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

PŘÍKLADY PROGRAMOVÁNÍ MIKROKONTROLÉRŮ HC08 NITRON

Jazyky symbolických adres patří mezi nejstarší programovací jazyky, se kterými se setkáváme již u počítačů první generace, tedy již před několika desítkami let. Dříve, než se u počítačů objevily, programovalo se přímo ve strojovém kódu, tj. většinou v binární, oktalové nebo hexadecimální reprezentaci instrukcí počítače. Všechny objekty, s nimiž počítač pracoval, byly pochopitelně označovány pouze číselně. Např. naplnění střadače konstantou (pro znalé, např. LDA #$14) má ve strojovém jazyce mikrokontroléru HC08 tvar:

1010 0101 0001 0100

Je zřejmé, že programování ve strojovém jazyce je obtížné a nepřehledné a mohlo vyhovovat jen v úplných začátcích, kdy programy byly krátké a jednoduché.

Se zdokonalováním technického vybavení rostly nároky na programy a programování ve strojovém jazyce začalo být neúnosné. Brzy se však zjistilo, že pracnost programování lze značně snížit, zrušíme-li nutnost označovat objekty v instrukcích číselně (vyhovuje stroji) a zavedeme-li symbolické označování objektů (vyhovuje člověku) s tím, že vazbu symbolů na jejich číselné vyjádření nebude provádět programátor, ale zajistí ji specializovaný program – překladač. Tak vznikly jazyky symbolických adres, v nichž se mohly operační znaky instrukcí a jejich operandy označovat symboly, a překladače jazyků symbolických adres (assemblery), které překládaly symbolický jazyk do strojového jazyka.

V dalším vývoji byly do jazyků symbolických adres doplňovány nové prostředky (např. makrojazyk, makra), přičemž vývoj byl obvykle spjat s rozšířením symbolických objektů. Makrojazyk je např. založen na možnosti pojmenovat posloupnost instrukcí, bez makrojazyka bylo možné pojmenovávat pouze objekty v instrukcích.

Brzy se ukázalo, že programátorovi lze práci ještě více usnadnit, zbavíme-li závislosti na instrukční síti počítače, se kterou byl spjat i při programování v jazyce symbolických adres. Objevily se proto jazyky nezávislé na počítači, které jsou navrženy tak, aby programátor instrukční část počítače vůbec nemusel znát. Jazyky symbolických adres však nebyly zcela nahrazeny, neboť stále existují úlohy, které ve vyšších programovacích jazycích nelze vyřešit. Při programování v jazyce symbolických adres se totiž dostáváme do nejužšího styku se systémem, s hardware.

Velké počítače obvykle řeší úlohy, snadno řešitelné pomocí vyšších programovacích jazyků, jednočipové mikropočítače a mikrokontroléry naopak úlohy, řešitelné assemblery. Vyšší programovací jazyky jsou implementovány řadou výrobců a pro mnoho různých procesorů pracujících pod různými operačními systémy, často pro určitý operační systém a procesor existuje i dosti velký počet různých překladačů z téhož jazyka. Různé verze překladačů téhož jazyka se liší rychlostí překladu, velikostí a rychlostí výsledného kódu, komfortem vývojového prostředí, knihovnami atd. Definice vlastního vyššího programovacího jazyka, jeho lexikální symboly, syntaxe, sémantika, je však často dána nějakou normou, ať mezinárodní (např. ANSI C, ANSI C++) nebo podnikovou (Java u SUNu). Proto různé příručky či učebnice např. jazyka C, C++, Javy jsou použitelné při práci s různými překladači. Můžeme říci: programuji v C++, Javě, Pascalu, SQL, C#. Prohlásit „programuji v assembleru“ již není tak jednoznačné. Z toho, co jsme o assemblerech zatím uvedli, je zřejmé, že máme mnoho různých assemblerů. Jednak různé procesory mají různé instrukční soubory, jednak jsou rozdíly i mezi assemblery pro určitý konkrétní typ procesoru. Vždy je nutné prostudovat dokumentaci k příslušnému assembleru.

My se v této kapitole budeme věnovat pouze jednomu assembleru a to CASM08W firmy P&E Microcomputer pro mikrokontroléry řady HC08. Je naprosto odlišný od assemblerů např. pro 8086 či x51, ATMEL AVR nebo MICROCHIP PIC. Na druhé straně bude mít hodně společného s assemblery jiných sw firem pro mikrokontroléry HC08, takže informace z této kapitoly můžeme s určitou dávkou opatrnosti použít i při práci s jinými assemblery pro HC08.

