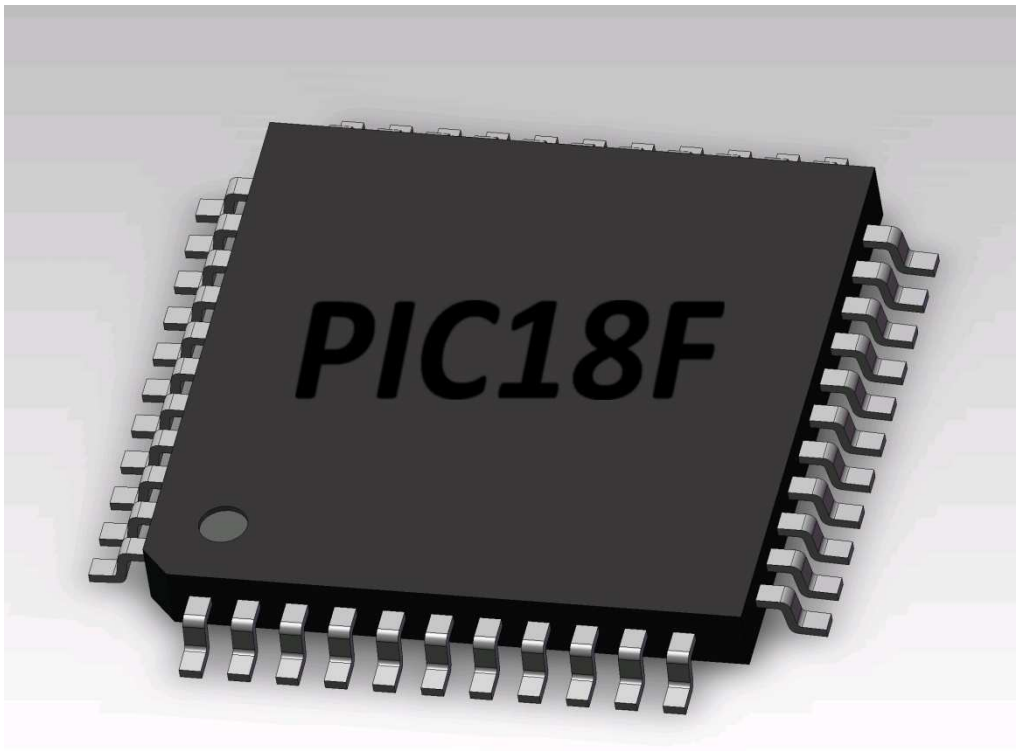
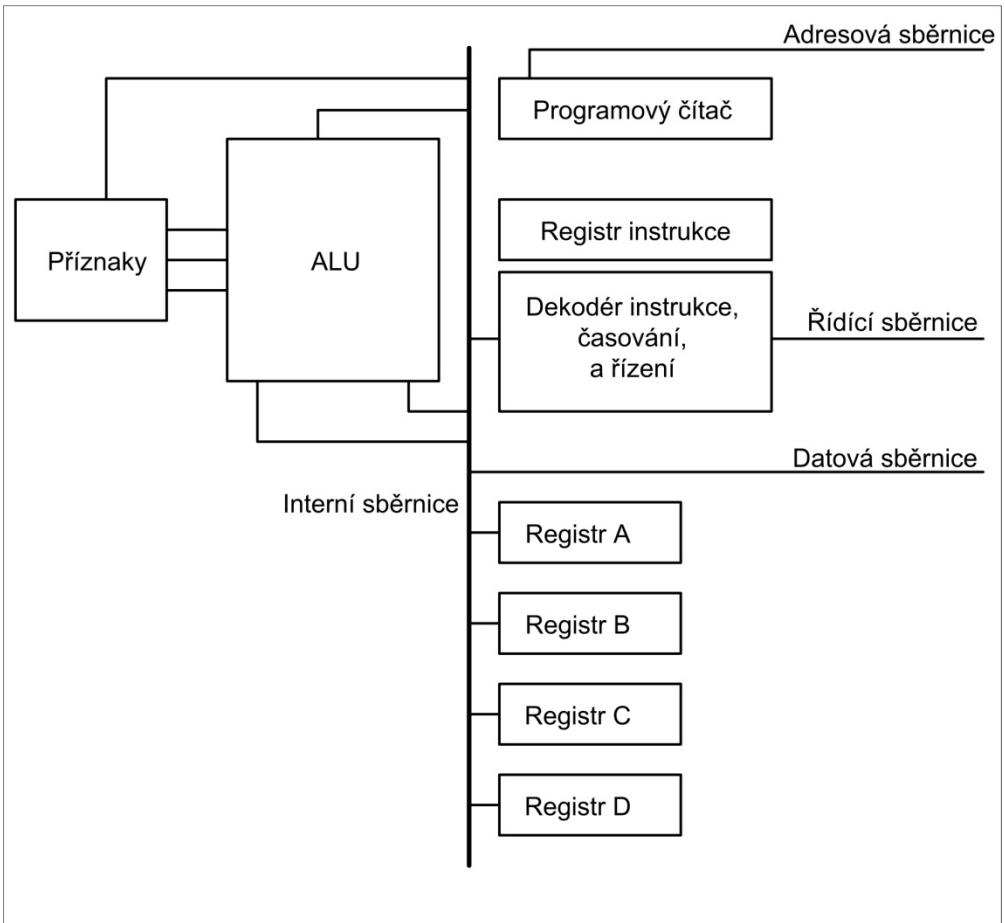
Mikrokontroléry PIC a vestavěné systémy

PIC18 – použití assembleru a jazyka C





Obrázek 0-11. Blokové schéma vnitřního uspořádání CPU

Uvnitř CPU

Program, uložený v paměti, obsahuje instrukce pro centrální jednotku k provedení akce. Akce mohou jednoduše sčítat data, jako jsou mzdová data nebo řídit stroje jako roboty. Funkcí CPU je, aby přenesla tyto instrukce z paměti a vykonala je. K vykonání akcí, jako je načtení a provedení instrukcí jsou procesory vybaveny následujícími zdroji:

1. Čelní místo mezi prostředky centrální jednotky zaujímá počet registrů. Procesor využívá registry pro dočasné ukládání informací. Tyto informace mohou být například dvě hodnoty, které mají být zpracovány, nebo třeba adresy dat stažené z paměti. Registry uvnitř procesoru mohou být 8bitové, 16bitové, 32bitové, nebo 64bitové, v závislosti na centrální jednotce. Obecně platí, že čím více větších registrů, tím lepší CPU. Nevýhodou většího počtu a větších registrů jsou zvýšené náklady na tyto procesory.
2. Centrální jednotka má také blok, který se nazývá ALU (aritmeticko-logická jednotka). Sekce ALU v procesoru je zodpovědná za vykonání aritmetických funkcí, jako je sčítání, odčítání, násobení a dělení a logické funkce, jako je AND, OR a NOT.

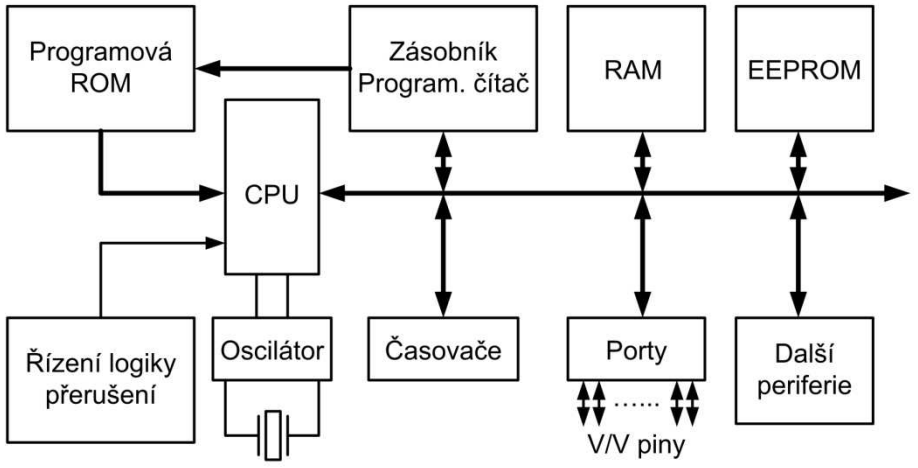
KAPITOLA 1

MIKROKONTROLÉRY PIC: HISTORIE A VLASTNOSTI

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Porovnat mikroprocesory a mikrokontroléry
- Popsat výhody mikrokontrolérů pro některé aplikace
- Vysvětlit koncept vestavěných systémů
- Diskutovat o kritériích pro posouzení mikrokontroléru
- Vysvětlit varianty rychlosti, pouzder, paměti a ceny na jednotku a jak tyto ovlivňují výběr mikrokontroléru
- Porovnat různé členy rodiny PIC
- Srovnat PIC s mikrokontroléry, nabízenými jinými výrobci



Obrázek 1-2. Zjednodušený pohled na mikrokontrolér PIC

Programová ROM v mikrokontrolérech PIC

V mikrokontrolérech slouží ROM k ukládání programů, a proto se nazývá programová nebo kódová ROM. Ačkoli PIC18 má možnost adresovat 2M (megabajty) programového (kódového) prostoru ROM, nemají členové rodiny tolik ROM nainstalováno. Velikost programové ROM, v době psaní tohoto článku, se liší od 4k do 128 kB, v závislosti na členu rodiny. Programová ROM PIC18 je k dispozici v různých typech pamětí, jako jsou Flash, OTP a maskované, všechny odlišené různým označením. Diskusi o různých typech ROM vedeme v Kapitole 14, pokud potřebujete osvěžit svou paměť o těchto důležitých paměťových technologiích. Všimněte si, že ačkoli existují různé charakteristiky PIC18 z hlediska rychlosti a množství implementované RAM/ROM v čipu, jsou vzájemně kompatibilní, pokud se jedná o instrukce. To znamená, že pokud píšete svůj program pro jeden, bude běžet na jiném, bez ohledu na označení čipu. Dále se budeme krátce zabývat programovou ROM rodiny PIC18.

Mikrokontrolér PIC s UV-EPROM

Některé mikrokontroléry PIC používají pro vnitřní programovou ROM technologii UV-EPROM. Pro použití těchto druhů čipů pro vývoj je vyžadován přístup k vypalovačce PROM, stejně jako mazačce UV-EPROM k vymazání obsahu ROM. Okno na čipu UV-EPROM umožňuje UV zářením vymazat ROM. Problém s UV EPROM je ten, že vymazání čipu trvá asi 20 minut, než je možné ho znovu naprogramovat. To vedlo Microchip k zavedení flash verze rodiny PIC. V tomto okamžiku flash naprosto nahradil UV-EPROM. Tabulka 1-2 ukazuje některé členy rodiny PIC18.

KAPITOLA 2

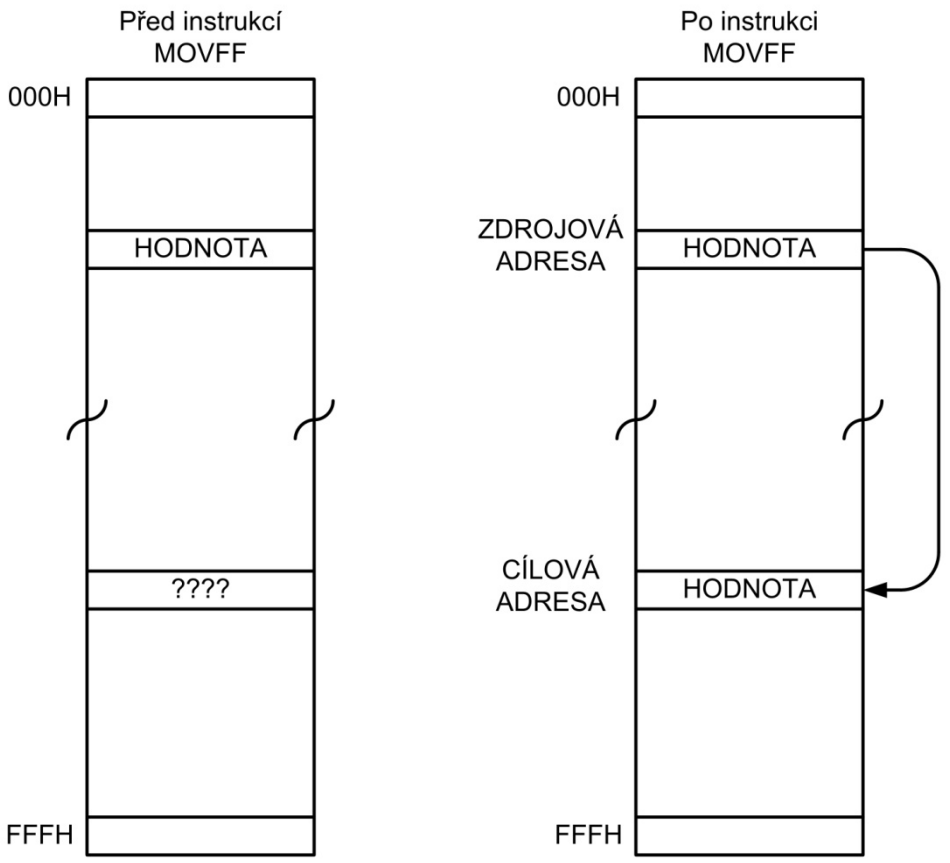
PIC ARCHITEKTURA & PROGRAMOVÁNÍ V ASSEMBLERU

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Prohlížet registry datové RAM mikrokontroléru PIC
- Manipulovat s daty s využitím WREG a instrukcí MOVE
- Vykonávat mikrokontrolérem PIC jednoduché operace jako ADD a MOVE použitím registrů a Access banky
- Objasnit účel stavového registru
- Diskutovat o přidělení prostoru paměti RAM v mikrokontroléru PIC
- Vypsat SFR (speciální funkční registry) mikrokontroléru PIC
- Kódovat jednoduché instrukce jazyka assembleru
- Charakterizovat datové typy a direktivy PIC
- Sestavit a spustit program PIC použitím MPLAB
- Prohlížet programy v paměti ROM PIC
- Objasnit paměťovou mapu ROM PIC
- Podrobně popsat instrukce jazyka assembleru PIC
- Pochopit RICS a harvardskou architekturu mikrokontroléru PIC
- Prohlížet registry PIC a datovou RAM s využitím simulátoru MPLAB

PIC18. Instrukce MOVFF nám umožňuje přenášet data v rámci 4k prostoru datové RAM, aniž by využívala registr WREG. (Viz Obrázek 2-6.) Porovnejte Příklady 2-5 a 2-7.



