DPIC / EPIC

User's Guide

ASIX®

ASIX s.r.o.
Grafická 37
150 00 Prague
Czech Republic

E-mail:	asix@asix.cz
WWW:	http://www.asix.cz/
Tel./fax:	+420 - 2 - 573 123 78

Table of Contents

1. General Information

Congratulations on selecting the EPIC16A emulator. It is designed to offer powerful capabilities while developing and debugging applications of the PIC microcontrollers (Microchip Technology Inc.) in most general use. Integrated development environment DPIC enables you quickly and easily develop applications for the PIC series 12C5xx, 16C5x and 16Cxx . All desirable tools are included in it: editor of source text, compiler (assembler), disassembler and interface (operating program) to emulator EPIC16A made by ASIX s.r.o.

The unique construction of the emulator does not use a common emulation chip but a latest programmable logic devices (ASIC). Besides of all standard tasks, it allows to perform many features, which are not usual or feasible with emulators of even much higher price category:

- You can watch and modify all registers (file registers as well as special internal registers, e.g. PC, STACK, PORT, TRIS, LATCH, OPTION, PRESCALER, WREG, ...) at full speed with the exception of STACK modification in Run mode.
- Extensive possibilities of break based on many various conditions, not only usual breaks in code memory on an arbitrary location, but even breaks in data memory with the following options: access, write to register and write the value with respect to mask. Further, special breaks are available: timer overflow, trace buffer overflow, watchdog overflow, stack overflow/underflow, break on external probe of emulator (rising or falling edge can be selected).
- Remarkably flexible setting of clock frequency of the emulated processor ranging 20 kHz through 20 MHz by 40 Hz step with the help of internal frequency synthesizer.
- Internal circuitry of emulator is separated from I/O pins, which allows to operate with the power supply in the whole range of PIC microcontrollers (from 3V to 6V). That is why the user is not forced to use just 5V supply voltage only.
- Emulator has an additional stabilized 5V power source with 100 mA current limitation to supply user application (peripherals of the PIC being emulated).
- Off-line mode: after exiting the operating program the emulator is capable of continuous stand alone emulation (PC host computer is not necessary any more).
- A well-arranged mode indication with the help of LED varied in color for watching the operation of emulator (even in off-line mode): orange LED - power on / emulation chip successfully configured, red LED - Halted, yellow LED - Break, green LED - Run / Step, blue LED - Sleep.

- If the emulator is in operation while the operating software is invoked, the environment is set according to the situation which it was just before exiting. Information needed for it are stored in the hardware of the emulator, not only in the configuration file of PC host.
- Optional enable/disable of processor reset caused by setting -MCLR pin low or by power supply voltage drop (when it goes below 2V). Even short reset is detected.
- Flexible tracing - off, on or tracing only selected portions of program (e.g. you can disable tracing of waiting loops and trace only the main program). Trace buffer capacity is 32 KB (8192 instructions).
- Eight probes for optional tracing of external signals synchronously to reading of I/O pins of emulated processor.
- The CLKOUT output has a proper frequency and phase according to selected oscillator type of emulated processor.
- Correct emulation of overflow of watchdog, TMR0 (RTCC) timers and write to internal EEPROM (PIC16C8x) even in Halted/Step mode
- Flexibility of emulator circuitry allows software upgrades (e.g. via Internet) of the emulator hardware and individual modifications with respect to individual user's needs.

The integrated environment contains a demo version of simulation program for PICs of all families supported by emulator as well. A full version (without restrictions) is available independently on the emulator.

Basic characteristics of the assembler:

- It is a line-oriented compiler. Thus, only one instruction is allowed in a line.
- It is allowed to work with the symbols on higher level of abstraction then other similar products are used to do. It allows to define and use more complicated data types (byte, bit, constant, byte array, bit array, table).
- Conditional compilation, Macroinstructions, 8 nesting levels
- Format of output binary code: INHX-16 and INHX8M
- Compatibility with the compiler by Microchip

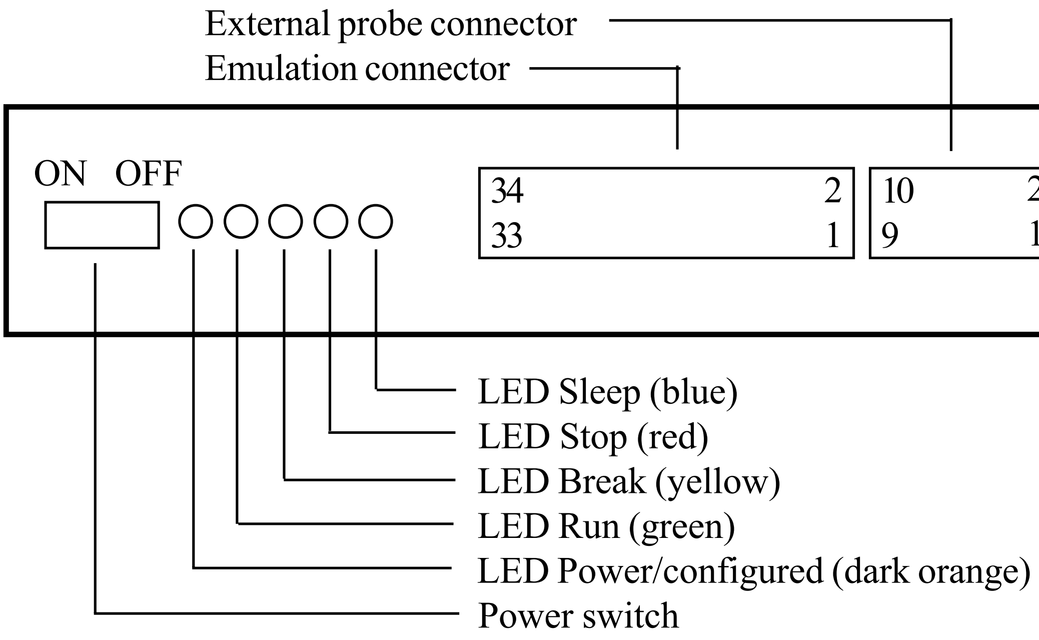
It is recommended to run DPIC on PC/AT or compatible computer with 486 processor, 16 MB RAM and about 4 MB free space on hard disk. The minimal PC configurations which allows to run DPIC means 386SX processor and 2 MB RAM.

