Integrované vývojové prostředí

# WinAVR

# 1 Obsah

# 2 Charakteristika prostředí WinAVR

Jde o komplexně řešený systém založený na moderní řadě mikroprocesorů AtmelAVR, určený pro výuku mikroprocesorové techniky, Tento systém se skládá z vlastního vývojového prostředí, HW podporu zajišťují emulátory a výukové moduly, pro realizaci vlastních projektů s využitím procesorů Atmel jsou k dispozici programátory. Pro využití v laboratorním prostředí je implementována možnost síťového přenosu, která umožňuje z libovolného počítače přístup na libovolný emulátor, čímž klesají pořizovací náklady při zachování dostupnosti hardwaru pro všechny uživatele. Softwarová část obsahuje programové moduly počínaje editorem kódu, překladačem, debuggerem až po praktické pomůcky pro nastavování a definování obsahů IO registrů. HW část obsahuje řadu modulů, zahrnující jednoduché přípravky pro práci s porty jako LED, tlačítka, klávesnice, přes DA a AD převodníky, až po moduly využívající multiprocesorovou komunikaci, bezdrátovou komunikaci, příp. komunikaci s PC a USB. Tyto moduly se připojují k HW emulátorům, které jsou rovněž součástí našeho systému. Součástí systému je i programátor procesorů Atmel AVR, který umožňuje jak klasické paralelní programování, tak programování přes ISP, tím umožňuje programovat procesor přímo ve Vaší aplikaci. Celá stavebnice je založena na procesorech AtmelAVR, jejichž cena je velmi příznivá a jejich sortiment velice široký. O těchto procesorech je současně k dostání velké množství literatury. Ke každému přípravku je k dispozici demoprogram, můžeme rovněž nabídnout výukové materiály pro mikroprocesorovou techniku.

## 2.1 Charakteristika vývojového prostředí:

- editor zdrojového textu s barevným zvýrazněním syntaktických symbolů a intelisense nápovědou
- kompilátor jazyka mikroprocesorů řady AVR, včetně knihoven symbolů jednotlivých typů procesorů
- debugger pro analýzu programu, umožňující krokování a trasování programu, zobrazení IO registrů a obsahů všech pamětí
- možnosti definovat vstupní průběhy i záznamy výstupních stavů a jejich následné zobrazení v libovolném časovém měřítku, snadné odečítání doby mezi jednotlivými událostmi. Toto je ideální pro výhodné pro ladění a demostraci činnosti dynamického displeje, sériového kanálu, komunicaci s PC
- emulátory umožňují připojení výukových a vývojových modulů nebo přímé připojení do Vašeho zařízení s možností ladění programu přímo ve vyvíjeném zařízení
- vlastní výukové a vývojové moduly jako displeje, klávesnice, převodníky, paměti atd.síťový přenos umožňující vícenásobné využití připojených výukových modulů
- programátory ProgAVR, umožňuje programovat všechny typy mikroprocesorů AtmelAVR, rovněž umožňuje programovat procesor v přímo ve Vaší aplikaci přes rozhraní SPI
- Operační systém W98, W2000, Windows XP

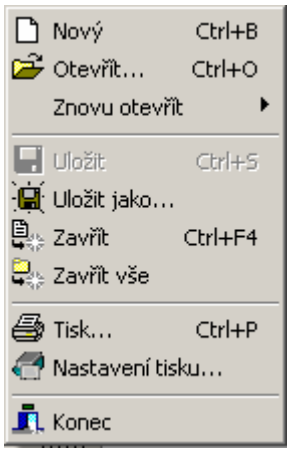
# 3 Popis aplikace

## 3.1 Hlavní menu

Hlavní menu aplikace WinAVR obsahuje tyto položky, práce s ním odpovídá standardu Windows.

### 3.1.1 Menu Soubor

Slouží ke správě dokumentů, otevírání, ukládání, zavírání a tisku souborů.

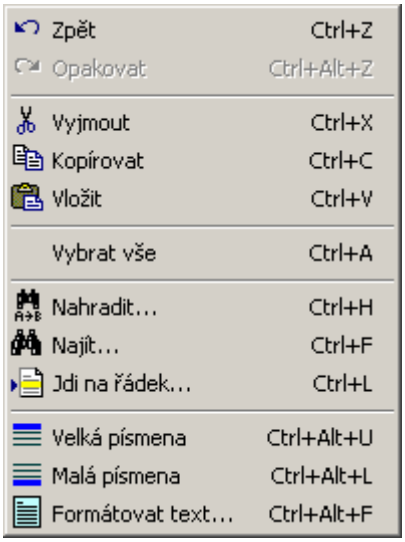



- **Nový, Ctrl+B:** založí nový soubor beze_jména.asm. Nový soubor lze otevírat i pomocí místní nabídky záložky souboru. Je-li ve stejné složce jako WinAVR.exe soubor WinAVR.asm, objeví se dotaz, zda má být tento soubor otevřen jako šablona.
- **Otevřít, Ctrl+O:** otevírá soubory s příponou .asm, .lst. bin, .hex. Soubor s příponou .lst je již přeložený program, který se poté načítá do debuggeru.
- **Znovu otevřít**: zde je možno si vybrat z deseti naposledy otevíraných souborů.
- **Uložit, Ctrl+S:** ukládá soubor, jestliže došlo ke změně. V menu *Nastavení* v položce *Nastavení editoru* je možno zadat, že se má soubor ukládat automaticky po zadaných minutách. Jestliže soubor nemá jméno vyvolá se okno *Uložit jako*, kde je nutné zadat jméno souboru a vybrat příslušný disk a adresář, do kterého se soubor uloží.
- **Uložit jako:** uloží soubor pod zadaným jménem. Tuto akci lze také vyvolat pomocí místní nabídky záložky souboru.
- **Zavřít, Ctrl+F4:** zavře aktuální soubor, kontroluje zda-li je uložen. Tuto akci lze vyvolat i pomocí místní nabídky pracovní plochy nebo místní nabídky záložky souboru.
- **Zavřít vše**: zavře všechny otevřené soubory, kontroluje zda-li jsou uloženy.

- **Tisk, Ctrl+P:** otevře se okno pro tisk známé z Windows. Můžeme tisknout celý soubor nebo jen vybranou část. Tiskne se barevně nebo černobíle podle nastavení v menu *Nastavení* v položce *Nastavení editoru*.
- **Nastavení tisku:** otevře se okno pro nastavení tisku známé z Windows. Je možné vybrat název tiskárny, formát, zdroj a orientaci papíru a nastavit vlastnosti tiskárny.
- **Konec:** ukončí program.

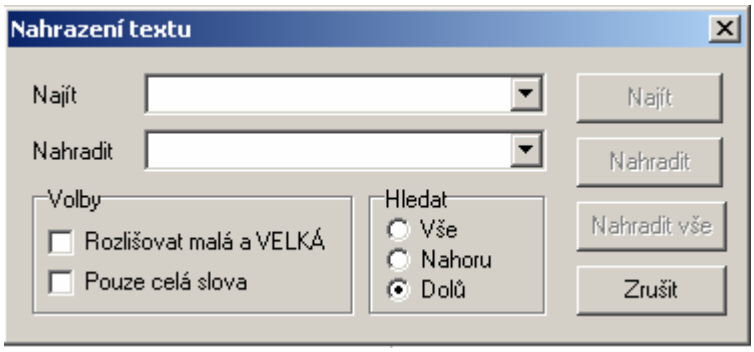
### 3.1.2 Menu Úpravy

Slouží k úpravě dokumentů, práci se schránkou, formátování, hledání a nahrazování textu.

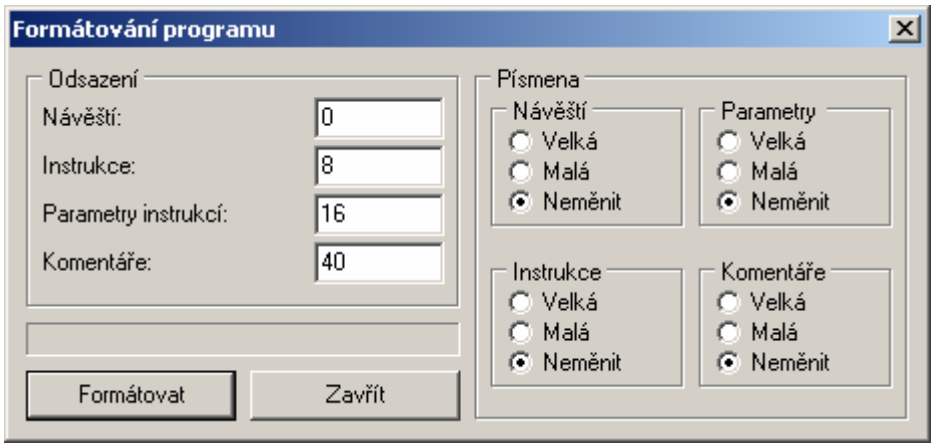



- **Zpět, Ctrl+Z, Alt+BackSpace:** vrátí naposledy provedenou akci.
- **Opakovat, Ctrl+Alt+Z:** vrátí naposledy smazanou akci.
- **Vyjmout, Ctrl+X:** označený text se přesune do schránky.
- **Kopírovat, Ctrl+C:** označený text se překopíruje do schránky.
- **Vložit, Ctrl+V:** text uložený ve schránce se vloží do editoru.
- **Vybrat vše, Ctrl+A:** označí se text celého souboru.
- **Nahradit, Ctrl+H:** otevře se okno *Nahrazení textu*. Do položky *Najít* se zadá hledaný text. Při otevření okna je do položky Najít zadáno automaticky slovo, na kterém je umístěn kurzor. Do položky *Nahradit* se zadá náhrada za nalezený text. Při stisku jakéhokoliv tlačítka se texty uloží do seznamů, které pak můžeme znovu vybrat. V okně Hledat udáme směr z hlediska umístění kurzoru (*Dolů*, *Nahoru*) a *Vše*, kde proběhne prohledávání od prvního znaku prvního řádku. Zaškrtávací políčka Pouze celá slova a *Rozlišovat malá a VELKÁ* určují způsob prohledávání. Tlačítko *Najít další* najde text a označí ho, pokud text nenajde do stavového řádku napíše *Text nenalezen*. Tlačítko *Nahradit* nahradí nalezený text a označí další. Tlačítko *Nahradit vše* nahradí všechny nalezené výskyty textu

- **Najít, Ctrl+F:** otevře se okno *Nalezení textu*, které funguje obdobně jako okno *Nahrazení textu*.




- **Jdi na řádek, Ctrl+L:** přejde na zadaný řádek v dokumentu.
- **Malá písmena, Ctrl+Alt+L:** převede označený text na malá písmena, při současném stisku klávesy Shift se změna neprovede na komentáře, tzn. od znaku ; doprava.
- **Velká písmena, Ctrl+Alt+U:** převede označený text na velká písmena, při současném stisku klávesy Shift se změna neprovede na komentáře, tzn. od znaku ; doprava
- **Formátovat text, Ctrl+Alt+F:** slouží pro naformátování již napsaného zdrojového textu, bližší popis funkce viz obrázek. Formátování je možné volat automaticky při překladu programu, viz *Nastavení editoru*




Některé běžné operace jako *Zpět*, *Vyjmout*, *Kopírovat*, *Vložit* lze provádět i pomocí místní nabídky editoru, pokud stisknete pravé tlačítko myši na pracovní ploše editoru. Kopírování a přesouvání textu je možno provádět i pomocí myši. Vybereme text, se kterým chceme pracovat, uchopíme ho myší a táhneme na požadované místo, jestliže chceme text kopírovat a ne pouze přesouvat stiskneme přitom ještě klávesu Ctrl.

### 3.1.3 Menu Překladač

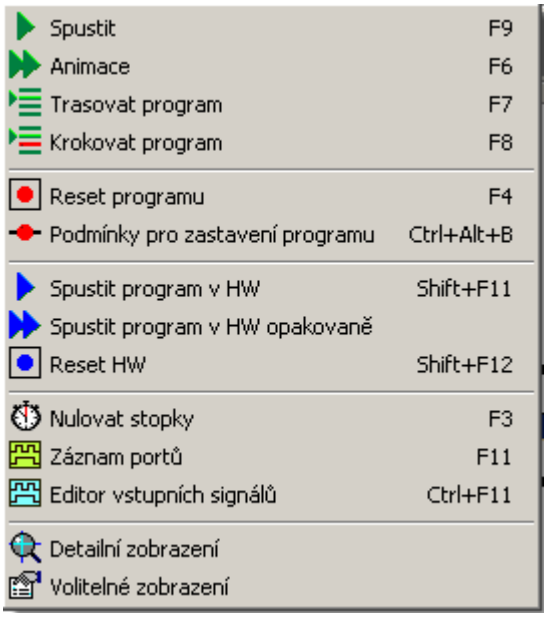
Přeloží editovaný soubor. Zobrazí se informativní znaky na liště řádků, otevře se protokol o překladu a aktualizuje se výpis použitých symbolů. Více v kapitole *Kompilátor*.

### 3.1.4 Menu WinProg

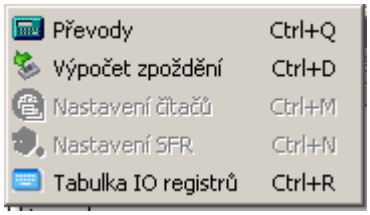
Odešle přeložený kód do emulátoru nebo programátoru. Průběh přenosu indikuje ukazatel. Umožňuje také načtení obsahu paměti již naprogramovaného procesoru. Více v kap. *Nastavení přenosu*


### 3.1.5 Menu Debugger

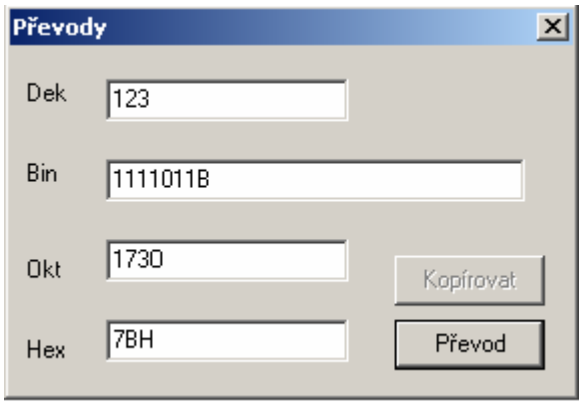
Nastavení parametrů pro debugger. Více v samostatné kapitole *Debugger*



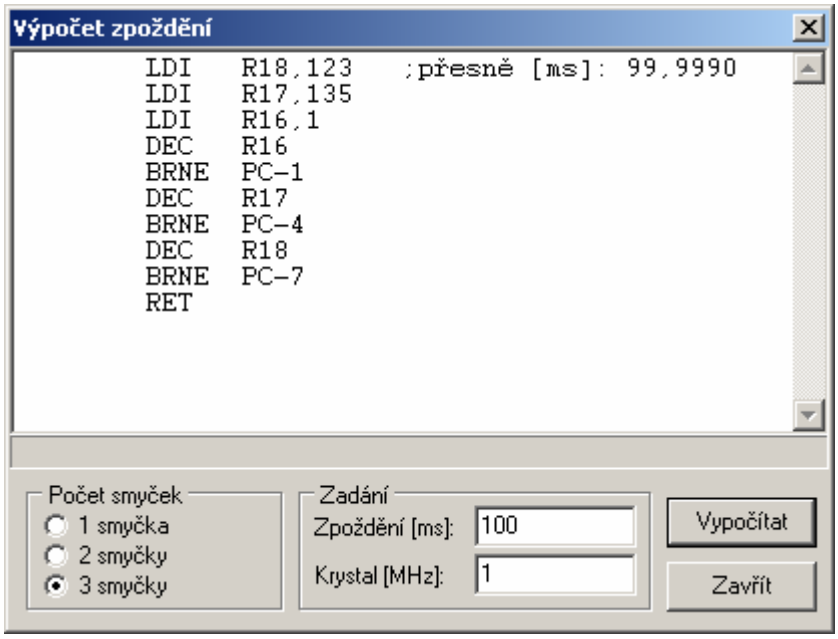

### 3.1.6 Menu Pomůcky



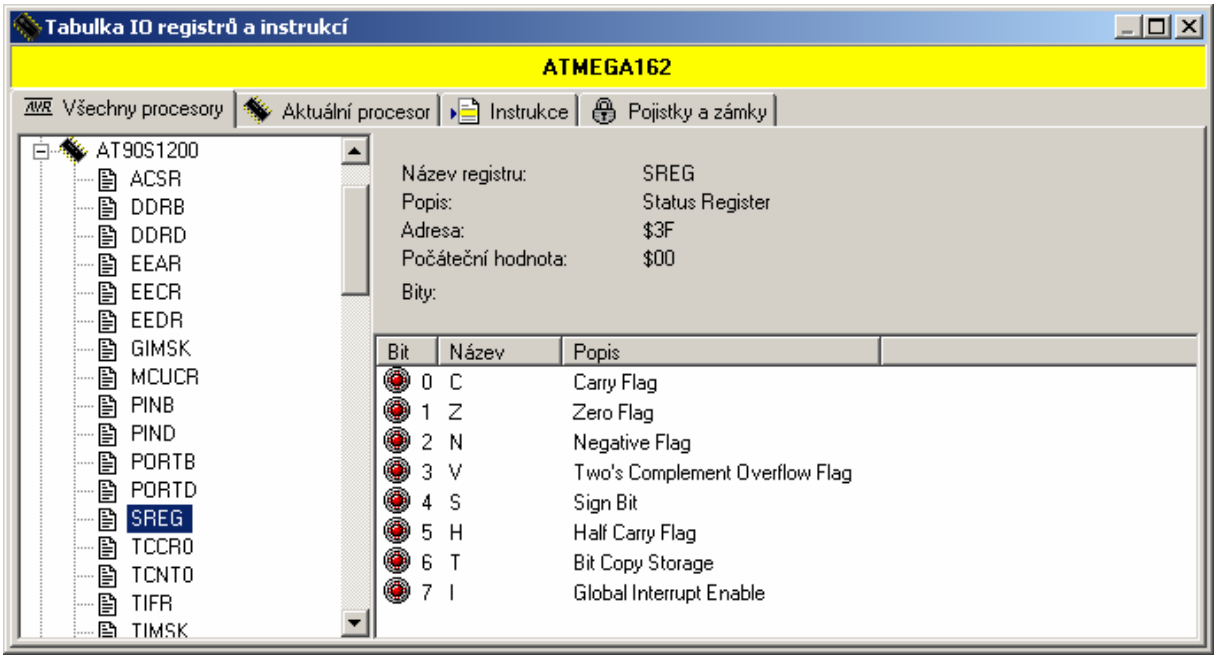

- **Převody, Ctrl+Q:** otevře se okno Převody. Umožňuje převody mezi číselnými soustavami. Do libovolného řádku napíšeme číslo v dané soustavě. Tlačítko *Převod* ho převede do ostatních číselných soustav. Tlačítko *Kopírovat* označené číslo překopíruje do editoru na místo kurzoru.




- **Výpočet zpoždění, Ctrl+D:** slouží pro rychlý výpočet zpožďovací smyčky. Umožňuje vypočtené hodnoty ve formě zdrojového textu přímo vložit do editoru. Hodnota použitého krystalu se zadává v MHz a je přebírána z nastavení debuggeru.

