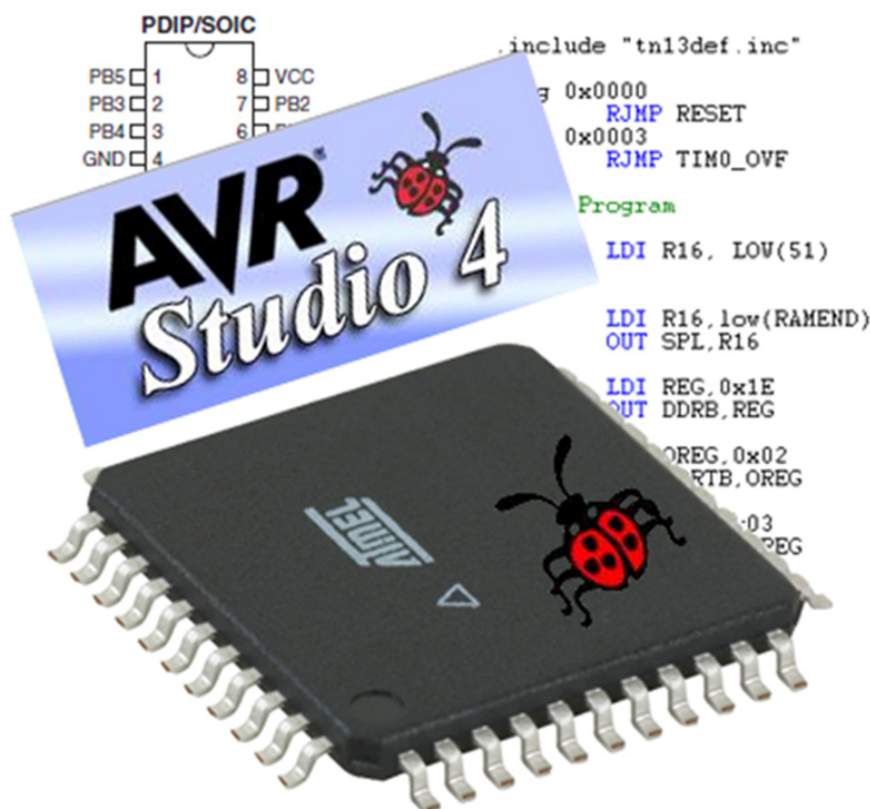


# Mikroprocesory AVR Tiny

## Sbírka úloh



Programování v assembleru (jazyk symbolických adres)

Vývojové diagramy

AVR Studio - vývojové prostředí (Stručný popis)

Ukázkové příklady

Přílohy

Použitá literatura



# Mikroprocesory AVR Tiny

Sbírka úloh

Autor: Ing. Miroslav Duch

Copyright © 2009 Střední průmyslová škola, Trutnov, Školní 101. Všechna práva vyhrazena.

Tato publikace vznikla v rámci projektu „Moderní výuka mikroprocesorové techniky“ spolufinancovaného Královéhradeckým krajem.



**Střední průmyslová škola, Trutnov, Školní 101**

Školní 101

541 01 Trutnov 1

Tel.: 499 813 071, fax: 499 814 729

E-mail: [skola@spstrutnov.cz](mailto:skola@spstrutnov.cz)

URL: <http://www.spstrutnov.cz>

**VAŠE SPOJENÍ SE VZDĚLÁNÍM**

# Obsah

<b>1. Programování v assembleru (jazyk symbolických adres)</b> .....	<b>- 1 -</b>
<b>2. Vývojové diagramy</b> .....	<b>- 1 -</b>
2.1. Prvky vývojového diagramu .....	- 2 -
<b>3. AVR Studio - vývojové prostředí (Stručný popis)</b> .....	<b>- 3 -</b>
3.1. Úvod .....	- 3 -
3.2. Vytvoření nového projektu .....	- 3 -
3.3. Otevření a uložení existujícího projektu (*.aps).....	- 5 -
3.4. Pracovní (vývojové) prostředí.....	- 5 -
3.5. Kompilace programového kódu .....	- 7 -
3.6. Spuštění (ladění) programu.....	- 8 -
3.7. Nahrávání (Load) programu do mikroprocesoru .....	- 8 -
<b>4. Ukázkové příklady</b> .....	<b>- 12 -</b>
4.1. Definice (Alias) registrů a konstant .....	- 12 -
4.2. Možnosti zápisu konstant do registrů .....	- 12 -
4.3. Cyklický program .....	- 13 -
4.4. Maskování .....	- 13 -
4.5. Negace.....	- 14 -
4.6. Naplnění a mazání bloku datové paměti RAM.....	- 14 -
4.7. Práce s tabulkou hodnot .....	- 15 -
4.8. definice a zápis na vstupně/výstupní (I/O) porty – Brány.....	- 16 -
4.9. Přerušování .....	- 16 -
4.10. Obdélníkový signál definovaného kmitočtu - Čítač.....	- 17 -
4.11. Relativní volání podprogramu – Blikání diody na portu B.....	- 18 -
4.12. Práce s datovou pamětí EEPROM.....	- 19 -
4.13. UART komunikace (procedury).....	- 20 -
<b>5. Přílohy</b> .....	<b>- 22 -</b>
5.1. Directivy.....	- 22 -
5.2. Stavový registr – SREG.....	- 23 -
5.3. Aritmetické a logické instrukce .....	- 23 -
5.4. Instrukce skoků (větvení programu) .....	- 24 -
5.5. Instrukce přesunu dat.....	- 25 -
5.6. Bitové instrukce (testování bitu) .....	- 26 -
5.7. Menu – AVR Studio.....	- 28 -
5.7.1. Souborové menu .....	- 28 -
5.7.2. Projektové menu .....	- 28 -
5.7.3. Menu kompilace (překlad souboru) .....	- 28 -
5.7.4. Edit menu .....	- 29 -
5.7.5. Tools menu .....	- 29 -
5.7.6. View menu (zobrazení).....	- 30 -
5.7.7. Debug menu (ladění).....	- 31 -
5.7.8. Důležité příkazové zkratky.....	- 32 -
<b>6. Použitá literatura</b> .....	<b>- 33 -</b>

## 1. Programování v assembleru (jazyk symbolických adres)

Proč zrovna assembler a ne jiný jazyk, to je otázka! Assemblery vznikaly už v padesátých letech jako "jazyk symbolických adres" pro tehdejší velké počítače, které se do té doby programovaly na nejnižší možné úrovni ve strojovém kódu s použitím absolutních adres. Je tedy zřejmé, že assembler má velmi blízko k samotnému hardware (elektronice) a algoritmizaci, což je na naší škole hlavní směr.

Assembler se využívá pro programování jednoduchých efektivních aplikací, kde je kritický nedostatek výpočetního času nebo programové paměti. Dále pro řídicí aplikace, ovládače zařízení a výpočetně náročné algoritmy. Pro složitější a rozsáhlejší programy je z důvodu přehlednosti vhodnější použít vyšší programovací jazyky jako například C, C++, Java, Pascal, Perl a další.

Assembler není Case sensitive (nezáleží na psaní velkých a malých písmen).

### Výhody assembleru:

- program může být velmi rychlý, neboť může být maximálně optimalizován pro daný stroj
- výsledný program může být velmi malý
- získáme maximální kontrolu nad počítačem

### Nevýhody assembleru:

- nízká úroveň abstrakce (práce s programovacím jazykem na nižší úrovni) a z toho vyplývající pracnost. Využívá se především pro jednoduché aplikace.
- Špatná přenositelnost – každý procesor (rodina procesorů) má vlastní konkrétní instrukční soubor

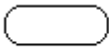





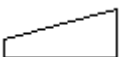

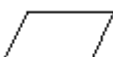


## 2. Vývojové diagramy

Vývojové diagramy znázorňují průběh či stavbu programu – jedná se o grafické znázornění algoritmu. Pokud se programátor rozhodne naprogramovat nějaký program, který má být funkční, pak je dobré nejdříve sestavit vývojový diagram. Celkem dobře se v něm hledají chyby, můžete sledovat postup programu a případně vše poopravit. Velkou výhodou „vývojáků“, jak se jim říká, je jejich univerzálnost. Každý program lze napsat v několika jazycích, ale podle jednoho vývojového

diagramu. Nejtěžší na programování je vymyšlení právě vývojáku (sestavení algoritmu, jak má program pracovat). Pokud máme vývojový diagram hotový, pak ho napsat v daném jazyce je hračka.

## 2.1. Prvky vývojového diagramu

Jak již bylo řečeno, vývojový diagram je grafické znázornění algoritmu dané problematiky. Níže jsou zobrazeny a popsány elementární prvky, která se nejčastěji používají.

	Začátek a konec algoritmu.
	Spojovací čára (ukazuje, směr odkud kam program míří).
	Spojovací bod (když máme více stránek - zřehlednění diagramu).
	Příkaz (načti soubor, ulož soubor, přečti hodnotu).
	Podmínka (pokud ano jdi do kladné větve, pokud ne, jdi do záporné větve).
	Cyklus s určeným počtem opakování (přečti 3x za sebou).
	Vstup zadávaný uživatelem, nebo z čidla (pokud uživatel zadává např. rodné číslo).
	Uložení souboru (v případě, že například návštěvník vloží vzkaz do návštěvní knihy => musíme jeho data uložit).
	Zobrazení souboru (zobrazení již uložených dat – např. pro kontrolu).
	Přechod na další stránku (pro rozsáhlejší vývojové diagramy).
	Nebo - logický součet.

Tabulka 1 – grafické prvky vývojového diagramu

Vývojový diagram směřuje doprava nebo dolů v takovém případě není nutné u spojovací čáry dělat šipku, pokud ovšem směřuje doleva či nahoru, tak se šipka dělat musí.

## 3. AVR Studio - vývojové prostředí (Stručný popis)

### 3.1. Úvod

AVR Studio umožňuje vytváření programových projektů, které shromažďují jeho jednotlivé části - programy (soubory s programovými kódy, informace o projektu, textové soubory atd.). Pokud je program složitější a skládá se z většího počtu dílčích souborů, pak je vytváření projektů velmi účelné a nezbytné. Editor dále umožňuje detailní pohled na projekt a jeho strukturu.

Vývojové prostředí obsahuje textový editor, ve kterém se zapisuje program (programový kód). Editor automaticky rozeznává části kódu (instrukce, komentář, čísla atd.) a pro přehlednost je barevně zvýrazňuje. AVR studio podporuje možnosti ladění kódu, kdy je možné program trasovat a vytvářet záchytné body (breakpoints). Během odlaďování programu lze monitorovat a nastavovat řídicí, datové a vstupně/výstupní registry AVR mikroprocesoru (CPU).

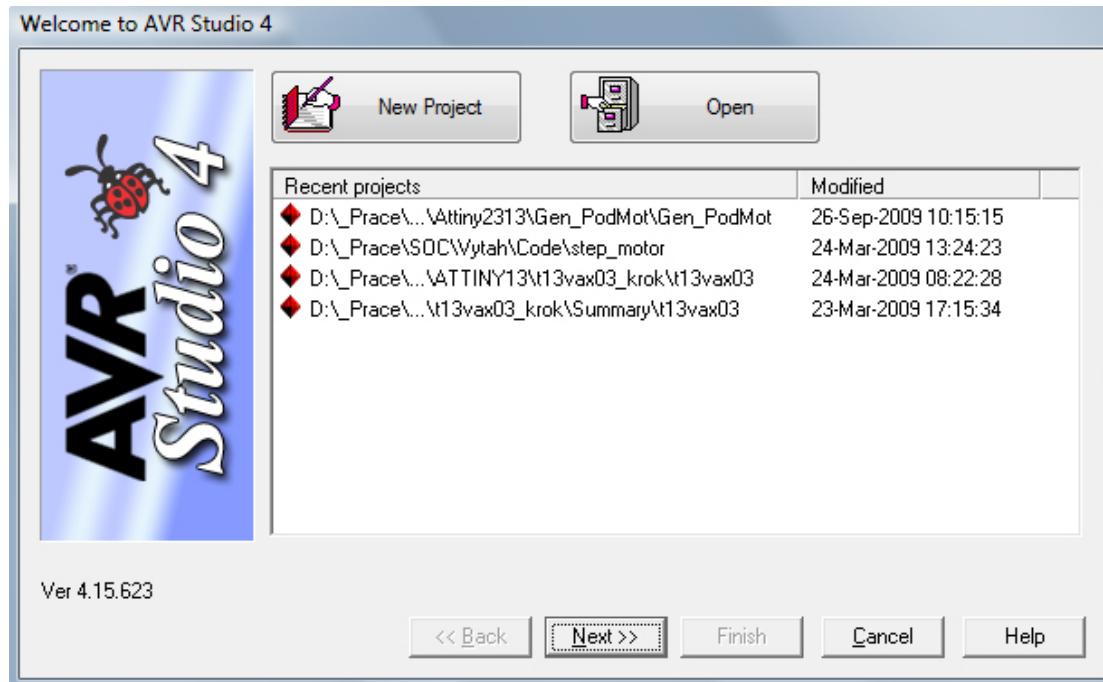
Vývojové prostředí AVR dále obsahuje paletu nástrojů, umístěných v nástrojové liště pro rychlé a snadné ovládání nejpoužívanějších funkcí (překlad programu, spuštění, krokování atd.). Součástí je i možnost přímého programování v aplikaci, které je podporováno MCU AVR (ISP).