2. EPIC16A Emulator - HW Description

Characteristic:

Height: 42 mm (1.65") x Width: 168 mm (6.6") x Depth: 140 mm (5.5")
Ambient temperature while operating: +15 - +30 °C (+59 - +86 °F)
Storage temerature range: -40 - +70 °C (-40 - +158 °F)
Operating relative humidity: 90 % max

2.1 Front Panel



LED Power/configured (dark orange). Blinking / continuous lighting means not configured / configured emulation chip.

2.1.1 Emulation Connector

1 - +5 V / 100 mA	10 - OSC1/CLKIN	19 - PORTA1	28 - PORTC0
2 - +5 V / 100 mA	11 - not connected	20 - PORTC4	29 - PORTB2
3 - GND	12 - OSC2/CLKOUT	21 - PORTA2	30 - PORTB7
4 - GND	13 - GND	22 - PORTC3	31 - PORTB3
5 - not connected	14 - PORTC7	23 - PORTA3	32 - PORTB6
6 - not connected	15 - not connected	24 - PORTC2	33 - PORTB4
7 - TOCKI (RTCC)	16 - PORTC6	25 - PORTB0	34 - PORTB5
8 - -MCLR	17 - PORTA0	26 - PORTC	
9 - VDD	18 - PORTC5	27 - PORTB1	

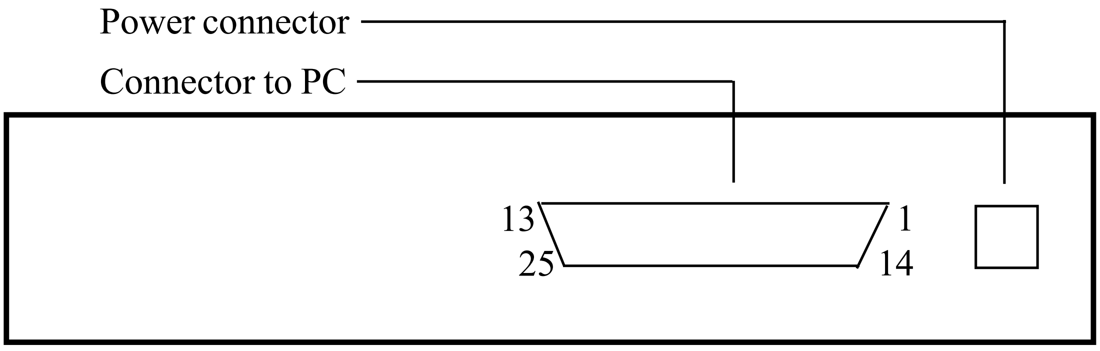
2.1.2 Probe Connector

1 - PROBE 6	6 - PROBE 3
2 - PROBE 7	7 - PROBE 0
3 - PROBE 4	8 - PROBE 1
4 - PROBE 5	9 - GND
5 - PROBE 2	10 - GND

2.1.3 Color Identification of Probes

PROBE 0 - white	PROBE 4 - green
PROBE 1 - grey	PROBE 5 - yellow
PROBE 2 - violet	PROBE 6 - orange
PROBE 3 - blue	PROBE 7 - red
GND - black	

2.2 Rear Panel



2.2.1 PC to EPIC16A Connection

Emulator provides communication via Canon 25 female connector to PC host parallel port. A 6-foot male-to-male data cable with 25-pin connectors is supplied with the emulator. All lines are wired straight through. Maximal length of data cable is 2m (6.5 foot).

1 - STROBE	6 - DATA 4	11 - BUSY	16 - INIT	21 - GND
2 - DATA 0	7 - DATA 5	12 - PE	17 - SLCTIN	22 - GND
3 - DATA 1	8 - DATA 6	13 - SLCT	18 - GND	23 - GND
4 - DATA 2	9 - DATA 7	14 - AUTOLF	19 - GND	24 - GND
5 - DATA 3	10 - ACK	15 - ERR	20 - GND	25 - GND

Caution: The cable must not be replaced by one with the connection other then specified above, e.g. by a cross-wired Laplink cable for LPT!

