Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

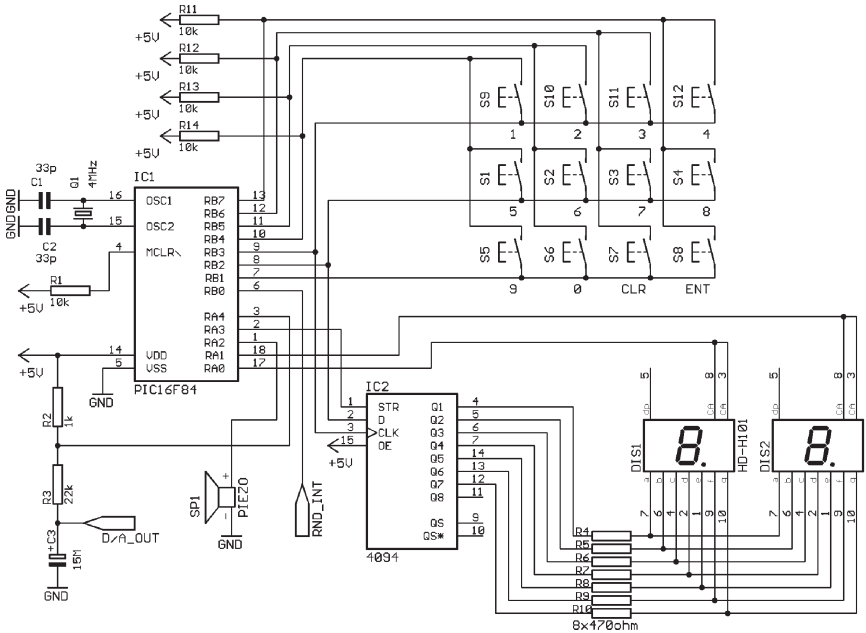
Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

4.2. Definice hardware jednotky

V této kapitole si ukážeme návrh hardware naší řídicí jednotky a dále si ukážeme jak naspecifikovat zapojení jednotlivých pinů procesoru do programu.

Jedno z možných zapojení jednotky je uvedeno na *obr. 15*.



obr. 15

Při definici hardware v programu vyjdeme z tohoto zapojení. Nejdříve si však stručně vysvětlíme funkci jednotlivých prvků.

Řídicí prvek jednotky procesor **PIC16F84** je konfigurován v tomto zapojení tak, že jeho hodinový oscilátor využívá krystal **Q1** ve spolupráci s kapacitami **C1** a **C2**. Odpor **R1**, který je připojen do vstupu MCLR, nám zajišťuje klidový stav na resetovacím vstupu procesoru. Procesor je napájen ze zdroje +5 V přes vývody č. 14 a 5. D/A převodník je integrátor pulzně šířkové modulace, která vystupuje z pinu **RA4** procesoru, a tento integrátor je tvořen prvky **R3** a **C3**. Odpor **R2** slouží pouze k předdefinování logické úrovně 1 pro pin **RA4**, protože tento pin je jako jediný definován od výrobce jako výstup s otevřeným kolektorem. Piezosirénka je připojena na výstup **RA2** a můžeme použít prakticky libovolný typ. Spínačové pole **S1** až **S12** nám tvoří klávesnici v zapojení s maticovou strukturou. U každého spínače je též uvedena jeho hodnota, kterou stisk tohoto spínače bude prezentovat. Kladný přerušovací impulz pro vygenerování náhodného čísla je zaveden na vstup **RB0**. Poslední část jednotky tvoří posuvný registr **74HC4094** a dva segmenty displeje s nízkým příkonem **DIS1** a **DIS2**. Tyto displeje jsou zapojeny v multiplexním režimu. To znamená, že v jednom okamžiku svítí segment **DIS1** a výstupy posuvného registru **IC2** mají jemu náležející logickou kombinaci a v druhém okamžiku svítí segment **DIS2**. Oba jsou přepínány takovou rychlostí, že to lidské oko nemůže postřehnout, ale ve skutečnosti svítí jenom jeden z nich. Jednotlivé piny portů procesoru mohou být konfigurovány libovolně jako vstupy nebo výstupy a záleží to pouze na definicích v programu. Po hardwarovém resetu (vypnutí napájení nebo logická 0 na vstup MCLR) jsou všechny piny portů nastaveny jako vstupy.