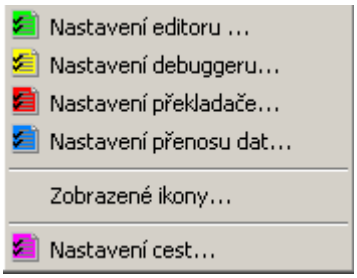



- **Tabulka IO registrů, Ctrl+R:** slouží pro rychlé a přehledné nastavení IO registrů. Obsahuje přehled všech IO registrů ve všech dostupných typech procesorů, jejich adresy, počáteční hodnoty, popis bitů a také obsahuje popis instrukčního souboru.

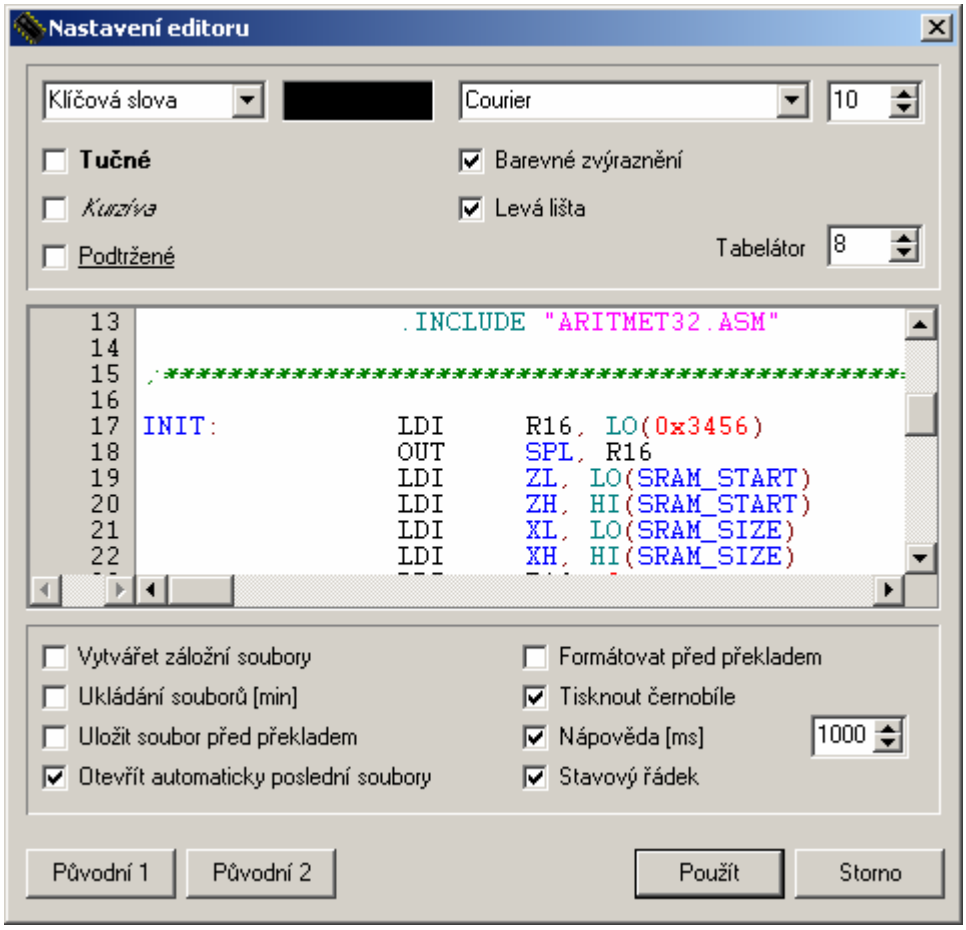



### 3.1.7 Menu Nastavení

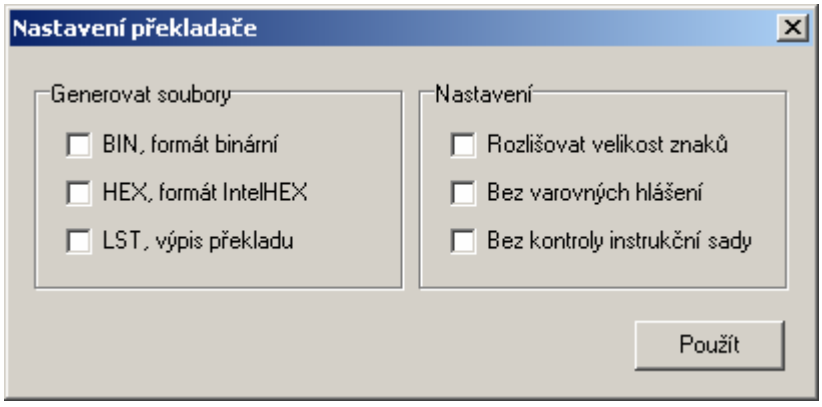
Slouží k celkovému nastavení všech vlastností programu, ukládá se do INI souboru a použije při příštím spuštění.



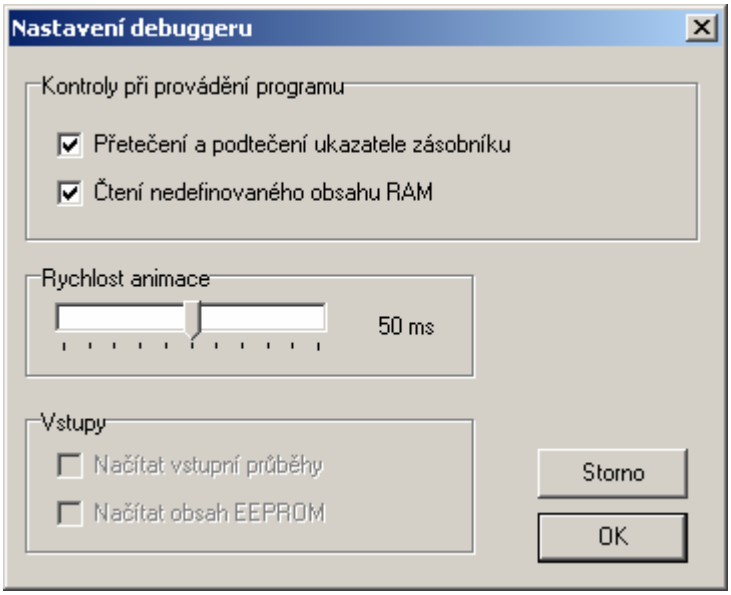

- **Nastavení editoru** – Zde můžeme nastavit barevné zvýrazňování a jednotlivé barvy u všech symbolů, velikost a řez písma. Dále můžeme nastavit ukládání záložních souborů s příponou .bak. Můžeme zadat po jaké době se má soubor automaticky ukládat. Zda se mají otevírat naposledy zpracovávané soubory. Zda-li se má zobrazovat stavový řádek a lišta s čísly řádků. Je možno nastavit zda se má tisknout černobíle nebo podle barevného zvýraznění. Tlačítko *Původní1, Původní2* slouží k rychlému nastavení předvoleného zobrazení. Tlačítko *Použít* aplikuje naše nastavení v editoru. Tlačítko *Storno* naše nastavení nepoužije.



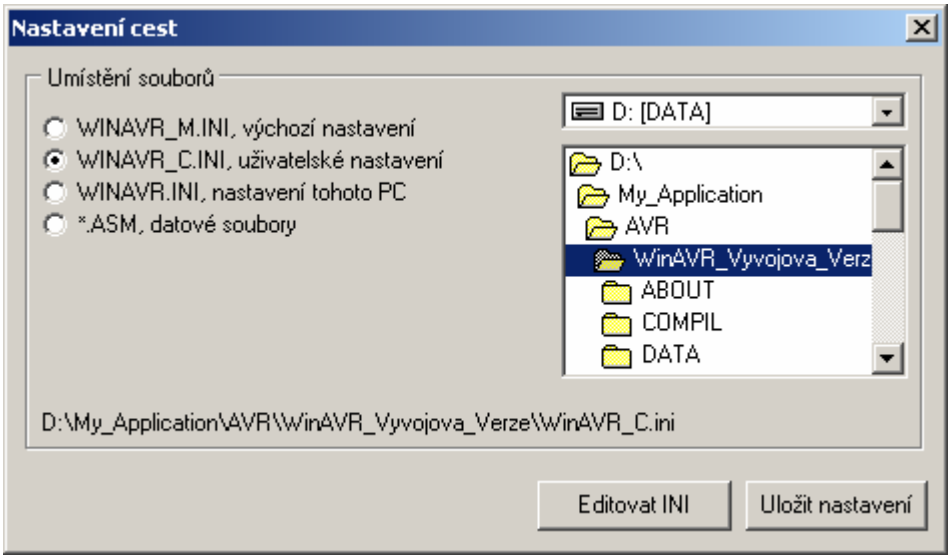

- **Nastavení překladače**: otevře okno s nastavením parametrů překladače. Lze volit typy výstupních souborů, BIN je čistě binární soubor obsahující pouze data, HEX je soubor ve formátu IntelHEX obsahující navíc kontrolní součty, LST je textový protokol o překladu. Lze zakázat generování varovných hlášení při překladu, vypnout kontrolu instrukční sady a také zvolit možnost rozlišení velikosti znaků v identifikátorech.



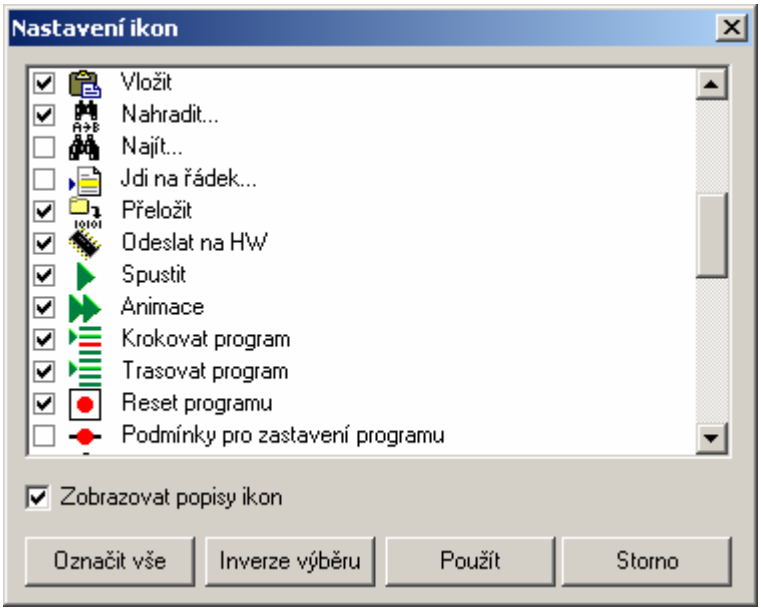

- **Nastavení debuggeru**: otevře okno s nastavením parametrů debuggeru. Při používání editoru vstupních signálů musí být zde zapnuto *Načítat vstupní průběhy*.



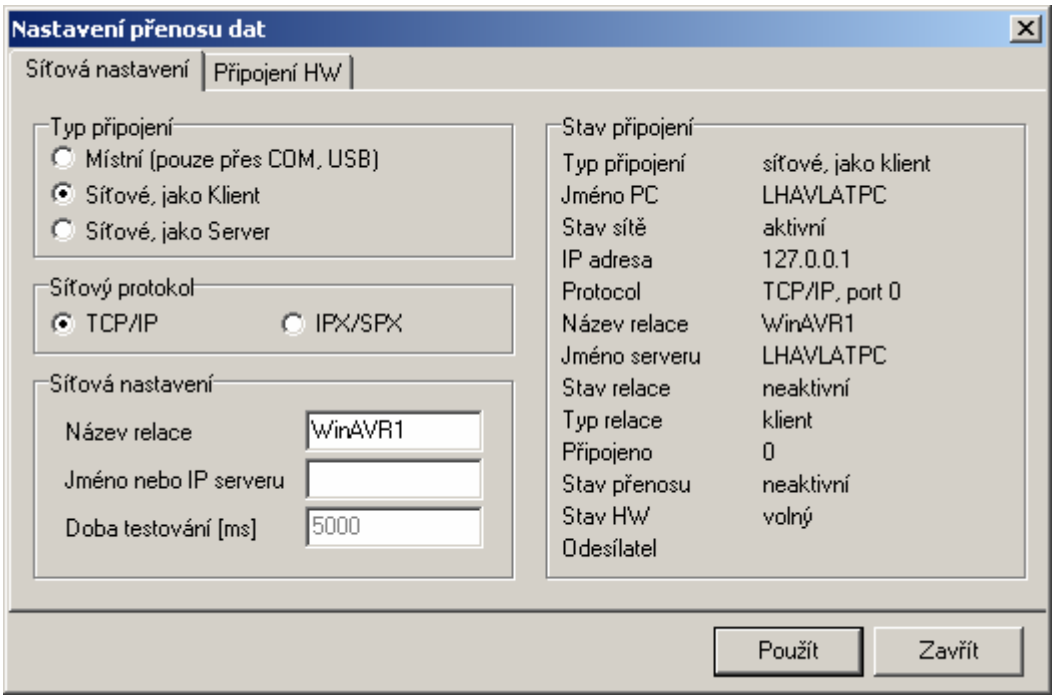

- **Nastavení cest:** otevře se okno, ve kterém nastavujeme cestu k uložení inicializačních a datových souborů. Zde je možné i zobrazit a měnit obsahy jednotlivých konfiguračních souborů. Cestu k WINAVR_M.INI nelze měnit a soubor musí být vždy ve stejné složce jako aplikace. Více v kapitole *Popis INI souborů.*




- **Zobrazené ikony**: otevře se okno, ve kterém nastavujeme tlačítka, která se mají zobrazit v panelech nástrojů. Zadáváme zda-li má u tlačítek zobrazit jejich popis. Řazení tlačítek je automatické a nelze jej ovlivnit ani měnit




- **Nastavení přenosu dat:** tato nabídka je poněkud složitější proto je popsána podrobněji:

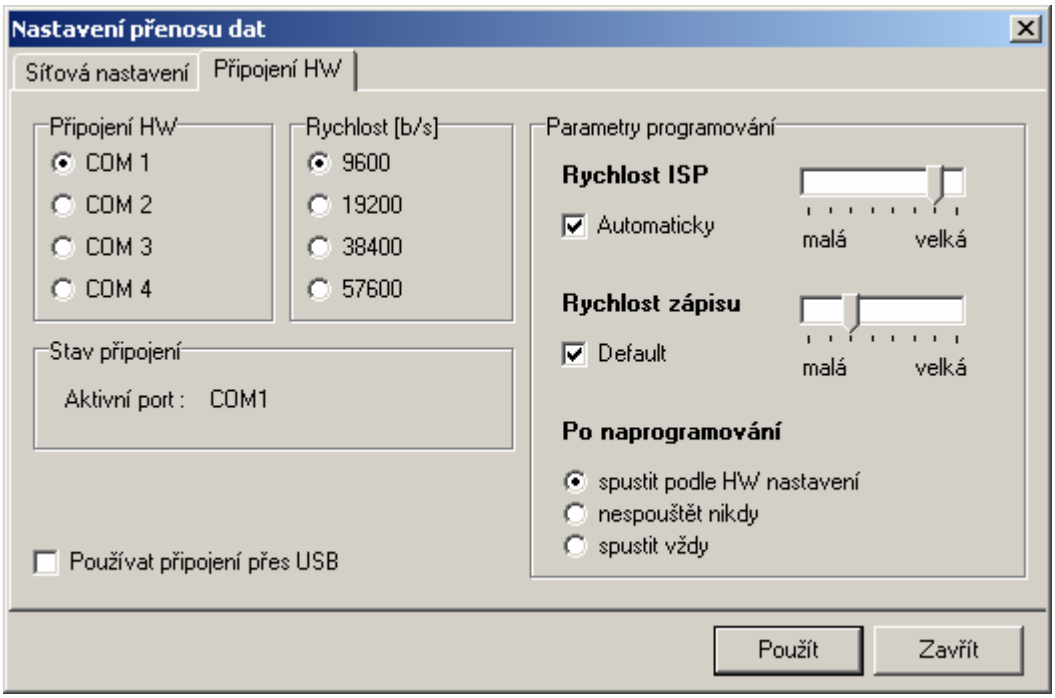



Připojení HW je možné třemi způsoby:

1. bez využívání sítě přímo na COM nebo USB
2. jako Klient, data jsou posílána prostřednictvím sítě na jiný počítač, který se chová jako server a k němuž je připojen emulátor

3. jako Server, pokud je zde připojen emulátor, pro místního uživatele je síťové připojení zcela transparentní, tzn. že program WinAVR umožňuje normální práci i v okamžiku přenosu dat z klientského PC do emulátoru.

Je-li daný počítač využíván jako klient, je třeba vyplnit název relace a jméno nebo IP adresu počítače, který má sloužit jako server jemuž se budou posílat data. Počet relací a tedy počet serverů na jedné síti není omezen, není omezen ani počet klientů. K vlastnímu přihlášení dojde po stisku tlačítka Použít nebo prvním odesláním dat. Je-li daný počítač využíván jako server, je třeba vyplnit pouze název relace, jméno ani adresu nevyplňujeme. Navíc je možné zadat dobu testování. Je to doba, po kterou nemůže nikdo přepsat testovaný program, který už byl odeslán do emulátoru. V tomto stavu se pokus o přenos dat neuskuteční a vypíše se hlášení *HW není dostupný*. Pouze ten, jehož program se momentálně provádí jej může znovu přepsat ještě před uplynutím této doby. Informační pole vpravo poskytuje informace o jménu PC, IP a aktuálním stavu připojení. Tyto informace jsou automaticky aktualizovány s intervalem asi 1s. Druhá záložka tohoto okna, Připojení HW, slouží pro výběr portu na kterém bude připojen emulátor nebo programátor. Rychlost přenosu dat je standardně 9600b/s, pokud nepoužíváme programátor nebo emulátor např. s optickým oddělením, které rychlost omezuje, je možné ji zvýšit. Programátor se přizpůsobí automaticky. Rychlost ISP je nejlépe ponechat na Automaticky, podobně rychlost zápisu na Default.