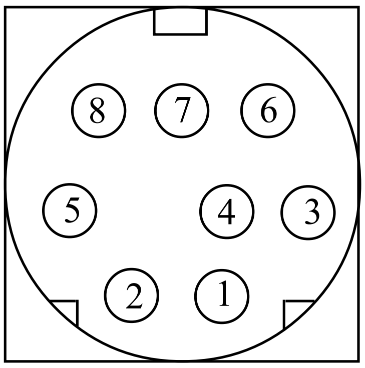
2.2.2 Power Supply Connector

Power supply connector type: AMP Shielded Miniature Circular DIN Plug 8 pin P/N 749179-1. Power supply 5 V +- 5 % / 1 A, 12 V +- 5 % / 500 mA. Computer Products SCL25-7618 power supply is recommended.

1 - +5 V	5 - GND
2 - +5 V	6 - GND
3 - +5 V	7 - GND
4 - +12 V	8 - GND

Caution: Pay attention to proper power connection, otherwise the emulator can be seriously damaged!

Rear view to the power supply connector of the emulator:

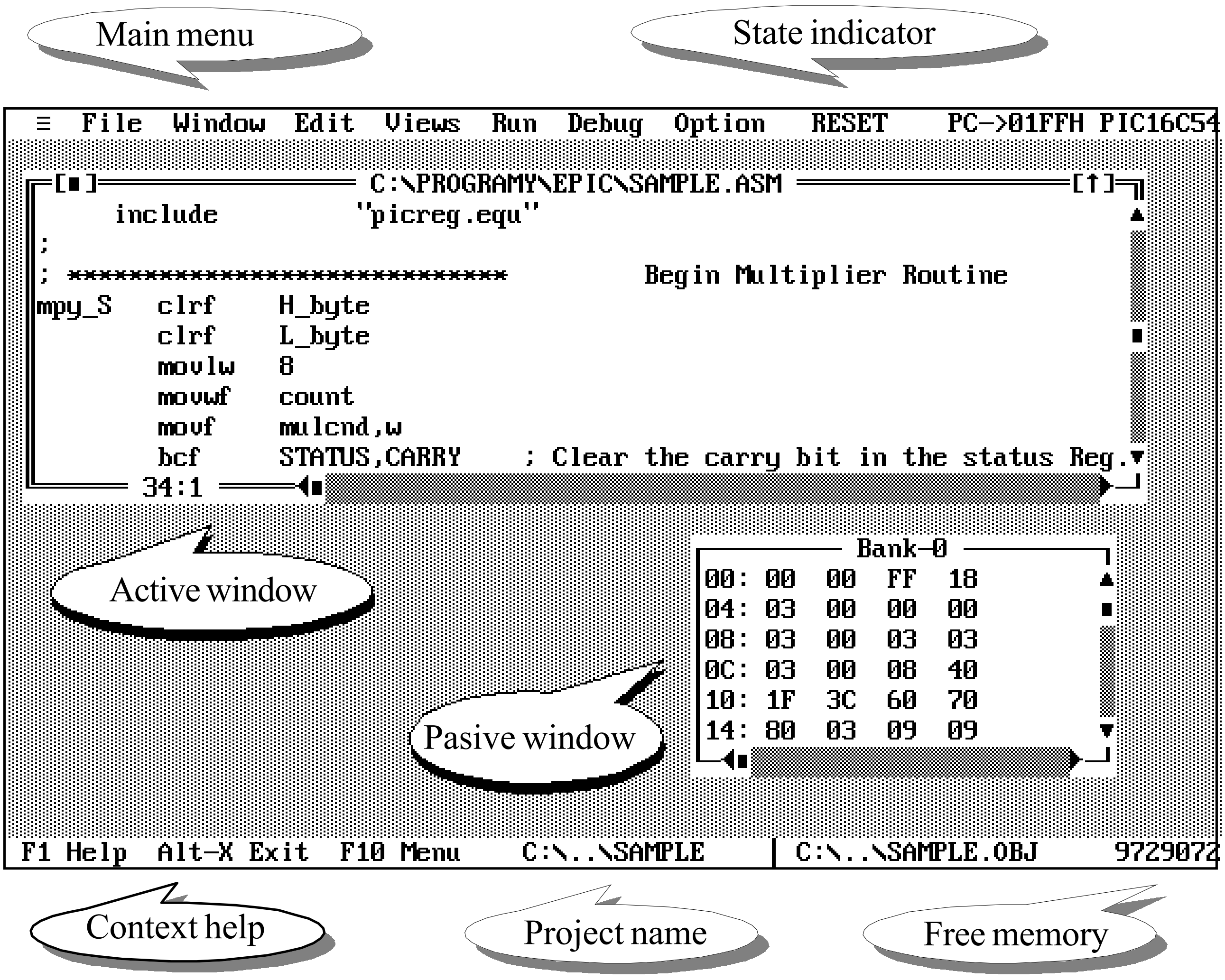


3. Integrated Development Environment

Integrated development environment (IDE) includes several basic modules: text editor, tools for memory editing of the emulated device, system for events entry and editing, project manager, compiler and drivers for all types of supported PICs. All functions of the environment are available via the basic menu. The most frequently used commands have Borland-like hot keys assigned to them. Mouse control is fully supported.

3.1 Screen Layout

Basic screen of the environment displays following information: On the top there is a main menu bar where the current state of the emulator is also displayed. The upper bar consists of the active help for three most important hot keys, name of the program loaded into emulation memory and free PC host memory. The rest of the screen area is intended for placing windows for program tools and elements.