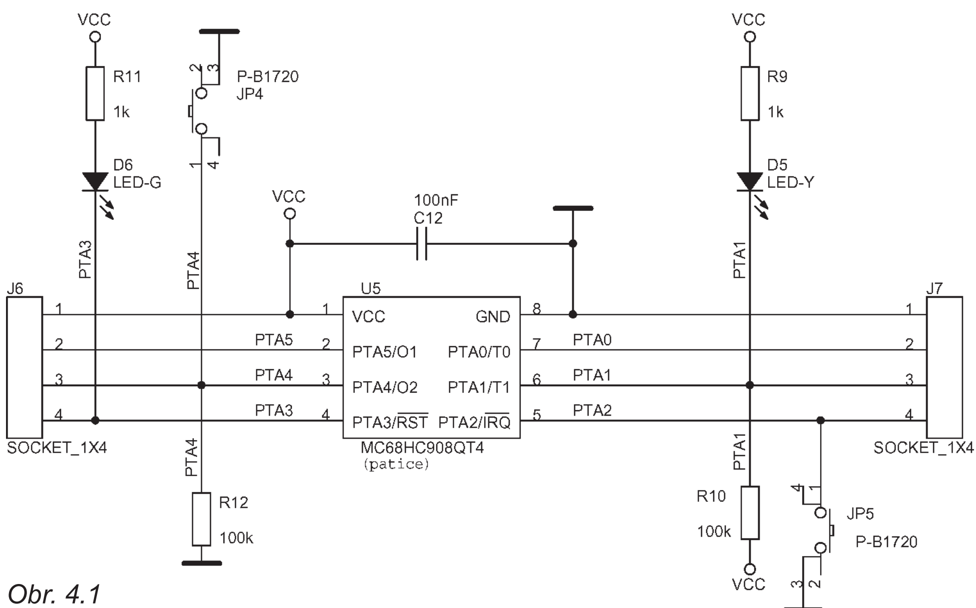
Zmiňovaný assembler je vlastně jen překladač. Potřebujeme dále ještě nějaký editor zdrojových kódů v assembleru a minimálně ještě software pro programování HC08. Naštěstí firma P&E Microcomputer to vše poskytuje zdarma v prostředí **WinIDE** pro 68HC08QT/QY **P&E Micro – ICS08** [8]. Tento vývojový software obsahuje kompletní vývojové prostředí assembleru pro všechny mikrokontroléry řady HC08. Kromě editoru a překladače obsahuje rovněž programátor, simulátor, obvodový simulátor a debugger. Práce s tímto prostředím je popsána i v **české příručce** „Vývojový kit – JANUS uživatelský manuál“, který najdete na CD, stejně jako instalační soubor k tomuto vývojovému prostředí. Proto si v této kapitole ukážeme na několika příkladech, jak programovat v assembleru a jak přeloženým programem naprogramovat mikrokontrolér a tím získat požadovanou aplikaci. V ICS08 můžeme provádět i simulace mikrokontroléru s našimi programy, stejně jako jejich trasování. Postup je dostatečně vyčerpávajícím způsobem popsán v zmiňované příručce.

4.1 První příklad – přepínání LED tlačítky

Příklady programů, které si budeme uvádět v této kapitole, jsou voleny tak, aby byly použitelné i na typ QT, tj. mikrokontrolér s 8 vývody. Kromě dvou pinů pro napájení nám zůstává už jen 6 pinů pro přístup k tomuto mikrokontroléru, viz též *obr. 2.5* z kapitoly 2. Proto mají tyto piny až tři možnosti využití. V daný okamžik však může mít určitý pin jen jendnu z těchto funkcí. Pokud tedy budeme např. používat vnější

zdroj hodinových pulzů, zmenší se nám počet pinů pro vlastní aplikaci. Proto budeme ve všech příkladech používat vnitřní oscilátor. Jeho výrobní přesnost kmitočtu je ±25 %. Oscilátor lze ale jemně doladit pomocí kalibračního registru OSCTRIM. Hodnota kalibrační konstanty je hrubě změřena ve výrobě a pro přesnost pod 5 % ji lze využít. K dispozici je v paměti FLASH na adrese 0xFFC0. Není však chráněna proti vymazání a při prvním smazání celé FLASH paměti se vymaže rovněž. **Proto před prvním programováním ještě nepoužitého čipu, nového mikrokontroléru si tuto konstantu přečtěte a zaznamenejte.** V prostředí ICS08 k tomu použijeme příkaz SM – Show Module a zadáme adresu FFC0 a přečteme si obsah paměti na této adrese – výše zmíněnou konstantu.

Nyní se již pustíme do našeho prvního programu. Budeme předpokládat zapojení podle *obr. 4.1* (vývojový kit JANUS).



Obr. 4.1

Budeme požadovat, aby výsledná aplikace měla dva stavy. V jednom stavu bude svítit jen zelená LEDka, v druhém stavu naopak jen žlutá dioda LED. Každé ze dvou tlačítek bude mít předělen jeden stav a po stisknutí toho tlačítka aplikace přejde do tohoto stavu. Pokud v něm už je, tak další stisknutí tlačítka již nebude mít žádný vliv. Hardwarově bychom mohli takovou aplikaci vytvořit pomocí číslicového klopného obvodu FLIP-FLOP.

Nejprve si ukážeme výsledný zdrojový kód takové aplikace, vysvětlíme si jeho funkci i jednotlivé syntaktické konstrukce. Pak si vysvětlíme, jak k realizaci tohoto programu využít prostředí ICS08.