Nyní přistoupíme k popisu definic zapojení hardware jednotky. Při definování názvů jednotlivých pinů portů používáme direktivu EQU, která textovému řetězci (návěští) přiřadí numerickou hodnotu, tedy číslo pořadí pinu portu. Jako příklad si uvedeme pin výstupu pro piezosirénku.

```
OUTPIP  EQU     2           ;  PORTA
```

Z příkladu je vidět, že textovému řetězci OUTPIP jsme přiřadili numerickou hodnotu 2 a v praxi to znamená, že napíšeme-li do parametru instrukce textový řetězec OUTPIP, pro překladač je to totéž, jako bychom tam napsali hodnotu 2. Teď si určitě kladete otázku, proč vlastně jeden znak nahrazujeme šesti, že je to vlastně zbytečná práce. Není tomu tak, protože při pohledu na

parametr z textového řetězce poznáme snadněji, jakou funkci tento pin v hardware má, ale hlavně výraz OUTPIP se může vyskytovat na mnoha místech v programu a budeme-li chtít jednoduše změnit zapojení tak, že piezosirénku z pinu č. 1 budeme chtít přesunout na pin č. 17, aby se nám třeba nekřížily spoje, tak to provedeme pouze změnou čísla přiřazení EQU a to z 2 na 0. Tato hodnota se nám pak změní v celém programu a budeme mít jistotu, že jsme na to nikde nezapomněli.

V praxi ještě bývá užitečné, si uvést do komentáře za přiřazovací direktivu EQU, kterého portu se toto přiřazení týká. V našem případě je to PORTA. V následujícím výpisu **PRIKLAD 2** již uvedu veškeré definice hardware jednotky, u kterých se postupovalo obdobně jako v předchozím příkladu. Přiřazované číslo znamená pořadí pinu v jednotlivých portech.

```
; ukazkovy program ucebnice programovani PIC
; verze 1.1
; lze pouze prelozit, obsahuje direktivy prekladace,
; ale nic nedela
; obsahuje definice hardware jednotky
LIST P = 16F84, R = DEC    ;  direktivy prekladace
include<p16f84.inc>        ;  definice nazvu registru
                           ;  definice zapojeni hardware
KL_R1      EQU     3       ;  portb
KL_R2      EQU     2       ;  portb
KL_R3      EQU     1       ;  portb
KL_S1      EQU     4       ;  portb
KL_S2      EQU     5       ;  portb
KL_S3      EQU     6       ;  portb
KL_S4      EQU     7       ;  portb
IN_RND     EQU     0       ;  portb
AD_OUT     EQU     4       ;  porta
OUT_STR    EQU     3       ;  porta
OUT_D      EQU     2       ;  portb
OUT_CLK    EQU     3       ;  portb
OUTPIP     EQU     2       ;  porta
AN1        EQU     1       ;  porta
AN2        EQU     0       ;  porta
; ------------------------------------------------
; promenne
; ------------------------------------------------
```

```
;  ---------------------------------------------------
;  kod programu
;  ---------------------------------------------------
        ORG     0
        GOTO    Start       ;  skok na zacatek programu
;  ---------------------------------------------------
;  hlavni program
;  ---------------------------------------------------
Start                       ;  inicializace procesoru
Main
        GOTO    Main        ;
;  ---------------------------------------------------
;  obsluzne procedury
;  ---------------------------------------------------
        END                 ;  konec programu
```

4.3. Inicializace programu a obsluhy přerušení

V této kapitole určíme počáteční podmínky hardware pro správnou funkci programu a také si ukážeme základní program pro obsluhu přerušení.

Určení počátečních podmínek pro správnou funkci programu se děje v části, která se nazývá **inicializace**. Zde se nastavují potřebné řídicí registry procesoru tak, aby správně vykonával, co je třeba dle zadání. Touto **inicializací** musí procesor projít vždy při startu programu, tj. při hardwarovém resetu na pinu MCLR nebo při resetu od časovače WATCHDOG.