Obrázek 2-6. Přesunutí dat přímo mezi lokacemi fileReg

Napište program k nepřetržitému získávání dat ze SFR Portu B a odešlete je na SFR Portu C použitím instrukce MOVFF. Porovnejte to s Příkladem 2-5 a vysvětlete rozdíl.

Řešení:

```
AGAIN       MOVFF   PORTB, PORTC   ; kopíruj data z Portu B do Portu C
            GOTO    AGAIN          ; dělej to donekonečna
```

V příkladu 2-5 máme:

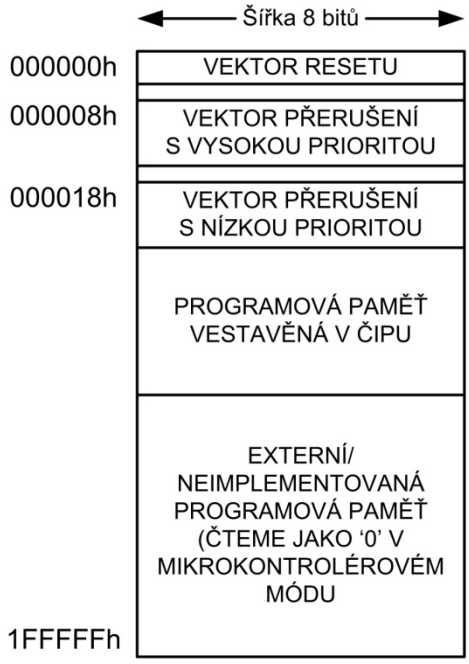
```
AGAIN       MOVF    PORTB, W       ; přenes data z Portu B do WREG
            MOVWF   PORTC          ; pošli to do Portu C
            GOTO    AGAIN          ; dělej to donekonečna
```

Použitím instrukce MOVFF jednoduše kopírujeme data z jednoho místa do druhého. Ale když použijeme WREG, můžeme vykonat aritmetické a logické operace před jejich přesunem.

Příklad 2-7

Když se PIC probudí při zapnutí

Jedna z otázek, na kterou se musíme zeptat u jakéhokoli mikrokontroléru (nebo mikroprocesoru) je: Na jakou adresu se procesor probudí, když je zapnuto napájení? Každý mikroprocesor je jiný. V případě mikrokontrolérů PIC bez ohledu na rodinu a variantu se mikrokontrolér po zapnutí probudí na adresu paměti 0000. Zapnutím máme na mysli použití V_{cc} na pin RESET, jak je popsáno v Kapitole 8. Jinými slovy, když se PIC zapne, PC (programový čítač) má nastavenu hodnotu 00000. To znamená, že se očekává, že první operační kód musí být uložen v ROM na adrese 00000H. Z tohoto důvodu musí být v systémech PIC první operační kód vypálen do lokace 00000H programové paměti ROM, protože to je místo, kde se hledá první instrukce po nabootování. Dosáhneme toho použitím výrazu ORG ve zdrojovém programu, jak už bylo ukázáno dříve. Dále prodiskutujeme krok za krokem akce programového čítače při přenášení a provádění ukázkového programu.



Obrázek 2-11 Prostor v programové ROM PIC18

Umístění kódu v programové paměti ROM

Pro lepší pochopení role programového čítače při načítání a spouštění programu, prozkoumáme akce programového čítače z hlediska každé přenesené a vykonané instrukce. Nejdříve se ještě jednou podíváme do výpisu ukázkového programu a ukážeme si, jak je kód umístěn v ROM čipu PIC. Jak je vidět, operační kód a operand každé instrukce jsou vypsány na levé straně výpisu souboru.

KAPITOLA 3

VĚTVENÍ, VOLÁNÍ A ČASOVÉ ZPOŽĎOVACÍ SMYČKY

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vytvářet kód smyček instrukcemi assembleru PIC
- Vytvářet kód podmíněného větvení instrukcemi assembleru
- Vysvětlit podmínky, které určují každé podmíněné větvení instrukcí
- Vytvářet kód pro nepodmíněný skok použitím instrukce GOTO (dlouhý skok)
- Vypočítat cílové adresy pro instrukce podmíněného větvení
- Kódovat podprogramy PIC
- Popsat zásobník a jeho využití v podprogramech
- Diskutovat o pipeliningu (zřetězené zpracování) v PIC
- Diskutovat o frekvenci krystalu proti instrukčnímu časovému cyklu v PIC
- Kódovat programy PIC pro tvorbu časového zpoždění

```
R1          EQU   0x25
R2          EQU   0x26
COUNT_1     EQU   d'10'
COUNT_2     EQU   d'70'

            MOVLW  0            ; vynuluj WREG
            MOVWF  TRISB        ; Port B jako výstupní
            MOVLW  0x55         ; WREG = 55H
            MOVWF  PORTB        ; PORTB = 55H
            MOVLW  COUNT_1      ; WREG = 10, hodnota vnější smyčky
            MOVWF  R1           ; nahraj 10 do lokace 25H (vnější smyčka – počet opakování)
LOP_1       MOVLW  COUNT_2      ; WREG = 70, hodnota vnitřní smyčky
            MOVWF  R2           ; nahraj 70 do lokace 26H
LOP_2       COMF   PORTB, F     ; doplněk SFR Portu B
            DECF   R2, F        ; dekrement lokace 26 fileReg (vnitřní smyčka)
            BNZ    LOP_2        ; opakuj 70 krát
            DECF   R1, F        ; dekrementuj lokaci 25 fileReg (vnější smyčka)
            BNZ    LOP_1        ; opakuj 10 krát
            END
```

Opakování, prováděné 100 000 krát

Vzhledem k tomu, že ze dvou registrů získáme maximální hodnotu 65025 (255 x 255 = 65025), můžeme použít tři registry, abychom dostali více než 16 milionů (2^{24}) iterací. Následující kód opakuje akci 100 000 krát:

```
R1          EQU        0x1          ; přiřadí lokaci RAM pro R1 – R2
R2          EQU        0x2
R2          EQU        0x3
COUNT_1     EQU        D'100'       ; pevná hodnota pro 100 000 opakování
COUNT_ 2    EQU        D'100'
COUNT _3    EQU        D'10'

            MOVLW      0x55
            MOVWF      PORTB
            MOVLW      COUNT_3
            MOVWF      R3
LOP_3       MOVLW      COUNT_2
            MOVWF      R2
LOP_2       MOVLW      COUNT_ 1
            MOVWF      R1
LOP_1       COMPF      PORTB, F
            DECF       R1 ,F
            BNZ        LOP_1
            DECF       R2, F
            BNZ        LOP_2
            DECF       R3, F
            BNZ        LOP_3
```

KAPITOLA 4

PROGRAMOVÁNÍ V/V PORTŮ PIC

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vypsat všechny porty PIC18
- Popsat duální roli pinů PIC18
- Použít kód assembleru pro vstupy a výstupy portů
- Vysvětlit duální roli Portu A, B, C a D
- Kódovat manipulaci V/V instrukcemi PIC
- Programovat kód pro manipulaci s bity V/V PIC
- Vysvětlit bitovou adresaci portů PIC

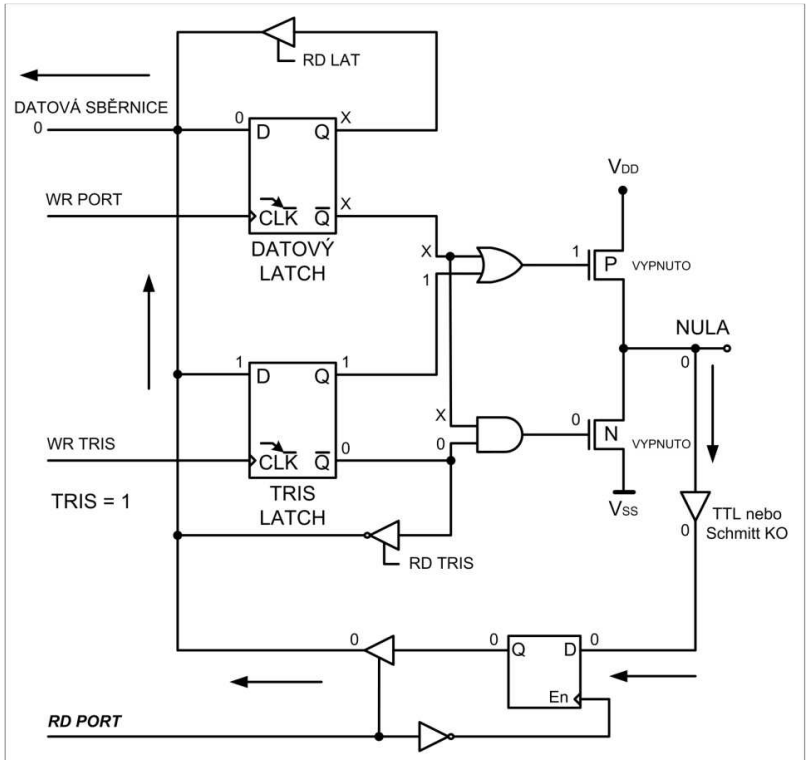
Role registru TRIS ve vstupu dat

Chcete-li nastavit port jako vstupní port, musíte nejprve vložit 1 do registru TRISx tohoto portu a pak přenést (přečíst) data přítomná na pinech. Všimněte si, že 0 znamená směr ven a 1 dovnitř. Je to snadno zapamatovatelné, protože 0 a O (Out) vypadají podobně, stejně jako vypadá 1 a I (Input). Následující kód vezme data, přítomná na pinech portu C a pošle je do portu B po přičtení hodnoty 5 a opakuje donekonečna:

```
        MOVLW  B'00000000'      ; WREG = 0000000 (binárně)
        MOVWF  TRISB            ; Port B jako výstupní port (0 pro O)
        MOVLW  B'11111111'      ; WREG = 11111111 (binárně)
        MOVWF  TRISC            ; Port C jako vstupní port (1 pro I)
L2      MOVF   PORTC,W          ; přesuň data z Portu C do WREG
        ADDLW  5                ; přičti hodnotu
        MOVWF  PORTB            ; pošli to do Portu B
        GOTO   L2               ; pokračuj donekonečna
```

Následuje jiná, efektivnější verze programu:

```
        CLRF   TRISB            ; smaž TRISB (Port B jako výstupní port)
        SETF   TRISC            ; nastav TRISC (Port C jako vstupní port)
L2      MOVF   PORTC,W          ; vezmi data z Portu C
        ADDLW  5                ; přičti hodnotu
        MOVWF  PORTB            ; pošli to do Portu B
        BRA    L2
```



Obrázek 4-5. Vstup (čtení) nuly z pinu PIC18

Čtení jednotlivého bitu

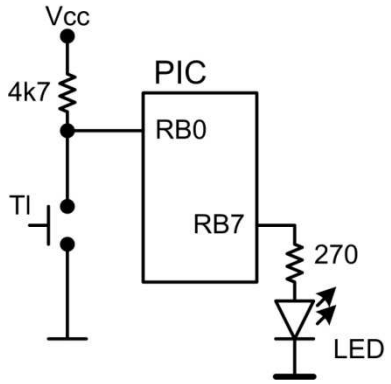
Můžeme také použít testovací bitové instrukce k přečtení stavu jednoho bitu a odeslat do jiného bitu nebo uložit. To je ukázáno v Příkladech 4-8 a 4-9.

Příklad 4-8

Spínač je připojen k pinu RB0 a LED k pinu RB7. Napište program k získání stavu spínače a odeslání hodnoty na LED.

Řešení:

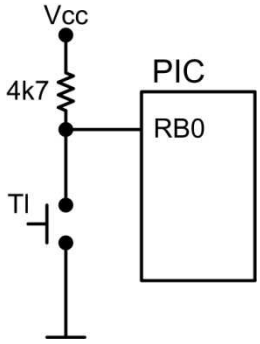
```
        BSF     TRISB, 0        ; nastav RB0 jako vstup
        BCF     TRISB, 7        ; nastav RB7 jako výstup
AGAIN   BTFSS   PORTB, 0        ; test bitu RB0 na hodnotu
                                ; JEDNA
        GOTO    OVER            ; při NULE (BRA je také OK)
        BSF     PORTB, 7
        GOTO    AGAIN           ; můžeme také použít BRA
OVER    BCF     PORTB, 7
        GOTO    AGAIN           ; můžeme také použít BRA
```



Příklad 4-9

Spínač je připojen k pinu RB0. Napište program k získání stavu spínače a uložení hodnoty do D0 lokace 0x20 fileReg.

```
MYBITREG  EQU   0X20            ; nastav lokaci 0x20
          BSF       TRISB, 0    ; nastav RB0 jako vstup
AGAIN     BTFSS   PORTB, 0      ; test bitu RB0 na hodnotu
                                ; JEDNA
          GOTO    OVER          ; při NULE (BRA je také OK)
          BSF     MYBITREG, 0   ; nastav bit 0 z fileReg
          GOTO    AGAIN         ; můžeme také použít BRA
OVER      BCF     MYBITREG, 0   ; nuluj bit 0 z fileReg
          GOTO    AGAIN         ; můžeme také použít BRA
```



Čtení vstupních pinů oproti LATx portu

Při čtení portu čtou některé instrukce stav pinů portu, zatímco jiné čtou stav vnitřního latch (záchytný registr) portu, nazývaného LATx. Proto máme při čtení portů dvě možnosti:

1. Čtení stavu vstupního pinu.
2. Čtení vnitřního latch z registru LAT.