Výsledný kód:

```
; ******************************************************************
;  priklad01.ASM - prepinani dvou LEDek pomoci tlacitek
;  po prekladu dostaneme priklad01.s19 a ten posleme do MC68HC908QT4
;
;  Cinnost ukazkoveho programu: na startkitu SK8 JANUS ovladat LED diody
;  D5 - zluta  a D6 - zelena pomoci tlacitek JP4 a JP5
;
;  pocatecni stav: D6 - zelena sviti, D5 - zluta nesviti
;  prepnuti do druheho stavu pomoci JP4, do prvniho pomoci JP5
;
;  druhy stav :   D6 - zelena nesviti, D5 - zluta sviti
;
;  vstupy: PTA4 - JP4 (leve tl.) - SET, PTA2 - JP5 (prave tl.) - RESET
;  vystupy: PTA1 - D5 (zluta LED), PTA3 - D6 (zelena LED),
;          LEDky pri nule sviti, pri jednicce nesviti
; ******************************************************************
;
; 3.7.2003

RAMStart     EQU  $0080
RomStart     EQU  $F800

$Include 'NITRON.inc'

        org     RamStart

        org     RomStart

;- MAIN = hlavni program -------------------------------------------
; vstupni bod programu je v tomto miste
Main:

; inicializace CPU  (registers, system configuration atd)
        rsp                     ; stack pointer reset
        clra                    ; akumulator A se vynuluje
        clrx                    ; indexovy rexistr X se vynuluje
        mov     #$31,CONFIG1    ; posle $31 do registru CONFIG1
        mov     #0,CONFIG2      ; IRQ vypnuto, RST vypnuto, OSC je zapnut

; inicializace I/O portu
        lda     #$FF            ; do akumulatoru A posleme $FF
```

```
        sta     PTA             ; posle obsah akumulatotu do PTA = Portu A
        sta     PTB             ; posle $FF do PTB = Portu B
        mov     #$0A,DDRA       ; posila $0A do DDRA
        mov     #0,DDRB         ; posila same nuly do DDRB
        mov     #$14,PTAPUE     ; posila $14 do PTAPUE

; pocatecni nastaveni stavu LEDek
        bset    1,PTA           ; nastavi jednicku na bit 1 Portu A
        bclr    3,PTA           ; vynuluje bit 3 v PTA => D6 sviti

main_loop:                         ; nekonecna smycka
        brclr   4,PTA,main_set     ;
        brclr   2,PTA,main_reset   ;
        bra     main_loop          ;

main_set:                          ; druhy stav
        bclr    1,PTA              ; vynuluje bit 1 v PTA => D5 sviti
        bset    3,PTA              ; nastavi bit 3 v PTA => D6 nesviti
        bra     main_loop          ; skok zpet na main_loop

main_reset:                        ; prvni stav
        bset    1,PTA              ; nastavi bit 1 v PTA => D5 nesviti
        bclr    3,PTA              ; vynuluje bit 3 v PTA => D6 sviti
        bra     main_loop          ; skok zpet na main_loop

;- MAIN ------------------------------------------------------------

;- RESET VECTOR ----------------------------------------------------
; misto pro vektor reset, popr. i pro vektory preruseni

        org     $FFFE
        dw      main             ; FFFE - Reset Vector

;- RESET VECTOR ----------------------------------------------------
```

Stejně jako u jiných programovacích jazyků je i u assembleru pro HC08 dobré doplňovat zdrojový kód poznámkami – komentáři. Slouží k porozumění napsaného kódu, který po nás bude někdo i číst, ale dobrý komentář dobře poslouží i autorovi programu. Po několika měsících si již těžko budeme pamatovat, co jsme zamýšleli při použití nějaké instrukce, různé finty, význam registrů atd. Je dobré okomentovat každou část programu, podprogramu, tabulek.

Komentář začíná středníkem. Vše, co je za ním na stejné řádce bude překladač ignorovat. V případě potřeby napsat komentář na několik řádek je třeba začít kaž-

dou řádku středníkem. Překladač veškeré komentáře ignoruje, výsledný kód je stejný jako kód, v němž bychom tyto komentáře vůbec neměli. Ve výpise zdrojového kódu našeho programu jsem **tučným písmem** vyznačil vše, co není komentář.

První, potřebný kód obsahuje *definice konstant*, takže při překladu, pokud překladač najde jméno **RAMStart**, nahradí ho konstantou **`$0080`**, obdobně místo **RomStart** použije **$F800**. V našich programech budeme používat i další definice, jména konstant jako PTA, CONFIG1, DDRA apod. Pokud budeme používat stále tentýž typ mikrokontroléru, budou ve všech programech tyto definice stejné. Bylo by nesmyslné, do každého našeho programu tyto definice znovu opisovat a navíc by zdrojový kód byl dlouhý, nepřehledný. Proto jsme tyto definice umístili do zvláštního souboru Nitron.inc a *direktivou překladače* **`$Include 'NITRON.inc'`** jsme překladači sdělili, že obsah tohoto souboru má před překladem vložit do tohoto místa. Další *direktivy překladače* **`org RamStart`** a **`org RomStart`** sdělují překladači, kam se má umístit výsledný kód, tj. na kterých adresách bude umístěn program (paměť FLASH) a na kterých data (paměť RAM). Můžete si tyto hodnoty najít na mapě paměti mikropočítače na *obr. 2.3*.

Na konci programu, v místě pro *vektory reset* a popř. i *vektory přerušení* najdeme ještě jednu direktivu **`org $FFFE`** říkající překladači, kde bude vstupní bod programu, adresa, z níž bude program spouštěn a dále je pomocí **`dw main`** definováno pojmenování pro toto místo. V zdrojovém kódu pak bude vstupní, startovní, bod programu pojmenován *návěštím* **`Main:`.**

Na začátku programu bude inicializace mikrokontroléru, tj. počáteční nastavení *ukazatele zásobníku* na *dno zásobníku* pomocí instrukce **`rsp`.** Tato instrukce naplní SP číslem $00FF (všimněte si vyjádření 16bitového čísla v hexaformátu). Paměť RAM našeho mikrokontroléru je alokována v rozmezí adres $0080 až $00FF, takže ukazatel se nastaví na konec RAM. Zásobník se totiž bude plnit směrem k nižším adresám.