Základní program pro obsluhu přerušení si pro lepší ilustraci vysvětlíme na ukázce PRIKLAD 3 uvedené v následujícím textu:

```
; ukazkovy program ucebnice programovani PIC
; verze 1.2
; lze pouze prelozit, obsahuje direktivy prekladace,
; ale nic nedela
; obsahuje definice hardware jednotky
; inicializace procesoru a zakladni osetreni preruseni
LIST P = 16F84, R = DEC    ;  direktivy prekladace
include<p16f84.inc>        ;  definice nazvu registru
                           ;  definice zapojeni hardware
KL_R1   EQU     3          ;  portb
KL_R2   EQU     2          ;  portb
KL_R3   EQU     1          ;  portb
KL_S1   EQU     4          ;  portb
KL_S2   EQU     5          ;  portb
KL_S3   EQU     6          ;  portb
KL_S4   EQU     7          ;  portb
IN_RND  EQU     0          ;  portb
AD_OUT  EQU     4          ;  porta
OUT_STR EQU     3          ;  porta
OUT_D   EQU     2          ;  portb
OUT_CLK EQU     3          ;  portb
OUTPIP  EQU     2          ;  porta
AN1     EQU     1          ;  porta
AN2     EQU     0          ;  porta
; ------------------------------------------------
; promenne
; ------------------------------------------------
OffRam  EQU     0x0C       ;  offset pameti pro PIC16F84
Mem_W   EQU     OffRam     ;  pamet pracovniho registru
Mem_StatEQU     OffRam+1   ;  pamet registru STATUS
; ------------------------------------------------
; kod programu
; ------------------------------------------------
        ORG     0
        GOTO    Start      ;  skok na zacatek programu
        ORG     4          ;
        GOTO    IntProc    ;  skok na obsluhu preruseni
```

```
; ------------------------------------------------------
; hlavni program
; ------------------------------------------------------
Start                       ;  inicializace procesoru
   BSF     STATUS,RP0       ;  registrova sada 1
   MOVLW   0xC5             ;  nastaveni OPTION
   MOVWF   OPTION_REG       ;  ok
   CLRF    TRISA            ;  piny PORTA jako vystupy
   MOVLW   0xF1             ;  RB0 a RB4 az RB7 vstupy
   MOVWF   TRISB            ;  jinak vystupy
   BCF     STATUS,RP0       ;  registrova sada 0
   CLRF    PORTA            ;  cely PORTA do 0
   MOVLW   0xFF             ;  cely PORTB do 1
   MOVWF   PORTB            ;  ok
   MOVLW   0xB0             ;  povoleni preruseni
   MOVWF   INTCON           ;  ok
Main
   GOTO    Main             ;
; ------------------------------------------------------
; obsluzne procedury
; ------------------------------------------------------
IntProc                     ;  obsluha preruseni
   MOVWF   Mem_W            ;  ulozeni registru W
   MOVF    STATUS,W         ;  ulozeni obsahu STATUS
   MOVWF   Mem_Stat         ;  ok
   BCF     INTCON,T0IF      ;  clr priznaku preruseni TMR0
   MOVF    Mem_Stat,W       ;  obnova stavu STATUS
   MOVWF   STATUS           ;  ok
   SWAPF   Mem_W,F          ;  obnova registru W
   SWAPF   Mem_W,W          ;  ok
   RETFIE                   ;  navrat z preruseni
   END                      ;  konec programu
```

Oproti předchozímu příkladu nám na ukázce přibyly definice proměnných. Jako první je definován tzv. offset paměti **RAM**, který je pro PIC16F84 0x0C. Je to adresa první buňky v datové paměti, které není přiřazena funkce speciálního registru. Od této adresy již můžeme data libovolně zapisovat a číst, aniž bychom měnili nastavení funkce procesoru. Následují dvě proměnné, které využívá procedura obsluhy přerušení. Dříve jsme si vysvětlili, že přerušení je asynchronní událost v chodu

programu a je to skok z libovolného místa programu na adresu 0x04. Tato adresa je stejná pro všechny procesory **PIC**. Nazýváme jí vektor přerušení. Při návratu z obsluhy přerušení se vždy vracíme do místa, kdy přerušení vzniklo a vykonání přerušení nesmí mít destruktivní vliv na chod základního programu, tzn. že přerušení nesmí změnit hodnotu pracovního registru ***W*** a hodnotu registru ***STATUS***, podle kterého se rozhodují podmínky skoků. To zajistíme tak, že jejich okamžité stavy v místě volání přerušení nejdříve uložíme do paměťové buňky **Mem_W** a **Mem_Stat**.

Po resetu program nejdříve proběhne od adresy 0 (zde leží instrukce GOTO Start) základní inicializace hardware. Datový paměťový prostor je u tohoto typu procesoru rozdělen do dvou registrových bank a většina registrů, které řídí chod procesoru, leží v bance 1. Tuto banku přepneme instrukcí BSF STATUS, RP0, kde **RP0** je řídicí bit pro banky registrů. Určitě je vám v tuto chvíli podivné, kde se berou názvy ***STATUS***, **RP0** a další, aniž bychom je v předchozím textu programu definovali. Veškeré definice jsou uvedeny ve vkládaném souboru **p16f84.inc**, který je do našeho programu vložen direktivou INCLUDE.

Jako další v inicializaci nastavíme registr **OPTION**.

Význam jednotlivých bitů tohoto registru.