KAPITOLA 5

ARITMETICKÉ A LOGICKÉ INSTRUKCE, PROGRAMY

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Definovat rozsah povolených hodnot čísel bez znaménka v PIC
- Kódovat instrukce sčítání a odčítání pro data bez znaménka
- Provádět sčítání BCD dat
- Kódovat instrukce násobení PIC s daty bez znaménka
- Kódovat programy PIC pro dělení
- Kódovat logické instrukce AND, OR a EX-OR assembleru PIC
- Používat logické instrukce PIC pro manipulaci s bity
- Používat instrukce porovnání a skoku pro řízení programu
- Kódovat rotační instrukce PIC a datovou serializaci
- Vysvětlit systém BCD (binary coded decimal) jako reprezentaci dat
- Kódovat programy PIC pro ASCII a BCD datovou konverzi

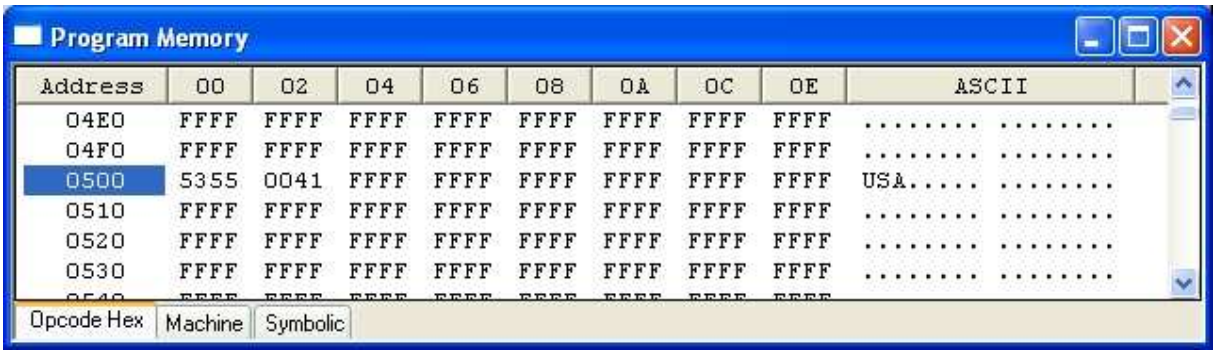
KAPITOLA 6

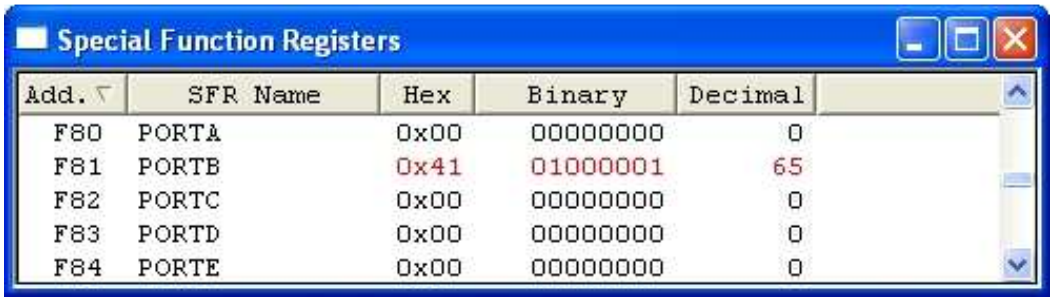
PŘEPÍNÁNÍ BANK, ZPRACOVÁNÍ TABULEK, MAKRA A MODULY

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vypsat všechny adresovací módy mikrokontroléru PIC18
- Rozlišit a porovnat adresovací módy
- Kódovat instrukce assembleru PIC s využitím každého adresovacího módu
- Přistupovat k datové RAM souboru registrů použitím různých adresovacích módů
- Kódovat instrukce PIC18 pro manipulaci s vyhledávací tabulkou
- Přistupovat k pevným datům, uloženým v prostoru programové ROM
- Debatovat o tom, jak vytvořit makra a moduly
- Diskutovat o tom, jak přistupovat k celému 4kB prostoru RAM v PIC18
- Vypsat adresy pro všech 16 bank PIC18
- Debatovat o přístupu ke všem bankám PIC18
- Diskutovat o přepínání bank v PIC18
- Kódovat programy PIC18 pro převody ASCII a BCD dat
- Kódovat programy PIC18 k vytvoření a testování kontrolního bajtu
- Vypsat výhody maker a modulů v programování





Příklad 6-10

Za předpokladu, že programový prostor ROM začíná na 250H a obsahuje "USA", napište program pro odeslání všech znaků do Portu B po jednom bajtu.

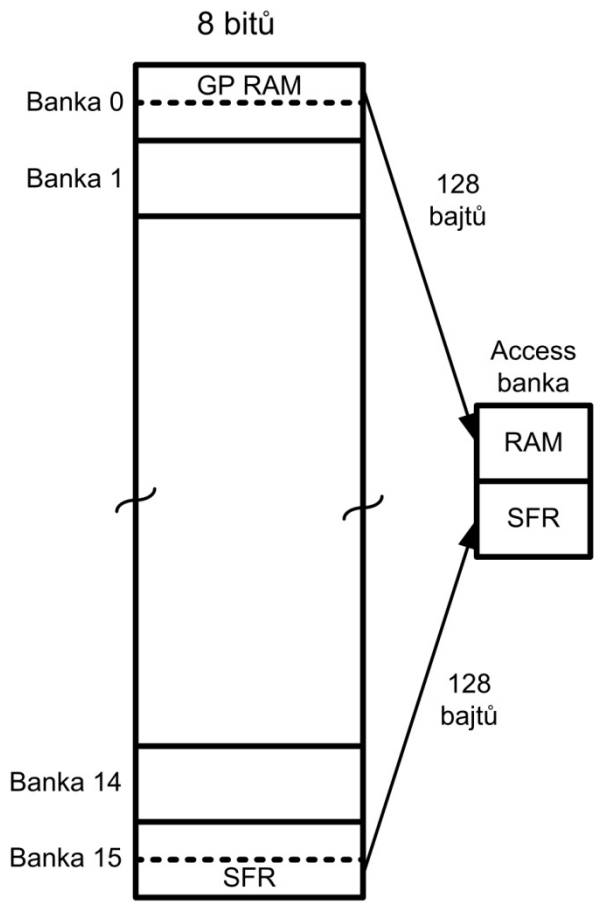
Řešení:

a) Tato metoda používá čítač

```
RCOUNT  EQU     0x20            ; lokace čítače ve fileReg
CNTVAL  EQU     0x3             ; hodnota čítače
        ORG     0000H           ; vypáleno do ROM, začíná na 0
        MOVLW   0x50            ; WREG = 50, adresa dolního bajtu
        MOVWF   TBLPTRL         ; vyhledávací tabulka adresa dolního bajtu
        MOVLW   0x02            ; WREG = 2, adresa horního bajtu
        MOVWF   TBLPTRH         ; vyhledávací tabulka adresa horního bajtu
        MOVLW   CNTVAL          ; WREG = 03, hodnota čítače
        MOVWF   RCOUNT          ; nahraj čítač
        CLRF    TRISB           ; TRISB = 00 (Port B jako výstupní)
B6      TBLRD*                  ; čti bajt tabulky, na který ukazuje TBLPTR
        MOVFF   TABLAT, PORTB   ; pošli to do Portu B
        INCF    TBLPTRL, F      ; inkrementuj ukazatel na další znak
        DECF    RCOUNT, F       ; dekrementuj čítač
        BNZ     B6              ; opakuj, dokud čítač nemá hodnotu nula
HERE    GOTO    HERE            ; zůstaň tady

; data jsou vypálena do prostoru kódové (programové) paměti, začínající na 250H
        ORG     0x250
MYDATA  DB      “USA“
        END
```

GP registr vždy začíná na adrese 000 a jde vzhůru, zatímco SFR začínají na druhém konci 4 kB na adrese FFF a jdou dolů. V současné době PIC používá pouze nejvyšších 128 bajtů banky F (adresy F80H - FFFH) SFR. V budoucnu by se pro SFR mohl začít používat zbytek banky F a mohla by se dokonce používat i banka E, a to v případě, že budou nadále přibývat speciální funkce, zabudované do PIC18. Přestože se počet bajtů v bance F, použitý pro SFR v čipu PIC18 liší v závislosti na funkcích zabudovaných do čipu, SFR vždy začínají na adrese FFF a jdou dolů. Tento bod je třeba zdůraznit. Například v případě access banky je druhá polovina banky F rezervována pro SFR, i když u některých členů rodiny není všech 128 bajtů potřebných, z důvodu omezeného počtu funkcí, podporovaných daným čipem. Nahlédněte do Tabulky 6-5, ať vidíte soubor registrů datové RAM některých čipů PIC18. Všimněte si, že i když můžeme použít libovolné adresovací módy, jako je adresování přímým operandem, přímé nebo nepřímé s přístupem k oblasti GP registrů, k přístupu k SFR registrům používáme pouze přímý adresovací mód. Pro lepší pochopení přepínání banky použijeme čip PIC18F458 k ukázce některých příkladů.



Obrázek 6-3. Registry datové RAM

SEKCE 7.1: DATOVÉ TYPY A ČASOVÉ ZPOŽDĚNÍ V C

V této části nejprve probereme datové typy C pro PIC18 a pak poskytneme kód pro funkce časového zpoždění.

Datové typy C pro PIC18

Jedním z cílů programátorů v C18 je vytvořit menší hexadecimální soubory, takže se vyplatí znovu přezkoumat datové typy C pro C18. Jinými slovy, je dobré pochopit datové typy C pro C18, což může pomoci programátorům vytvářet menší hexadecimální soubory. V této části se zaměříme na konkrétní datové typy C, které jsou nejužitečnější a nejčastěji používané mikrokontroléry PIC18. Tabulka 7-1 ukazuje datové typy a velikosti.

Datový typ	Velikost v bitech	Datový rozsah/Použití
Unsigned char	8	0 až 255
Char	8	- 128 až +127
Unsigned int	16	0 až 65535
Int	16	- 32768 až +32767
Unsigned short	16	0 až 65535
Short	16	- 32768 až +32767
Unsigned short long	24	0 až 16777215
Short long	24	- 8388608 až + 8388607
Unsigned long	32	0 až 4294967295
Long	32	- 2147483648 až + 2147483648

Tabulka 7-1. Některé široce používané datové typy v C18

Unsigned char

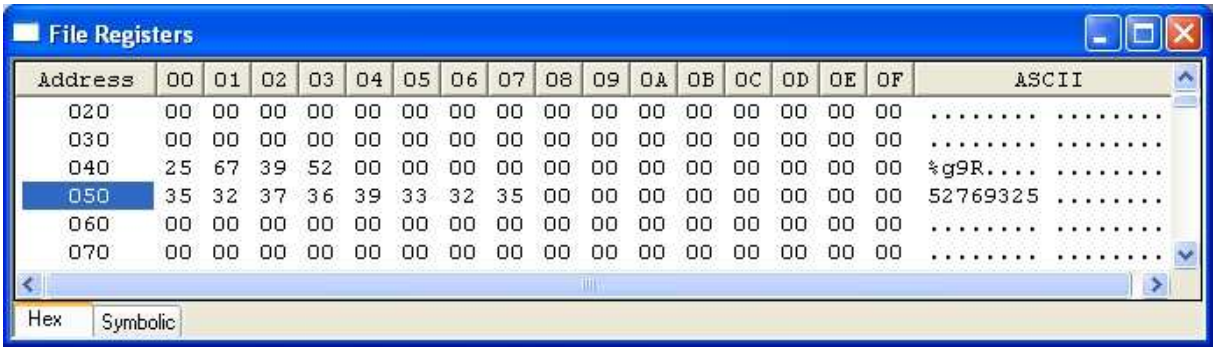
Protože PIC18 je 8bitový mikrokontrolér, je nejpřirozenější volbou pro mnoho aplikací datový typ char (charakter - znak). Unsigned char (char bez znaménka) je 8bitový datový typ, který má hodnotu v rozmezí 0-255 (00 - FFH). Je to jeden z nejvíce používaných datových typů pro PIC18. V mnoha situacích, jako je nastavení hodnoty čítače, kde nepotřebujeme data se znaménkem, bychom měli používat unsigned char, místo signed char. Pamatujte si, že kompilátory C používají signed char jako výchozí, pokud nevložíme klíčové slovo *unsigned* před char (viz Příklad 7-1). Můžeme také použít datový typ unsigned char pro řetězec ASCII znaků, včetně znaků rozšířené sady ASCII. Příklad 7-2 ukazuje řetězec ASCII znaků. Viz Příklad 7-3 pro přepínání portů.

V deklaraci proměnných musíme věnovat velkou pozornost velikosti dat a snažit se používat unsigned char místo int, pokud je to možné. Vzhledem k tomu, že mikroprocesor PIC18 má omezený počet registrů a datových lokací RAM, použití int místo char může vést k větší velikosti hexadecimálního souboru. Takové zneužití datových typů pro překladače, jakým je Microsoft Visual C++ není pro x86 IBM významný problém.

```
        DECF    COUNTREG, F         ; dekrementuj čítač
        BNZ     B2                  ; smyčka, dokud nebude čítač = nula
        RETURN

;-----pošli ASCII data do Portu B
DISPLAY
        CLRF    TRISB               ; nastav PORTB jako výstup (TRISB = FFH)
        MOVLW   CNTVAL1             ; WREG = 8, pošli 8 bajtů dat
        MOVWF   COUNTREG            ; nahraj čítač, čítač = 8
        LFSR    2, ASC_RAM          ; nahraj ukazatel. FSR2 = 50H
B3      MOVF    POSTINC2, W         ; kopíruj RAM do WREG a inkrementuj ukazatel
        MOVWF   PORTB               ; kopíruj WREG do PORTB
        DECF    COUNTREG, F         ; dekrementuj čítač
        BNZ     B3                  ; smyčka, dokud nebude čítač = nula
        RETURN

;---------moje BCD data v programové ROM
                ORG   0x500
MYBYTE          DB    0x25, 0x67, 0x39, 0x52, 0x00
        END
```



Obrázek 6-9. Výsledky Programu 6-2 po proběhnutí

Konverzní program binární (hexadecimální) číslo do ASCII

Mnoho čipů ADC (analogově-digitálních převodníků) poskytuje výstup dat v binárním (hex) tvaru. Pro zobrazení údajů na LCD nebo monitoru počítače ho potřebujeme převést do ASCII. Kód pro převod z binárního čísla do ASCII uvádíme v Programu 6-3. Všimněte si, že podprogram obdrží bajt 8bitových binárních (hex) dat z Portu B a převede jej na desítková čísla a druhý podprogram převádí desítková čísla na ASCII znaky a uloží je. Dolní číslici ukládáme do dolní adresové lokace a horní číslo do vyšší adresové lokace. Toto je konvence malého endianu (tj. dolní bajt do dolní lokace a horní bajt do horní lokace). Všechny produkty PIC18 používají konvenci malého endianu. Podívejte se na algoritmus převodu binárního čísla do ASCII v Kapitole 5.