### 3.2. Vytvoření nového projektu

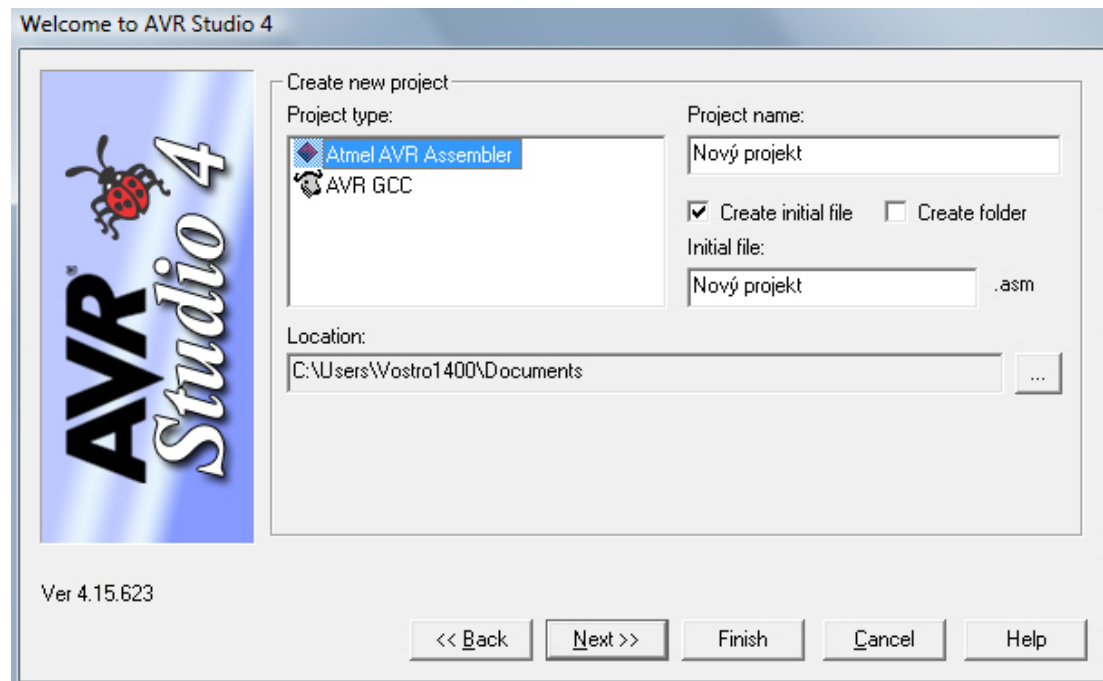
Po spuštění programu AVR studio se otevře dialogové okno pro výběr naposledy použitých projektů (**Open**) nebo pro vytvoření nového projektu (**New Project**). Nový projekt lze také založit z menu **Project - New Project** (viz obrázek 1).

Pokud zvolíte nabídku **New Project**, pak se otevře dialogové okno (viz obrázek 2), ve kterém zvolíme překladač - programovací jazyk projektu (Atmel AVR Assembler – kód assembleru, AVR GCC – kód v jazyce C) a zapíšeme název projektu.

Okno také nabízí možnost vytvoření výchozího souboru s kódem (**create initial file**) a vytvoření složky s projektem (**create folder**) ve zvoleném adresáři (**location**). Je vhodné použít volbu jak vytvoření výchozího souboru tak i vytvoření složky. Po zvolení potvrďte tlačítkem **Next >>**.



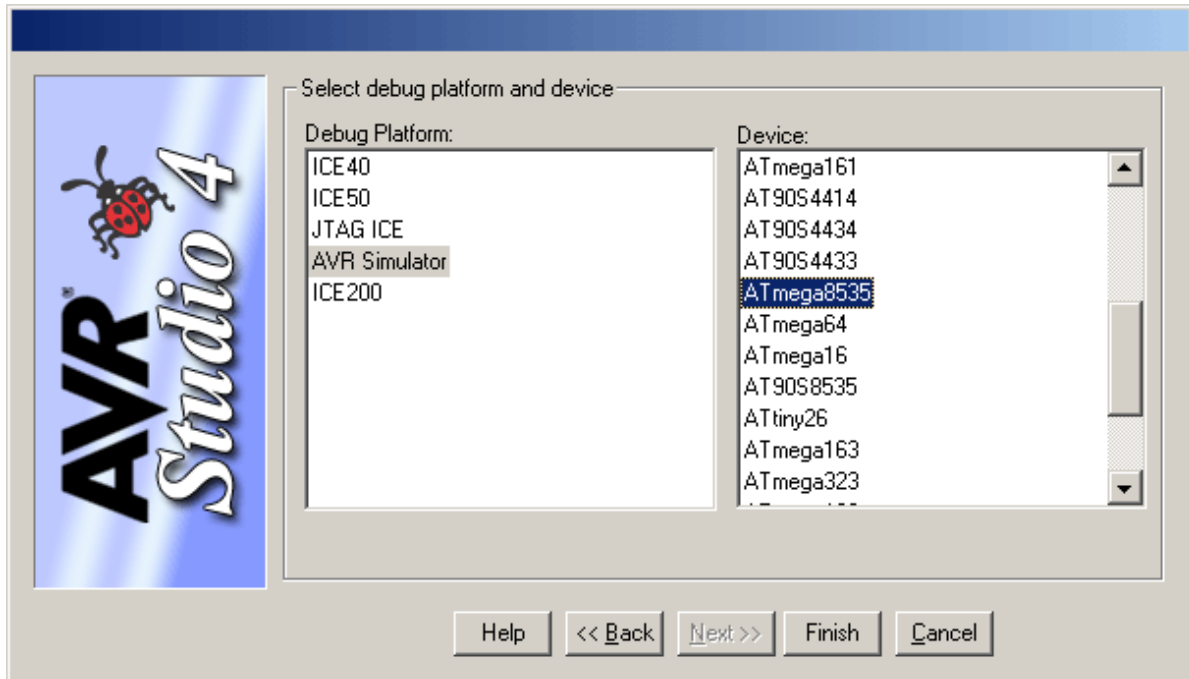
Obrázek 1 - Otevření (vytvoření nového) projektu



Obrázek 2 - Umístění a název projektu a volba programovacího jazyka.

Objeví se další dialogové okno (obrázek 3), kde zvolíme ladící systém (**debug platform**) a typ MCU (**device**). Pokud nebude při ladění využit žádný hardwarový nástroj, zvolte AVR simulátor. Volbu

potvrďte tlačítkem **Finish**. Otevře se pracovní (vývojové) prostředí studia AVR, do kterého můžeme zapisovat kódy instrukcí. Nově vytvořený projekt bude mít koncovku \*.aps.



Obrázek 3 - Výběr simulátoru AVR a typu mikroprocesoru

### 3.3. Otevření a uložení existujícího projektu (\*.aps)

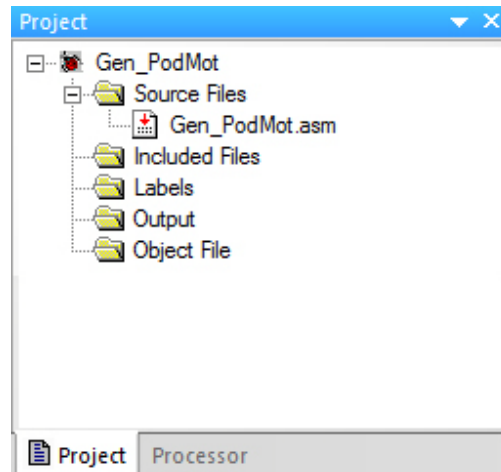
V první řadě lze využít první okno průvodce k vytvoření projektu (obrázek 1). Druhou možnost využijeme v případě, kdy jsme již programovali v pracovním prostředí AVR Studia - zvolíme z horního menu nabídku **Project - Open Project**, poté vybereme soubor s koncovkou \*.aps.

Uložení projektu a vytvořených souborů se provádí pomocí nabídky **Project - Save Project** nebo pomocí nabídky **File - Save All**.

### 3.4. Pracovní (vývojové) prostředí

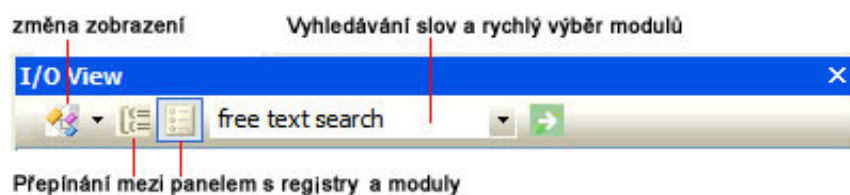
Pracovní prostředí je rozděleno do několika oblastí. V horní části se nachází menu s příkazy a lišta s ikonami. Tato část je podrobně popsána v kapitole 5.7. V levé části se nachází okno (obrázek 4)

s kompletní strukturou aktivního projektu nebo základní parametry procesoru (Program Counter, Stack Pointer, XYZ pointer, SREG, Frekvence a registry pro obecné využití R0 až R31).



Obrázek 4 - Parametry procesoru a struktura aktivního projektu

Na pravém okraji jsou zobrazeny veškeré řídicí a datové I/O registry. Tato část pracovní plochy umožňuje několik módů zobrazení. Nejpodrobnější je mód je „**Module split view**“, který zvolíme kliknutím na ikonu „**změna zobrazení**“ viz obrázek 5. Nyní mohou být zobrazeny jednotlivé moduly s registry a zároveň jejich detailní přehled. Zobrazené moduly s registry mohou například být – modul čítače/časovače, moduly portů, modul CPU, modul paměti EEPROM, modul watch dog a další.



Obrázek 5 - Zobrazení I/O portů procesoru

Detailním pohledem na jednotlivé registry získáme informaci o názvu a adrese registru, jeho obsahu v binární a hexadecimální podobě.

V dolní části obrazovky se nachází informační panel s několika záložkami, které obsahují zprávy o překladu program (správnost překladu, chyby v kódu), o použitých souborech a zařízeních (typ procesoru, typ programátoru nebo simulátoru) a zprávy o breakpointech (zarážky).

Na závěr poslední okno umístěné uprostřed obrazovky, slouží k zápisu samotných instrukcí v daném programovacím jazyce. Okno obsahuje záložky s otevřenými soubory. Instrukce, parametry a text jsou barevně odlišeny, což přispívá k dobré orientaci v programu. Žlutá šipka na levém okraji určuje, na které instrukci se právě nacházíme v případě krokování přeloženého kódu.

### 3.5. Kompilace programového kódu

Kompilací programu napsaného v assembleru docílíme překladu programu do strojového kódu (binární hodnoty 1,0 nebo hexadecimální čísla). Výsledek kompilovaného programu se uloží do souboru s koncovkou \*.HEX a \*.OBJ (podrobněji viz tabulka 2).

Kompilace (překlad) se spustí pomocí menu **Project - Build** (nebo klávesou **F7**). Před samotnou kompilací se programový kód musí uložit, aby kompilátor mohl zpracovávat aktuální verzi. Jednotlivé výstupní informace o průběhu kompilace, včetně chyb se vypisují do okna „Output“. Při bezchybné kompilaci překladač vypíše do výstupního okna hlášení „Assembly complete with no errors“ a v adresáři projektu se vytvoří již zmiňovaný soubor s příponou \*.HEX (\*.OBJ) a také soubor s příponou \*.EEP, který obsahuje data EEPROM paměti.

Extended Intel hex	.hex	Tento formát běžně využívají firmy pro vývoj a testování aplikací. Formát neobsahuje informace pro odladění, a proto není vhodný pro odladění programu přímo v mikro počítačích. Soubor s touto koncovkou obsahuje pouze naprogramovaná data (instrukce) přeložená do strojového kódu.
AVR Assembler formát	.obj	AVR assembler souborový formát obsahuje ladící informace a informace potřebná pro krokování programu přímo za běhu mikroprocesoru. Tento formát je interní záležitost mikroprocesorů AVR.


Tabulka 2 – koncovky souborů obsahující přeložený strojový kód

### 3.6. Spuštění (ladění) programu

Programový kód je možné spustit a ladit přímo v prostředí AVR Studia. Vše se provede volbou z menu **Project - Build and Run** (nebo kombinací kláves **Ctrl+F7**). Pokud byl při vytvoření projektu zvolen jako ladící nástroj AVR Simulator, bude ladění probíhat pouze v prostředí AVR Studia softwarově. Menu **Debug** obsahuje jednotlivé příkazy ladění: **Step into**, **Step Over**, **Run to cursor** a další. Jejich detailní popis je možné najít v kapitole 5.7.7. Ladění je možné ukončit pomocí příkazu z menu **Debug - Stop Debugging** (nebo kombinací kláves **Ctrl+Alt+F5**).

