

MCU 8051 IDE handbook draft

Martin Ošmera <martin.osmera@gmail.com>

December 28, 2012

I would like to thank to the following people for their support during the project development:

- **Andre Cunha** (*Brazil*) for review of this document.
- **Yuanhui Zhang** (*China*) for bug reports and help with debugging.
- **Kara Blackowiak** (*USA*) for certain code reviews.
- **Marek Nožka** (*Moravia, CZ, EU*) for help with debugging.
- **Kostya V. Ivanov** (*Russia*) for bug fixes in the simulator engine.
- **Shakthi Kannan** (*India*) for adding this software to the FEL project and for a few patches.
- **Trevor Spiteri** for help with debugging (patches) the HD44780 simulator.
- **Miroslav Hradílek** (*EU*) for bug reports and suggestions
- **Fabricio Alcalde** (*Argentina*) for suggestions and bug reports.
- **Francisco Albani** (*Argentina*) for suggestions and a few bug reports.

Contents

Preface	5
Goals of the project	5
Requirements	6
Intended Audience	7
1 Brief introduction	9
1.1 Main components of MCU 8051 IDE	9
1.2 What is MCS-51	12
1.3 What is the Assembly language	12
2 Quick start	13
2.1 Demonstration project	13
2.2 Your first project in MCU 8051 IDE	14
3 Detailed introduction to GUI	15
3.1 Source code editor	15
3.1.1 Syntax highlight and validation	15
3.1.2 Spell checking	15
3.1.3 Auto-completion	16
3.1.4 Editor command line	16
3.2 Bottom panel	18
3.2.1 Main panel of the MCU simulator	18
3.2.2 C variables	18
3.2.3 Graph showing voltage levels	19
3.2.4 Messages panel	19
3.2.5 Notes	20
3.2.6 Calculator	20
3.2.7 Find in files	21
3.2.8 Terminal emulator	21
3.3 Left panel	22
3.3.1 List of opened files	22

3.3.2	List of project files	22
3.3.3	SFR watches	22
3.3.4	File system browser	22
3.4	Right panel	22
3.4.1	List of bookmarks	22
3.4.2	List of breakpoints	23
3.4.3	Instruction details	23
3.4.4	Data register watches	23
3.4.5	Subprograms call monitor	24
3.4.6	List of symbols	24
3.4.7	HW plug-ins manager	24
3.5	Other tools	25
3.5.1	SFR map	25
3.5.2	Map of bit addressable area	25
3.5.3	Stack monitor	25
3.5.4	Symbol viewer	26
3.5.5	ASCII chart	27
3.5.6	8051 Instruction Table	27
3.5.7	8-segment editor	27
3.5.8	Stopwatch	27
3.5.9	Scribble notepad	27
3.5.10	Base converter	28
3.5.11	RS-232 debugger	28
3.5.12	Hexadecimal editors	29
3.5.13	Hibernation of simulated program	30
3.5.14	Interrupt monitor	30
3.5.15	Conversions between *.hex, *.bin and *.adf files	31
3.5.16	Normalization of source code indentation	31
3.5.17	Change letter case	31
3.5.18	User defined commands	32
3.5.19	Clean-up project folder	33
3.5.20	File statistic	33
3.6	Configuration dialogues	33
4	Build-in macro-assembler	37
4.1	Statements	37
4.2	Symbols	37
4.3	Constants	38
4.4	Expressions	39
4.5	The instruction set processing	40
4.6	Assembler directives	41

4.7	Assembler Controls	45
4.8	Predefined Symbols	47
4.9	Segment type	49
4.10	Conditional Assembly	51
4.11	Macro Processing	52
4.12	Reserved keywords	56
4.13	Compatibility with ASEM-51	57
4.14	List File Format	58
4.15	Specification of Intel®8 HEX Format	60
5	Disassembler	61
6	MCU simulator	63
6.1	Short introduction	63
6.2	Modes of simulation	63
6.3	Warning conditions	64
6.4	Limitations	64
6.5	Virtual hardware	65
6.5.1	DS1620 temperature sensor	65
6.5.2	File interface	65
6.5.3	LED Panel	66
6.5.4	Single LED Display	66
6.5.5	Multiplexed LED Display	66
6.5.6	LED Matrix	67
6.5.7	Matrix Keypad	67
6.5.8	Simple Keypad	68
6.5.9	LCD display controlled by HD44780	68
7	Writing hardware tool control plug-ins	69
7.1	Foreword	69
7.2	How to write your own plug-in	70
7.3	Using MCU 8051 IDE API	71
7.4	A basic example	72
7.5	Random remarks	73
8	Command Line Interface	75
9	Translating the IDE into different languages	77
A	License	79

B Regression testing	81
B.1 Foreword	81
B.2 More about the implementation	81
C Project web page and other media	83
C.1 Official project web page	83
C.2 Other media	84
C.3 GIT repository	84
D 8051 Instructions in numerical Order	85
E 8051 Instructions in alphabetical order	91
F List of supported micro-controllers	97
F.0.1 Intel®	97
F.0.2 Atmel®	97
G Change log	99

Preface

Goals of the project

MCU 8051 IDE is an integrated development environment for microcontrollers based on MCS-51 intended for Assembly language and C language. This IDE is currently available on GNU/Linux and Microsoft®Windows® (since version 1.3.6). This program was originally intended for education purposes, but now the area of potential usage is surely significantly wider. This program was created to fill a gap in the open source software of this kind. User interface, source codes, documentation, web pages, etc., are written in English in order to make this software available to as many user as possible, but there is support for internationalization using `i18n` since version 1.3.10. This documentation is written in \LaTeX . It is very important to note that this software was not developed for any company, person or something similar and it is completely noncommercial, open source software distributed under [GNU GPLv2](#) license intended for a group of people with common interest, in this case 8051.

MCU 8051 IDE should offer:

- ① A transparent view on a simulated program for 8051;
- ② Easy source code editing even for an user with small knowledge of the assembly language;
- ③ User friendly advanced IDE for MCS-51.

List of the most important parts of MCU 8051 IDE:

- ☞ Source code editor;
- ☞ Optimization capable macro-assembler;
- ☞ Advanced MCU simulator;
- ☞ Hexadecimal editor;

- ☞ Interface for hardware tool control plug-ins;
- ☞ Scientific calculator and special calculator optimized for 8051.

Requirements

Hardware requirements are not defined. This program should run without problems on all POSIX systems (like GNU/Linux, etc.), where all of the software dependencies were satisfied. The IDE is officially distributed as a source code package (primary programming language is TCL), RPM package (currently available in official RHEL repositories), DEB package (currently available in official Debian repositories) and ebuild for Gentoo Linux (currently NOT available in the portage tree).

Package	Min. version	Download location
Required packages:		(The IDE will not run without these packages)
tcl	8.5	http://www.tcl.tk/software/tcltk/downloadnow85.html
tk	8.5	http://www.tcl.tk/software/tcltk/downloadnow85.html
bwidget	1.8	http://sourceforge.net/projects/tcllib
itcl	3.4	http://sourceforge.net/projects/incrtcl
tdom	0.8	http://www.tdom.org
tkimg	1.3	http://sourceforge.net/projects/tkimg
tcllib	1.6	http://sourceforge.net/projects/tcllib
Optional packages:		(Functionality might be unnecessarily limited without these packages)
Telx	8.4 <i>(Signal handling (signals like SIGINT))</i>	http://tclx.sourceforge.net
cmake	2.6 <i>(If you prefer this way of installation: “./configure && make && make install”)</i>	http://www.cmake.org/HTML/Download.html
rxvt-unicode	8.3 <i>(If you want terminal emulator)</i>	http://software.schmorp.de
asem-51	1.3 <i>(If you want to use a really good assembler :))</i>	http://plit.de/asem-51/download.htm
sdcc	2.9 <i>(If you want to used C language compiler)</i>	http://sdcc.sourceforge.net/
doxygen	1.3 <i>(If you want to use doxygen directly from the IDE)</i>	www.doxygen.org/
indent	1.2 <i>(If you want to use auto-indent function for C language)</i>	http://www.gnu.org/software/indent/
hunspell	1.2 <i>(If you want to have spell checker function available)</i>	http://hunspell.sourceforge.net
bash	4.0 <i>(If you want to have spell checker function available)</i>	http://tiswww.case.edu/php/chet/bash/bashtop.html
gawk	3.1 <i>(If you want to have spell checker function available)</i>	http://www.gnu.org/software/gawk/

Table 1: Software requirements

Intended Audience

This manual is intended for any individual, regardless of his or her experience with assembler, C language, MCU 8051 IDE or Linux, but it is assumed here that the reader is familiar with basic concepts of assembly language programming and with 8051 processor architecture. Advanced users are not likely to read this manual, but all suggestions on documentation will be considered. If you would like to contribute to this project or the documentation, please consult the project web page.

Thanks for your cooperation which helps to make this software better.

Chapter 1

Brief introduction

This chapter will provide you with a brief introduction about the main components that are part of MCU 8051 IDE. The purpose of this chapter is to contextualize you on the software, informing about the parts that composes it. The next chapter will cover rapidly the Graphical User Interface, which will be described in further details on chapter.

1.1 Main components of MCU 8051 IDE

Editor The code editor is featured with syntax highlighting and validation, auto-completion and spell checking for comments ¹, as well as a command line that speeds up the access to various editor options. It also provides a panel showing line numbers, bookmarks, breakpoints and warnings from syntax validator. Editor is capable to export the source code within it as XHTML and L^AT_EX and contains a number of useful tools like automatic indentation, searching and replacement of expressions, copy to clipboard, paste from clipboard, among others.

Assembler The assembler is one of the integral parts of MCU 8051 IDE. It is a macro assembler with support for dozens of directives and capable of performing peephole optimizations. Support for peephole optimizations means that the assembler can attempt to optimize the resulting code for higher execution speed and lower size without tempering with its very functionality. It is important to note that automatic peephole optimization can sometimes be harmful and so it is disabled by default. A macro assembler is a software that allows the user to define a **macro instruction**, which consists of a sequence

¹Spell checking for comments is available only if you have installed the Hunspell program. This feature is currently not available on MS[®]Windows[®]OS.

of basic instructions, and use it later instead of repeatedly copying and pasting the set of instructions over and over along the source code. Assembler behavior can be configured either globally, using the proper configuration dialog, or locally in source code, by means of assembler directives and control sequences (e.g. `$TITLE('Some title to show in the code listing')`). The assembler is capable of generating four kinds of output code:

- ☞ Object code (machine code) as a hexadecimal file, with `.hex` extension and in Intel® 8 HEX format;
- ☞ Object code (machine code) as a binary file, with `.bin` extension and in format of raw binary data;
- ☞ Code listing, in `.lst` extension;
- ☞ Code for integrated MCU simulator, in `.adf` extension.

Simulator The simulator is a software component intended for the simulation of the chosen microcontroller in a virtual environment. It allows user to monitor precisely what is happening in the MCU in an exact moment in time, as well as to modify its components, for instance by altering the value of a register, canceling an interrupt or forcing a subprogram to return. In that way it might be possible to ferret out certain flaws in the program being debugged, which would be hard or nearly impossible to find and/or fix in other ways. Even though it is better to have ICD (In-Circuit Debugger) or ICE (In-Circuit Emulator) at hand, MCU 8051 IDE in current version does not support neither of them MCU simulator implemented in this IDE supports dozens of microcontrollers and most of them are treated in slightly different way allowing to take into account most of the nuances between the supported MCUs. User can adjust simulator behavior to fit his or her needs by modifying clock frequency, size of connected external code, data memory and others, or for instance by disabling or enabling certain warnings, which pops up when the simulated program do something “strange”, like some kind of invalid access into memory or stack overflow or underflow. Besides that, it is possible for the user to modify all registers which the MCU deals with, including those which are not accessible by the running program, like the Program Counter. User have always an overview of all possible, pending and active interrupts and can temper with them at any time. The simulator also allows for altering code memory and all kinds of data memories. The program being simulated can be at any time "hibernated" into a file, preferably with `.m51hib` extension, and resumed later from this same file. Such a file contains the entire state of the simulator at the point in which the program was hibernated.

Project management It is a functionality that allows the IDE to remember certain set of source code files along with a set of configuration parameters. Projects are stored in XML (eXtensible Markup Language) files with extension `.mcu8051ide`. These files are human readable and their precise formatting is described in their inline DTD (Document Type Declaration). Their encoding is UTF-8 (Unicode Transformation) and as EOL (End Of Line) character they use LF (Line Feed). The reason for that is to make it possible for the user to implement his or her own tools for manipulating with them.

Scientific calculator MCU 8051 IDE scientific calculator is implemented as a simple scientific calculator capable of computation in four number systems: hexadecimal, decimal, octal and binary, and with three angle units: radians, degrees and grad. Integral part of the calculator is also a simple tool intended solely for computing preset values for MCU timers.

Special calculator The experience in MCU programming shows that it is very useful to have some tools at hand, capable of performing recurrent boring calculations that spend time to be done by hand. MCU 8051 IDE special calculator is intended for performing certain simple specialized calculations related to 8051. For instance, this calculator is capable of generating assembly language code implementing a wait loop with specified parameters.

Hexadecimal editor This utility is used here for watching and modifying large blocks of raw data in various memory types of the simulated MCU (Code, IDATA, XDATA, EEPROM, etc.). There is also hexadecimal editor intended for editing Intel® HEX 8 files. Other hexadecimal editors are specially designed to fit specific needs of the given purpose; for example, there is an hexadecimal editor for viewing and editing code memory, which displays the current position of the program counter in the machine code of the simulated program.

Disassembler This tool can translate once assembled code back to source code. It is important to note that it is somewhat improbable that the resulting source code will look "reasonable" It is due to DB and DW and not fixed instruction word length on 8051. Nevertheless, such a generated source code must possess exactly the same functionality when it gets assembled again. Disassembler implemented in this IDE is frankly speaking only a little more than just a toy. If you want a really capable disassembler, maybe you should try some tool like D52 <http://www.8052.com/users/disasm/>.

Notepad In this IDE, it is a simple rich text editor for writing user notes of whatever kind. Originally, it was intended for writing a list of things which remain to be done in your project.

Command Line Interface (CLI) It is an useful tool that allows the use of some IDE functions without entering it's GUI. You can get list of available options by typing `mcu8051ide -h` or `mcu8051ide -help` to your terminal emulator. You can, for example, use just the assembler of the IDE or convert an Intel® HEX 8 file to a raw binary file.

1.2 What is MCS-51

The Intel MCS-51 is a Harvard architecture, single chip microcontroller series which was developed by Intel in 1980 for use in embedded systems. Today there is a vast range of enhanced 8051-compatible devices manufactured by a number of independent manufacturers. They have 8-bit ALU, accumulator and 8-bit Registers (hence they are an 8-bit microcontrollers), they have 16-bit address bus, 8-bit data bus, boolean processing engine which allows bit-level boolean logic operations to be carried out directly and efficiently on select internal registers and select RAM locations, etc.

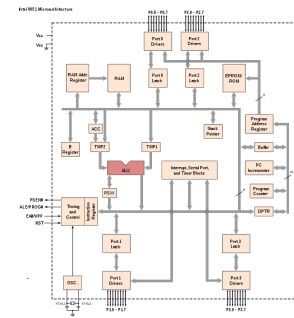


Figure 1.1: i8051 micro-architecture

1.3 What is the Assembly language

An assembly language is a low-level programming language for computers, microprocessors, microcontrollers and other integrated circuits. It implements a symbolic representation of the binary machine codes and other constants needed to program a given CPU architecture. Processors based on MSC-51 have compatible instruction set, similar registers and many other things are generally very similar among them.

Here is an example of how a piece of 8051 assembly code looks like:

Code 1 An example piece of code written in 8051 assembly language

```

main:
if test=2
    mov    R0, #25h
        ; Configure EEPROM
    orl    EECON, #38h
    inc    R0
endif
XOMI:
    anl    EECON, #(OFFh - 020h)
    movx  @R0, A

```

Chapter 2

Quick start

2.1 Demonstration project

The aim of the demonstration project is to provide an easy way to explore the IDE without reading long and boring documents like this one. :) The demonstration project can be opened from the welcome dialog (“Main Menu” → “Help” → “Welcome dialog” → “Open demonstration project”). Demonstration project should introduce new user into usage of the most common functions of the IDE like assembling the code, running simulator and so on. Demonstration project cannot be modified by the user in order to make it “less volatile”.

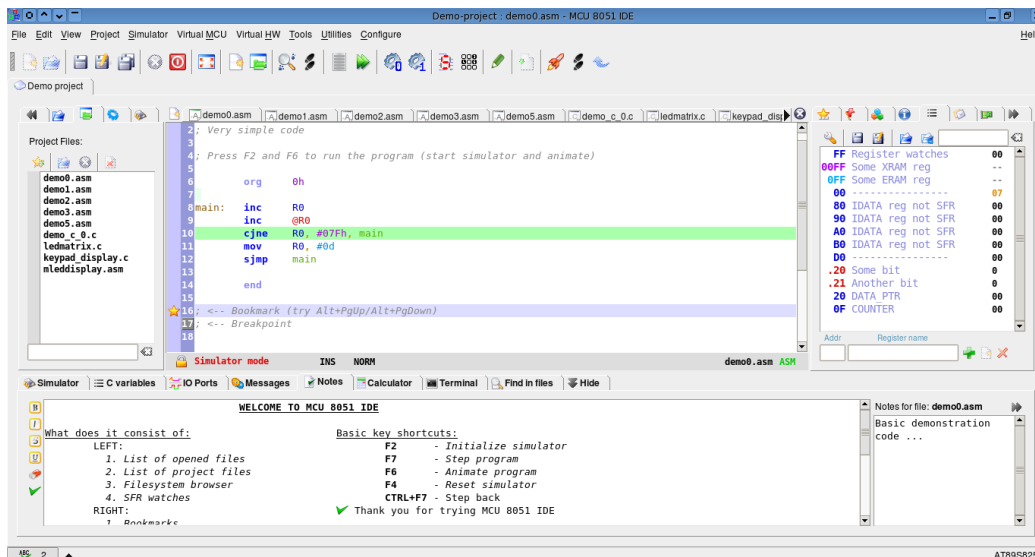


Figure 2.1: MCU 8051 IDE with the demonstration project opened within it

2.2 Your first project in MCU 8051 IDE

At first let me explain what the MCU 8051 IDE's project really is. It is a set of some files in some directory, let's call this directory the project directory. And this along with the file with extension `.mcu8051ide` forms the project. The file with `.mcu8051ide` extension defines what source code files belongs to the project and contains additional information about the project, like who is the project author or for what exact MCU is the project intended.

To create you project in you have to specify the project directory and the MCU type for which you will develop your code. This is done in project creation dialog. This dialog can be accessed from main menu "Main Menu" → "Project" → "New". After this step you can specify some additional information about the project in project editing dialog.

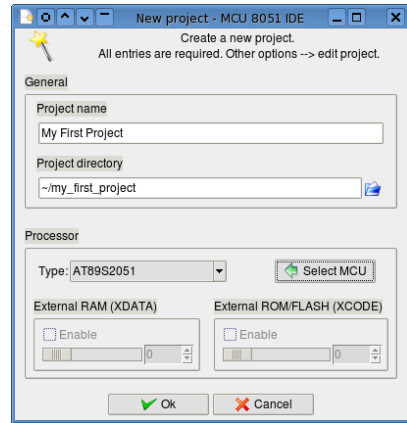


Figure 2.2: Project creation dialog

Once you have created a new project you can begin to develop you code from your chosen processor. When you want to save your code press `Ctrl+S`, `Ctrl+N` creates a new file and an existing file can be opened by `Ctrl+O`. Each opened file can be added or removed to/from your current project. `Ctrl+B` creates or deletes bookmark and `Ctrl+Shift+B` creates or deletes breakpoint. Project files, the files which are parts of the project, are opened each time you open the project. You can have more than one project opened at the time.

Simulator can be started and shut down by pressing `F2` key and assembler or compiler is run when `F11` is pressed. Output from assembler or compiler is displayed on the bottom panel in tab "Messages". And main MCU simulator panel is also available on the bottom panel in tab "Simulator".

On the left side you can find list of currently opened source code files and list of project files. And on the right side probably most useful tool at the beginning might be "Instruction details", this tool displays help for instruction in the code editor on line with cursor. In the right panel you can find for example also list of bookmarks and breakpoints.

Chapter 3

Detailed introduction to GUI

3.1 Source code editor

3.1.1 Syntax highlight and validation

The editor is equipped with an implementation of a syntax highlighting algorithm based on simplified syntax analysis. And that enables a limited on-line syntax validation. That means that as the user writes down the code, editor tries to check it for syntactical correctness. Syntax validator marks “strange looking” lines with exclamation mark and tries to underline exact point of potential syntax errors. This feature can be disabled as well as syntax highlighting can be disabled. By disabling these features you can make the editor work faster, but it would probably mean only a unnecessary limitation. There are three levels of syntax validation:

- 0: Disabled
- 1: Fast basic validation
- 2: Slow advanced validation

Syntax validation configuration button react to left and right click with the mouse pointer. Right button click decreases the level of validation and the left button click increases it.