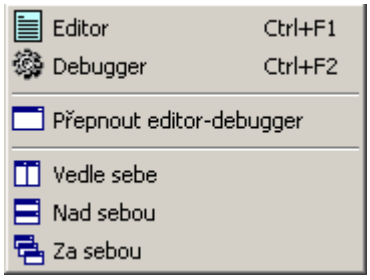

Síťový port, přes který bude aplikace komunikovat je možné nastavit prostřednictvím souboru společného nastavení SETUP_M.INI, který je vždy uložen vedle souboru WinAVR.EXE. Více v kapitole Popis INI. Příklad klíče *Network* síťového nastavení v SETUP_M.INI

```
[Network]
PortTCPIP=0                    ;port TCP-IP 0=auto
EnumTime=500                   ;timeout pro pripojeni relaci
UseGlobalSettings=0            ;pouzit sitove nastaveni prednostne
```

```
NetOutTime=1000              ;out time prenosu
TestTime=5000                ;doba uzivatele po prenosu
SyncTime=1000                ;interval synchronizace
SyncOutTime=5000             ;outtime synchronizace
PackSize=1000                ;velikost paketu dat
```

### 3.1.8 Menu Okno

Slouží k přepínání mezi editorem a debuggerem a k uspořádání oken. Jeho funkce je obvyklá v prostředí Windows. Používá tyto klávesové zkratky:



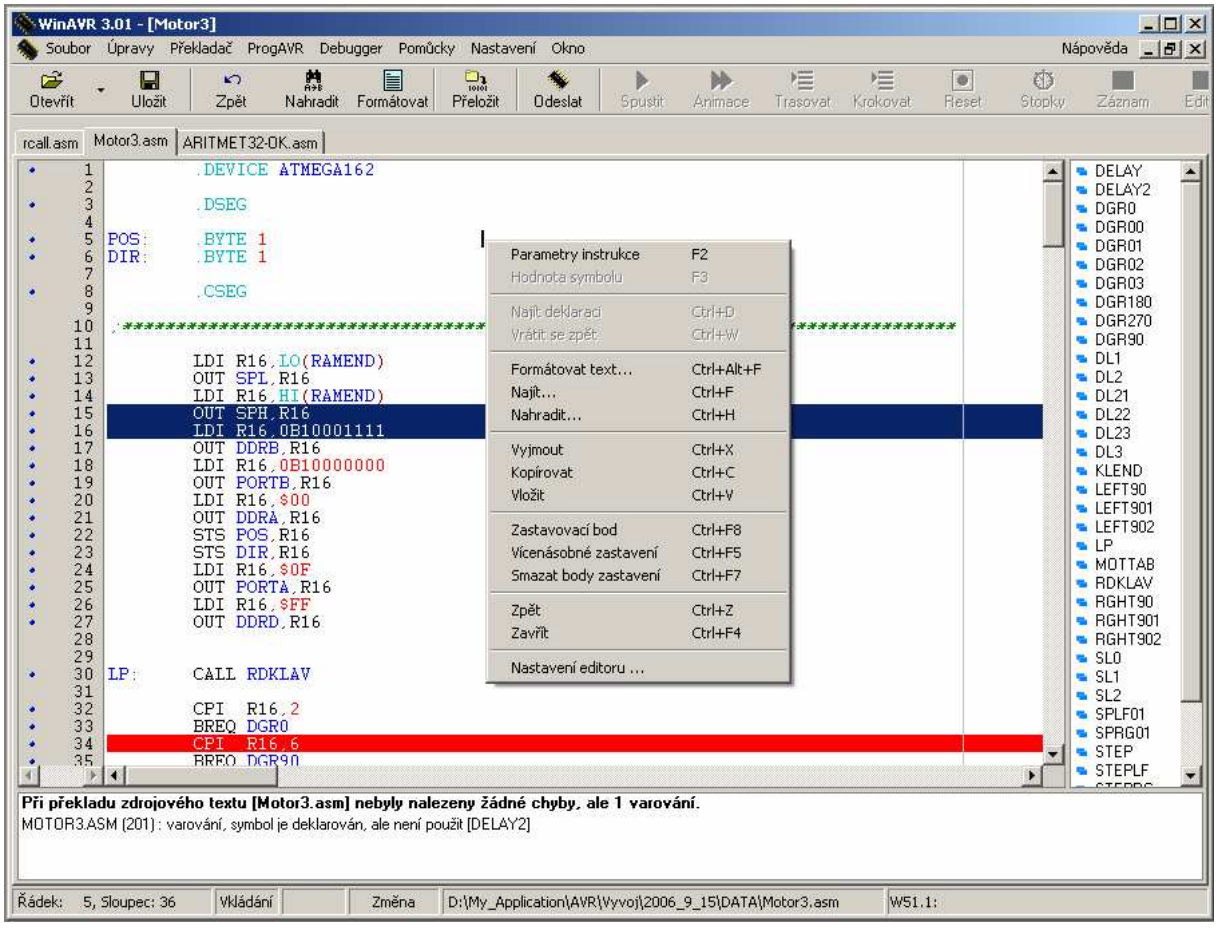

- **Editor, Ctrl+F1:** okno editoru je aktivní.
- **Debugger, Ctrl+F2:** okno debuggeru je aktivní.
- **Přepnout editor-debugger:** dochází k přepínání mezi oběma okny
- **Vedle sebe:** uspořádá okno editoru a debuggeru
- **Nad sebou:** uspořádá okno editoru a debuggeru
- **Za sebou:** uspořádá okno editoru a debuggeru

Pozn. pro rychlé přepínání mezi okny editoru a debuggeru je možné použít klávesovou zkratku **CTRL+TAB**

# 4 Textový editor

## 4.1 Popis editoru

Pro editaci textu se používají příkazy a zkratky obvyklé v systému Windows. Rovněž ukládání, otevírání a tisk souborů je obvyklé jako v ostatních textových editorech. Hlavní možnosti nastavení popisuje okno *Nastavení editoru*.




### 4.1.1 Lišta řádků

Vlevo textový editor obsahuje lištu, která zobrazuje čísla řádků. Můžeme ji vypnout pomocí nabídky *Nastavení* položka *Nastavení editoru*. Po překladu se u každého řádku, který generuje kód, zobrazí modrá tečka (funkční) nebo červená tečka (došlo k chybě při překladu). Kromě toho při chybě nastaví viditelný první řádek, na němž došlo k chybě a je zvýrazněn tmavě červenou barvou. Toto zvýraznění zmizí po zásahu do textu, ale červená tečka zůstává až do dalšího překladu.

### 4.1.2 Pracovní plocha

Nejdůležitější částí editoru je **pracovní plocha**, která obsahuje vodorovný a svislý posuvník a svislou šedou čáru určující šířku stránky. Posuvníky jsou aktivní jen v případě, že zobrazená šířka a výška dokumentu přesahuje vyhrazenou pracovní plochu dokumentu. Při spuštění programu se na pracovní ploše otevřou naposledy zpracovávané soubory, pokud je tato volba zaškrtnuta v menu *Nastavení* v položce *Nastavení editoru*. Editor si pamatuje všechna nastavení, která byla provedena. Na každý otevřený soubor se dostaneme tak že klikneme na jeho záložku. Kliknutím pravým tlačítkem na tuto záložku se vyvolá místní nabídku pro *zavření, nový soubor a uložení pod jménem*.

### 4.1.3 Protokol o překladu

Okno se zobrazí po překladu. Obsahuje jméno souboru a čísla řádků, na kterých došlo k chybě, popis chyby a nápovědu k chybě. Po kliknutí na řádek v tomto okně se dostaneme ihned na daný řádek v okně editoru. Zobrazují se zde jednak samotné chyby, ty jsou navíc v editoru vyznačeny tmavě červeným zvýrazněním řádku, tak i varování, což jsou pomocné informace sloužící pro minimalizování následních běhových chyb jako např. zápis na nepovolenou adresu, použití nedeklarované adresy apod. Generování varovných hlášení je možno vypnout v nabídce *Nastavení-Nastavení překladače*. Při kliknutí pravým tlačítkem na tomto okně vyvoláme místní plovoucí nabídku.

### 4.1.4 Použité symboly

Toto okno se zobrazí také po překladu a obsahuje seznam všech symbolů použitých v programu v abecedním pořadí. Zobrazují se použitá návěští, symbolické deklarace a bitové deklarace. Kliknutím na příslušný symbol se dostaneme na příslušný řádek ve zdrojovém textu, kde je symbol deklarován. Je-li zdrojový text složen z více souborů (použitím .INCLUDE), zobrazují se pouze symboly příslušné k danému souboru. Při kliknutí pravým tlačítkem na tomto okně opět vyvoláme místní plovoucí nabídku.

### 4.1.5 Stavový řádek

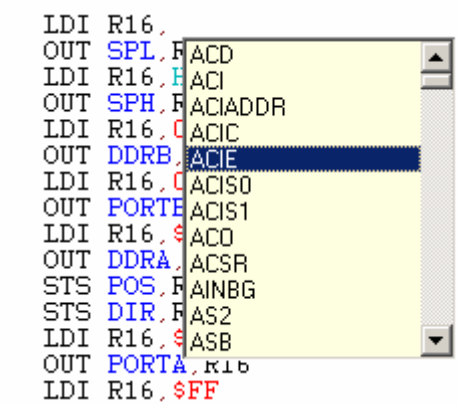
Poslední součástí okna dokumentu je stavový řádek. Je umístěn pod spodním okrajem okna pod vodorovným posuvníkem. Stavový řádek obsahuje číslo řádku a sloupce, na kterém se právě nacházíme, dále informuje, zda-li se nacházíme v přepisovacím nebo vkládacím režimu, máme-li zapnutou klávesu *Caps Lock* a zda-li v dokumentu došlo ke změně. V poslední části stavového řádku je zobrazena místní nápověda.

## 4.2 Plovoucí nabídka

Kliknutím pravým tlačítkem myši na pracovní plochu editoru se vyvolá místní (plovoucí) nabídka. Ta slouží k rychlému vyvolání často používaných příkazů. Jsou to mimo jiné tyto:

### 4.2.1 Parametry instrukce

Textový editor obsahuje zajímavou funkci (intelisense), která se vyvolá stiskem klávesy F2. Pokud napíšeme klíčové slovo asembleru a stiskneme klávesu F2 (nebo místní nabídku, položku *Parametry instrukce*) dojde k vyvolání okna, ve kterém jsou zobrazeny všechny přípustné parametry instrukce, doba trvání a počet bytů. Pomocí dvojstisku myši nebo klávesy Enter přeneseme parametry do editoru. Je také možné toto funkci aktivovat automaticky, tím že v *Nastavení editoru* zadáme parametr *Nápověda [ms]* jiný než nula. Potom se při napsání klíčového slova tato nápověda objeví za specifikovaný počet ms. Pokud během této doby pokračujeme v psaní, okno se neobjeví.




### 4.2.2 Hodnota symbolu

Pomocí této volby můžeme, pokud je kurzor umístěn na uživatelem definovaném symbolu, buď proměnné nebo návěští zjistit číselnou hodnotu tohoto symbolu. Je však nutné, aby zdrojový text neobsahoval chyby, které by znemožnily jeho přeložení.

### 4.2.3 Najít deklaraci

Pomocí místní nabídky můžeme také najít deklaraci symbolického jména, na které ukazuje kurzor.

- **Najít deklaraci, Ctrl+D:** pokud je kurzor umístěn na uživatelem definovaném symbolu, buď proměnné nebo návěští, lze se touto funkcí rychle dostat na místo, kde byl definován. Tzn. do deklarační oblasti programu nebo na začátek podprogramu.
- **Vrátit se zpět, Ctrl+W:** k tomu, aby bylo možno se rychle vrátit na původní místo v programu odkud byla vyvolána funkce Najít deklaraci je určena tato funkce. Vrátí pohled na editovaný text do původního místa.

Je však nutné, aby zdrojový text neobsahoval chyby, které by znemožnily jeho přeložení.

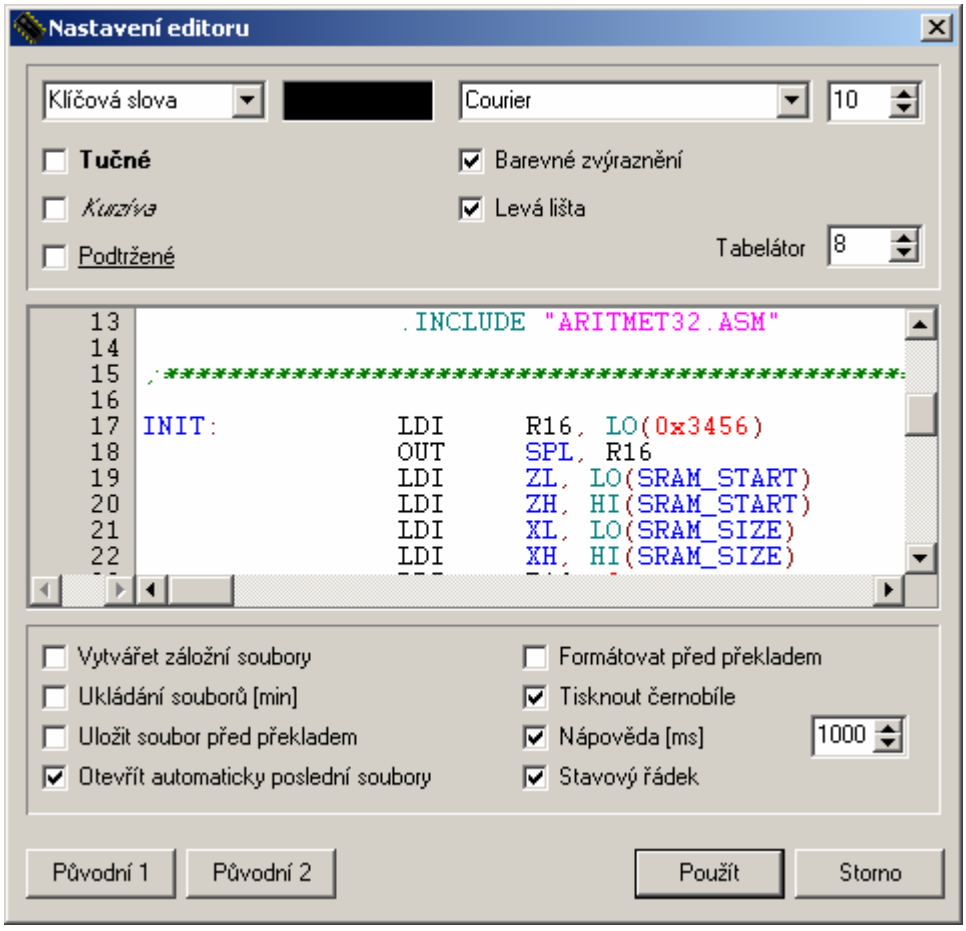
### 4.2.4 Zastavovací body

Pomocí místní nabídky můžeme také vložit zastavovací bod:

- **Zastavovací bod, Ctrl+F8:** slouží k zadání řádku programu, na kterém se má program při průchodu zastavit (tzv. breakpoint). Aktivní breakpoint je zvýrazněn červeně. Breakpoint se smaže tak, že ho opětovně zadáme na tento řádek. Je možno zadat i breakpoint na řádek, kde není platná instrukce, tyto breakpointy se však smažou po překladu zdrojového textu. Nastavení breakpointů se neukládá se souborem, je trvalé pouze po dobu otevření souboru.
- **Vícenásobné zastavení, Ctrl+F5:** vícenásobný breakpoint. K zastavení programu dojde až po určitém počtu průchodů tímto místem. Počet průchodů zadáme v okně, které se objeví po vyvolání této funkce. Je možné zadat počet průchodů v rozsahu 1..65000. Tento breakpoint se smaže pomocí předchozí funkce.
- **Smazat body zastavení, Ctrl+F7:** smaže všechny breakpointy v aktuálním okně editoru, jak jednoduché tak i vícenásobné.

Zastavovací bod lze rychle vložit také tím, že klikneme na levé liště na místo, kde se zobrazují barevné tečky, symbolizující přeložení řádku. Vícenásobný tak, že současně držíme klávesu Shift, smazání jednoho bodu provedeme jeho opětovným označením, všech bodů současně pomocí klávesy Ctrl. Více v kapitole Debugger.

### 4.2.5 Nastavení editoru

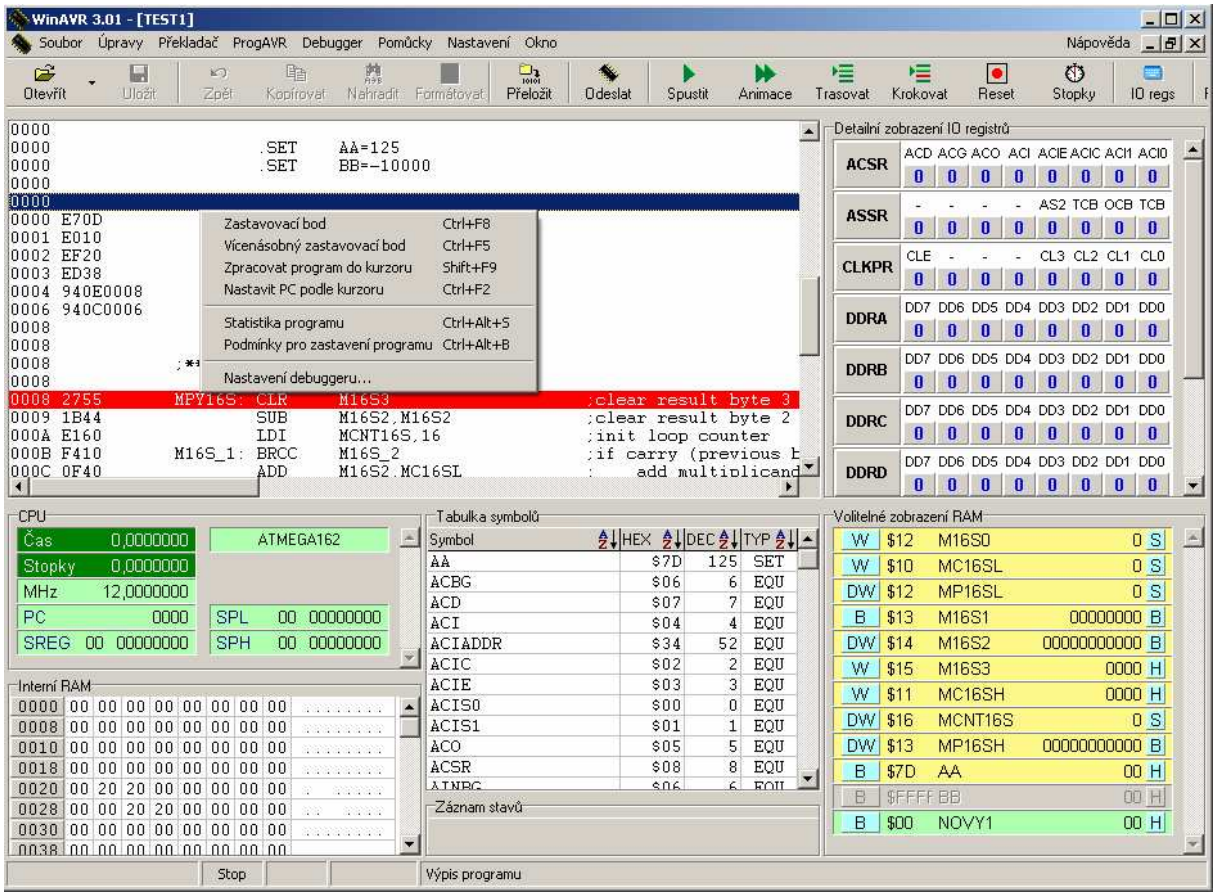


Toto okno umožňuje uživatelské nastavení editoru. Lze nastavit barevné zvýrazňování a jednotlivé barvy u všech syntaktických kategorií, velikost a řez písma. Dále lze můžeme nastavit ukládání záložních souborů s příponou bak. Můžeme zadat, po jaké době se má soubor automaticky ukládat a zda se mají otevírat naposledy zpracovávané soubory. Zda-li se má zobrazovat stavový řádek a lišta s čísly řádků. Je možno nastavit jestli se má tisknout vždy černobíle nebo podle barevného zvýraznění. Tlačítko *Původní 1*, *Původní 2* slouží k nejpoužívanějšímu nastavení. Tlačítko *Použít* aplikuje naše nastavení v editoru. Tlačítko *Storno* naše nastavení nepoužije.

# 5 Debugger

## 5.1 Popis oken debuggeru

Hlavní okno debuggeru se skládá z následujících částí:




### 5.1.1 Výpis po překladu

V levém horním rohu debuggeru se nachází okno, ve kterém se po překladu zobrazí zdrojový text programu, včetně strojového kódu a adres jednotlivých instrukcí. Vyvoláním plovoucí nabídky pravým tlačítkem je možno zadávat zastavovací body (breakpointy) a podmínky pro zastavení programu zápisem na adresu nebo dosažením hodnoty na určité adrese (viz Místa přerušení).

Breakpoint zadaný v tomto okně trvá pouze do dalšího překladu. Překladem se smaže. Trvalý breakpoint je třeba zadat již v editoru. Ten se pak zobrazí i zde, ihned po překladu programu. Breakpoint zadaný v editoru je možno zrušit zase pouze v editoru. Zadat lze buď jednoduchý breakpoint, tzn. program se zastaví vždy, když projde tímto místem, nebo vícenásobný, tzn. k zastavení dojde až po určitém počtu průchodů tímto místem.

