

Popis vývojového prostředí

WinIDE51

1 Úvod

1.1 Využití WinIDE51

Dovolujeme si Vám představit **Integrované vývojové prostředí WINIDE51**, program určený pro výuku mikroprocesorové techniky na základě procesoru Intel 8051. Umožňuje aplikaci poznatků získaných v hodinách teorie v praktických cvičeních v laboratoři a využít těchto znalostí při konstrukci elektronických zařízení řízených procesorem. Výrobky využívající mikroprocesor mohou být náplní ročníkových nebo maturitních projektů Vašich žáků. Součástí vývojového prostředí je také obsluha HW emulátoru (lze bez jakékoliv úpravy využít také rozšířené emulátory EASYSOFT) a dále obsluha programátoru mikroprocesorů firmy ATMEL naší vlastní konstrukce. Tento programátor si mohou snadno vyrobit i žáci ve školních podmínkách, čímž jsou motivováni k dalšímu poznávání mikroprocesorové techniky. Program je určen a byl testován v prostředí Windows 9x, Win2000 a WinXP. Byl konzultován zkušenými pedagogy, tak aby maximálně vyhovoval výuce středoškolských studentů. Samozřejmě není vyloučeno ani jeho profesionální použití. Vývojové prostředí WinIDE 51 obsahuje:

- **editor** zdrojového textu s barevným zvýrazněním syntaktických symbolů
- výkonný 32 bitový **kompilátor** jazyka mikroprocesorů řady Intel 8051
- **debugger** pro analýzu programu, umožňující trasování programu, zobrazení SFR a paměti RAM.
- debugger dále umožňuje záznam stavů na portech jejich následné zobrazení v libovolném časovém měřítku a snadné odečítání doby mezi jednotlivými událostmi např. při práci s dynamickým displejem, sériovým kanálem apod.
- HW doplňkem je **emulátor** umožňující ladění programu přímo ve vyvíjeném zařízení. Po připojení modulu portů je možné snadno demonstrovat funkci základních periferních zařízení jako displeje, klávesnice, převodníku, paměti apod.
- v laboratoři s více pracovišti lze použít **přepínací modul**, umožňující připojení jednoho emulátoru až ke 12ti PC (odesláním z libovolného počítače dojde k automatickému přepnutí, žák má možnost vyzkoušet svůj program, poté se emulátor uvolní pro dalšího). Toto řešení současně zaručuje rovnocenné podmínky pro všechny žáky při minimálních pořizovacích nákladech.
- je-li k dispozici **síťové propojení počítačů**, je možné posílat data do emulátoru po síti, v tomto případě je počet připojení libovolný, na jedné síti je možné provozovat několik relací s několika emulátory současně.
- **programátor** WINPROG51, umožňuje programovat všechny běžné typy μ P firmy ATMEL (AT89C2051, AT89C51..55)

Kontakt

Mgr. Andrea Odehnalová
 Gen. Svobody 623/21, 674 01 Třebíč
 IČO: 65848764, tel./fax 568826160
 e-mail: aodehnalova@atlas.cz

2 Popis aplikace

2.1 Titulní lišta

Program WinIDE51 obsahuje **titulní lištu**, která se nachází v horní části okna, je zde uveden název právě zpracovávaného dokumentu. V pravé části titulní lišty se nachází 3 tlačítka, pomocí nichž můžeme okno dokumentu minimalizovat, maximalizovat nebo zavřít. Pod titulní lištou se nachází hlavní nabídka.

2.2 Hlavní nabídka

Hlavní nabídka obsahuje menu, sloužící k ovládání editoru. V pravé části obsahuje 3 tlačítka sloužící k minimalizaci, maximalizaci editoru a k uzavření aktuálního souboru. Nyní se seznámíme s jednotlivými položkami hlavní nabídky.

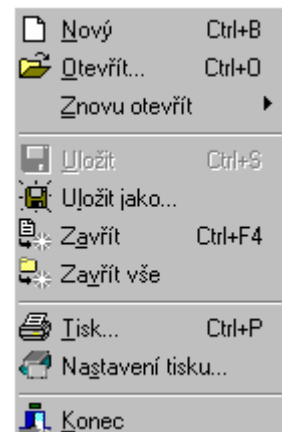
2.3 Panel nástrojů

Pod hlavní nabídkou se nachází **panel nástrojů**, který obsahuje ovládací tlačítka pro často používané příkazy. V místní nabídce prostoru panelu nástrojů nebo pomocí nabídky Nastavení položka Zobrazené ikony si můžeme vybrat zobrazení dalších ovládacích tlačítek.

2.3.1 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 Win51.exe soubor Win51.asm, objeví se dotaz, zda má být tento soubor otevřen jako šablona.
- **Otevřít Ctrl+O** - Otevírá soubory s příponou .asm, .lst Soubor s příponou .asm je program psaný v assembleru, 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.

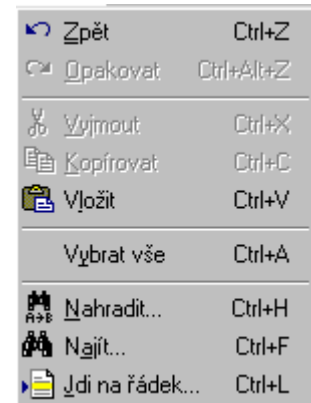


- **Tisk Ctrl+P** – Otevře se okno pro tisk známé z Windows. Můžeme tisknout celý soubor nebo jen vybranou část. Tiskne se barevně nebo černobíle podle nastavení v menu *Nastavení* v položce *Nastavení editoru*.
- **Nastavení tisku** – Otevře se okno pro nastavení tisku známé z Windows. Je možné vybrat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

2.3.2 Menu Úpravy

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

- **Zpět Ctrl+Z, Alt+BackSpace** – Vráti naposledy provedenou akci.
- **Opakovat Ctrl+Alt+Z** – Vráti 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. Zaškrtnutí 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 texty.
- **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.



Operace *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 kopírovat a ne přesouvat stiskneme přitom ještě klávesu Ctrl.

2.3.3 Menu Překladač

Přeloží editovaný soubor. Zobrazí se znaky (tečky a šipky) na liště řádků, otevře se protokol o překladu a zobrazí se výpis použitých symbolů. Více v kapitole Kompilátor.

2.3.4 Menu WinProg

Odešle se přeložený kód do emulátoru. Průběh přenosu indikuje ukazatel.

2.3.5 Menu Pomůcky

Nastavení SFR **Ctrl+N** – slouží pro rychlé a přehledné nastavení libovolného SFR, umožňuje hodnotu přímo vložit do editoru. Zapsáním binární, desítkové nebo hexadecimální hodnoty do editačního pole se po stisku Enter nastaví příslušné bity a režimy. Rovněž umožňuje hodnotu přímo vložit do editoru.

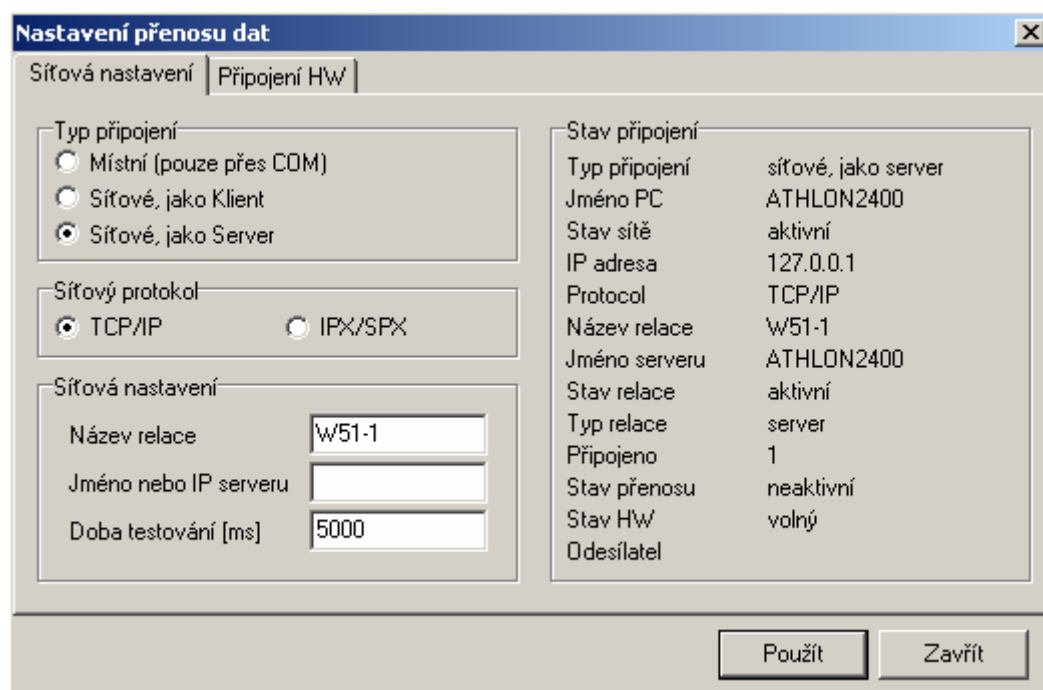
Převody **Ctrl+Q** – Otevře se okno Převody. Umožňuje převody mezi číselnými soustavami. Do libovolného řádku napíšeme číslo v dané soustavě. Tlačítko Převod ho převede do ostatních číselných soustav. Tlačítko Kopírovat označené číslo překopíruje do editoru na místo kurzoru.