Main menu
State indicator
≡ File Window Edit Views Run Debug Option RESET PC->01FFH PIC16C54
Active window
Pasive window
F1 Help Alt-X Exit F10 Menu
Context help
Project name
Free memory

```
[■]═══════════ C:\PROGRAMY\EPIC\SAMPLE.ASM ═══════════[↑]
    include        "picreg.equ"
;
; ******************************        Begin Multiplier Routine
mpy_S   clrf    H_byte
        clrf    L_byte
        movlw   8
        movwf   count
        movf    mulcnd,w
        bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
════ 34:1 ════

───── Bank-0 ─────
00: 00  00  FF  18
04: 03  00  00  00
08: 03  00  03  03
0C: 03  00  08  40
10: 1F  3C  60  70
14: 80  03  09  09

F1 Help  Alt-X Exit  F10 Menu    C:\..\SAMPLE    | C:\..\SAMPLE.OBJ    9729072
```

3.1.1 Program Elements of an Integrated Environment

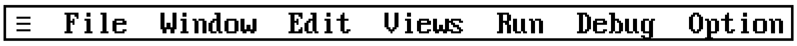
Program objects of integrated environment mean basic types of display information about the state of the system:

- ***Menu*** - object for invoking selected commands, events and so on
- ***Watch view*** - object for enclosing a list on the screen
- ***Dialog box*** - window which allows setting of values, system settings and so on. Unlike list window, only one dialog box can be active (able to accept commands) at a time. Thus, all other objects on the screen are inactive until the dialog box is closed.
- ***Scroll bar*** - list window is usually provided with horizontal and vertical bars. They are mouse-sensitive and allow scrolling of the text in the window. They are used especially if the number of columns or lines exceeds the area reserved to the window.
- ***Button*** - (group of) buttons are used for setting and selections like Yes/No, and for current command confirmation or cancellation. They should be without check (an event is invoked just after a single press). Mouse or pressing the key assigned to the highlighted letter can be used). Buttons with check can allow multiple or exclusive choices. Multiple choice buttons can be set on or off independently on each other. Exclusive buttons can only be in a group, and only one of them can be set active in the group at a time.
- ***Line editor*** - an object which allows to enter character strings or constant numbers. Editors are provided with history. It is a list containing strings of previous entries of the command. In many cases editors have even a list of available names, e.g. names of defined variables. These lists are usually updated after a successful compilation of source text.
- ***List box*** - an object for a group of commands or selection options. You can use a mouse or the Enter key to make a choice.

3.1.2 Menu

You can use a keyboard or a mouse to access all menu items. Pressing the *F10* key makes the menu bar active. Use the arrow keys to select desirable menu item or press the key corresponding to highlighted letter of the item. Besides of this, main menu items and some commands have hot keys assigned to them as a shortcut for the selection.

F10 = Menu



Alt-Space Alt-F Alt-W Alt-E Alt-V Alt-R Alt-D Alt-O

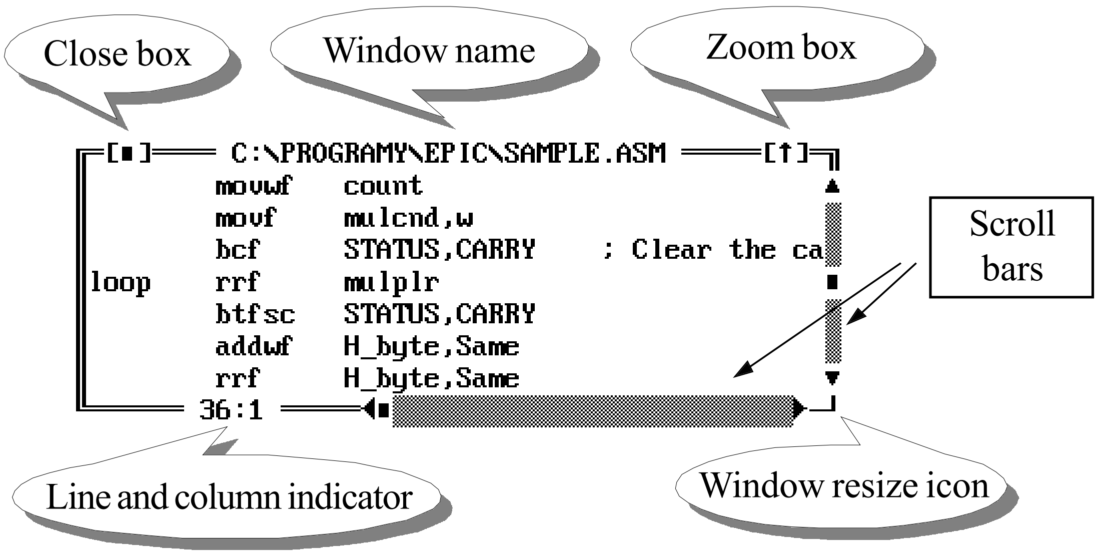
All main menu bar hot keys are handled in the uniform way using a highlighted letter and the *Alt* key (e.g. *Alt-F* for File, *Alt-D* for Debug). Selection of a menu command is done by selecting the menu item with the mouse cursor and clicking the left mouse button.

3.1.3 Watch Window

Watch windows of lists, editors etc. can be active (selected) or inactive. The active window is the one you are currently working in. This has a double-lined border around it, all inactive windows have single-lined ones.

The active window accepts commands from the keyboard and menu commands. Inactive windows ignore commands. Only one window can be active at a time.