3.1.2 Spell checking

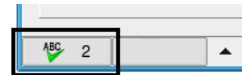


Figure 3.1: Syntax validation configuration button

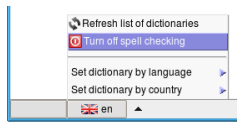


Figure 3.2: Spell checker configuration button

There is also configurable spell checking function available. It underlines words which are marked by Hunspell¹ as incorrectly spelled. This function applies to comments in the code or the entire code in case that the syntax highlight function has been disabled. User can choose from any of Hunspell or Myspell dictionaries installed on his or her system. This feature can also be turned off. It makes sense that this function is

completely dependent on the Hunspell program, if it is not installed, spell checking won't work here.

3.1.3 Auto-completion

Pop-up based auto-completion is function which should make it easier to use long names for labels, macros, variables, functions, constants, etc. This function is interconnected with syntax editor's analyzer used for syntax highlight and validation and for the table of symbols in the right panel. So it maintains an overview of all symbols defined in your source code file and then when you write just a few characters which a symbol starts with, this function will pop-up window offering you all defined symbols beginning with that letters. Note that this feature can be disabled in editor configuration dialog and note also that besides symbols it offers also list of assembly language instruction mnemonics and assembler directives.

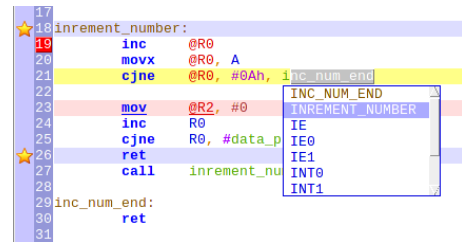


Figure 3.3: Syntax highlight, syntax validation and the pop-up based auto-completion all in action

3.1.4 Editor command line

Editor is featured with a command line, which can be invoked by pressing F10 key by default. The command line appears below the editor above its status bar. From the command line you can perform variety of operations like conversions between various numerical bases, run simulator, insert current date and many more. In the command line it is sufficient to write just a few characters which the requested command starts with and which are sufficient to uniquely identify the command and press enter. You can see help for each

¹Hunspell is a spell checker and morphological analyzer. See <http://hunspell.sourceforge.net> for details.

command by running command `help list`. Command line is featured with its own color highlight, history and auto-completion.

Command	Arguments	Description
d2h	<decimal number>	Convert decimal number into hexadecimal
d2o	<decimal number>	Convert decimal number into octal
d2b	<decimal number>	Convert decimal number into binary
h2d	<hexadecimal number>	Convert hexadecimal number into decimal
h2o	<hexadecimal number>	Convert hexadecimal number into octal
h2b	<hexadecimal number>	Convert hexadecimal number into binary
o2h	<octal number>	Convert octal number into hexadecimal
o2d	<octal number>	Convert octal number into decimal
o2b	<octal number>	Convert octal number into binary
b2h	<binary number>	Convert binary number into hexadecimal
b2d	<binary number>	Convert binary number into decimal
b2o	<binary number>	Convert binary number into octal
animate		Animate simulated program
assemble		Run assembler
auto-indent		Automatically indent the edited code
bookmark		Create or delete bookmark on the current line
breakpoint		Create or delete breakpoint on the current line
capitalize		Capitalize selected text
clear		Clear history
comment		Comment selection
copy		Copy selection
custom	<command number>	Run user command
cut		Cut selection
date	<date format>	Insert current time and/or date
exit		Leave command line
exit-program		Exit the IDE
find	<string>	Find a string
goto	<line number>	Go to the specified line
help	<command name>	Display help for the specified command
char	<character code>	Insert a character
indent		Indent selection
kill-line		Delete current line
open	<file name>	Open the specified file
paste		Paste text from clipboard
redo		Take back last undo
reload		Reload current document
replace	<string> <replacement>	Replace a string with another string
run		Run simulator in animation mode
save		Save the current file
set-icon-border		Show/Hide icon border
set-line-numbers		Show/Hide line numbers
sim		Engage/Disengage simulator
step		Step simulated program
tolower		Convert selected text to lowercase
toupper		Convert selected text to uppercase
uncomment		Comment current line
undo		Undo the last text editing operation
unindent		Decrease indentation level of the current line
hibernate	[<target file>]	Hibernate simulated program
resume	[<source file>]	Resume hibernated program
switch-mcu	<MCU name>	Switch current MCU simulation mode to another MCU
set-xcode	<size of XCODE mem.>	Set size external data memory for simulated MCU
set-xdata	<size of XDATA mem.>	Set size external program memory for simulated MCU

Table 3.1: Available commands

3.2 Bottom panel

3.2.1 Main panel of the MCU simulator

This panel is the main part of the simulator user interface. It shows all MCU registers along with content of internal data memory. And contains small toolbar with 6 buttons: “Start”/“Shutdown”, “Reset”, “Step back”, “Step”, “Step over”, “Animate” and “Run”. All visible registers can be modified from here and most SFR registers are represented by enumeration of bits, where each particular bit can be modified separately, green color means logical one and red means zero. Each bit has its own tooltip help with short description of its purpose and status bar with bit address and bit name.

Figure 3.4: Main panel of the simulator

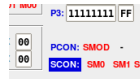
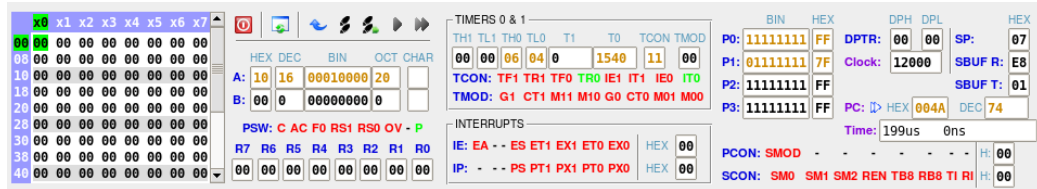


Figure 3.5: Highlighted SFR register

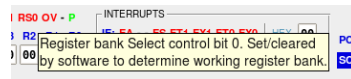


Figure 3.6: Tool tip help for a special function bit



Figure 3.7: Representation of a register value in various numeric bases

3.2.2 C variables

This panel is a part of simulator user interface that maintains a list of global static variables defined in your C language code. Names of variables are displayed along with their current values in simulated MCU. If you program is not written in C language then this tool has no purpose for you at all. Otherwise the purpose of this panel is to make it easier to simulate a program for 8051 written in C language and see what is “really” happening in there. This tool is capable of extracting variable values from multiple registers and the displaying them as one variable, one value. Alteration of variable values is also possible. And search panel in the top right corner of the panel might

help you with finding exact variable which you need to see. But note that functionality of this tool is in fact severely limited, it supports only global static variables, integers and floats, but variable value modification is allowed only for integer variables, no floats.

3.2.3 Graph showing voltage levels

This panel might help you to see what is happening on simulated GPIO² lines. Resolution and grid can be adjusted to better fit your needs. There are three graphs, one for port latches, one for port outputs (without any virtual HW) and one for the most realistic GPIO simulation which this IDE can do.

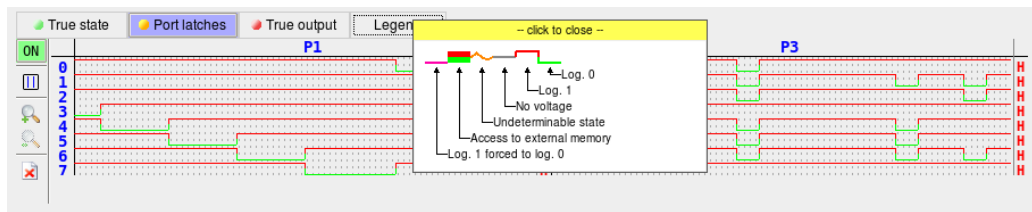


Figure 3.8: GPIO Graph

3.2.4 Messages panel

This panel displays output from the build-in assembler, external assemblers, C compiler and other external tools used in this IDE, which prints something important to standard output. Output from assemblers and SDCC (C compiler) is parsed to highlight warnings and errors and convert them to hyperlinks pointing to source code if possible. The panel also implements a tool for searching strings in the displayed text. User can make this tools visible by pressing Ctrl+F.



Figure 3.9: Messages panel

²General Purpose Input Output

3.2.5 Notes

This is your personal notes for whatever you want. Originally it was intended for writing down a list of things which you need to finish in your work, so some sort of a to do list. But it is just a simple rich text editor with separate file specific notepad. User can use it as he or she consider appropriate.

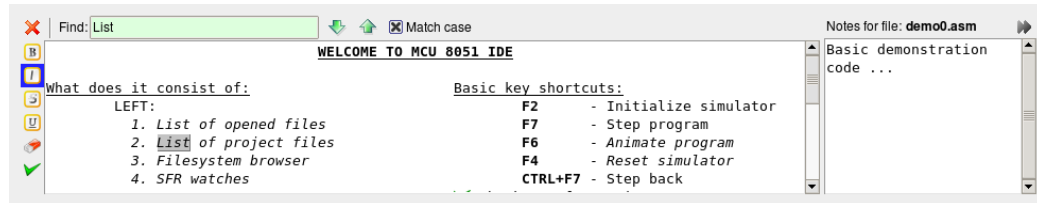


Figure 3.10: Personal notes

3.2.6 Calculator

Calculator is here more or less just for completeness. But you might still find it to a real asset to your efforts. This calculator is capable of performing common arithmetical operations, computing trigonometric functions, logical operations, etc. Supported numeral systems are hexadecimal, decimal, octal and binary in both integer and real numbers. Supported angular measurement units are degrees, radians and gradians. The calculator is also equipped with three memory cells where you can save arbitrary numbers for future computations. On the right side there is a simple calculator dedicated to calculation timers preset values for the specified time, clock frequency, etc.

³

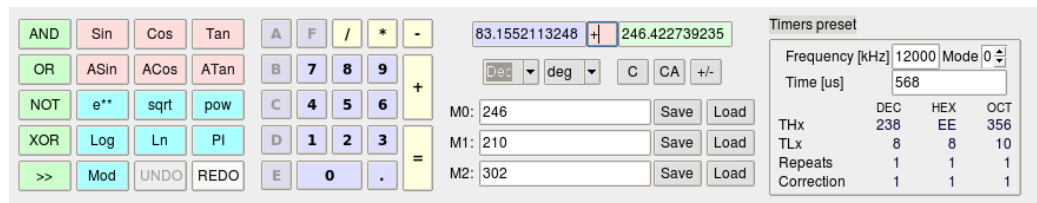


Figure 3.11: Calculator

³Essentially the same but much more advanced function has also the special calculator.

3.2.7 Find in files

With this tool you can search all files in certain directory which names matches specified GLOB⁴ pattern. The search is made for a plain string or regular expression match. This tool might be very useful when you are dealing with many, possibly large, source code files and you suddenly want to find something specific in them. Each line printed in the list of found entries is a hypertext link which opens the file mentioned in it in the source code editor and navigates the editor to line matching the item. In other words it generally the same as well known Unix command “grep”⁵, but with graphical user interface.

3.2.8 Terminal emulator

This is a common color VT102⁶ terminal emulator for the X Window System⁷ as you probably know. More precisely It’s embedded [rxvt-unicode](#) terminal emulator by Marc A. Lehmann and others. Background and foreground colors used in the terminal emulator are configurable in “Terminal configuration Dialog”. Note that this feature is not available on Microsoft® Windows® operating system and probably will never be, because terminal emulator would have only a little use there.

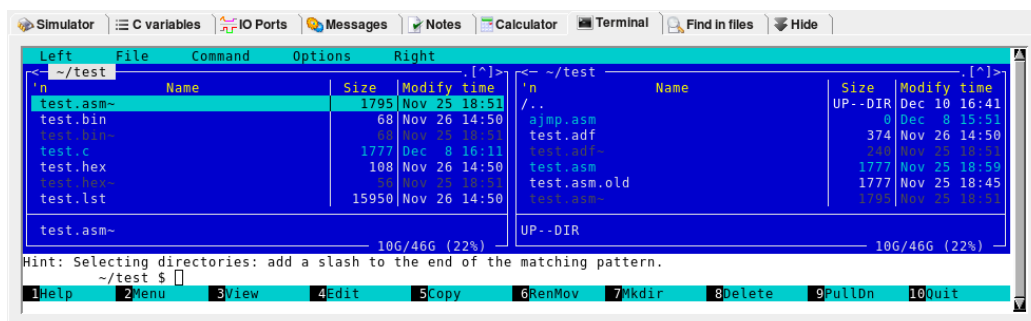


Figure 3.12: Embedded [rxvt-unicode](#) terminal emulator, with the [Midnight Commander](#) running in it

⁴An instance of pattern matching behavior, for example “*.c++” matches all files with “.c++” extension.

⁵A command line text search utility originally written for Unix. The name is taken from the first letters in global/regular expression/print. Its official date of creation is given as March 3, 1973.

⁶A video terminal that was made by Digital Equipment Corporation (DEC). Its detailed attributes became the de facto standard for terminal emulators.

⁷Computer software system and network protocol that provides a basis for graphical user interfaces.

3.3 Left panel

3.3.1 List of opened files

Shows list of all files opened withing the current project. Each entry has its own pop-up menu. Noteworthy features are search bar, sorting by name, size, etc. and open with an external editor. Each file can be added or removed from the list of project files. There is not much to say about it, it's just a simple list with a few nice features but nothing complex.

3.3.2 List of project files

Shows list of all files assigned to the current project. Each entry has its own pop-up menu. Noteworthy features are search bar, sorting by name, size, etc. and open with an external editor. Each file can be excluded from the list of project files, opened or close withing the project.

3.3.3 SFR watches

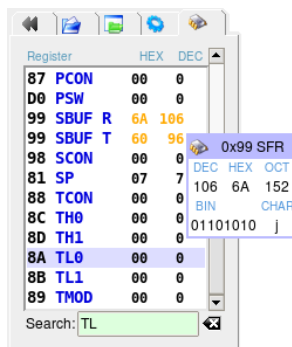


Figure 3.13: SFR watches

From here you can see all special function registers on your chosen MCU in one compact list. Search panel might help you locating particular SFR in this panel and also in the main simulator panel. Each register has two numerical representations of its value in the simulated MCU, decimal and hexadecimal.


3.3.4 File system browser

This panel should help you quickly navigate in your file system in order to open files you want to see as quickly as possible. But many people generally don't like panels like this and will always use only file selection dialog instead.

3.4 Right panel

3.4.1 List of bookmarks


From here you can easily navigate trough all bookmarks made in the current source code file. The panel also highlights item in the list which corresponds

to the current line (line with cursor) in the source code editor. You can also remove all bookmarks at once by pressing the “ Clear all” button.

3.4.2 List of breakpoints

Pretty much the same as list of bookmarks, but this panel shows breakpoints instead of bookmarks, that is the only difference.

3.4.3 Instruction details

When you are writing a code in the assembly language, this panel might be a great help for you. It shows all valid sets of operands for the instruction on your current cursor position in the source code and highlights the set which you are probably using. The same works also for directives. Each line in list has its own help window which appears when user points at it by the pointer. This help window shows additional details regarding the exact instruction. Note also the “ Show legend” button in the upper right corner of the panel.

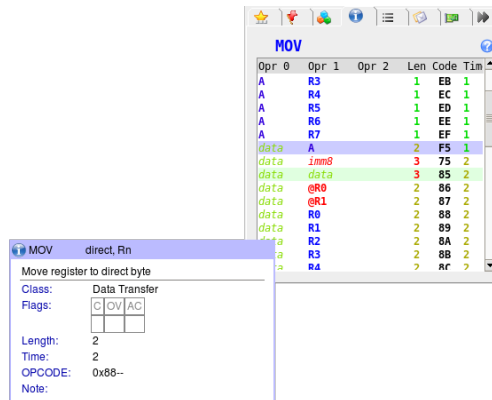


Figure 3.14: Instruction details

3.4.4 Data register watches

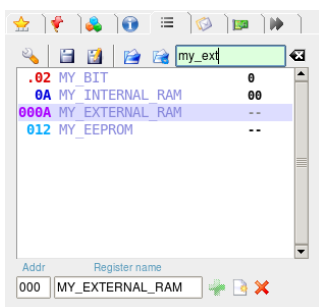


Figure 3.15: Data register watches

This panel might help you to keep track of specific data registers, except for SFR and EEPROM. User can add arbitrary data memory registers which he or she consider to be the most important for his or her current work. You can add a register in the bottom part of the panel. And you can search for specific register, configure the panel and save or load the list of register in the top panel.

This tool is capable of extraction of used sym-

bols from a code listing file⁸ generated by an assembler. This feature can be enabled or disabled in the panel's configuration menu. The current list of watched registers can be saved into a file and loaded from a file⁹.

Memory segments are distinguished by format of the addresses. As you can see in the example, the meaning is this:

Address format	Memory segment
1 or 2 digits	Internal RAM (not SFR)
3 digits	Expanded RAM
4 digits	External RAM
dot and 2 digits	Bit (including SFR area)

Table 3.2: Data register watches: Register address

3.4.5 Subprograms call monitor

From here you can monitor all subprogram and interrupt calls in your program. For each entry there is mentioned the type of call, `acall`, `lcall` or `interrupt`, return address and address from which the call was invoked. And you can force each of them to premature return.

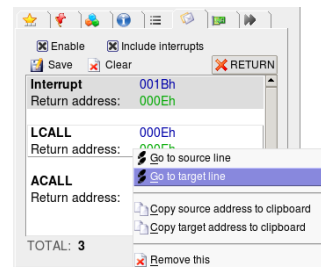


Figure 3.16: Subprograms call monitor

3.4.6 List of symbols

This tool shows a list of symbols defined in source code of your program, works for both assembly language and C language. The list is managed automatically as the user edits the code and is featured with a search panel for easy navigation. Types of symbols can be distinguished by their colors and icons. Colors of particular symbols correspond to the colors used in the source code editor to highlight them.

3.4.7 HW plug-ins manager

This tool does just one thing, allows user to use plug-ins in MCU 8051 IDE. Primary purpose of these plug-ins should be implementation of inter-operation

⁸File with `.lst` file name extension.

⁹These files usually have extension `.wtc`







	Label
	Constant
	Macro
	C variable
	C function
	Other

Table 3.3: Symbol colors and icons in default settings

with certain hardware tools, most probably MCU programmers. if you are interested in writing these plug-ins, please refer to chapter 7.

3.5 Other tools

3.5.1 SFR map

A tabular overview of all available SFRs on your MCU. This tool has similar graphical form as tables of SFR often used in 8051 manuals, but the most important difference is that this one is connected to the simulator and is capable of representing and modifying current values of SFRs in the MCU simulator.

3.5.2 Map of bit addressable area

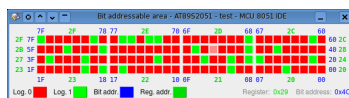


Figure 3.17: Map of the bit addressable area

This tool is a part of the simulator user interface. It shows all bits in the bit addressable area of the simulated MCU. Each square represents one bit, when simulator is on, you can also change value of each one of them by clicking on it. Labels and color used here should be hopefully clear from

the legend at the bottom.

3.5.3 Stack monitor

This tool makes it possible to see entire MCU stack in one view. You can also push any value you want onto the stack or pop a value from it at any time. However this particular tool does not allow for changing the values on the tack in any other way than these.

Each line in the stack monitor represents one octet in the stack, each octet is represented in four numerical

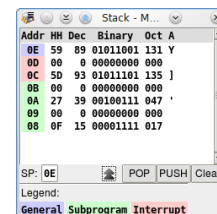


Figure 3.18: Stack monitor

bases, hexadecimal, decimal, binary and octal and also as a character according to ASCII chart. Newly added values are pushed on the top of the list. And their origins are distinguished by background color of the address. These colors are explained in the legend on bottom.

Note that button “Clear” does not clear the stack but instead it clear only the monitor! Buttons “POP” and “PUSH” are intended for manipulation with the stack’s content.

3.5.4 Symbol viewer

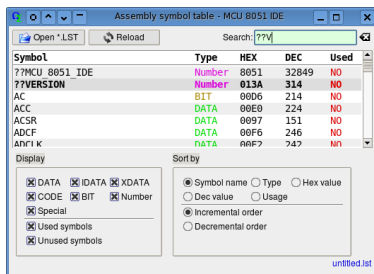
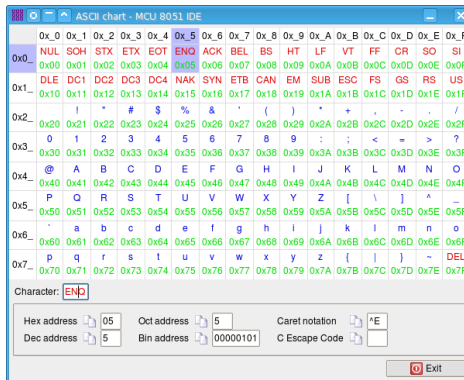


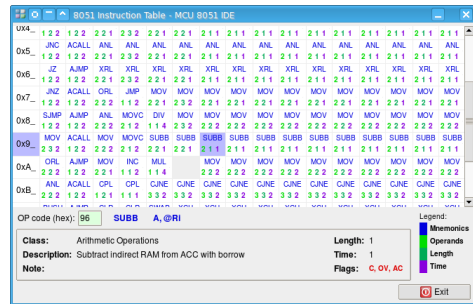
Figure 3.19: Symbol viewer

Figure 3.20: ASCII chart



Symbol viewer shows the table of symbols defined in your program, it works only for assembly language. The table content is taken from code listing generated by assembler. In the top part of the window you can find search bar, and in the bottom part you can specify filter criteria for what you want to see in the table and specify sorting order of the symbols displayed. Symbol in this context are various constants and labels.