Nastavení čítačů a časovačů **Ctrl+M** – slouží pro rychlé a přehledné nastavení SFR při práci s časovači. Umožňuje hodnotu přímo vložit do editoru. Hodnota použitého krystalu se zadává v MHz.

2.3.6 Menu Nastavení

Slouží k celkovému nastavení všech vlastností programu, ukládá se do 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 jestli se má tisknout vždy černobíle nebo podle barevného zvýraznění. Tlačítko *Původní1*, *Původní2* slouží k nejpoužívanějšímu nastavení. Tlačítko *Použít* aplikuje naše nastavení v editoru. Tlačítko *Storno* naše nastavení nepoužije.
- **Nastavení překladače** – Otevře okno s nastavením parametrů překladače.
- **Nastavení debuggeru** – Otevře okno s nastavením parametrů debuggeru.
- **Nastavení cest** – Otevře se okno, ve kterém nastavujeme cestu k uložení inicializačních a datových 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.
- **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
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 WinIDE 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.

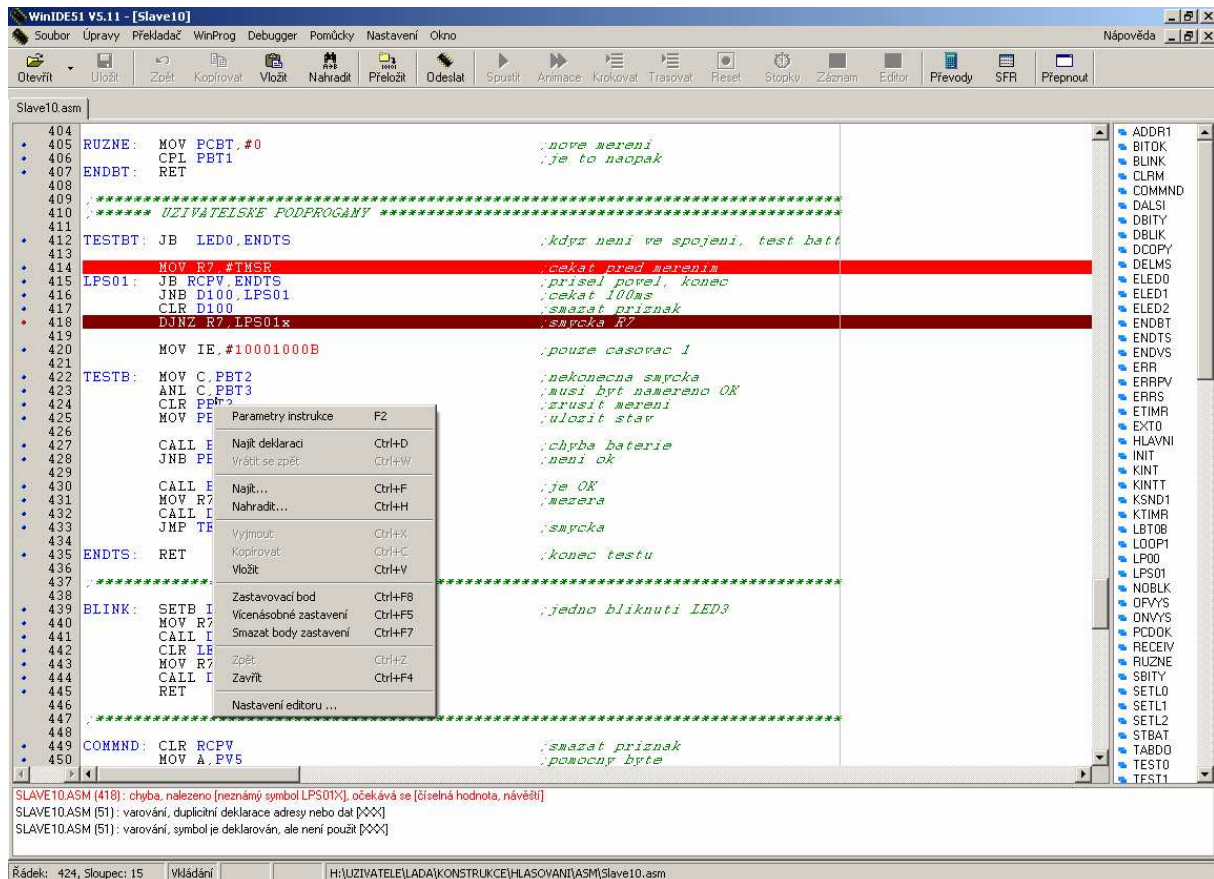
2.3.7 Menu Okno

Slouží k přepínání mezi editorem a debuggerem a k uspořádání oken. Jeho funkce je obvyklá v prostředí Windows.

- **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, Nad sebou, Za sebou** – uspořádá okno editoru a debuggeru

3 Textový editor

3.1 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.

3.1.1 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.

3.1.2 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 a. Na každý otevřený soubor se dostane me, jestliže klikneme na jeho záložku. Kliknutím pravým tlačítkem na záložku se vyvolá místní nabídka pro zavření, nový soubor a uložení pod jménem.

3.1.3 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 na daný řádek v okně editoru. Zobrazují se jednak samotné chyby, ty jsou navíc v editoru vyznačeny tmavě červeným zvýrazněním řádku, tak i varování, což jsou pomocné informace sloužící pro minimalizování běhových chyb. 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.

3.1.4 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 (\$INCLUDE), zobrazují se pouze symboly příslušné k danému souboru. Při kliknutí pravým tlačítkem na tomto okně opět vyvoláme místní plovoucí nabídku.

3.1.5 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 prepisovacím nebo vkládacím režimu, máme zapnutou klávesu Caps Lock a zda-li v dokumentu došlo ke změně. V poslední části stavového řádku je zobrazena místní nápověda.

3.2 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:

3.2.1 Parametry instrukce

Textový editor obsahuje zajímavou funkci, která se vyvolá stiskem klávesy F2. Pokud napíšeme klíčové slovo assembleru a stiskneme klávesu F2 (nebo místní nabídku, položku Parametry instrukce) dojde k vyvolání

```
PUSH DPL
PUSH DPH
MOV
```

@R0,#	(1c 2B)
@R1,#	(1c 2B)
@R0,A	(1c 1B)
@R1,A	(1c 1B)
@R0,DIRECT	(2c 2B)
@R1,DIRECT	(2c 2B)
A,#	(1c 2B)
A,@R0	(1c 1B)
A,@R1	(1c 1B)
A,DIRECT	(1c 2B)
A,R0	(1c 1B)
A,R1	(1c 1B)
A,R2	(1c 1B)
A,R3	(1c 1B)
A,R4	(1c 1B)

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 tuto 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.

3.2.2 Najít deklaraci

Pomocí místní nabídky můžeme 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. Vráť pohled na editovaný text do původního místa.

3.2.3 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 breakpoints 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 breakpoints 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 Kompilátor

4.1 Úvod

Kompilátor jazyka symbolických adres mikroprocesorů řady Ix51, který je obsažen v integrovaném prostředí WinIDE51 je založen na výkonné 32bitové technologii. Jedná se o jednopřechodový 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 WinIDE51, výstupem pak protokol o překladu *.LST a výsledné (cílové) soubory ve formátu Intel-HEX *.HEX a absolutní binární soubor *.BIN. Generování těchto souborů 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).

4.2 Popis syntaxe jazyka

Syntaxe jazyka je téměř shodná se syntaxí akceptovanou kompilátorem CASS51, tak aby bylo možné přímo využívat již vytvořené programy a neodchyluje se od obecně rozšířené syntaxe definované firmou Intel pro mikroprocesory řady Ix51.

4.3 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 reдекларace hodnot rezervovaných jmen není přípustná. V obou případech kompilátor ohlásí chybu.