There are several ways to make a window active: *"Window→Next" (F6)*, *"Window→Previous" (Shift-F6)* commands or clicking the mouse anywhere in the window.



```
[■]═══ C:\PROGRAMY\EPIC\SAMPLE.ASM ═══[↑]
        movwf   count
        movf    mulcnd,w
        bcf     STATUS,CARRY    ; Clear the ca
loop    rrf     mulplr
        btfsc   STATUS,CARRY
        addwf   H_byte,Same
        rrf     H_byte,Same
═══ 36:1 ═══
```

Windows have the following attributes:

- ***Window name*** - the name is centered at the top border line. In case of a text editor it means name of the file to be edited, otherwise it means the type of the list in the window.
- ***Scroll bar*** - allows to scroll the text horizontally or vertically in the window.
- ***Close box*** - in the upper left corner of the border. Click it to close and remove the window from the screen. Another way for this is the *"Window→Close"* command *(Alt-F3)* from the main menu.
- ***Window resize icon*** - in the lower right corner of the border. Drag it by the mouse to make the window larger or smaller. Another way for this is the *"Window→Resize"* command *(Ctrl-F5)* from the main menu.

- ***Window zoom box*** - in the upper right corner of the border. Click either to enlarge to maximal size or shrink back the window. Another way for this is the *"Window→Zoom"* command *(F5)* from the main menu.
- ***Line and column indicator*** - in the lower left corner of the border. In case of text editor it displays current cursor position. The format is [line:column].

3.1.3.1 Moving a Window Around the Screen

Windows can be moved, enlarged or reduced on the screen.

a) ***Moving a window by the mouse*** - move the mouse pointer to border of the active window except of the Window resize icon, Close box and Zoom box. Drag the mouse cursor where you want to have the window while pressing the left mouse button, and then release it.

b) ***Moving a window from keyboard*** - use the *"Window→Move"* command and arrow keys to move the window where you want to have it, then press *Enter*.

c) ***Resize window by mouse*** - move the mouse pointer to Window resize icon, press the left mouse button and drag then. The lower left corner follows the pointer while the position of the rest of the window leaves still unchanged. Fix the desired window size by releasing the mouse button.

d) ***Resize window from keyboard*** - the *"Window→Resize"* command switches the window to the size edit mode. Set the window size by pressing the *Shift* key and arrow keys. Fix the desired window size by the *Enter* key.

e) ***Zoom a window by the mouse*** - Enlarge to maximal size or shrink back the window by clicking the Zoom box.

f) ***Zoom a window from keyboard*** - Enlarge to maximal size or shrink back the window by pressing *F5* hot key or by *"Window→Zoom"* command from the main menu.

3.1.3.2 Moving Through a Text in a Window

a) ***Moving through a text by the mouse*** - you can scroll the text in the window using scroll bars. Click the arrow at either end of the horizontal or vertical bar to scroll one line or column at a time. Similarly, you can click the area between either arrow and the scroll box to scroll a page at a time. Finally, you can drag the scroll box to any spot on the bar to quickly move to a spot in the window relative to the position of the scroll box.

b) ***Moving through a text from keyboard*** - move the cursor to a window edge where you want to scroll the text. To do it, press the appropriate arrow key.

3.1.4 Dialog Box

A dialog box is a convenient way to view and set multiple options of integrated environment. Only one window of this type may be open on the screen at a time. Activity of other objects on the screen is suspended until the dialog box is closed. Onscreen controls (especially buttons and input boxes) can be inside of a dialog box. Only one of these objects is active at a time. If you want to switch to another one, press the *Tab* key and repeat it until the desired object is set active, or click to the desired object. Finally, you can use a hot key assigned to the object, which is the *Alt* key and the key corresponding to highlighted letter of the object. See chapters Buttons, Line editors, List boxes, History list and Offer box to learn about editing and setting of object parameters.

Dialog box has a mouse sensitive close icon to leave the setting unchanged. Buttons without check are intended for confirmation or cancelling changes just made in dialog box and closing the window then. One of the buttons of the dialog box is always active, which enables to invoke it from any object of the dialog box by pressing *Enter*.

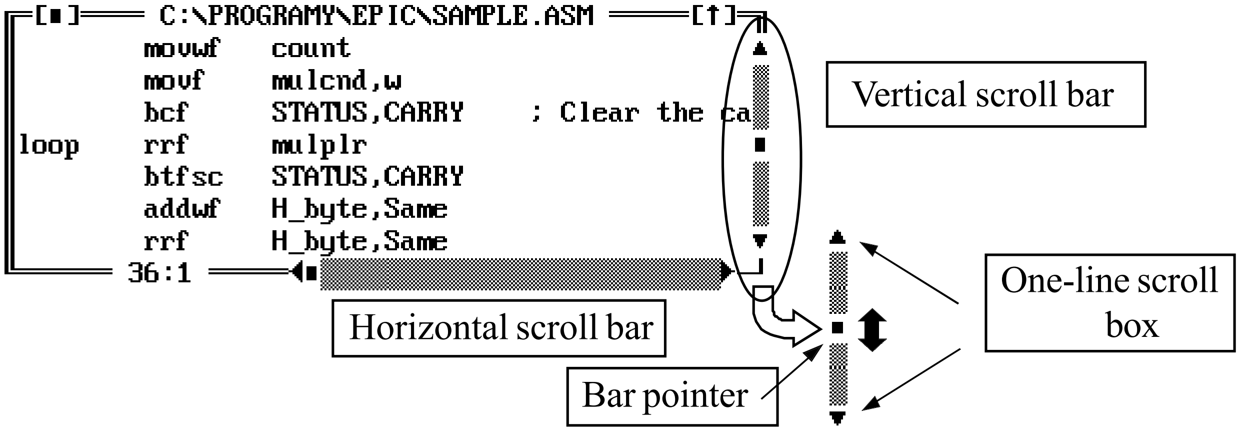
3.1.5 Scroll Bar

Scroll bar is a control object to support especially Watch windows. This object has three mouse sensitive boxes:

a) ***One-line scroll box*** - "press" one of these icons (depending on desired direction) at the end of scroll bar to scroll one line

b) ***Bar pointer*** - scroll bar pointer can be moved between one-line scroll boxes. Move the mouse pointer to the scroll bar pointer, press the left mouse button and drag the mouse. Scroll bar pointer follows the mouse pointer. Release the mouse button to perform the appropriate scrolling.

c) ***Area on the scroll bar*** - click the mouse at this area between the bar pointer and one-line scroll boxes to scroll a page forward/backward.



```
[■]═══ C:\PROGRAMY\EPIC\SAMPLE.ASM ═══[↑]
        movwf   count
        movf    mulcnd,w
        bcf     STATUS,CARRY    ; Clear the ca
loop    rrf     mulplr
        btfsc   STATUS,CARRY
        addwf   H_byte,Same
        rrf     H_byte,Same
═══ 36:1 ═══
```

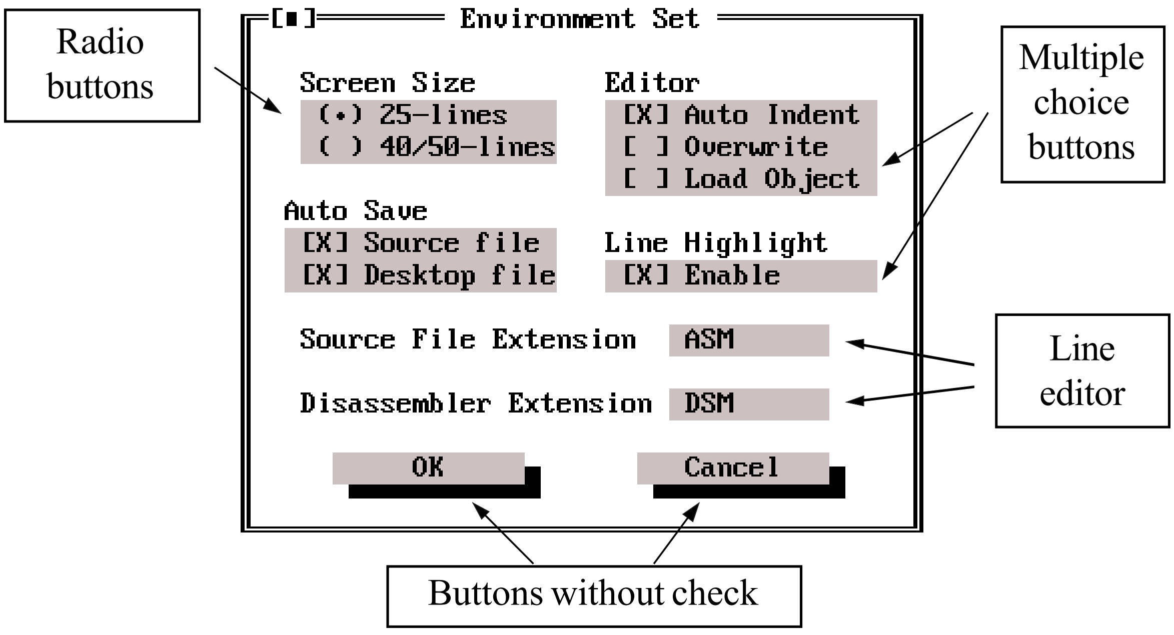
3.1.6 Buttons

There are three types of buttons:

a) ***Button without check*** - this is an object for confirmation or cancelling selections. It can be handled by left mouse button or from keyboard by pressing the key corresponding the highlighted letter of the command name.

b) ***Multiple choice buttons*** - these buttons are usually grouped by sets into a single object. They are named by names with a highlighted letters which correspond to hot keys (*Alt* and the appropriate keys). Use arrow keys to move around them. To set the selected button on/off hit the spacebar or click a left mouse button with mouse pointer on the icon of the button ([ ] or [X]).

c) ***Exclusive (radio) buttons*** - these buttons are also grouped, but unlike of multiple choice buttons only one of them can be set active in the group at a time. Handling radio buttons is similar to handling multiple choice ones.



```
[■]══════ Environment Set ══════
 Screen Size          Editor
  (•) 25-lines         [X] Auto Indent
  ( ) 40/50-lines      [ ] Overwrite
                       [ ] Load Object
 Auto Save
  [X] Source file     Line Highlight
  [X] Desktop file     [X] Enable

 Source File Extension   ASM

 Disassembler Extension  DSM

      OK              Cancel
```

3.1.7 Line Editors

Line editors are intended for numeric or text entry. They usually have a history object and a list of available symbols.