Figure 3.21: 8051 Instruction Table



3.5.5 ASCII chart

Colorful interactive ASCII chart, it may prove handy especially when you are dealing with serial communication and this sort of things.

3.5.6 8051 Instruction Table

Colorful interactive 8051 instruction table, very much alike the ASCII chart. But instead of ASCII code you can find there the complete table of 8051 instruction mnemonics, OP codes and related things.

3.5.7 8-segment editor

With this tool you can easily determine what value you have to set on a port to display a digit on a numerical LED display. In the left part of the dialog window, you can find numerical values corresponding to the digit displayed in the middle part. These values are represented for both common cathode and anode and in three numerical bases, hexadecimal, decimal and octal. Buttons on left side from entry boxes copy value from adjacent entry box into clipboard.

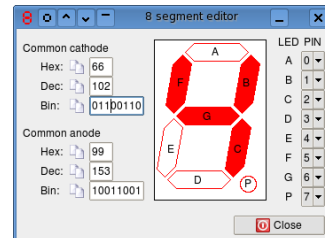


Figure 3.22: 8-segment editor

In the right part of the window you can set what port pin is connected to what LED segment.

3.5.8 Stopwatch

Stopwatch is a tool which can measure certain things in the simulated processor, such as number of instructions processed so far, number of microseconds which would it take for a real processor to execute, number of breakpoints met so far etc. User can also set it to stop the simulation when certain limit in the measurement has been met or exceeded.

3.5.9 Scribble notepad

This is something like a small whiteboard, where you can draw or write your notes. It is a little bit more free than conventional text editor. You can also insert images, supported image formats are PNG and a few others. But don't rely on the scribble notepad too much, this tool has no save or load functions, anything you draw or write there is just temporary and it will not recover upon next start of the IDE.

3.5.10 Base converter

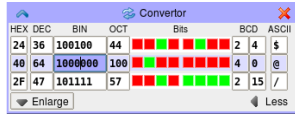


Figure 3.23: Base converter

When you are programming micro-controllers, you might want to convert numbers between various numeric bases. One could say that everyone dealing with such things as micro-controllers would be able to do these conversion without use of any tool. But this doesn't mean that such a tool can never be useful. Values written in the entry boxes of the base converter are saved when user leaved the IDE and are recovered upon next start along with all opened base converter tool windows.

3.5.11 RS-232 debugger

This tool is capable of transmitting and receiving data to/from RS-232 port in your computer, today personal computers usually do not have this type of port, but you can always use something like a USB to RS-232 bridge.

I assume here that the reader is familiar with the RS-232 communication protocol and related terms. This tool acts as a DTE¹⁰.

On the diagram in the upper left corner you can see current logical level on each of RS-232 wires except for RxD and TxD. You can also set value for wires DTR¹¹ and RTS¹² and trigger the break by button BREAK.

Right upper corner contains configuration controls, their functions should be mostly obvious. Check-box "Enable reception" enables or disables writing to hexadecimal editor "Received data". Button "Close" closes the opened physical port. And button "🔄" refreshes the list of available physical ports.

In the bottom part you can see two hexadecimal editors: "Data to send" and "Received data". These are representations of data which we are dealing with. By button "Receive here" you can set address in the hexadecimal editor

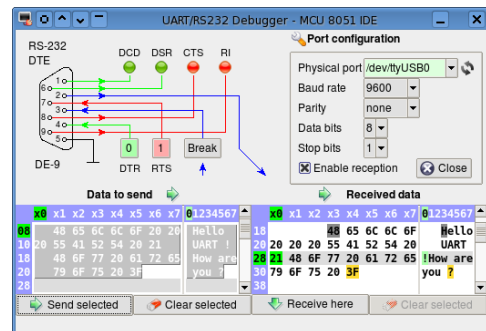


Figure 3.24: UART/RS-232 debugger

¹⁰Data Terminal Equipment, the other side is DCE (Data Circuit-terminating Equipment).

¹¹Data Terminal Ready

¹²Ready To Send

where the received data will be written. And by button “Send selected” you can trigger transmission over the opened physical port, selected chunk of the data will be send then. Button “Clear selected” are intended for removing data from the hexadecimal editors editors.

3.5.12 Hexadecimal editors

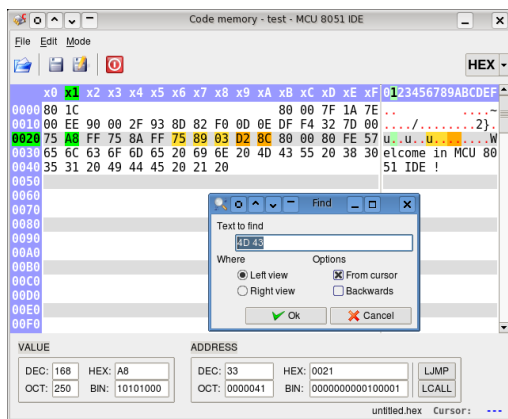


Figure 3.25: MCU code memory editor

In this IDE there are several hexadecimal editors used for various purposes. Each of these editors is equipped with a string search tool and address bars of the left and top side. And in some cases with file saving and loading capability, numerical base switch, ASCII view and a navigation bar at the bottom. Editing is allowed only in overwrite mode, copy and paste works as usual, search dialog can be invoked by pressing Ctrl+F and user can switch between view (left and right) by pressing Tab key. Non printable

characters in ASCII view are displayed in red color.

MCU code memory editor allows user to see and modify contents of the CODE memory of the simulated micro-controller. Special feature of this particular editor is that instruction OP code currently pointed by program counter (PC) is highlighted with dark orange background along with the instruction’s operands. And the same applies also for the previously executed instruction but highlight color is light orange in this case.

MCU data/xdata/eeprom memory editor allows user to see and modify contents of the IDATA/XDATA/EEPROM memory of the simulated micro-controller. Special features of this editors are that recently changed octets are highlighted with light orange foreground color and octets currently being written into the memory are highlighted with gray background color.

MCU eeprom write buffer editor allows to see and modify EEPROM write buffer. Current EEPROM write offset is displayed as well.

Independent hexadecimal editor is universal hexadecimal editor with maximum capacity of 64kB and support for Intel®8 HEX file format. This tool is completely independent from your project in the IDE. This too might be particularly useful when you want to and possibly modify content of a Intel®8 hex file, but do not alter the simulated MCU.

3.5.13 Hibernation of simulated program

The IDE is capable of saving execution state of the simulated program into a file and resuming the program from it anytime later. The file, usually with extension `.m5ihib`, contains values of all data registers including SFR in the simulated MCU along with other values determining MCU state as for example list of active interrupts. The file is in XML format, human readable and usually occupies a few tens of kilobytes. The file does not contain content of the CODE memory, so it has to be available somewhere else in a separate file.

3.5.14 Interrupt monitor

Interrupts monitor is a specialized tool intended for viewing and manipulating with interrupts in simulated MCU. With interrupt monitor you can invoke any interrupt you want at any time, force any interrupt at any time to return, change interrupt priorities or disable or enable particular interrupts. You can also see all interrupts synoptically in one window and alter values of their configuration flags.

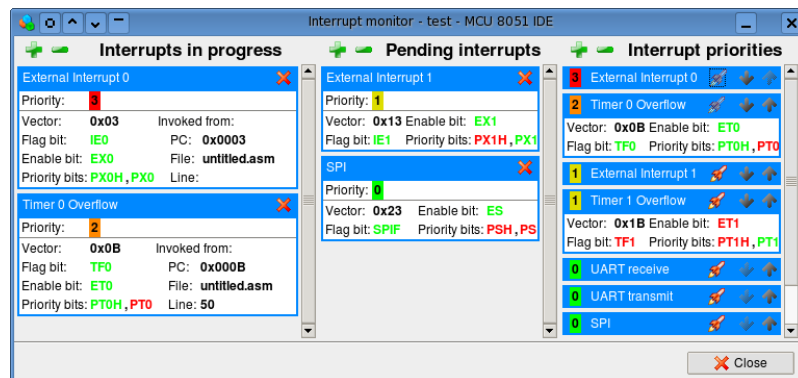







Figure 3.26: Interrupt monitor

3.5.15 Conversions between *.hex, *.bin and *.adf files

Sometimes it might prove helpful to have some tool to convert a binary file to Intel®8 Hex and vice versa. For this purpose MCU 8051 IDE is equipped with a simple tool set for this purpose. In the “Main Menu” → “Utilities” you can find these tools:

-  **HEX → BIN**
Convert Intel®8 Hex file to raw binary file
-  **BIN → HEX**
Convert raw binary file to Intel®8 Hex
-  **SIM → HEX**
Convert simulator assembler debug file (.adf) to Intel®8 Hex file
-  **SIM → BIN**
Convert simulator assembler debug file (.adf) to raw binary file
-  **Normalize Hex**
Read and rewrite the given Intel®8 Hex file, so that all records satisfies specified maximum length (can be set in the assembler configuration dialog), all records are in incremental order and no records overlaps with others.

3.5.16 Normalization of source code indentation

Uniformly indented code is always more aesthetically pleasing and more readable. When you don't have the luxury of having such a code from the first hand, perhaps you will find this feature helpful. This function is available for assembly language and C language if program indent is installed on your system. User can access this function from the “Main Menu” → “Tools” → “Auto indent”.

A small example of the auto indent function in action

Original code:

```

    abc    DATA    7Fh
    ; Start at address 0x00
ORG     0h
label0:inc    R0
        inc    @R0
cjne   R0    , #abc    ,label0
mov  R0,    #0h
        sjmp   label0
; End of assembly
                                END

```

Automatically intended code:

```

abc    DATA    7Fh
; Start at address 0x00
ORG     0h
label0: inc    R0
        inc    @R0
cjne   R0, #abc, label0
mov  R0, #0h
        sjmp   label0
; End of assembly
END

```

3.5.17 Change letter case

This tool can change letter casing to upper or lower case of certain types tokens which your source consists of of. For example you can easily convert all instruction mnemonics in the code to uppercase. It is intended for users who strictly prefers one or another convention of letter casing in assembly language. You can invoke the tool from “Main Menu” → “Tools” → “Change letter case”.

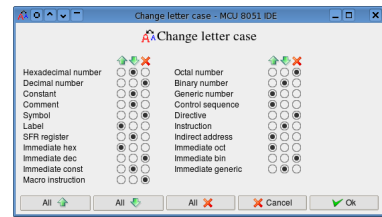





Figure 3.27: Change letter case dialog

-  Convert to uppercase
-  Convert to lowercase
-  Keep current case

3.5.18 User defined commands

Introduction This feature was added in order to enable for use of any auxiliary tools which might useful while working in this IDE. For instance, some hardware tools or some sort of a source code management system like Git or SVN. These custom commands are basically mere Bash scripts with some kind of pseudo-variables available in it. These pseudo-variables are formed as strings beginning with “%”. Before each script execution they are expanded to values corresponding to their meaning. For instance “%filename” expands to the name of the current file. Note that “%%” is expanded as single “%”.

Pseudo-variable	Meaning
%URL	The full URL of the current file
%URLS	List of the URLs of all open documents
%directory	Project directory
%filename	The file name of the current document
%basename	Same as %filename, but without extension
%mainfile	Name of project main file
%line	Number of the current line
%column	Number of the current column
%selection	The selected text in the current file
%text	The full text of the current file

Table 3.4: List of pseudo-variables

Configuration There is specialized configuration dialog for these custom commands.

Execution After the script is executed successfully or not, dialog showing the results will appear upon completion of the script. This dialog contains all textual output from the script caught on standard output and standard error output. If the script outputs anything to the standard error output it is considered unsuccessful.

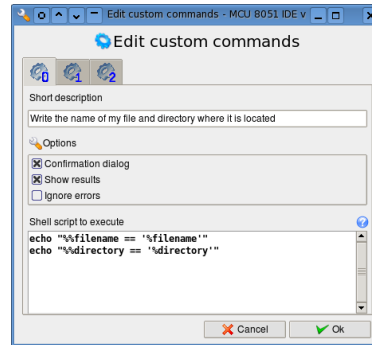


Figure 3.28: Custom commands configuration dialog

3.5.19 Clean-up project folder

This tool can prove useful particularly when your project directory gets “polluted” with lots of unnecessary files, and you want to get rid of them easily and first of all safely. It removes files with certain file name extensions from the project folder. The list of removed files is then written in results dialog. Available from “Main Menu” → “Tools” → “Clean up project folder”.

3.5.20 File statistic

Display certain statistical information about the current source code file. “Main Menu” → “File” → “File statistic”.

3.6 Configuration dialogues

Configuration dialogues are graphical tools for customization of this integrated development environment. And they comprises of these components:

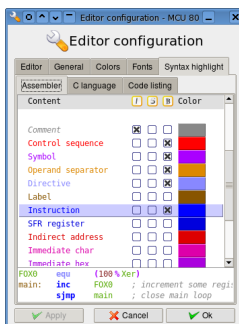


Figure 3.29: Editor configuration dialog

Editor configuration In editor configuration dialog user can change preferred editor from default built-in editor to for example Vim or Emacs and modify configuration the built-in editor. Configurable are colors used for syntax highlight, colors for text area background and so on, font used by editor, indentation mode, auto-save interval and others.

Compiler configuration Compiler configuration dialog allows user to configure behavior of the built-in assembler, chose another assembler instead of this one. Configure the preferred assembler and configure the C compiler (SDCC). Compiler configuration is stored in the project file (the file with `.mcu8051ide` extension).

So these setting are specific to the one specific MCU 8051 IDE project.

Currently supported external assemblers are these:

- ASEM-51 ¹³
- ASL ¹⁴
- AS51 ¹⁵

How to link multiple files when using C language:¹⁶

1. Write makefile,
2. set the IDE to use your makefile instead of calling the C compiler directly (Configuration -> Compiler configuration -> GNU make utility),
3. start compilation as usual.

Simulator configuration Simulator configuration dialog configures these:

1. How to treat indeterminable values in simulator engine
2. How many steps will be remembered during the simulation for later backward steps.
3. What warning conditions will be ignored during the simulation

Right panel configuration Configures colors used in tools “Instruction details” and “Register watches” in the right panel.

Main toolbar configuration Configures contents of main application toolbar.

¹³A really useful assembler written by W.W. Heinz. You can find it at <http://plit.de/asem-51/home.htm>

¹⁴Available at <http://linux.maruhn.com/sec/asl.html>

¹⁵Available at <http://www.pjrc.com/tech/8051>

¹⁶This feature is not yet supported on MS Windows.



Figure 3.30: Main toolbar

Custom commands configuration Configures user defined commands, which are essentially Bash scripts. This feature is currently not available on MS®Windows®OS.

Shortcuts configuration Configures key shortcuts used in the IDE.

Terminal emulator configuration Configures terminal emulator at the bottom panel. This terminal emulator is embedded [rxvt-unicode](#). User can set foreground color and background color of the terminal emulator window and the font. This feature is currently not available on MS®Windows®OS.

Global MCU 8051 IDE configuration Changes settings like GUI language, size of fonts used in the GUI, GUI widget style, whether splash screen should be displayed each time when the IDE is started and so on.

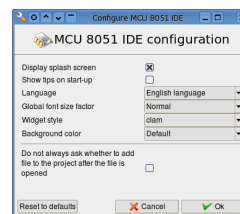


Figure 3.31: Global configuration dialog

Chapter 4

Build-in macro-assembler

In this chapter we will be concerned with MCU 8051 IDE build-in assembler.
¹ With syntax of its statements, directives and 8051 assembler instructions. I assume that the reader is familiar with general concepts of assembly language programming and 8051 architecture. So I will not explain these here.

4.1 Statements

Source code files for this assembler must be text files where lines are formed like these:

```
[ label: ] [ instruction [ operand [ , operand [ , operand ] ] ] [ ;comment ]
[ label: ] directive [ argument ] [ ;comment ]
symbol directive argument [ ;comment ]
```

Everything in square brackets is optional. Compilation does not go beyond line containing “end” directive, so after that directive the code do not have to be syntactically valid. Empty lines are allowed as well as line containing only comment or label. Statements can be separated by spaces, NBSP characters² and tabs. Statements are case insensitive and their length is not limited, overall line length is also not limited.

4.2 Symbols

Symbol names for numbers, macros or addresses defined by user in the code using appropriate directive. Like with “equ” directive you can define a new

¹This assembler manual is inspired by ASEM-51 manual, a great work done by W.W. Heinz

²No Breaking Space (0xC2)

Code 2 An example of well formed assembly language code

```

start:  ; Start timer 0 in mode 2
        mov     R5, #0h
        mov     IE, #0FFh
        mov     TLO, #255d
        mov     TMOD, #03h
        setb    TR0
        sjmp    main

; Main loop
main:    sjmp    $          ; Infinite loop

; Program end
        end

```

symbol and assign a value to it right away. Symbols may consist of upper and lower case letter, digits and underscore character (“_”), their length is not limited, they are case insensitive and they can be the same as language keywords. Be aware of that there cannot coexists two or more symbols in the same memory segment which differs only by letter casing, in other words symbols “abc” and “ABC” are completely the same thing.

4.3 Constants

There are two types of constants numeric constants and character constants. Numeric constants consist of a sequence of digits allowed for the numeric base used and followed by the radix specifier. If the number begins with a letter, there must be the zero digit placed before the number. For example “abh” is not valid numeric constant, but “0abh” is. Character constants consist of sequence of one or more characters enclosed by quote character (’). C escape sequences can be used in character constants. If you want to place quote character (’) into the constant, you can either place two quotes instead of one (“’’’) or escape the quote, that means place backslash “

” before it. There is significant difference between single character constant and multiple character one. Single character constant is regarded by assembler as 8 bin integer number and multiple character constant is a string, a sequence of characters. Since version 1.4.1 it is possible to use prefix “0x” (and “0X”) as radix specifier for hexadecimal numbers, so “0xaf” is the same as “0afh”, etc.

Constant type	Allowed digits	Radix specifier
Binary	0 .. 1	B
Octal	0 .. 7	O or Q
Decimal	0 .. 9	D or none
Hexadecimal	0 .. 9, A .. F	H

Table 4.1: Radix specifiers

Code 3 An example of constants

```

; These are the same number
a set 100111b    ; Binary
a set 47q        ; Octal
a set 39d        ; Decimal
a set 27h        ; Hexadecimal
a set ''''       ; Character

; This is an example of string
db 'string'      ; String

```

4.4 Expressions

Arithmetical expressions are evaluated at compilation time and replaced by assembler with constant corresponding the their resulting value. Expressions comprises of arithmetical operators, constants, symbols and another expressions. An example of such expression might be (X XOR 0FF00H)

Operator	Description	Example
Unary Operators		
NOT	one's complement	NOT 0a55ah
HIGH	high order byte	HIGH 0a55ah
LOW	low order byte	LOW 0a55ah
Binary Operators		
+	unsigned addition	11 + 12
-	unsigned subtraction	13 - 11
*	unsigned multiplication	3 * 5
/	unsigned division	20 / 4
MOD	unsigned remainder	21 MOD 4
SHL	logical shift left	32 SHL 2
SHR	logical shift right	32 SHR 2
AND	logical and	48 AND 16
OR	logical or	370q OR 7
XOR	exclusive or	00fh XOR 005h
.	bit operator	P1.4
EQ, =	equal to	11 EQ 11
NE, <>	not equal to	11 NE 11
LT, <	less than	11 LT 12
LE, <=	less or equal than	11 LT 11
GT, >	greater than	12 GT 11
GE, >=	greater or equal than	12 GT 11

Table 4.2: Expression operators

Code 4 An example of expressions

```
abc    EQU    ( 2000 * 3 / 100 )
xyz    SET    ( LOW abc )
IF ( abc > ( 5 MOD 2 ) )
    MOV    A, # ( ( 15h XOR 12 ) OR xyz )
ELSE
    ADDC   A, # ( HIGH 1234h )
ENDIF
```

4.5 The instruction set processing

This assembler is capable of translating all 8051 instructions with all possible sets of operands. And extends this set with 2 pseudo-instructions: “CALL” and “JMP” which do not stand for any operation code, but are translated according to the used operand. “CALL” can be translated as “ACALL” or “LCALL”, “JMP addr” can be translated as “SJMP”, “AJMP” or “LJMP”.