4.3.1 Seznam rezervovaných jmen a jejich hodnot

Symbol	Hodnota	Symbol	Hodnota	Symbol	Hodnota	Symbol	Hodnota
AC	0D6H	IE	0A8H	R1	001H	SPSR	0AAH
ACC	0E0H	IE0	089H	R2	002H	T0	0B4H
AR0	000H	IE1	08BH	R3	003H	T1	0B5H
AR1	001H	INT0	0B2H	R4	004H	T2MOD	0C9H
AR2	002H	INT1	0B3H	R5	005H	T2CON	C8H
AR3	003H	IP	0B8H	R6	006H	TB8	09BH
AR4	004H	IT0	088H	R7	007H	TCON	088H
AR5	005H	IT1	08AH	RB8	09AH	TF0	08DH
AR6	006H	OV	0D2H	RCAP2H	0CBH	TF1	08FH
AR7	007H	P	0D0H	RCAP2L	0CAH	TH0	08CH
B	0F0H	P0	080H	RD	0B7H	TH1	08DH

CY	0D7H	P1	090H	REN	09CH	TH2	0CDH
DPH	083H	P2	0A0H	RI	098H	TI	099H
DPL	082H	P3	0B0H	RS0	0D3H	TL0	08AH
DL1	084H	PCON	087H	RS1	0D4H	TL1	08BH
EA	0AFH	PCT	0BFH	RXD	0B0H	TL2	0CCH
ES	0ACH	PS	0BCH	SBUF	099H	TMOD	089H
ET0	0A9H	PSW	0D0H	SCON	098H	TR0	08CH
ET1	0ABH	PT0	0B9H	SM0	09FH	TR1	08EH
ET2	0ADH	PT1	0BBH	SM1	09EH	TXD	0B1H
EX0	0A8H	PT2	0BDH	SM2	09DH	WMCON	096H
EX1	0AAH	PX0	0B8H	SP	081H	WR	0B6H
F0	0D5H	PX1	0BAH	SPCR	0D5H		
F1	0D1H	R0	000H	SPDR	086H		

4.3.2

4.3.3 Seznam rezervovaných symbolů

A	ELSE	MOD	R7
AB	END	MOV	RET
ACALL	ENDIF	MOVC	RETI
ADD	EQU	MOVX	RL
ADDC	HI	MUL	RLC
AJMP	HIGH	NOLIST	RR
AND	IF	NOP	RRC
ANL	INC	NOT	SET
BIT	INCLUDE	NOTW	SETB
C	JB	OR	SHL
CALL	JBC	ORG	SHR
CJNE	JC	ORL	SJMP
CLR	JMP	PC	SUBB
CPL	JNB	POP	SWAP
DA	JNC	PUSH	TEXT
DB	JNZ	R0	USING
DEC	JZ	R1	UNIT
DIV	LCALL	R2	XCH
DJNZ	LIST	R3	XCHD
DPTR	LJMP	R4	XOR
DS	LO	R5	XRL
DW	LOW	R6	XTAL

4.4 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

4.5 Konstanty

Kompilátor rozlišuje 5 možných způsobů zápisu konstant.

4.5.1 Číselné konstanty

dekadické obsahují pouze číslice 0-9 a mohou končit znakem D nebo d

Příklad:

10

1930D

59d

binární obsahují pouze číslice 0,1 a musí končit znakem B nebo b

Příklad:

10101010b

10101101B

hexadecimální obsahují pouze číslice 0-9 a znaky A-F. Vždy musí začínat číslicí a končit znakem H nebo h.

Příklad:

10h

0A1BH

4.5.2 Znakové konstanty

Obsahují libovolný ASCII znak, který je uzavřený do jednoduchých uvozovek

Příklad:

'A'

4.5.3 Symbol \$

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

4.6 Operátory

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

Operátor	Popis	Ekvivalentní zkrácený zápis
+	Sčítání	nemá
-	Odčítání	nemá
*	Násobení	nemá
/	Dělení	nemá
MOD	Celočíselný zbytek po dělení	%
OR	Logický součet	
AND	Logický součin	&
XOR	Logické exclusive or	^
NOT	Negace s 8 bitovým výsledkem	~
NOTW	Negace s 16 bitovým výsledkem	\
SHL	Bitový posun vlevo	<
SHR	Bitový posun vpravo	>
LOW()	Dolní byte operandu	LO()
HIGH()	Horní byte operandu	HI()

Ekvivalentní zkrácený zápis lze použít namísto delšího zápisu daného operátoru.

Příklad:

```
MOV A, #(ADR OR 00110011b)
```

je ekvivalentní zápisu

```
MOV A, #(ADR | 00110011b)
```

Operátory jsou uspořádány podle vzrůstající priority takto :

- + -
- / * MOD
- AND OR XOR SHR SHL
- NOT NOTW HIGH LOW

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

4.6.1 Operátory +, -, *, /

Syntaxe

```
<výraz1> + <výraz2>
```

```
<výraz1> - <výraz2>
```

```
<výraz1> * <výraz2>
```

```
<výraz1> / <výraz2>
```

+ , - , * , / jsou běžně známé matematické operátory.

4.6.2 Operátor MOD

Syntaxe:

```
<výraz1> MOD <výraz2>  
<výraz1> % <výraz2>
```

Výsledkem operátoru MOD je celočíselný zbytek po dělení výrazu1 výrazem2.

Příklad:

```
DB 10 MOD 3  
bude přeloženo jako:  
DB 1
```

4.6.3 Operátor OR, XOR, AND

Syntaxe:

```
<výraz1> OR <výraz2>  
<výraz1> | <výraz2>  
<výraz1> XOR <výraz2>  
<výraz1> ^ <výraz2>  
<výraz1> AND <výraz2>  
<výraz1> & <výraz2>
```

OR, XOR, AND jsou běžně známé operátory Booleovy algebry.

Příklad:

```
MOV A, #(ZNAK AND 10) | (MASK XOR 0F0h)
```

4.6.4 Operátor NOT

Syntaxe:

```
NOT <výraz>  
~ <výraz>
```

NOT je běžně známý operátor negace z Booleovy algebry. Výsledek je kompilátorem oříznut na 8 bitů, tj. může nabývat hodnoty pouze 0-255 (0-0FFh).

Příklad:

```
DB NOT(MASK XOR 0F0h)  
DB ~(MASK XOR 0F0h)
```

4.6.5 Operátor NOTW

Syntaxe:

```
NOTW <výraz>  
\ <výraz>
```


NOTW je běžně známý operátor negace z Booleovy algebry. Výsledek je kompilátorem oříznut na 16 bitů, tj. může nabývat hodnoty pouze 0-65535 (0-0FFFFh).

Příklad:

```
DW NOTW(MASK XOR 0F0FEh)
DW \ (MASK XOR 0F0FEh)
```

Existence dvou operátorů negace je nutná z důvodu, že kompilátor všechny výrazy vyhodnocuje s rozlišením 32 bitů. Toto rozlišení pak umožňuje správně kontrolovat rozsahy výsledků vyhodnocení výrazů. Negace výsledku s hodnotou 1 (NOT 1) však dá výsledek 0FFFFFFFh což představuje decimálně 4294967294. V takovém případě by kompilátor hlásil chybu překročení rozsahu. V případě zavedení dvou operátorů negace, provede kompilátor po negaci ještě operaci AND 0FFh pro NOT resp. AND 0FFFFh pro NOTW. Tím se výsledek dostane do správného rozsahu.

4.6.6 Operátor SHL (SHift Left)

Syntaxe:

```
<výraz1> SHL <výraz2>
<výraz1> < <výraz2>
```

Operátor SHL posune bity ve výsledku výrazu1 vlevo o počet bitů daných výsledkem výrazu2. Bity na pravé straně jsou nahrazeny 0. Operátor pracuje v 32 bitové aritmetice.

Příklad:

```
DB 00000001b SHL 3
bude přeloženo jako:
DB 00001000b
```

4.6.7 Operátor SHR (SHift Right)

Syntaxe:

```
<výraz1> SHR <výraz2>
<výraz1> > <výraz2>
```

Operátor SHR posune bity ve výsledku výrazu1 vpravo o počet bitů daných výsledkem výrazu2. Bity na levé straně jsou nahrazeny 0. Operátor pracuje v 32 bitové aritmetice.

Příklad:

```
DB 10000000b SHR 3
bude přeloženo jako:
DB 00010000b
```

4.6.8 Operátor LOW()

Syntaxe:

```
LOW(<výraz>)
LO(<výraz>)
```

Operátor LOW () vrací nižších 8 bitů výsledku výrazu. Operátor pracuje v 16 bitové aritmetice.

Příklad:

```
DB LOW ( 0FAFEh )  
bude přeloženo jako:  
DB 0FEh
```

4.6.9 Operátor HIGH()

Syntaxe:

```
HIGH( <výraz> )  
HI ( <výraz> )
```

Operátor HIGH () vrací vyšších 8 bitů výsledku výrazu. Operátor pracuje v 16 bitové aritmetice.