KAPITOLA 7

PROGRAMOVÁNÍ PIC V C

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vyzkoušet datové typy v C pro PIC18
- Kódovat programy C18 pro časová zpoždění a V/V operace
- Kódovat programy C18 pro bitovou manipulaci vstupů a výstupů
- Kódovat programy C18 pro logické a aritmetické operace
- Kódovat programy C18 pro datové konverze ASCII a BCD
- Kódovat programy C18 pro konverze z binárních (hex) do desítkových formátů
- Kódovat programy C18 pro datovou serializaci
- Pochopit alokace RAM a ROM kompilátoru C18

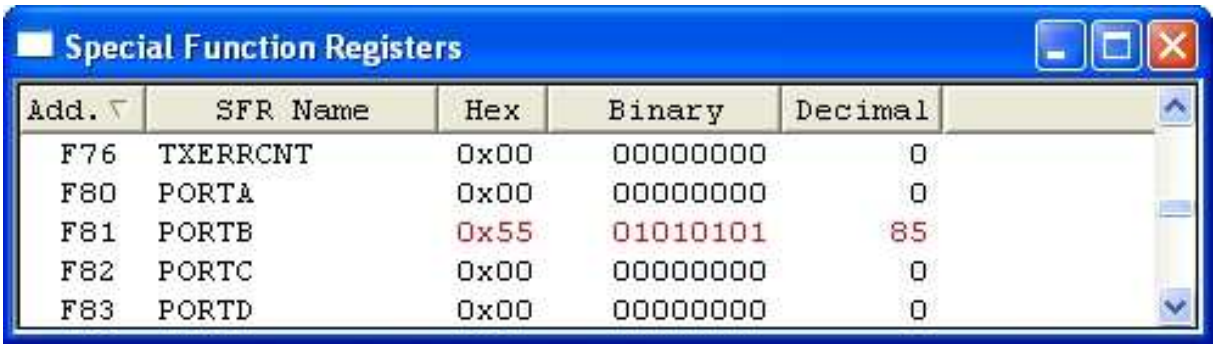
Příklad 7-3

Napište program pro nepřetržité přepínání všech bitů Portu B.

Řešení:

```
// Nepřetržité přepínání PB
#include      <P 18F458. h>
void   main (void)
   {
      TRISB = 0;                  // nastav Port B jako výstupní
      for (; ;)                   // opakuj stále
         {
            PORTB = 0x55;         // 0x  ukazuje, že data jsou v hex (binární)
            PORTB = 0xAA;
         }
   }
```

Nechte běžet výše uvedený program ve vašem simulátoru, abyste viděli nepřetržité přepínání Portu B.



Special Function Registers

Add.	SFR Name	Hex	Binary	Decimal
F76	TXERRCNT	0x00	00000000	0
F80	PORTA	0x00	00000000	0
F81	PORTB	0x55	01010101	85
F82	PORTC	0x00	00000000	0
F83	PORTD	0x00	00000000	0

Signed char

Signed char je 8bitový datový typ, který používá nejvýznamnější bit (D7 z D7 – D0) k reprezentaci - nebo + hodnoty. Ve výsledku máme pouze 7 bitů pro velikost čísla se znaménkem, což nám dává hodnoty -128 až +127. V situacích, kdy + a - je nutné k zobrazení určité kvantity, jako je teplota, je použití datového typu signed char nutné.

Opět si všimněte, že pokud nepoužijeme klíčové slovo *unsigned*, je výchozí hodnota se znaménkem. Z tohoto důvodu bychom se měli držet unsigned char, pokud nepotřebujeme reprezentovat data se znaménkem.

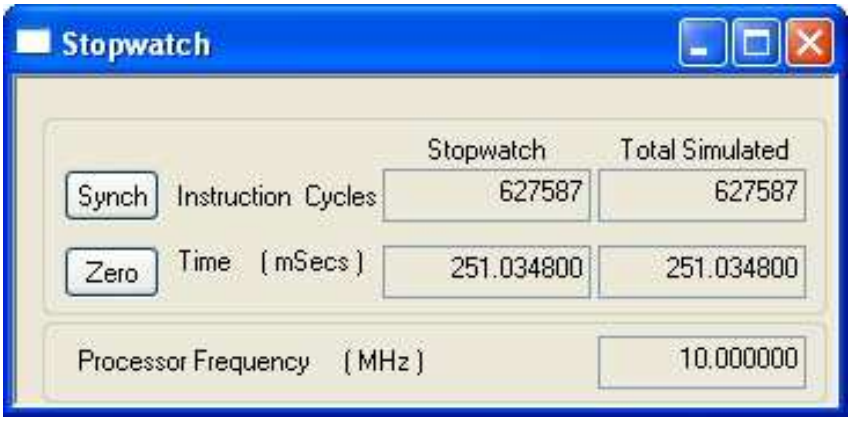
```
void   MSDelay(unsigned     int     itime)
   {
       unsigned     int i; unsigned   char j;
       for (i = 0; i < itime; i++)
         for (j = 0; j < 165; j++);
   }
```

Příklad 7-8

Napište program v C18 pro průběžné přepínání všech bitů Portu C a Portu D se zpožděním 250 ms.

Řešení:

```
#include      <P18F458.h>
void   MSDelay (unsigned     int );
void    main (void)
   {
       TRISC = 0;
       TRISD = 0;                          // nastav Porty C a D jako výstupy
       while (1)                           // jiný způsob, jak to dělat do nekonečna
           {
               PORTC = 0x55;
               PORTD = 0x55;
               MSDelay (250);
               PORTC = 0xAA;
               PORTD = 0xAA;
               MSDelay (250);
           }
   }
void MSDelay (unsigned  int  itime)
   {
       unsigned int  i;   unsigned  char j;
       for (i = 0; i < itime; i++)
       for (j = 0; j < 165; j++);
   }
```



Simulátor MPLAB má funkci stopky, která nám umožňuje prohlížet časové zpoždění ještě před naprogramováním mikrokontroléru.

Obrázek 7-2. Měření časového zpoždění pro Příklad 7-8 použitím MPLAB

NEAR a FAR pro kód

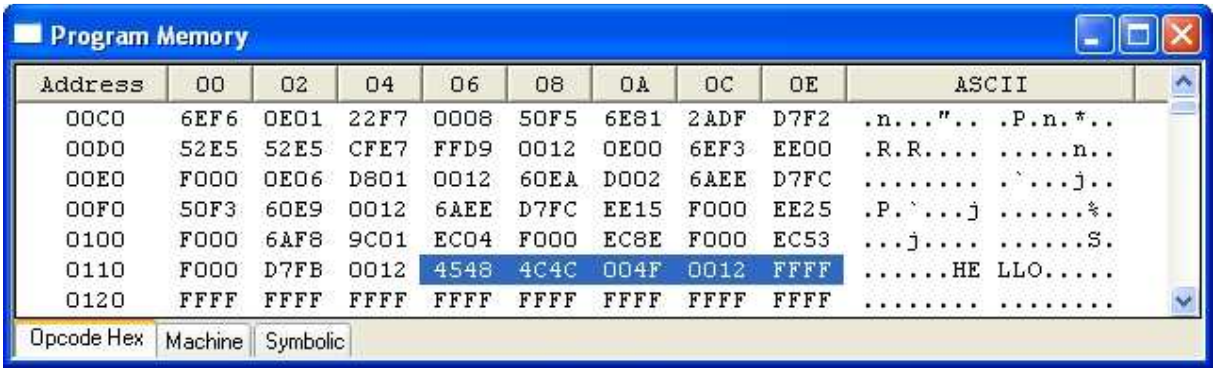
Jak jsme se zmínili dříve, čip mikrokontroléru PIC18 má maximálně 2 MB vestavěného prostoru ROM. Avšak, ne každý člen rodiny přichází s tak velkou vestavěnou programovou pamětí ROM. Některé čipy PIC18 přicházejí s málem, jako jsou 4 kB a některé mají 128 kB programové ROM. Pro efektivnější využití prostoru kódu umožňuje kompilátor C18 použití blízkého a vzdáleného úložiště pomocí kvalifikátorů near a far, které udávají, v jakém regionu by měla být data a kód umístěny. Kvalifikátor near se používá k označení, že datové proměnné programové paměti se nachází v prvních 64 kB programové ROM. Abychom ukázali, že datové proměnné mohou být v programové ROM nalezeny kdekoli v 2 MB prostoru ROM, musíme použít kvalifikátor far. Podívejte se do Tabulky 7-6. Prohlédněte si také Program 7-2A. Všimněte si, že ukládací kvalifikátor far je pro C18 výchozím, pokud nebudeme specifikovat v našem programu jinak.

Ukládací kvalifikátor	ROM
near	V programovém prostoru 0000 – FFFFH (64 kB)
far	V programovém prostoru 000000 – 1FFFFFH (2 MB)

Tabulka 7-6: Použití NEAR a FAR pro ROM

Program 7-2A

```
// Program 7-2A
#include      <P18F458.h >
   near  rom   const      char mydata[] = "HELLO";     // data programové ROM
void   main (void)
   {
       unsigned      char z;
       TRISB = 0;                                       // nastav Port B jako výstupní
       for (z = 0; z < 5; z++)
           PORTB = mydata[z];
   }
```



Všimněte si 4 číslic pro adresy (0000 – FFFF)

Obrázek 7-15. Použití ukládacího kvalifikátoru NEAR, jak je ukázáno v MPLAB

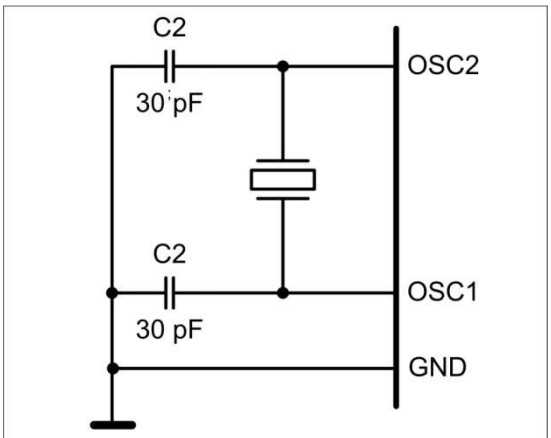
KAPITOLA 8

PŘIPOJENÍ HARDWARE PIC18F A ZAVADĚČE ROM

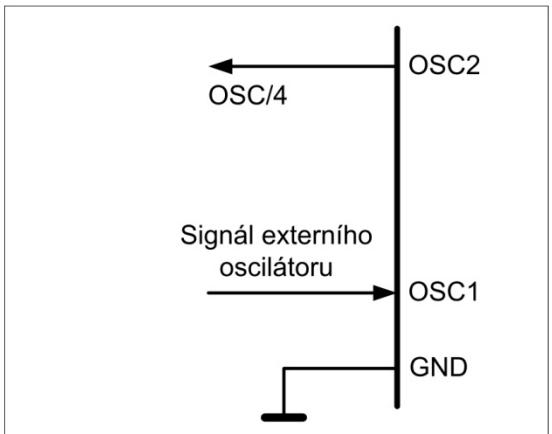
Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vysvětlit funkci pinu reset mikrokontroléru PIC18F
- Ukázat připojení hardware čipu PIC18F
- Ukázat, jak se používá krystalový oscilátor pro hodinový zdroj
- Vysvětlit, jak navrhnout systém, založený na PIC18F
- Vysvětlit roli napěťového resetu brown-out (BOR) v systému resetu
- Vysvětlit roli registrů CONFIG v systému, založeném na PIC18
- Zobrazit návrh PIC Trainer
- Kódovat testovací program v assembleru a C pro zkoušení PIC18
- Ukázat, jak nahrát programy do systému PIC18 s využitím Microchip PICkit2
- Vysvětlit charakteristiky hexadecimálního souboru Intel pro 32bitové a 16bitové adresy



Obrázek 8-6a. Připojení OSC1–OSC2 ke krystalovému oscilátoru



Obrázek 8-6b. Připojení OSC1–OSC2 ke krystalovému oscilátoru

výkon procesoru, jak je popsáno v Příloze C. Pro mnoho obvodů, popsaných v této publikaci používáme HS (high speed). Pokud připojíme piny OSC1 - OSC2 k 10 MHz krystalovému oscilátoru a zvolíme možnost PLLHS, pak procesor pracuje na 40 MHz, protože PLLHS používá pro procesor fázový závěs se čtyřnásobnou rychlostí zdroje hodin. PLLHS má také nejvyšší ztrátový výkon. Všimněte si, že varianta RC (111) je nejlevnější, zatímco volba LP (000) má nejmenší ztrátový výkon.

OSCSEN

Bit OSCSEN (D5) z CONFIG1H umožňuje procesoru přepnout na vnitřní zdroj hodin, který má pevnou frekvenci 32 kHz. Přepnutí zdroje hodin z externího oscilátoru, připojeného k pinům OSC1 a OSC2 na interní zdroj hodin 32 kHz sníží ztrátový výkon na absolutní minimum v mnoha systémech, napájených z baterie. Použitím této volby spolu s volbou LP pro kmitočet krystalu, můžeme snížit spotřebu elektrické energie na rozsah nanowattů.

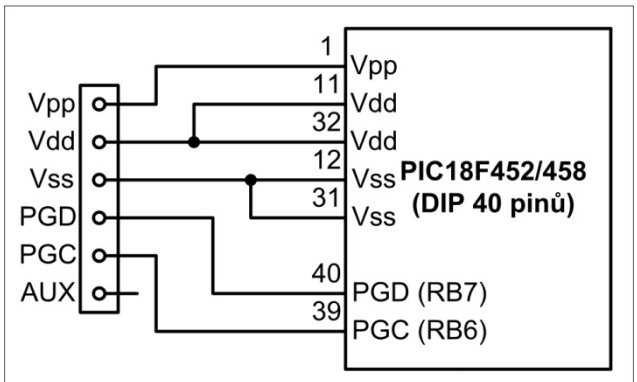
Volba oscilátoru	Frekvence krystalu	Rozsah C1	Rozsah C2
LP	32 kHz	33 pF	33 pF
LP	200 kHz	15 pF	15 pF
XT	200 kHz	47 – 65 pF	47 – 65 pF
XT	1 MHz	15 pF	15 pF
XT	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
HS	8 MHz	15 – 33 pF	15 – 33 pF
HS	20 MHz	15 – 33 pF	15 – 33 pF
HS	25 MHz	15 – 33 pF	15 – 33 pF

Tabulka 8-7: Volba frekvence oscilátoru a rozsahu kondenzátorů pro mikrokontrolér PIC18F458

Všimněte si, že nízkofrekvenční zdroj hodin 32 kHz je společně s externím hodinovým zdrojem připojen k pinům OSC1 a OSC2. Tento sekundární zdroj hodin 32 kHz je nezávislý na zdroji hodin OSC1-OSC2 a bude i nadále poskytovat procesoru hodiny v případě, že se

b. S 16 kB prostoru vestavěné paměti ROM máme 16384 bajtů (16 x 1024 = 16384), které zabírají 0000 – 3FFFH.
c. S 32 kB máme 32768 bajtů (32 x 1024 = 32768). Převeďme 32.768 na hexadecimální tvar a dostaneme 8000H, a proto paměťový prostor je 0000 až 7FFFH.