### 3.7. Nahrávání (Load) programu do mikroprocesoru

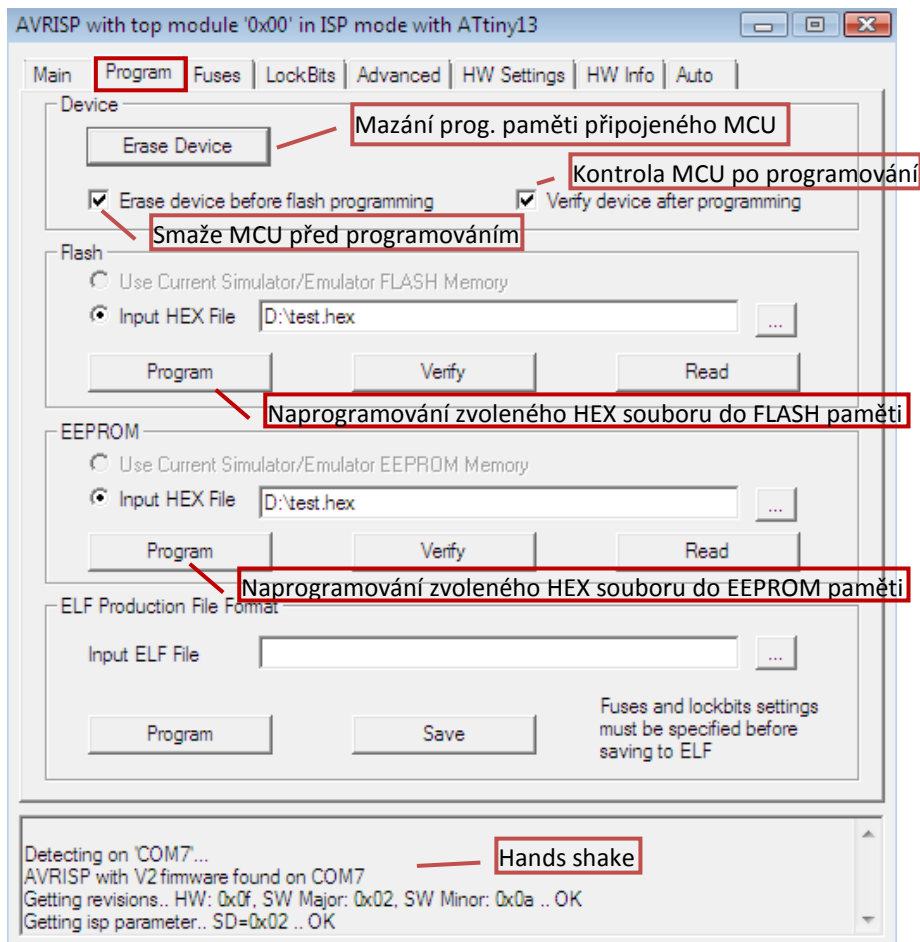
Program se zapisuje do MCU do paměti FLASH (EEPROM). Programátor, kterým programujeme MCU využívá rozhraní ISP. ISP – In System Programming je dnes běžná věc, která šetří čas při vývoji, protože díky tomu není nutné stále přenášet MPU mezi programátorem a patičkou umístěnou v aplikaci. Jedná se o sériové programování s procesorem umístěným přímo v aplikaci.

Dialogové okno pro připojení spustíme volbou z menu **Tools -> Program AVR -> Connect** nebo kliknutím na ikonu Connect: 

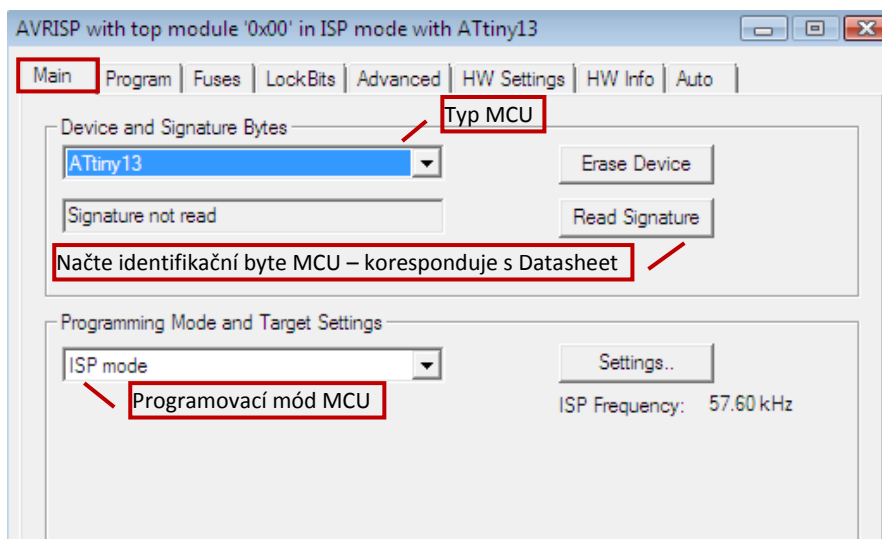
Dále vybereme typ programátoru – **STK 500** a správný sériový COM port. Přenosová rychlost musí být nastavena na 115200Bd, pak klikneme na tlačítko Connect. V této chvíli se programátor připojí k MCU.

Správné připojení indikuje nové okno AVRISP – v spodní části okna proběhne hands shake. Okno AVRISP obsahuje osm záložek, které jsou dále stručně popsány.

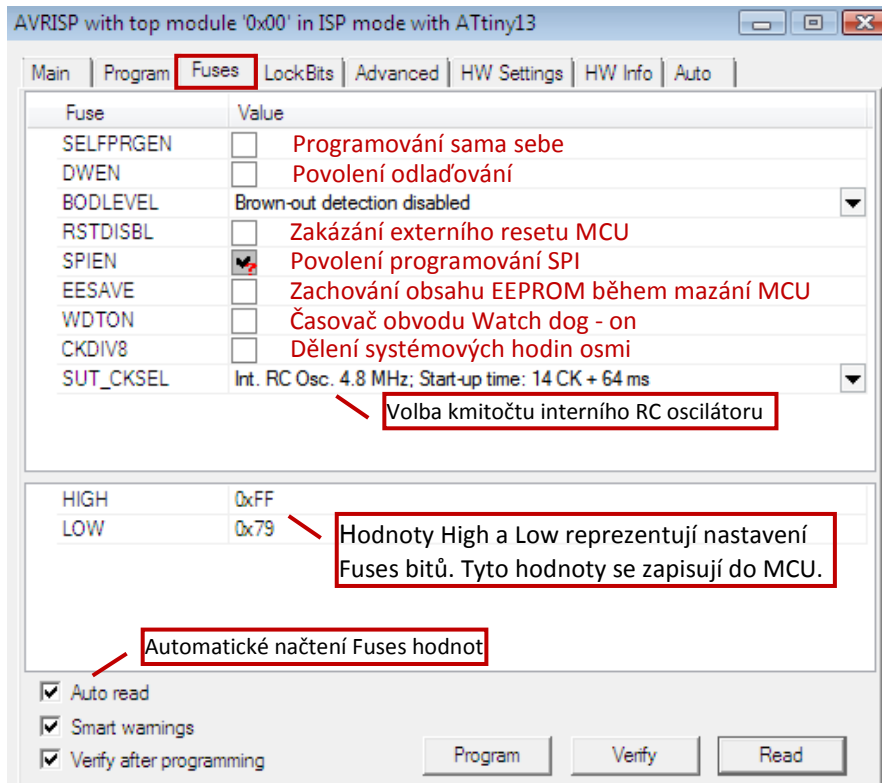
Pokud chceme naprogramovat MCU s využitím ISP, musíme počítat s tím, že pin RESET nebude dále použitelný. Tento pin detekuje resetovací impuls od sériového programátoru, na jehož základě programujeme MCU. Pokud bychom nastavili RESET pin do stavu běžného využití, nebudeme moci ISP programování využít. Délka resetovacího impulsu je aktivní v logické nule a trvá zhruba 20ns. Doporučuje se po aktivním resetovacím impulsu přistupovat k programovacím pinům MCU zhruba po 500ms.



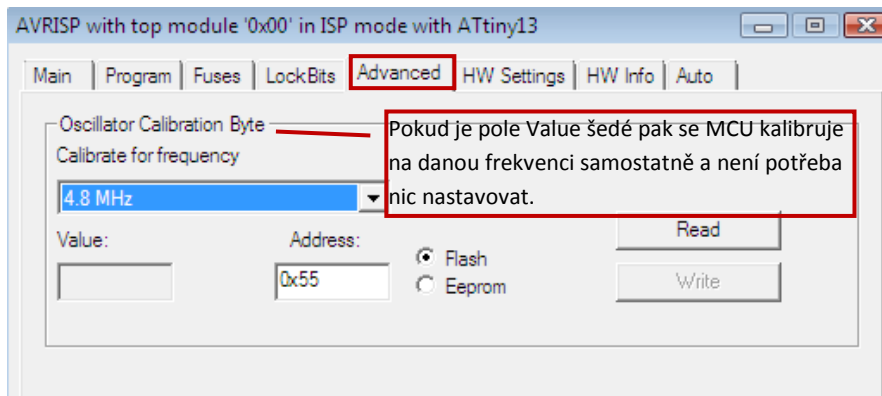
Obrázek 6 – Dialogové okno ISP - záložka „Program“



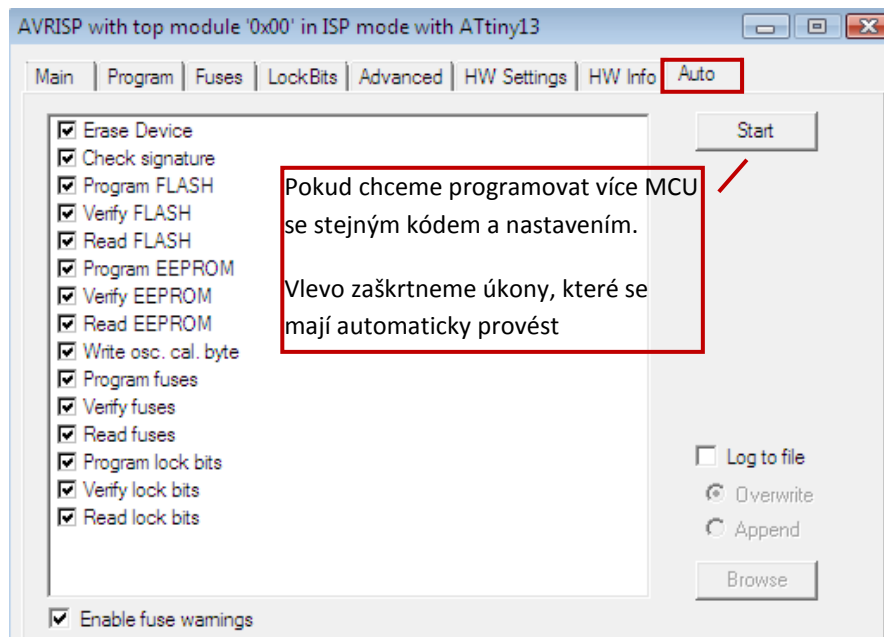
Obrázek 7 - Dialogové okno ISP – záložka „Main“



Obrázek 8 - Dialogové okno ISP – záložka „Fuse“



Obrázek 9 - Dialogové okno ISP – záložka „Advanced“



Obrázek 10 - Dialogové okno ISP – záložka „Auto“

## 4. Ukázkové příklady

### 4.1. Definice (Alias) registrů a konstant

Pro přehlednost programování je vhodné jednotlivé registry (R0 – R31) a funkce pojmenovávat dle jejich účelu. Program je pak srozumitelnější. Pro konstanty se kromě toho získá možnost jednoduše změnit její hodnotu, i když je použita v programu na více místech.