Příklad:

```
DB HIGH( 0FAFEh )  
bude přeloženo jako:  
DB 0FAh
```

4.7 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]
```

4.7.1 Návěští

Návěští je symbolické jméno, které musí být ukončeno dvojtečkou. Dvojtečka není součástí symbolického jména, používá se pouze pro 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:

```
Navestil:  
Navestil:
```

4.8 Pseudoinstrukce

Kompilátor má implementovány následující pseudoinstrukce:

4.8.1 Definice hodnot symbolických jmen - EQU, SET

Syntaxe:

<jméno> EQU <výraz>

<jméno> SET <výraz>

Symbolickému jménu je přiřazena hodnota získaná vyhodnocením výrazu. Pseudoinstrukce EQU a SET jsou vzájemně ekvivalentní. Kompilátor rozeznává obě pseudoinstrukce z důvodu kompatibility. Vzhledem ke striktní kontrole při překladu nemůže být jednomu jménu přiřazena různá hodnota vícekrát, tzn. není přípustná redeklarace.

Příklad:

RAM1 EQU 100

LOCK SET RAM1*2-10

4.8.2 Výběr segmentu – CSEG, BSEG, DSEG, ISEG, XSEG

Syntaxe:

CSEG [AT <konst>]

BSEG [AT <konst>]

DSEG [AT <konst>]

ISEG [AT <konst>]

XSEG [AT <konst>]

Pseudoinstrukce slouží pro výběr jednoho z 5 adresových prostorů, výchozím segmentem je CSEG. Aktuální segment určuje, ve kterém paměťovém prostoru se vyhrazuje paměť při použití pseudoinstrukce DS, DBIT, popř. nastavuje čítač segmentu použitím ORG. Nepovinná část AT umožňuje nastavit čítač segmentu na výchozí zadanou hodnotu, její použití

```
DSEG AT 32
ABC: DS 1
```

je ekvivalentní zápisu

```
DSEG
ORG 32
ABC: DS 1
```

Přehled segmentů, rozsahů a použitelných pseudoinstrukcí pro definici symbolů:

Segment	Název	Rozsah	Přímá definice	Vyhrazení
Přímo adresovatelný	DSEG	00-FFh	DATA	DS
Bitový	BSEG	00-7Fh	BIT	DBIT
Nepřímo adresovatelný	ISEG	00-FFh	IDATA	DS
Externí paměť DAT	XSEG	0000-FFFFh	XDATA	DS
Programový	CSEG	0000-FFFFh	CODE	ORG

4.8.3 Definice symbolických jmen bytových adres - DATA, IDATA, XDATA

Syntaxe:

```
<jméno> DATA <konst>  
<jméno> IDATA <konst>  
<jméno> XDATA <konst>
```

Pseudoinstrukce DATA, IDATA, XDATA umožňují definovat symbolická jména v příslušném segmentu, tedy v segmentech DSEG, ISEG, XSEG. Jako hodnota může být použita pouze číselná konstanta v rozsahu daného segmentu, tedy 0-0FFh, 0-0FFh, 0..0FFFFh.

4.8.4 Definice symbolických jmen bitových adres - BIT

Syntaxe:

```
<jméno> BIT <konst>
```

Pseudoinstrukce BIT umožňuje přiřadit přímo adresovatelným bitům ve vnitřní paměti procesoru symbolická jména (v segmentu BSEG). Jako hodnota musí být použita pouze číselná konstanta v intervalu 0-0FFh, symbolické označení bitu dle mnemoniky výrobce (viz. seznam rezervovaných jmen) nebo programátorem definované symbolické jméno s hodnotou v intervalu 0-0FFh.

Příklad:

```
LED1 BIT 34H  
TXBIT BIT TxD
```

Pokud je třeba použít místo konstanty výraz, musí se místo výrazu použít symbolické jméno a hodnotu výrazu mu přiřadit instrukcí SET nebo EQU.

Příklad:

```
LED1V EQU POZICE AND 00001000B  
LED1B BIT LED1V
```

Hodnoty symbolických jmen reprezentující bitové adresy lze definovat také za pomoci tečkové metody.

Syntaxe:

```
<jméno> BIT <konst>.<0..7>
```

Tuto tečkovou metodu nelze použít u všech bitových adres. Bázové adresy u kterých lze použít tečkovou konvenci jsou: 20h až 2Fh. Číslo za tečkou udává pořadí bitu zprava a může nabývat pouze hodnoty 0-7. Dále lze touto metodou adresovat všechny bitově adresovatelné SFR.

Příklad:

```
LED1 BIT 0C0h.0  
LED2 BIT 0C0h.1  
OUT BIT P0.7
```

V případě pokusu adresovat tečkovou metodou bit s jinou báзовou adresou než je povoleno, ohlásí kompilátor chybu. Pseudoinstrukci lze použít ve všech segmentech ale data se vyhradí vždy v BSEG

4.8.5 Definice hodnoty adresového symbolu - CODE

Syntaxe:

```
<jméno> DATA <konst>
```

Pseudoinstrukce CODE umožňuje přímo přiřadit symbolu adresu v programovém segmentu CSEG, jako by se jednalo o návěští.

4.8.6 Rezervování paměti v bytech - DS (Define Space)

Syntaxe:

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

Při překladu je v aktuálním segmentu rezervován počet bajtů daný výrazem. Tuto pseudoinstrukci lze použít ve všech datových segmentech, kromě BSEG, tam se použije DBIT. V tomto případě musí být hodnoty případných proměnných, použitých ve výrazu, definovány před jeho vyčíslením. Hodnota čítače segmentu se zvyšuje o velikost vyhrazené paměti.

4.8.7 Rezervování bitové paměti - DBIT (Define Bit)

Syntaxe:

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

Při překladu je v aktuálním segmentu rezervován počet bajtů daný výrazem. Tuto pseudoinstrukci lze použít pouze v segmentu BSEG. V tomto případě musí být hodnoty případných proměnných, použitých ve výrazu, definovány před jeho vyčíslením. Hodnota čítače segmentu se zvyšuje o velikost vyhrazené paměti.

4.8.8 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 - 0FFh. Povoleny jsou také řetězce znaků uzavřené v apostrofech. Například posloupnost znaků '123' se uloží jako 31H, 32H, 33H. Počet výrazů za jednou pseudoinstrukcí DB je omezen pouze délkou řádku. Lze použít pouze v programovém segmentu CSEG.

Příklad:

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

4.8.9 Definice 16 bitových konstant v paměti programu - DW (Define Word)

Syntaxe:

[návěštlí:] 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 pouze v programovém segmentu CSEG

Příklad:

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

4.8.10 Definice textů v paměti programu - TEXT

Syntaxe:

[návěštlí:] TEXT '<text>' [, "<text>"]

Pseudoinstrukce TEXT postupně ukládá ASCII kódy znaků řetězce uvedeného v uvozovkách na aktuální adresy, počínaje současnou adresou programového čítače. Počet znaků řetězce za jednou pseudoinstrukcí TEXT je omezen pouze délkou řádku.

4.8.11 Nastavení programového čítače - ORG

Syntaxe:

[návěštlí:] 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 1000h
```

4.8.12 Nastavení banky registrů – USING

Určuje, která banka je aktivní, potom symboly AR0..AR7 odpovídají adresám z dané banky. Fyzické přepnutí v procesoru musí zajistit programátor nastavením bitů RS0 a RS1

Příklad:

```
SET RS0
SET RS1
```

```
USING 3
MOV A,AR0 ;AR0 je na adr. 18H
```

4.8.13 Podmíněný překlad - IF, ELSE, ENDIF

Syntaxe:

```
[návěštlí:] IF <výraz>
[návěštlí:] ELSE
[návěštlí:] ENDIF
```

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 není 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.

Příklad:

```
SVITI EQU 1
IF SVITI
SETB P0.1
ELSE
CLR P0.1
ENDIF
```

4.8.14 Ukončení zdrojového textu - END

Syntaxe:

```
[návěštlí:]      END
```

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

4.8.15 Vložení externího souboru - \$INCLUDE, \$UNIT

Syntaxe:

```
$INCLUDE (jméno souboru)
$UNIT    (jméno souboru)
```

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.

4.8.16 Frekvence krystalu v debuggeru - XTAL

Syntaxe:

```
XTAL SET <konst>
```

Slouží pro nastavení frekvence krystalu v debuggeru. K aktualizaci dochází překladem zdrojého textu. Zadána frekvence se udává v Hz

4.9 Instrukce

Instrukce procesoru řady Ix51 jsou definovány firmou Intel. Popis jejich sémantického významu naleznete v odborné literatuře a není součástí tohoto manuálu. Všechny instrukce a standardní operandy jsou zároveň rezervovanými jmény kompilátoru.

4.10 Operandy

Většina instrukcí instrukčního souboru procesorů řady Ix51 obsahuje operandy:

4.10.1 Přímá (direct) adresa

Pokud je na místě operandu přímá adresa, je možno použít buď numerickou hodnotu nebo hodnotu definovanou rezervovaným jménem speciálního funkčního registru (viz. kap. 2.1). Stejně pravidlo platí i pro bitový operand s tím, že jako symbolických jmen se používá názvů přímo adresovatelných bitů podle zavedené mnemoniky definovaných v souboru rezervovaných jmen kompilátoru nebo symbolických jmen definovaných pseudoinstrukcí BIT.