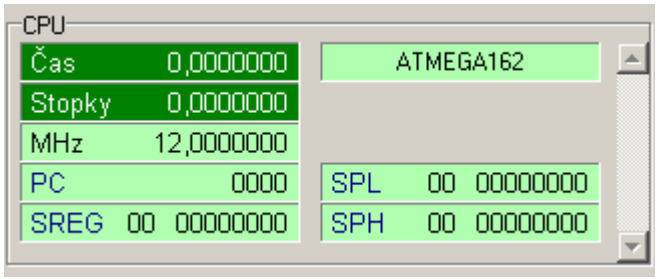
**Dvojím kliknutím** na samotný programový řádek v místě, kde se zobrazuje fyzická adresa v programu se nastaví čítač programu na adresu tohoto řádku, a tak je možné přejít na jiné místo programu, mimo obvyklé pořadí zpracovávání instrukcí. Není možno přejít na řádek, který neobsahuje platnou instrukci, např. komentář apod.

### 5.1.2 Statistika programu

Umožňuje optimalizaci programu, vzhledem k rychlosti a využití systémových zdrojů, přerušení, časovačů, zásobníku apod. Zobrazuje procentuální využití strojového času a rozsah využití zásobníku.

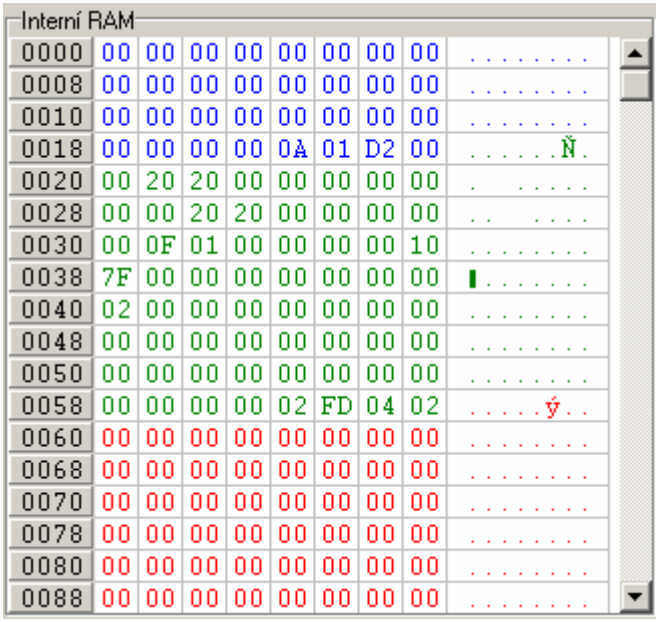
### 5.1.3 Okno stavu CPU

Další je blok s označením **CPU**. Zde je možné sledovat a nastavovat hlavní registry procesoru a volit fyzický kmitočet krystalu. Funkce stopky slouží k měření času mezi dvěma událostmi v procesoru. Tuto funkci je vhodné využívat v kombinaci s breakpointy. Stopky se vynulují buď Nulováním stopek (F3). nebo Resetem programu (F4).



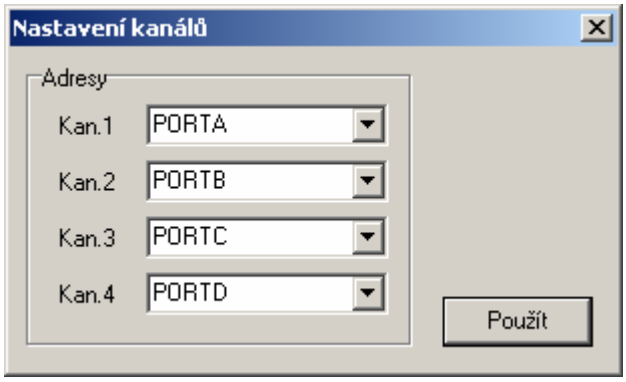

### 5.1.4 Výpis obsahu RAM

Pod tímto oknem se nachází **výpis paměti RAM**. Je zde možné sledovat libovolnou adresu od 00h do MAXRAM podle skutečné velikosti RAM, tedy včetně IO-registrů. Registry jsou označeny modře, IO registry zeleně, rozšířená část IO prostoru červeně a RAM černě.V tomto okně je možné měnit obsahy paměťových míst kliknutím na jednotlivá adresová místa. Editace je zvýrazněna žlutou barvou.



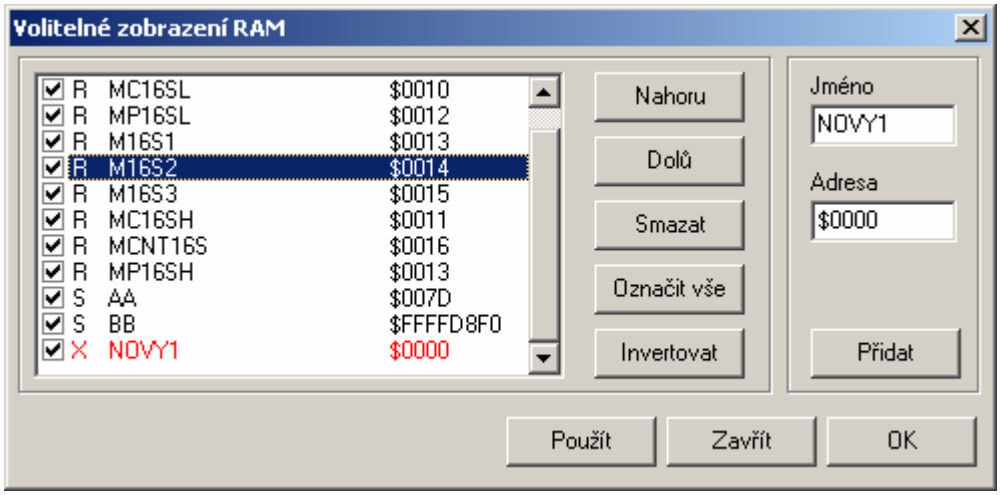

### 5.1.5 Záznam stavů

Okno *Stavy portů* zaznamenává veškeré události na portech nebo v paměti RAM, včetně času, kdy k této události došlo. Tento záznam je možné následně zobrazit pomocí funkce *Záznam stavů*. *Nastavení adresy*, která se má sledovat se provádí v *Záznamu průběhů – Nastavení – Nastavení kanálů*, standardně jsou nastaveny porty PA..PD

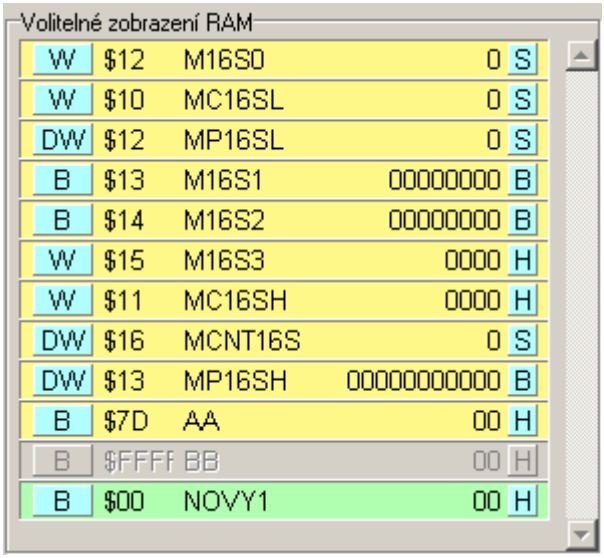



### 5.1.6 Volitelné zobrazení RAM

Okno *Volitelné zobrazení RAM* slouží ke sledování a změně obsahu libovolného paměťového místa uživatelské RAM. Chceme-li přidat nebo ubrat nějakou adresu, vyvoláme v menu *Debugger* položku *Volitelné zobrazení*. Lze také kliknout na plochu tohoto okna nebo na jméno některé již zobrazované proměnné. V okně, které se tím vyvolá, se objeví seznam všech symbolů definovaných pomocí direktivy .EQU, .SET nebo .DEF. Zaškrtnutím příslušného symbolu se hodnota tohoto symbolu bude zobrazovat při zpracovávání programu. Chceme-li sledovat adresu, která není pojmenována pomocí EQU, zvolíme její název a stiskneme tlačítko Přidat. Adresa může být zadána jak desítkově (45) tak hexadecimálně ($2D). Lze sledovat pouze adresy v celém rozsahu RAM, tzn. 0..MAXRAM. Adresy je možno sledovat jako BYTE, WORD nebo DWORD. Zadaná adresa je chápána jako první (nejnižší). Je možné zvolit zobrazení v binárním (B), hexadecimálním (H), desítkovém (D) nebo znaménkovém (S) tvaru. Dále je možné pomocí tlačítek *Nahoru* a *Dolů* volit pořadí zobrazovaných symbolů.

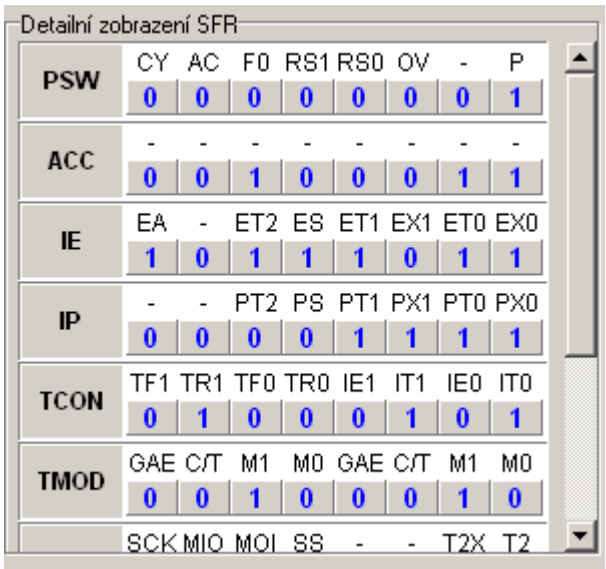



Zde se zobrazují všechny použité, uživatelem definované symboly a jejich hodnoty. Symboly, které jso nadefinovány, ale nejsou použity jsou zvýrazněny zelenou barvou pozadí. Kliknutím na horní záhlaví je možno setřídit zobrazení symbolů do abecedního pořadí, podle libovolného sloupce.



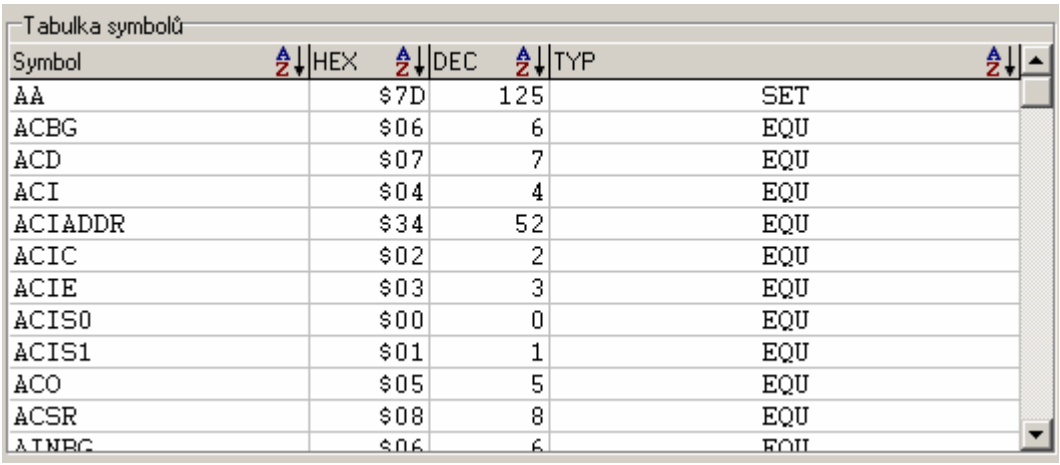

Symboly definované pomocí EQU jsou znázorněny žlutou barvou. Adresy přímo zadané mají barvu šedou. Každá adresa může být zobrazena pouze jednou. Není tedy možné zobrazit současně symbol na určité adrese a tuto adresu přímo. Vzhledem k přehlednosti, však nelze přímé zadávání adres vůbec doporučit a to také z následujícího důvodu: Pokud se během ladění programu rozhodneme pro určitý symbol použít jinou adresu, změnou definice EQU, není třeba v tomto okně nic měnit. Adresy symbolů se aktualizují po každém překladu, takže se zobrazí vždy správná adresa. Toto ale s přímo zadanou adresou nemůže fungovat.

### 5.1.7 Detailní zobrazení IO



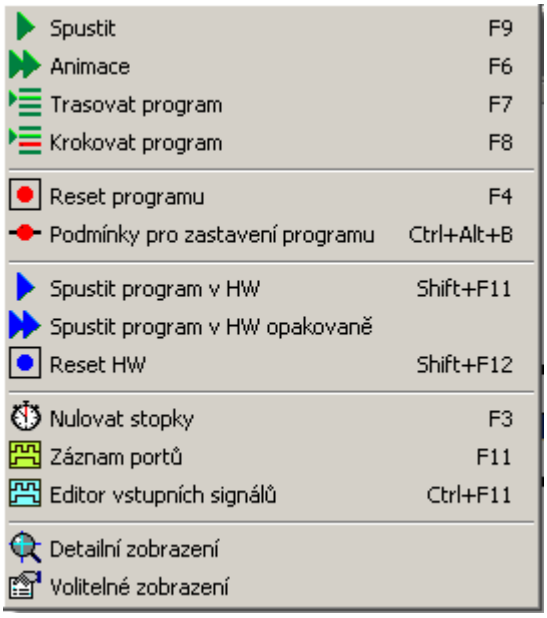

*Detailní zobrazení IO* umožňuje snadno pracovat po bitech s jednotlivými funkčními registry. IO, které chceme zobrazovat, volíme v menu *Debugger-Volitelné zobrazení*. Lze také kliknout na jméno některého již zobrazovaného IO nebo na pozadí tohoto okna. Kliknutím na příslušné tlačítko, lze snadno změnit hodnotu bitu. Nad tlačítkem je zobrazeno symbolické pojmenování každého bitu. Při pohybu kurzoru nad jménem IO se zobrazuje ve stavovém řádku název a fyzická adresa daného IO registru.

### 5.1.8 Tabulka symbolů




| Symbol | HEX | DEC | TYP |
|---|---|---|---|
| AA | $7D | 125 | SET |
| ACBG | $06 | 6 | EQU |
| ACD | $07 | 7 | EQU |
| ACI | $04 | 4 | EQU |
| ACIADDR | $34 | 52 | EQU |
| ACIC | $02 | 2 | EQU |
| ACIE | $03 | 3 | EQU |
| ACIS0 | $00 | 0 | EQU |
| ACIS1 | $01 | 1 | EQU |
| ACO | $05 | 5 | EQU |
| ACSR | $08 | 8 | EQU |
| AINBG | $06 | 6 | EQU |

## 5.2 Popis funkcí debuggeru




### 5.2.1 Funkce Spustit

(klávesa F9) Tato funkce spouští rychlé vykonávání programu. Při běhu programu nejsou aktualizovány změny obsahů registrů, IO, portů apod. Zobrazuje se pouze strojový čas procesoru a stopky. Na rychlejších počítačích je možno dosáhnout reálné rychlosti simulovaného procesoru. Všechny změny obsahů se zobrazí až v okamžiku, kdy je vykonávání programu zastaveno nebo program narazí na některé místo přerušení nebo dojde k běhové chybě při vykonávání programu. Vykonávaný program je možno zastavit opětovným vyvoláním funkce Spustit nebo funkcí Reset. Všechny změny na portech se zaznamenávají a je možno si je následně zobrazit pomocí funkce Záznam portů.

### 5.2.2 Funkce Animace programu

Tato funkce (klávesa F6) vykonává pomalé provádění programu po instrukci, všechna zobrazovaná paměťové místa jsou však po vykonání instrukce aktualizována. Rychlost animace je možné volit v nabídce *Nastavení Debuggeru*. Jinak je funkce stejná jako *Spustit*, nelze však dosáhnout maximální rychlosti provádění programu.

### 5.2.3 Funkce Krokovat program

Vykoná jeden řádek programu (klávesa F8). Pokud je tomto řádku volání podprogramu (instrukce CALL, RCALL, ICALL) nebo volání makra, provede se celý podprogram popř. makro. Je-li vykonávání podprogramu nebo makra příliš dlouhé, můžeme jeho vykonávání dočasně přerušit (dostaneme se do místa, podle aktuálního PC) a opětovným stiskem klávesy F8 pokračovat jeho provádění.

### 5.2.4 Funkce Trasovat program

Vykoná pouze jednu instrukci programu (klávesa F7). Všechna zobrazovaná paměťové místa jsou po vykonání instrukce ihned aktualizována. Po vykonání instrukce je provádění programu zastaveno. Pokud je následující instrukce skryta direktivou NOLIST, např. tělo makra nebo podprogramu, provádění programu pokračuje do té doby, než narazí na první viditelnou instrukci.

### 5.2.5 Funkce Reset programu

Nastavuje (klávesa F4) čítač programu na adresu $0000. Obnovuje výchozí hodnoty ve IO registrech. Nuluje strojový čas a čas stopek. Také maže záznam stavu portů. Obsah ostatní paměti RAM se nemění. Dále se aktualizují vícenásobné breakpointy.

### 5.2.6 Funkce Spustit program v HW

Spustí program v emulátoru a čeká na dosažení breakpointu, po jeho dosažení přenese a zobrazí obsah paměti RAM z procesoru do debuggeru a čeká.

### 5.2.7 Funkce Spustit program v HW opakovaně

Spustí program v emulátoru a čeká na dosažení breakpointu, po jeho dosažení přenese a zobrazí obsah paměti RAM z procesoru do debuggeru a znovu spustí program na následující instrukci.

### 5.2.8 Reset HW

Zastaví provádění programu v HW tím, že aktivuje signál RESET. Pokud je program již zastaven (při opakovaném zavolání této funkce) provede se také reset debuggeru.

### 5.2.9 Funkce Nulovat stopky

Vynuluje čas změřený na stopkách. Stopky jsou určeny pro měření časových úseků programu např. mezi dvěma breakpointy.

# 5.3 Sledování programu v HW

### 5.3.1 Princip HW breakpointu

IDE WinAVR umožňuje sledovat funkci programu přímo v procesoru. Slouží k tomu systém tzv. hardwarových breakpointů. Jejich princip spočívá v tom, že do uživatelského prováděného programu se přidá krátký podprogram. Pokud se na něj při provádění programu narazí, je provádění programu dočasně zastaveno, do WinAVR je odeslán obsah celé paměti RAM procesoru, a ten je zobrazen v debuggeru. Pro pokračování programu je třeba znovu zvolit funkci *Spustit program v HW*. Nebo můžeme zvolit funkci *Spustit program v HW opakovaně*, která po přenesení a zobrazení dat automaticky program opět spustí. Sledování stavu spuštěného programu je možné buď v emulátoru, nebo i pomocí programátoru, připojeného do zařízení pomoci rozhraní ISP. Je možné použít jakýkoliv procesor, který disponuje dostatečně velkou pamětí programu i dat a který lze připojit přes ISP. Prakticky to znamená, že kromě nejmenších typů Tiny11, 12, 15 apod. lze debugging využít téměř u všech typů. U připojeného procesoru musí být volné vývody pro ISP. Pro úspěšné použití tohoto nástroje je třeba dodržet několik zásad:

- musí být definován typ procesoru
- musí být vložena jednotka DBG.INC
- musí být posunut zásobník
- na místě, kde má dojít k zastavení použijeme makro BREAKPOINT

dále je vhodné na začátku paměti nulovat obsah registrů a RAM, protože data z procesoru se přenášejí komprimovaná a nulování paměti může zásadním způsobem urychlit přenos do WinAVR. Přidané makro BREAKPOINT zabere asi 120 bytů paměti a vyžaduje asi 6 bytů na zásobníku v RAM. Je možné bez omezení používat všechny registry i IO registry kromě systému ISP.