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
.DEF POMOC_REG = R16            ;REGISTR R16 MUŽEM DÁLE POUŽÍVAT S OZNAČENÍM „POMOC_REG“
.DEF KONTRO_REG = R17          ;REGISTR R17 MUŽEM DÁLE POUŽÍVAT S OZNAČENÍM „KONTROL_REG“
.DEF HIGH_REG = R18            ;REGISTR R18 MUŽEM DÁLE POUŽÍVAT S OZNAČENÍM „HIGH_REG“

.EQU N_CYKLU = 12              ;NÁZVU „N_CYKLU“ BUDE PŘIŘAZENA KONSTANTA (HODNOTA) 12
.EQU MAXIMUM = 255             ;NÁZVU „MAXIMUM“ BUDE PŘIŘAZENA KONSTANTA (HODNOTA) 255
.EQU MAXIMUM = 2*30            ;NÁZVU „MAXIMUM“ BUDE PŘIŘAZENA KONSTANTA (HODNOTA) 2*30 = 60

.SET ADRESB = 0x18             ;FUNGUJE PODOBNĚ JAKO .EQU, ALE LEZ HODNOTU V PROGRAMU MĚNIT
.SET PINB = ADRESB+2          ;PROPOJENÍ S PŘEDCHOZÍ INSTRUKCÍ
```

### 4.2. Možnosti zápisu konstant do registrů

Při programování pracujeme podle uživatelských potřeb s různými soustavami a typy konstant. Do registru jdou zapsat i konkrétní bity řídicích registrů (nastavení čítače, přerušení, atd.).

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR

.SET KONSTANTA = 12             ;ALIAS - DEFINICE KONSTANTY – VIZ PŘÍKLAD 4.1
LDI R16, KONSTANTA             ;ZÁPIS KONSTANTY DO REGISTRU R16
.SET KONSTANTA=KONSTANTA+2     ;ZMĚNA HODNOTY KONSTANTY
LDI R17, KONSTANTA             ;ZÁPIS KONSTANTY DO REGISTRU R16

LDI R16, 12                    ;ZÁPIS KONSTANTY DO REGISTRU R16 V DEKADICKÉM TVARU
LDI R16, 0xFA                  ;ZÁPIS KONSTANTY DO REGISTRU R16 V HEXADECIMÁLNÍM TVARU
LDI R16, 0b01010101           ;ZÁPIS KONSTANTY DO REGISTRU R16 V BINÁRNÍM TVARU
LDI R16, '2'                   ;ZÁPIS KONSTANTY DO REGISTRU R16 V ASCII TVARU
LDI R16, 'H'                   ;ZÁPIS KONSTANTY DO REGISTRU R16 V ASCII TVARU
LDI R16,(1<<CS00);|(1<<CS01) ;ZAPSÁNÍ DAT NA KONKRÉTNÍ BITY KONKRÉTNÍCH ŘÍDÍČÍHO REGISTRU
```

### 4.3. Cyklický program

**N – násobný cyklus** – cyklus počítačového programu se opakuje pouze N krát, pak pokračuje dál.

Využívá podmíněný skok.

```

;N-NÁSOBNÝ CYKLUS
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
LDI R16, 12                       ;ZÁPIS KONSTANTY DO REGISTRU R16 – NÁSOBNOST CYKLU
LOOP:                              ;ZAČÁTEK NEKONEČNÉ SMYČKY (NÁVĚŠTÍ) – NÁSLEDUJE TĚLO CYKLU
    INC R17                        ;R17=R17+1 – V KAŽDÉM CYKLU SE HODNOTA REGISTRU ZVĚTŠÍ O 1
    CP R17,R16                     ;KONTROLA JESTLI JIŽ PROBĚHLO N CYKLŮ
    BRNE LOOP                       ;PODMÍNĚNÝ SKOK – REAGUJE NA PŘEDCHOZÍ POROVNÁNÍ DVOU REGISTRŮ

```

**Nekonečný cyklus** - je cyklus počítačového programu, který se neustále do nekonečna opakuje.

Využívá nepodmíněný skok. Nekonečný cyklus lze vytvořit i pomocí podmíněného skoku, přičemž nikdy nedojde ke splnění podmínky. Nekonečný cyklus se v programu často objevuje jako chyba.

```

;NEKONEČNÝ CYKLUS
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
LOOP:                              ;ZAČÁTEK NEKONEČNÉ SMYČKY - NÁVĚŠTÍ
    NOP                             ;TĚLO NEKONEČNÉ SMYČKY (INSTRUKCE NIC NEDĚLÁ)
    RJMP LOOP                       ;NEPODMÍNĚNÝ SKOK NA NÁVĚŠTÍ

```

### 4.4. Maskování

V praxi se vám může stát, že potřebujete detekovat jedničku nebo nulu na jednom nebo více bitech z bajtu (detekce rozsvíceného světla na výstupním portu). K tomu se využívá maskování binárního čísla = logický součin AND registru s maskou (konstantou). Masky obsahuje na bitech, které chceme v registru detekovat hodnoty log „1“, ostatní bity jsou nulové. Po vykonání logického součinu, zůstávají v registru nezměněné bity pouze na těch místech, kde byla v masce jednička. Pro přehlednost je vhodné masku zapisovat v binárním tvaru.

```

.INCLUDE "TN13DEF.INC"
.DEF DETEKOVANY_REG = R18         ;REGISTR R18 MUŽEM DÁLE POUŽÍVAT S OZNAČENÍM „DETEKOVANY_REG“
.EQU MASKA = 0b00001000         ;NÁZVU „MASKA“ BUDE PŘÍRAZENA KONSTANTA (BINÁRNÍ HODNOTA) 00001000

LDI DETEKOVANY_REG, 0b01011010   ;NAČTENÍ KONSTANTY DO REGISTRU, KTERÝ CHCEME MASKOVAT
ANDI DETEKOVANY_REG, MASKA       ;LOGICKÝ SOUČIN REGISTRU R16 A MASKY

```

## 4.5. Negace

Instrukce, která by negovala celý byte registru R0 – R31 u procesorů ATMEL AVR s architekturou RISC neexistuje. Proto je nutné vytvořit si vlastní funkci, která bude negaci vykonávat. Řešení je několik, nejjednodušší se jeví využití funkce XOR a masky obsahující samé logické jedničky.

A	b	$Y = a \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabulka 3 – Pravdivostní tabulka logické funkce XOR

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
.DEF NEG_REG = R16               ;„NEG_REG“ – REGISTR, JEHOŽ HODNOTU BUDEME NEGOVAT
.DEF MASKA = R17                 ;„MASKA“ REGISTR, DO KTERÉHO BUDE ULOŽENA MASKA

LDI NEG_REG, 0b01011010         ;NAČTENÍ KONSTANTY DO REGISTRU, KTERÝ CHCEME NEGOVAT
LDI MASKA, 0b11111111          ;NAČTENÍ MASKY DO REGISTRU (KONSTANTY)
EOR NEG_REG, MASKA              ;LOGICKÁ FUNKCE XOR – VÝSLEDEK SE ZAPÍŠE ZPĚT DO NEG_REG
```

## 4.6. Naplnění a mazání bloku datové paměti RAM

Interní datová paměť SRAM je organizována po 8 bitech od adresy 0x0060h do adresy, která je dána typem a velikostí paměti jednotlivých mikroprocesorů (Attiny 13 – 0x009F, Attiny 2313 – 0x00DF). Přístup a zápis hodnot (dat) do paměti se provádí nepřímo pomocí registru Z (R30,R31). Nulování paměti se provádí zápisem hodnoty 0x00 do paměti.

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
.EQU KONEC_ADR = 0x8C           ;KONEČNÁ ADRESA, PO KTEROU BUDEME ZAPISOVAT DO DATOVÉ PAMĚTI
.DEF DATA = R17                 ;REGISTR OBSAHUJÍCÍ ZAPISOVANÁ DATA NA DANOU ADRESU

LDI R30,0x60                     ;NASTAVENÍ POČÁTEČNÍ ADRESY DO REGISTRU Z (DOLNÍCH 8 BITŮ)
CLR R31                           ;VYNULOVÁNÍ HORNÍCH 8 BITŮ 16 BITOVÉ ADRESY
;ZAČÁTEK CYKLU PRO ZÁPIS DO PAMĚTI DAT (INTERNÍ SRAM)
CYKL:
LDI DATA,0xAA                   ;DATA KTERÁ SE BUDOU ZAPISOVAT DO PAMĚTI DAT
ST Z+,DATA                       ;ZÁPIS DAT NA ADRESU UMÍSTĚNOU V REGISTRU Z
```

```

CPI R30,KONEC_ADR           ;KONTROLA ZDA JE NAPLNĚNÝ CELÝ ZVOLENÝ ROZSAH PAMĚTI
BRNE CYKL                   ;POKUD SE ZAPÍŠOU DATA NA POSLEDNÍ DEFINOVANOU ADRESU, TAK CYKL KONČÍ

NEK_LOOP:                   ;NEKONEČNÁ SMYČKA
RJMP NEK_LOOP

```

## 4.7. Práce s tabulkou hodnot

Tabulku je vhodné využít například pro vygenerování atypických signálů, nebo pro vytváření krokových tabulek pro ovládání krokových motorů a další aplikace. Tabulka je jediný případ, kdy se data ukládají do paměti programu a nikoliv do paměti dat. Paměť programu je adresovaná po 16 bitech, proto je potřeba s tímto faktem počítat při adresaci tabulky. K vyčítání dat z adres paměti programu se využívá 16 bitový registr Z (R30,R31).

```

.INCLUDE "TN13DEF.INC"      ;TINY13 DEFINIČNÍ SOUBOR
.DEF UKAZATEL = R16         ;UKAZATEL – ZAJIŠŤUJE POHYB V TABULCE HODNOT
.DEF NULA = R17            ; NULA – SLOUŽÍ JAKO REGISTR PRO NULOVÁNÍ PŘÍPADNÉHO OFFSETU
.DEF DATA_TAB = R18       ;DATA_REG – DO TOHOTO REGISTRU SE VYPISUJI HODNOTY Z TABULKY
.EQU HODNOT_TAB = 16      ;UDÁVÁ POČET HODNOT V TABULCE (URČUJE KONEC TABULKY)

CLEAR:
CLR UKAZATEL               ;VYNULOVÁNÍ UKAZATELE - ČTENÍ HODNOT Z TABULKY OD ZAČÁTKU
CLR NULA                   ;NULOVÝ REGISTR

;CYKLIKÝ VÝPIS DAT Z TABULKY DO REGISTRU „DATA_TAB“
CYKL:
LDI R30,LOW(ADR_TAB)       ; DOLNÍCH 8 BITŮ REGISTRU Z (ADRESA DAT Z TABULKY V PAMĚTI PROGRAMU)
LDI R31,HIGH(ADR_TAB)     ;HORNÍCH 8 BITŮ REGISTRU Z
ADD R30,UKAZATEL          ;POČÁTEČNÍ ADRESA TABULKY SE ZVĚTŠÍ O UKAZATEL
ADC R31,NULA              ;VYMAZÁNÍ OFFSETU
LPM DATA_TAB, Z          ;ZÁPIS DAT Z ADRESY ULOŽENÉ V Z DO REGISTRU DATA_TAB
INC UKAZATEL              ;V KAŽDÉM CYKLU UKAZATEL UKAZUJE NA NÁSLEDUJÍCÍ HODNOTU V TABULCE
CPI UKAZATEL,HODNOT_TAB   ;TESTOVÁNÍ KONCE TABULKY (ZDA UKAZATEL UKAZUJE NA POSLEDNÍ HODNOTU)
BREQ CLEAR               ;POKUD UKAZATEL JE ROVEN POČTU HODNOT V TABULCE PAK HO NULUJEME
RJMP CYKL                 ;KONEC CYKLU

TAB:
.DB 0, 1
.DB 2, 3
.DB 4, 5
.DB 6, 7
.DB 8, 9
.DB 10, 11
.DB 12, 13
.DB 14,15
.EQU ADR_TAB = 2*TAB      ;ADRESA TABULKY NÁSOBENÁ DVĚMA – 16. TI BITOVÁ ARCHITEKTURA PAMĚTI

```

## 4.8. definice a zápis na vstupně/výstupní (I/O) porty – Brány

Mikroprocesory AVR jsou dle typu vybaveny 8bitovými obousměrnými bránami, které se označují PORT A, PORT B atd. Brány jsou mapovány třemi adresami: vstupní vývody (PINx), výstupní registr (PORTx), směrový registr (DDRx).

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
.DEF POMOC_REG = R16           ;POMOCNÝ REGISTR
;INICIALIZACE PORTU B
LDI POMOC_REG,0b00000001      ;INICIALIZACE PORTU B - VÝSTUPNÍ PIN PBO
OUT DDRB,POMOC_REG           ;DDR - URČUJE FUNKCI I/O PINŮ (VSTUPNÍ/VÝSTUPNÍ)
;BITOVÉ OPERACE S PORTEM B
SBIS PINB, 0                  ;DETEKCE PINU – KDYŽ JE PBO=„1“ BUDE NÁSLEDUJÍCÍ INSTRUKCE PŘESKOČENA
CBI PORTB, 1                  ;NASTAVENÍ PINU PB1 DO STAVU LOG 1
SBI PORTB, 2                  ;NASTAVENÍ PINU PB2 DO STAVU LOG 0
LDI R16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0) ;.....
LDI R17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);.....
;BAJTOVÉ OPERACE S PORTEM B
OUT PORTB, POMOC_REG         ;ZÁPIS CELÉHO BAJTU NA PORT B
IN POMOC_REG, PINB           ;PŘÍSTUP K FYZICKÉ HODNOTĚ PORTU B, NEJEDNÁ SE O REGISTR ALE O ADRESU
```

Zde je třeba si uvědomit, že po každém ukončení programování přes ISP, nastavuje RESET programovací SW do neaktivního režimu, čímž v podstatě okamžitě spouští program, který byl do CPU nahrán. Doporučuje se nepřistupovat na piny ISP z P1 prvních cca 500 ms po resetu. Pokud tedy například nastavujete po RESETu všechny I/O piny do logické 0 použijte před nastavením ISP pinů (P1.5 - P1.7) čekací smyčku.

## 4.9. Přerušení

Pokud přijde žádost o přerušení od některého zdroje přerušení, kterým mikroprocesor disponuje, pak se přeruší běh hlavního programu, vykoná se poslední instrukce a program skočí na hardware adresu tohoto zdroje. Zde najde nepodmíněný skok na uživatelsky nadefinované návěští. Vykoná se podprogram a instrukce RETI program vrátí zpět do hlavního programu, tam kde jsme ho opustili.

