLAB ICD 2
Note: If you wish to power your application from the MPLAB ICD 2, then the MPLAB ICD 2 must be powered with an independent power supply. (MPLAB ICD 2 can not draw enough power from the USB cable to program the target.)
See the MPLAB ICD 2 Users Manual for more details.
< Zpět
Další >
Storno
Nápověda

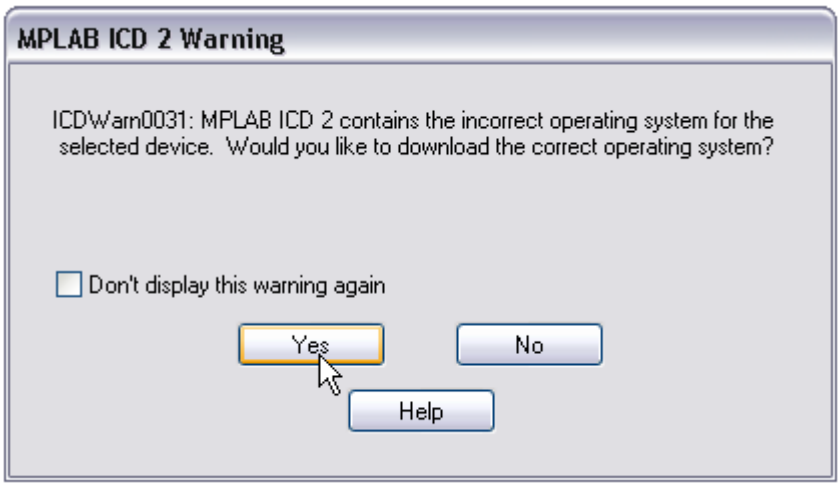

MPLAB ICD 2 Setup Wizard
Connection
Enable auto-connection.
Would you like MPLAB IDE to automatically connect to the MPLAB ICD 2 on start up?
If enabled, MPLAB IDE conveniently connects to the MPLAB ICD 2 at project startup.
If disabled, MPLAB IDE waits until the user initiates a connect operation before attempting to communicate with the MPLAB ICD 2.
Note: The potential downside to enabling auto connection is that MPLAB IDE will always attempt to communicate with the MPLAB ICD 2 on project startup. If there is a device other than the MPLAB ICD 2 on the selected communications channel it might be damaged by this attempted communication.
MPLAB IDE automatically connects to the MPLAB ICD 2
< Zpět
Další >
Storno
Nápověda

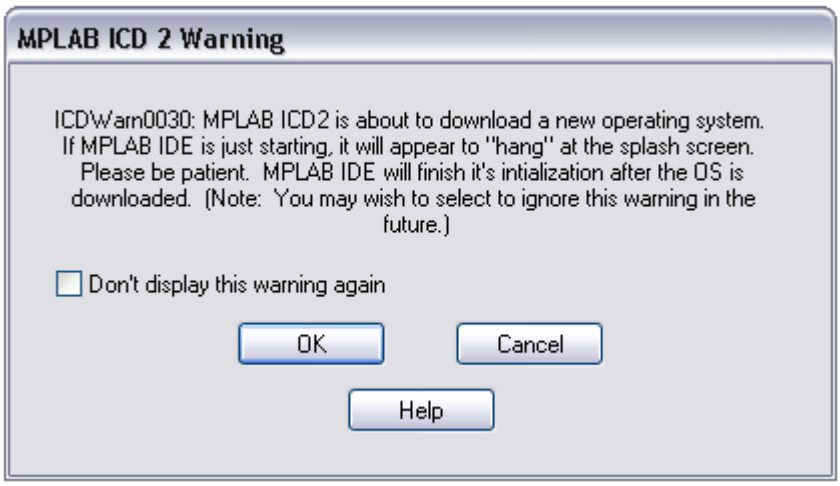

MPLAB ICD 2 Setup Wizard
Download
Enable auto-download of operating systems.
Would you like MPLAB IDE to automatically download (without prompting) MPLAB ICD 2 operating systems when needed?
The MPLAB ICD 2 uses different operating systems for different target families.
If enabled, MPLAB ICD 2 will automatically download the appropriate operating system whenever it detects that the operating system currently residing within the MPLAB ICD 2 is not the correct one for the selected target device, or whenever MPLAB IDE detects a newer version of the operating system on the users computer.
If disabled, MPLAB ICD 2 will prompt the user for permission to download before actually downloading.
MPLAB ICD 2 automatically downloads the required operating system
< Zpět
Další >
Storno
Nápověda


MPLAB ICD 2 Setup Wizard
MICROCHIP
MPLAB® IDE
Summary
Click 'Finish' to accept these MPLAB ICD 2 settings
MPLAB ICD2 Settings
Com Port: USB
Baud Rate: 19200
Power From: Target
AutoConnect: Disabled
AutoDownload: Disabled
External End Address:
< Zpět
Dokončit
Storno
Nápověda



Mikrořadiče firmy Microsoft jsou rozděleny do několika tříd. Každá z těchto tříd vyžaduje nahrání jiného operačního systému do MPLAB ICD2. Rychlost nahrávání závisí na výkonu řídícího PC a může trvat až několik minut. Průběh nahrávání je indikován „běžícím páskem“ na spodní liště Windows.