Příklad jednoduchého programu při použití HW breakpointu

```
      .DEVICE     ATMEGA162   ;musí být definován typ procesoru

      RJMP  INIT              ;přeskočit vložené jednotky

      .INCLUDE "DBG.INC"      ;jednotka pro HW debugging

INIT: LDI   R16, LO(RAMEND)   ;musí být definován zásobník
      OUT   SPL, R16
      LDI   R16, HI(RAMEND)
      OUT   SPH, R16

MAIN: INC   R0                ;hlavní programová smyčka
      BREAKPOINT              ;vlastní HW breakpoint
      RJMP  MAIN
```

### 5.3.2 Postup při ladění programu

Program se normálně přeloží a odešle do HW. Aby nedošlo k jeho okamžitému spuštění, můžeme v menu *Nastavení přenosu dat* zvolit *Po odeslání nespouštět nikdy*. Tímto se program po odeslání nespustí. Zvolíme funkci *Spustit program v HW*. Program je spuštěn, a provádí se tak dlouho, dokud nenarazí na podprogram BREAKPOINT. Tehdy je jeho činnost zastavena. WinAVR se periodicky dotazuje (perioda 0,3 s), zda nebyl tento stav dosažen (Je možné toto sledovat na panelu CPU, kde se objeví ukazatel znázorňující provádění programu. Jedno posunutí ukazatele znamená jeden dotaz na stav programu. Pokud nedochází ke komunikaci s HW, je debugging přerušen a zobrazí se zpráva „HW neodpovídá“. Je-li možné procesor programovat, může být problém ve starší verzi SW v programátoru, která nepodporuje debugging). Pokud je breakpointu dosaženo, obsah procesoru je přenesen do debuggeru a zobrazen. Program stojí a čeká na další spuštění nebo pokračuje dále automaticky, jestliže byl spuštěn funkcí *Spustit program v HW opakovaně*. Zvolením funkce *Reset HW* se procesor resetuje a program se zastaví.

## 5.4 Chyby při spuštění programu

Při spuštění programu mohou nastat chyby, které nelze odhalit během překladu. Nejčastěji to je přístup do zakázané oblasti RAM. V tomto případě je program přerušen a vypíše se některé z těchto chybových hlášení:

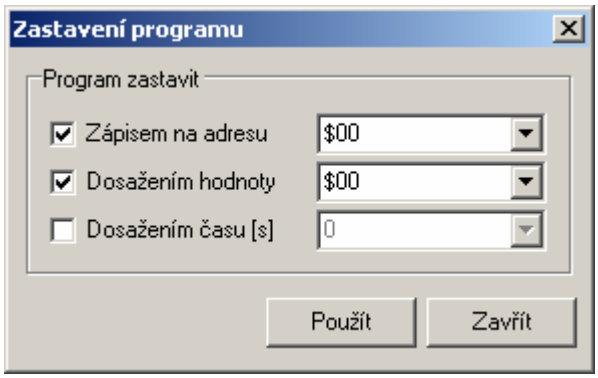
- **Na $xxxx nebyla nalezena platná instrukce** - k tomuto stavu může dojít např. při skoku na vypočítanou adresu IJMP nebo při neošetřeném přerušovacím vektoru.
- **Neznámá instrukce na : $xxxx -** může nastat za stejných podmínek jako předchozí chyba.
- **Zápis do nepřístupné adresy $xxxx v oblasti RAM** - chyba v adresování. Např. při
- **Čtení z nepřístupné adresy $xxxx v oblasti RAM** - stejné jako předchozí chyba.
- **Zápis do FLASH, nepřístupná adresa $xxxx** - stejné jako předchozí chyba.
- **Čtení z FLASH, nepřístupná adresa $xxxx –** adresa mimo rozsah.
- **Obsah adresy $xxxx v RAM není definován**
- **Hodnota bitu na adrese $xxxx není definována**

Pomocí tlačítka Ignorovat potlačíme zobrazování daného typu chyby až do příštího překladu programu. V nabídce Nastavení debuggeru je možno chybová hlášení zcela vypnout.

Dále může být program korektně zastaven splněním zastavovací podmínky, v tomto případě se vypíše toto hlášení: **Instrukce na adrese $xxxx splnila zastavovací podmínku**. Symbol $xxxx je konkrétní adresa instrukce, na níž došlo k chybě. Ukazatel prováděné instrukce (modrý řádek) je v tomto okamžiku již na následující instrukci.

# 5.5 Místa přerušení.

Zastavení průchodem daným místem. Vyvoláním plovoucí nabídky pravým tlačítkem je možno zadávat zastavovací body (breakpointy) a podmínky pro zastavení programu zápisem na adresu nebo dosažením hodnoty na určité adrese.




Breakpoint zadaný v tomto okně trvá pouze do dalšího překladu, potom se smaže. Trvalý breakpoint je třeba zadat již v editoru. Ten se pak zobrazí se i zde ihned po překladu programu. Breakpoint zadaný v editoru je možno zrušit zase pouze v editoru. Zadat lze buď jednoduchý breakpoint, tzn. program se zastaví vždy, když projde tímto místem, nebo vícenásobný, tzn. k zastavení dojde až po určitém počtu průchodů tímto místem.

### 5.5.1 Zastavení zápisem na adresu

Program je dále možno zastavit zápisem na určitou adresu. Tento přerušovací bod zadáme v menu *Debugger-Podmínky pro zastavení programu*. Zvolíme *Zastavit zápisem na adresu* a zadáme požadovanou adresu. Kdykoliv je na danou adresu proveden zápis (i bitově) vykonávání programu je zastaveno.

### 5.5.2 Zastavení dosažením hodnoty

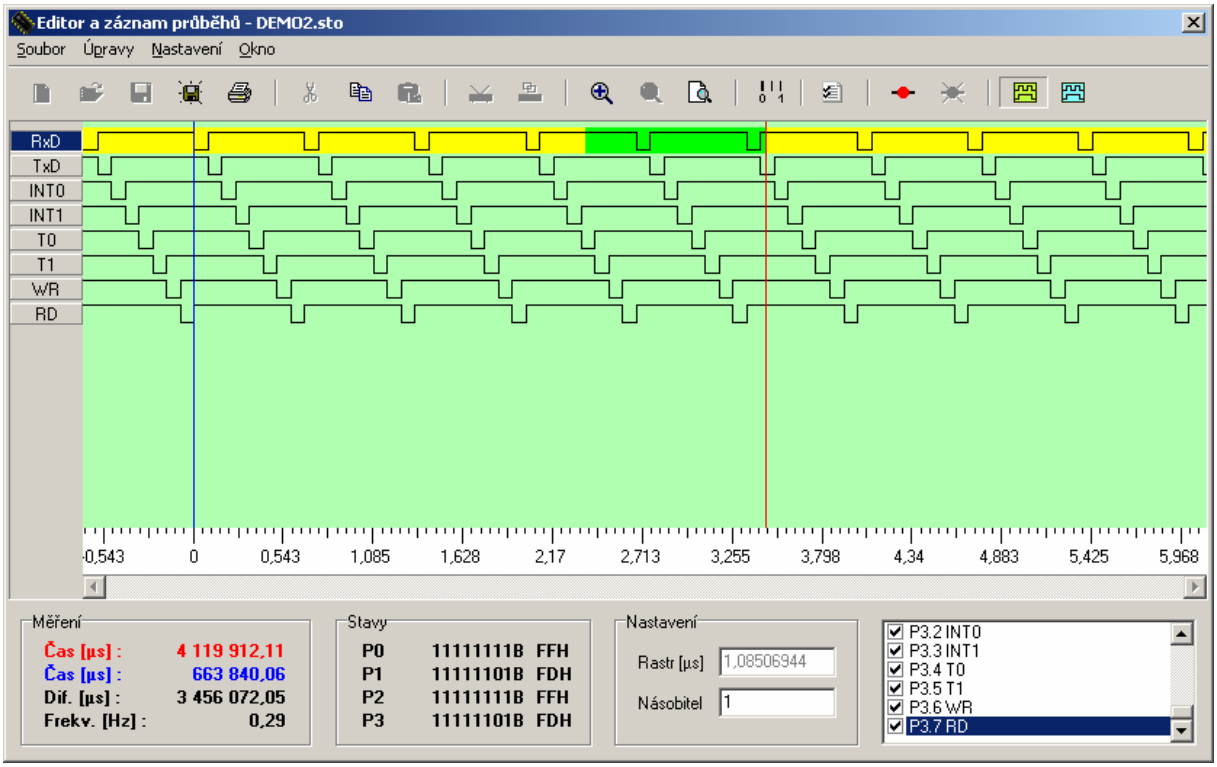
Program se zastaví až dosažením konkrétní hodnoty na požadované adrese. V tomto případě zvolíme *Zastavit dosažením hodnoty*. Obě tyto možnosti se velice dobře uplatní při hledání chyby v programu, kdy přesně nevíme, která instrukce danou adresu modifikovala.

# 5.6 Editor a záznam průběhů

Casový záznam na portech nebo libovolného paměťového místa a editování vstupních průběhů umožňuje vestavěný Editor a záznamník průběhů. Chová se jako 32-kanálový logický analyzátor. Umožňuje sledovat celkem čtyři volitelné adresy nebo SFR. Standardně se používá pro sledování průběhů na portech P0-P3. Editor průběhů umožňuje v definovaných časech přivádět na porty

procesoru logické úrovně a tak simulovat spolupráci procesoru s okolními periferiemi. Toto načítání musí být povoleno v menu Nastavení debuggeru.

### 5.6.1 Záznam průběhů

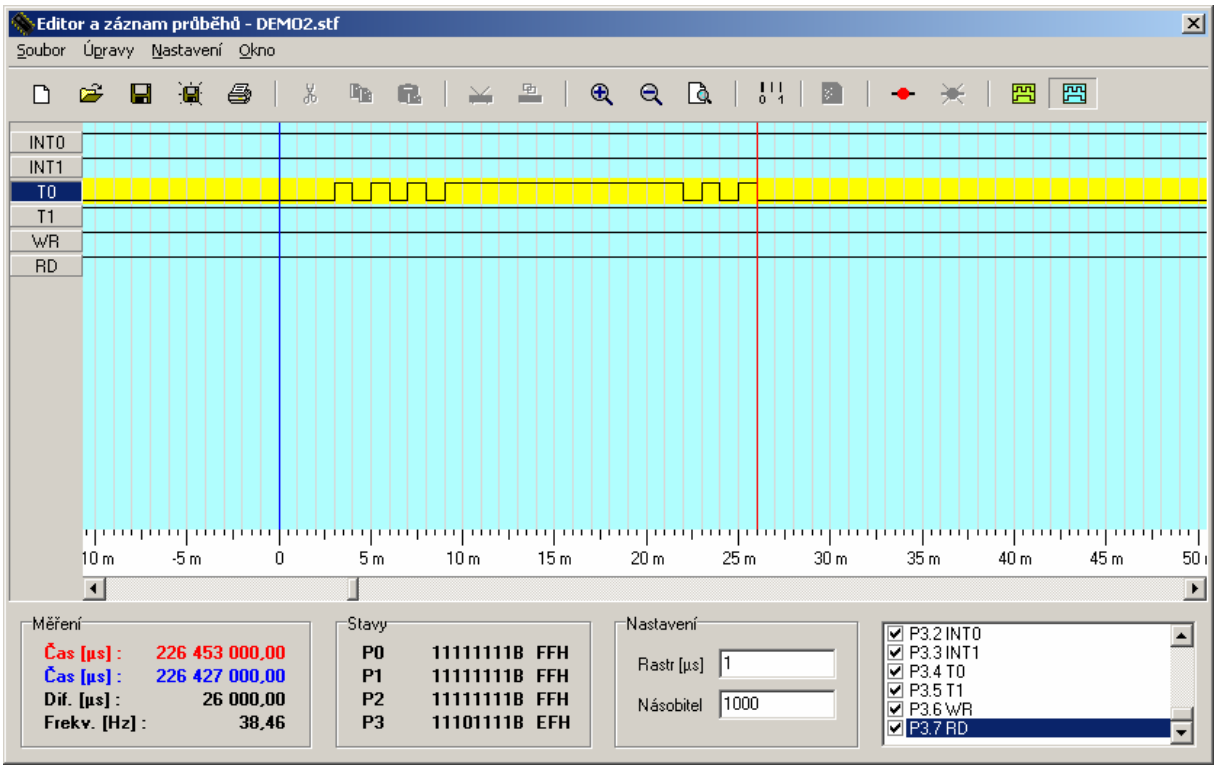



Zobrazuje průběhy na zvolených bitech během vykonávání programu. Je možné zobrazit 16 bitů současně. Pokud jsou bity symbolicky pojmenovány pomocí direktivy BIT, obrazuje se přímo toto pojmenování. Adresy, které chceme sledovat zvolíme v nabídce *Nastavení kanálů*. Bity, které chceme sledovat, volíme v zaškrtávacím poli tohoto okna. Tato volba se automaticky ukládá.

Velikost pohledu na záznam je možné libovolně měnit pomocí tlačítek se symbolem lupy (nebo klávesami Q,W,E) až po jednotlivé fáze strojního cyklu. Okno obsahuje tři kurzory. Hlavní odměřovací kurzor červené barvy se ovládá levým tlačítkem myši. Pomocný kurzor modré barvy se ovládá pravým tlačítkem nebo klávesou mezera. Kliknutím mezerníkem se modrý kurzor položí pod červený (vznikne relativní nula) a pohybem červeného odměřujeme čas. Rozdíl časů hlavního a pomocného kurzoru je automaticky převáděn na kmitočet. Třetí kurzor znázorňuje polohu časového breakpointu. Je zobrazen tehdy, je-li funkce zastavení aktivní (čas je jiný než nula). Jeho polohu je možné zadat buď tlačítkem na panelu nástrojů nebo přímo v menu *Podmínky pro zastavení programu*. Také je možné uložit si místo které sledujeme a po provedení změn a novém spuštění programu se na toto místo rychle vrátit. Je pouze nutné aby byl program znovu spuštěn nejméně po dobu než projde sledovaným místem. Tato funkce se ovládá tlačítky *Uložit pohled* a *Načíst pohled*.

Je také možné část průběhu vložit do schránky a použít jako vstupní průběh *v Editoru průběhů*. V tomto případě dojde k automatickému přepočítání časových jednotek, tak aby průběh byl časově správný i v editoru. K tomu je třeba kliknutím na tlačítko na levé liště zaktivovat jeden příslušný bit a pomocí hlavního kurzoru označit blok, který potom zkopírujeme do schránky.

### 5.6.2 Editor vstupních průběhů




Slouží k simulaci vstupních stavů přiváděných na porty procesoru. Umožňuje definovat vstupní signály pro vnější přerušení, sériový kanál, nebo simulovat stisk tlačítek připojených k některému bitu portu apod. Při spuštění programu nebo krokování se nadefinované signály kopírují na porty procesoru a tím napodobují reálné prostředí ve kterém procesor bude pracovat. Reakce programu na vstupní signály a výstupní signály se po proběhnutí potřebné doby zobrazují v okně ZÁZNAM PORTŮ. Je možné využívat také techniku zastavovacích bodů.

Editovat je možné kterýkoliv V-V bit procesoru, ale současně lze zobrazit pouze 16 bitů. Pokud jsou bity symbolicky pojmenovány pomocí direktivy BIT, zobrazuje se přímo toto pojmenování. Bity, které chceme editovat, volíme v zaškrtávacím poli tohoto okna. Tato volba se automaticky ukládá. Časový rozsah editovaných signálů závisí na zvoleném časovém rastru. Implicitně je zvolen časový krok 1us. Při tomto rastru je možné editovat signály v max. době 1000 s. Pokud zvolíme jiný rastr, bude max. doba odpovídající tomuto nastavení (časová osa může mít max. 1 000 000 000 změn, z toho plyne doba 1000 s při rastru 1us).

Velikost pohledu je možné libovolně měnit pomocí tlačítek se symbolem lupy (nebo klávesami Q,W,E) až po maximální rozlišení odpovídající zvolenému rastru (např. 1us). Pro snadné zadávání průběhů je možné zvolit tzv. násobitel rastru. To je časový násobek rastru, a takto se pohybuje kurzor při stisku kurzorové šipky. Po jednotlivých bodech rastru se bude kurzor pohybovat, pokud současně držíme klávesu Shift nebo naopak, při Ctrl se bude pohybovat rychleji. Pro zvolený násobek se zobrazují pomocné šedé svislé čáry, jsou kresleny tak, že vždy prochází nulou, tzn. místem, v němž leží modrý pomocný kurzor. Pro zobrazení čar je třeba mít vhodnou velikost pohledu. Velikost pohled a poloha kurzorů se automaticky ukládá při uzavření tohoto okna. Hlavní odměřovací kurzor červené barvy se ovládá levým tlačítkem myši nebo kurzorovými klávesami. Pomocný kurzor modré barvy se ovládá pravým tlačítkem nebo klávesou mezera.

Kliknutím pravým tlačítkem nebo mezerníkem se modrý kurzor položí pod červený (vznikne relativní nula) a pohybem červeného odměřujeme čas. Rozdíl časů hlavního a pomocného kurzoru je automaticky převáděn na kmitočet.

Vlastní editování průběhů začneme tím, že v zaškrtávacím poli zvolíme, které bity mají být zobrazeny. Nastavíme vhodnou velikost rastru a násobitele. Nastavíme velikost pohledu tak, aby se objevily pomocné svislé čáry. Potom vybereme kliknutím na levém panelu bit, který chceme editovat. Posuneme kurzor na požadovaný začátek průběhu a položíme tam relativní nulu (modrý kurzor). Nyní stiskem kurzorových šipek nahoru a dolů dochází ke kreslení průběhu. Kurzor se posune o vždy o jeden násobek rastru doprava.