4.6 Assembler directives

ENDM

ifn IF Not, conditional assembly

Syntax:

```
IFN <expr>
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
ELSE
    MOV A, #40h
ENDIF
```

ifdef IF DEFined

Syntax:

```
IFDEF <symbol>
```

Example:

```
IFDEF CND
    MOV A, #20h
ELSE
    MOV A, #40h
ENDIF
```

ifndef IF Not DEFined

Syntax:

```
IFNDEF <symbol>
```

Example:

```
IFNDEF CND
    MOV A, #20h
ELSE
    MOV A, #40h
ENDIF
```

rept REPeaT Macro

Syntax:

```
REPT <expr>
```

Example:

```
REPT 5
    NOP
ENDM
```

times REPeaT Macro

Syntax:

```
TIMES <expr>
```

Example:

```
TIMES 5
    NOP
```

if Conditional assembly

Syntax:

```
IF <expr>
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
ELSE
    MOV A, #40h
ENDIF
```

else Conditional assembly

Syntax:

```
ELSE
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
ELSE
    MOV A, #40h
ENDIF
```

elseif Conditional assembly

Syntax:

```
ELSEIF <expr>
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
ELSEIF SOMETHING_ELSE
    MOV A, #40h
ENDIF
```

elseifn Conditional assembly

Syntax:

```
ELSEIF <expr>
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
ELSEIF SOMETHING_ELSE
    MOV A, #40h
ENDIF
```

elseifdef Conditional assembly

Syntax:

```
ELSEIF <expr>
```

Example:

```
IF(2 * 4 - CND)
    MOV A, #20h
```

```

ELSEIFDEF SOMETHING_ELSE
  MOV A, #40h
ENDIF

```

elseifndef Conditional assembly
Syntax:

```

ELSEIF <expr>
Example:
IF (2 * 4 - CND)
  MOV A, #20h
ELSEIFNDEF SOMETHING_ELSE
  MOV A, #40h
ENDIF

```

endif Conditional assembly
Syntax:

```

ENDIF
Example:
IF (2 * 4 - CND)
  MOV A, #20h
ELSE
  MOV A, #40h
ENDIF

```

endm END of Macro definition
Syntax:

```

ENDM
Example:
ABC MACRO
  MOV B, #12d
ENDM

```

end END of the program
Syntax:

```

END
Example:
END

```

list enable code LISTing
Syntax:

```

LIST
Example:
NOP
NOLIST
NOP

```

```

NOP
LIST
NOP

```

nolist disabled code listing
Syntax:

```

NOLIST
Example:
NOP
NOLIST
NOP
LIST
NOP

```

dseg switch to DATA segment [at address]
Syntax:

```

DSEG [AT <expr>]
Example:
DSEG at 20d

```

iseg switch to IDATA segment [at address]
Syntax:

```

ISEG [AT <expr>]
Example:
ISEG at 10d

```

bseg switch to BIT segment [at address]
Syntax:

```

BSEG [AT <expr>]
Example:
BSEG at 5d

```

xseg switch to XDATA segment [at address]
Syntax:

```

XSEG [AT <expr>]
Example:
XSEG at 30d

```

cseg switch to CODE segment [at address]
Syntax:

```

CSEG [AT <expr>]

```

	<code><symbol> CODE <expr></code>
Example:	Example:
<code>CSEG at 40d</code>	<code>TBL CODE 600h</code>
flag define a FLAG bit	data define address in the DATA memory
Syntax:	Syntax:
<code><symbol> FLAG <expr></code>	<code><symbol> DATA <expr></code>
Example:	Example:
<code>F4 FLAG 16h</code>	<code>UIV DATA 20h</code>
Note:	idata define address in the Internal DATA memory
<code>Deprecated directive. Consider directive idata instead.</code>	Syntax:
skip SKIP bytes in the code memory	<code><symbol> IDATA <expr></code>
Syntax:	Example:
<code>SKIP <expr></code>	<code>UIV IDATA 20h</code>
Example:	xdata define address in the External DATA memory
<code>SKIP 5</code>	Syntax:
equ EQUivalent	<code><symbol> XDATA <expr></code>
Syntax:	Example:
<code><symbol> EQU <expr></code>	<code>UIV XDATA 400h</code>
Example:	macro MACRO definition
<code>ABC EQU R0</code>	Syntax:
<code>XYZ EQU 4Eh+12</code>	<code><macro> MACRO [<arg0> [,<arg1> ...]</code>
bit define BIT address	Example:
Syntax:	<code>ABC MACRO X</code>
<code><symbol> BIT <expr></code>	<code>MOV X, #12d</code>
Example:	<code>ENDM</code>
<code>ABC BIT P4.5</code>	local define a LOCAL label inside a macro
set SET numeric variable or variable register	Syntax:
Syntax:	<code>LOCAL <label></code>
<code><symbol> SET <expr></code>	Example:
<code><symbol> SET <register></code>	<code>ABC MACRO X</code>
Example:	<code>LOCAL xyz</code>
<code>ALPHA SET R0</code>	<code>xyz: MOV X, #12d</code>
<code>ALPHA SET 42*BETA</code>	<code>ENDM</code>
code define address in the CODE memory	ds Define Space
Syntax:	Syntax:
<code>DS <expr></code>	Example:

```
DS 2+4
```

dw Define Words

Syntax:

```
DW <expr1> [,<expr2> ... ]
```

Example:

```
DW 0,02009H,2009,4171
```

db Define Bytes

Syntax:

```
DB <expr1> [,<expr2> ... ]
```

Example:

```
DB 24, 'August', 09, (2*8+24)/8
```

dbit Define BITS

Syntax:

```
DBIT <expr>
```

Example:

```
DBIT 4+2
```

include INCLUDE an external source code

Syntax:

```
INCLUDE <filename>
```

Example:

```
INCLUDE 'my file.asm'
```

org ORiGin of segment location

Syntax:

```
ORG <expr>
```

Example:

```
ORG 0Bh
```

using USING register banks

Syntax:

```
USING <expr>
```

Example:

```
USING 2
```

byte define BYTE address in the data memory

Syntax:

```
<symbol> BYTE <expr>
```

Example:

```
UIV BYTE 20h
```

Note:

```
Deprecated directive. Consider directive DATA instead.
```

4.7 Assembler Controls

\$date	Inserts date string into page header Syntax: <code>\$DATE(string)</code> Example: <code>\$DATE(1965-12-31)</code>	\$noli Don't list subsequent source lines Syntax: <code>\$NOLI</code> Example: <code>\$NOLI</code>
\$da	Inserts date string into page header Syntax: <code>\$DA(string)</code> Example: <code>\$DA(1965-12-31)</code>	\$nolist Don't list subsequent source lines Syntax: <code>\$NOLIST</code> Example: <code>\$NOLIST</code>
\$eject	Start a new page in list file Syntax: <code>\$EJECT</code> Example: <code>\$EJECT</code>	\$nomod Disable predefined SFR symbols Syntax: <code>\$NOMOD</code> Example: <code>\$NOMOD</code>
\$ej	Start a new page in list file Syntax: <code>\$EJ</code> Example: <code>\$EJ</code>	\$nomo Disable predefined SFR symbols Syntax: <code>\$NOMO</code> Example: <code>\$NOMO</code>
\$include	Include a source file Syntax: <code>\$INCLUDE(string)</code> Example: <code>\$INCLUDE(somefile.asm)</code>	\$nomod51 Disable predefined SFR symbols Syntax: <code>\$NOMOD51</code> Example: <code>\$NOMOD51</code>
\$inc	Include a source file Syntax: <code>\$INC(string)</code> Example: <code>\$INC(somefile.asm)</code>	\$paging Enable listing page formatting Syntax: <code>\$PAGING</code> Example: <code>\$PAGING</code>
\$list	List subsequent source lines Syntax: <code>\$LIST</code> Example: <code>\$LIST</code>	\$pi Enable listing page formatting Syntax: <code>\$PI</code> Example: <code>\$PI</code>
\$li	List subsequent source lines Syntax: <code>\$LI</code> Example: <code>\$LI</code>	\$nopi Disable listing page formatting Syntax: <code>\$NOPI</code> Example: <code>\$NOPI</code>
		\$nopaging Disable listing page formatting Syntax: <code>\$NOPAGING</code>

- Example:
`$NOPAGING`
- \$pagelength** Set lines per page for listing
 Syntax:
`$PAGELENGTH(int)`
 Example:
`$PAGELENGTH(64)`
- \$pl** Set lines per page for listing
 Syntax:
`$PL(int)`
 Example:
`$PL(64)`
- \$pagewidth** Set columns per line for listing
 Syntax:
`$PAGEWIDTH(int)`
 Example:
`$PAGEWIDTH(132)`
- \$pw** Set columns per line for listing
 Syntax:
`$PW(int)`
 Example:
`$PW(132)`
- \$symbols** Create symbol table
 Syntax:
`$SYMBOLS`
 Example:
`$SYMBOLS`
- \$sb** Create symbol table
 Syntax:
`$SB`
 Example:
`$SB`
- \$nosymbols** Don't create symbol table
 Syntax:
`$NOSYMBOLS`
 Example:
`$NOSYMBOLS`
- \$nosb** Don't create symbol table
 Syntax:
`$NOSB`
 Example:
`$NOSB`
- \$title** Inserts title string into page header
 Syntax:
`$TITLE(string)`
 Example:
`$TITLE(My firts code)`
- \$tt** Inserts title string into page header
 Syntax:
`$TT(string)`
 Example:
`$TT(My firts code)`
- \$noobject** Do not create Intel HEX file
 Syntax:
`$NOOBJECT`
 Example:
`$NOOBJECT`
- \$object** Specify file name for Intel HEX
 Syntax:
`$OBJECT(string)`
 Example:
`$OBJECT(my_hex.hex)`
- \$print** Specify file name for list file
 Syntax:
`$PRINT(string)`
 Example:
`$PRINT(my_list.lst)`
- \$noprint** Do not create list file at all
 Syntax:
`$NOPRINT`
 Example:
`$NOPRINT`
- \$nomacrosfirst** Define and expand macro instructions after! conditional assembly and definitions of constants
 Syntax:
`$NOMACROFIRST`
 Example:
`$NOMACROFIRST`

4.8 Predefined Symbols

There are symbols which are defined by default by assembler. The aim is to make it a little easier to write code in assembly language for 8051, because user don not have to define all these symbols in his or her code. This feature can be turned of by “\$NOMOD” control sequence.

Table 4.3: Code addresses

Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value
RESET	000h	EXTI0	003h	TIMER0	00Bh	EXTI1	013h
TIMER1	01Bh	SINT	023h	TIMER2	02Bh	CFINT	033h

Table 4.4: Plain numbers, these symbols are always defined!

Symbol	Value
??MCU_8051_IDE	8051h
??VERSION	0139h ³

Table 4.5: Predefined SFR bit addresses

Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value
IT0	088h	IE0	089h	IT1	08Ah	IE1	08Bh
TR0	08Ch	TF0	08Dh	TR1	08Eh	TF1	08Fh
RI	098h	TI	099h	RB8	09Ah	TB8	09Bh
REN	09Ch	SM2	09Dh	SM1	09Eh	SM0	09Fh
FE	09Fh						
EX0	0A8h	ET0	0A9h	EX1	0AAh	ET1	0ABh
ES	0ACh	ET2	0ADh	EC	0AEh	EA	0AFh
RXD	0B0h	TXD	0B1h	INT0	0B2h	INT1	0B3h
T0	0B4h	T1	0B5h	WR	0B6h	RD	0B7h
PX0	0B8h	PT0	0B9h	PX1	0BAh	PT1	0BBh
PS	0BCh	PT2	0BDh	PC	0BEh		
PPCL	0BEh	PT2L	0BDh	PSL	0BCh		
PT1L	0BBh	PX1L	0BAh	PT0L	0B9h	PX0L	0B8h
TF2	0CFh	EXF2	0CEh	RCLK	0CDh	TCLK	0CCh
EXEN2	0CBh	TR2	0CAh	CT2	0C9h	CPRL2	0C8h
P	0D0h	OV	0D2h	RS0	0D3h		
RS1	0D4h	F0	0D5h	AC	0D6h	CY	0D7h
CR	0DEh	CCF4	0DCh				
CCF3	0DBh	CCF2	0DAh	CCF1	0D9h	CCF0	0D8h

Table 4.6: Predefined SFR addresses

Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value
P0	080h	SP	081h	DPL	082h	DPH	083h
PCON	087h	TCON	088h	TMOD	089h	TL0	08Ah
TL1	08Bh	TH0	08Ch	TH1	08Dh	P1	090h
SCON	098h	SBUF	099h	P2	0A0h	IE	0A8h
P3	0B0h	IP	0B8h	PSW	0D0h	ACC	0E0h
B	0F0h	P4	0C0h	WDTCON	0A7h	EECON	096h
DP0H	083h	DP0L	082h	DP1H	085h	DP1L	084h
T2CON	0C8h	T2MOD	0C9h	RCAP2L	0CAh	RCAP2H	0CBh
TL2	0CCh	TH2	0CDh	AUXR1	0A2h	WDTRST	0A6h
CLKREG	08Fh	ACSR	097h	IPH	0B7h	SADDR	0A9h
SADEN	0B9h	SPCR	0D5h	SPSR	0AAh	SPDR	086h
AUXR	08Eh	CKCON	08Fh	WDTPRG	0A7h		
CH	0F9h	CCAP0H	0FAh	CCAP1H	0FBh	CCAP2H	0FCh
CCAP3H	0FDh	CCAP4H	0FEh	CCAPL2H	0FCh	CCAPL3H	0FDh
CCAPL4H	0FEh	ADCLK	0F2h	ADCON	0F3h	ADDL	0F4h
ADDH	0F5h	ADCF	0F6h	P5	0E8h	CL	0E9h
CCAP0L	0EAh	CCAP1L	0EBh	CCAPL2L	0ECh	CCAPL3L	0EDh
CCAPL4L	0EEh	CCON	0D8h	CMOD	0D9h	CCAPM0	0DAh
CCAPM1	0DBh	CCAPM2	0DCh	CCAPM3	0DDh	CCAPM4	0DEh
P1M2	0E2h	P3M2	0E3h	P4M2	0E4h	P1M1	0D4h
P3M1	0D5h	P4M1	0D6h	SPCON	0C3h	SPSTA	0C4h
SPDAT	0C5h	IPL0	0B8h	IPL1	0B2h	IPH1	0B3h
IPH0	0B7h	BRL	09Ah	BDRCON	09Bh	BDRCON_1	09Ch
KBLS	09Ch	KBE	09Dh	KBF	09Eh	SADEN_0	0B9h
SADEN_1	0BAh	SADDR_0	0A9h	SADDR_1	0AAh	CKSEL	085h
OSCCON	086h	CKRL	097h	CKCON0	08Fh		

4.9 Segment type

Segment type specifies the address space to which a symbol is assigned. For example if you define symbol ABC using “XDATA” directive, then ABC is assigned to XDATA segment. Purpose of this is to semantically distinguish between different types of symbols. For example if we use a symbol as address to program memory it has different meaning that if we used it as address to bit addressable area.

DATA	Internal data memory and SFR
IDATA	Internal data memory only
XDATA	External data memory only
BIT	Bit addressable area only
CODE	Program memory only
NUMBER	Arbitrary value

Table 4.7: Segment types

Symbols might be assigned to these segment types by these directives:

- DATA (segment DATA)
- IDATA (segment IDATA)
- XDATA (segment XDATA)
- BIT (segment BIT)
- CODE (segment CODE)
- EQU, SET (segment NUMBER)

Code 5 Example of symbol definitions

```

MY_A DATA '\n' ; DATA segment (internal data memory and SFR)
MY_B IDATA 0AAH ; IDATA segment (internal data memory only)
MY_C XDATA 14Q ; XDATA segment (external data memory only)
MY_D BIT P1.2 ; BIT segment (bit addressable area only)
MY_E CODE 62348D ; CODE segment (program memory only)
MY_F EQU 242Q ; Segment NUMBER (arbitrary value)

; Segment NUMBER (arbitrary value)
MY_G SET MY_A + MY_B + MY_C + MY_D + MY_E + MY_F

```

Code 6 Example of address space reservation

```

; CODE segment
    cseg at 40h      ; Start this segment at address 40 hexadecimal (64d)
my_c  CODE  00abch  ; Define an address in code memory
word:  DW   01234h  ; Define a word in code memory, will be written to code memory
my_cs:  DB   'abcdef'; Define a string in code memory, will be written to code memory

; DATA segment
    dseg at 10q     ; Start this segment at address 10 octal (8d)
my_d   DATA  'd'   ; Define address in internal data memory or SFR area
my_ds:  DS   4      ; Reserve 4 bytes here and set 'my_ds' to point there

; IDATA segment
    iseg at 10d     ; Start this segment at address 10 decimal
my_i   IDATA  'i'   ; Define address in internal data memory
my_is:  DS   4      ; Reserve 4 bytes here and set 'my_is' to point there

; BIT segment
    bseg at 10b     ; Start this segment at address 10 binary (2d)
my_bit BIT   'b'   ; Define address in bit addressable area
my_bs:  dbit  4     ; Reserve 4 bits here and set 'my_bs' to point there

; XDATA segment
    xseg at 10      ; Start this segment at address 10 decimal
my_x   XDATA  'x'   ; Define address in external data memory
my_xs:  DS   4     ; Reserve 4 bytes here and set 'my_xs' to point there

address equ 0h      ; Define symbol 'address' in the NUMBER segment

org     address    ; Start writing program code at address defined by symbol 'address'

; Clear 1st bit in BIT array 'my_bs'
clr     my_bs+1

; Move 10d to 2nd byte in DATA array 'my_ds'
mov     my_ds+2, #10d

; Move 88d to 3rd byte in IDATA array 'my_is'
mov     my_is+3, #88d

; Move 55h to 0th byte in XDATA array 'my_xs'
mov     A, #55h
mov     DPTR, #( my_xs + 0 )
movx    @DPTR, A

; Read 1st byte from CODE array 'my_cs'
mov     DPTR, #my_cs
mov     A, #1
movc    A, @A+DPTR

sjmp    $          ; Infinite loop ('$' stands for address of current instruction)

end     ; End of assembly, everything after this directive is ignored

```

4.11 Macro Processing

Macro is a sequence of instructions which can be expanded anywhere in the code and for any number of times. That may reduce necessity of repeating code fragments as well as source code size and make the solved task easier to comprehend and solve. Unlike subprograms macros do not add extra run-time overhead and repeating usage of macros may significantly increase size of the resulting machine code. Macros supported by this assembler are divided to named and unnamed ones.

MACRO	Define a new named macro
REPT	Define a new unnamed macro and expand it right away for the specified number of times
TIMES	Exactly the same as “REPT”
ENDM	End of macro definition

Table 4.8: Directives directly related to macros

This can be well demonstrated on examples:

Code 8 An exaple of REPT directive

```

rept 3 ; Repeat the code 3 times
    mov a, p0
    cpl a
    mov p1, a
endm

; This is the same as if you wrote this:
    mov a, p0
    cpl a
    mov p1, a
    mov a, p0
    cpl a
    mov p1, a
    mov a, p0
    cpl a
    mov p1, a

```

Code 9 An exaple of simple named macro

```
abc    macro    ; Define named macro 'abc'
        mov     a, p0
        cpl    a
        mov     p1, a
endm

abc    ; Expand macro 'abc' here
abc    ; Expand macro 'abc' here

; This is the same as if you wrote this:
        mov     a, p0
        cpl    a
        mov     p1, a
        mov     a, p0
        cpl    a
        mov     p1, a
```

Code 10 An exaple of named macro with two parameters

```
; Define macro named as 'xyz' with 2 mandatory parameters
xyz    macro    foo, bar
        mov     foo, #10h
        cpl    bar
endm

xyz    a, c      ; Expand macro 'xyz' here
xyz    p0, p1.0  ; Expand macro 'xyz' here

; This is the same as if you wrote this:
; xyz a, c
        mov     a, #10h
        cpl    c
; xyz p0, p1.0
        mov     p0, #10h
        cpl    p1.0
```

Code 11 An exaple of named macro used with if statement

```
ijk    macro    foo
      add     A, @R0

      if foo = 4d
          nop
      endif

      subb   A, #foo
endm

ijk    5
ijk    4
```

; This is the same as if you wrote this:

```
; ijk 5
add   A, @R0
if 5 = 4d
nop
endif
subb  A, #5
; ijk 4
add   A, @R0
if 4 = 4d
nop
endif
subb  A, #4
```

Code 12 An exaple of nested macros

```
; Suppose we have these macros:
```

```

abc    macro
      mov    a, p0
      cpl   a
      mov    p1, a
endm

ijk    macro    foo
      add    A, @R0

      if foo = 4d
          nop
      endif

      subb   A, #foo
endm

xyz    macro    foo, bar
      ijk    foo
      ijk    bar

      abc
endm

```

```
; And we expand 'xyz' like this:
```

```
xyz    4, 5
```

```
; Then we get this result:
```

```

; ijk    4
add    A, @R0
nop    ; <-- Note this
subb   A, #4
; ijk    5
add    A, @R0
subb   A, #5
; abc
mov    a, p0
cpl   a
mov    p1, a

```

Code 13 An exaple of nested macros, which will not work

```

abc    macro
; Unnamed macro cannot be contained inside a named one
times 2
      mov    a, p0
      cpl   a
      mov    p1, a

      endm
endm

```

4.12 Reserved keywords

Table 4.9: Special instruction operands

\$	Location counter
A	Accumulator
AB	A/B register pair
C	Carry flag (in PSW register)
DPTR	Data pointer
PC	Program counter
R0..R7	Registers

Table 4.10: Instruction mnemonics

ACALL	ADD	ADDC	AJMP	ANL	CJNE	CLR	CPL	DA
DEC	DIV	DJNZ	INC	JB	JBC	JC	JMP	JNB
JNZ	SJMP	JNC	CALL	JZ	LCALL	LJMP	MOV	MOVC
MOVX	MUL	NOP	ORL	POP	PUSH	RET	RETI	RL
RLC	RR	RRC	SETB	SUBB	SWAP	XCH	XCHD	XRL

Table 4.11: Directives

BIT	BSEG	BYTE	CODE	CSEG
DATA	DB	DBIT	DS	DSEG
DW	ELSE	ELSEIF	ELSEIFDEF	ELSEIFN
ELSEIFNDEF	END	ENDIF	ENDM	EQU
FLAG	IDATA	IF	IFDEF	IFN
IFNDEF	INCLUDE	ISEG	LIST	MACRO
NAME	NOLIST	ORG	REPT	SET
SKIP	TIMES	USING	XDATA	XSEG

Table 4.12: Expression operators

AND	EQ	GE	GT	HIGH
LE	LOW	LT	MOD	NE
NOT	OR	SHL	SHR	XOR

Table 4.13: Assembler controls

DA	DATE	EJ	EJECT
LI	LIST	NOLI	NOLIST
NOMACROFIRST	NOMO	NOMOD	NOMOD51
NOOBJECT	NOPAGING	NOPI	NOPRINT
NOSB	NOSYMBOLS	OBJECT	PAGELength
PAGewidth	PAGING	PI	PL
PRINT	PW	SB	SYMBOLS
TITLE	TT		

4.13 Compatibility with ASEM-51

Not yet specified.