```
.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
;VEKTORY PŘERUŠENÍ - DEFINOVÁNÍ INSTRUKCÍ (SKOKŮ) NA HARDWARE ADRESÁCH JEDNOTLIVÝCH ZDROJŮ PŘERUŠENÍ
.ORG 0x0000    RJMP RESET         ;RESET (EXTERNÍ, OD NAPÁJENÍ, WATCHDOG)
```

```

.ORG 0x0001   RJMP EXT_INT           ;EXTERNÍ ZDROJ PŘERUŠENÍ
.ORG 0x0002   RJMP PCINT0         ;Č/Č 2, PŘETEČENÍ Č/Č
.ORG 0x0003   RJMP TIM0_OV        ; Č/Č 0, PŘETEČENÍ Č/Č
.ORG 0x0004   RJMP EE_RDY         ;OBSLUHA PAMĚTI EEPROM (READY)
.ORG 0x0005   RJMP AN_COMP        ;ANALGOVÝ KOMPARÁTOR
.ORG 0x0006   RJMP TIM0_COMPA     ;Č/Č 1 POROVNÁNÍ HODNOT – VÝSTUP A
.ORG 0x0007   RJMP TIM0_COMPB    ;Č/Č 0 POROVNÁNÍ HODNOT – VÝSTUP B
.ORG 0x0008   RJMP WATCHDOG      ;PŘETEČENÍ OBVODU WATCHDOG
.ORG 0x0009   RJMP ADC            ; A/D PŘEVOD DOKONČEN

WATCHDOG:    ;NÁVĚŠTÍ PRO OBSLUHU PŘERUŠENÍ BLOKU WATCHDOG
    RETI      ;NÁVRAT Z PODPROGRAMU ZPĚT DO HLAVNÍHO PROGRAMU

```

#### 4.10. Obdélníkový signál definovaného kmitočtu - Čítač

Pokud je frekvence interního oscilátoru ( $f_{osc}$ ) rovna například 1MHz, pak z toho plyne, že jeden cyklus (Cy) bude trvat 1 $\mu$ s (0,25 $\mu$ s). Číslo (N), které uložíme do registru OCR0A bude určovat počet cyklů, které proběhnou mezi jednotlivými přerušeními a tedy i čas ( $T_{přeruš}$ ).

$$Cy = T_{osc} = 1/f_{osc} \quad [s] \quad \text{Rovnice 1}$$

$$T_{přeruš} = Cy \cdot N \quad [s] \quad \text{Rovnice 2}$$

Registr OCR0A se porovnává s registrem TCNT0 běžícího čítače a v případě jejich schody dojde k vyvolání přerušení a vykonání podprogramu, který dané přerušení obslouží. Hlavním programem je v našem případě nekonečná smyčka, která nic nevykonává. Podprogram provádí negaci bitu PB0 při každém přerušení hlavního programu.

```

.INCLUDE "TN13DEF.INC"           ;TINY13 DEFINIČNÍ SOUBOR
.DEF POMOC_REG = R16             ;POMOCNÝ REGISTR
.EQU CASO_SETUP = 120           ;NASTAVENÍ MASKY ČASU MEZI PŘERUŠENÍMI
.CSEG                             ;PROGRAMOVÝ SEGMENT
.ORG 0x0000   RJMP RESET         ;ULOŽENÍ HLAVNÍHO PROGRAMU AŽ ZA VŠECHNY VEKTORY PŘERUŠENÍ
.ORG 0x0006   RJMP PRERUSENI_CAS ;DEFINICE VEKTORU PŘERUŠENÍ PRO ČASOVAČ 0

RESET:
;NASTAVENÍ UKAZATELE (SP)
    LDI R16,LOW(RAMEND)         ;NAČTENÍ POSLEDNÍ ADRESY Z DATOVÉ PAMĚTI
    OUT SPL,R16                 ;NASTAVENÍ UKAZATELE NA ZÁSOBNÍK (STACK POINTER)

```

```

;INICIALIZACE PORTŮ
    LDI POMOC_REG,0b00000001 ;INICIALIZACE PORTU PORTU B - VÝSTUPNÍ PIN PBO
    OUT DDRB,POMOC_REG ;DDR - URČUJE FUNKCI I/O PINŮ (VSTUPNÍ/VÝSTUPNÍ)

;INICIALIZACE ČASOVAČE
    LDI POMOC_REG,(2<<WGM00) ;NASTAVENÍ MÓDU ČÍTAČE/ČASOVAČE - WGM=2...CTC MODE
    OUT TCCR0A, POMOC_REG ;TCCR0A - MÓD ČÍTAČE/ČASOVAČE, NASTAVENÍ I/O PINU OCA0
    LDI POMOC_REG, (1<<CS00) ;NASTAVENÍ ČASOVÉ ZÁKLADNY - CS0=0...F=1MHZ
    OUT TCCR0B, POMOC_REG ;TCCR0B - CS00,CS01,CS02 DEFINUJÍ ČASOVOU ZÁKLADNU
    LDI POMOC_REG, CAS0_SETUP ;HODNOTA KTERÁ DEFINUJE PERIODU PŘERUŠENÍ
    OUT OCR0A, POMOC_REG ;OCR0A SE POROVNÁVÁ S TCNT0 – PŘI ROVNOSTI DOJDE K PŘERUŠENÍ
    IN POMOC_REG, TIMSKO ;ULOŽENÍ PŘEDEFINOVANÝCH POVOLENÍ PŘERUŠENÍ ČÍTAČE 0
    ORI POMOC_REG, (1<<OCIE0A) ;NASTAVENÍ POVOLENÍ PŘERUŠENÍ PRO ČÍTAČ 0
    OUT TIMSKO, POMOC_REG ;POVOLENÍ PŘERUŠENÍ: 1)SHODA A 2)PŘETEČENÍ 3)SHODA B
    SEI ;GLOBÁLNÍ PŘERUŠENÍ

;POČÁTEČNÍ HODNOTY
    CLR POMOC_REG ;VYMAZÁNÍ PORTU B
    OUT PORTB,POMOC_REG

;HLAVNÍ PROGRAM
NEK_LOOP: ;NEKONEČNÁ SMYČKA
    RJMP NEK_LOOP

;PODPORGRAM
PRERUSENI_CAS:
    SBIS PINB,0 ; POKUD JE PIN PBO V NULE BUDEME HO NASTAVOVAT DO LOG 1
    RJMP NASTAV ;NASTAVENÍ PINU PBO DO STAVU LOG 1
    SBIC PINB,0 ; POKUD JE PIN PBO NASTAVEN, BUDEME HO NULOVAT
    CBI PORTB,0 ;NASTAVENÍ "LOG 0" NA PINU PBO
    RJMP KONEC ;OPUŠTĚNÍ PODPROGRAMU

NASTAV:
    SBI PORTB,0 ;NASTAVENÍ "LOG 1" NA PINU PBO

KONEC:
    RETI ;NÁVRAT Z PŘERUŠENÍ

```

#### 4.11. Relativní volání podprogramu – Blikání diody na portu B

Pokud bychom v napsaném kódu opakovali několikrát stejnou skupinu instrukcí, například blok zpoždění a další, pak je vhodné využít volání podprogramu. Ten bude tuto skupinu stejných instrukcí při každém zavolání vykonávat. Relativní volání na danou adresu lze vykonávat v rozsahu 2K (jednotka kilo) na obě strany.

```

.INCLUDE "TN13DEF.INC" ;TINY13 DEFINIČNÍ SOUBOR
.DEF POMOC_REG = R16 ;POMOCNÝ REGISTR
.DEF MASKA = R17 ;REGISTR, KTERÝ BUDE OBSAHOVAT MASKU OVLÁDAJÍCÍ VÝSTUPNÍ PORT B
.EQU N_TAKTU = 255 ;REGISTR URČUJÍCÍ DÉLKU ZPOŽDĚNÍ SMYČKY

```

```
;POČÁTEČNÍ HODNOTY A NASTAVENÍ PORTU B
```

```
LDI MASKA, 0b00000001
LDI POMOC_REG, 0b00111111
OUT DDRB, POMOC_REG
```

```
;HLAVNÍ PROGRAM PROVÁDÍ POSUN JEDNIČKY PO JEDNOTLIVÝCH PINECH VÝSTUPNÍHO PORTU B (MÁ POUZE 6 PINŮ)
```

```
HL_PROG:
```

```
RCALL ZPOZDENI ;RELATIVNÍ VOLÁNÍ PODPROGRAMU
OUT PORTB, MASKA ;ZÁPIS HODNOTY NA VÝSTUPNÍ PORT B
ROL MASKA ;POSUNUTÍ LOGICKÉ JEDNIČKY NA DALŠÍ PIN PORTU B
SBRC MASKA, 6 ;PORT MÁ POUZE 6 PINU – POKUD JE JEDNIČKA NA 6 PINU, PAK SE PŘEPÍŠE MASKA
LDI MASKA, 0b00000001 ;NOVÁ HODNOTA PRO MASKU – JEDNIČKA JDE ZNOVU OD ZAČÁTKU
RJMP HL_PROG ;NEPODMÍNĚNÝ SKOK NA ZAČÁTEK CYKLU
```

```
;PODPROGRAM
```

```
ZPOZDENI:
```

```
LDI POMOC_REG, N_TAKTU;PŘI ZAVOLÁNÍ PODPROGRAMU SE OBNOVÍ HODNOTA URČUJÍCÍ DÉLKU ZPOŽDĚNÍ
LOOP: ;ZAČÁTEK SMYČKY (NÁVĚŠTÍ) – NÁSLEDUJE TĚLO CYKLU
DEC POMOC_REG ;DEKREMENTACE REGISTRU S KAŽDÝM PRŮCHODEM SMYČKOU
TST POMOC_REG ;TEST ZDA JE POMOCNÝ REGISTR NULOVÝ – NASTAVÍ SE PŘÍZNAK Z (ZERO - SREG)
BRNE LOOP ;PODMÍNĚNÝ SKOK SKÁČE, POKUD JE Z=0. TO SIGNALIZUJE VYNULOVANÍ REGISTRU
RET
```

## 4.12. Práce s datovou pamětí EEPROM

```
.INCLUDE "TN13DEF.INC" ;TINY13 DEFINIČNÍ SOUBOR
.DEF POMOC_REG = R16
.DEF ADR_REG = R17 ;REGISTR PRO UCHOVÁNÍ ADRESY PAMĚTI EEPROM
.DEF DATA_REG = R18 ;OBSAHUJE DATA PRO EEPROM
;HLAVNÍ PROGRAM
LDI DATA_REG, 0b01010101 ;DATA URČENÁ PRO ZÁPIS DO PAMĚTI EEPROM
RCALL EEPROM_WRITE: ;VOLÁNÍ PODPROGRAMU - ZÁPIS DO PAMĚTI EEPROM
RCALL EEPROM_READ: ;VOLÁNÍ PODPROGRAMU – ČTENÍ Z PAMĚTI EEPROM

EEPROM_WRITE:
SBIC EECR, EEPE ;ČEKÁNÍ DOKUD NENÍ PŘEDCHOZÍ ZÁPIS KOMPLETNÍ – KONTROLA BITU EEPE
RJMP EEPROM_WRITE ;ČEKÁNÍ DOKUD NENÍ PŘEDCHOZÍ ZÁPIS KOMPLETNÍ
LDI POMOC_REG, (0<<EEPROM1)|(0<<EEPROM0) ;NASTAVENÍ PROGRAMOVACÍHO MÓDU - ZÁPIS
OUT EECR, POMOC_REG ;ZÁPIS PŘEDNASTAVENÝCH HODNOT Z R16 DO ŘÍDÍCÍHO REGISTRU
OUT EEARL, ADR_REG ;ZAPSÁNÍ ADRESY DO ADRESOVÉHO REGISTRU EEPROM
OUT EEDR, DATA_REG ;ZÁPIS DAT DO DATOVÉHO REGISTRU EEPROM
SBI EECR, EEMPE ;KONEČNÉ POVOLENÍ ZÁPISU DO PAMĚTI – BIT EEMPE = LOG 1
SBI EECR, EEPE ;START ZÁPISU DO EEPROM – BIT EEPE = LOG 1
RET

EEPROM_READ: ;
SBIC EECR, EEPE ;ČEKÁNÍ DOKUD NENÍ PŘEDCHOZÍ ČTENÍ KOMPLETNÍ – KONTROLA BITU EEPE
RJMP EEPROM_READ ;ČEKÁNÍ DOKUD NENÍ PŘEDCHOZÍ ČTENÍ KOMPLETNÍ
```

```

OUT EEARL, ADR_REG      ;ZÁPIS ADRESY, Z KTERÉ BUDEME ČÍST DATA
SBI EECR,EERE          ;POVOLENÍ A START ČTENÍ DAT Z PAMĚTI
IN DATA_REG, EEDR     ;VÝPIS Z PAMĚTI (Z DATA REGISTRU) DO REGISTR DAT_REG
RET

```

### 4.13. UART komunikace (procedury)

Procedury, které následují, není úplně jednoduché odladit v AVR studiu. Debugger studia není dostatečně silný, aby byl schopen sám nastavovat příznaky příjmu a vysílání (RXC, TXC). Proto pokud chceme funkci UART sami simulovat, bude nutné tyto příznaky nastavovat samostatně (v reálu se příznaky nastavují samostatně) přímo v registru UCSRA.