Připojení výukové desky PIC18



Obrázek 8-17. Připojení PIC18F k PICkit2 pomocí 6pinové hlavice

Poznámka: Připojení, které používá hlavici PICkit2 je použitelné pro všechny rodiny mikrokontrolérů PIC. Rozdíl je pouze v číslech pinů a pojmenování.

Pro výukovou desku, založenou na PIC18, jsme vybrali PIC18F458, protože vám umožní snadno ovíjet spoje a je levná, ale výkonná k použití v práci a doma. Obrázek 8-17 ukazuje připojení systému, založeném na PIC18F s využitím programátoru PICkit2.

PICkit2 je levný programátor, dostupný na internetových stránkách Microchip. Na webových stránkách www.MicroDigitalEd.com ukazujeme schéma připojení pro výukovou desku, založenou na PIC18.

Nahrání programu do výukové desky PIC18

Po sestavení našeho systému, založeného na PIC18, můžeme stáhnout program do výukové desky s použitím programátorské utility PICkit2. Viz Obrázek 8-18. Microchip průběžně aktualizuje MPLAB IDE pro podporu PICkit2 při programování všech mikrokontrolérů PIC.

Testování programu pro PIC18 v assembleru a C

K otestování hardwarového spojení s PIC18, můžeme spustit jednoduchý test, ve kterém všechny bity PORTB neustále přepínáme s určitým zpožděním mezi stavy "ZAP" a "VYP". Viz Programy 8-3 a 8-3C. Všimněte si v těchto programech, že zpoždění je založeno na krystalu 10 MHz. Při vývoji svého programu, můžete použít kostru programu z Obrázků 8-19 a 8-20.

KAPITOLA 9

PROGRAMOVÁNÍ ČASOVAČE PIC18 V ASSEMBLERU A C

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vypsat časovače PIC18 a jim přiřazené registry
- Charakterizovat různé módy časovačů PIC18
- Programovat časovače PIC18 v assembleru a C a generovat časová zpoždění
- Programovat čítače PIC18 v assembleru a C jako čítače událostí

6. Timer0 čítá vzhůru s příchodem každého pulsu hodin z krystalového oscilátoru. Jak časovač čítá, prochází stavy FFF3, FFF4, FFFS, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB a tak dále, až dosáhne FFFFH. Ještě jeden puls hodin, vynuluje se a nastaví příznak Timer0 (TMR0IF = 1). V tomto místě, instrukce "BTFSS INTCON, TMR0IF" obejde instrukci "BRA AGAIN".
7. Timer0 se zastaví instrukcí "BCF T0CON, TMR0ON" a celý proces se opakuje.

Všimněte si, že pro opakování procesu musíme znovu načíst registry TMR0L a TMR0H a znovu spustit časovač.

Příklad 9-4

V Příkladu 9-3 vypočítejte hodnotu časového zpoždění, generovanou časovačem. Předpokládejme, že XTAL = 10 MHz.

Řešení:

Časovač pracuje s Fosc/4, proto máme 10 MHz/4 = 2,5 MHz jako frekvenci časovače. Výsledkem je, že každý tik má periodu T = 1/2,5 MHz = 0,4 µs. Jinými slovy, Timer0 čítá každé 0,4 µs, což nás vede k výslednému zpoždění = počet tiků x 0,4 µs.

Počet tiků čítání do vynulování je FFFFH - FFF2H = 0DH (13 desítkově). Nicméně, přičteme extra puls k 13, protože vynuluje časovač z FFFF na 0 a nastaví příznak TMR0IF. To dává 14 x 0.4 µs = 5.6 µs pro polovinu pulsu. Pro celou periodu časového zpoždění, generovanou časovačem dostaneme T = 2 x 5,6 µs = 11,2 µs.

Příklad 9-5

Vypočítejte frekvenci pravoúhlého průběhu, generovaného pinem PORTB.5.

Řešení:

K získání přesnějšího časování musíme přičíst cykly kvůli instrukcím ve smyčce.

```
                                        Cykly
        BCF     TRISB,5
        MOVLW   0x08
        MOVWF   T0CON
        BCF     INTCON, TMR0IF

HERE    MOVLW   0xFF                    1
        MOVWF   TMR0H                   1
        MOVLW   -D'48'                  1
```

```
        BCF     T2CON, TMR2ON       ; zastav Timer2
HERE    BRA     HERE
```

Příklad 9-39

Použijte předdělič a postdělič, najděte nejdelší dobu zpoždění, kterou může vytvořit použitý Timer2. Předpokládejme, že XTAL = 10 MHz.

Řešení:

Vytvořit nejdelší zpoždění můžeme tím, že PR2 = 255. Když TMR2 dosáhne hodnoty 255 (desítkově), přepne se pin.

```
        BCF     TRISB, 4            ; nastav PORTB4 jako výstupní
        BCF     PORTB, 4            ; vypni PORTB4
        MOVLW   B'01111011'         ; Timer2, předdělič = 16, postdělič = 16
        MOVWF   T2CON               ; nahraj registr T2CON
        MOVLW   0x0                 ; TMR2 = 0
        MOVWF   TMR2                ; nahraj Timer2
HERE    MOVLW   D'255'              ; PR2 = 255, registr periody
        MOVWF   PR2                 ; nahraj PR2
        BCF     PIR1, TMR2IF        ; smaž příznakový bit přerušení časovače
        BSF     T2CON, TMR20N       ;  Timer2
AGAIN   BTFSS   PIR1, TMR2IF        ; monitoruj příznak Timer2
        BRA     AGAIN
        BTG     PORTB, 4            ; zapni PORTB4
        BCF     T2CON, TMR20N       ; stop Timer2
        BRA     HERE
```

Vzhledem k tomu, že XTAL = 10 MHz, TMR2 připočítává každou 0,4 µs. Proto, když máme TMR2H = PR2 = 255, bude RB4 zapínat a vypínat každých 52 ms, protože 255 x 0,4 µs x 16 x 16 = 26.112 ms.

Příklad 9-40

Předpokládejme, že XTAL = 10 MHz, napište program v C18, který zapne pin PORTB4, když TMR2 dosáhne hodnoty 100. Jedná se o opakování Příkladu 9-13 v C18.

Řešení:

```
#include    <p18f4580.h>
#define    mybit   PORTBbits.RB4
void   main (void)
    {
        TRISBbits.TRISB4 = 0;           // PORTB4 jako výstup
        T2CON = 0x0;                    // Timer2, bez předděliče/postděliče
        TMR2 = 0x00;                    // TMR2 = 0
        mybit = 0;                      // PB.4 = 0
        PR2 = 100;                      // nahraj registr periody 2
        T2CONbits.TMR2ON = 1;           // zapni  T0
```

KAPITOLA 10

PROGRAMOVÁNÍ SÉRIOVÉHO PORTU PIC18 V ASSEMBLERU A C

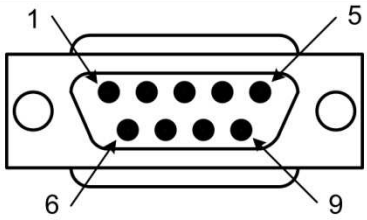
Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Vyhodnotit a porovnat sériovou a paralelní komunikaci
- Vypsat výhody sériové komunikace proti paralelní
- Vysvětlit sériový komunikační protokol
- Porovnat synchronní a asynchronní komunikaci
- Vyhodnotit poloduplexní a plně duplexní přenos
- Vysvětlit proces datového framingu (rámování)
- Popsat rychlost přenosu dat a bps rychlost
- Definovat standard RS232
- Vysvětlit použití čipů MAX232 a MAX233
- Rozhraní PIC18 s konektorem RS232
- Pohovořit o přenosové rychlosti PIC18
- Popsat vlastnosti sériové komunikace PIC18
- Popsat základní registry, používané v sériové komunikaci PIC18
- Naprogramovat sériový port PIC18 v assembleru a C

protože přijímací zařízení nemusí mít v sériovém přenosu dat žádný prostor pro data, ale musí existovat způsob, jak informovat odesílatele, aby zastavil odesílání dat. Pro signály řízení toku dat se používá mnoho pinů konektoru RS-232. Jejich popis je uveden níže jen jako odkaz a můžeme je vynechat, protože nejsou podporovány čipem UART PIC18.

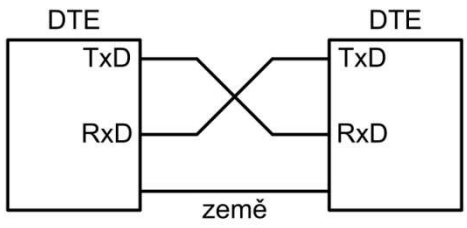
1. DTR (data terminal ready). Pokud se terminál (nebo PC COM port) zapne, po ukončení auto-testu vyšle signál DTR, což znamená, že je připraven pro komunikaci. Pokud je něco v nepořádku s portem COM, tento signál nebude aktivován. Jedná se o signál, aktivní v dolní úrovni a může být použit k informování modemu, že je počítač zapnutý. Jedná se o výstupní pin z DTE (port COM PC) a vstup do modemu.
2. DSR (data set ready). Když je DCE (modem), zapnut a prošel auto-test, potvrdí DSR, že je připraven komunikovat. Je to výstup z modemu (DCE) a vstup do PC (DTE). Jedná se o signál aktivní v nule. Pokud se z nějakého důvodu nemůže modem připojit k telefonu, zůstává neaktivní a indikuje do PC (nebo terminálu), že nemůže přijmout nebo odeslat data.
3. RTS (request to send). Když má zařízení DTE (jako např. PC) vysílat bajt, potvrdí RTS signálem modemu, že má bajt dat k přenosu. RTS je výstup, aktivní v nule z DTE a vstupuje do modemu.
4. CTS (clear to send). Jako odpověď na RTS, pokud má modem prostor pro uložení přijímaných dat, vyšle signál CTS na DTE (PC), což znamená, že nyní může přijímat data. Tento vstupní signál DTE je použit k zahájení vysílání.
5. DCD (data carrier detect). Modem potvrdí signálem DCD a informuje DTE (PC), že detekoval platnou nosnou, a že je vytvořen kontakt mezi ním a dalším modemem. Proto je DCD výstupem z modemu a vstupem pro PC (DTE).
6. RI (ring indicator). Výstup z modemu (DCE) a vstup do PC (DTE) znamená, že zvoní telefon. RI se zapíná a vypíná v synchronizaci s vyzváněcím tónem. Ze šesti signálů řízení toku dat (handshake) je nejméně používán, protože modemy se nestarají o zvedání telefonu. Pokud je však v daném systému PC odpovědný za odpovídání na telefonování, lze tento signál použít.



Obrázek 10-5. DB9 9pinový konektor

Pin	Popis
1	Detekována nosná (DCD)
2	Přijímaná data (RxD)
3	Vysílaná data (TxD)
4	Datový terminál připraven (DTR)
5	Signálová země (GND)
6	Datová sada připravena (DSR)
7	Požadavek na vysílání (RTS)
8	Povolení k vysílání (CTS)
9	Indikátor zvonku (RI)

Tabulka 10-2: Signály DB-9 IBM PC



Obrázek 10-6. Zapojení nulového modemu

1. Do registru TXREG je zapsán bajt znaku, který má být vysílán.
2. Příznak TXIF je interně nastaven do 1 a ukazuje, že TXREG obsahuje bajt a nebude přijímat další bajt, dokud se stávající neodešle.
3. Registr TSR (posuvný registr vysílání) čte bajt z TXREG a začne přenášet bajt, počínaje bitem start.
4. TXIF se maže a ukazuje, že poslední bajt odešel a TXREG je připraven přijmout další bajt.
5. 8bitový znak se přenáší bit po bitu.
6. Na základě sledování příznaku TXIF máme jistotu, že jsme nepřetížili registr TXREG. Pokud zapíšeme další bajt do registru TXREG před tím, než TSR přenesl poslední, může dojít ke ztrátě starého bajtu před jeho odesláním.

Z výše uvedené diskuze jsme došli k závěru, že kontrolou příznakového bitu TXIF víme, zda je či není PIC18 připraven k přenosu dalšího bajtu. Příznakový bit TXIF může být ověřen instrukcí "BTFSS PIR1, TXIF", nebo můžeme použít přerušení, jak uvidíme v Kapitole 11. V Kapitole 11 si ukážeme, jak využít přerušení k sériovému přenosu dat a vyhnout se úvazku mikrokontroléru na instrukce jako je "BTFSS PIR1, TXIF".

Programování PIC18 k sériového příjmu dat

Když chceme naprogramovat PIC18 pro sériový příjem znakových bajtů, musíme splnit následující kroky:

1. Do registru RCSTA vložíme hodnotu 90H, abychom umožnili trvalý příjem v 8bitovém datovém formátu.
2. Do registru TXSTA je vložena hodnota 00H, která vybere nízkou přenosovou rychlost.
3. Do registru SPBRG je natažena některá z hodnot Tabulky 10-4, pro nastavení přenosové rychlosti (za předpokladu, že XTAL = 10 MHz).
4. Nastavíme pin RX z PORTC (RC7) jako vstup pro data, přicházející do PIC18.
5. Příznakový bit RCIF registru PIR1 je monitorován, zda je rovný 1, abychom viděli, zda už byl přijat celý znak.
6. Po nastavení RCIF obsahuje registr RCREG bajt. Jeho obsah přestěhujeme na bezpečné místo.
7. Pro přijetí dalšího znaku přejděte ke Kroku 5.