Další instrukce **`clra`** a **`clrx`** vynulují obsah střadače A a index registru X – jednoduše naplní tyto registry obsahem $00. Další dvě instrukce slouží k nastavení konfiguračních registrů CONFIG1 a CONFIG2. Konkrétní hodnoty bitů těchto i dalších nastavovaných registrů jsou závislé na tom, jakým způsobem potřebujeme mít mikrokontrolér nakonfigurován a význam těchto bitů najdeme v tištěné či elektronické verzi příručky **MC68HC908AY/QT data sheet** na CD. V našem konkrétním případě pomocí instrukce **`mov #$31,CONFIG1`** pošleme do registru CONFIG1 hodnotu $031, tj. 0011 0001. Bit na nulté pozici označené COPD (COP Disable Bit) provádí zapnutí či vypnutí bloku COP tj. Watch Dogu – jednička znamená, že je vyřazen. Čtvrtý bit, LVIPWRD, mající rovněž úroveň 1 znamená vyřazení LVI a pátý bit LVIRSTD v jedničce je vyřazení i resetu tohoto modulu. Další instrukce pošle do CONFIG2 samé nuly. V důsledku toho je IRQ vypnuto, RST vypnuto a vnitřní oscilátor je zapnut.

U instrukcí **mov** si povšimneme, že první parametr je zdrojem dat (source), druhý cílem (destination).

Následující tři instrukce

```
lda     #$FF
sta     PTA
sta     PTB
```

způsobí to, že nejdříve se pomocí LDA pošlou do střadače samé jedničky a poté se odtud instrukcemi STA tyto jedničky nakopírují do portů PTA a PTB (do portu PTB ovšem jen u typu QY4).

Další instrukce `mov #$0A,DDRA` pošle do registru DDRA hodnotu 0000 1010. Tento registr určuje, které bity portu PTA budou vstupní, a které výstupní. Výstupní budou ty, do kterých jsme poslali jedničky, tj. bit 1 a 3. Máme k nim totiž připojené LED diody.

Další instrukce `mov #$14,PTAPUE` posílá do registru PTAPUE hodnotu `$14`, tj. jedničky do druhého a čtvrtého bitu. Tím se k těmto bitům u portu PTA připojí vnitřní pull-up odpory. K těmto bitům jsou připojena i obě tlačítka. Přes pull-up odpory je totiž na tyto vstupní piny přiváděna úroveň odpovídající logické jedničce, takže stisknutí tlačítka odpovídá nula.

Dále následuje nastavení počátečního stavu

```
bset    1,PTA
bclr    3,PTA
```

Význam těchto instrukcí je ten že, **bset** nastavuje jedničku, **bclr** naopak nulu v bitu, jehož číslo je uvedeno jako první parametr této instrukce v paměťovém prostoru na adrese, která je druhým parametrem. V našem případě je PTA jméno pro adresu odpovídající portu PTA – porty jsou totiž mapovány do paměťového prostoru. Proto na portu PTA bude na prvním bitu nastavena jednička, takže D5 nebude svítit, na bitu 3 bude nula, a proto D6 se rozsvítí.

Nyní jsme se dostali až k návěští `main_loop:`, není to žádná instrukce, tj. nic neprovádí. Je to jen označení pro adresu, na níž se nachází následující instrukce, tj. v našem případě `brclr 4,PTA,main_set`. Funkce této instrukce je ta, že v případě, kdy 4. bit portu PTA bude nulový (tj. když tlačítko je stisknuté), provede se skok na adresu odpovídající návěští `main_set` a začnou se provádět instrukce umístěné za tímto návěštím. Pokud nebude 4 bit portu PTA nulový, tj. tlačítko není stisknuté, skok se neprovede a bude se pokračovat další instrukcí. Její činnost je naprosto obdobná, tj. provádí test druhého tlačítka a v případě jeho stisknutí skok za návěští `main_reset`. Nebude-li ani toto tlačítko stisknuté, bude se provádět následující instrukce, tedy `bra main_loop`. To ovšem není nic jiného, než skok za návěští `main_loop`. Proto při nestisknutých tlačítkách poběží program v nekonečné smyčce.

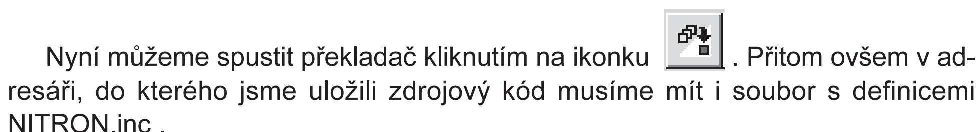
Pokud bude některé z tlačítek stisknuté, provede se odpovídající skok a poté instrukce, nacházející se za návěštím, které bylo cílem skoku. Jsou tam především instrukce **brset** a **brclr**. Jejich funkci jsme si již popsali, takže víme, že způsobí

rozsvícení jedné a zhasnutí druhé diody LED. Poté se již provede skok na začátek nekonečné smyčky, tj. návěští `main_loop`. S našimi současnými znalostmi jsme si jistě všimli, že stisk každého z tlačítek způsobí skok k jiné dvojici instrukcí **brset** a **brclr**, tj. stisk každého z tlačítek způsobí rozsvícení různých LED diod, a pochopitelně i zhasnutí těch zbývajících.