```

;PROCEDURA PRO ZÁPIS BYTE Z UARTU, PŘI NASTAVENÍ TXC
.INCLUDE "TN2313DEF.INC"      ;TINY2313 DEFINIČNÍ SOUBOR
.DEF DATA = R16

```

```

VYSLANI_DAT:
    SBIS UCSRA, TXC          ;KONTROLA PŘÍZNAKU ÚSPĚŠNÉHO ODESLÁNÍ BYTE
    RJMP VYSLANI_DAT        ;POKUD JE BYTE ODESLÁN MŮŽE SE ODESLAT DALŠÍ
    OUT UDR, DATA          ;ODESLÁNÍ DAT DO DATOVÉHO REGISTRU USART
RET

```

```

;PROCEDURA PRO ČTENÍ BYTE Z UARTU, PŘI NASTAVENÍ RXC
.INCLUDE "TN2313DEF.INC"      ;TINY2313 DEFINIČNÍ SOUBOR
.DEF DATA = R16

```

```

PRIJEM_DAT:
    SBIS UCSRA, RXC          ;KONTROLA PŘÍZNAKU ÚSPĚŠNÉHO PŘÍJMU BYTE
    RJMP PRIJEM_DAT         ;POKUD JE BYTE ODESLÁN MŮŽE SE ODESLAT DALŠÍ
    IN DATA, UDR           ;PŘÍJEM DAT DO DATOVÉHO REGISTRU USART
RET

```

```

;PROGRAM PRO OPĚTOVNÉ VYSÍLÁNÍ A PŘÍJÍMÁNÍ ZNAKU S NASTAVENÍM ŘÍDÍCÍCH REGISTRŮ UART
;ŘÍDÍCÍ REGISTRY JSOU PODROBNĚ POPSÁNY V SKRIPTECH MIT

```

```

.INCLUDE "TN2313DEF.INC"      ;TINY2313 DEFINIČNÍ SOUBOR
.DEF ZNAK = R16

```

```

.CSEG
RJMP RESET

```

```

RESET:
    LDI R16, LOW(RAMEND)     ;NAČTENÍ POSLEDNÍ ADRESY Z DATOVÉ PAMĚTI (LOW)
    OUT SPL, R16            ;NASTAVENÍ UKAZATELE NA ZÁSOBNÍK (STACK POINTER)

```

```

LDI R16, 0b00000000      ;DATA PRO ŘÍDÍCÍ REGISTR UCSRA
OUT UCSRA, R16           ;NASTAVENÍ ŘÍDÍCÍHO REGISTRU UCSRA
LDI R16, 0b00011000      ; DATA PRO ŘÍDÍCÍ REGISTR UCSRB
OUT UCSRB, R16          ; NASTAVENÍ ŘÍDÍCÍHO REGISTRU UCSRB
LDI R16, 0b10000110      ; DATA PRO ŘÍDÍCÍ REGISTR UCSRC
OUT UCSRC, R16          ;NASTAVENÍ ŘÍDÍCÍHO REGISTRU UCSRC
LDI R16, 0b00000000      ; DATA PRO ŘÍDÍCÍ REGISTR UBRRH
OUT UBRRH, R16          ;NASTAVENÍ ŘÍDÍCÍHO REGISTRU UBRRH
LDI R16, 0b00110011      ; DATA PRO ŘÍDÍCÍ REGISTR UBRRL
OUT UBRRL, R16          ;NASTAVENÍ ŘÍDÍCÍHO REGISTRU UBRRL

LDI ZNAK, 0b00000000     ;NASTAVENÍ POČÁTEČNÍHO ZNAKU

START:
RCALL NACTI_ZNAK         ;NAČTENÍ VYSLANÉHO ZNAKU
RCALL ZAPIS_ZNAK        ;VYSLÁNÍ NÁSLEDUJÍCÍHO ZNAKU
RJMP START

NACTI_ZNAK:              ;VIZ VÝŠE - PROCEDURA PRO ČTENÍ BYTE Z UARTU
SBIS UCSRA, RXC
RJMP NACTI_ZNAK
IN ZNAK, UDR

RET

ZAPIS_ZNAK:
SBIS UCSRA, UDRE        ;KONTROLA ZDA JE REGISTR VYPRÁZDNĚNÝ (NEDOJDE K PŘEPSÁNÍ DAT)
RJMP ZAPIS_ZNAK
INC ZNAK                ;BUDE SE VYSÍLAT DALŠÍ ZNAK
OUT UDR, ZNAK           ;VYSÁLÁNÍ DALŠÍHO ZNAKU