Příklad 10-4 ukazuje kódování výše uvedených kroků.

Příklad 10-4

Naprogramujte PIC18 pro sériový příjem bajtů dat a vložte je do PORTB. Nastavte přenosovou rychlost na 9600, 8bitová data a 1 stop bit.

KAPITOLA 11

PROGRAMOVÁNÍ PŘERUŠENÍ V ASSEMBLERU A V C

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Srovnat a porovnat přerušení oproti dotazování (polling)
- Vysvětlit účel ISR (interrupt service routine = rutina obsluhy přerušení)
- Vypsat všechna hlavní přerušení PIC18
- Vysvětlit účel tabulky vektorů přerušení
- Povolit nebo zakázat přerušení PIC18
- Naprogramovat časovače PIC18 za pomocí přerušení
- Popsat vnější hardwarová přerušení PIC18
- Naprogramovat sériovou komunikaci PIC18, založenou na přerušení
- Definovat prioritu přerušení PIC18
- Naprogramovat přerušení PIC18 v jazyce C

```
        MOVLW  0x00                ; TMR0H = 00, horní bajt
        MOVWF  TMR0H                ; nahraj Timer0, horní bajt
        BCF    INTCON, TMR0IF       ; smaž příznakový bit přerušení časovače
        GOTO   CHK_INT
; ------------------------- ISR pro Timer2
T1_ISR
        ORG    300H
        BTG    PORTB,6              ; přehoď PORTC.6
        MOVLW  D' 255'              ; TMR1H = 255
        MOVWF  TMRIH                ; nahraj Timer1, horní bajt
        MOVLW  -D'200'              ; TMR1L = 0
        MOVWF  TMRIL                ; nahraj Timer1, dolní bajt
        BCF    PIR1, TMR1IF         ; smaž příznakový bit přerušení Timer1
        GOTO   CHK_INT
        END
```

Všimněte si, že v Programech 11-2 a 11-3 používáme instrukci "GOTO CHK_INT" místo RETFIE jako poslední instrukci ISR. To je proto, že kontrolujeme, aby nedošlo k aktivaci vícenásobného přerušení.

Programování přerušení PIC18 v C, použitím kompilátoru C18

V Kapitole 7 jsme probrali použití direktivy "#pragma code" kompilátorem C18 k umístění kódu na konkrétní adresu ROM. Vzhledem k tomu, že C18 neukládá ISR do tabulky vektorů přerušení automaticky, musíme použít instrukci assembleru GOTO pro přenesení řízení vektoru přerušení na ISR. To se provede následovně:

```
#pragma code high_vector =0x0008    // lokace přerušení s vysokou prioritou
 void My_HiVect_Int (void)
 {
 asm
 GOTO my_isr
_endasm
 }
 #pragma code                       // Konec kódu
```

Teď jsme přesměrovali z adresové lokace 00008 do jiného programu k nalezení zdroje přerušení a konečně ISR. To se provádí pomocí klíčového slova **interrupt** takto:

```
#pragma interrupt my_isr            // interrupt je rezervované klíčové slovo
 void my_isr (void)                 // použité pro přerušení s vysokou prioritou
 {

 // C18 sem umisťuje RETFIE automaticky kvůli
 // klíčovému slovu interrupt
 }
```

```
    }
// -------------  ISR pro přijímač
void    RC_ISR (void)
    {
        PORTB = RCREG;
        RCSTAbits.CREN = 0;                     // smaž CREN pro odstranění jakýchkoli chyb
        RCSTAbits.CREN = 1;                     // nastav CREN pro kontinuální příjem
    }
```

Přerušení uvnitř přerušení

Co se stane, když PIC18 vykonává ISR patřičné přerušení a je aktivováno další přerušení? V takových případech může přerušení s vysokou prioritou přerušit přerušení s nízkou prioritou přerušení. Jedná se o přerušení uvnitř přerušení. V PIC18 může být přerušení s nízkou prioritou přerušení přerušeno přerušením s vyšší prioritou, ale ne dalším přerušením s nízkou prioritou. Přestože jsou všechna přerušení zaznamenána a interně udržována, nemůže přerušení s nízkou prioritou získat okamžitou pozornost procesoru, dokud PIC18 nedokončí obsluhu všech přerušení s vysokou prioritou. Bity GIE (který je také nazýván GIEH) a GIEL hrají důležitou roli v procesu přerušení uvnitř přerušení. Co se týká konceptu přerušení uvnitř přerušení, musí být zdůrazněny následující body:

1. Když je přerušení s vysokou prioritou nasměrováno na adresu 0008H je bit GIE zakázán (GIEH = 0), čímž blokuje další přicházející přerušení (s nízkou nebo vysokou prioritou). Instrukce RETFIE na konci ISR automaticky povolí GIE (GIE = 1), což znovu umožňuje reagovat na přerušení. Pokud chceme umožnit reagovat na jiné přerušení s vysokou prioritou v průběhu provádění aktuálního ISR, pak musíme nastavit GIE = 1 na začátku aktuálně prováděného ISR.
2. Když je přerušení s nízkou prioritou nasměrováno na adresu 0018H, je bit GIEL zakázán (GIEL = 0), čímž blokuje další přicházející přerušení s nízkou prioritou. Instrukce RETFIE na konci ISR automaticky povolí GIEL (GIEL = 1), což dovolí opět reagovat na přerušení s nízkou prioritou. Všimněte si, že přerušení s nízkou prioritou nemůže blokovat příchod přerušení s vysokou prioritou během vykonávání stávajícího ISR s nízkou prioritou, protože GIEH je stále nastaven do horní úrovně (GIEH = 1).
3. Když mají dvě nebo více přerušení stejnou prioritu. V tomto případě jsou obsluhovány podle postupu, podle kterého je program kontroluje v tabulce vektorů přerušení. V této kapitole jsme viděli mnoho příkladů. Podívejte se na Příklad 11-3.

Rychlé ukládání kontextu v přepínání úloh

V mnoha aplikacích, jako jsou multitaskingové real-timové operační systémy (RTOS), vezme procesor v daném času jednu úlohu (práci nebo proces) a provede ji dřív, než se přesune na další. Při provádění každé úlohy, která se často organizuje jako obslužná rutina přerušení, je kritický rychlý přístup ke všem zdrojům procesoru při včasném plnění úloh. Ve starších

KAPITOLA 12

ROZHRANÍ LCD A KLÁVESNICE

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Popsat funkce pinů typického LCD
- Vypsat instrukce příkazových kódů pro programování LCD
- Rozhraní LCD a PIC18
- Programovat LCD v assembleru a C
- Vysvětlit základní operace s klávesnicí
- Popsat stisk klávesy a mechanismus detekce
- Rozhraní klávesnice 4x4 pro PIC18 s využitím C a assembleru

```
        for (j = 0; j < 135; j++);
    }
```

Program 12-3C

```
// Program 12-3C:  Program 12-3 ve verzi C Zobrazení dat in ROM
#include      <P18F458.h>
#define       ldata  PORTD              // PORTD = datové piny LCD (Obrázek 12-2)
#define       rs     PORTBbits.RB0      // rs = PORTB.0
#define       rw     PORTBbits.RB1      // rw = PORTB.1
#define       en     PORTBbits.RB2      // en = PORTB.2

#pragma   romdata   mycom = 0x300  // příkaz do ROM na adresu 0x300
far     rom  const   char  mycom[] = {0x0E, 0x01, 0x06, 0x84};

#pragma    romdata    mydata = 0x320//data v ROM na adresu 0x320
far     rom  const   char   mydata[] = "HELLO“;

void   main ()
    {
        unsigned    char  z = 0;
        TRISD = 0;                        // porty B a D jako výstupní
        TRISB = 0;
        en = 0;                           // povol nečinnost
        MSDelay (250);
        lcdcmd (0x38);
        MSDelay (250);
            // odešli příkaz

            for ( ; z < 4; z++)
                {
                    lcdcmd (mycom [z]);
                    MSDelay (15);
                }
            // odešli data
            for (z = 0; z < 5; z++)
                {
                    lcddata (mydata [z]);
                    MSDelay (15);
                }
            while (1);                    // nekonečná smyčka
    }
void   lcdcmd (unsigned  char value)
    {
        ldata = value;                    // vlož hodnotu value na piny
        rs = 0;
        rw = 0;
        en = 1;                           // strobuj pin enable
        MSDelay (1);
        en = 0;
    }
```

KAPITOLA 13

ADC, DAC A ROZHRANÍ SENZORU

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Diskutovat o ADC (analogově-digitální převodník), který je součástí čipu PIC18
- Rozhraní teplotních senzorů pro PIC18
- Vysvětlit proces sběru dat s využitím ADC
- Popsat faktory, zvažované při výběru čipu ADC
- Programovat ADC PIC18 v C a assembleru
- Popsat základní operace s čipem DAC (digitálně-analogový převodník)
- Rozhraní čipu DAC a PIC18
- Programovat čip DAC pro tvorbu sinusového průběhu na osciloskopu
- Programovat čipy DAC v C PIC18 a assembleru
- Objasnit funkci přesných integrovaných senzorů teploty
- Popsat úpravu signálu a jeho roli v získávání dat

Tento bit spolu s bity ADCS1 a ADCS0 registru ADCON0 rozhoduje o hodinách pro převod v ADC. Výchozí hodnota pro ADCS2 je 0, což znamená, že nastavení hodnot ADCS0 a ADCS1 v ADCON0 nám může dát hodiny pro převod Fosc/2, Fosc/8 a Fosc/32. Podívejte se na registr ADCON0.

PCFG: Řídící bity konfigurace A/D portu:

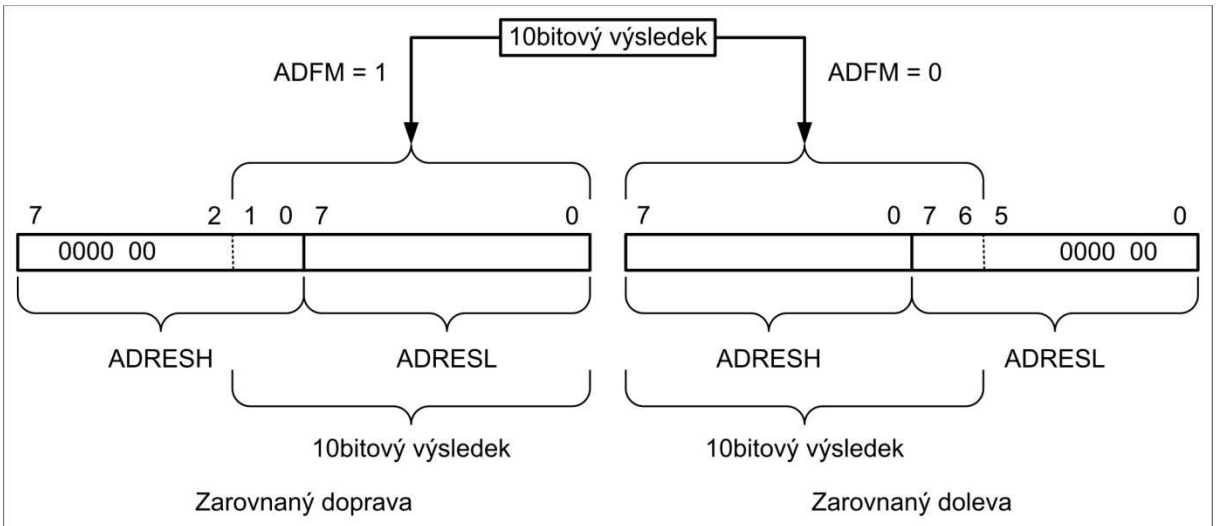
PCFG	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	Vref+	Vref-	C/R
0000	A	A	A	A	A	A	A	A	Vdd	Vss	8/0
0001	A	A	A	A	Vref+	A	A	A	AN3	Vss	7/1
0010	D	D	D	A	A	A	A	A	Vdd	Vss	5/0
0011	D	D	D	A	Vref+	A	A	A	AN3	Vss	4/1
0100	D	D	D	D	A	D	A	A	Vdd	Vss	3/0
0101	D	D	D	D	Vref+	D	A	A	AN3	Vss	2/1
011x	D	D	D	D	D	D	D	D	---	---	0/0
1000	A	A	A	A	Vref+	Vref-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	Vdd	Vss	6/0
1010	D	D	A	A	Vref+	A	A	A	AN3	Vss	5/1
1011	D	D	A	A	Vref+	Vref-	A	A	AN3	AN2	4/2
1100	D	D	D	A	Vref+	Vref-	A	A	AN3	AN2	3/2
1101	D	D	D	D	Vref+	Vref-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A		Vss	1/0
1111	D	D	D	D	Vref+	Vref-	D	A	AN3	AN2	1/2

A = analogový vstup D = digitální V/V

C/R = # analogových kanálů/# pinů, použitých pro referenční napětí A/D

Standardní volba je 0000, která nám dává 8 kanálů analogových vstupů a používá Vdd z PIC18 jako Vref.

Obrázek 13-7. ADCON1 (řídící registr 1 A/D)



Obrázek 13-8. Bit ADFM a registry ADRESx

tabulkové hodnoty celá čísla, představující velikost napětí pro sinus théta. Tato metoda zajišťuje, že na výstupu DAC mikrokontroléru PIC18 jsou pouze celá čísla. Tabulka 13-5 ukazuje úhly, hodnoty sinus, velikosti napětí a celočíselné hodnoty, které reprezentují velikost napětí pro každý úhel (přírůstky po krocích 30°). K vygenerování Tabulky 13-5, předpokládejme plný rozsah napětí 10 V pro výstup DAC (podle návrhu v Obrázku 13-11). Plného výstupu DAC je dosaženo, když jsou všechny datové vstupy DAC v horní úrovni. Proto, aby bylo dosaženo plného rozsahu 10 V výstupu, můžeme použít následující rovnici.

$$Vout = 5\,V + (5\,x \sin\theta)$$

Vout DAC pro různé úhly se vypočítá a je uvedeno v Tabulce 13-5. Podívejte se na Příklad 13-9, abyste si ověřili výpočty.

Příklad 13-8

Za účelem vytvoření schodovité rampy, sestavte obvod na Obrázku 13-11 a připojte výstup na osciloskop. Pak napište program pro odesílání dat do DAC k vytvoření průběhu schodovité rampy.