Nyní je MPLAB ICD2 připraven k použití.

7.

Poznámky k psaní zdrojového textu:

- veškerý text píšeme zásadně bez diakritických českých znaků
- jednotlivé bloky textu (návěští, příkaz, proměnné, poznámka) od sebe pro lepší přehlednost oddělujeme tabelátorem
- text začínající středníkem (;) je považován za poznámku a nebude překládán
- výrazy ***list, config, org, #define, equ, end*** jsou tzv. directiva překladače. Definují konstanty a uživatelské proměnné - nejsou to tedy programové instrukce a nebudou překládány.
- výrazy ***Init, Cteni*** jsou tzv. ***návěští***. Představují v zápisu symbolickou adresu instrukce a při překladu jsou nahrazeny její konkrétní hodnotou. Díky existenci symbolických adres nemusíme mít neustále na paměti přesnou adresu skoku, volání podprogramu, návratu z podprogramu atd., místo toho se obracíme na symbolické adresy - návěští.
- zápis zdrojového textu programu musí vždy končit directivem ***end***

Pozn.: Činnost programu je jasná na první pohled - po úvodní konfiguraci a definicích portů program čte data z ***portu A*** a ihned je předává ***portu B***. Úroveň L, přečtena na vstupním portu při stlačení tlačítka, uzemní LED diodu na výstupním portu a ta se rozsvítí. Program se pak vrací zpět na čtení *portu A* a vše se opakuje v nekonečné smyčce. Když pomineme úvodní konfigurace, vše zajišťují pouhé tři instrukce.

. Pár tipů

7.Literatura a internetové adresy

Jiří Hrbáček: Moderní učebnice programování PIC
Václav Vacek: Učebnice programování PIC
Jiří Hrbáček: komunikace mikrokontrolérů s okolím
David Matoušek: Číslicová technika
Katalogové listy firmy MICROCHIP
http://www.microchip.com
http://www.asix.cz
http://www.gme.cz
http://www.radioplus.cz

8. Závěr

Smyslem této učebnice je podat nejdůležitější informace o problematice mikrořadičů, se zaměřením na typ PIC16F873 a stručný popis vývojového systému MPLAB ICD-2. Učebnice je zaměřena spíše na praktickou stránku věci, tak, aby absolvent byl schopen připojit mikrořadič k jednoduchým periferním obvodům, sestavit a odladit jednoduchý program. Měl by být pro studenty pomůckou během výuky mikrořadičů v celém rozsahu učebního plánu.

Přílohy.

Příloha č. xx: Přehled Special Functions Registers

Zde je uveden pouze stručný přehled SFR a jejich umístění v paměti. Podrobný popis SFR je mimo rozsah této učebnice a je třeba jej vyhledat v katalogových listech obvodu, nejlépe na stránkách výrobce: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2046&redirects=datasheets.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
01h	TMR0	Timer0 Module Register								xxxx xxxx	47
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	29
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	33
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	35
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	36
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
0Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	22
0Dh	PIR2	—	(5)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	24
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	51
11h	TMR2	Timer2 Module Register								0000 0000	55
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	55
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	70, 73
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	67
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	57
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	57
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	58
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	96
19h	TXREG	USART Transmit Data Register								0000 0000	99
1Ah	RCREG	USART Receive Data Register								0000 0000	101
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	57
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	57
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	58
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	116
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	111

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	29
86h	TRISB	PORTB Data Direction Register								1111 1111	31
87h	TRISC	PORTC Data Direction Register								1111 1111	33
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	35
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	37
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
8Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	21
8Dh	PIE2	—	(5)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	23
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	25
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	68
92h	PR2	Timer2 Period Register								1111 1111	55
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	73, 74
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	66
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	95
99h	SPBRG	Baud Rate Generator Register								0000 0000	97
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	—	Unimplemented								—	—
9Dh	—	Unimplemented								—	—
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	116
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	112

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
101h	TMR0	Timer0 Module Register								xxxx xxxx	47
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	26
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
10Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	41
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	41
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						xxxx xxxx	41
10Fh	EEADRH	—	—	—	EEPROM Address Register High Byte					xxxx xxxx	41
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	31
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
18Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	41, 42
18Dh	EECON2	EEPROM Control Register2 (not a physical register)								---- ----	41
18Eh	—	Reserved maintain clear								0000 0000	—
18Fh	—	Reserved maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.
5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.