RET

```

## 5. Přílohy

### 5.1. Directivy

**.EQU – definice symbolu** [.equ symbol=výraz] - slouží pro pojmenování literálu nebo pro zavedení nového jména pro stávající symbol.

**.SET – nastavení hodnoty symbolu** [.set symbol=výraz] – má podobný účel jako .EQU, rozdíl je v tom, že symbol zavedený pomocí .EQU nemůže měnit svou hodnotu.

**.DEF – definice symbolického jména registru** [.def symbol=registr] -umožňuje zavádět pro registry nové symboly. Jeden registr může mít více takových aliasů, symboly je možné předefinovat.

**.CSEG / .DSEG / .ESEG – výběr segmentu** slouží k výběru jednoho ze tří paměťových prostorů

- .CSEG – programový segment (paměť programu) – výchozí segment
- .DSEG – datový segment (paměť SRAM)
- .ESEG – segment pro paměť E2PROM.

**.ORG – nastavení hodnoty lokačního čítače** [.org výraz] - výchozí hodnota adres pro segmenty. Pro .CSEG a .ESEG je nula (0), pro .DSEG je \$60 (přeskočí se registrové pole a V/V registry).

**.BYTE – vyhrazení prostoru v bajtech** [[návěští:] .byte výraz] - slouží k vyhrazení prostoru v datovém segmentu (paměti RAM). Výraz udává počet bajtů, které chceme pro proměnnou vyhradit.

**.DB – uložení konstanty do paměti programu** [.db výraz [,výraz...]] - slouží k uložení konstanty (v rozměru bajtu) do FLASH nebo E<sup>2</sup>PROM – programové paměti.

**.DW – uložení konstanty do paměti programu** [[návěští:] .dw výraz[,výraz...]] - slouží k uložení konstanty (v rozměru slova) do FLASH nebo E<sup>2</sup>PROM.

**.INCLUDE – vložení obsahu externího souboru** [.include „jméno\_souboru"] - vloží na místo direktivy obsah jiného souboru. V adresáři C:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes je pro každý typ mikroprocesoru AVR jeden definiční soubor, který obsahuje:

- určení typu mikroprocesoru
- definice V/V registrů
- definice ukazatelových registrů
- definice konců paměťových segmentů
- definice vektorů přerušení.

**.EXIT – uložení překladu** [.exit] - signalizuje konec zdrojového kódu, předčasně ukončí překlad.

**.LIST – zapnutí generování výpisu** [.list] – opakem je direktiva .NOLIST – vypne generování výpisu.

## 5.2. Stavový registr – SREG



**C (Bit 0)** – Carry příznak indikuje přenos do vyššího řádu při aritmetických a logických operacích.

**Z (Bit 1)** – Zero příznak indikuje nulový výsledek aritmetických a logických operací.

**N (Bit 2)** – Příznak Negative, indikuje záporný výsledek aritmetických a logických operací.

**V (Bit 3)** – Příznak přetečení dvojkového doplňku.

**S (Bit 4)** – Jedná se o exklusive OR mezi příznaky N xor V a určuje tedy znaménko výsledku.

**H (Bit 5)** – Half Carry určuje přenos mezi třetím a čtvrtým bitem (Vyžití u BCD).

**T (Bit 6)** – Transfer bit, využívá se při kopírování BLD a BTS jako cílový nebo zdrojový bit.

**I (Bit 7)** – Global Interrupt Enable poslouží pro globální povolení všech přerušení.

## 5.3. Aritmetické a logické instrukce

Syntaxe instrukce		Popis		Příznaky	C
<b>ADD</b>	<b>Rd,Rr</b>	Součet bez přenosu	$Rd = Rd + Rr$	Z,C,N,V,H,S	1
<b>ADC</b>	<b>Rd,Rr</b>	Součet s přenosem C	$Rd = Rd + Rr + C$	Z,C,N,V,H,S	1
<b>ADIW</b>	<b>Rd, K</b>	Přičtení konstanty ke slovu	$Rd+1:Rd, K$	Z,C,N,V,S	2
<b>SUB</b>	<b>Rd,Rr</b>	Odčítání bez přenosu	$Rd = Rd - Rr$	Z,C,N,V,H,S	1
<b>SUBI</b>	<b>Rd,K8</b>	Odčítání konstanty	$Rd = Rd - K8$	Z,C,N,V,H,S	1
<b>SBC</b>	<b>Rd,Rr</b>	Odčítání s přenosem	$Rd = Rd - Rr - C$	Z,C,N,V,H,S	1
<b>SBCI</b>	<b>Rd,K8</b>	Odčítání konstanty s přenosem	$Rd = Rd - K8 - C$	Z,C,N,V,H,S	1
<b>SBIW</b>	<b>Rdl,K6</b>	Odečtení konstanty od slova	$Rdh:Rdl = Rdh:Rdl - K 6$	Z,C,N,V,S	2
<b>AND</b>	<b>Rd,Rr</b>	Logický AND (součin)	$Rd = Rd \cdot Rr$	Z,N,V,S	1
<b>ANDI</b>	<b>Rd,K8</b>	Logický AND s konstantou	$Rd = Rd \cdot K8$	Z,N,V,S	1

<b>OR</b>	<b>Rd,Rr</b>	Logický OR	$Rd = Rd \vee Rr$	Z,N,V,S	1
<b>ORI</b>	<b>Rd,K8</b>	Logický OR s konstantou	$Rd = Rd \vee K8$	Z,N,V,S	1
<b>EOR</b>	<b>Rd,Rr</b>	Logický XOR (Exclusive OR)	$Rd = Rd \oplus Rr$	Z,N,V,S	1
<b>COM</b>	<b>Rd</b>	Jednotkový doplněk	$Rd = \$FF - Rd$	Z,C,N,V,S	1
<b>NEG</b>	<b>Rd</b>	Dvojkový doplněk bytu	$Rd = \$00 - Rd$	Z,C,N,V,H,S	1
<b>SBR</b>	<b>Rd,K8</b>	Nastavení bitů (K) v registru	$Rd = Rd \vee K8$	Z,C,N,V,S	1
<b>CBR</b>	<b>Rd,K8</b>	vynulování bitů (K) v registru	$Rd = Rd \cdot (\$FF - K8)$	Z,C,N,V,S	1
<b>INC</b>	<b>Rd</b>	Inkrementace registru	$Rd = Rd + 1$	Z,N,V,S	1
<b>DEC</b>	<b>Rd</b>	Dekrementace registru	$Rd = Rd - 1$	Z,N,V,S	1
<b>TST</b>	<b>Rd</b>	Testování na nulu nebo na zápornou hodnotu	$Rd = Rd \cdot Rd$	Z,C,N,V,S	1
<b>CLR</b>	<b>Rd</b>	Vymazání registru	$Rd = 0$	Z,N,V,S	1
<b>SER</b>	<b>Rd</b>	Nastavení celého registru na „1“	$Rd = \$FF$	None	1

## 5.4. Instrukce skoků (větvení programu)

<i>Syntaxe instrukce</i>		<i>Popis</i>		<i>Příznaky</i>	<i>C</i>
<b>RJMP</b>	<b>k</b>	Relativní skok na návěští	$PC = PC + k + 1$	None	2
<b>IJMP</b>	<b>None</b>	Nepřímý skok (na adr. Z)	$PC = Z$	None	2
<b>RCALL</b>	<b>k</b>	Relativní volání podprogramu	$STACK = PC + 1, PC = PC + k + 1$	None	3/4
<b>ICALL</b>	<b>None</b>	Nepřímé volání podprogramu	$STACK = PC + 1, PC = Z$	None	3/4
<b>RET</b>	<b>None</b>	Návrat z podprogramu	$PC = STACK$	None	4 / 5
<b>RETI</b>	<b>None</b>	Návrat z přerušení	$PC = STACK$	I	4 / 5
<b>CPSE</b>	<b>Rd,Rr</b>	V případě rovnosti přeskoč	$\text{if}(Rd == Rr) PC = PC + 2 \text{ or } 3$	None	1/2/3
<b>CP</b>	<b>Rd,Rr</b>	Porovnání a nastavení příznaků	$Rd - Rr$	Z,C,N,V,H,S	1
<b>CPC</b>	<b>Rd,Rr</b>	Porovnání s přenosem	$Rd - Rr - C$	Z,C,N,V,H,S	1
<b>CPI</b>	<b>Rd,K8</b>	Srovnání s konstantou	$Rd - K$	Z,C,N,V,H,S	1
<b>SBRC</b>	<b>Rr,b</b>	Přeskoč, když bit v registru je „0“	$\text{if}(Rr(b) == 0) PC = PC + 2 \text{ or } 3$	None	1/2/3
<b>SBRS</b>	<b>Rr,b</b>	Přeskoč, když bit v registru je „1“	$\text{if}(Rr(b) == 1) PC = PC + 2 \text{ or } 3$	None	1/2/3
<b>SBIC</b>	<b>P,b</b>	Přeskoč, když bit v I/O reg. je „0“	$\text{if}(I/O(P,b) == 0) PC = PC + 2 \text{ or } 3$	None	1/2/3
<b>SBIS</b>	<b>P,b</b>	Přeskoč, když bit v I/O reg. je „1“	$\text{if}(I/O(P,b) == 1) PC = PC + 2 \text{ or } 3$	None	1/2/3
<b>BRBC</b>	<b>s,k</b>	Přeskoč, když Status flag je „0“	$\text{if}(SREG(s) == 0) PC = PC + k + 1$	None	1/2
<b>BRBS</b>	<b>s,k</b>	Přeskoč, když Status flag je „1“	$\text{if}(SREG(s) == 1) PC = PC + k + 1$	None	1/2
<b>BREQ</b>	<b>k</b>	Skok při rovnosti (Z = „1“)	$\text{if}(Z == 1) PC = PC + k + 1$	None	1/2
<b>BRNE</b>	<b>k</b>	Skok při nerovnosti (Z = „0“)	$\text{if}(Z == 0) PC = PC + k + 1$	None	1/2
<b>BRCS</b>	<b>k</b>	Skok pokud je nastavený bit C	$\text{if}(C == 1) PC = PC + k + 1$	None	1/2
<b>BRCC</b>	<b>k</b>	Skok pokud je vynulovaný bit C	$\text{if}(C == 0) PC = PC + k + 1$	None	1/2
<b>BRSH</b>	<b>k</b>	Skok když je rovny nebo větší	$\text{if}(C == 0) PC = PC + k + 1$	None	1/2
<b>BRLO</b>	<b>k</b>	Skok pokud je menší	$\text{if}(C == 1) PC = PC + k + 1$	None	1/2
<b>BRMI</b>	<b>k</b>	Skok při záporné hodnotě	$\text{if}(N == 1) PC = PC + k + 1$	None	1/2
<b>BRPL</b>	<b>k</b>	Skok při kladné hodnotě	$\text{if}(N == 0) PC = PC + k + 1$	None	1/2
<b>BRGE</b>	<b>k</b>	Skok při větší nebo rovno (znam.)	$\text{if}(S == 0) PC = PC + k + 1$	None	1/2
<b>BRLT</b>	<b>K</b>	Skok při menší než (znaménkově)	$\text{if}(S == 1) PC = PC + k + 1$	None	1/2
<b>BRHS</b>	<b>K</b>	Skok při nastaveném příznaku H	$\text{if}(H == 1) PC = PC + k + 1$	None	1/2

<b>BRHC</b>	K	Skok při vynulovaném příznaku H	if(H==0) PC = PC + k + 1	None	1/2
<b>BRTS</b>	K	Skok při nastaveném příznaku T	if(T==1) PC = PC + k + 1	None	1/2
<b>BRTC</b>	K	Skok při vynulovaném příznaku T	if(T==0) PC = PC + k + 1	None	1/2
<b>BRVS</b>	K	Skok při přetečení dvojk. doplňku	if(V==1) PC = PC + k + 1	None	1/2
<b>BRVC</b>	K	Skok – dvojk. doplněk nepřetekl	if(V==0) PC = PC + k + 1	None	1/2
<b>BRIE</b>	K	Skok když je povolené přerušení	if(I==1) PC = PC + k + 1	None	1/2
<b>BRID</b>	K	Skok když není povolené přerušení	if(I==0) PC = PC + k + 1	None	1/2

## 5.5. Instrukce přesunu dat

<i>Syntaxe instrukce</i>		<i>Popis</i>		<i>Příznaky</i>	<i>C</i>
<b>MOV</b>	<b>Rd,Rr</b>	Překopíruje obsah registru	Rd = Rr	None	1
<b>LDI</b>	<b>Rd,K8</b>	Přesun konstanty do registru R16-R31	Rd = K	None	1
<b>LDS</b>	<b>Rd,k</b>	Přímí přesun z datového prostoru	Rd = (k)	None	2*
<b>LD</b>	<b>Rd,X</b>	Přesun dat z adresy X do registru	Rd = (X)	None	2*
<b>LD</b>	<b>Rd,X+</b>	Přesun dat z adresy X do registru, inc	Rd = (X), X=X+1	None	2*
<b>LD</b>	<b>Rd,-X</b>	Přesun dat z adresy X do registru, dec	X=X-1, Rd = (X)	None	2*
<b>LD</b>	<b>Rd,Y</b>	Přesun dat z adresy Y do registru	Rd = (Y)	None	2*
<b>LD</b>	<b>Rd,Y+</b>	Přesun dat z adresy Y do registru, inc	Rd = (Y), Y=Y+1	None	2*
<b>LD</b>	<b>Rd,-Y</b>	Přesun dat z adresy Y do registru, dec	Y=Y-1, Rd = (Y)	None	2*
<b>LDD</b>	<b>Rd,Y+q</b>	Přesun dat z adresy Y+q do registru	Rd = (Y+q)	None	2*
<b>LD</b>	<b>Rd,Z</b>	Přesun dat z adresy Z do registru	Rd = (Z)	None	2*
<b>LD</b>	<b>Rd,Z+</b>	Přesun dat z adresy Z do registru, inc	Rd = (Z), Z=Z+1	None	2*
<b>LD</b>	<b>Rd,-Z</b>	Přesun dat z adresy Z do registru, dec	Z=Z-1, Rd = (Z)	None	2*
<b>LDD</b>	<b>Rd,Z+q</b>	Přesun dat z adresy Z+q do registru	Rd = (Z+q)	None	2*
<b>STS</b>	<b>k,Rr</b>	Přímý přesun do datového prostoru	(k) = Rr	None	2*
<b>ST</b>	<b>X,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu X	(X) = Rr	None	2*
<b>ST</b>	<b>X+,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu X, inc	(X) = Rr, X=X+1	None	2*
<b>ST</b>	<b>-X,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu X, dec	X=X-1, (X)=Rr	None	2*
<b>ST</b>	<b>Y,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Y	(Y) = Rr	None	2*
<b>ST</b>	<b>Y+,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Y, inc	(Y) = Rr, Y=Y+1	None	2
<b>ST</b>	<b>-Y,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Y, dec	Y=Y-1, (Y) = Rr	None	2
<b>ST</b>	<b>Y+q,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Y+q	(Y+q) = Rr	None	2
<b>ST</b>	<b>Z,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Z	(Z) = Rr	None	2
<b>ST</b>	<b>Z+,Rr</b>	Nepřímý přesun z reg. do datového prostoru po mocí indexu Z, inc	(Z) = Rr, Z=Z+1	None	2

ST	-Z,Rr	Nepřímý přesun z reg. do datového prostoru po mocí indexu Z, dec	$Z=Z-1, (Z) = Rr$	None	2
ST	Z+q,Rr	Nepřímý přesun z reg. do datového prostoru po mocí indexu Z+q	$(Z+q) = Rr$	None	2
LPM	None	Přesun z programové paměti do R0	$R0 = (Z)$	None	3
LPM	Rd,Z	Přesun z programové paměti do Rd	$Rd = (Z)$	None	3
LPM	Rd,Z+	Přesun z program. paměti do Rd, inc	$Rd = (Z), Z=Z+1$	None	3
IN	Rd,P	Přesun dat z I/O registru do Rd	$Rd = P$	None	1
OUT	P,Rd	Přesun obsahu registru Rd do I/O	$P = Rr$	None	1
PUSH	Rd	Uložení do zásobníku	$STACK = Rr$	None	2
POP	Rr	Čtení ze zásobníku	$Rd = STACK$	None	2

## 5.6. Bitové instrukce (testování bitu)

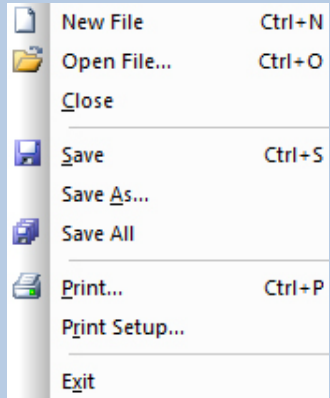
Syntaxe instrukce		Popis		Příznaky	C
LSL	Rd	Logický posun vlevo	$Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)$	Z,C,N,V,H,S	1
LSR	Rd	Logický posun vpravo	$Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)$	Z,C,N,V,S	1
ROL	Rd	Rotace vlevo s přenosem	$Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)$	Z,C,N,V,H,S	1
ROR	Rd	Rotace vpravo s přenosem	$Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)$	Z,C,N,V,S	1
ASR	Rd	Aritmetický posun vpravo	$Rd(n)=Rd(n+1), n=0,\dots,6$	Z,C,N,V,S	1
SWAP	Rd	Prohození 4 bitů v Rd	$Rd(3..0)=Rd(7..4), Rd(7..4)=Rd(3..0)$	None	1