Řešení:

```
        CLRF    TRISB       ; PORTB jako výstup
        CLRF    PORTB       ; smaž PORTB
AGAIN:  INCF    PORTB, F    ; čítej od 0 do FFH a pošli to do DAC
        RCALL   DELAY       ; zotavení DAC
        BRA     AGAIN
```

Úhel θ (stupně)	Sin θ	Vout (velikost napětí) 5 V + (5 V x sin θ)	Hodnota, poslaná do DAC (desítkově) (velikost napětí x 25,6)
0	0	5	128
30	0,5	7,5	192
60	0,866	9,33	238
90	1,0	10	255
120	0,866	9,33	238
150	0,5	7,5	192
180	0	5	128
210	-0,5	2,5	64
240	-0,866	0,669	17
270	-1,0	0	0
300	-0,866	0,669	17
330	-0,5	2,5	64
360	0	5	128

Tabulka 13-5: Úhel, oproti velikosti napětí pro sinusový průběh

KAPITOLA 14

POUŽITÍ PAMĚTÍ FLASH A EEPROM PRO UKLÁDÁNÍ DAT

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Rozeznat a porovnat různé typy polovodičových pamětí z hlediska kapacity, organizace a doby přístupu
- Popsat vztah mezi počtem paměťových lokací na čipu, počtem datových pinů a kapacitou čipu
- Definovat paměť ROM Flash a popsat její použití v systémech, založených na PIC18
- Rozeznat a porovnat PROM, EPROM, UV-EPROM, EEPROM, paměť Flash EPROM, paměti ROM s maskou
- Psát programy pro PIC18 v assembleru a jazyku C pro zápis dat do prostoru paměti Flash v PIC18
- Psát programy pro PIC18 v assembleru a jazyku C pro smazání paměti Flash v PIC18
- Vysvětlit, jak zapisovat data do paměti EEPROM PIC18
- Vysvětlit, jak číst data z paměti EEPROM PIC18

```
        MOVWF  COUNT
        MOVLW  upper (NEW_DATA)   ; nahraj TBLPTR
        MOVWF  TBLPTRU
        MOVLW  high (NEW_DATA)
        MOVWF  TBLPTRH
        MOVLW  low (NEW_DATA)
        MOVWF  TBLPTRL
        CLRF   TRISB              ; PORTB jako výstupní port
        MOVLW  0xB                ; čítač = 8
        MOVWF  COUNT
LN      TBLRD*+                   ; čti znak
        MOVF   TABLAT,W
R1      BTFSS  PIR1, TXIF         ; čekej na dokončení posledního
        BRA    R1                 ; zůstaň ve smyčce
        MOVWF  TXREG              ; nahraj hodnotu pro vysílání
        DECFSZ COUNT              ; smyčka dokud je buffer plný
        BRA    LN                 ; opakuj
```

Kroky, potřebné pro vymazání paměti Flash

I když můžeme použít externí programátor Flash k vymazání obsahu paměti Flash, PIC18 nám umožňuje napsat program pro mazání paměti Flash. Mazací proces pracuje s daty velikosti bloků, nikoli velikosti bajtů. Minimální velikost bloku pro vymazání je 64 bajtů. To znamená, že nejnižších 6 bitů z adresy jsou samé nuly, což je 64bajtová hranice bloku. Můžeme použít následující kroky k vymazání jednoho 64bajtového bloku paměti Flash:

1. Vložte do registru TBLPTR adresu bloku, který bude vymazán.
2. Nastavte registr EECON1 pro operaci mazání nastavením
   (a) EEPGD = 1, (b) CFGS = 0, (c) WREN = 1 a (d) FREE = 1.
3. Zakázat globálně všechna přerušení pomocí "BCF INTCON, GIE".
4. Zapište 55H do fiktivního registru EECON2.
5. Zapište AAH do fiktivního registru EECON2.
6. Nastavte WR do 1 instrukcí "BSF EECON1, WR". S WR = 1 začíná mazací cyklus.
7. Bude trvat asi 2 ms, než se dokončí vymazání bloku 64 bajtů. Během tohoto cyklu mazání je procesor pozastaven a neumožňuje načítání operačního kódu. Po dokončení cyklu vymazání se bit WR vrátí automaticky z horní úrovně a signalizuje dokončení cyklu mazání.
8. Znovu povolte globální přerušení pomocí "BCF INTCON, GIE".

Program 14-4 ukazuje, jak smazat 64bajtový blok.

```
; Program 14-4: Tento program maže paměť Flash, začíná v lokaci 0x500.
        ORG    0
        MOVLW  upper (MYDATA)
        MOVWF  TBLPTRH            ; nahraj vyšší adresu
        MOVLW  high (MYDATA)
        MOVWF  TBLPTRH            ; nahraj horní bajt adresy
        MOVLW  low (MYDATA)
```

KAPITOLA 15

PROGRAMOVÁNÍ CCP A ECCP

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Pochopit funkci porovnávacího a zachytávacího modulu PIC18
- Prozkoumat využití časovačů v modulech CCP a ECCP
- Vysvětlit, jak pracují funkce porovnávacích modulů CCP a ECCP
- Vysvětlit, jak pracují funkce zachytávacích modulů CCP a ECCP
- Kódovat programy pro funkce porovnávání a zachytávání v assembleru a C
- Vysvětlit, jak pracuje PWM (pulzní šířková modulace) v CCP a ECCP
- Kódovat programy pro vytváření PWM v assembleru a C

```
        MOVWF  CCPR1L          ; CCPR1L = 0x50
OVER    CLRF   TMR1H           ; smaž TMR1H
        CLRF   TMR1L           ; smaž TMR1L
        BCF    PIR1, CCP1IF    ; smaž CCP1IF
        BSF    T1CON, TMR1ON   ; start Timer1
B1      BTFSS  PIR1, CCP1IF
        BRA    B1
        ; CCP přepíná pin CCP1 po shodě
        BCF    T1CON, TMR1ON   ; stop Timer1
        GOTO   OVER            ; udržuj to
```

Program 15-2C

```
// Program 15-2C je verzí C Programu 15-2.
    CCP1CON = 0x02;                 // porovnávací mód, přepíná po shodě
    T3CON = 0x0;                    // Timer1 pro porovnání, předdělič 1:1
    T1CON = 0x0;                    // Timer1 interní hodiny, předdělič 1:1
    TRISCbits.TRISC2 =0;            // nastav pin CCP1 jako výstup
    TRISCbits.TRISC0 = 1;           // nastav pin T1CLK jako výstup
    CCPR1H = 0xC3;                  // nahraj CCPR1H
    CCPR1L = 0x50;                  // nahraj CCPR1L
    while (1)
      {
        TMR1H = 0;                      // smaž Timer1
        TMR1L = 0;
        PIR1bits.CCP1IF = 0;            // smaž příznak CCP1IF
        T1CONbits.TMR10N = 1;           // zapni Timer1
        while (PIR1bits.CCP1IF == 0);   // čekej na CCP1IF
        // CCP přepíná pin CCP1 po shodě
        T1CONbits.TMR10N = 0;           //stop Timer1
      }
```

Kontrolní otázky

1. Pravda nebo nepravda. Pro porovnávací mód můžeme použít jakýkoliv časovače.
2. Pravda nebo nepravda. K dispozici je jeden pin, přiřazený k porovnávacímu módu.
3. Pravda nebo nepravda. V porovnávacím módu musí být pin CCP nakonfigurován jako vstupní pin.
4. Který registr se používá k výběru časovače pro porovnávací režim?

což znamená, že minimální možná PWM = Fosc/16,382.

Prozkoumejte Příklady 15-3 až 15-5, abyste viděli výpočet periody PWM.

Příklad 15-3

Najděte hodnotu PR2 a předděliče, potřebné k získání následujících frekvencí PWM. Předpokládejme, že XTAL = 20 MHz.

1. 1,22 kHz
2. 4,88 kHz
3. 78,125 kHz

Řešení:

1. Hodnota PR2 = [(20 MHz/(4 * 1,22 kHz)] - 1 = 4.097, což je větší než 255, maximální hodnota povolená pro PR2. Vybereme předdělič 16 a dostaneme Hodnota PR2 = [(20 MHz/(4 * 1,22 kHz * 16)] - 1 = 255
2. Hodnota PR2 = [(20 MHz / (4 x 4,88 kHz)] - 1 = 1.023, což je větší než 255, maximální hodnota povolená pro PR2. Vybereme předdělič 4 a dostaneme Hodnota PR2 = [(20 MHz/(4 * 4,88 kHz * 4)] - 1 = 255
3. Hodnota PR2 = [(20 MHz/(4 * 78,125 kHz)] - 1 = 63

Příklad 15-4

Najděte hodnotu PR2 pro následující frekvence PWM. Předpokládejme, že XTAL = 10 MHz a předdělič = 1.

1. 10kHz
2. 25 kHz

Řešení:

1. Hodnota PR2 = [(10 MHz/(4 * 10 kHz * 1)] - 1 = 250 - 1 = 249
2. Hodnota PR2 = [(10 MHz/4 * 25 kHz * 1) - 1 = 100 - 1 = 99

Příklad 15-5

Najděte minimální a maximální frekvence Fpwm, možné pro XTAL = 10 MHz. Uveďte hodnoty PR2 a předděliče pro minimální a maximální Fpwm.

Řešení:

Položíme PR2 = 255 a vybereme předdělič 16, abychom dostali minimální Fpwm, což nám dává 10 MHz/(4 * 16 * 256) = 610 Hz.

Položíme PR2 = 1 a vybereme předdělič 1, abychom dostali maximální Fpwm, což nám dává 10 MHz/(4 * 1 * 1) = 2,5 MHz.

KAPITOLA 16

PROTOKOL SPI A ROZHRANÍ DS1306 RTC

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Pochopit protokol SPI (sériové periferní rozhraní)
- Vysvětlit, jak pracují operace čtení a zápisu v SPI
- Prozkoumat piny SDO, SDI, CE a SCLK modulu SPI
- Kódovat programy pro SPI v assembleru a C
- Vysvětlit, jak pracuje obvod RTC (hodiny reálného času)
- Vysvětlit funkce pinů DS1306 RTC
- Vysvětlit funkce registrů DS1306 RTC
- Pochopit připojení DS1306 RTC k PIC18
- Kódovat programy pro zobrazení času a data v assembleru a C
- Poznat a programovat alarm a funkce přerušení RTC

Adresové lokace (hexadecimálně)		Funkce	Rozsah dat	
Čtení	Zápis		**BCD**	**Hexadecimálně**
00	80	Sekundy	00 – 59	00 – 59
01	81	Minuty	00 - 59	00 – 59
02	82	Hodiny, 12hodinový mód	01 – 12	41 – 52 AM
		Hodiny, 12hodinový mód	01 – 12	61 – 72 PM
		Hodiny, 24hodinový mód	01 - 23	00 – 23
03	83	Den v týdnu, Ne = 1	01 - 07	01 – 07
04	84	Den v měsíci	01 - 31	01 – 31
05	85	Měsíc	01 – 12	01 – 12
06	86	Rok	00 - 99	00 – 99

Tabulka 16-2: Adresové lokace DS1306 pro datum a čas (vybráno z Tabulky 16-1)

Příklad 16-1

S použitím Tabulky 16-1 ověřte hodnoty lokace hodin v Tabulce 16-2.

Řešení:

(a) pro 24hodinový režim máme D6 = 0. Proto je rozmezí od 00**00 0000** do 00**100011**, což je 00 - 23 v BCD.
(b) pro 12hodinový režim máme D6 = 1 a D5 = 0 pro AM. Proto je rozsah od 010**0 0001** do 010**1 0010**, což je 41 až 52 v BCD.
(c) pro 12hodinový režim máme D6 = 1 a D5 = 1 pro PM. Proto je rozsah od 011**0 0001** do 011**1 0010**, což je 61 až 72 v BCD.

Připojení DS1306 k PIC18 s použitím modulu MSSP

DS1306 podporuje SPI i 3vodičový mód. V DS1306 zvolíme režim SPI připojením pinu SERMODE k Vcc. Pokud je SERMODE = GND, pak použijeme 3vodičový protokol. V této sekci budeme používat pouze režim SPI. Modul MSSP (Master Synchronous Serial Port) v PIC18 podporuje protokol sběrnice SPI. K SPI modulu MSSP jsou přiřazeny tři registry. Jsou to SSPBUF, SSPCON1 a SSPSTAT. K přenesení bajtu dat ho umístíme do registru SSPBUF. Registr SSPBUF také udržuje bajt, přijatý přes sběrnici SPI. Obrázky 16-10 a 16-11 ukazují další dva hlavní registry PIC18 pro rozhraní SPI. SSPCON1 používáme pro výběr režimu provozu SPI v PIC18. Všimněte si, že bit SSPEN v registru SSPCON1 musí být nastaven do horní úrovně, aby umožnil použití pinů PIC18 pro protokol sběrnice SPI. Musíme si také vybrat mód SPI Master pomocí bitů SSPM3:SSPM0 z SSPCON1. V naší aplikaci budeme používat rychlost Fosc/64 pro nejlepší výkon v datovém přenosu mezi PIC18 a DS1306 RTC.

Po výběrech v SSPCON1 také musíme vybrat bity pro vhodné časování v registru SSPSTAT, jak je znázorněno na Obrázku 16-11. V naší aplikaci posíláme data do zařízení SPI při náběžné hraně a přijímáme data ze zařízení SPI uprostřed pulzu hodin SCLK.