4.14 List File Format

Code listing serves as an additional information about the assembled code and the progress of the assembly process. It contains information about all symbols defined in the code. Where and how were they were defined, what are their values and whether they were used in the code. Also detailed information about all macros defined in the code and/or expanded in the code. Conditional compilation configuration, instruction OP codes, address space reservations, inclusion of code from another files. And all warnings, errors and notes generated during the assembly by the assembler. There are assembler control sequences which alters formatting of the code listing file. These control sequences will be discussed here. Format of code listing generated by this assembler is very similar to the one generated Metalink®ASM51. Code listing contains entire source code which was assembled but with each

Code 14 A simple code listing

```
demo0
1      ; MCU 8051 IDE - Demonstration code
2      ; Very simple code
3
4      ; Press F2 and F6 to run the program (start simulator and animate)
5
6      org      0h
7
0000 08      8      main:  inc      R0
0001 06      9      inc      @R0
0002 B87FFB 10     cjne     R0, #07Fh, main
0005 7800    11     mov     R0, #0d
0007 80F7    12     sjmp    main
13
14     end
ASSEMBLY COMPLETE, NO ERRORS FOUND, NO WARNINGS
```

line prefixed with line number and some additional information which will be explained later. Besides the original code there is also table of symbols defined during the assembly unless it was turned off. Code listing is divided into pages separated by form feed character, this behavior may be altered by certain assembler control sequences as well as page height and width.

Each line of code listing which contains original source code line may contain beside line number also some additional information regarding the compilation of the given line of code. Such a additional information might look like this and is composed of these parts:

Control sequences affecting format of the generated code listing.

Code 15 Explanation code listing format

0055			18	X data 55h
0014	1122	=1	33	l: inc A
			35	+1 abc ; Expand macro "abc" here
001E	E580		36	+1 mov a, p0
0020	F4		37	+1 cpl a
0021	F590		38	+1 mov p1, a

Line number
Level of file inclusion
Level of macro expansion
Address in code memory
Machine code or another value to be stored in the code memory
Value of a symbol
Original line

Table 4.14: Control sequences affecting code listing

\$ject	\$ej	\$nolist	\$noli
\$list	\$li	\$paging	\$pi
\$nopaging	\$nopi	\$pagewidth	\$pw
\$pagelength	\$pl	\$title	\$tt
\$date	\$da	\$nosb	\$nosymbols
\$noprnt	\$symbols	\$sb	\$print

Code 16 A more complicated example of code listing

```

complicated_lst
001C          1  abc    equ    ( 14 * 2 )    ; Define symbol abc
              2          org    0              ; Start writing code at address 0
              3
              =1 4          include 'my_macros.asm' ; Include file my_macros.asm
              =1 5          ; This is the beginning of file my_macros.asm
              =1 6  my_cpl macro  foo
              =1 7          mov    A, foo
              =1 8          cpl    A
              =1 9          mov    foo, A
              =1 10         endm
              =1 11         ; This is the end of file my_macros.asm
              12
              13 +1 main:  my_cpl P0          ; Expand macro my_cpl here
0000 E580    14 +1          mov    a, p0
0002 F4      15 +1          cpl    a
0003 F580    16 +1          mov    p0, a
0005 80F9    17          sjmp   main          ; Jump back to label main
              18          end          ; End of assembly

ASSEMBLY COMPLETE, NO ERRORS FOUND, NO WARNINGS

```

4.15 Specification of Intel®8 HEX Format

Intel®8 HEX is a popular object file format capable of containing up to 64kB of data. Hex files have usually extension `.hex` or `.ihx`. These files are text files consisting of a sequence of records, each line can contain at most one record. Records starts with “:” (colon) character at the beginning of the line and ends by end of the line. Everything else besides records should be ignored. Records consist of a sequence of 8-bit hexadecimal numbers (e.g. “a2” or “8c”). These numbers are divided into “fields” with different meaning, see the example below.

Code 17 An example of an Intel®8 hex code

```

: 0F 0000 00 E580F4F590E580F4F590E580F4F590 57
: 0F 000F 00 E580F4F590E580F4F590E580F4F590 48
: 0F 001E 00 E580F4F590E580F4F590E580F4F590 39
: 10 002D 00 E580F4F5907410B3758010B2907410B3 30
: 10 003D 00 758010B2902694052600940426940526 0A
: 10 004D 00 00940426009404269405E580F4F59026 8A
: 0B 005D 00 009404269405E580F4F590 63
: 00 0000 01 FF

```

	Start code
	Byte count
	Address
	Record type
	Data
	Checksum (two's complement of 8-bit sum of entire record, except for the start code and the checksum itself)

Record types available in Intel®8 HEX

00 Data record

01 End of file record

Chapter 5

Disassembler

Disassembler is a tool intended to generate assembly language code from an object file. In other words it has certain level of capability of reversing the assembly process and regaining the original source code from any object code. But there are some restriction to that capability and the whole thing is not so simple after all. So let's discuss disassembly process deeper. In MCU 8051 IDE you can invoke disassembler from the main menu "Main Menu" → "Tools" → "Disassemble".

A simple example of a code generated by disassembler

Original code:	Code generated by disassembler
<code>org 0h ; Start at address 0x00</code>	<code>ORG 0h</code>
<code>main: inc R0 ; Increment R0</code>	<code>label0: inc R0</code>
<code>inc @R0</code>	<code>inc @R0</code>
<code>cjne R0, #07Fh, main</code>	<code>cjne R0, #7Fh, label0</code>
<code>mov R0, #0d ;</code>	<code>mov R0, #0h</code>
<code>sjmp main ; Jump back to label main</code>	<code>sjmp label0</code>
<code>end ; End of assembly</code>	<code>END</code>

As you can see from the example above, the code generated by disassembler is the same as the original code. But of course original symbol names have vanished as well as comments, indentation and other tiny details which cannot be determined from the object code. This is caused by the simple fact that the object code contains only the machine code. It contains no information regarding how exactly the original code looked like. Just instructions with their operands and data directly written to the code memory by "DB" and "DW" directives. And here we are getting to the real problem which emerges every time when you try to disassemble "not exactly a simple" code.

8051 instructions comprises of 1, 2 or 3 bytes, the first byte determinates what instruction are we dealing with and so what is its length in bytes. But

if the original code contained directives “DB” or “DW” then the disassembler “thinks” that these values are instructions too. If the disassembler consider a arbitrary value given by “DB” or “DW” instruction to be an instruction, it determinates its length according to its OP code (the 1st argument to the directive). And so it takes 0, 1 or 2 bytes next and interprets them as operands to that instruction. Then when it encounters a real instruction OP code it might think of it as another operand to something and so misinterpret it. Then you might end up with a code that is completely different from the original code and makes no sense at all to human. But if you reassemble such a “non sense” code with disabled peep hole optimization you must get the original object code back, and its functionality must not be changed. Even if the code seems to be absolutely non sense. In that case I strongly recommend to use another disassembler than is the built-in one. Consider for example D52 <http://www.8052.com/users/disasm/>. The built-in diassembler is provided just for “completeness”, but its suitability for a real reverse engineering is highly questionable.

A simple example of a BADLY generated code by disassembler

Original code:	Code generated by disassembler
<code>org 0h ; Start at address 0x00</code>	<code>label0 CODE 11h</code>
<code>main: inc R0 ; Increment R0</code>	<code>ORG 0h</code>
<code>inc @R0</code>	<code>label1: inc R0</code>
<code>jmp cont</code>	<code>inc @R0</code>
<code>db 'some stringx'</code>	<code>ljmp label0</code>
<code>cont: cjne R0, #07Fh, main</code>	<code>jmp @A+DPTR</code>
<code>mov R0, #0d ;</code>	<code>xrl A, R7</code>
<code>sjmp main ; Jump back to label main</code>	<code>xrl A, R5</code>
<code>end ; End of assembly</code>	<code>xrl A, 20h</code>
	<code>jmp @A+DPTR</code>
	<code>mov A, #72h</code>
	<code>xrl A, R1</code>
	<code>xrl A, R6</code>
	<code>xrl A, @R1</code>
	<code>mov R0, #0B8h</code>
	<code>mov R7, #0ECh</code>
	<code>mov R0, #0h</code>
	<code>sjmp label1</code>
	<code>END</code>

Chapter 6

MCU simulator

6.1 Short introduction

The MCU simulator is a tool designed to mimics behavior of a real MCU as much as possible. But it can have certain limitations, the most expectable limitation is of course the speed of simulation. This simulator is very slow, but offers some extra features.

6.2 Modes of simulation

There are 4 modes of simulation:

Step Execute exactly one intruction, no matter how many machine cycles it will take. This does not apply for macro-instruction, in that case each instruction of the macro is executed separately.

Step over Execute as many instructions as possible until simulator cursor changes its location from one line of source code to another.

Animate Do the same as “step” but in a loop, one after another until stopped by a waring condition or user request.

Run This is generally the same as “animate”, but much faster, because GUI is not updated so offten as in the “animate” mode.

(Step Back) Take back the last performed step. There is limited number of step which can be taken back.

Virual HW can be enabled or disabled, it significantly affects speed of the simulation. Of course simulation is slower when virtual HW is on.

6.3 Warning conditions

- Stack overflow
- Stack underflow
- Invalid instructions
- Watchdog overflow
- Invalid return from interrupt
- Reading from write only register
- Invalid access to IDATA/SFR
- Invalid access to EDATA
- Invalid access to XDATA
- Invalid access to bit
- Invalid access to CODE
- EEPROM write failure
- EEPROM write abort
- UART frame discard
- Illegal UART mode change
- Illegal Timer/Counter mode change

6.4 Limitations

1. UART simulation is limited in current version
2. SPI simulation is not implemented
3. Simulation of reduced power consumption modes is not supported
4. Simulated MCU is many times slower the real one would be on “normal” conditions

6.5 Virtual hardware

MCU 8051 IDE simulator is also equipped with a few simulated simple hardware devices, which can be connected to the simulated MCU. These virtual hardware components are intended primarily to offer a better insight into programs interacting with things like LEDs or keys.

All wires connected to specific GPIO lines are colored according to the voltage level present on them, colors are the same as for graph in the bottom panel (GREEN == log. 0; RED == log. 1; GRAY == not connected, etc.) Each of these virtual HW components has its own configuration menu and help text available through that menu. Configuration can be saved to a file with extension `.vhc`, and can be loaded from that file later. The configuration menu is accessible through the button with this icon: “🔧”. The “ON”/“OFF” button enables or disables entire subsystem of virtual hardware simulation including the graph of GPIO wires on the bottom panel.

6.5.1 DS1620 temperature sensor

Simulates DALLAS®DS1620 thermometer along with its 3-wire communication interface. Temperature which this simulated device should measure can be set by used on the scale in DS1620 simulator window. All internal registers of the device are displayed to the user and are modifiable by the user, current configuration of the device simulator including DS1620 non-volatile registers can be save into a file for further use. All communications with the simulated MCU and internal operation of the simulated thermometer are displayed in simulator log, log can be accessed via the DS1620 simulator configuration menu (“🔧”).

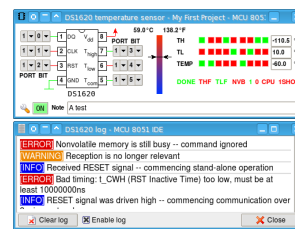


Figure 6.1: DS1620 simulator and its log window

6.5.2 File interface

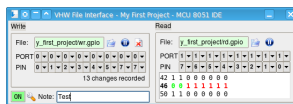


Figure 6.2: PALE I/O interface

This tool can automatically switch states of GPIO lines of the simulated according to certain definition file and it can also record all changes occurring on arbitrary GPIO line to a specified output file. This function can be particularly useful when you are dealing with a 8051 program which extensively works with I/O ports.

This tool can automatically switch states of GPIO lines of the simulated according to certain definition file and it can also record all changes occurring on arbitrary GPIO line to a specified output file. This function can be particularly useful when you are dealing with a 8051 program which extensively works with I/O ports.

6.5.3 LED Panel

This is the simplest example of such a virtual hardware component. A simple panel consisting of 8 independent LEDs with common anode. Each LED can be connected to separate port and pin and react immediately to any change in voltage level on that line. Connections with the μC are made with combo boxes on the bottom side of the panel.

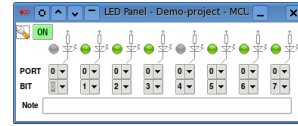


Figure 6.3: LED Panel

6.5.4 Single LED Display

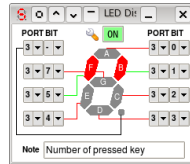


Figure 6.4: LED Display

Single 8 segment LED display with one decimal point. Common electrode for the LEDs can be configured as well as the LED color. Each LED can be independently connected to any port and pin and reacts immediately to any change in voltage level on that pin. Common electrode is statically connected to either common ground or Vcc.

6.5.5 Multiplexed LED Display

4 digits LED displays intended for run in multiplexed mode, LEDs are fading with configurable delay. Each digit has its own common electrode which polarity is configurable, this common electrode is connected to output from an inverter or transistor. There are four color shades for each LED segment, one for inactive, one for active, one for fast blinking and one for segment which was recently dimmed, that makes it possible to see an image which would probably appear on a real display when viewed just by the eye. Unit for the fading interval is one instruction cycle.

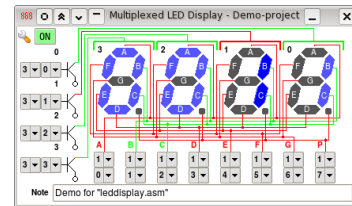


Figure 6.5: M LED Display

6.5.6 LED Matrix

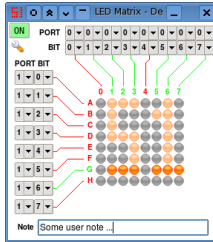


Figure 6.6: LED Matrix

Simple one color 8 x 8 LED matrix indented for run in multiplexed mode, LEDs are fading with configurable delay. Everything here is the same as for the Multiplexed LED Display, except for one thing, mapping. Mapping can be set to row, column or random, which is default. Row mapping means that row which has been activated right now immediately forgets which LEDs were shining last time and which were not. Column mapping is the same but for columns and random mapping means that each LED will dim after specified interval and not when its row or column is activated. So in random mapping you have to wait until all LEDs are gray before you can draw a new image without being affected by the last one.

6.5.7 Matrix Keypad

4 x 4 Matrix keypad, each row and column can be connected to any GPIO line. Connections with the μC are made with the combo boxes. Keys can be configured to behave as radio button¹. Note that this tool can be also used to interconnect some port pins together statically, like wires in a bread board. Any key press takes effect immediately in all other virtual hardware components connected to the same line.

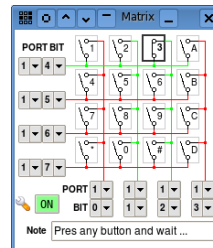


Figure 6.7: Matrix Keypad

¹Radio buttons that means that one one key can be pressed at the time and when you press another key, the originally pressed key will return back to non pressed state

6.5.8 Simple Keypad

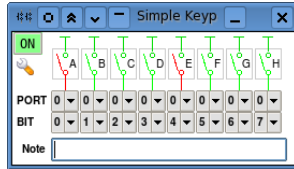


Figure 6.8: Simple Keypad

Array of 8 independent keys, each one of them can connect any GPIO line to the ground. Any key press takes effect immediately in all other virtual hardware components connected to the same line. Keys can also be configured to behave as radio buttons.

6.5.9 LCD display controlled by HD44780

This tool simulates a HD44780 character LCD of any size up to 2 rows and 40 columns. There are 11 I/O lines serving as interface for the MCU, “E”, “RS”, “R/W” and “D0”..“D7”. User can view end modify content of the display data RAM (DDRAM), the character generator RAM (CGRAM) and certain HD44780 registers: instruction register (IR), data register (DR), address counter (AC) and display shift, these registers are shown in hexadecimal representation. User can also view content of character generator ROM (CGROM) and set font to use. All of the driver commands are fully supported and all important events occurring in the simulated driver (HD44780) are recorded in the simulator log. User can also see and modify certain HD44780 configuration flags like “B”, “S”, “D” and so on. And the window is collapsible.

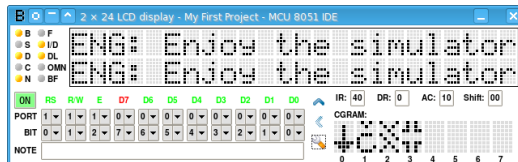


Figure 6.9: Simulated LCD display

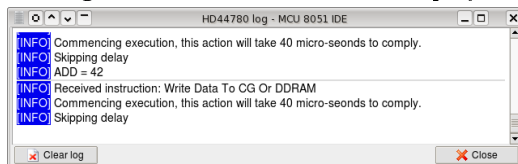


Figure 6.10: HD44780 Log

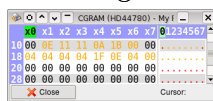


Figure 6.11: CGRAM

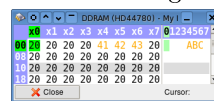


Figure 6.12: DDRAM

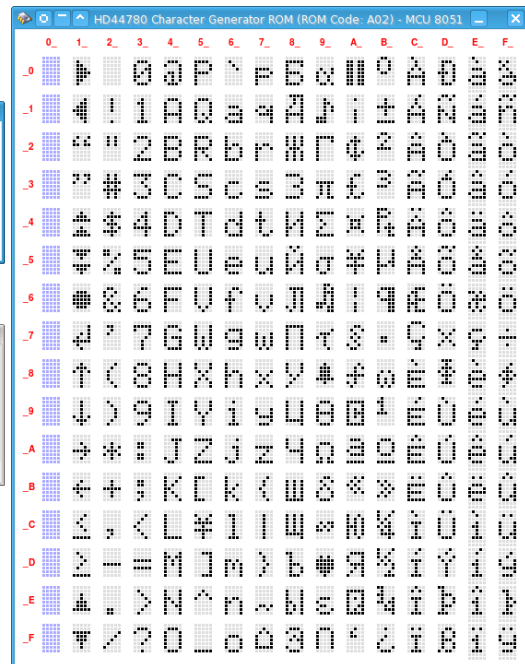


Figure 6.13: View on CGROM

Chapter 7

Writing hardware tool control plug-ins

7.1 Foreword

It is not surprising that IDE for micro-controllers should be capable of inter-operation with certain hardware tools. MCU 8051 IDE has tool named HW plug-ins manager which is responsible for loading and managing plug-ins written in order to to “integrate” exiting hardware tools into this IDE. With this feature every author of a 8051 programmer, ICD, ICE etc. who knows Tcl/Tk language has the opportunity to make his or her own tool working in direct cooperation with the IDE. These plug-ins have to be written at least partially in the TCL language and use the Tk library along with API of MCU 8051 IDE. But that doesn't mean that they should be written entirely in Tcl/Tk. On the contrary I would encourage usage of another languages for example SSP89S, also a part of MCU 8051 IDE project, is written almost completely in C++/Qt4, but only a “small” piece of the software is written in Tcl/Tk. Tcl/Tk can easily inter-operate with C and C++, also it is possible to run arbitrary separate process from inside of Tcl/Tk program and control it vie for example TCP sockets or its stdin/stdout or something else.