You can edit the text by standard editing keys:

- ***arrow keys*** - cursor moving
- ***Ins*** - Insert/Overwrite switching
- ***Del*** - character deleting
- ***Home*** - moves the cursor to the beginning of the text
- ***End*** - moves the cursor to the end of the text

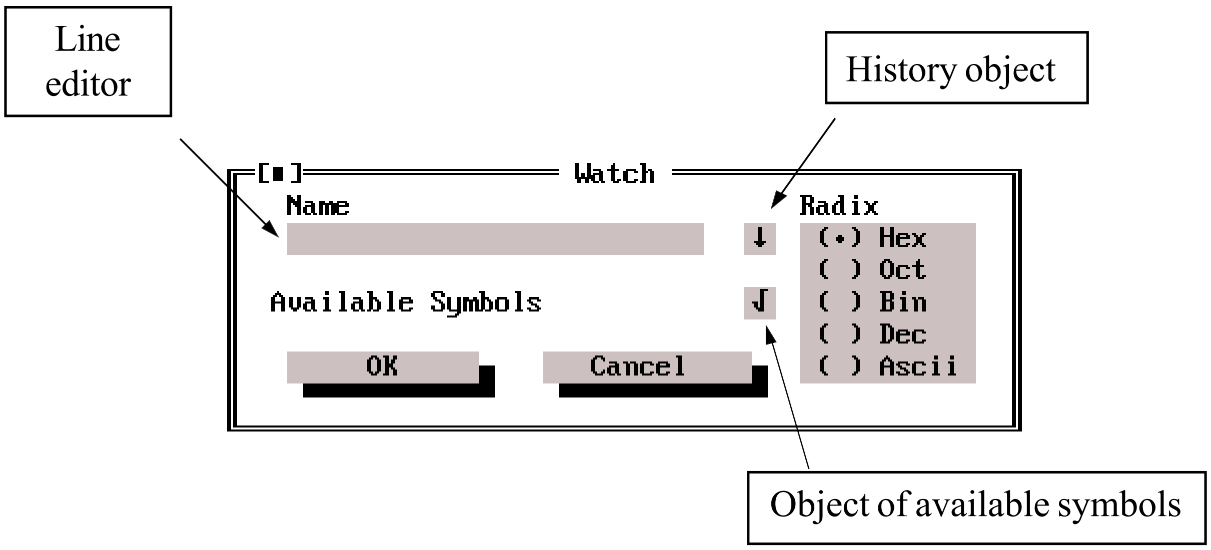
To invoke subsidiary program objects the following keys can be used:

- ***Down arrow*** - to invoke the history
- ***Ctrl-Enter*** - to invoke the list of available symbols

Subsidiary program elements are lists with the same way of handling as it was described in Chapters 3.1.8. and 3.1.9.

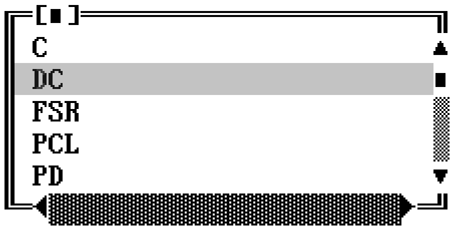
A history object contains strings of previous entries. You can easily and quickly reinvoke text strings.

To make the entry easier, a list of available symbols contains all known names and symbols defined after successful compilation.



3.1.8 List Boxes

List boxes are objects containing list of names and symbols and handle them like a database. Select an item by the cursor and confirm the selection by *Enter*, or by mouse pointer and left mouse button. If there is more items than it is possible to display in the window, the window has an additional scroll bar to browse all items of the list.



3.1.9 Subsidiary Program Elements

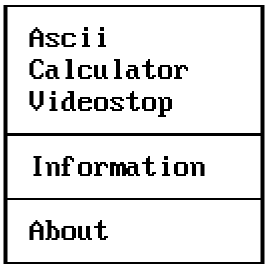
These are intended for user-friendly entry of names, symbols and mathematical expressions. The integrated environment offers the following subsidiary elements:

- ***History*** [↓] - this object is always assigned to a line editor. On a dialog window area there is an icon dedicated to it. The icon can be used only if the appropriate editor is invoked. After pushing the down arrow key, the history icon displays the list of previously entered text strings. After pressing *Enter* the selected string is imported to the editor and the history window is closed. To close the window without importing any string, press the *Esc* key.
- ***Available symbols*** [√] - this object is a list similar to the history list. The managing and consequences are also the same. Unlike the history list this list is filled after successful compilation. Available symbols mean names which can be evaluated in the expression. After pressing *Ctrl-Enter* the list of available symbols is displayed.

3.2 Main Menu

Main menu of the integrated environment is represented by the menu bar on the top of the screen. The main menu consists of ≡, File, Window, Edit, Views, Run, Debug and Options.