4.10.2 Přímá data (#)

Je-li jako operand uvedena přímá hodnota (#data), je možno použít výrazu, jehož vyčíslením se tato hodnota získá. Ve výrazu se mohou vyskytovat čísla, symbolická jména a rezervovaná jména představující konkrétní hodnoty.

Příklad:

```
JMP $+300h  
RED EQU 10h  
MOV A, #2*RED-2
```

4.10.3 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 1000h,2589 ;definice tabulky  
;zde je definována tabulka konstant
```

4.11 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 MOV A,KLV  
019C 540F ANL A,#00001111B
```

4.11.1 Chybová hlášení

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

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

Pokus 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.

4.11.2 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 **MOV A**, vygeneruje chybové hlášení: **Řádek 1: chyba, nalezeno [konec řádku], očekává se [A, direct, SFR, bitový SFR, Rx, @Ri, data]**. Kromě toho se mohou vyskytnout další chybová hlášení. Všechna jsou generována do okna protokolu o překladu:

- **chybné číslo bitu:** výraz za tečkou není v rozsahu 0..7
- **chybná bitová adresa:** bazová adresa je mimo povolené adresy pro tečkovou metodu adresace bitů.
- **nepřímá adresa může být pouze @R0 nebo @R1:** pokus o nepřímé adresování jiným registrem než R0 nebo R1
- **relativní skok mimo rozsah:** příliš dlouhý skok, nejčastěji u podmíněných skoků
- **skok mimo aktuální stránku:** chyba v instrukci AJMP nebo ACALL
- **aktuální hodnota PC je větší než požadovaná:** hodnota za pseudoinstrukcí ORG je vyšší než aktuální hodnota čítače adres.
- **nalezena neočekávaná pseudoinstrukce IF:** kompilátor našel vnořené IF v dosud neukončené rozhodovací struktuře.
- **nalezena neočekávaná pseudoinstrukce ELSE:** kompilátor našel pseudoinstrukci ELSE bez úvodu rozhodovací struktury definovaného pseudoinstrukcí IF.
- **nalezena neočekávaná pseudoinstrukce ENDIF:** kompilátor našel pseudoinstrukci ENDIF bez úvodu rozhodovací struktury definovaného pseudoinstrukcí IF.
- **hodnota mimo rozsah 0..255:** výsledek výrazu je větší než cílový operand např. MOV A,#256
- **hodnota mimo rozsah 0..65535:** výsledek výrazu je větší než cílový operand např. MOV DPTR,#65536
- **hodnota mimo rozsah 0..3:** chyba v deklaraci použité sady registrů např. USING 4
- **očekává se operand:** chybí operand např.: MOV A,#1+
- **chyba v konstantě:** chyba v číselné konstantě např.: MOV A,#00000002B
- **dělení nulou:** při vyhodnocování výsledku výrazu našel kompilátor na pravé straně operátoru / (děleno) hodnotu nula.
- **nalezen neznámý identifikátor:** kompilátor našel 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. MOV A,1+R0
- **očekává se ':** neukončený textový řetězec
- **neočekávaný operátor:** typicky chybí operand např. MOV A, MOD 5
- **program je větší jak 64kB**
- **nelze použít v aktuálním segmentu:** chybné použití rezervace paměti
- **hodnota je mimo rozsah aktuálního segmentu:** chybné použití rezervace paměti
- **požadovaný adresový prostor nebo jeho část je již rezervován(a):** chybné použití rezervace paměti
- **nelze rezervovat nulovou velikost adresového prostoru:** chybné použití rezervace paměti
- **neočekávaná pseudoinstrukce**

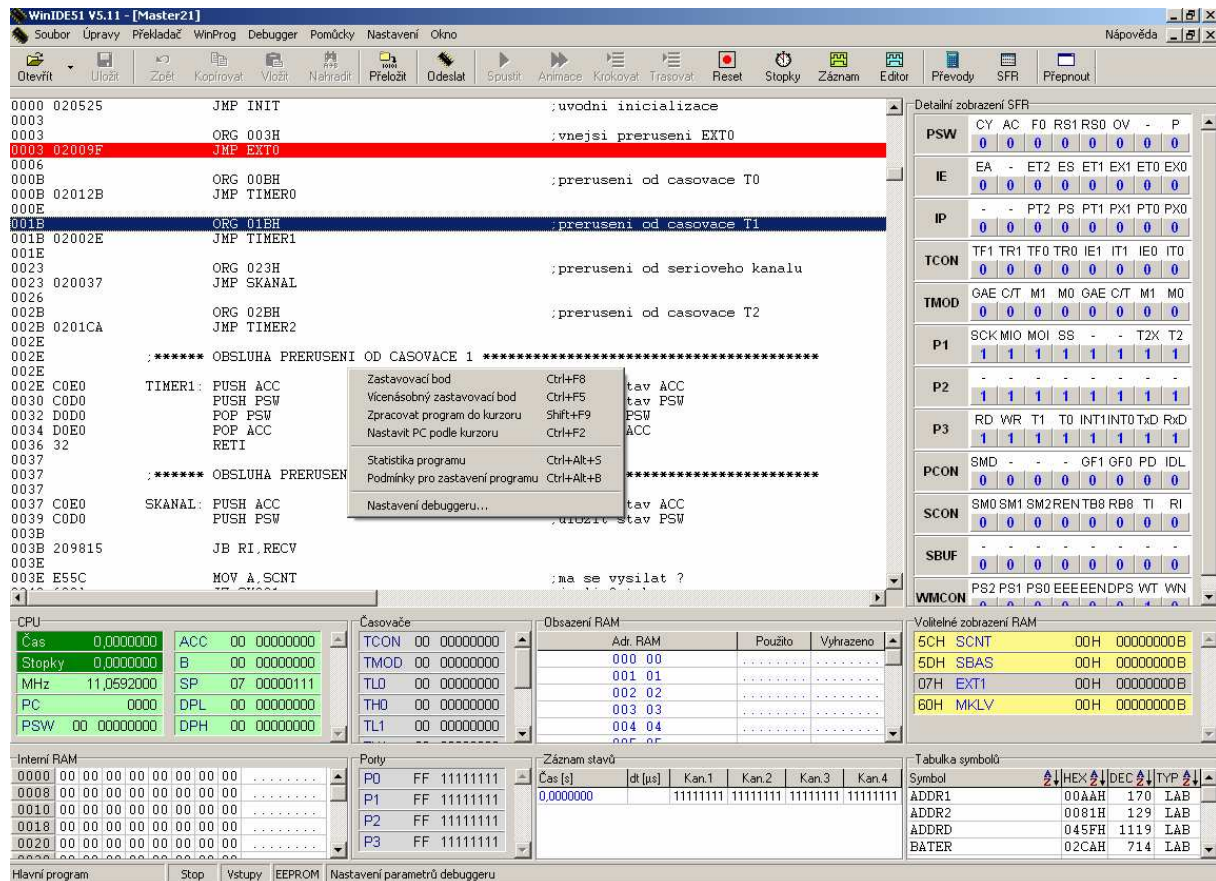
4.12 Varovná hlášení

Pro usnadnění odladování 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.

- **používá se nedeklarovaná přímá adresa:** pro zvýšení přehlednosti programu je vhodné pojmenovat všechny používané adresové místa pomocí pseudoinstrukce DATA, BIT nebo DS, DBIT. Napíšeme-li např. MOV A,45 není to chyba, ale vhodnější zápis je MOV A,POM kde je POM DATA 45. Budeme-li tuto konvenci dodržovat snadno odhalíme častou chybu kdy chybí # v definici přímých dat. Protože instrukce MOV A,#45 má zcela jiný význam než MOV A,45
- **čtení nebo zápis v nepřístupné oblasti SFR,** adresa: instrukce se pokouší o čtení nebo zápis dat do nepřístupné oblasti SFR na specifikovanou adresu
- **skok na symbol, který není definován jako návěští:** např. instrukce JMP 123 není chybná, ale je lépe nechat absolutní adresu spočítat kompilátor. Adresa nemusí souhlasit při vložení nebo smazání některé instrukce. Správné je použít návěští JMP INIT. Rovněž tak např JMP POM, kde je POM CODE 45.
- **duplicitní deklarace adresy nebo dat:** toto hlášení oznamuje, že daná adresa byla již jednou přiřazena pomocí pseudoinstrukce EQU. Např. POM1 DATA 10, POM2 DATA 10. Chceme-li definovat např. číselné konstanty můžeme použít pseudoinstrukci SET u které se duplicita nekontroluje.

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. Zadát 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.

Dvojným 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.