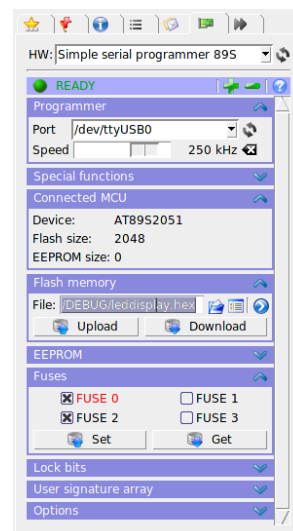


Figure 7.1: An example of HW control plug-in

7.2 How to write your own plug-in

At first take these steps:

1. Create the plug-in directory

On POSIX system the plug-in directory have to be placed in “/usr/share/mcu8051ide/hwplugins”, on Microsoft®Windows®the directory is “<YourInstallationDirectory>\hwplugins”. The plug-in directory must carry the name of your plug-in, where spaces are replaced with “_” (underscore) characters. For example suppose you want to create a plug-in named as “My First Plug-in v1.0”, then your plug-in directory directory is “My_First_Plug-in_v1.0”.

2. Create the plug-in initialization file

Plug-in initialization file tells the IDE that there is some plug-in at all. The file contains basic initialization of the plug-in environment and must follow certain rules. The file name must also follow name of the plug-in in the same way as the plug-in directory. But the initialization file have to be placed in directory hwplugins. And must have extension .tcl! For example consider again our “My First Plug-in v1.0” plug-in, as we mentioned before. The plug-in directory is: “/usr/share/mcu8051ide/hwplugins/My_First_Plug-in_v1.0” then the initialization file is: “/usr/share/mcu8051ide/hwplugins/My_First_Plug-in_v1.0.tcl”.

3. Define mandatory variables

```
set AUTHOR      "<your name>"           ;# e.g. "Homer Simpson"
set EMAIL       "<your_email@example.com>" ;# e.g. "superman.spiderman@supehero.ru"
set P_VERSION   "<version_of_your_plug-in>" ;# e.g. "1.2.3" or "0.9"
set MIN_IDE_VER "1.1"                   ;# Minimal required version of MCU 8051 IDE
```

4. Define mandatory functions

```
# Free resources occupied by this plugin
proc dispose {} { ... }

# Initialize the plug-in
proc init {main_frame current_namespace directory} { ... }

# Restore previous session
proc restore_session {session_data} { ... }

# Save plug-in session
proc save_session {} { ...; return <String> }

# Is plugin busy ?
proc is_busy {} { ...; return <BooleanValue> }
```


“...” means any code you want there. See “/usr/share/mcu8051ide/hwplugins/plugin_template.txt” or “<YourInstallationDirectory>\hwplugins\plugin_template.txt” for more details and for a template for such file.

When you have these steps completed you have prepared basic environment for the plug-in. Then the “HW plug-ins manager” in the right panel in IDE’s GUI should now recognize your plug-in and be able to attempt to load it. If it is not so, then there is definitely something wrong. Any other files which your plug-in consist of and just whatever you want there should be placed in your plug-in directory (1). And the initialization file should do nothing else than source some real plug-in’s file(s) and call appropriate functions inside them. One more important thing, the plug-in runs it **dynamically assigned namespace**. Take it into account, otherwise your plug-in wont work! Function “init” takes the name of this namespace in parameter “current_namespace”. So as you can see, it’s quite easy you have just to define 4 variables and 5 functions and you can interface with the IDE.

7.3 Using MCU 8051 IDE API

You can used any part the API you want, but the entire IDE’s API is wast and may change in future without notice. So there is special API dedicated to use in hardware control plug-ins, it is located in “::HwManager” namespace and consists of merely 10 simple functions. This API is available since version 1.4. Here is the list of its functions:

```
## Check whether there is some project opened in the IDE
# @return Bool - 1 == Yes, there is; 0 == No there is not
proc is_project_opened {}

## Check whether MCU simulator is engaged
# @return Bool - 0 == 1 == Yes, it is; No it is not (or no project is opened)
proc is_simulator_engaged {}

## Get full name of file which is currently displayed in the source code editor
# @return String - Full file name including path or empty string in case there is no project opened
proc get_current_file {}

## Get full name of file which has been chosen as the project main file
# @return String - Full file name or empty string
proc get_project_main_file {}

## Get path the directory of currently active project
# @return String - Directory path or empty string in case there is no project opened
proc get_project_dir {}

## Get name of the current project
# @return String - Name of the current project or empty string in case there is no project opened
proc get_project_name {}

## Initiate compilation if at least one of the source files was modified
```

```
# @parm String success_callback - Any command to execute after successful compilation
# @parm String failure_callback - Any command to execute after unsuccessful compilation
# @return Bool - 1 == Process successfully started; 0 == Unable to comply (no project is opened)
proc compile_if_necessary_and_callback {success_callback failure_callback}

## Open the specified Intel® 8 hex file in hexadecimal editor
# @parm String filename - Name of file to open (including path)
# @return Bool - 1 == Success; 0 == Failure
proc open_in_hexeditor {filename}

## Start MCU simulator if possible
# @return Bool - 1 == Success; 0 == Unable to comply
proc start_simulator {}

## Shutdown MCU simulator if possible
# @return Bool - 1 == Success; 0 == Unable to comply
proc shutdown_simulator {}
```

7.4 A basic example

Lets write just a simple plug-in which merely demonstrates usage of some of the above mentioned functions.

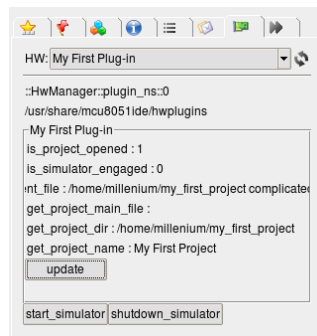


Figure 7.2: A basic example of a plug-in

```
set AUTHOR      "Homer Simpson"
set EMAIL       "superman.spiderman@supehero.ru"
set P_VERSION   "1.0"
set MIN_IDE_VER "1.3.11"

proc dispose {} {
    tk_messageBox \
        -title "My First Plug-in" \
        -message "Called: dispose {}"
}

proc init {main_frame project_object current_namespace directory} {
    pack [label $main_frame.10 -text $current_namespace] -anchor w
    pack [label $main_frame.11 -text $directory] -anchor w

    set f [labelframe $main_frame.f -text "My First Plug-in"]
    pack [label $f.10 -text "is_project_opened : [::HwManager::is_project_opened]" ] -anchor w
```

```

pack [label $f.l1 -text "is_simulator_engaged : [::HwManager::is_simulator_engaged]" ] -anchor w
pack [label $f.l2 -text "get_current_file : [::HwManager::get_current_file]" ] -anchor w
pack [label $f.l3 -text "get_project_main_file : [::HwManager::get_project_main_file]" ] -anchor w
pack [label $f.l4 -text "get_project_dir : [::HwManager::get_project_dir]" ] -anchor w
pack [label $f.l5 -text "l : [::HwManager::get_project_name]" ] -anchor w

pack [ttk::button $f.b0 -text "update" -command "
    $f.l0 configure -text \"is_project_opened : \[::HwManager::is_project_opened\]"
    $f.l1 configure -text \"is_simulator_engaged : \[::HwManager::is_simulator_engaged\]"
    $f.l2 configure -text \"get_current_file : \[::HwManager::get_current_file\]"
    $f.l3 configure -text \"get_project_main_file : \[::HwManager::get_project_main_file\]"
    $f.l4 configure -text \"get_project_dir : \[::HwManager::get_project_dir\]"
    $f.l5 configure -text \"get_project_name : \[::HwManager::get_project_name\]"
"] -anchor w
pack $f -fill both -expand 1

pack [ttk::button $main_frame.b1 \
    -text "start_simulator" \
    -command {::HwManager::start_simulator} \
] -side left
pack [ttk::button $main_frame.b2 \
    -text "shutdown_simulator" \
    -command {::HwManager::shutdown_simulator} \
] -side left
}

proc restore_session {session_data} {
    tk_messageBox \
        -title "My First Plug-in" \
        -message "Called: restore_session {$session_data}"
}

proc save_session {} {
    tk_messageBox \
        -title "My First Plug-in" \
        -message "Called: save_session {}"
    return "my data, time: [clock format [clock seconds] -format {%T}]"
}

proc is_busy {} {
    return [expr {
        [tk_messageBox \
            -title "My First Plug-in" \
            -message "Called: is_busy {}" \
            -type {yesno}
        ]
        == {yes}}]
}

```

7.5 Random remarks

- Don't forget that your plug-in runs the main thread as well as the GUI of the entire IDE. So if your plug-in does some time expensive operations and it is probable that it does. Then these operations have to be performed in either separate thread or have to run in separate process. It is also possible to regularly "update" the application by reentering the event loop using Tcl's **update** command.
- Plug-in files must use encoding UTF-8 and should use LF (Line Feed) char-

acter as line end delimiter. In other words Unix line termination sequence.

- Although it is possible to name the plug-in directory in any way what your OS accept. It is generally a good idea to follow the mentioned recommendation. At least the name of the initialization file have to follow the mentioned recommendation.
- Plug-ins have unrestricted access to the entire application. So they should be written carefully, because plug-ins can theoretically crash down the entire IDE.
- Program errors which occurs during loading or unloading of a plug-in are reported via a special dialog. In this dialog plug-in author and his or her email address are mentioned.
- The above mentioned API provided by “`::HwManager`” is just a facade¹. Before version 1.4 plug-ins must have had to be written to access directly the functionality currently hidden behind “`::HwManager`”, so it was much more complicated.
- Each instance of a hardware plug-ins manager is bond to its project. But actual plug-ins don't have to be. That's the reason why there is function “`::HwManager::is_project_opened`”. All functions inside the above mentioned API (`::HwManager::*`) works with the current project, not necessarily with the project which is the HW manager bond with.

¹A design pattern as described in the GOF book

Chapter 8

Command Line Interface

MCU 8051 IDE's CLI makes it possible to use entire IDE just as an assembler, disassembler or converter between .hex files and binary files. MCU 8051 IDE supports these switches:

Switch	Meaning
General	
-help, -h	Show help for CLI
-quiet, -q	Do not show initialization progress on start-up
-nosplash	Do not show the splash screen
-nocolor, -n	Do not show colorful output in console
-version, -V	Show program version and exit
-defaults	Run program in empty session
-minimalized	Run in minimalized window
-config-file filename	Specify path to an alternative configuration file
-check-libraries	Verify whether all required libraries are available
-ignore-last-session	Start with an empty session
-open-project project	Open just this project
-reset-user-settings	Reset all user setting to defaults
Data conversions	
-auto-indent input	Reformat indentation the specified file
-hex2bin input output	Convert Intel 8 Hex into a binary file
-bin2hex input output	Convert a binary file in Intel 8 HEX
-sim2hex input output	Convert MCU 8051 IDE simulator file to Intel 8 Hex file
-sim2bin input output	Convert MCU 8051 IDE simulator file to binary file
-normalize-hex input	Normalize Intel 8 HEX (force incremental addressing order)
Assembler/Disassembler	
-disassemble hex_file	Disassemble Intel 8 HEX file to hex_file.asm
-assemble asm_file	Assemble the specified file
-compile asm_file	The same as "--assemble"
-iram-size size	Set size of internal data memory for assembler
-code-size size	Set size of program data memory for assembler
-xram-size size	Set size of external data memory for assembler
-no-opt	Disable peephole optimization
-comp-quiet	Suppress text output from the assembler
-no-sim	Disable generation of .adf file
-no-bin	Disable generation of .bin file
-no-lst	Disable generation of .lst file

Switch	Meaning
-no-hex	Disable generation of .hex file
-warning-level 0-3	Set warning level to the specified level

Interesting examples:

```
# Reset all IDE settings to defaults
mcu8051ide --reset-user-settings

# Use MCU 8051 IDE as assembler (without GUI)
mcu8051ide --compile /some_directory/my_file.asm

# Use MCU 8051 IDE as dissembler (without GUI)
mcu8051ide --disassemble /some_directory/my_file.hex

# Use MCU 8051 IDE as convertor from binary files to Intel 8 HEX (without GUI)
mcu8051ide --bin2hex /some_directory/my_file /some_directory/my_file.hex
```

Chapter 9

Translating the IDE into different languages

The IDE can be translated to almost any language. The translation can be accomplished by creating of a translation definition file. Such a file must follow certain strict rules in order to work properly. Translation files are normally located in directory “/usr/share/mcu8051ide/translations”, on Microsoft®Windows®the directory is “<YourInstallationDirectory>\translations”. There you can find file “template.txt” which is template of MCU 8051 IDE translation file. Along with it there is also file “languages.txt” which defines names of languages to which the IDE was already translated. These names are written in these languages. Translation files look like this “ru.msg” (Russian translation) or “cs.msg” (Czech translation), these files need to be regularly updated. Note also that these files are quite big, each about 0.5MB and each contains about 4500 translated sentences. Further details regarding the translation are mentioned directly in the files related to translation, particularly in file “template.txt”. Refer to them if you are interested in making your own translation of the IDE. This is an open-source project so any help is appreciated.

The first several lines in file “template.txt”:

```
# This is a template of MCU 8051 IDE translation file
#
# This file allows to localize the the user environment of the IDE to almost any
# language.
#
# HOW TO MAKE IT WORK:
# 1) Copy this file (template.txt) to <lang_code>.msg in the same directory.
#    Where '<lang_code>' is supposed to be replaced with language code of
#    the translation. For example 'ru' means Russian, or 'cs' means Czech.
#    The language code must be lowercase.
# 2) Translate all sentences enclosed by '§' (paragraph) character. And be
#    sure to remove the '§' character.
# 3) Modify file 'languages.tcl' and add name of language which you are
```

CHAPTER 9. TRANSLATING THE IDE INTO DIFFERENT LANGUAGES

```
#      making the translation for. Name should be specified in that language.
#
# IMPORTANT RULES FOR TRANSLATION:
# 1) Be aware of that this file is very sensitive.
# 2) Everything besides actual sentences for translation must not be modified
#    in any way! Otherwise the file might cause serious program instability.
# 3) Escape sequences and all special characters must be preserved.
# 4) Sentences enclosed with ‘‘’’ (double quote) character, can be translated
#    into sentences with different length. But the same does not apply for
#    sentences enclosed with ‘{’ and ‘}’ (curly brackets) characters,
#    their lengths must stay preserved.
# 5) Do not translate ‘$’ dollar symbol, it has a special meaning here, not
#    related to currency.
# 6) Keep UTF-8 encoding and if possible, please keep also Unix line ends.
#
# NOTES:
# 1) ‘;# <-- NOT TRANSLATED YET’ is just a comment and can be removed at
#    any time
# 2) Nothing is perfect ... if you find anything strange or not functional
#    here, please report it as a regular bug.
# 3) Recommended syntax highlight pattern for this file is "Tcl/Tk"
# 4) Please don't hesitate to ask any questions.
#
# Thank you for your cooperation, which helps us make the software better!
#
```

A random piece of the translation definition file template

```
mcset $1 "Replace all" \
    "$Replace all$" ;# <-- NOT TRANSLATED YET
mcset $1 "Find next" \
    "$Find next$" ;# <-- NOT TRANSLATED YET
mcset $1 "Close" \
    "$Close$" ;# <-- NOT TRANSLATED YET
mcset $1 "Replace confirmation - %s" \
    "$Replace confirmation - %s$" ;# <-- NOT TRANSLATED YET
mcset $1 "Go to line" \
    "$Go to line$" ;# <-- NOT TRANSLATED YET
mcset $1 "Ok" \
    "$Ok$" ;# <-- NOT TRANSLATED YET
mcset $1 "Cancel" \
    "$Cancel$" ;# <-- NOT TRANSLATED YET
mcset $1 "Go to line - MCU 8051 IDE" \
    "$Go to line - MCU 8051 IDE$" ;# <-- NOT TRANSLATED YET
mcset $1 "Choose directory - MCU 8051 IDE" \
    "$Choose directory - MCU 8051 IDE$" ;# <-- NOT TRANSLATED YET
```


Appendix A

License

MCU 8051 IDE and all its components is distributed under terms of GNU GPLv2.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundations software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each authors protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program).

Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Programs source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions

either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the programs name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

```
The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.
```

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yooyodine, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B

Regression testing

B.1 Foreword

The IDE is featured with a regression testing environment, the aim of this to make the software as reliable as possible. Currently there is prepared environment for testing the simulator engine and the built-in assembler. This allows to write test cases for specific software features and check whether the results of these tests conform to expected results. Regretfully the test cases are **NOT PREPARED YET**. Each time when a change is made to the IDE's code, these regression test should be run in order to detect possible bug introductions caused by recent changes. Regression test also serves as a proof of certain software functionality and reliability.



```

assembler - bash
$ ./runtest
Starting Assembler regression testing ... 3 testcases to go
-----
Testcase: "001_Trivial_test" [OK]
Testcase: "002_Dummy_test" [OK]
Testcase: "003_Dummy_test" [OK]
-----
Statistic:
TOTAL: 3
SUCCESSFUL: 3
FAILED: 0
  
```

Figure B.1: Assembler regression test run in terminal emulator

B.2 More about the implementation

Additional details can be found in the MCU 8051 IDE development snapshot downloaded from the project's GIT repository in directory "regression_tests" in various "README" files. Here we will mention just general overview.

Each test have its own directory, like 'test_something' or 'another_test', let's call this directory the test directory. Each test consist of a set of test cases. Each test case should test one and only one specific function of the tested software. Test cases are represented by files with extension `.in` located in directory named 'testcases' inside the test directory.

Directory "results" inside the test directory should be left empty. It is used by the testing environment for storing temporary files generated during the tests. The

“testcases” directory also contains files intended for comparison with files generated during the test and stored in the “results” directory, these files must have extension “.<x>.exp”. Where “<x>” must be substituted with extension of a file which this file is supposed to be compared to. In another words, if I want to check whether for example “./results/something.abc” was generated as it should be, I have to create file “./testcases/something.abc.exp” and this file will be automatically compared with “./results/something.abc”.

And that’s it! This is simple, isn’t it? It’s just about comparing files. But how are the tests run and how the files in the “results” directory gets generated? For that we need some Bash script, which is used to run the test, let’s call this script the runtest script. The runtest script must be located in the test directory and must include the “rte.lib.sh” file, using the “source” command (or ‘.’ command). This script should have set permissions to be executable and this script specifies how exactly should be the test performed and also runs the test itself.

When the script is about to exit, this condition is trapped and the ‘rte.lib.sh’ reacts by starting the test. So there is no need to explicitly run the test by invoking some function or something like that. It runs the test automatically when there is nothing else left to do.

Appendix C

Project web page and other media

C.1 Official project web page

Official web page of the MCU 8051 IDE project provides basic presentation of the project. Contains news about the project development, users comments and forums for users. Also the project's hardware tools are described here and there is some personal information about authors of the project. All components of the IDE can be downloaded from sourceforge.net, which provides hosting for the web pages and entire project. The address is <http://mcu8051ide.sf.net>.

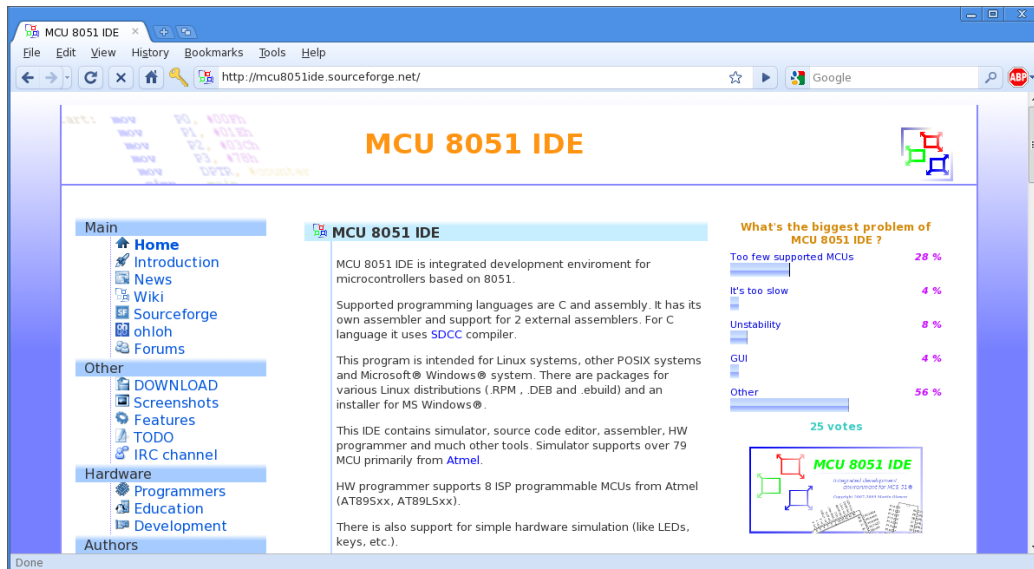


Figure C.1: Official web page of the MCU 8051 IDE project

C.2 Other media

Project has also its own page on Source Forge, ohloh and Fresh meat. Installation packages are in official Fedora repositories and Ubuntu repositories. There is currently also one Gentoo overlay providing ebuild for the IDE. Project has its own IRC channel, although it is rarely used. And wiki pages. Not yet in a good shape. Project is also mentioned on Wikipedia. Project official web page is written in PHP5, XHTML-1.1, CSS2 and JavaScript and uses MySQL as database. Volunteers who would like to improve the web page are also welcomed as contributors to the project.