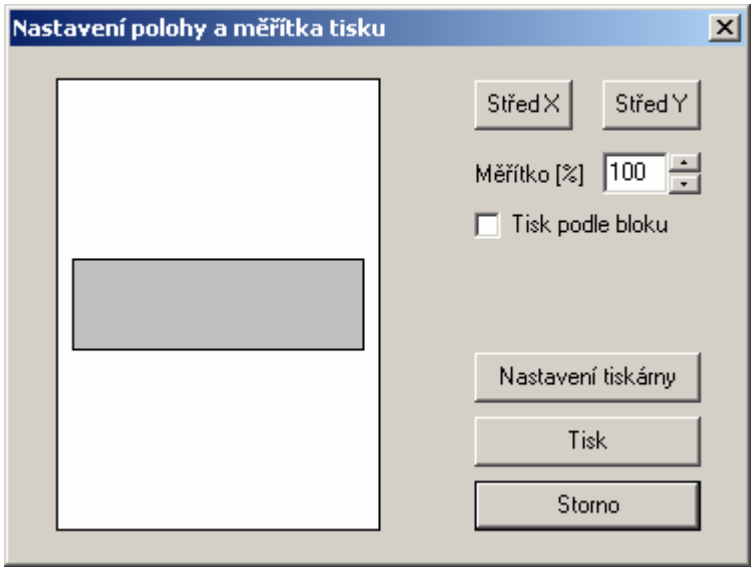
Tlačítko *Kopírovat blok* slouží k snadnému nakreslení opakujících se průběhů. Modrý kurzor označuje začátek kopírovaného signálu, červený konec. Vezme se ta část průběhu, která je uzavřena mezi modrý a červený kurzor a zkopíruje se za červený kurzor a ten se posune o příslušný časový úsek doprava. Chceme-li např. simulovat přiváděný obdélníkový signál, nakreslíme pouze jednu periodu signálu a stiskneme *Kopírovat blok*, vytvoří se druhá perioda. Dalším stisknutím vzniknou 4 periody, dále 8, 16, 32 ... atd. Velice rychle tak lze vytvořit signál i o tisících periodách. Lze tak snadno i smazat celý bit. Kopíruje se pouze bit, který je momentálně aktivní. Pro vymazání určitého úseku slouží tlačítko *Vymazat blok*. Uzavřením tohoto okna se průběh uloží a můžeme začít testovat program. Další tlačítka V*ložit, Vyjmout, Zkopírovat* pracují se schránkou podle zvyklostí Windows.

**Seznam zkratkových kláves:**

| | |
|---|---|
| INSERT | vloží jeden časový násobek rastru na místo za červeným kurzorem |
| DELETE | smaže jeden časový násobek rastru na místo za červeným kurzorem |
| HOME | skok na čas nula |
| END | skok na konec průběhu |
| PG UP | přepne aktivní bit směrem nahoru |
| PG DOWN | přepne aktivní bit směrem dolů |
| Q,W,E | ovládání zoomu |
| MEZERA | položí modrý kurzor, relativní nula |
| ŠIPKY | posun kurzoru, kreslení průběhu |
| SHIFT | jemný posun kurzoru |
| CTRL | rychlý posun kurzoru |

### 5.6.3 Tisk průběhů

Pro snadný tisk editovaných i zaznanenaných průběhů slouží nabídka *Tisk*. Tiskne se ta část průběhu, která je viditelná nebo ta, která je označena blokem. Pomocí myši lze snadno nastavit polohu tisku. Tisknou se vždy ty bity, které jsou zobrazeny. Tlačítky *StředX* a *StředY* je možné rychle vyrovnat tisknutý průběh na středy papíru. Dále možné zvolit měřítko tisku.




# 5.7 Simulace vnější paměti EEPROM

## 5.7.1 Pamět EEPROM

Některé procesory mají integrovánu vnitřní paměť EEPROM. Debugger umožňuje její využití. Je-li povolen zápis do této paměti nastavením příslušných bitů, je možné s touto pamětí pracovat. Pokud program využívá práci s EEPROM je její obsah automaticky zálohován do souboru stejného jména s příponou DAT. Tento soubor se při otevření zdrojového textu načítá a aktualizuje se tak její obsah. Toto načítání musí být povoleno v menu *Nastavení debuggeru*.

# 6 Kompilátor

## 6.1 Úvod

Kompilátor jazyka symbolických adres mikroprocesorů řady AVR, který je obsažen v integrovaném prostředí WinAVR je založen na výkonné 32bitové technologii. Jedná se o jednoprůchodový kompilátor s dopřednou analýzou zdrojového textu. Symboly které nelze vyjádřit ihned, jsou zpětně doplněny po průchodu celého zdrojového textu.

Kompilátor provádí striktní kontrolu rozsahů výsledků matematických a logických operací, tak, aby již ve fázi překladu bylo eliminováno maximum možných logických chyb způsobených programátorem. Generuje jak chybová hlášení, tak pomocné informace, varování, které nejsou sice syntaktickou chybou, ale mohou pomoci při odlaďování zdrojového textu.

Vstupem pro kompilaci je soubor *.ASM otevřený v integrovaném editoru prostředí WinAVR, výstupem pak protokol o překladu *.LST a výsledné (cílové) soubory ve formátu Intel-HEX *.HEX a absolutní binární soubor *.BIN. Generování těchto souboru je možné vypnout v nabídce Nastavení překladače. Zde je rovněž možné vypnout generování varovných hlášení a zvolit zda kompilátor bude rozlišovat velikost znaků (case-sensitive).

## 6.2 Popis syntaxe jazyka

Syntaxe jazyka je téměř shodná se syntaxí akceptovanou kompilátorem AVR, tak aby bylo možné přímo využívat již vytvořené programy a neodchyluje se od obecně rozšířené syntaxe definované firmou Intel pro mikroprocesory řady AVR.

## 6.3 Rezervovaná jména

Rezervovaná symbolická jména (symboly) mají v syntaxi jazyka vyhrazený konkrétní význam a programátor je může použít pouze předem určeným způsobem. Není přípustné jejich použití jako návěští. Rovněž tak redeklarace hodnot rezervovaných jmen není přípustná. V obou případech kompilátor ohlásí chybu.

### 6.3.1 Seznam rezervovaných jmen a jejich hodnot

| Symbol | Hodnota | Symbol | Hodnota | Symbol | Hodnota | Symbol | Hodnota |
|---|---|---|---|---|---|---|---|
| R0 | $0000 | R8 | $0009 | R16 | $0011 | R24 | $0019 |
| R1 | $0001 | R9 | $000A | R17 | $0012 | R25 | $001A |
| R2 | $0002 | R10 | $000B | R18 | $0013 | R26 | $001B |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| R3 | $0003 | R11 | $000C | R19 | $0014 | R27 | $001C |
| R4 | $0004 | R12 | $000D | R20 | $0015 | R28 | $001D |
| R5 | $0005 | R13 | $000E | R21 | $0016 | R29 | $001E |
| R6 | $0006 | R14 | $000F | R22 | $0017 | R30 | $001F |
| R7 | $0007 | R15 | $0010 | R23 | $0018 | R31 | $0019 |

### 6.3.2 Seznam rezervovaných symbolů

| | | | | | |
|---|---|---|---|---|---|
| .ORG | ADIW | CBI | INC | ROL | X |
| .BYTE | ANDI | CBR | ISREG | ROR | Y |
| .CSEG | ASR | CLC | JMP | SBC | Z |
| .DB | BCLR | CLH | LD | SBCI | PRT |
| .DEF | BLD | CLI | LDD | SBI | DDR |
| .DEVICE | BRBC | CLN | LDI | SBIC | PIN |
| .DSEG | BRBS | CLR | LDS | SBIS | REG |
| .DW | BRCC | CLS | LO | SBIW | .UNIT |
| .ELSE | BRCS | CLT | LOG2 | SBR | .ENDUNIT |
| .ENDIF | BREAK | CLV | LOW | SBRC | .EXPORT |
| .ENDMACRO | BREQ | CLZ | LPM | SBRS | .MAIN |
| .ENDSECT | BRGE | COM | LSL | SEC | |
| .EQU | BRHC | CP | LSR | SEH | |
| .ESEG | BRHS | CPC | LWRD | SEI | |
| .EXFUSE | BRID | CPI | MOV | SEN | |
| .EXIT | BRIE | CPSE | MOVW | SER | |
| .HIFUSE | BRLO | DEC | MUL | SES | |
| .IBIT | BRLT | EICALL | MULS | SET | |
| .IF | BRMI | EIJMP | MULSU | SEV | |
| .IFDEF | BRNE | ELPM | NEG | SEZ | |
| .INCLUDE | BRPL | ELPM | NOP | SLEEP | |
| .LIST | BRSH | ELPM | OR | SPM | |
| .LOCK | BRTC | EOR | ORI | ST | |
| .LOFUSE | BRTS | EXP2 | OUT | STD | |
| .MACRO | BRVC | FMUL | PAGE | STS | |
| .NOLIST | BRVS | FMULS | POP | SUB | |
| .RBIT | BSET | FMULSU | PUSH | SUBI | |
| .RESW | BST | HI | R0 - R31 | SWAP | |

| | | | | | |
|---|---|---|---|---|---|
| .SECT | BYTE1 | HIGH | RCALL | SYMDEF | |
| .SET | BYTE2 | HWRD | RET | SYMVAL | |
| ADC | BYTE3 | IJMP | RETI | TST | |
| ADD | BYTE4 | IN | RJMP | WDR | |

# 6.4 Symbolická jména definovaná programátorem

Symbolická jména (symboly) definovaná programátorem mohou být tvořeny písmeny, číslicemi a znakem podtržení '_'. První znak musí být vždy písmeno nebo znak podtržení. Maximální délka jména není omezena, ale max. délka řádku může být 255 znaků, do této délky se nepočítá poznámka. Kompilátor volitelně rozlišuje malá a velká písmena.

Příklad:

```
CYKLUS1
cyklus1
```

jsou z hlediska kompilátoru shodná jména pokud není zapnuto rozlišování velikosti znaků

# 6.5 Konstanty

Kompilátor rozlišuje 3 možné způsoby zápisu konstant.

## 6.5.1 Číselné konstanty

**Dekadické** obsahují pouze číslice 0-9, nesmí začínat znakem 0

Příklad:

```
DB 10
DB 1930
DB 59
```

**Binární** obsahují pouze číslice 0,1 a musí začínat prefixem 0b

Příklad:

```
DB 0B10101010
DB 0B10101101
```

**Hexadecimální** obsahují pouze číslice 0-9 a znaky A-F. Musí začínat prefixem 0x nebo $

Příklad:

```
DB 0X10
DB 0X10
DB $10
```

**Oktalové** obsahují znaky 0-7, musí začínat znakem 0

Příklad:

```
DB 010
DB 0777
```

### 6.5.2 Znakové konstanty

Obsahují libovolný ASCII znak, který je uzavřený do jednoduchých uvozovek

Příklad:

```
DB 'A','ABC'
```

### 6.5.3 Symbol PC

Zvláštní význam má pak symbol PC, kterému je při překladu přiřazena aktuální hodnota programového čítače adres.

## 6.6 Operátory

V matematických a logických výrazech jsou symbolická jména a konstanty spojeny operátory. Kompilátor má implementovány následující operátory :

| Operátor | Popis | Priorita |
|---|---|---|
| ! | Logická negace | 14 |
| ~ | Binární negace | 14 |
| - | Unární mínus | 14 |
| * | Násobení | 13 |
| / | Celočíselné dělení | 13 |
| - | Odčítání | 12 |
| + | Sčítání | 12 |
| >> | Bitový posun vpravo | 11 |
| << | Bitový posun vlevo | 11 |
| > | Větší než | 10 |

| | | |
|---|---|---|
| < | Menší než | 10 |
| >= | Větší nebo rovno | 10 |
| <= | Menší nebo rovno | 10 |
| == | Rovno | 9 |
| != | Není rovno | 9 |
| & | Bitový AND | 8 |
| ^ | Bitový XOR | 7 |
| \| | Bitový OR | 6 |
| && | Logický AND | 5 |
| \| \| | Logický OR | 4 |

Výrazy jsou vyhodnocovány podle priority. Při stejné úrovni priority je výraz vyhodnocován zleva doprava. Pořadí vyhodnocování může být změněno použitím závorek.

### 6.6.1 Logická negace

Symbol: !
Popis: Unární operátor, který vrací 1 jestliže výraz je nulový, jinak vrací 1

Priorita: 14
Příklad: `LDI R16,!0XF0`

### 6.6.2 Binární negace

Symbol: ~
Popis: Unární operátor, který invertuje všechny bity vstupního výrazu

Priorita: 14
Příklad: `LDI R16,~0XF0`

### 6.6.3 Minus

Symbol: -
Popis: Unární operátor, který vrací aritmetickou negaci vstupního výrazu

Priorita: 14
Příklad: `LDI R16,-2`

### 6.6.4 Násobení

Symbol:*
Popis: Binární operátor, který vrací výsledek násobení dvou výrazů

Priorita: 13
Příklad: `LDI R30,LABEL*2`

### 6.6.5 Dělení

Symbol: /
Popis: Binární operátor který vrací celočíselný podíl levého výrazu děleného pravým výrazem
Priorita: 13
Příklad: `LDI R30,LABEL/2`

### 6.6.6 Sčítání

Symbol:+
Popis: Binární operátor, který vrací součet dvou výrazů
Priorita: 12
Příklad: `LDI R30,C1+C2`

### 6.6.7 Odečítání

Symbol: -
Popis: Binární operátor, který odečte pravý výraz od levého výrazu

Priorita: 12

Příklad: `LDI R17,C1-C2`

### 6.6.8 Posun doleva

Symbol: <<
Popis: Binární operátor, který vrací levý výraz posunutý vlevo o počet bitů daný pravým výrazem. Bity na pravé straně jsou nahrazeny 0. Operátor pracuje v 32 bitové aritmetice.

Priorita: 11
Příklad: `LDI R17,1<<BITMASK`

### 6.6.9 Posun doprava

Symbol: >>
Popis: Binární operátor, který vrací levý výraz posunutý vpravo o počet bitů daný pravým výrazem. Bity na levé straně jsou nahrazeny 0. Operátor pracuje v 32 bitové aritmetice.

Priorita: 11
Příklad: `LDI R17,C1>>C2`

### 6.6.10 Menší než

Symbol: <
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz je menší než znaménkový pravý výraz, jinak vrací 0
Priorita: 10
Příklad: `ORI R18,BITMASK*(C1<C2)+1`

### 6.6.11 Menší nebo rovno

Symbol: <=
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz je menší nebo roven znaménkovému pravému výrazu, jinak vrací 0
Priorita: 10
Příklad: `ORI R18,BITMASK*(C1<=C2)+1`

### 6.6.12 Větší než

Symbol: >
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz je menší než znaménkový pravý výraz, jinak vrací 0
Priorita: 10
Příklad: `ORI R18,BITMASK*(C1>C2)+1`

### 6.6.13 Větší nebo rovno

Symbol: >=
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz je větší nebo roven znaménkovému pravému výrazu, jinak vrací 0
Priorita: 10
Příklad: `ORI R18,BITMASK*(C1>=C2)+1`

### 6.6.14 Rovno

Symbol:==
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz je roven znaménkovému pravému výrazu, jinak vrací 0

Priorita: 9
Příklad: `ANDI R19,BITMASK*(C1==C2)+1`

### 6.6.15 Není rovno

Symbol: !=
Popis: Binární operátor, který vrací 1 jestliže znaménkový levý výraz není roven znaménkovému pravému výrazu, jinak vrací 0

Priorita: 9
Příklad: `SET FLAG=(C1!=C2)`

### 6.6.16 Bitový AND

Symbol: &
Popis: Binární operátor, který vrací bitový AND mezi dvěma výrazy

Priorita: 8
Příklad: `LDI R18,HIGH(C1&C2)`

### 6.6.17 Bitový XOR

Symbol: ^
Popis: Binární operátor, který vrací bitový XOR mezi dvěma výrazy

Priorita: 7
Příklad: `LDI R18,LOW(C1^C2)`

### 6.6.18 Bitový OR

Symbol: |
Popis: Binární operátor, který vrací bitový OR mezi dvěma výrazy
Priorita: 6
Příklad: `LDI R18,LOW(C1|C2)`

### 6.6.19 Logický AND

Symbol: &&
Popis: Binární operátor, který vrací 1 pokud jsou oba výrazy nenulové, jinak 0
Priorita: 5
Příklad: `LDI R18,LOW(C1&&C2)`

### 6.6.20 Logický OR

Symbol: ||
Popis: Binární operátor, který vrací 1 pokusd je alespoň jeden výrazy nenulový
Priorita: 4
Příklad: `LDI R18,LOW(C1||C2)`

## 6.7 Funkce

Jsou definovány následující funkce:

| Funkce | Popis výsledku | Příklad |
|---|---|---|
| LOW | nejnižší byte výrazu | .DB LOW($1234) |
| LO | nejnižší byte výrazu | .DB LO($1234) |
| HIGH | druhý byte výrazu | .DB HIGH($1234) |
| HI | druhý byte výrazu | .DB HI($1234) |
| BYTE1 | nejnižší byte výrazu | .DB BYTE1($1234) |
| BYTE2 | druhý byte výrazu | .DB BYTE2($1234) |
| BYTE3 | třetí byte výrazu | .DB BYTE3($12340000) |
| BYTE4 | čtvrtý byte výrazu | .DB BYTE4($12340000) |
| LWRD | bity 0-15 | .DB LWRD($12340000) |
| HWRD | bity 16-31 | .DB HWRD($12340000) |
| PAGE | bity 16-21 | .DB PAGE($12340000) |
| EXP2 | mocninu 2 | .DB EXP(5) |
| LOG2 | celočíselná část log2 | .DB LOG(1024) |
| SYMDEF | vraci 1 pokud existuje | .IF SYMDEF(ABC) |
| SYMVAL | vraci hodnotu symbolu | LDI R16, SYMVAL(R0) |
| ISREG | Vraci 1 je-li registr | .IF ISREG(ABC) |
| REG | Převod výrazu na registr | MOV REG(15),R16 |

| | | |
|---|---|---|
| DDR | Převod bitu PORT na DDR | SBI DDR(CLK) |
| PIN | Převod bitu PORT na PIN | SBI PIN(CLK) |
| PRT | Převod bitu PORT na PRT | SBI PRT(CLK) |

### 6.7.1 Funkce DDR, PIN, PRT

Tyto funkce slouží pro zjednodušení práce s IO porty. Jejich vstupem je hodnota typu IBIT nebo EQU. Příslušná funkce vyhledá k portu PORTx odpovídají DDRx nebo PINx a vrátí tuto hodnotu. Přitom se kontroluje, zda argumentem této funkce je opravdu symbol odvozený od PORTx.

Příklad:

```
.DEVICE   ATMEGA162
.IBIT     DAT = PORTA, 1
.EQU      KLV = PORTB

CBI  DDR(DAT)
CBI  PRT(DAT)
SBIS PIN(DAT)
IN   R1, PIN(KLV)
```

## 6.8 Formát zdrojového textu

Obecný formát jednoho řádku zdrojového textu je definován následovně. Přesná délka jednotlivých polí není určena. Pro vyšší přehlednost a pohodlnější práci se však doporučuje vyhradit pro každé pole 8 znaků. Integrovaný editor má tabelátor nastavený standardně na tuto velikost.

```
[NÁVĚŠTÍ:] (PSEUDO)INSTRUKCE [OPERAND(Y)] [;KOMENTÁŘ]
```

### 6.8.1 Návěští

Návěští je symbolické jméno, které musí být ukončeno dvojtečkou. Dvojtečka není součástí symbolického jména, používá se pouze při vyšší přehlednost, případně pro kompatibilitu s jinými kompilátory. Tomuto jménu je při překladu přiřazena konkrétní adresa cílového kódu. Každé návěští smí být deklarováno pouze jednou. Při vícenásobném výskytu téhož jména v poli návěští hlásí kompilátor chybu.

Příklad:

```
NAVESTI1:
NAVESTI1:
```

# 6.9 Pseudoinstrukce (direktivy assembleru)

Assembler obsahuje mnoho direktiv. Tyto nejsou přímo překládány na strojový kód, ale slouží pro řízení překladu, rezervaci paměti, definice symbolů apod. Je-li parmetrem direktivy symbol, musí být jeho hodnota známá ihned při překladu této direktivy, není možná dopředná deklara tohoto symbolu (jako např. návěští). Kompilátor má implementovány následující direktivy:

### 6.9.1 Definice hodnot symbolických jmen - EQU, SET

Syntaxe:

```
.EQU <jméno>=<výraz>
.SET <jméno>=<výraz>
```

Symbolickému jménu je přiřazena hodnota získaná vyhodnocením výrazu. Pseudoinstrukce EQU a SET jsou vzájemně téměř ekvivalentní. Kompilátor rozeznává obě pseudoinstrukce z důvodu kompatibility. U direktivy SET je na rozdíl od EQU přípustná redeklarace s novou hodnotou.