KAPITOLA 17

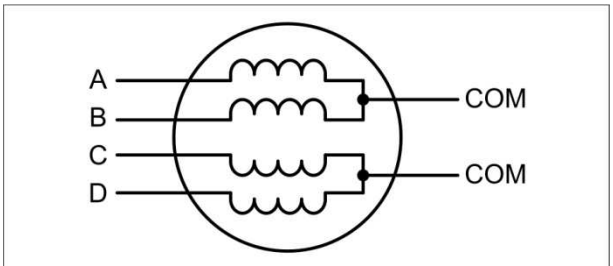
ŘÍZENÍ MOTORŮ: RELÉ, PWM, DC A KROKOVÉ MOTORY

Cíle:

Po dokončení této kapitoly byste měli být schopni:

- Popsat základní funkce relé
- Rozhraní PIC18 s relé
- Popsat základní funkce s optoizolátorem
- Rozhraní PIC18 s optoizolátorem
- Popsat základní funkce krokového motoru
- Rozhraní PIC18 s krokovým motorem
- Kódovat programy PIC18 pro řízení a provoz krokového motoru
- Definovat provoz krokového motoru z hlediska úhlu kroku, kroků na otáčku, počtu pólů, rychlosti otáčení a otáček za minutu
- Popsat základní provoz DC motoru
- Rozhraní PIC18 s DC motorem
- Kódovat programy PIC18 pro řízení a provoz DC motoru
- Popsat jak je použita PWM pro řízení rychlosti motoru
- Kódovat programy CCP pro řízení a provoz DC motoru
- Kódovat programy ECCP pro řízení a provoz DC motoru

magnetické póly stejné polarity odpuzují a s opačnou polaritou přitahují. Řízení směru otáčení je dáno polem statoru. Statorové pole je určeno aktuálním proudem, který probíhá cívkami. Tak, jak probíhá změna směru proudu, mění se také polarita, která způsobí opačný pohyb rotoru. Zde diskutovaný krokový motor má celkem šest vodičů: čtyři vodiče zastupují čtyři vinutí statoru a dva společné jsou vývody společných zemí. Rotor se začne otáčet po přivedení sledu budicích signálů na každé statorové vinutí. Existuje několik široce používaných sekvencí, z nichž každá má jiný stupeň přesnosti. Tabulka 17-3 ukazuje dvoufázovou, čtyřkrokovou sekvenci.



Obrázek 17-8. Konfigurace statorového vinutí

Všimněte si, že ačkoli můžeme začít libovolnou sekvencí Tabulky 17-3, jakmile začneme, musíme pokračovat ve správném pořadí. Například, pokud začneme s krokem 3 (0110), musíme pokračovat ve sledu kroků 4, 1, 2, atd.

Po směru hodinových ručiček	Krok	Vinutí A	Vinutí B	Vinutí C	Vinutí D	Proti směru hodinových ručiček
	1	1	0	0	1	
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	

Tabulka 17-3: Normální čtyřkroková sekvence

Úhel kroku

Jak velký pohyb je spojen s jedním krokem? To závisí na vnitřní konstrukci motoru, zejména v počtu zubů na statoru a rotoru. *Úhel kroku* je minimální stupeň natočení, spojený s jedním krokem. Různé motory mají různé úhly kroku. Tabulka 17-4 ukazuje některé úhly kroku pro různé motory. V Tabulce 17-4 si všimněte výrazu *kroky na otáčku*. To je celkový počet kroků, potřebný k úplnému otočení o jednu celou otáčku nebo 360 stupňů (např. 180 kroků * 2° = 360°).

Úhel kroku	Kroků na otáčku
0,72	500
1,8	200
2,0	180
2,5	144
5,0	72
7,5	48
15	24

Tabulka 17-4: Úhel kroku krokového motoru

Musíme poznamenat, možná na rozdíl od prvního dojmu, že krokový motor nepotřebuje další svorky ke statoru pro dosažení menších kroků. Všechny krokové motory, popsané v této sekci mají čtyři vodiče pro vinutí statoru a dva vodiče pro COM ke středu vinutí. I když někteří výrobci vyčlenili jen jeden vodič pro společný signál COM místo dvou, vždy máme

Provoz motoru	SW1	SW2	SW3	SW4
Vypnuto	vypnuto	vypnuto	vypnuto	vypnuto
Točí ve směru hodinových ručiček	sepnuto	vypnuto	vypnuto	sepnuto
Točí proti směru hodinových ručiček	vypnuto	sepnuto	sepnuto	vypnuto
Neplatné	sepnuto	sepnuto	sepnuto	sepnuto

Tabulka 17-10: Některé logické konfigurace H-můstku pro Obrázek 17-14

I když nezobrazujeme řídící relé pro H-můstek, Příklad 17-5 ukazuje jednoduchý program pro provoz základního H-můstku.

Příklad 17-5

Spínač je připojen na pin RD7 (PORT0. 7). Pomocí simulátoru napište program pro simulaci H-můstku z Tabulky 17-10. Musíme provést následující:

(a) Pokud je DIR = 0, DC motor točí ve směru hodinových ručiček.
(b) Pokud je DIR = 1, DC motor točí proti směru hodinových ručiček.

Řešení:

```
        BCF     TRISB, 0        ; PORTB.0 jako výstup pro spínač 1
        BCF     TRISB, 1        ; PORTB.1 jako výstup pro spínač 2
        BCF     TRISB, 2        ; PORTB.2 jako výstup pro spínač 3
        BCF     TRISB, 3        ; PORTB.3 jako výstup pro spínač 4
        BSF     TRISD, 7        ; nastav PORTD.7 jako vstup DIR
MONITOR:
        BTFSS   PORTD, 7
        BRA     CLOCKWISE
        BSF     PORTB, 0        ; spínač 1
        BCF     PORTB, 1        ; spínač 2
        BCF     PORTB, 2        ; spínač 3
        BSF     PORTB, 3        ; spínač 4
        BRA     MONITOR
CLOCKWISE:
        BCF     PORTB, 0        ; spínač 1
        BSF     PORTB, 1        ; spínač 2
        BSF     PORTB, 2        ; spínač 3
        BCF     PORTB, 3        ; spínač 4
        BRA     MONITOR
```

Zobrazte si výsledky na svém simulátoru. Tento příklad je pouze pro simulaci a neměl by být používán na připojeném systému.

Podívejte se na http://www.MicroDigitalEd.com na doplňující informace o použití H-můstků.

Obrázek 17-18 ukazuje připojení L293 k PIC18. Uvědomte si, že L293 bude generovat během provozu teplo. Pro trvalý provoz motoru použijte chladič. Příklad 17-6 ukazuje řízení L293.

Program 17-3

Program 17-3 ukazuje implementaci plného můstku pro PWM s modulem ECCP. Pro realizaci plného můstku a další aplikace PWM modulu ECCP naleznete v manuálu PIC18.

```
; Program 17-3
        CLRF    TRISD           ; nastav PORTD jako výstupní
        MOVLW   D'100'
        MOVWF   PR2             ; perioda = 100 * 16/Fosc
        MOVLW   D'50'
        MOVWF   ECCPR1L         ; střída = 50%
        MOVLW   0xCF
        MOVWF   ECCP1CON        ; reverzní plný můstek PWM
        MOVLW   0x24
        MOVWF   T2CON           ; postdělič 4, zapni Timer2
AGAIN   CLRF    TMR2            ; start pulzu
        BCF     PIR1,TMR2IF     ; smaž příznak
WAIT    BTFSS   PIR1,TMR2IF     ; čekej na periodu
        BRA     WAIT
        BRA     AGAIN           ; opakuj znovu
```

Následuje verze výše uvedeného programu v C.

Program 17-3C

```
// Program 17-3C
#include    <p18f458.h>

void main ()
  {
     TRISD = 0;                          // nastav PORTD jako výstupní
     PR2 = 100;                          // perioda = 100 * 16/Fosc
     ECCPR1L = 50;                       / / střída = 50%
     ECCP1CON = 0xCF;                    // reverzní plný můstek PWM
     T2CON = 0x24;                       // postdělič 4, zapni Timer2
  while (1)
     {
        TMR2 = 0;                        // start pulze
        PIR1bits.TMR2IF = 0;             // smaž příznak
        while (PIR1bits.TMR2IF == 0);    // čekej na periodu
     }
  }
```

Kontrolní otázky

1. Pravda nebo nepravda. S ECCP1 používáme pro plný můstek piny RD3 – RD0.
2. Pravda nebo nepravda. Pro ECCP1 musíme piny P1A až P1D nakonfigurovat jako výstupní.
3. V ECCP1 používáme …….. k nastavení periody pro PWM.
4. V ECCP1 používáme …….. k nastavení střídy pro PWM.

PŘÍLOHA A

INSTRUKCE PIC18:

FORMÁT A POPIS

Obecný přehled:

V první sekci tohoto dodatku popíšeme formát instrukcí PIC18. Zvláštní důraz položíme na instrukce, které používají WREG a soubor registrů. Tato sekce obsahuje výpis strojových cyklů pro každou instrukci PIC18.

V druhé části tohoto dodatku popíšeme jednotlivé instrukce PIC18. Ve většině z nich uvedeme jednoduchý programový příklad, vysvětlující instrukci.

SEKCE A. 2: INSTRUKČNÍ SADA PIC18

V této sekci nabízíme stručný popis jednotlivých instrukcí s příklady.

ADDLW k Add Literal to WREG

Funkce: Přičti hodnotu literálu k registru WREG

Syntaxe: ADDLW k

Instrukce přičte hodnotu literálu k do registru WREG a výsledek uloží zpět do WREG. Vzhledem k tomu, že registr WREG má šířku jednoho bajtu, musí být operand k také ve velikosti jednoho bajtu.

Instrukce ADD se používá jak pro čísla se znaménkem, tak pro čísla bez znaménka. Každou z nich popíšeme samostatně. Prohlédněte si Kapitolu 5, kde jsme probírali čísla se znaménkem.

Sčítání čísel bez znamének

Když sčítáme čísla bez znaménka, může se stav C, DC, Z, N a OV změnit. Nejdůležitější z těchto příznaků, je C. Nastaví se do 1, když nastane přenos z D7 ven v 8bitové operaci (D0-D7).

Příklad:

```
        MOVLW  0x45          ; WREG = 45H
        ADDLW  0x4F          ; WREG = 94H (45H + 4FH = 94H)
                             ; C = 0
```

Příklad:

```
        MOVLW  0xFE          ; WREG = FEH
        ADDLW  0x75          ; WREG = FE + 75 = 73H
                             ; C = 1
```

Příklad:

```
        MOVLW  0x25          ; WREG = 25H
        ADDLW  0x42          ; WREG = 67H (25H + 42H = 67H)
                             ; C = 0
```

Všimněte si, že ve všech uvedených příkladech jsme ignorovali stav příznaku OV. I když instrukce ADD ovlivňují OV, význam příznaku OV souvisí se znaménkem. Toto probereme dále.

Sčítání čísel se znaménkem a záporná čísla

Při sčítání čísel se znaménkem musí být věnována zvláštní pozornost příznaku přetečení (OV), protože to znamená, že ve výsledku sčítání je chyba. Existují dvě pravidla pro nastavení OV v operaci čísel se znaménkem. Příznak přetečení je nastaven na 1:

1. Pokud je přenos z D6 do D7 a není přenos z D7 ven.

PŘÍLOHA C

TECHNOLOGIE INTEGROVANÝCH OBVODŮ

A PROBLÉMY SYSTÉMOVÉHO NÁVRHU

Obecný přehled:

Tento dodatek obsahuje přehled technologie integrovaných obvodů a rozhraní PIC18. Dále se podíváme na systémy, založené na mikrontrolérech jako celku a prozkoumáme některé obecné otázky při návrhu systému.

Nejdříve v Sekci C. 1 nabídneme přehled technologií integrovaných obvodů.

Potom v Sekci C. 2 probíráme interní detaily a připojení V/V portů PIC18.

Sekce C. 3 se zabývá problémy systémového návrhu.

V PIC18 se úsporný režim označuje jako *režim spánku* (sleep mode). Dále si režim spánku popíšeme.

Mód spánku

V režimu spánku je vestavěný oscilátor zmražen, což vypne frekvenci procesoru a periferní funkce, jako jsou sériové porty, přerušení a časovače. Všimněte si, že i když tento způsob přináší spotřebu energie až na absolutní minimum, je obsah RAM a SFR registrů uložen a zůstává beze změny.

Odrazy od země

Jedním z hlavních problémů, se kterým se návrháři vysokofrekvenčních systémů musí vypořádat, jsou odrazy od země. Než budeme definovat odrazy od země, prodiskutujeme indukčnost vedení vývodů obvodu. S každým vývodem integrovaného obvodu je spojená určitá velikost kapacity, odporu a indukčnosti. Velikost těchto prvků se liší v závislosti na mnoha faktorech, jako je délka, plocha a tak dále.

Indukčnost pinů je běžně označována jako *vlastní indukčnost*, kterou odlišujeme od termínu, nazývaného *vzájemná indukčnost*, jak si ukážeme níže. Ze tří vlastností, které jsou kapacita, odpor a indukčnost, je vlastní indukčnost tou, která způsobuje většinu problémů v návrhu vysokofrekvenčních systémů, protože výsledkem mohou být odrazy od země. Odraz od země nastane tehdy, když obrovské množství proudu, protékající do zemnícího pinu způsobí na více výstupech změny z horní do dolní úrovně ve stejnou dobu. Viz Obrázek C-15 (a). Napětí souvisí s indukčností zemního vedení následujícím způsobem:

$$V = L\frac{di}{dt}$$

Při zvyšování frekvence systému, míra dynamického proudu (di/dt), rovněž roste, což vede k nárůstu indukovaného napětí na indukčnosti L (di/dt) zemnícího vývodu. Vzhledem k tomu, že stav NULA (zem) má malou šumovou odolnost, naindukované napětí může způsobit falešný signál. Pro snížení vlivu odrazů od země, musí být pokud možno splněny následující kroky:

1. Piny Vcc a zem na čipu musí být umístěny uprostřed pouzdra než na opačných koncích čipu integrovaného obvodu (14pinový logický obvod TTL využívá piny 7 a 14 pro zem a Vcc). To je přesně to, co vidíme ve vysoce výkonných logických hradlech, jako je zdokonalená logika rodin Texas Instruments AC11000 a ACT11000. Například ACT11013 je 14pinový obvod DIP, ve kterém jsou piny číslo 4 a 11 používány pro zem a Vcc, místo 7 a 14 tradiční TTL rodiny. Můžeme také použít pouzdra SOIC místo DIP.
2. Jiným řešením je použít tolik pinů pro zem a Vcc, kolik je možné a snížit tak délku vedení. To je přesně důvod, proč všechny výkonné mikroprocesory a logické rodiny používají mnoho pinů pro Vcc a zem, namísto jednoho tradičního pinu pro Vcc a

PŘÍLOHA D

VÝVOJOVÝ DIAGRAM A PSEUDOKÓD

Obecný přehled:

Tato příloha poskytuje úvod do zápisu vývojových diagramů a pseudokódu.