C.3 GIT repository

GIT is a distributed revision control system originally developed by Linus Torvalds. MCU 8051 IDE also takes advantage of GIT and uses it as its tool for managing current development version. The project's GIT repository is hosted by Source Forge and is available at address "`git://mcu8051ide.git.sourceforge.net/gitroot/mcu8051ide/mcu8051ide`". Access to the repository is for reading only unless you possess the required clearance. In the GIT repository you can always find the newest development version with the newest bug fixes and features. List of latest changes is available on <http://mcu8051ide.git.sourceforge.net>. Here is a short description to download and install the latest development version of the IDE:

1. Install GIT
2. Run "`git clone git://mcu8051ide.git.sourceforge.net/gitroot/mcu8051ide/mcu8051ide`". It will create your own copy of the Git repository in the current directory.
3. Once you have an existing copy of the repository you can just update it each time when you want the fresh version by this command:
`git fetch origin master`
4. Then you can try the downloaded IDE version without installation using the following sequence of commands (for POSIX only)
 - (a) `cd mcu8051ide/lib`
 - (b) `./main.tcl`
5. Or install it and use it using the following sequence of commands (for POSIX only)
 - (a) `cd mcu8051ide`
 - (b) `./configure && make`
 - (c) `sudo su # or just "su"`
 - (d) `make install`

Appendix D

8051 Instructions in numerical Order

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x00	NOP		1		1
0x01	AJMP	code11	2		2
0x02	LJMP	code16	3		2
0x03	RR	A	1		1
0x04	INC	A	1	P	1
0x05	INC	data	2		1
0x06	INC	@R0	1		1
0x07	INC	@R1	1		1
0x08	INC	R0	1		1
0x09	INC	R1	1		1
0x0A	INC	R2	1		1
0x0B	INC	R3	1		1
0x0C	INC	R4	1		1
0x0D	INC	R5	1		1
0x0E	INC	R6	1		1
0x0F	INC	R7	1		1
0x10	JBC	bit code8	3		2
0x11	ACALL	code11	2		2
0x12	LCALL	code16	3		2
0x13	RRC	A	1	CY P	1
0x14	DEC	A	1	P	1
0x15	DEC	data	2		1
0x16	DEC	@R0	1		1
0x17	DEC	@R1	1		1
0x18	DEC	R0	1		1
0x19	DEC	R1	1		1
0x1A	DEC	R2	1		1
0x1B	DEC	R3	1		1
0x1C	DEC	R4	1		1
0x1D	DEC	R5	1		1
0x1E	DEC	R6	1		1
0x1F	DEC	R7	1		1
0x20	JB	bit code8	3		2
0x21	AJMP	code11	2		2

APPENDIX D. 8051 INSTRUCTIONS IN NUMERICAL ORDER

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x22	RET		1		2
0x23	RL	A	1		1
0x24	ADD	A #imm8	2	CY AC OV P	1
0x25	ADD	A data	2	CY AC OV P	1
0x26	ADD	A @R0	1	CY AC OV P	1
0x27	ADD	A @R1	1	CY AC OV P	1
0x28	ADD	A R0	1	CY AC OV P	1
0x29	ADD	A R1	1	CY AC OV P	1
0x2A	ADD	A R2	1	CY AC OV P	1
0x2B	ADD	A R3	1	CY AC OV P	1
0x2C	ADD	A R4	1	CY AC OV P	1
0x2D	ADD	A R5	1	CY AC OV P	1
0x2E	ADD	A R6	1	CY AC OV P	1
0x2F	ADD	A R7	1	CY AC OV P	1
0x30	JNB	bit code8	3		2
0x31	ACALL	code11	2		2
0x32	RETI		1		2
0x33	RLC	A	1	CY	P 1
0x34	ADDC	A #imm8	2	CY AC OV P	1
0x35	ADDC	A data	2	CY AC OV P	1
0x36	ADDC	A @R0	1	CY AC OV P	1
0x37	ADDC	A @R1	1	CY AC OV P	1
0x38	ADDC	A R0	1	CY AC OV P	1
0x39	ADDC	A R1	1	CY AC OV P	1
0x3A	ADDC	A R2	1	CY AC OV P	1
0x3B	ADDC	A R3	1	CY AC OV P	1
0x3C	ADDC	A R4	1	CY AC OV P	1
0x3D	ADDC	A R5	1	CY AC OV P	1
0x3E	ADDC	A R6	1	CY AC OV P	1
0x3F	ADDC	A R7	1	CY AC OV P	1
0x40	JC	code8	2		2
0x41	AJMP	code11	2		2
0x42	ORL	data A	2		1
0x43	ORL	data #imm8	3		2
0x44	ORL	A #imm8	2		P 1
0x45	ORL	A data	2		P 1
0x46	ORL	A @R0	1		P 1
0x47	ORL	A @R1	1		P 1
0x48	ORL	A R0	1		P 1
0x49	ORL	A R1	1		P 1
0x4A	ORL	A R2	1		P 1
0x4B	ORL	A R3	1		P 1
0x4C	ORL	A R4	1		P 1
0x4D	ORL	A R5	1		P 1
0x4E	ORL	A R6	1		P 1
0x4F	ORL	A R7	1		P 1
0x50	JNC	code8	2		2
0x51	ACALL	code11	2		2
0x52	ANL	data A	2		1
0x53	ANL	data #imm8	3		2
0x54	ANL	A #imm8	2		P 1
0x55	ANL	A data	2		P 1
0x56	ANL	A @R0	1		P 1
0x57	ANL	A @R1	1		P 1
0x58	ANL	A R0	1		P 1

Opcode	Mnemonic	Operands		Bytes	Flags	Cycles
0x59	ANL	A	R1	1	P	1
0x5A	ANL	A	R2	1	P	1
0x5B	ANL	A	R3	1	P	1
0x5C	ANL	A	R4	1	P	1
0x5D	ANL	A	R5	1	P	1
0x5E	ANL	A	R6	1	P	1
0x5F	ANL	A	R7	1	P	1
0x60	JZ	code8		2		2
0x61	AJMP	code11		2		2
0x62	XRL	data	A	2		1
0x63	XRL	data	#imm8	3		2
0x64	XRL	A	#imm8	2	P	1
0x65	XRL	A	data	2	P	1
0x66	XRL	A	@R0	1	P	1
0x67	XRL	A	@R1	1	P	1
0x68	XRL	A	R0	1	P	1
0x69	XRL	A	R1	1	P	1
0x6A	XRL	A	R2	1	P	1
0x6B	XRL	A	R3	1	P	1
0x6C	XRL	A	R4	1	P	1
0x6D	XRL	A	R5	1	P	1
0x6E	XRL	A	R6	1	P	1
0x6F	XRL	A	R7	1	P	1
0x70	JNZ	code8		2		2
0x71	ACALL	code11		2		2
0x72	ORL	C	bit	2	CY	2
0x73	JMP	@A+DPTR		1		2
0x74	MOV	A	#imm8	2	P	1
0x75	MOV	data	#imm8	3		2
0x76	MOV	@R0	#imm8	2		1
0x77	MOV	@R1	#imm8	2		1
0x78	MOV	R0	#imm8	2		1
0x79	MOV	R1	#imm8	2		1
0x7A	MOV	R2	#imm8	2		1
0x7B	MOV	R3	#imm8	2		1
0x7C	MOV	R4	#imm8	2		1
0x7D	MOV	R5	#imm8	2		1
0x7E	MOV	R6	#imm8	2		1
0x7F	MOV	R7	#imm8	2		1
0x80	SJMP	code8		2		2
0x81	AJMP	code11		2		2
0x82	ANL	C	bit	2	CY	2
0x83	MOVC	A	@A+PC	1	P	2
0x84	DIV	AB		1	CY OV P	4
0x85	MOV	data	data	3		2
0x86	MOV	data	@R0	2		2
0x87	MOV	data	@R1	2		2
0x88	MOV	data	R0	2		2
0x89	MOV	data	R1	2		2
0x8A	MOV	data	R2	2		2
0x8B	MOV	data	R3	2		2
0x8C	MOV	data	R4	2		2
0x8D	MOV	data	R5	2		2
0x8E	MOV	data	R6	2		2
0x8F	MOV	data	R7	2		2

APPENDIX D. 8051 INSTRUCTIONS IN NUMERICAL ORDER

90

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x90	MOV	DPTR #imm16	3		2
0x91	ACALL	code11	2		2
0x92	MOV	bit C	2		2
0x93	MOVC	A @A+DPTR	1		P 2
0x94	SUBB	A #imm8	2	CY AC OV P	1
0x95	SUBB	A data	2	CY AC OV P	1
0x96	SUBB	A @R0	1	CY AC OV P	1
0x97	SUBB	A @R1	1	CY AC OV P	1
0x98	SUBB	A R0	1	CY AC OV P	1
0x99	SUBB	A R1	1	CY AC OV P	1
0x9A	SUBB	A R2	1	CY AC OV P	1
0x9B	SUBB	A R3	1	CY AC OV P	1
0x9C	SUBB	A R4	1	CY AC OV P	1
0x9D	SUBB	A R5	1	CY AC OV P	1
0x9E	SUBB	A R6	1	CY AC OV P	1
0x9F	SUBB	A R7	1	CY AC OV P	1
0xA0	ORL	C /bit	2	CY	2
0xA1	AJMP	code11	2		2
0xA2	MOV	C bit	2	CY	1
0xA3	INC	DPTR	1		2
0xA4	MUL	AB	1	CY OV P	4
0xA5	Invalid opcode				
0xA6	MOV	@R0 data	2		2
0xA7	MOV	@R1 data	2		2
0xA8	MOV	R0 data	2		2
0xA9	MOV	R1 data	2		2
0xAA	MOV	R2 data	2		2
0xAB	MOV	R3 data	2		2
0xAC	MOV	R4 data	2		2
0xAD	MOV	R5 data	2		2
0xAE	MOV	R6 data	2		2
0xAF	MOV	R7 data	2		2
0xB0	ANL	C /bit	2	CY	2
0xB1	ACALL	code11	2		2
0xB2	CPL	bit	2		1
0xB3	CPL	C	1	CY	1
0xB4	CJNE	A #imm8 code8	3	CY	2
0xB5	CJNE	A data code8	3	CY	2
0xB6	CJNE	@R0 #imm8 code8	3	CY	2
0xB7	CJNE	@R1 #imm8 code8	3	CY	2
0xB8	CJNE	R0 #imm8 code8	3	CY	2
0xB9	CJNE	R1 #imm8 code8	3	CY	2
0xBA	CJNE	R2 #imm8 code8	3	CY	2
0xBB	CJNE	R3 #imm8 code8	3	CY	2
0xBC	CJNE	R4 #imm8 code8	3	CY	2
0xBD	CJNE	R5 #imm8 code8	3	CY	2
0xBE	CJNE	R6 #imm8 code8	3	CY	2
0xBF	CJNE	R7 #imm8 code8	3	CY	2
0xC0	PUSH	data	2		2
0xC1	AJMP	code11	2		2
0xC2	CLR	bit	2		1
0xC3	CLR	C	1	CY	1
0xC4	SWAP	A	1		1
0xC5	XCH	A data	2		P 1
0xC6	XCH	A @R0	1		P 1

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0xC7	XCH	A @R1	1	P	1
0xC8	XCH	A R0	1	P	1
0xC9	XCH	A R1	1	P	1
0xCA	XCH	A R2	1	P	1
0xCB	XCH	A R3	1	P	1
0xCC	XCH	A R4	1	P	1
0xCD	XCH	A R5	1	P	1
0xCE	XCH	A R6	1	P	1
0xCF	XCH	A R7	1	P	1
0xD0	POP	data	2		2
0xD1	ACALL	code11	2		2
0xD2	SETB	bit	2		1
0xD3	SETB	C	1	CY	1
0xD4	DA	A	1	CY P	1
0xD5	DJNZ	data code8	3		2
0xD6	XCHD	A @R0	1	P	1
0xD7	XCHD	A @R1	1	P	1
0xD8	DJNZ	R0 code8	2		2
0xD9	DJNZ	R1 code8	2		2
0xDA	DJNZ	R2 code8	2		2
0xDB	DJNZ	R3 code8	2		2
0xDC	DJNZ	R4 code8	2		2
0xDD	DJNZ	R5 code8	2		2
0xDE	DJNZ	R6 code8	2		2
0xDF	DJNZ	R7 code8	2		2
0xE0	MOVX	A @DPTR	1	P	2
0xE1	AJMP	code11	2		2
0xE2	MOVX	A @R0	1	P	2
0xE3	MOVX	A @R1	1	P	2
0xE4	CLR	A	1	P	1
0xE5	MOV	A data	2	P	1
0xE6	MOV	A @R0	1	P	1
0xE7	MOV	A @R1	1	P	1
0xE8	MOV	A R0	1	P	1
0xE9	MOV	A R1	1	P	1
0xEA	MOV	A R2	1	P	1
0xEB	MOV	A R3	1	P	1
0xEC	MOV	A R4	1	P	1
0xED	MOV	A R5	1	P	1
0xEE	MOV	A R6	1	P	1
0xEF	MOV	A R7	1	P	1
0xF0	MOVX	@DPTR A	1		2
0xF1	ACALL	code11	2		2
0xF2	MOVX	@R0 A	1		2
0xF3	MOVX	@R1 A	1		2
0xF4	CPL	A	1	P	1
0xF5	MOV	data A	2		1
0xF6	MOV	@R0 A	1		1
0xF7	MOV	@R1 A	1		1
0xF8	MOV	R0 A	1		1
0xF9	MOV	R1 A	1		1
0xFA	MOV	R2 A	1		1
0xFB	MOV	R3 A	1		1
0xFC	MOV	R4 A	1		1
0xFD	MOV	R5 A	1		1

**APPENDIX D. 8051 INSTRUCTIONS IN NUMERICAL
ORDER**

92

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0xFE	MOV	R6 A	1		1
0xFF	MOV	R7 A	1		1

Table D.1: 8051 Instructions in numerical Order

Appendix E

8051 Instructions in alphabetical order

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x11	ACALL	code11	2		2
0x31	ACALL	code11	2		2
0x51	ACALL	code11	2		2
0x71	ACALL	code11	2		2
0x91	ACALL	code11	2		2
0xB1	ACALL	code11	2		2
0xD1	ACALL	code11	2		2
0xF1	ACALL	code11	2		2
0x24	ADD	A #imm8	2	CY AC OV P	1
0x25	ADD	A data	2	CY AC OV P	1
0x26	ADD	A @R0	1	CY AC OV P	1
0x27	ADD	A @R1	1	CY AC OV P	1
0x28	ADD	A R0	1	CY AC OV P	1
0x29	ADD	A R1	1	CY AC OV P	1
0x2A	ADD	A R2	1	CY AC OV P	1
0x2B	ADD	A R3	1	CY AC OV P	1
0x2C	ADD	A R4	1	CY AC OV P	1
0x2D	ADD	A R5	1	CY AC OV P	1
0x2E	ADD	A R6	1	CY AC OV P	1
0x2F	ADD	A R7	1	CY AC OV P	1
0x34	ADDC	A #imm8	2	CY AC OV P	1
0x35	ADDC	A data	2	CY AC OV P	1
0x36	ADDC	A @R0	1	CY AC OV P	1
0x37	ADDC	A @R1	1	CY AC OV P	1
0x38	ADDC	A R0	1	CY AC OV P	1
0x39	ADDC	A R1	1	CY AC OV P	1
0x3A	ADDC	A R2	1	CY AC OV P	1
0x3B	ADDC	A R3	1	CY AC OV P	1
0x3C	ADDC	A R4	1	CY AC OV P	1
0x3D	ADDC	A R5	1	CY AC OV P	1
0x3E	ADDC	A R6	1	CY AC OV P	1
0x3F	ADDC	A R7	1	CY AC OV P	1
0x01	AJMP	code11	2		2
0x21	AJMP	code11	2		2

APPENDIX E. 8051 INSTRUCTIONS IN ALPHABETICAL ORDER

94

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x41	AJMP	code11	2		2
0x61	AJMP	code11	2		2
0x81	AJMP	code11	2		2
0xA1	AJMP	code11	2		2
0xC1	AJMP	code11	2		2
0xE1	AJMP	code11	2		2
0x52	ANL	data A	2		1
0x53	ANL	data #imm8	3		2
0x54	ANL	A #imm8	2		P 1
0x55	ANL	A data	2		P 1
0x56	ANL	A @R0	1		P 1
0x57	ANL	A @R1	1		P 1
0x58	ANL	A R0	1		P 1
0x59	ANL	A R1	1		P 1
0x5A	ANL	A R2	1		P 1
0x5B	ANL	A R3	1		P 1
0x5C	ANL	A R4	1		P 1
0x5D	ANL	A R5	1		P 1
0x5E	ANL	A R6	1		P 1
0x5F	ANL	A R7	1		P 1
0x82	ANL	C bit	2	CY	2
0xB0	ANL	C /bit	2	CY	2
0xB4	CJNE	A #imm8 code8	3	CY	2
0xB5	CJNE	A data code8	3	CY	2
0xB6	CJNE	@R0 #imm8 code8	3	CY	2
0xB7	CJNE	@R1 #imm8 code8	3	CY	2
0xB8	CJNE	R0 #imm8 code8	3	CY	2
0xB9	CJNE	R1 #imm8 code8	3	CY	2
0xBA	CJNE	R2 #imm8 code8	3	CY	2
0xBB	CJNE	R3 #imm8 code8	3	CY	2
0xBC	CJNE	R4 #imm8 code8	3	CY	2
0xBD	CJNE	R5 #imm8 code8	3	CY	2
0xBE	CJNE	R6 #imm8 code8	3	CY	2
0xBF	CJNE	R7 #imm8 code8	3	CY	2
0xC2	CLR	bit	2		1
0xC3	CLR	C	1	CY	1
0xE4	CLR	A	1		P 1
0xB2	CPL	bit	2		1
0xB3	CPL	C	1	CY	1
0xF4	CPL	A	1		P 1
0xD4	DA	A	1	CY	P 1
0x14	DEC	A	1		P 1
0x15	DEC	data	2		1
0x16	DEC	@R0	1		1
0x17	DEC	@R1	1		1
0x18	DEC	R0	1		1
0x19	DEC	R1	1		1
0x1A	DEC	R2	1		1
0x1B	DEC	R3	1		1
0x1C	DEC	R4	1		1
0x1D	DEC	R5	1		1
0x1E	DEC	R6	1		1
0x1F	DEC	R7	1		1
0x84	DIV	AB	1	CY OV P	4
0xD5	DJNZ	data code8	3		2

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0xD8	DJNZ	R0 code8	2		2
0xD9	DJNZ	R1 code8	2		2
0xDA	DJNZ	R2 code8	2		2
0xDB	DJNZ	R3 code8	2		2
0xDC	DJNZ	R4 code8	2		2
0xDD	DJNZ	R5 code8	2		2
0xDE	DJNZ	R6 code8	2		2
0xDF	DJNZ	R7 code8	2		2
0x04	INC	A	1	P	1
0x05	INC	data	2		1
0x06	INC	@R0	1		1
0x07	INC	@R1	1		1
0x08	INC	R0	1		1
0x09	INC	R1	1		1
0x0A	INC	R2	1		1
0x0B	INC	R3	1		1
0x0C	INC	R4	1		1
0x0D	INC	R5	1		1
0x0E	INC	R6	1		1
0x0F	INC	R7	1		1
0xA3	INC	DPTR	1		2
0x20	JB	bit code8	3		2
0x10	JBC	bit code8	3		2
0x40	JC	code8	2		2
0x73	JMP	@A+DPTR	1		2
0x30	JNB	bit code8	3		2
0x50	JNC	code8	2		2
0x70	JNZ	code8	2		2
0x60	JZ	code8	2		2
0x12	LCALL	code16	3		2
0x02	LJMP	code16	3		2
0x74	MOV	A #imm8	2	P	1
0x75	MOV	data #imm8	3		2
0x76	MOV	@R0 #imm8	2		1
0x77	MOV	@R1 #imm8	2		1
0x78	MOV	R0 #imm8	2		1
0x79	MOV	R1 #imm8	2		1
0x7A	MOV	R2 #imm8	2		1
0x7B	MOV	R3 #imm8	2		1
0x7C	MOV	R4 #imm8	2		1
0x7D	MOV	R5 #imm8	2		1
0x7E	MOV	R6 #imm8	2		1
0x7F	MOV	R7 #imm8	2		1
0x85	MOV	data data	3		2
0x86	MOV	data @R0	2		2
0x87	MOV	data @R1	2		2
0x88	MOV	data R0	2		2
0x89	MOV	data R1	2		2
0x8A	MOV	data R2	2		2
0x8B	MOV	data R3	2		2
0x8C	MOV	data R4	2		2
0x8D	MOV	data R5	2		2
0x8E	MOV	data R6	2		2
0x8F	MOV	data R7	2		2
0x90	MOV	DPTR #imm16	3		2