Nyní nám zbývá popsat si konkrétní postup při vytváření tohoto programu: Spustíme prostředí **WinIDE** pro 68HC08QT/QY **P&E Micro** – **IC**, potom v menu vybereme **File --> NewFile**. Otevře se editační okno pojmenované Noname1, do kterého vložíme zdrojový text našeho programu. Obvyklým postupem **File --> SaveFile** uložíme zdrojový text do souboru, který pojmenujeme např. Priklad1.asm:

```
WIN IDE - [DEMO_qtqy.PPF]
File  Edit  Environment  Search  Window  Help

C:\nitron_PRIKLADY\priklad01\priklad01.asm
; *****************************************************************
;   priklad01.ASM - prepinani dvou LEDek pomoci tlacitek
;   po prekladu dostaneme priklad01.s19 a ten posleme do MC68HC908QT4
;
;  Cinnost ukazkoveho programu : na startkitu SK8 JANUS ovladat LED diody
;  D5 - zluta  a D6 - zelena pomoci tlacitek JP4 a JP5
;
;   pocatecni stav: D6 - zelena sviti, D5 - zluta nesviti
;   prepnuti do druheho stavu pomoci JP4 , do prvniho pomoci JP5
;
;   druhy stav :   D6 - zelena nesviti, D5 - zluta sviti
;
;  vstupy: PTA4 - JP4 (leve tl.) - SET, PTA2 - JP5 (prave tl.) - RESET
;  vystupy: PTA1 - D5 (zluta LED) , PTA3 - D6p(zelena LED) ,
;         LEDky pri nule sviti, pri jednicce nesviti
; *****************************************************************
;
; 3.7.2003

RAMStart     EQU  $0080
RomStart     EQU  $F800
VectorStart  EQU  $FFDE

$Include 'NITRON.inc'

        org    RamStart

        org    RomStart

;- MAIN = hlavni program -----------------------------------------------------------------------------
; vstupni bod programu je v tomto miste
Main:

; inicializace CPU  (registers, system configuration atd)

        rsp                     ; stack pointer reset = do ukazatele na zasobnik SP se posle $FF t.j. $00FF
                                ; SP je 16ti bitovy t.j. jeho obsah je popsany ctyrmi hexaznaky
                                ; pozn. RAM 128 bytu je v rozsahu adres $0080 az $00FF , takze ukazatel se
                                ; nastavi na konec RAM , zasobnik se totiz bude plnit smerem k nizsim adtresam
        clra                    ; akumulator A se vynuluje, neboli napni se $00 , pozn. A je 8mi bitovy, proto
                                ; jeho obsah plne urci dva hexaznaky
        clrx                    ;indexovy rexistr X se vynuluje, t.j. naplni se nulami $00

1:1 | Total: 92 | Top: 1 | Bytes: 4960 | Insert
```

Obr. 4.2



Nyní můžeme spustit překladač kliknutím na ikonku . Přitom ovšem v adresáři, do kterého jsme uložili zdrojový kód musíme mít i soubor s definicemi NITRON.inc .

Bude-li v našem programu chyba, označí překladač řádek s první chybou vyskytující se ve zdrojovém kódu – viz *obr. 4.3*.

WIN IDE - [DEMO_qtqy.PPF]

File Edit Environment Search Window Help

C:\nitron_PRIKLADY\priklad01\priklad01.asm

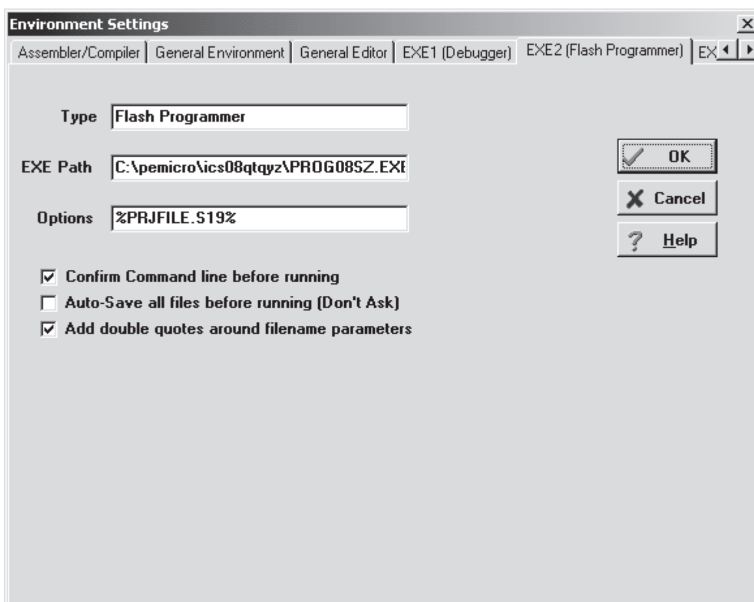
```
;  D5 - zluta  a D6 - zelena pomoci tlacitek JP4 a JP5
;
;   pocatecni stav: D6 - zelena sviti, D5 - zluta nesviti
;   prepnuti do druheho stavu pomoci JP4 , do prvniho pomoci JP5
;
;   druhy stav :   D6 - zelena nesviti, D5 - zluta sviti
;
;  vstupy: PTA4 - JP4 (leve tl.) - SET, PTA2 - JP5 (prave tl.) - RESET
;  vystupy: PTA1 - D5 (zluta LED) , PTA3 - D6p(zelena LED) ,
;          LEDky pri nule sviti, pri jednicce nesviti
; ******************************************************************
;
; 3.7.2003

RAMStart     EQU  $0080
RomStart     EQU  $F800
VectorStart  EU  $FFDE

$Include 'NITRON.inc'

        org     RamStart

        org     RomStart

;- MAIN = hlavni program --------------------------------------------------------------------------------
; vstupni bod programu je v tomto miste
Main:

; inicializace CPU  (registers, system configuration atd)

        rsp                     ; stack pointer reset = do ukazatele na zasobnik SP se posle $FF t.j. $00FF
                                ; SP je 16ti bitovy t.j. jeho obsah je popsany ctyrmi hexaznaky
                                ; pozn. RAM 128 bytu je v rozsahu adres $0080 az $00FF , takze ukazatel se
                                ; nastavi na konec RAM , zasobnik se totiz bude plnit smerem k nizsim adtresam
        clra                    ; akumulator A se vynuluje, neboli napni se $00 , pozn. A je 8mi bitovy, proto
                                ; jeho obsah plne urci dva hexaznaky
        clrx                    ;indexovy rexistr X se vynuluje, t.j. naplni se nulami $00
        mov     #$31,CONFIG1    ; posle $31 t.j.  0011 0001 do registru CONFIG1 t.j. na adresu $001F
                                ; tim se nastavi LVI = Low-Voltage Inhibit , zakaz nizkeho napeti, neboli LVI
                                ; bude v 3V modu a dale se nastavi podpora COP = Computer Operating Properly
                                ; COP je znam tez pod jmenem WatchDog
        mov     #0,CONFIG2      ; IRQ vypnuto, RST vypnuto, vnitrni OSCilator je zapnut
```