Statistika programu Rz539	
Provádění hlavního programu :	0,000 %
Externí přerušení EXT0 :	0,000 %
Externí přerušení EXT1 :	0,000 %
Přerušení od časovače T0 :	0,000 %
Přerušení od časovače T1 :	0,000 %
Přerušení od časovače T2 :	0,000 %
Od sériového kanálu RI+TI :	0,000 %
Minimální hodnota SP :	7
Maximální hodnota SP :	7

5.1.3 Okno stavu CPU

CPU	
Čas	0,000000
Stopky	0,000000
MHz	11,059200
PC	0181
PSW	00 00000000
ACC	00 00000000
B	00 00000000
SP	07 00000111
DPL	00 00000000
DPH	00 00000000

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

stopky (F3). nebo Resetem programu (F4).

Interní RAM									
0058	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00
0068	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00
0078	00	00	00	00	00	00	00	00	00
0080	FF	07	00	00	00	00	00	00	00
0088	00	00	00	00	00	00	00	00	00
0090	FF	00	00	00	00	00	02	00	00

5.1.4 Výpis obsahu RAM

Pod tímto oknem se nachází **výpis paměti RAM**. Je zde možné sledovat libovolnou adresu od 00h do FFh, tedy včetně SFR. Protože u procesorů řady 8052 se paměťový prostor RAM a SFR v oblasti od 80H výše překrývá, je oblast **SFR vyznačena modrou barvou**.

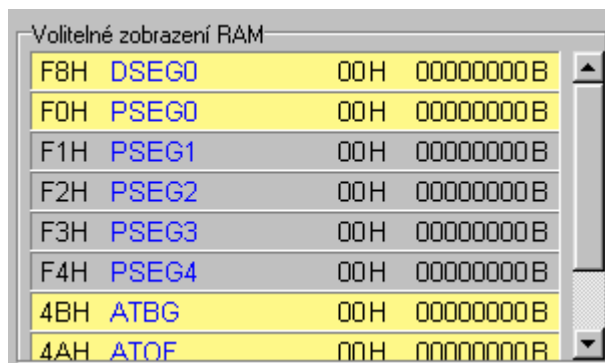
V tomto okně je možné měnit obsahy paměťových míst kliknutím na jednotlivá adresová místa. Při pohybu kurzoru nad tímto polem se zobrazuje ve stavovém řádku význam jednotlivých adresových míst, včetně SFR

5.1.5 Záznam stavů

Záznam stavů			
Čas	dt	Kan.0	K.
0,5596190	36,00	10001011	1...
0,5596550	12,00	10001011	1...
0,5596670	36,00	10001100	1...
0,5597030	12,00	10001100	1...
0,5597150	36,00	10001101	1...
0,5597510	12,00	10001101	1...
0,5597630	36,00	10001101	1...

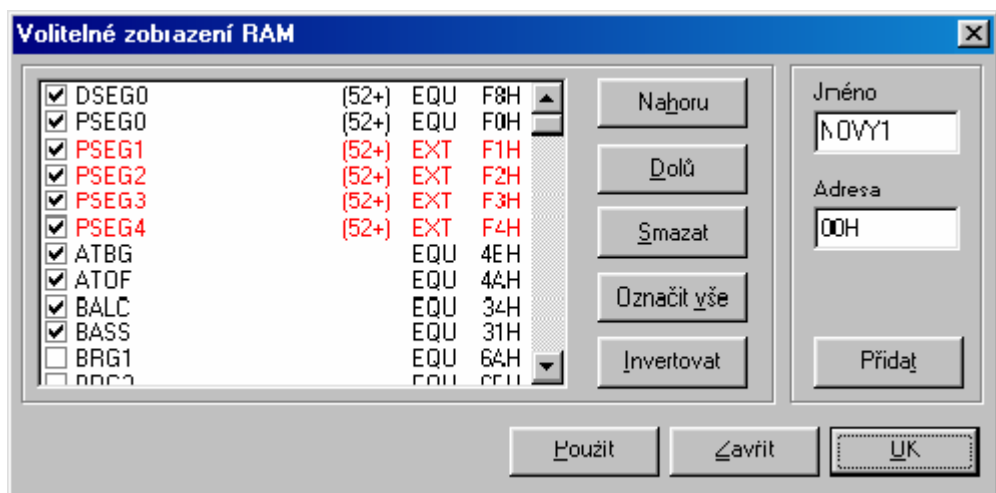
Okno Stavů portů zaznamenává veškeré události na portech nebo v paměti RAM, včetně času, kdy k této události došlo. Tento záznam je možné následně zobrazit pomocí funkce Záznam stavů.

5.1.6 Volitelné zobrazení RAM



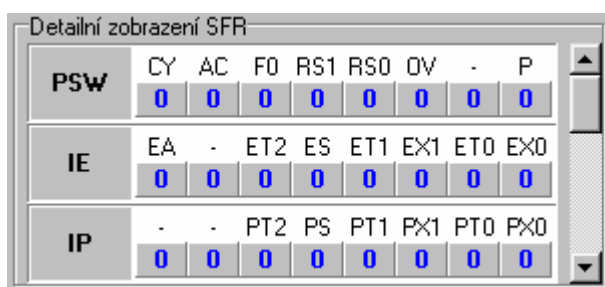
Okno Volitelné zobrazení RAM slouží ke sledování a změně obsahu libovolného paměťového místa uživatelské RAM. Chceme-li přidat nebo ubrat nějakou adresu, vyvoláme v menu *Debugger* položku *Volitelné zobrazení*. Lze také kliknout na plochu tohoto okna nebo na jméno některé již zobrazované proměnné. V okně, které se tím vyvolá, se objeví seznam všech symbolů definovaných pomocí direktivy EQU, SET nebo DATA. 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 (2DH). Lze sledovat pouze adresy v uživatelské části RAM, tzn. 0..255. Přitom adresy v horní polovině jsou přístupné pouze u procesoru Ix52. Dále je možné pomocí tlačítek *Nahoru* a *Dolů* volit pořadí zobrazovaných symbolů.

Symbole 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 SFR



Detailní zobrazení SFR umožňuje snadno pracovat po bitech s jednotlivými funkčními registry. SFR, které chceme zobrazovat, volíme v menu *Debugger-Volitelné zobrazení*. Lze také kliknout na jméno některého již zobrazovaného SFR 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 SFR se zobrazuje ve stavovém řádku název a fyzická adresa daného SFR.

5.1.8 Tabulka symbolů

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

Tabulka symbolů			
Symbol	HEX	DEC	TYP
ABC	0BH	11	NUM
ABOUT	1535H	5429	LAB
ABUDIK	079DH	1949	LAB
ACE	1CEBH	7403	LAB

5.1.9 Obsazení RAM

Tabulka ukazuje obsazení vnitřní paměti RAM. Hvězdička na daném místě udává, že daný bit se používá. Ve sloupci Použito se zobrazují všechny adresy na které bylo zapsáno nebo z nich čteno pomocí přímého adresování nebo bitových operací, a to i ty, které nemají uživatelsky definované jméno. Ve sloupci Vyhrazeno se promítají deklarace a vyhrazování paměti pomocí pseudoinstrukcí BIT, DBIT, DATA a DS.

Obsazení RAM		
Adr. RAM	Použito	Vyhrazeno
046 2E	*****	*****
047 2F	... *****	... *****
048 30	*****
049 31	*****
050 32	*****
051 33	*****

5.2 Popis funkcí debuggeru

5.2.1 Funkce Spustit

Tato funkce spouští rychlé vykonávání programu. Při běhu programu nejsou aktualizovány změny obsahu registrů, SFR, portů apod. Zobrazuje se pouze strojový čas procesoru a stopky. Na rychlejších počítačích je možno dosáhnout reálné rychlosti simulovaného procesoru. Všechny změny obsahu 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.

	Spustit	F9
	Animace	F6
	Krokovat program	F8
	Trasovat program	F7
	Reset programu	F4
	Podmínky pro zastavení programu	Ctrl+Alt+B
	Nulovat stopky	F3
	Záznam portů	F11
	Editor vstupních signálů	Ctrl+F11
	Detailní zobrazení	
	Volitelné zobrazení	

5.2.2 Funkce Animace programu

Tato funkce vykonává pomalé provádění programu po instrukci, všechna zobrazovaná paměťová místa jsou však po vykonání instrukce aktualizována. Rychlost animace je možné volit v nabídce Nastavení Debuggeru. Jinak je funkce stejná jako Spustit, nelze však nikdy dosáhnout takové rychlosti provádění programu.

5.2.3 Funkce Krokovat program

Vykoná jeden řádek programu. Pokud je tomto řádku volání podprogramu (instrukce CALL), provede se celý podprogram. Je-li vykonávání podprogramu příliš dlouhé a potřebujeme z nějakého důvodu běh programu přerušit (např. podprogram obsahuje nekonečnou smyčku), je to možné opětovným vyvoláním této funkce. Funkce Krokovat pracuje tímto způsobem: Je-li současná instrukce CALL, na následující řádek programu se umístí neviditelný breakpoint, poté se volá funkce Spustit (rychlé provádění programu) a po úspěšném vykonání podprogramu a zastavení programu se tento breakpoint smaže. Pokud tedy před tímto breakpointem přerušíme provádění podprogramu, dostaneme se do místa, kde se nachází tělo podprogramu. Chceme-li se pak rychle vrátit na zpět, na místo odkud byl podprogram volán, nesmíme pomoci Krokovat, ale vyvoláme nyní funkci Spustit a ta se zastaví na instrukci následující po CALL na neviditelném breakpointu - žlutý řádek (samozřejmě podprogram nesmí obsahovat právě nekonečný cyklus). Toto platí i v případě, že i vykonávaný podprogram obsahuje jiný breakpoint.