D7	D6	D5	D4	D3	D2	D1	D0	
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	OPTION_ REG [adr.0x81]

- **$\overline{\text{RBPU}}$** Příznak zapojení rezistorů mezi pinem a napájením.
  - *1* = Rezistory PORTB jsou odpojeny.
  - *0* = Rezistory PORTB jsou připojeny.
- **INTEDG** Řídicí bit hrany přerušení.
  - *1* = Přerušení generováno na náběžnou hranu pulzu na pinu RB0.
  - *0* = Přerušení generováno na spádovou hranu pulzu na pinu RB0.
- **T0CS** Příznak zdroje hodin časovače TMR0**.**

1 = Časovač čítá pulzy na pinu RA4.

0 = Časovač čítá pulzy z vnitřního zdroje hodin.

- **T0SE** Příznak hrany čítání časovače TMR0**.**

  1 = Časovač čítá na spádovou hranu pulzu na pinu RA4.

  0 = Časovač čítá na náběžnou hranu pulzu na pinu RA4.

- **PSA** Příznak přiřazení předděličky.

  1 = Předdělička je předřazena časovači WATCHDOG.

  0 = Předdělička je předřazena časovači TMR0.

- **PS2:PS0** Řídicí bity dělicího poměru předděličky.

  Kombinace řídicích bitů předděličky řídí dělicí poměr a to způsobem, který je uveden ve firemním manuálu **PIC16F84**.

Do registru **OPTION** je zapisováno číslo 0xC5 což znamená, že odpory do napájení **portu B** budou vypnuty, přerušení bude generováno na náběžnou hranu pinu RB0, zdroj hodin pro časovač TMR0 bude vnitřní oscilátor, čítání časovače TMR0 bude nezávislé na nastavení bitu příznaku hrany čítání TMR0, předdělička bude předřazena časovači TMR0 a kombinace posledních 3 bitů nám určí dělicí poměr předděličky. V našem případě to je 1:64.

Jak jsme k tomuto číslu dospěli? Externí hodinový oscilátor procesoru má připojen krystal 4 MHz. Tento kmitočet je pro vnitřní časování CP vydělen 4. My budeme chtít, aby časovač TMR0 nám generoval při svém přetečení přerušení každých 10 ms. Z toho vyplývá, že dělicí poměr celkového řetězce musí být 40 000. Kmitočet krystalu je pevně vydělen 4 a časovač TMR0 nám dělí 256, protože je 8bitový. Toto dá dohromady dělicí poměr 1024. Podělíme-li číslo 40 000 tímto dělicím poměrem, získáme požadovaný dělicí poměr předděličky 39. Toto však není možné nastavit, neboť předdělička nastavuje svůj dělicí poměr pouze v mocninách čísla 2, proto my zvolíme číslo tomuto nejbližší vyšší, a to je číslo 64. Poté nám vyjde, že časovač TMR0 musí dělit přibližně 156, a to je možné zařídit. Tento způsob jsme volili proto, abychom se při znovunaplňování časovače TMR0 (protože už nebude čítat v plném svém rozsahu) dopustili pokud možno co nejmenší chyby.

V inicializaci bude následovat nastavení charakteru pinů **portu A** a **portu B**. Slouží k tomu registry TRISA a TRISB, které leží také v registrové bance 1. Jednotlivé bity registru TRIS a portu si svou polohou odpovídají a je-li bit

registru TRIS nastaven na úroveň 0, odpovídající bit portu je nastaven jako výstup. Je-li bit registru TRIS nastaven do úrovně 1, odpovídající bit portu je vstup.

Po této konfiguraci následuje přepnutí na registrovou sadu 0, vynulujeme všechny výstupy **portu A** a všechny piny **portu B** nastavíme do úrovně 1. V tomto stavu se nám ve vnějším hardware nic nenastavuje ani nám nic nekoliduje, takže toto bude pro nás výchozí stav.

Jako poslední v této části inicializace nastavíme řídicí registr přerušení **INTCON**.

Struktura řídicího registru přerušení INTCON.

D7	D6	D5	D4	D3	D2	D1	D0	
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	INTCON [adr.0x0B,0x8B]

- **GIE** Globální povolení přerušení.
  1 = Přerušení jsou povolena.
  0 = Přerušení jsou zakázána.
- **EEIE** Příznak přerušení při dokončení zápisu do paměti EEPROM.
  1 = Přerušení po zápisu do EEPROM povoleno.
  0 = Přerušení po zápisu do EEPROM zakázáno.
- **T0IE** Povolení přerušení při přetečení časovače TMR0.
  1 = Přerušení od TMR0 povoleno.
  0 = Přerušení od TMR0 zakázáno.
- **INTE** Příznak povolení přerušení od pinu RB0**.**
  1 = Přerušení od RB0 povoleno.
  0 = Přerušení od RB0 zakázáno.
- **RBIE** Příznak povolení přerušení při změně logické úrovně na pinech RB4 až RB7**.**
  1 = Přerušení při změně RB povoleno.
  0 = Přerušení při změně RB zakázáno.