BSET	s	Nastavení bitů v SREG reg.	$SREG(s) = 1$	SREG(s)	1
BCLR	s	Smazání bitů v SREG reg.	$SREG(s) = 0$	SREG(s)	1
SBI	P,b	Nastavení bitu I/O registru	$I/O(P,b) = 1$	None	2
CBI	P,b	Smazání bitu I/O registru	$I/O(P,b) = 0$	None	2
BST	Rr,b	Přenesení bitu z Rd do T(SREG)	$T = Rr(b)$	T	1
BLD	Rd,b	Přenesení bitu T(SREG) do Rd	$Rd(b) = T$	None	1
SEC	None	Nastavení příznaku Carry	$C = 1$	C	1
CLC	None	Vymazání příznaku Carry	$C = 0$	C	1
SEN	None	Nastavení příznaku záporného výsledku	$N = 1$	N	1
CLN	None	Vymazání příznaku záporného výsledku	$N = 0$	N	1
SEZ	None	Nastavení příznaku nuly	$Z = 1$	Z	1
CLZ	None	Vymazání příznaku nuly	$Z = 0$	Z	1
SEI	None	Nastavení příznaku přerušení	$I = 1$	I	1
CLI	None	Vymazání příznaku přerušení	$I = 0$	I	1
SES	None	Nastavení znaménko. příznaku	$S = 1$	S	1
CLN	None	Vymazání znaménko. příznaku	$S = 0$	S	1
SEV	None	Nastavení příznaku přetečení	$V = 1$	V	1
CLV	None	Vymazání příznaku přetečení	$V = 0$	V	1
SET	None	Nastavení příznaku bitu T	$T = 1$	T	1

<b>CLT</b>	<b>None</b>	Vymazání příznaku bitu T	T = 0	T	1
<b>SEH</b>	<b>None</b>	Nastavení příznaku Half Carry	H = 1	H	1
<b>CLH</b>	<b>None</b>	Vymazání příznaku Half Carry	H = 0	H	1
<b>NOP</b>	<b>None</b>	Prázdná instrukce	None	None	1
<b>SLEEP</b>	<b>None</b>	Zapnutí úsporného režimu	See instruction manual	None	1
<b>WDR</b>	<b>None</b>	Reset časovače Watchdog	See instruction manual	None	1

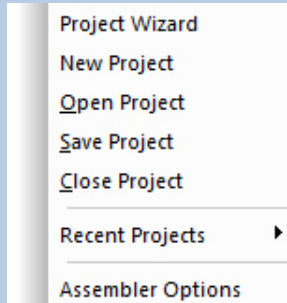
<b>Rd</b>	Registry R0 – R30 - cílový registry pro obecné využití
<b>Rr</b>	Registry R0 – R30 - zdrojové registry pro obecné využití
<b>R0 – R15</b>	Registry R0 k R15 mají různé adresující schopnosti (nelze použít instrukci LDI) oproti registrům R16 až R31.
<b>b</b>	Určuje pozici bitu v registru a nabývá hodnot 0 – 7
<b>s</b>	Určuje pozici bitu v registru SREG, nabývá hodnot 0 – 7
<b>P</b>	Adresa brány I/O
<b>K</b>	Data, konstanta - 8 bitů
<b>k</b>	Adresa
<b>q</b>	Offset (posun) adresy – pro přímé adresování
<b>X,Y,Z</b>	Registry pro nepřímé adresování (16bitů) tvořeny pomocí R26 až R31
<b>Návěští</b>	zvláštní případ symbolu, návěští je ukončeno dvojtečkou (:). Používá se k označení určitého místa v programu (relativní adresa) a pro pojmenovávání proměnných
<b>PC</b>	<b>program counter</b> – obsahuje adresu (paměti programu) právě vykonávané instrukce

## 5.7. Menu – AVR Studio

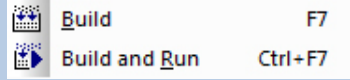
### 5.7.1. Souborové menu

	<b>New File</b>	Vytvoření prázdného nového souboru
	<b>Open File</b>	Otevření nového souboru v textovém editoru
	<b>Close</b>	Zavření aktivního textového souboru
	<b>Save</b>	Uložení aktuálního souboru
	<b>Save As</b>	Uložení aktivního souboru s vlastním jménem
	<b>Save All</b>	Uložení všech souborů a nastavení projektu
	<b>Print</b>	Tisk aktivního souboru
	<b>Print Setup</b>	Nastavení tisku
	<b>Exit</b>	Exit – projekty jsou automaticky uloženy














### 5.7.2. Projektové menu

	<b>Project Wizard</b>	Průvodce projektem – aktivní projekt musí být zavřen
	<b>New Project</b>	Nový projekt – aktivní projekt musí být zavřen
	<b>Open Project</b>	Otevření projektu – soubor s koncovkou *.aps
	<b>Save Project</b>	Uložení aktivního projektu se všemi nastaveními
	<b>Close Project</b>	Uzavření aktivního projektu
	<b>Recent Project</b>	Výpis naposledy otevřených projektů – volba jednoho
	<b>Assembler Options</b>	Otevře se konfigurační dialog pro aktuální projekt.

















### 5.7.3. Menu kompilace (překlad souboru)

	<b>Build</b>	Překlad aktuálního projektu (*.asm, *.c)
	<b>Build and Run</b>	Překlad, pokud nejsou chyby, spustí se odladění

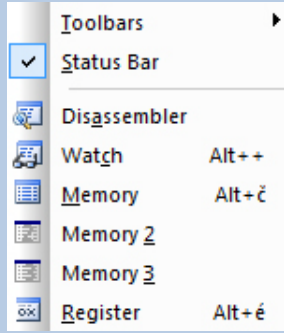
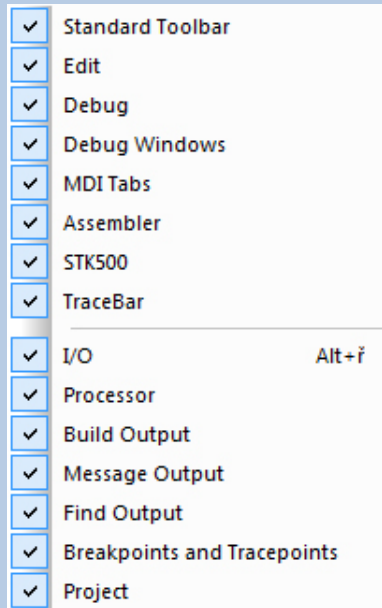
## 5.7.4. Edit menu

 <b>U</b> ndo	Ctrl+Z	<b>Undo</b>	Zrušení poslední editace
 <b>R</b> edo	Ctrl+Y	<b>Redo</b>	Obnovení jakékoliv zrušené editace
 <b>Cu</b> t	Ctrl+X	<b>Cut</b>	Vyjmutí a zkopírování textu
 <b>C</b> opy	Ctrl+C	<b>Copy</b>	Kopírování textu
 <b>P</b> aste	Ctrl+V	<b>Paste</b>	Vložení textu ze schránky do editoru
 <b>T</b> oggle Bookmark	Ctrl+F2	<b>Toggle Book.</b>	Zobrazení záložek na vybraném řádku
 Remove Bookmarks	Ctrl+Shift+F2	<b>Remove Book.</b>	Odstranění záložek
 <b>F</b> ind...	Ctrl+F	<b>Find</b>	Vyhledávání v textu aktivního souboru
 Find in Files...		<b>Find in Files</b>	Vyhledávání ve všech souborech projektu
 <b>R</b> eplace...	Ctrl+H	<b>Replace</b>	Nalezení a nahrazení textu v souboru
 Next Error	F4	<b>Next Error</b>	Nalezení chyby v kódu, pokud nějaká je
 Show Whitespace		<b>Show Whitesp.</b>	Zobrazení skrytých znaků (tab, mezery, ...)
 Font and Color...		<b>Font and color</b>	Nastavení fontů a barev prac. prostředí

## 5.7.5. Tools menu

 <b>A</b> VR Prog...		<b>Avr Prog</b>	Spustí se program AVR Prog (je-li nainstal.)
 ICE50 <b>U</b> pgrade...		<b>UPDATE</b>	
 ICE50 <b>S</b> elftest...		<b>ICE50, JTAGICE, AVR ONE, AVRISP, AVR Dragon, STK600</b>	Aktualizace součástí AVR studia
 JTAGICE mkII <b>U</b> pgrade...		<b>Customize.</b>	Uživatelské nastavení menu, nástrojů, příkazů
 AVR ONE! <b>U</b> pgrade...		<b>Options</b>	Nastavení editoru, pracovního prostředí, Breakpointy (zarážky)
 AVRISP mkII <b>U</b> pgrade		<b>Show Key Assignm.</b>	Zkratky příkazů – nutno vybrat modul
 AVR Dragon <b>U</b> pgrade		<b>Plug-in manager</b>	Volba Plug in (zásuvných) modulů
 STK600 <b>U</b> pgrade		<b>Program AVR</b>	Programování CPU (hardware)
 <b>C</b> ustomize...		<b>FLIP3 Information</b>	In-system programování procesoru (za chodu)
 <b>O</b> ptions...		<b>AVR Wireless Studio</b>	Komponenta AVR Studia
 Show Key Assignments		<b>AVR Battery Studio</b>	Komponenta AVR Studia
 <b>P</b> lug-in Manager...			
 <b>P</b> rogram AVR			
 <b>F</b> LIP3 Information			
 <b>A</b> VR Wireless Studio			
 <b>A</b> VR Battery Studio			






















## 5.7.6. View menu (zobrazení)

	<b>Toolbars</b>	Zobrazení ikon v nástrojové liště – viz níže
	<b>Status Bar</b>	Zobrazení stavové linky v dolní části okna
	<b>Disassembler</b>	Zobrazení podrobných informací o kódu
	<b>Watch</b>	Monitoring vybraných registrů (hodnot)
	<b>Memory</b>	Nahlížení do paměti (programu, I/O, dat, atd.)
	<b>Memory 2</b>	Další okno pro nahlížení do paměti
	<b>Memory 3</b>	Další okno pro nahlížení do paměti
	<b>Register</b>	Okno s pomocnými registry R0 až R31
	<b>Standard Toolbar</b>	Zobrazení menu se základními nástroji
	<b>Edit</b>	Zobrazení Edit menu
	<b>Debug</b>	Zobrazení menu pro odladění programu
	<b>Debug Windows</b>	Zobrazení lišty s ikonami, které spouští doplňující okna pro odladění programu
	<b>Assembler</b>	Zobrazení menu build
	<b>STK500</b>	Zobrazení menu pro připojení k programátoru
	<b>TraceBar</b>	Zobrazení menu pro trasování programu
	<b>I/O</b>	Zobrazení pravé části pracovní plochy (I/O registry)
	<b>Processor</b>	Zobrazení levé části pracovní plochy se základními registry procesoru
	<b>Build Output</b>	Zobrazení informací o překladu
	<b>Message Output</b>	Zobrazení informací o překladu
	<b>Find Output</b>	Zobrazení informací o překladu
	<b>Breakpoints ...</b>	Zobrazení záznamů od jednotliv. breakpointů
	<b>Project</b>	Zobrazení struktury projektu























Kliknutím prvním tlačítkem myši na jakékoliv otevřené okno lze upravit jeho parametry. Například pokud si necháme zobrazit okno s pomocnými registry a klikneme do tohoto okna pravým tlačítkem myši, pak máme možnost zobrazit data uložené v těchto registrech v binární, hexadecimální, dekadické nebo ASCII podobě. Jednotlivá okna lze pomocí myši přiřadit do jednotlivých částí pracovní plochy, tak aby nebyla plovoucí.

Menu **tool bars** se dá také vyvolat kliknutím pravého tlačítka myši na lištu s ikonami v horní části obrazovky.

### 5.7.7. Debug menu (ladění)

 Start <u>D</u> ebugging      Ctrl+Shift+Alt+F5	<b>Start Debug</b>	Spuštění odladění programu
 <u>S</u> top Debugging      Ctrl+Shift+F5	<b>Stop Debug</b>	Zastavení odladování programu
 <u>R</u> un      F5	<b>Run</b>	Spuštění aktivního kódu
 <u>B</u> reak      Ctrl+F5	<b>Break</b>	Zastavení běhu aktivního kódu
 <u>R</u> eset      Shift+F5	<b>Reset</b>	Zastavení a návrat na začátek
 <u>S</u> tep <u>I</u> nto      F11	<b>Step Into</b>	Přeskok na následující instrukci
 <u>S</u> tep <u>O</u> ver      F10	<b>Step Over</b>	Přeskočí i celé funkce nebo příkazy
 <u>S</u> tep <u>O</u> ut      Shift+F11	<b>Step Out</b>	Opustí aktuální funkci
 <u>R</u> un to <u>C</u> ursor      Ctrl+F10	<b>Run to Cursor</b>	Program poběží ke kurzoru
 <u>A</u> uto Step      Alt+F5	<b>Auto Step</b>	Auto krokování dokud nenarazí na nějaký breakpoint nebo Break
 Next Breakpoint      Ctrl+F9	<b>Next Break.</b>	Přeskočí na další záchytný bod
 <u>N</u> ew Breakpoint	<b>New Break.</b>	Vytvoří nový záchytný bod
 <u>T</u> oggle Breakpoint      F9	<b>Toggle Break.</b>	Zobrazuje záchytné body
 Remove all Breakpoints	<b>Remove all Br.</b>	Vymaže všechny záchytné body kódu
 Trace	<b>Trace</b>	Menu pro sledování chodu programu
 Stack Monitor	<b>Stack Monitor</b>	Pouze pro ICE50
 <u>S</u> how Next Statement      Alt+Num *	<b>Show Next Stat</b>	Zobrazení žluté šipky u aktiv. příkazu
 <u>Q</u> uickwatch      Shift+F9	<b>Quick Watch</b>	Okno pro rychlý vlastní přehled prog.
 Select <u>P</u> latform and Device...	<b>Select Platform</b>	Zobrazí se Obrázek 3
 Up/Download Memory	<b>Up/Download</b>	Upload / download bloku paměti
 AVR Simulator Options      Alt+O	<b>Avr Sim. Device</b>	Nastavení konkrétního procesoru

## 5.7.8. Důležité příkazové zkratky

	<b>New file</b>	<b>Ctrl + N</b>	Vytvoření nového souboru
	<b>Open File</b>	<b>Ctrl + O</b>	Otevření souboru
	<b>Save File</b>	<b>Ctrl + S</b>	Uložení souboru
	<b>Print File</b>	<b>Ctrl + P</b>	Vytištění aktuálního souboru
	<b>Assemble</b>	<b>F7</b>	Překlad aktuálního zdrojového kódu
	<b>Assemble and run</b>	<b>Ctrl + F7</b>	Překlad zdrojového kódu a spuštění odladění
	<b>Start Debugging</b>	<b>Ctrl + Alt + Shift + F5</b>	Spuštění odladění zdrojového kódu
	<b>Stop Debugging</b>	<b>Ctrl + Shift + F5</b>	Zastavení odladění zdrojového kódu
	<b>Run</b>	<b>F5</b>	Spuštění programu
	<b>Break</b>	<b>Ctrl + F5</b>	Zastavení programu
	<b>Reset</b>	<b>Shift + F5</b>	Zastavení programu a návrat na začátek
	<b>Step Into</b>	<b>F11</b>	Přeskok na následující instrukci
	<b>Step Over</b>	<b>F10</b>	Přeskočí i celé funkce nebo příkazy
	<b>Run to Cursor</b>	<b>Ctrl + F10</b>	Program běží, dokud nenarazí na kurzor
	<b>Auto Step</b>	<b>Alt + F10</b>	Automatické krokování
	<b>Breakpoints</b>	<b>F9</b>	Zobrazení záchytných bodů
	<b>Undo</b>	<b>Ctrl + Z</b>	Zrušení poslední editace
	<b>Redo</b>	<b>Ctrl + Y</b>	Obnovení jakékoliv zrušené editace
	<b>Cut</b>	<b>Ctrl + X</b>	Vyjmutí a zkopírování textu
	<b>Copy</b>	<b>Ctrl + C</b>	Kopírování textu
	<b>Paste</b>	<b>Ctrl + V</b>	Vložení textu ze schránky do editoru
	<b>Find</b>	<b>Ctrl + F</b>	Vyhledávání v textu aktivního souboru
	<b>Raplace</b>	<b>Ctrl + H</b>	Nalezení a nahrazení textu v souboru
	<b>Help</b>	<b>F1</b>	Nápověda AVR Studia (velmi podrobná)

Další klávesové zkratky lze nalézt v menu **Tools – Show Key Assignments**, nebo v nápovědě (**F1**).

## 6. Použitá literatura

Seznam použité literatury, internetových adres a zajímavé odkazy na tematiku AVR.

- [1] Vladimír Váňa: „Mikrokontroléry ATMEL AVR popis procesorů a instrukční soubor“ BEN technická literatura, Praha 2003
- [2] <http://programujte.com>
- [3] <http://www.atmel.com>
- [4] <http://www.hw.cz>
- [5] <http://avr-forum.cz>