5.2.4 Funkce Trasovat program

Vykoná pouze jednu instrukci programu. Všechna zobrazovaná paměťová místa jsou po vykonání instrukce ihned aktualizována. Po vykonání instrukce je provádění programu zastaveno.

5.2.5 Funkce Reset programu

Nastavuje čítač programu na adresu 0000. Obnovuje výchozí hodnoty ve SFR. 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 Nulovat stopky

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

5.3 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 adrese xxxxH nebyla nalezena platná instrukce** - k tomuto stavu může dojít např. při skoku na vypočítanou adresu JMP @A+DPTR nebo při neošetřeném přerušovacím vektoru.
- **Neznámá instrukce na adrese: xxxxH** - může nastat za stejných podmínek jako předchozí chyba.
- **Zápis do nepřístupné adresy v oblasti SFR na adrese xxxxH** - chyba v adresování. Např. při MOV @Ri.
- **Čtení z nepřístupné adresy v oblasti SFR na adrese xxxxH** - stejné jako předchozí chyba.
- **Zápis do EEPROM mimo rozsah, instrukce na xxxxH** - chyba v adrese u procesoru 89S8252.

- Čtení z EEPROM mimo rozsah, instrukce na xxxxH - stejné jako předchozí chyba.

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 xxxxH splnila zastavovací podmínku.** Symbol xxxxH je konkrétní adresa instrukce, na níž došlo k chybě. Ukazatel prováděné instrukce (modrý řádek) je v tomto okamžiku již na následující instrukci.

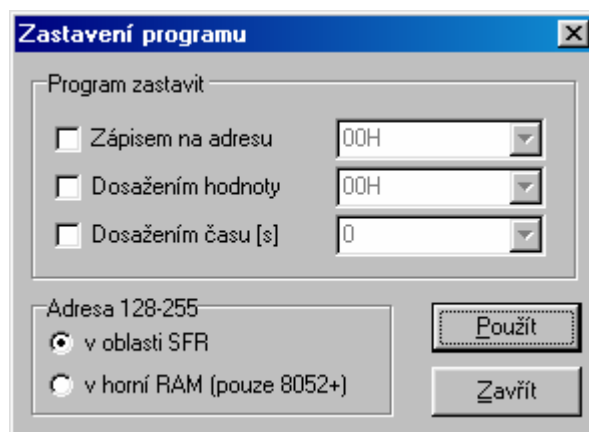
5.4 Místa přerušení.

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.

5.4.1 Zastavení zápisem na adresu

Dále je možno program 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.



5.4.2 Zastavení dosažením hodnoty

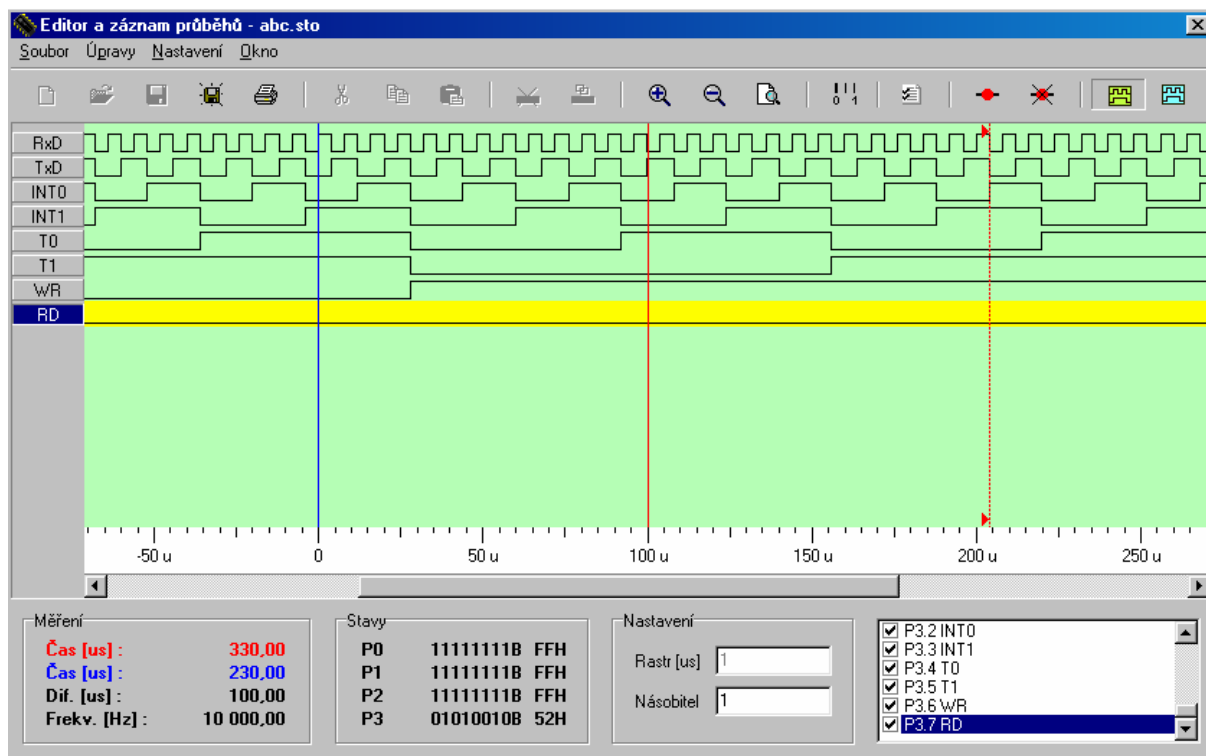
Program se zastaví až dosažením konkrétní hodnoty na požadované adrese. V tomto případě zvolíme *Zastavit dosažením hodnoty*. Obě tyto možnosti se velice dobře uplatní při hledání chyby v programu, kdy přesně nevíme, která instrukce danou adresu modifikovala.

5.5 Editor a záznam průběhů

Časový 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.

5.5.1 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 zaškrťávacím 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*.

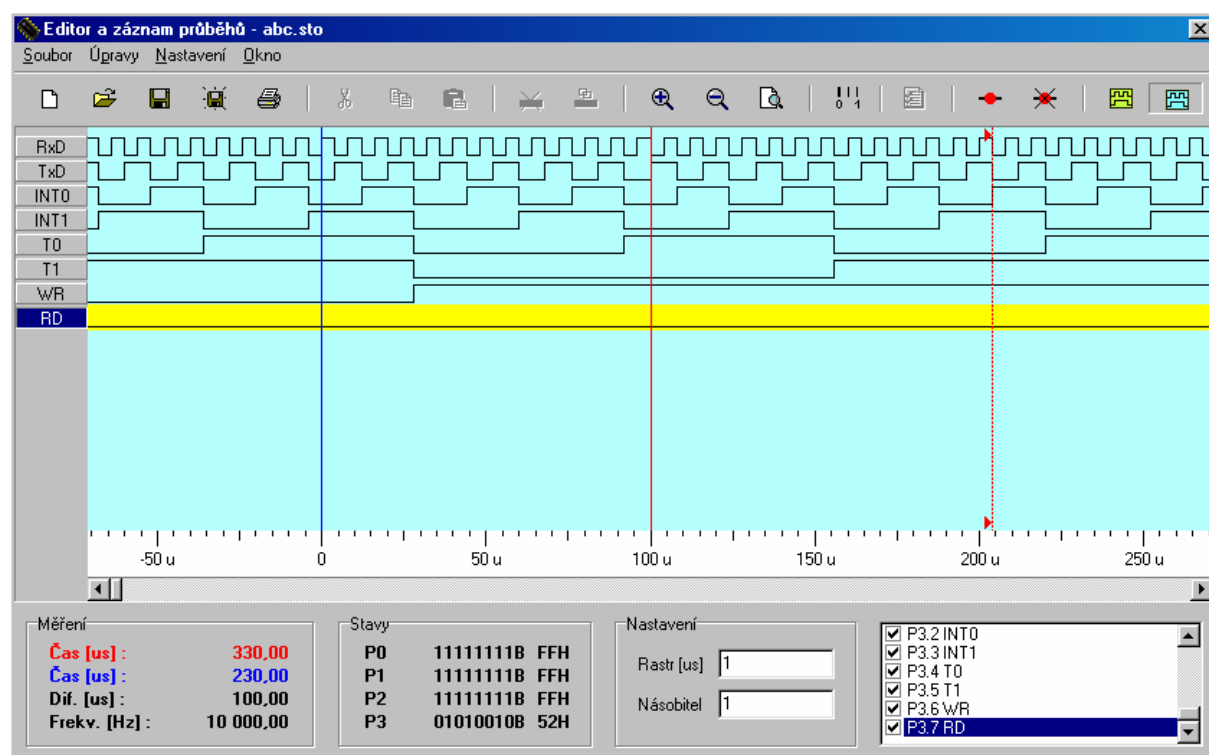
Je také možné část průběhu vložit do schránky a použít jako vstupní průběh v *Editoru průběhů*. V tomto případě dojde k automatickému přepočítání časových jednotek, tak aby průběh byl časově správný i v editoru. K tomu je třeba kliknutím na tlačítko na levé liště zaktivovat jeden příslušný bit a pomocí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

5.5.2 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 zaškrťovacím 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 pohledu 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 zaškrťovacím 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 *Vložit*, *Vyjmout*, *Zkopírovat* pracují se schránkou podle zvyklostí Windows.