Příklad:

```
.EQU RAM1=100
.EQU LOCK=RAM1*2-10
.SET RAM1=100
.SET RAM1=RAM1+1
```

### 6.9.2 Definice symbolických jmen registru - DEF

Syntaxe:

```
.DEF <jméno>=<registr>
```

Symbolickému jménu je přiřazena hodnota registru, není přípustná redeklarace

Příklad:

```
.DEF POC1=R18
.DEF POC2=R12
```

### 6.9.3 Výběr segmentu – CSEG, DSEG, ESEG

Syntaxe:

```
.CSEG <výraz>
.DSEG <výraz>
.ESEG <výraz>
```

Pseudoinstrukce slouží pro výběr jednoho z 3 adresových prostorů, výchozím segmentem je CSEG. Aktuální segment určuje, ve kterém paměťovém prostoru se vyhrazuje paměť při použití pseudoinstrukce .BYTE, popř. nastavuje čítač segmentu použitím ORG. Přehled segmentů, rozsahů a použitelných pseudoinstrukcí pro definici symbolů:

| **Segment** | **Název** | **Vyhrazení** |
|---|---|---|
| Programový | CSEG | .ORG |
| Datový | DSEG | .BYTE |
| EEPROM | BSEG | .BYTE |

Příklad:

```
          .DSEG 100
VAR1:     .BYTE 1
TABLE:    .BYTE TAB_SIZE

          .ESEG
EEVAR1:   .DW 0XFFFF

          .CSEG
START:    JMP INIT
```

### 6.9.4 Definice 8 bitových konstant v paměti programu - DB (Define Byte)

Syntaxe:

```
[návěští:] DB <výraz> [,<výraz>]
```

Pseudoinstrukce DB postupně ukládá jednobajtové hodnoty, získané vyhodnocením jednotlivých výrazů na aktuální adresy, počínaje současnou adresou programového čítače. Výrazy musí nabývat hodnot v rozsahu 0 - $FF. Povoleny jsou také řetězce znaků uzavřené v apostrofech. Například posloupnost znaků '123' se uloží jako $31, $32, $33. Počet výrazů za jednou pseudoinstrukcí DB je omezen pouze délkou řádku. Lze použít v programovém segmentu CSEG nebo datovém ESEG. Je-li tato direktiva použita v segmentu ESEG jsou tyto hodnoty naprogramovány do EEPROM v okamžiku programování paměti FLASH (*WinPROG – Odeslat na HW*).

Příklad:

```
TEXT1:    DB 'AHOJ'
TABS1:    DB 10,0FEH,LO(12345)
```

### 6.9.5 Definice 16 bitových konstant - DW (Define Word)

Syntaxe:

```
[návěští:] DW <výraz> [,<výraz>]
```

Pseudoinstrukce DW postupně ukládá dvoubajtové hodnoty, získané vyhodnocením jednotlivých výrazů, na aktuální adresy, počínaje současnou adresou programového čítače. Výrazy musí nabývat hodnot v rozsahu 0 až 0FFFFh. Vyšší byte je ukládán na nižší adresu a nižší byte na vyšší adresu. Počet výrazů za jednou pseudoinstrukcí DW je omezen pouze délkou řádku. Lze použít v programovém segmentu CSEG nebo ESEG. Je-li tato direktiva použita v segmentu ESEG jsou tyto hodnoty naprogramovány do EEPROM v okamžiku programování paměti FLASH (*Win-PROG – Odeslat na HW*).

Příklad:

```
TABLE:    DW 1000H, 2589
          DW TEXT1, TEXT1+1
```

### 6.9.6 Nastavení programového čítače - ORG

Syntaxe:

```
[návěští:] ORG <výraz>
```

Pseudoinstrukce ORG nastaví hodnotu čítače aktuálního segmentu na hodnotu danou vyhodnocením výrazu. PřI pokusu o nastavení větší hodnoty než je fyzický rozsah aktuálního segmentu nebo na hodnotu menší, než je hodnota aktuální, ohlásí kompilátor chybu. Lze použít pouze ve všech segmentech

Příklad:

```
INT3:     .ORG 3
TABLE:    .ORG $1000
```

### 6.9.7 Podmíněný překlad - IF, IFDEF, ELSE, ENDIF

Syntaxe:

```
[NÁVĚŠTÍ:] .IF <VÝRAZ>
[NÁVĚŠTÍ:] .ELSE
[NÁVĚŠTÍ:] .ENDIF
[NÁVĚŠTÍ:] .IFDEF <SYMBOL>
```

Kompilátor vyhodnotí výraz za IF a pokud je výsledek různý od nuly, překládá následující zdrojový text až po pseudoinstrukci ELSE nebo ENDIF. Pokud je nalezena pseudoinstrukce ELSE ignoruje kompilátor zdrojový text až do nalezení pseudoinstrukce ENDIF. Je-li výraz za IF nulový, ignoruje kompilátor následující zdrojový text až po pseudoinstrukci ELSE nebo ENDIF. Pokud je nalezena pseudoinstrukce ELSE překládá kompilátor zdrojový text až do nalezení

pseudoinstrukce ENDIF. Vnoření pseudoinstrukcí IF, ELSE, ENDIF je povoleno. Kompilátor musí být schopen vyhodnotit výraz již v prvním průchodu. To znamená, že výraz musí obsahovat pouze taková symbolická jména, jejichž hodnota již byla definována. V opačném případě kompilátor oznámí chybu při překladu. Pro IFDEF je funkce stejna s tím rozdílem že se vyhodnocuje existenci symbolu.

Příklad:

```
.EQU SVITI=1
.IF SVITI
NOP
.
.
.
.ELSE
NOP
.
.
.
.ENDIF
```

### 6.9.8 Ukončení zdrojového textu - EXIT

Syntaxe:

```
[návěští:]     .EXIT
```

Pseudoinstrukce EXIT oznamuje kompilátoru konec překládané části zdrojového textu. Jakýkoli další text je ignorován. Pseudoinstrukce END je nepovinná.

### 6.9.9 Vložení externího souboru - INCLUDE

Do souboru se přidá obsah specifikovaného souboru. Pokud není zadána cesta, hledá se tento soubor ve stejné složce jako je aktuální zdrojový text. Pokud je nalezena chyba v externím souboru a ten není otevřen, kliknutím na chybový výpis se soubor otevře a nastaví se na řádek s chybou. S externím souborem je možné pracovat stejně jako s hlavním, tzn zadávat breakpointy, krokovat apod. Externí soubor může obsahovat odkaz na další externí soubor.

Syntaxe:

```
.INCLUDE <“jméno souboru“>
```

Příklad:

```
.INCLUDE “FILE001.ASM“
```

### 6.9.10 Specifikace hlavního souboru – MAIN

V případě, že program je rozdělen do více souborů (s využitím direktivy INCLUDE ), lze pomocí direktivy MAIN určit, který soubor je hlavní, tzn. kterým začíná překlad. Tato direktiva se uvádí na začátek vkládaného souboru. Jejím použitím je možné začít překládat zdrojový text souborem který nemusí být v dané chvíli aktuálně editovaným souborem, dokonce ani nemusí být v editoru otevřen.

Příklad: projekt se skládá ze souboru P1.ASM a P2.ASM. Přitom P2.ASM je vložen do P1.ASM pomocí direktivy INCLUDE “P2.ASM” a využívá proměnné deklarované v P1.ASM. Potom máme-li v editoru aktuální soubor P2.ASM a přeložíme-li, jej nejsou tyto proměnné vidět a překlad skončí s chybou. Museli bychom se přepnout do P1.ASM, přeložit a vrátit se zpět do P2.ASM. Pokud však použijeme v P2.ASM direktivu MAIN “P1.ASM” provede se toto automaticky a překlad proběhne v pořádku. Význam použití této direktivy je tedy především v rozsáhlém projektu, který se skládá z více do sebe vzájemně zanořených souborů.

Příklad:

```
.MAIN “FILE001.ASM“
```

### 6.9.11 Výběr typu procesoru – DEVICE

Direktiva DEVICE umožňuje definovat podmínky překladu pro konkrétní typ procesoru. Použitím této direktivy se přizpůsobí instrukční soubor, nahrají se názvy IO registrů a dalších konstant. Dále se nastaví velikosti paměťových prostorů RAM, EEPROM a FLASH v debuggeru. Pokud není tato direktiva použita nejsou překladači známy symboly příslušné pro daný typ procesoru a instrukční soubor není omezen.

Syntaxe:

```
.DEVICE <typ procesoru>
```

Příklad:

```
.DEVICE AT90S1200
```

### 6.9.12 Direktiva LIST

Slouží pro zapnutí výpisu programu v okně debuggeru a v souboru LST. Standardní hodnota je zapnuto

Syntaxe:

```
.LIST
```

### 6.9.13 Direktiva NOLIST

Slouží pro potlačení výpisu programu v okně debuggeru a v souboru LST

Syntaxe:

```
.NOLIST
```

### 6.9.14 Direktiva RESW

Je-li použita před definici symbolu, je tento symbol považován za reservované slovo

Syntaxe

```
.RESW <vyraz>
```

Příklad:

```
.RESW     1
.EQU      ABC = 123
.RESW     0
```

### 6.9.15 Direktivy MACRO, ENDMACRO

Umožňují nadefinovat posloupnost instrukcí, kterou je možno opakovaně vložit do zdrojového textu. Makro může mít až 10 parametrů, označených symboly @0..@9. Zvláštní význam má symbol @10 za který se dosadí skutečný počet předávaných parametrů. Je-li jako parametr registr a v definici makra je konstrukce @0+1 dosadí se následující registr. Makro se v debuggeru provádí jako jedna instrukce.

Syntaxe

```
.MACRO <jmeno makra>
.ENDMACRO
```

Příklad:

```
.MACRO    LDI32
LDI  @0 + 0,   BYTE1(@1)
LDI  @0 + 1,   BYTE2(@1)
LDI  @0 + 2,   BYTE3(@1)
LDI  @0 + 3,   BYTE4(@1)
.ENDMACRO

LDI32 R16,12345
```

### 6.9.16 Direktivy SECT, ENDSECT

Umožňují nadefinovat posloupnost instrukcí (sekci). V sekci je možno opakovaně použít stejná návěští. Typické využití je pro podprogramy. Sekce se volá stejně jako návěští. Není možné

definovat sekci uvnitř sekce. Existuje-li návěští stejného jména v sekci i mimo ni, provede se případný skok na návěští uvnitř sekce. Pokud tam takové návěští není provede se skok mimo sekci. Dovnitř sekce není možno přímo skočit, ani z jedné sekce do druhé.

Syntaxe

```
.SECT <jmeno sekce>
.ENDSECT
```

Příklad:

```
          .SECT     DELAY1
          LDI       R17, 0
LOOP2:    LDI       R16, 0
LOOP1:    DEC       R16
          BRNE      LOOP1
          DEC       R17
          BRNE      LOOP2
          RET
          .ENDSECT

          CALL      DELAY1
```

### 6.9.17 Direktivy IBIT, RBIT

Umožňují definovat bitovou adresu pro instrukce BLD, BST, CBI, SBRS, SBRC, CBI, SBI, SBIS, SBIC. IBIT definuje bit v IO prostoru, RBIT definuje bit v registru

Syntaxe

```
.IBIT <jmeno> = <výraz>, <výraz>
.RBIT <jmeno> = <registr>, <výraz>
```

Příklad:

```
.IBIT CLK = PORTA, 5
.RBIT TMP = R16, 2

SBI   CLK
CBI   CLK
BLD   TMP
```

### 6.9.18 Deklarace jednotky - UNIT, ENDUNIT, EXPORT

Má podobnou funkci jako sekce, umožňuje logicky související část programu uzavřít mezi direktivy UNIT a ENDUNIT. Tím vznikne tzv. jednotka, jejíž symbolový obsah (deklarace, návěští, sekce) je navenek neviditelný. Tímto je umožněno mimo jiné opakované použití názvů symbolů, ale hlavním smyslem je zabránit nekoordinovanému volání podprogramů navzájem a zpřehlednit deklaraci proměnných a tak zabezpečit větší přehlednost programu.

Symbol deklarovaný uvnitř jednotky je možné zviditelnit směrem nahoru, nad rámec jednotky pomocí direktivy EXPORT. Směrem dolů platí pravidlo, že symboly deklarované na vyšší úrovni jsou na nižší úrovni (uvnitř sekce nebo jednotky) vždy viditelné, pokud nejsou předeklarovány (zastíněny) symbolem deklarovaným na úrovni nižší. Tedy symbol lokální stejného jména má přednost před symbolem globálním - vyšším. Symboly ze sousedních jednotek nejsou vzájemně viditelné, pokud nejsou exportovány pomocí direktivy EXPORT.

Pravidla a omezení:

- exportovat lze pouze symboly těchto typů: LABCSG (sekce), LABDSG, LABESG, EQU, SET, RBIT a IBIT.
- uvnitř jednotky lze definovat lokální makro, avšak jméno tohoto makra se nesmí shodovat se jménem makra na vyšší úrovni (makro nesmí zastínit jiné makro). Makro nelze exportovat, proto makro definované uvnitř jednotky je navenek neviditelné.
- stejný princip přidělování jmen platí také pro lokální definice symbolů EQU, SET, DEF, RBIT, IBIT.
- pro symboly typu návěští LABCSG, LABDSG, LABESG lze uvnitř jednotky použít stejné jméno jako je již použité na vyšší úrovni (lze je zastínit). Potom však tento symbol nelze exportovat.
- jméno sekce má stejné vlastnosti jako návěští LABCSG
- v sousední jednotce lze libovolně používat jména symbolů deklarovaných v jiné jednotce pokud tyto symoly nejsou exportovány

Syntaxe:

```
.UNIT <jmeno sekce>
.EXPORT <symbol>
<LOKALNI DEKLARACE>
<LOKALNI MAKRA>
<LOKALNI SEKCE>
.ENDUNIT
```

Příklad:

```
          .UNIT     UNIT1
          .EXPORT   PROC1
          .EXPORT   PROC2
          .DSEG
ABC:      .BYTE 1             ;LOKALNI PROMENNA DSEG
          .CSEG
          .SECT     PROC1     ;EXPORTOVANO
```

```
LAB1:     NOP
          .ENDSECT
          .SECT     PROC2     ;EXPORTOVANO
LAB1:     NOP
          .ENDSECT
          .SECT     PROC3     ;NENI EXPORTOVANO
LAB1:     NOP
          .ENDSECT
          .ENDUNIT
```

### 6.9.19 Komentář

Komentář je libovolný text, který začíná středníkem. Narazí-li kompilátor při překladu na středník, pak celý zbytek textu až do konce řádku považuje za poznámku, která není kompilována a je pouze přenesena do výstupního protokolu o překladu. Mimo obecný formát zdrojového řádku jsou možné i tyto formáty:

```
[návěští:] ;komentář
;komentář
```

Příklad:

```
TABLE .DW 1000,2589  ;DEFINICE TABULKY
;ZDE JE DEFINOVANA TABULKA KONSTANT
```

## 6.10 Protokol o překladu a chybová hlášení

Kompilátor během překladu vytváří protokol o překladu (soubor *.LST ). Tento protokol zahrnuje opis zdrojového textu, přeložený cílový kód, případná chybová hlášení a na závěr výpis tabulky symbolů. Jednomu řádku zdrojového textu může odpovídat i více řádků v protokolu o překladu, zvláště v případě rozsáhlých definic za pomoci pseudoinstrukcí DB, DW a TEXT . Obecný formát jednoho řádku protokolu o překladu je následující:

<adresa> <cíl.kód> <popis zdrojového textu>

Adresa je tvořena čtyřmi číslicemi v hexadecimálním tvaru a udává absolutní adresu, na kterou se ukládá cílový kód. Cílový kód je tvořen hexadecimálními číslicemi operačního kódu a případných operandů. Opis zdrojového textu je odpovídající řádek ze zdrojového souboru.

Příklad:

```
0198 C2B6 CLR KLV.6
019A E5B0 IN  R16,KLV
019C 540F ANL A,#00001111B
```

### 6.10.1 Chybová hlášení

Skončí-li překlad bez chyby, pak je v protokolu o překladu uveden text :

- **Při překladu zdrojového textu nebyly nalezeny žádné chyby.**

Pokud se nevyskytují chyby, ale varovné hlášení je text tento:

- **Při překladu zdrojového textu nebyly nalezeny žádné chyby, ale xx varování**

Při překladu se zjištěnými chybami se zobrazuje seznam chyb s číslem řádku a popisem chyby. Kompilátor vždy překládá vstupní text až do konce zdrojového souboru, bez ohledu na to, zda při překladu došlo k chybám.