Assemble Compile File - Hotkey F4

Obr. 4.3

Po opravě chyb a následném překladu dostaneme jako produkt úspěšného překladu soubor se stejným jménem, jako je jméno zdrojového souboru. Tento výsledný soubor bude mít koncovku (extenzi) S19. Obsahuje absolutní kód, tj. přeložený program do operačních kódů instrukcí s jejich absolutním umístěním v programové paměti. Formát tohoto souboru je popsán v knížce o HC11 [4]. Tento výsledný kód použijeme k naprogramování mikrokontroléru. Vývojový kit JANUS nebo obdobný připojíme pomocí sériového kabelu k PC.

Poznámka: *Po nainstalování P&E vývojového prostředí je nastaveno, že jméno výsledného S19 souboru obsahuje ve svém názvu řetězec PRJFILE.S19. Raději toto omezení odstraníme ještě před prvním spuštěním programátoru. Provedeme to následujícím způsobem:*
V menu vybíráme ***Environment --> SetupEnvironment...*** *. Objeví se okno, ve kterém vybereme záložku* ***EXE2 (Flash Programmer)*** *– viz obr. 4.4.*

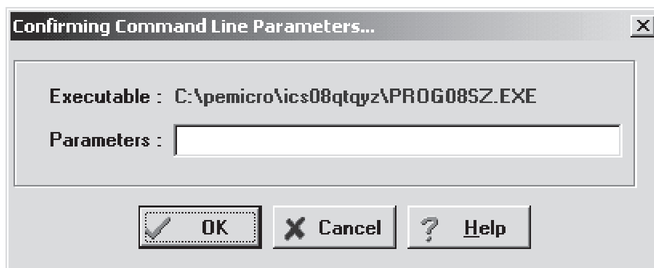


*Obr. 4.4 Okno **Enviroment Settings** ještě před vymazáním hodnoty v poli **Options***

V tomto okně v **Options** vymažeme jeho obsah a potvrdíme **OK**.

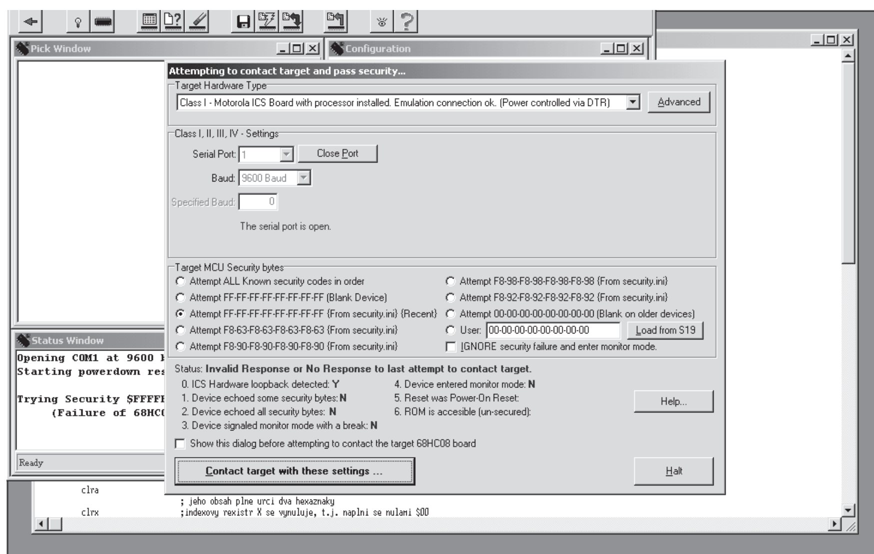
Nyní již naprogramujeme paměť flash naším programem:
Na startkitu propojíme JP1 – na PTA2 je přiveden signál DTR ze sériového portu COM počítače PC (pochopitelně po převodu úrovní RS232 – TTL v obvodu HIN232). Dále máme propojeny 1 a 2 na JP2, tj. na PTA5 je přiveden signál 9,8304 MHz z vnějšího oscilátoru a ještě je propojen, JP3 – PTA0 je tak připojen k signálům RXout, TXin u HIN232. Přepínač SW2 je nestisknut – tj. je navolen **MONITOR MODE**. Ještě pro jistotu provedeme reset stisknutím tlačítka SW1 (na chvíli se tím odpojí napájení a po jeho opětném připojení se provede reset).