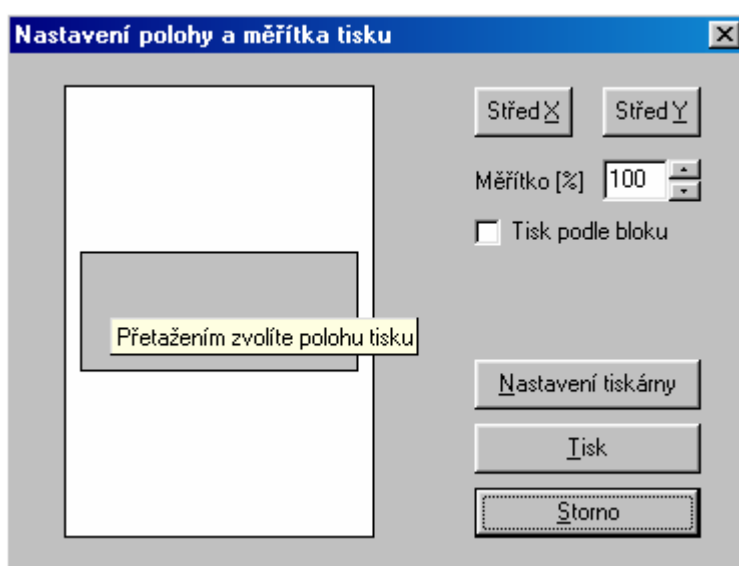
Seznam zkratkových kláves:

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
ŠÍPKY	posun kurzoru, kreslení průběhu
SHIFT	jemný posun kurzoru
CTRL	rychlý posun kurzoru

5.5.3 Tisk průběhů

Pro snadný tisk editovaných i zaznamenaných 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.

Tlačítka *StředX* a *StředY* je možné rychle vyrovnat tisknutý průběh na středy papíru. Dále možné zvolit měřítko tisku.



5.6 Simulace vnější paměti RAM a paměti EEPROM

5.6.1 Vnější paměť RAM

Debugger umožňuje používat instrukce MOVX pro práci s vnější pamětí dat, tak jako kdyby byla připojena externí paměť o velikosti 64kB. Obsah paměti se nemění ani překladem ani resetem programu. Pouze při spuštění programu WinIDE51 se její obsah vynuluje. Není možné ani editace a zobrazování obsahu vnější RAM. Obsah této paměti se ani neukládá do souboru ani nenačítá.

5.6.2 Paměť EEPROM

Procesory řady 89S8252 mají integrovanou vnitřní paměť EEPROM o velikosti 2kB. Debugger umožňuje její využití. Je-li povolen zápis do této paměti nastavením příslušných bitů, je možné pomocí MOVX s touto pamětí pracovat. Je simulován i relativně pomalý zápis do této paměti, tím že bit RDY v registru WMCON se nastaví asi za 2,5ms, což je typická doba zápisu do EEPROM. 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.

1	Úvod.....	2
1.1	Využití WinIDE51	2
2	Popis aplikace.....	3
2.1	Titulní lišta	3
2.2	Hlavní nabídka	3
2.3	Panel nástrojů	3
2.3.1	Menu Soubor	3
2.3.2	Menu Úpravy.....	4
2.3.3	Menu Překladač.....	4
2.3.4	Menu WinProg	4
2.3.5	Menu Pomůcky	5
2.3.6	Menu Nastavení.....	6
2.3.7	Menu Okno.....	7
3	Textový editor	8
3.1	Popis editoru.....	8
3.1.1	Lišta řádků.....	8
3.1.2	Pracovní plocha	8
3.1.3	Protokol o překladu	9
3.1.4	Použité symboly	9
3.1.5	Stavový řádek	9
3.2	Plovoucí nabídka	9
3.2.1	Parametry instrukce.....	9
3.2.2	Najít deklaraci	10
3.2.3	Zastavovací body.....	10
4	Kompilátor	11
4.1	Úvod.....	11
4.2	Popis syntaxe jazyka	11
4.3	Rezervovaná jména	11
4.3.1	Seznam rezervovaných jmen a jejich hodnot.....	11
4.3.2	12
4.3.3	Seznam rezervovaných symbolů.....	12
4.4	Symbolická jména definovaná programátorem.....	12
4.5	Konstanty	13
4.5.1	Číselné konstanty	13
4.5.2	Znakové konstanty	13
4.5.3	Symbol \$	13
4.6	Operátory.....	14
4.6.1	Operátory +, -, *, /	14
4.6.2	Operátor MOD	15
4.6.3	Operátor OR, XOR, AND	15
4.6.4	Operátor NOT	15
4.6.5	Operátor NOTW.....	15
4.6.6	Operátor SHL (SHift Left)	16
4.6.7	Operátor SHR (SHift Right).....	16
4.6.8	Operátor LOW().....	16
4.6.9	Operátor HIGH()	17
4.7	Formát zdrojového textu	17
4.7.1	Návěští.....	17
4.8	Pseudoinstrukce.....	17

4.8.1	Definice hodnot symbolických jmen - EQU, SET	18
4.8.2	Výběr segmentu – CSEG, BSEG, DSEG, ISEG, XSEG	18
4.8.3	Definice symbolických jmen bytových adres - DATA, IDATA, XDATA..	19
4.8.4	Definice symbolických jmen bitových adres - BIT	19
4.8.5	Definice hodnoty adresového symbolu - CODE.....	20
4.8.6	Rezervování paměti v bytech - DS (Define Space).....	20
4.8.7	Rezervování bitové paměti - DBIT (Define Bit).....	20
4.8.8	Definice 8 bitových konstant v paměti programu - DB (Define Byte)	20
4.8.9	Definice 16 bitových konstant v paměti programu - DW (Define Word)....	21
4.8.10	Definice textů v paměti programu - TEXT	21
4.8.11	Nastavení programového čítače - ORG	21
4.8.12	Nastavení banky registrů – USING.....	21
4.8.13	Podmíněný překlad - IF, ELSE, ENDIF	22
4.8.14	Ukončení zdrojového textu - END.....	22
4.8.15	Vložení externího souboru - \$INCLUDE, \$UNIT.....	22
4.8.16	Frekvence krystalu v debuggeru - XTAL	23
4.9	Instrukce	23
4.10	Operandy	23
4.10.1	Přímá (direct) adresa	23
4.10.2	Přímá data (#)	23
4.10.3	Komentář	23
4.11	Protokol o překladu a chybová hlášení	24
4.11.1	Chybová hlášení	24
4.11.2	Chyby při překladu	24
4.12	Varovná hlášení	26
5	Debugger	27
5.1	Popis oken debuggeru	27
5.1.1	Výpis po překladu	27
5.1.2	Statistika programu	28
5.1.3	Okno stavu CPU.....	28
5.1.4	Výpis obsahu RAM.....	28
5.1.5	Záznam stavů.....	28
5.1.6	Volitelné zobrazení RAM	29
5.1.7	Detailní zobrazení SFR	29
5.1.8	Tabulka symbolů	30
5.1.9	Obsazení RAM	30
5.2	Popis funkcí debuggeru	30
5.2.1	Funkce Spustit	30
5.2.2	Funkce Animace programu	30
5.2.3	Funkce Krokovat program	31
5.2.4	Funkce Trasovat program.....	31
5.2.5	Funkce Reset programu.....	31
5.2.6	Funkce Nulovat stopky.....	31
5.3	Chyby při spuštění programu	31
5.4	Místa přerušení.	32
5.4.1	Zastavení zápisem na adresu	32
5.4.2	Zastavení dosažením hodnoty	32
5.5	Editor a záznam průběhů.....	32
5.5.1	Záznam průběhů	33
5.5.2	Editor vstupních průběhů	34

5.5.3	Tisk průběhů.....	35
5.6	Simulace vnější paměti RAM a paměti EEPROM.....	36
5.6.1	Vnější paměť RAM.....	36
5.6.2	Paměť EEPROM.....	36