- **T0IF** Příznak požadavku na přerušení při přetečení TMR0**.**
  1 = TMR0 přetekl (bit musí být nulován programově).
  0 = TMR0 ještě nepřetekl.

- **INTF** Příznak požadavku přerušení od pinu RB0**.**
  1 = Je požadavek přerušení od pinu RB0.
  0 = Není požadavek přerušení od pinu RB0.

- **RBIF** Příznak požadavku přerušení při změně na bitech RB4 až RB7**.**
  1 = Byl změněn logický stav na jednom z bitů RB (musí být nulován programově).
  0 = Žádný z bitů **RB** nebyl změněn.

Do registru **INTCON** je zapsána hodnota 0xB0 a to znamená, že přerušení jsou povolena. Je povoleno přerušení od časovače TMR0 a od pinu RB0 a zároveň jsou vynulovány všechny příznaky požadavků přerušení. Tímto jsme naši základní inicializaci ukončili a program nám cyklicky probíhá pouze skok na návěští **Main**. V pravidelných časových intervalech nám však přeteče vždy časovač TMR0 a to má za následek skok na adresu 0x04, což je vygenerování přerušení. Na adrese 0x04 máme umístěnu instrukci skoku na obslužnou proceduru přerušení.

Jak jsme již uvedli v předchozím textu, je nezbytně nutné při obsluze přerušení uchovat hodnotu pracovního registru ***W*** a registru ***STATUS***. Toto se děje na začátku přerušovací procedury. Nejdříve uložíme stav registru ***W*** do proměnné ***Mem_W*** a následuje uložení stavu registru ***STATUS*** do proměnné ***Mem_Stat*** a poté vynulování příznaku požadavku na přerušení od časovače TMR0. Kdybychom toto neprovedli, přerušení by se nám generovalo hned po návratu z této přerušovací procedury a to přesto, že požadavek na přerušení od TMR0 nevznikl. Dále máme vynechané místo na další obslužné instrukce, kde se bude něco dít a těsně před návratem z procedury obsluhy přerušení musíme obnovit stav registru ***W*** a ***STATUS***. To učiníme v opačném pořadí, než když jsme tyto hodnoty ukládali. Nejdříve obnovíme hodnotu registru ***STATUS*** a poté obnovíme hodnotu registru ***W***.

Na závěr přijde instrukce RETFIE, která způsobí návrat z obsluhy přerušení do místa programu, kde toto přerušení vzniklo. Obnovu pracovního registru ***W*** je nutné učinit instrukcí SWAPF, jelikož tato instrukce nemění

žádný z bitů registru ***STATUS***. Aby byla zachována hodnota registru ***W***, protože instrukce SWAPF prohazuje vzájemně horní a dolní 4 bity, musí tato instrukce být vykonána dvakrát. Poprvé s cílem výsledku zpět do registru **Mem_W** a podruhé s cílem do pracovního registru ***W***.

Dle návodu v předchozí kapitole si v simulátoru prostředí MPLAB založte breakpoint na návěští INTPROC a zkuste si program nejdříve odkrokovat a poté spustit příkazem RUN. Obsluha přerušení bude zavolána přibližně za 18 ms. Toto změříme otevřením okna **STOPWATCH**. Tento čas je delší, protože časovač TMR0 nám čítá do 256, nikoliv číslo 156, které je nutné pro nastavení limitu 10 ms.

4.4. Přerušení a multiplexní provoz displeje

☞ *Tuto kapitolu věnuji další části našeho programu, vysvětlím multiplexní obsluhy displeje, dosažení času k měření přerušení.*

Pro snazší pochopení celkového výpisu této etapy programu si nejprve vysvětlíme volané procedury obsluhy displeje.

Pro obsluhu displeje je zapotřebí těchto tří procedur. Procedura **Segment** nám převede znak v binární soustavě na sedmisegmentový kód potřebný k fyzickému vygenerování znaku na displeji. Sedmisegmentový kód je závislý na hardwarovém zapojení displeje naší řídicí jednotky.

Nejlépe si funkci vysvětlíme na ukázce:

```
Segment                     ;  procedura prevede znak na
                            ;  sedmisegmentovy kod
        ADDWF PCL,F         ;  pricteno poradi znaku
        RETLW 0x40          ;  0
```