Nyní již můžeme kliknout na ikonku Flash Programmer . Objeví se *obr. 4.5.*



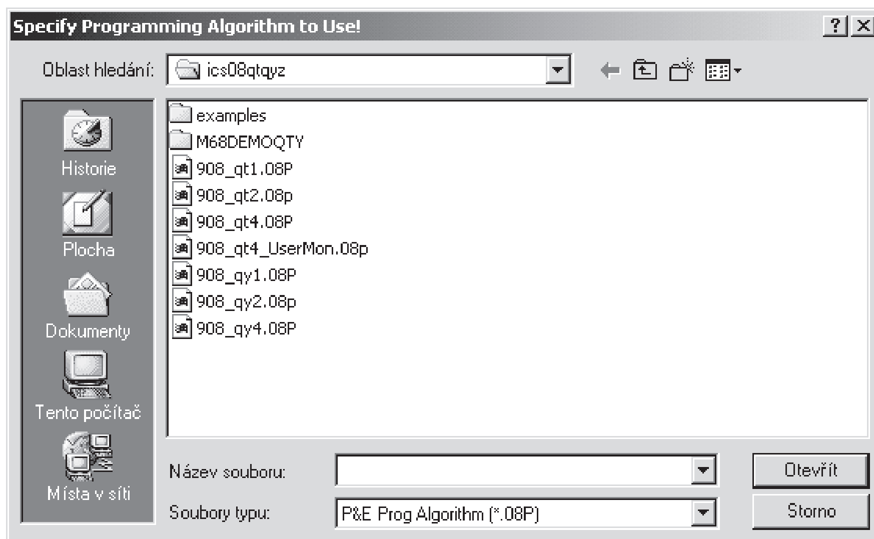
Obr. 4.5

Žádné parametry nevyplňujeme, jen potvrdíme OK. Při prvním spuštění programátoru se objeví:



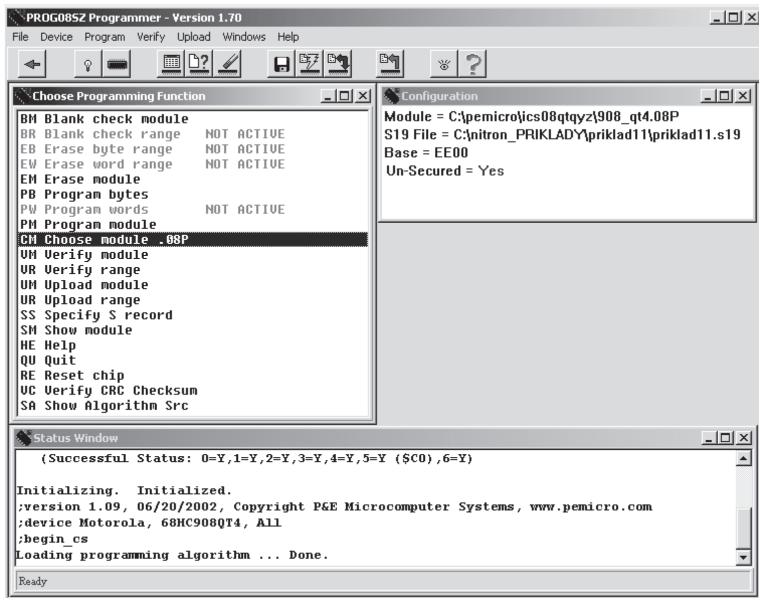
Obr. 4.6

Stiskneme tlačítko **Contact Target with these settings...** a poté již dostaneme:



Obr. 4.7

V něm vybereme **908_qt4.08P** a stiskneme **Otevřít**. Objeví se okno:

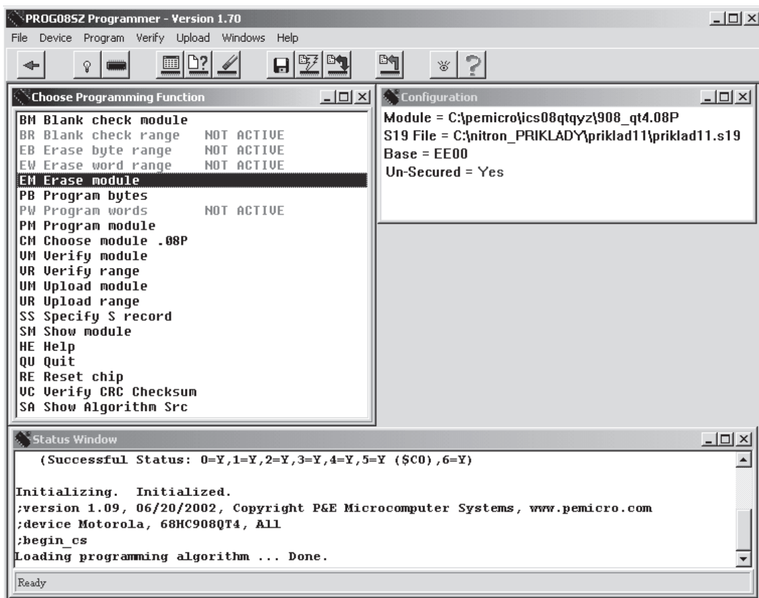


Obr. 4.8

Pokud budeme Programátor spouštět příště, mělo by se objevit toto okno okamžitě po kliknutí na ikonku **Flash Programmer**.

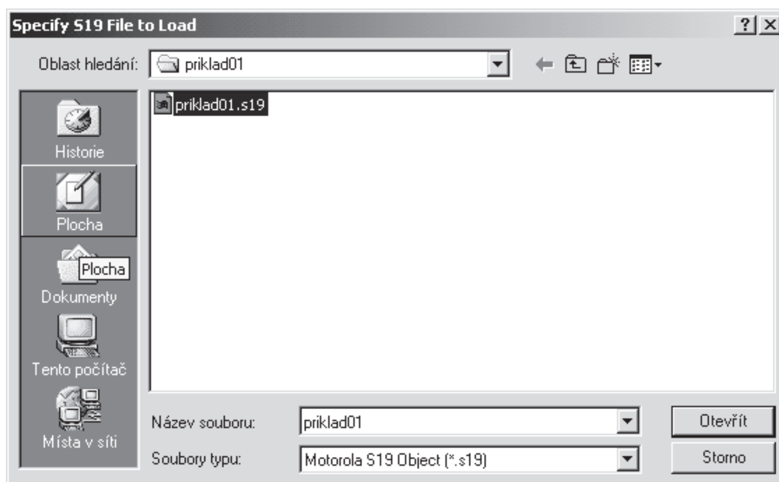
Objeví-li se ale okno **Attempting to Contact target and pass security**, je to obyčejně tím, že jsme zapomněli přepnout z **uživatelského módu** do **monitor mode**.