### 6.10.2 Chyby při překladu

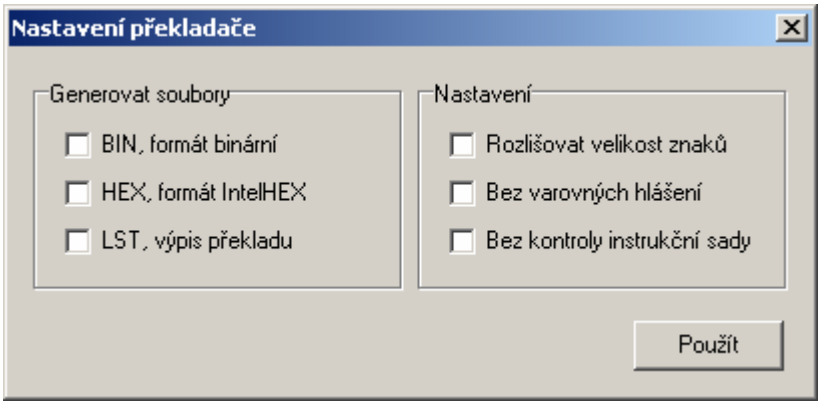
Kompilátor ve většině případů generuje chybové hlášení typu: **Řádek 1234: chyba, nalezeno [xxx], očekává se [yy1,yy2,....].** Kde xxx je chybný nalezený syntaktický prvek a yyy jsou správné možnosti na tomto místě. Např. zápis **LDI R16,** vygeneruje chybové hlášení: **Řádek 1: chyba, nalezeno [konec řádku], očekává se [výraz, symbol EQU, symbol SET, label]**. Kromě toho se mohou vyskytnout další chybová hlášení. Všechna jsou generována do okna protokolu o překladu:

- **chyba v konstantě**: chyba v číselné konstantě
- **očekává se operand**: chybí operand např.: LDI R16, 1+
- **dělení nulou**: při vyhodnocování výsledku výrazu nalezl kompilátor na pravé straně operátoru / (děleno) hodnotu nula.
- **nalezen neznámý identifikátor**: kompilátor nalezl symbolické jméno, jež doposud nebylo definováno
- **nalezena nadbytečná ):** přebývá závorka na konci výrazu
- **očekává se )** : chybí závorka na konci výrazu
- **očekává se (** : např. při zápisu LO 4567 musí být uvedeno LO(45670)
- **nesouhlasí počet závorek**: různý počet pravých a levých závorek
- **výraz nelze vyhodnotit**: většinou nekompatibilní typy dat. Nelze sčítat např. konstantu a registr. Registr nepředstavuje číselnou hodnotu, např. ST 1-X,R0
- **očekává se '**: neukončený textový řetezec
- **neočekávaný operátor**: typicky chybí operand např. LDI R16, MOD 5
- **program je příliš velký, používáte omezenou verzi WinAVR**: překládaný program je větší než 1024 bytů

- **chybné číslo bitu:** číslo bitu mimo rozsah 0..7, např. BSET 10
- **chybný indexovací registr**: chyba v použití indexovacího registru, např: LDD R0, X+1
- **relativní skok mimo rozsah**: příliš dlouhý skok, nejčastěji u podmíněných skoků
- **skok mimo aktuální stránku**: chyba v rozsahu adresy
- **aktuální hodnota PC je větší než požadovaná nebo mimo rozsah**: hodnota za pseudoinstrukcí ORG je vyšší než aktuální hodnota čítače adres
- **nalezena neočekávaná direktiva IF**: kompilátor nalezl vnořené IF v dosud neukončené rozhodovací struktuře
- **nalezena neočekávaná direktiva ELSE**: kompilátor nalezl pseudoinstrukci ELSE bez úvodu rozhodovací struktury definovaného pseudoinstrukcí IF
- **nalezena neočekávaná direktiva ENDIF**: kompilátor nalezl pseudoinstrukci ENDIF bez úvodu rozhodovací struktury definovaného pseudoinstrukcí IF
- **hodnota mimo rozsah 0..31**: výsledek výrazu je větší než cílový operand
- **hodnota mimo rozsah 0..63**: výsledek výrazu je větší než cílový operand
- **hodnota mimo rozsah 0..255**: výsledek výrazu je větší než cílový operand
- **hodnota mimo rozsah 0..65535**: výsledek výrazu je větší než cílový operand
- **symbol již byl deklarován**: použití již deklarovaného symbolu na místě návěští
- **duplicitní deklarace adresy nebo dat**
- **nelze použít v aktuálním segmentu**
- **hodnota je mimo rozsah aktuálního segmentu**
- **nelze rezervovat nulovou velikost adresového prostoru**
- **neočekávaná direktiva**
- **číslo registru mimo rozsah**
- **chybný registr nebo registrový pár**
- **typ procesoru musí být uveden jako první**
- **typ procesoru nelze redeklarovat**: direktivu .DEVICE lze použít pouze jednou
- **instrukce není v instrukční sadě aktuálního procesoru**: použití instrukce, která není v sadě procesoru zvoleného .DEVICE
- **tento byte již byl definován**: pokus o redefinici bytu FUSE nebo LOCK
- **hodnota mimo rozsah 0..31**
- **neplatná adresa portu:** ve funkci PRT, DDR, PIN
- **(lokální) symbol, nelze přímo vyhodnotit, protože umožňuje dopřednou deklaraci:** chyba při překladu direktiv, kdy jejich parametrem je dopředně deklarovaný symbol

# 6.11Varovná hlášení

Pro usnadnění odlaďování programu se generují i varovná hlášení. Nejsou to přímo syntaktické chyby avšak ukazují na některé nestandardní akce, které mohou vést k nesprávnému chování programu. Jejich výpis je možné potlačit v nabídce Nastavení překladače.




- **požadovaný adresový prostor nebo jeho část je již rezervován(a)**
- **nalezen lichý počet bytů, doplněn nulový byte**: chyba v direktivě .DW
- **symbol je deklarován, ale není použit**
- **nedovolená kombinace indexu a registru**
- **neznámý typ procesoru**: chybný identifikátor v .DEVICE
- **registr již byl definován direktivou .DEF**: pokus o redefinici registru
- **frekvence musi být 1..100 MHz**: překročen rozsah direktivy XTAL

# 7 Popis INI souborů

## 7.1 WinAVR_C.ini – nastaveni uzivatele

Tento soubor obsahuje většinu nastavení prostředí WinAVR. Soubor může být umístěn ve stejné složce jako aplikace, nebo v případě, že je aplikace spoušťena ze společného síťového disku se hledá ve složce která je uvedena v souboru WINAVR_M.INI jako položka CUSTINI. Nastavení je pak přečteno z namapované síťové cesty konkrétního přihlášeného uživatele.

```
Toto má každý uživatel svoje

[Editor]
AutoTime=0                    ;ukladání
ParmTime=1000                 ;doba pro napovedu instrukci
BackFile=0                    ;vytvaret BAK
CompFile=0                    ;ulozit pri prekladu
StatusBar=1                   ;zobrazovat stavovy radek
LastOpen=1                    ;otevirat posledni
LastName=H:\DATA\DEMO2.ASM    ;posledni soubor
CountFile=2                   ;pocet souboru
ActivFile=1                   ;aktivni soubor
AutoFormat=0                  ;formatovat po prekladu
FontStyle=0                   ;font
FontColor=16777215            ;font
FontName=Courier              ;font
FontSize=12                   ;font
BackColor=0                   ;barva pozadi
HighLighter=1                 ;barevna syntae
GutterVisible=1               ;leva lista
PrintBW=1                     ;cernobily tisk
TabWidth=8                    ;tabelator

[HighLight]                   ;barevna syntaxe
Style0=2
Color0=32768
Style1=0
Color1=8421376
Style2=0
Color2=16711680
Style3=0
Color3=0
Style4=0
Color4=255
Style5=0
```

```
Color5=536870911
Style6=0
Color6=16711935
Style7=0
Color7=128

[Format]                        ;formatovani textu
E1=0
E2=16
E3=24
E4=48
R1=2
R2=2
R3=2
R4=2
C1=1

[Panel]                         ;zobrazeni panelu
Ver=2                           ;verze
P1=1
P2=1
P3=1
P4=1

[Height]
H1=235                          ;vyska panelu
H2=235                          ;vyska panelu
H3=125                          ;vyska panelu
H4=125                          ;vyska panelu
H5=125                          ;vyska panelu
H6=125                          ;vyska panelu

[Animate]
Delay=5                         ;rychlost animace

[Load]
Ports=0                         ;nahravat prubehy na portech
Eeprom=0                        ;nahravat stav EEPROM

[Xtal]
Freq=12                         ;frekvence krystalu v debuggeru

[Debugger]
Check=0                         ;kontroly v debuggeru

[LastFile]
File01=H:\DATA\DEMO2.ASM        ;posledni otevrene soubory

[LastOpen]
File00=H:\ASM\Emul51_12.asm     ;posledni otevrene soubory
```

```
[Used Icons]
Action3=1                       ;pouzite ikony

[IcoCap]
Cap=1                           ;zobrazovat popisy ikon

[CPL]
BIN=0                           ;generovat soubor BIN
HEX=0                           ;generovat soubor HEX
LST=0                           ;generovat soubor LST
CSV=0                           ;rozlisovat velikost znaku
NWR=0                           ;preklad bez varovnych hlaseni

[KEY]
MODE=0                          ;typ HW klice
```

## 7.2 WinAVR_M.ini – nastaveni pro síť

Pokud existuje, tento soubor musí ležet ve stejné složce jako aplikace. Obsahuje cestu k souborům WIN_C.INI, což je uživatelské nastavení jednotlivých uživatelů při síťové instalaci a společné parametry síťového přenosu.

```
Toto je společne pro všechny uživatele

[PATH]
CUSTINI=C:\data                 ;cesta k souboru WIN_C.INI
DATAINI=C:\data                 ;default cesta pro data

[CREATE]
ASSOCIATION=0                   ;vytvaret asociaci pro *.asm a *.lst
SHORTCUT=0                      ;vytvaret zastupce na plose
STARTMENU=0                     ;vytvaret zastupce v STARTu

[Network]
PortTCPIP=0                     ;port TCP-IP 0=auto
EnumTime=500                    ;timeout pro pripojeni relaci
UseGlobalSettings=0             ;pouzit sitove nastaveni prednostne
NetOutTime=1000                 ;out time prenosu
TestTime=5000                   ;doba uzivatele po prenosu
SyncTime=1000                   ;interval synchronizace
SyncOutTime=5000                ;outtime synchronizace
PackSize=1000                   ;velikost paketu dat
```

## 7.3 WinAVR.ini – nastavení pro místní počítač

Tento soubor obsahuje informace pro konkrétní počítač, nezávisle na přihlášeném uživateli. Většinou se jedná o HW nastavení přenosu dat.

**Toto je pro konkretni počítač nezávisle na uživateli**

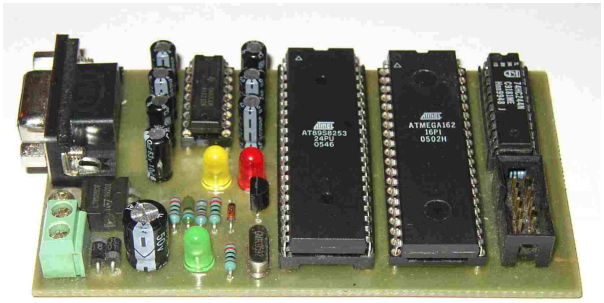
```
[ConnectHW]
HwTimeRst=500                   ;doba signalu reset
HwTimeOut=300                   ;timeout hardwaru
SleepTime=20                    ;systemove cekani na pozadi
ReplyTime=100                   ;interval aktualizace progressbaru
COM=0                           ;COM kde je HW, COM1=0, COM2=1 ...
CommHeader=1                    ;vysilat infoblok pred daty
DisLocked=0                     ;neaktivovat zamykaci bity v procesoru
USB_Description=                ;název otevíraného USB zařízení
USB_SerialNumber=               ;sériové číslo USB (má přednost)
USB=0                           ;USB>0 = pouzit USB pro komunikaci
RunAfterSend=0                  ;rezim spusteni programu

[Network]
PortTCPIP=0                     ;port TCP-IP 0=auto
NetOutTime=1000                 ;out time prenosu
TestTime=5000                   ;doba uzivatele po prenosu
SyncTime=1000                   ;interval synchronizace
SyncOutTime=5000                ;outtime synchronizace
PackSize=1000                   ;velikost paketu dat
HostName=                       ;IP nebo jmeno MASTER
SessName=Win51-1                ;nazev relace
Protocol=0                      ;protokol prenosu
SendType=0                      ;co jsem (local, klient nebo server)
```

# 8 Hardware

## 8.1 Emulátor AVR-ICE

Slouží k hardwarovému testování laděného programu. Je možno jej používat pro testování programu přímo ve vyvíjeném zařízení, kdy se tento emulátor připojí přes plochý kabel do patice pro mikroprocesor. Po připojení modulu portů je možné využívat všechny čtyři porty mikroprocesoru pro připojení přípravků. Tento emulátor se připojuje k PC přes opticky oddělené sériové rozhraní RS232. Napájení emulátoru je buď přímo ze zařízení nebo pomocí konektoru 8..15V DC. Příslušenstvím emulátoru je napájecí kabel, 40-žilový kabel s konektorem PFL40 a volitelně také modul zakončený čtyřmi konektory PSL.


## 8.2 Programátor PROGAVR-COM, USB



Programátor PROG-AVR, je určen pro přímé paralelní programování a čtení obsahu paměti jednočipových procesorů řady ATMEGA a ATTINY firmy ATMEL, nebo pro programování v uživatelské aplikaci (in system programming) prostřednictvím rozhraní SPI. Pro přímé paralelní programování v programátoru jsou podporovány typy procesorů v pouzdru DIL8, DIL20, DIL28 a DIL40, přitom pro některé typy je třeba použít redukci např. DIL40 na DIL28 a dále pro typy

DIL40, které mají napájení uprostřed pouzdra. Procesory v pouzdrech pro povrchovou montáž lze programovat pouze přes SPI. Seznam podporovaných procesorů:

**ATMEGA128, 161, 162, 163, 165, 168, 169, 16, 2560, 2561, 323, 3250, 325, 3290, 329, 32, 406, 48, 649, 64, 8515, 8535, 88, 8, ATTINY11, 12, 13, 15, 22, 2313, 26, 28, 45, AT90CAN128, AT90PWM2, AT90PWM3.**

Všechny tyto typy je možno programovat přes rozhraní SPI přímo v uživatelské aplikaci. Paralelní rozhraní je rychlejší a umožňuje programovat některé funkce navíc. Např. zákaz externího resetu, popř. odemknutí zamknutého procesoru u kterého je zakázáno SPI apod.

### 8.2.1 Programová obsluha:

Pro obsluhu programátoru je určeno prostředí WinAVR. Jako program řídícího procesoru se používá AVRPROGx.BIN. WinAVR umožňuje volit rychlost přenosu dat od 9600 do 57600 b/s. Řídící procesor si automaticky zjišťuje použitou rychlost. Ale ta je omezena např. použitím oddělovacích optočlenů. Jako standardní se používá 9600b/s. Je-li přenos bez problémů můžeme rychlost zvýšit. Programátor může být nakonfigurován i jako HW klíč.

### 8.2.2 Postup pro přímé programování:

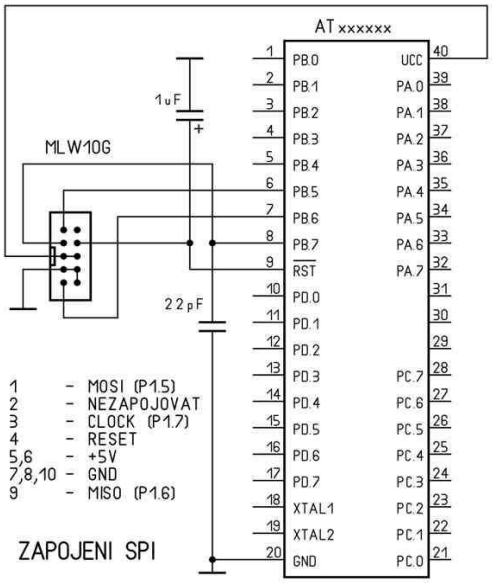
- Vložte procesor do patice (ne pro ISP)
- Připojte napájecí napětí (asi 13-15V), programátor identifikuje vložený procesor a dvakrát blikne červená LED, pokud není procesor vložen, nebo není rozpoznán, bliká LED trvale
- Ve WinAVR stiskněte tlačítko ODESLAT, červená LED svítí během programování, po úspěšném přenosu dat dioda začne blikat systémem dvě bliknutí a dlouhá mezera. Pokud bliká rychle, nebo trvale svítí, došlo k chybě při programování nebo procesor je zřejmě vadný. Pokud programujete přes rozhraní ISP, rozsvítí se po úspěšném naprogramování žlutá LED a program je spuštěn od adresy 0x0000.
- Vypněte napájecí napětí (ne pro ISP)
- Vyjměte procesor z patice (ne pro ISP)

Pozn.: Při jakémkoliv způsobu programování se provádí automaticky 100% verifikace obsahu paměti.

### 8.2.3 Postup při programování v aplikaci:

V programovací patici nesmí být vložen žádný procesor. Programátor a procesor v aplikaci propojíme plochým kabelem. Zapneme napájecí napětí ve vyvíjeném zařízení. Programátor se v tomto případě napájí z připojeného zařízení (aplikace), není tedy potřeba žádné vlastní napájení. Pokud dojde ke správné identifikaci procesoru 2x blikne červená LED, rozsvítí se žlutá LED a pokud aplikační procesor už obsahuje nějaký program, spustí se (není-li procesor rozpoznán nebo je vadný, červená LED bliká trvale). Programátor je nyní připraven k přenosu dat. Stiskněte tlačítko ODESLAT. Aplikační procesor se resetuje, naprogramuje a po správném naprogramování se nový program automaticky spustí. Není třeba vypínat napájecí napětí ani rozpojovat propojova-

cí kabel. Během programování svítí červená LED, po úspěšném přenosu zhasne a program se spustí. Dojde-li k chybě vinou přenosu dat nebo chybou procesoru začne LED blikat trvale a program se nespustí. V tomto případě je třeba vypnout napájení celého systému, znovu zapnout a pokusit se celý postup opakovat. Nepomůže-li to, je zřejmě procesor vadný a je třeba jej vyměnit. Je tedy možné tímto způsobem velmi pohodlně ladit vyvíjený program, aniž bychom se zdržovali rozpojováním kabelů, vypínáním zdroje nebo vyjímáním a vkládáním procesoru do patic.




Pro správnou funkci rozhraní SPI je třeba dodržet několik zásad:

- propojovací kabel mezi programátorem a deskou zařízení musí být co nejkratší, doporučená délka je do 25 cm, rychlost přenosu je velká a pravděpodobnost chyby roste s délkou kabelu. Programátor automaticky určuje rychlost ISP v případě velké délky kabelu se rychlost zmenšuje.
- na desce je třeba zapojit všechny 3 vodiče GND aby bylo spojení zemí co nejlepší s co nejmenší indukčností, rovněž tak je třeba zapojit oba vodiče +5V.
- resetovací kondenzátor má mít kapacitu 1 – 2 □F, bez dalšího přídavného odporu
- mezi vývodem P1.7 a GND je vhodné připojit keramický kondenzátor 22pF proti zemi pro potlačení zákmitů na vedení
- vývody P1.5, P1.6 a P1.7 je možné, i přes to, že jsou součástí SPI, dále využívat v naší aplikaci. Je však třeba, aby jejich funkce nebyla blokována vnějším obvodem. Budič 74HCT244 na programátoru je schopen dodat značný proud a tak „vnutit“ na vývod procesoru potřebné logické úrovně. V zásadě je to možné tehdy, pokud se daný vývod procesoru využívá jako výstupní. Ale není to možné pokud bude daný vývod použitý jako vstup a přímo spojen např. s výstupem nějakého logického členu !!! (pozn. nejcitlivější na rušení je vývod CLK - P1.7, proto jej takto využijeme jen v krajním případě). Po naprogramování přechází výstupy budiče do třetího stavu a nijak neblokují funkci procesoru.
- během přenosu dat do procesoru nezapínejte v okolí žádné elektrické zařízení, rušení, které vzniká zapnutím nebo vypnutím může způsobit chybu přenosu dat

- pozor – programátor není elektricky oddělen od PC, pro napájení Vašeho zařízení používejte vždy bezpečný, dokonale oddělený napájecí zdroj. Má-li Vaše zařízení zem spojenou s ochranným kolíkem zásuvky, zapojte jeho zdroj a zdroj PC do jedné zásuvky, minimalizujete tak vliv zemních smyček na přenos signálu

### 8.2.4 Vnitřní EEPROM

Procesory AVR obsahují vnitřní paměť EEPROM, kterou je možné také snadno programovat. Stačí k tomu ve zdrojovém textu zvolit segment ESEG a pomocí direktivy .DB nebo .DW nadefinovat obsah této paměti.

### 8.2.5 Programování FUSE, LOCK

Pomocí direktiv .LOFUSE, .HIFUSE, .EXFUSE a .LOCK lze snadno definovat obsah těchto konfiguračních bytů. Není-li příslušná direktiva použita, konfigurační byte zůstává beze změny. Zde je možné si nepozorností zakázat komunikaci přes ISP, v takovém případě lze procesor odblokovat paralelním způsobem programování.

### 8.2.6 Zapojení kabelu RS232:

Tento typ programátoru využívá kabel, který je zapojen jako prodlužovací kabel 1:1 na Cannon 9, tedy všechny vývody jsou spojeny souhlasně. S výhodou lze využít koupený hotový kabel. Pokud budeme kabel vyrábět, postačí zapojit pouze vývody číslo: 3 (TxD), 5 (GND), 6 (DSR), 2 (RxD).

### 8.2.7 Čtení paměti programu

Programátor umožňuje snadné čtení paměti FLASH, EEPROM i konfiguračních bytů. Použijeme k tomu nabídku Přečíst paměť HW. Musíme zadat počet bytů, které se mají načíst a kam se má načtený soubor uložit.