**APPENDIX E. 8051 INSTRUCTIONS IN ALPHABETICAL
ORDER**

96

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x92	MOV	bit C	2		2
0xA2	MOV	C bit	2	CY	1
0xA6	MOV	@R0 data	2		2
0xA7	MOV	@R1 data	2		2
0xA8	MOV	R0 data	2		2
0xA9	MOV	R1 data	2		2
0xAA	MOV	R2 data	2		2
0xAB	MOV	R3 data	2		2
0xAC	MOV	R4 data	2		2
0xAD	MOV	R5 data	2		2
0xAE	MOV	R6 data	2		2
0xAF	MOV	R7 data	2		2
0xE5	MOV	A data	2		P 1
0xE6	MOV	A @R0	1		P 1
0xE7	MOV	A @R1	1		P 1
0xE8	MOV	A R0	1		P 1
0xE9	MOV	A R1	1		P 1
0xEA	MOV	A R2	1		P 1
0xEB	MOV	A R3	1		P 1
0xEC	MOV	A R4	1		P 1
0xED	MOV	A R5	1		P 1
0xEE	MOV	A R6	1		P 1
0xEF	MOV	A R7	1		P 1
0xF5	MOV	data A	2		1
0xF6	MOV	@R0 A	1		1
0xF7	MOV	@R1 A	1		1
0xF8	MOV	R0 A	1		1
0xF9	MOV	R1 A	1		1
0xFA	MOV	R2 A	1		1
0xFB	MOV	R3 A	1		1
0xFC	MOV	R4 A	1		1
0xFD	MOV	R5 A	1		1
0xFE	MOV	R6 A	1		1
0xFF	MOV	R7 A	1		1
0x83	MOVC	A @A+PC	1		P 2
0x93	MOVC	A @A+DPTR	1		P 2
0xE0	MOVX	A @DPTR	1		P 2
0xE2	MOVX	A @R0	1		P 2
0xE3	MOVX	A @R1	1		P 2
0xF0	MOVX	@DPTR A	1		2
0xF2	MOVX	@R0 A	1		2
0xF3	MOVX	@R1 A	1		2
0xA4	MUL	AB	1	CY OV P	4
0x00	NOP		1		1
0x42	ORL	data A	2		1
0x43	ORL	data #imm8	3		2
0x44	ORL	A #imm8	2		P 1
0x45	ORL	A data	2		P 1
0x46	ORL	A @R0	1		P 1
0x47	ORL	A @R1	1		P 1
0x48	ORL	A R0	1		P 1
0x49	ORL	A R1	1		P 1
0x4A	ORL	A R2	1		P 1
0x4B	ORL	A R3	1		P 1
0x4C	ORL	A R4	1		P 1

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x4D	ORL	A R5	1	P	1
0x4E	ORL	A R6	1	P	1
0x4F	ORL	A R7	1	P	1
0x72	ORL	C bit	2	CY	2
0xA0	ORL	C /bit	2	CY	2
0xD0	POP	data	2		2
0xC0	PUSH	data	2		2
0x22	RET		1		2
0x32	RETI		1		2
0x23	RL	A	1		1
0x33	RLC	A	1	CY P	1
0x03	RR	A	1		1
0x13	RRC	A	1	CY P	1
0xD2	SETB	bit	2		1
0xD3	SETB	C	1	CY	1
0x80	SJMP	code8	2		2
0x94	SUBB	A #imm8	2	CY AC OV P	1
0x95	SUBB	A data	2	CY AC OV P	1
0x96	SUBB	A @R0	1	CY AC OV P	1
0x97	SUBB	A @R1	1	CY AC OV P	1
0x98	SUBB	A R0	1	CY AC OV P	1
0x99	SUBB	A R1	1	CY AC OV P	1
0x9A	SUBB	A R2	1	CY AC OV P	1
0x9B	SUBB	A R3	1	CY AC OV P	1
0x9C	SUBB	A R4	1	CY AC OV P	1
0x9D	SUBB	A R5	1	CY AC OV P	1
0x9E	SUBB	A R6	1	CY AC OV P	1
0x9F	SUBB	A R7	1	CY AC OV P	1
0xC4	SWAP	A	1		1
0xC5	XCH	A data	2	P	1
0xC6	XCH	A @R0	1	P	1
0xC7	XCH	A @R1	1	P	1
0xC8	XCH	A R0	1	P	1
0xC9	XCH	A R1	1	P	1
0xCA	XCH	A R2	1	P	1
0xCB	XCH	A R3	1	P	1
0xCC	XCH	A R4	1	P	1
0xCD	XCH	A R5	1	P	1
0xCE	XCH	A R6	1	P	1
0xCF	XCH	A R7	1	P	1
0xD6	XCHD	A @R0	1	P	1
0xD7	XCHD	A @R1	1	P	1
0x62	XRL	data A	2		1
0x63	XRL	data #imm8	3		2
0x64	XRL	A #imm8	2	P	1
0x65	XRL	A data	2	P	1
0x66	XRL	A @R0	1	P	1
0x67	XRL	A @R1	1	P	1
0x68	XRL	A R0	1	P	1
0x69	XRL	A R1	1	P	1
0x6A	XRL	A R2	1	P	1
0x6B	XRL	A R3	1	P	1
0x6C	XRL	A R4	1	P	1
0x6D	XRL	A R5	1	P	1
0x6E	XRL	A R6	1	P	1

**APPENDIX E. 8051 INSTRUCTIONS IN ALPHABETICAL
98 ORDER**

Opcode	Mnemonic	Operands	Bytes	Flags	Cycles
0x6F	XRL	A R7	1	P	1
0xA5	Invalid opcode				

Table E.1: 8051 Instructions in lexical Order

Appendix F

List of supported micro-controllers

F.0.1 Intel®

8051	http://download.intel.com/design/MCS51/MANUALS/27238302.pdf
80C51	http://download.intel.com/design/MCS51/MANUALS/27238302.pdf
8052	http://download.intel.com/design/MCS51/MANUALS/27238302.pdf

F.0.2 Atmel®

Flash (Reprogrammable)

AT89C2051	http://www.atmel.com/dyn/resources/prod_documents/doc0368.pdf
AT89C4051	http://www.atmel.com/dyn/resources/prod_documents/doc1001.pdf
AT89C51	http://www.atmel.com/dyn/resources/prod_documents/doc0265.pdf
AT89C51RC	http://www.atmel.com/dyn/resources/prod_documents/doc1920.pdf
AT89C52	http://www.atmel.com/dyn/resources/prod_documents/doc0313.pdf
AT89C55WD	http://www.atmel.com/dyn/resources/prod_documents/doc1921.pdf
AT89LV51	http://www.atmel.com/dyn/resources/prod_documents/doc0303.pdf
AT89LV52	http://www.atmel.com/dyn/resources/prod_documents/doc0375.pdf
AT89LV55	http://www.atmel.com/dyn/resources/prod_documents/doc0811.pdf

Flash ISP (Programable via ISP)

AT89S52	http://www.atmel.com/dyn/resources/prod_documents/doc1919.pdf
AT89LS51	http://www.atmel.com/dyn/resources/prod_documents/doc3053.pdf
AT89LS52	http://www.atmel.com/dyn/resources/prod_documents/doc2601.pdf
AT89S8253	http://www.atmel.com/dyn/resources/prod_documents/doc3286.pdf
AT89S2051	http://www.atmel.com/dyn/resources/prod_documents/doc3390.pdf
AT89S4051	http://www.atmel.com/dyn/resources/prod_documents/doc3390.pdf

OTP (One-Time Programmable)

T87C5101	http://www.atmel.com/dyn/resources/prod_documents/doc3c0c80904bc57.pdf
T83C5101	http://www.atmel.com/dyn/resources/prod_documents/doc3c0c80904bc57.pdf
AT80C32X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
TS87C52X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
AT87C52X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
AT80C54X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf

ROM

TS83C5102	http://www.atmel.com/dyn/resources/prod_documents/doc3c0c80904bc57.pdf
TS80C32X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
TS80C52X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
AT80C58X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
AT87C54X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
AT87C58X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
TS80C54X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
TS80C58X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
TS87C54X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf
AT80C52X2	http://www.atmel.com/dyn/resources/prod_documents/doc4184.pdf
TS87C58X2	http://www.atmel.com/dyn/resources/prod_documents/doc4431.pdf

ROMless

TS80C31X2	http://www.atmel.com/dyn/resources/prod_documents/doc4428.pdf
AT80C31X2	http://www.atmel.com/dyn/resources/prod_documents/doc4428.pdf

Appendix G

Change log

1.3.11 -> 1.4

- * Bug fixes
- * Added new Virtual HW component: LCD display controlled by HD44780
- * Added new Virtual HW component: simulated DS1620 temperature sensor
- * Added new Virtual HW component: File interface
- * Added AT89S51
- * Improved performance of Virtual HW
- * Added support for spelling checker (Hunspell)
- * Added 8051 Instruction table
- * Improved table of symbols on the right panel
- * Final draft of the handbook

1.3.10 -> 1.3.11

- * Bug fixes
- * New interface for hardware control plug-ins
- * Added new assembler directives: ELSEIF ELSEIFN ELSEIFDEF ELSEIFNDEF
- * Removed assembler directive: EXITM

1.3.9 -> 1.3.10

- * Bug fixes
- * Extended help menu
- * Extended global configuration dialog
- * Added support for multiple widget styles and GUI background colors
- * Added draft of handbook
- * Added basic support for assembler and simulator regression testing
- * Added regular support for i18n (internationalization)
- * Modified welcome dialog
- * Added support for external links in the GUI

1.3.8 -> 1.3.9

- * Bug fixes

1.3.7 -> 1.3.8

- * Bug fixes
- * Added feature "Global Font Size Factor" (see MCU 8051 IDE configuration dialog)
- * Added breakpoint validation

1.3.6 -> 1.3.7

- * Bug fixes

1.3.5 -> 1.3.6

```

    * Bug fixes

1.3.4 -> 1.3.5
    * Bug fixes

1.3.3 -> 1.3.4
    * "Modernized" GUI
    * Bug fixes

1.3.1 -> 1.3.3
    * Bug fixes

1.3.1 -> 1.3.2
    * Bug fixes

1.3 -> 1.3.1
    * Dependency on TclX is now only optional
    * Important change !: Native assembler now expands macro instructions before doing conditional
      assembly and before defining constants and variables ! Control sequence $NOMACROSFIRST can
      be used to change this behavior to the state of previous versions.
    * Added support for AS31 assembler
    * Added files notepad
    * Improved instruction help panel
    * Native assembler was extended to support these directives: "IFN IFDEF IFNDEF BYTE FLAG REPT
      TIMES" and these control sequences: "$NOXR $NOXREF $XR $XREF $NOSB $SB $RESTORE $RS $SA
      $SAVE $PHILIPS $NOPI $PI $NOTABS $NOMOD51 $NOBUILTIN $NOMO $MO $MOD51 $NOMACRO $NOMR $LI
      $NOLI $GENONLY $GO $NOGEN $NOGE $GEN $GE $ $EJ $NODEB $NODEBUG $DB $DEBUG $CONDONLY $NOCOND
      $COND $TT $PW $PL $MR $MACRO $INC $WARNING $ERROR $DA $NOMACROSFIRST"
    * Added stack monitor
    * Various bug fixes

1.2 -> 1.3
    * New dependency: TclX (tested with v8.4)
    * Added RS232/UART debugger
    * A few changes in assembler
    * Bug fixes (Thanks to Miroslav Hradilek for many useful bug reports)

1.1.1 -> 1.2
    * Bug fixes
    * Added tab bar

1.1 -> 1.1.1
    * Added "Special calculator"
    * Added "Base converter"
    * Many tiny improvements

1.0.9 -> 1.1
    * Added support for new MCUs from Intel®: 8031, 8751, 8032, 8752, 80C31, 87C51, 80C52,
      87C52, 80C32, 80C54, 87C54, 80C58, 87C58
    * Added support for simulating virtual hardware
    * Improved simulator (Implemented UART (experimental support), improved support for timers, etc.)
    * Improved register watches
    * Improved editor (improved autocompletion and many other things)
    * Improved panel "Instruction details"
    * Improved 8-segment editor
    * Bug fixes in assembler, disassembler and simulator engine
    * Some other bug fixes
    * Added utility "Scribble notepad"
    * Improved graph panel

```

-
- 1.0.7 -> 1.0.9
- * Added support for C language
 - * Added map of bit addressable area
- 1.0.6 -> 1.0.7
- * Added Stopwatch
 - * Improved code editor
 - * Some bug fixes
- 1.0.5 -> 1.0.6
- * Fixed critical bug in Assembler v1.0.5 (related to peephole code optimization)
 - * Added 8 segment LED display editor
 - * Added ASCII chart
 - * Added Assembly symbol table viewer
- 1.0 -> 1.0.5
- * Added support for external assemblers ("ASEM-51" and "ASL")
 - * Added support for external editors ("emacs", "gvim", "kwrite" and "gedit")
 - * Added support for embedded editors ("emacs", "vim", "nano", "dav" and "le")
 - * Added embedded terminal emulator (rxvt-unicode)
 - * Added function "File statistics"
 - * Improved assembler
 - * Added syntax highlight for code listing (*.lst)
 - * Added search bars for "Messages" and "Todo"
 - * Removed dependency on "tcl-thread" and "tclxml"
 - * Added dependency on "TkImg" and "tdom"
 - * Improved hex editor
 - * Improved simulator (especially simulation across multiple files)
 - * Added panel "Find in files"
 - * Modified GUI
 - * New error handling dialog
 - * Some bug fixes (especially critical bug in disassembler and a few bugs in assembler)
 - * All images are now in PNG (Portable Network Graphics) (Requires TkImg)
 - * Some more improvements
- 0.9.5 -> 1.0
- * MANY BUG FIXES ! (including critical)
 - * Added support for some new MCUs (
AT89S52, AT89LS51, AT89LS52, AT89S8253, AT89S2051, AT89S4051,
T87C5101, T83C5101, T83C5102, TS80C32X2, TS80C52X2, TS87C52X2,
AT80C32X2, AT80C52X2, AT87C52X2, AT80C54X2, AT80C58X2, AT87C54X2,
AT87C58X2, TS80C54X2, TS80C58X2, TS87C54X2, TS87C58X2, TS80C31X2,
AT80C31X2
)
 - * Added support for peephole optimization
 - * Faster project opening
 - * Added interrupt monitor
 - * Added subprograms monitor
 - * Added SFR map
 - * Added SFR watches
 - * Extended command line interface
 - * Compiler now checks for valid memory addressing (new CLI options --iram-size, --eram-size, --xram-size, --code-size)
 - * Added program hibernation capability
 - * Added editor commands hibernate, resume, switch-mcu, set-xcode and set-xdata
 - * Added desktop file and application icon
 - * Some more improvements

0.9.1 -> 0.9.5

- * Implemented support for 80C51, 8052, AT89C2051, AT89C4051, AT89C51, AT89C51RC, AT89C52, AT89C55WD, AT89LV51, AT89LV52 and AT89LV55
- * Simulator can now step back
- * Added popup-based completion for editor
- * Added tool tips for bits in simulator control panel
- * Added simulator configuration dialog
- * Added auto save function
- * Manual page
- * Added support for multi-view (editor can be now splitted vertically or horizontally)
- * Many bug fixes (in compiler, editor, file selection dialog, syntax highlight, simulator, etc.)
- * Some minor improvements (graph, disassembler, etc.)
- * Thread extension is no longer required to run this program (but custom commands will won't work without it)

0.9.0 -> 0.9.1

- * New hexadecimal editor
- * New file selection dialog
- * Added file system browser tab on left panel
- * Added tips on start-up
- * Added editor command line
- * Improved editor configuration dialog
- * A few bug fixes
- * Removed dependency on IWidgets and Tix
- * Some minor improvements

0.8.7 -> 0.9.0

- * Implemented graph
- * Many bug fixes (GUI, compiler, memory leaks)
- * Editable shortcuts
- * Bookmarks for opened and project files
- * Search panels in left and right panel
- * Modified GUI (checkboxes, radio buttons ...)
- * Support for various encodings and EOLs
- * Added "Tools" -> "Change letter case", "Normalize HEX" and "SIM -> BIN"
- * Added editor functions "Lowercase", "Uppercase" and "Capitalize"
- * Added help windows for opened and project files and opened projects
- * Added pop-up menus for entry and text widgets (globally)
- * Fixed problem with fonts (bad sizes)
- * Implemented support for line wrapping (experimental)
- * Added new command line options (see 'mcu8051ide --help')
- * More status tips and tool tips
- * Added welcome dialog
- * Added demonstration project
- * Cleaner, faster and safer compiler
- * Some more minor improvements

0.8.5 -> 0.8.7

- * Implemented code validation
- * Added tab "Instruction details" (on the right panel)
- * Added Clean Up dialog
- * Added Right Panel configuration dialog
- * Added Toolbar configuration dialog
- * Added support for custom commands
- * Fixed some bugs (in GUI)
- * Fixed many memory leaks
- * Cleaner code

0.8.4 -> 0.8.5

- * Fixed many bugs in GUI
- * Improved editor
- * Extended calculator
- * Redesigned editor configuration dialog
- * Added functions "Tools -> Reformat code" and "Tools -> Sim2Hex"
- * Extended CLI (--reset-user-settings, --config-file, --compile, --hex2bin ...)

0.8.1 -> 0.8.4

- * Fixed many bugs ... (including critical)
- * Added compiler configuration dialog
- * Added calculator timers preset
- * Added dialog about
- * Added support for exporting highlighted source code to LaTeX source
- * Added many ToolTips
- * Added StatusBar tips
- * Added splash screen
- * Added support for command line options
- * All images are now *.XPM (X PixMap) (require Tix package)
- * Changed installation procedure

0.8.0 -> 0.8.1

- * Fixed some bugs in compiler (not critical)
- * Fixed bug in to do list (saving text as SGML)
- * Fixed bug in project management
- * Added pop-up menu to to do list

List of Tables

1	Software requirements	6
3.1	Available commands	17
3.2	Data register watches: Register address	24
3.3	Symbol colors and icons in default settings	25
3.4	List of pseudo-variables	32
4.1	Radix specifiers	39
4.2	Expression operators	39
4.3	Code addresses	47
4.4	Plain numbers, these symbols are always defined!	47
4.5	Predefined SFR bit addresses	47
4.6	Predefined SFR addresses	48
4.7	Segment types	49
4.8	Directives directly related to macros	52
4.9	Special instruction operands	56
4.10	Instruction mnemonics	56
4.11	Directives	56
4.12	Expression operators	56
4.13	Assembler controls	56
4.14	Control sequences affecting code listing	59
D.1	8051 Instructions in numerical Order	90
E.1	8051 Instructions in lexical Order	96

List of Figures

1.1	i8051 micro-architecture	12
2.1	MCU 8051 IDE with the demonstration project opened within it	13
2.2	Project creation dialog	14
3.1	Syntax validation configuration button	15
3.2	Spell checker configuration button	16
3.3	Syntax highlight, syntax validation and the pop-up based auto-completion all in action	16
3.4	Main panel of the simulator	18
3.5	Highlighted SFR register	18
3.6	Tool tip help for a special function bit	18
3.7	Representation of a register value in various numeric bases	18
3.8	GPIO Graph	19
3.9	Messages panel	19
3.10	Personal notes	20
3.11	Calculator	20
3.12	Embedded rxvt-unicode terminal emulator, with the Midnight Commander running in it	21
3.13	SFR watches	22
3.14	Instruction details	23
3.15	Data register watches	23
3.16	Subprograms call monitor	24
3.17	Map of the bit addressable area	25
3.18	Stack monitor	25
3.19	Symbol viewer	26
3.20	ASCII chart	26
3.21	8051 Instruction Table	26
3.22	8-segment editor	27
3.23	Base convertor	28
3.24	UART/RS-232 debugger	28
3.25	MCU code memory editor	29
3.26	Interrupt monitor	30

3.27	Change letter case dialog	32
3.28	Custom commands configuration dialog	33
3.29	Editor configuration dialog	33
3.30	Main toolbar	35
3.31	Global configuration dialog	35
6.1	DS1620 simulator and its log window	65
6.2	PALE I/O interface	65
6.3	LED Panel	66
6.4	LED Display	66
6.5	M LED Display	66
6.6	LED Matrix	67
6.7	Matrix Keypad	67
6.8	Simple Keypad	68
6.9	Simulated LCD display	68
6.10	HD44780 Log	68
6.11	CGRAM	68
6.12	DDRAM	68
6.13	View on CGROM	68
7.1	An example of HW control plug-in	69
7.2	A basic example of a plug-in	72
B.1	Assembler regression test run in terminal emulator	81
C.1	Official web page of the MCU 8051 IDE project	83