Bude-li tedy vše v pořádku, spustíme příkaz **EM Erase module** výběrem v okně:



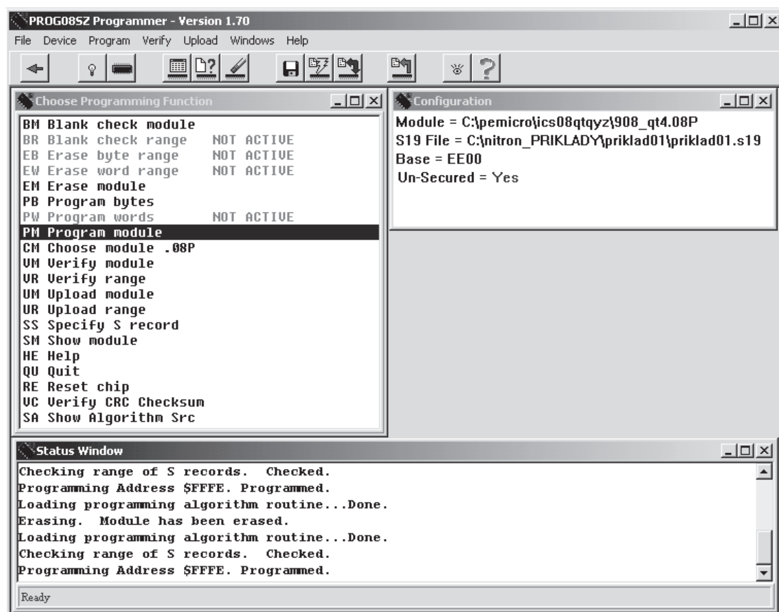
Obr. 4.9

a poté ještě vybereme příkaz **SS Specify S record** a vybereme soubor, jehož obsahem chceme naprogramovat mikrokontrolér:



Obr. 4.10

Potvrdíme **Otevřít** a v okně **Choose Programming function** vybereme příkaz **PM Program Module**. Úspěšné naprogramování mikrokontroléru je potvrzeno výpisy do **Status Window**.



Obr. 4.11

Nyní stisknutím SW2 přepneme do **User Mode**. Musíme ještě rozpojit JP1 – jinak bude trvale svítit zelená LED D6. Nakonec stiskem SW1 zresetujeme mikrokontrolér a ten bude vykonávat náš program.

JP2 a JP3 ve většině našich příkladů nebudeme přepínat, tj. budou nastaveny stejně v User Mode i v Monitor Mode. Pouze v případě, kdy budeme používat A/D převodník a budeme jeho vstup připojovat k odporovému trimru R7, budeme v user mode JP2 přepínat do polohy spojující špičky 2 a 3.

V popise dalších příkladů se budeme zabývat již jenom popisem zdrojových kódů, postup programování bude stejný jako u tohoto prvního programu.

Pro jistotu si ještě zopakujeme, že nesmíme zapomínat na přepínání mezi User Mode a Monitor Mode, a na spojování a rozpojování JP1.

4.2 Druhý příklad – blikač, použití podprogramů

V tomto programu se pokusíme naprogramovat blikač, který bude pracovat tak, že střídavě bude svítit jedna z diod D4 a D5 v témže zapojení, jako v prvním příkladu. Začátek programu obsahující inicializaci procesoru a inicializaci portu bude stejný, jako v předchozím. Proto tuto část překopírujeme z předchozího příkladu. Takto budeme postupovat i v příkladech 3 až 8, takže si dále inicializační část nebudeme popisovat.

Po inicializaci program poběží v nekonečné smyčce, tak jak tomu bylo v prvním programu a stejně tomu bude i ve všech následujících programech. Stejně jako v prvním programu budeme mít mimo nekonečnou smyčku umístěny dva úseky kódu rovněž označené `main_set` a `main_reset`. V těchto úsecích jsou umístěny instrukce **bset** a **bclr** sloužící k nastavení či vynulování příslušného bitu v paměťovém prostoru, v našem případě v portu PTA. Porovnáme-li `main_set` a `main_reset` v příkladu 2 se stejně pojmenovanými úseky v předchozím příkladu 1, vidíme, že jejich funkce je stejná, tj. přivést naši aplikaci do jednoho ze dvou stavů. Je zde však jeden podstatný rozdíl. V příkladu 1 se odskok z hlavní, nekonečné smyčky do těchto úseků programu provedl jen při stisknutém tlačítku a návrat se **provedl na začátek nekonečné smyčky**.

V našem druhém příkladu se však požaduje odskok při každém průchodu nekonečnou smyčkou. Pokud by návrat z těchto úseků programu byl na začátek nekonečné smyčky, nikdy by nedošlo k provádění kódu umístěném v úseku `main_reset`, protože v hlavní smyčce je nejprve umístěn odskok do `main_set`, tj. vůbec by nedocházelo ke změně stavů.

K tomu, aby aplikace byla funkční, a vytvořili jsme blikač, je třeba, aby po návratu z některého ze zmiňovaných úseků programu **pokračoval program za tímto úsekem**. Docílíme toho tak, že tyto úseky napíšeme jako podprogramy. Pro skok do podprogramu, *volání podprogramu*, použijeme instrukci `jsr`. *Návrat z podprogramu na místo následující za místem volání podprogramu provede instrukce* `rts`. Pokud bychom v předchozím příkladu nahradili instrukce testující tlačítka instrukcemi