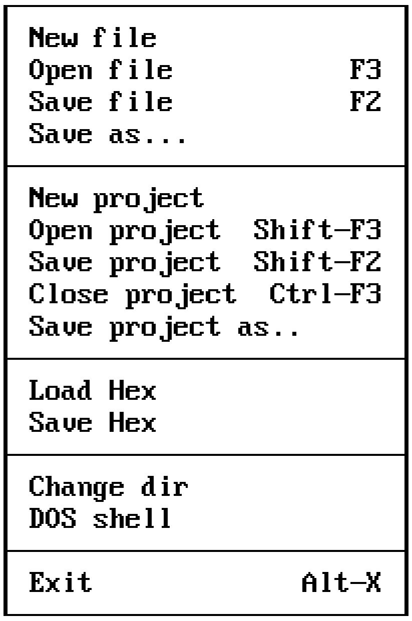
3.2.1 Menu ≡



The command contains supporting tools and information to help developing of application:

- ***Ascii*** - displays an extended table of ASCII characters and their numeric values.
- ***Calculator*** - a calculator with fixed decimal point, basic arithmetic operations included numeration system conversions.
- ***Videostop*** - a simple game to recover from total exhausting
- ***Information*** - displaying of basic information concerning version of the integrated environment, compiler and emulator.
- ***About*** - displaying names and information about authors

3.2.2 File



The File command offers commands for managing files, projects, directories and so on. These commands can be separated into groups as follows:

The File group:

- ***New File*** - opens a new file for editing
- ***Open File*** *(F3)* - opens an existing file or create a new source text file
- ***Save File*** *(F2)* - saves any object on the screen (text file or contents of any program tool to edit memory of emulated processor).
- ***Save as*** - to save source text under a different name.

The Project group:

This group is intended for project managing. The project manager allows to work simultaneously with more projects. It lets store settings of the current project if the work is interrupted. After reopening this project the environment is set according to the project definition file.

- ***New Project*** - allows to open a new project. The environment is set by default in this case.
- ***Open Project*** *(Shift-F3)* - opening of the existing file
- ***Save Project*** *(Shift-F2)* - saves the project setting to the project definition file
- ***Close Project*** *(Ctrl-F3)* - stops working on current project
- ***Save Project as*** - saves the project setting under a different name

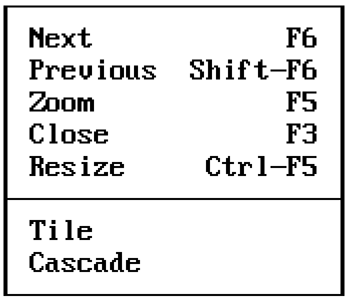
The Input/Output group:

- ***Load Hex*** - loads the hex file into emulation memory. The input format is Intel HEX-16.
- ***Save Hex*** - writes the contents of the emulation memory to the file. The output format is Intel HEX-16.

Other commands:

- ***Change Dir*** - change of current directory
- ***DOS Shell*** - invoking the DOS command line. Because of the PC conventional memory is highly utilized by the integrated environment, it is usually not possible to run another program from DOS Shell.
- ***Exit*** *(Alt-X)* - stops the DPIC program and exits to DOS.

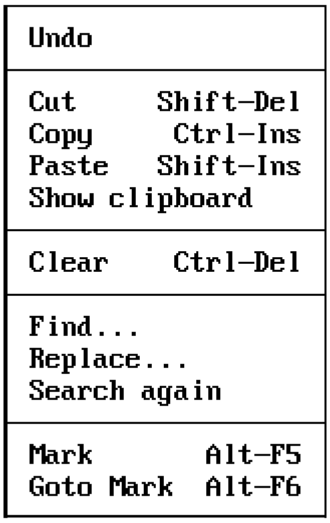
3.2.3 Window



The Window menu contains management commands for windows operating with program tools for emulation memory and source text editing:

- ***Next*** *(F6)* - switches from the active window to another one. The active window is pointed up with double-lined border.
- ***Previous*** *(Shift-F6)* - switches from the active window to previous one.
- ***Zoom*** *(F5)* - either enlarging to maximal size or shrinking back the window.
- ***Close*** *(Alt-F3)* - closes the active window.
- ***Resize*** *(Ctrl-F5)* - switching the window into the enlarging/reducing mode, or moving the window around the screen.
- ***Tile*** - placement of the windows on the screen in a way that all open windows are visible. This mode is convenient if only a few windows are open.
- ***Cascade*** - placement of the windows stacked like cards in a card index.

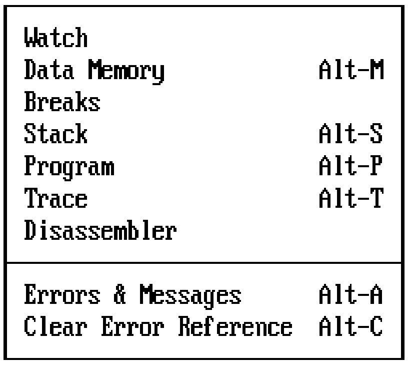
3.2.4 Edit



The Edit menu lets you manage editing commands and tools:

- ***Undo*** - this command takes back the last editing command, e.g. deleting a word or a line.
- ***Cut*** *(Shift-Del)* - this command removes the selected block from your text and places it in the clipboard.
- ***Copy*** *(Ctrl-Ins)* - this command copies the selected block in the clipboard.
- ***Paste*** *(Shift-Ins)* - this command inserts text from the clipboard into the current window
- ***Show clipboard*** - this command displays the clipboard window
- ***Clear*** *(Ctrl-Del)* - this command deletes the selected block but does not put it into the clipboard
- ***Find*** - lets you search for a text string
- ***Replace*** - lets you search for a text and replace it by another one
- ***Search again*** - repeating the last *"Find"* command
- ***Mark*** - marking the text at a cursor position intended for future return to this location using the *"Goto Mark"* command
- ***Goto Mark*** - returns to the line previously marked using the *"Mark"* command

3.2.5 Views

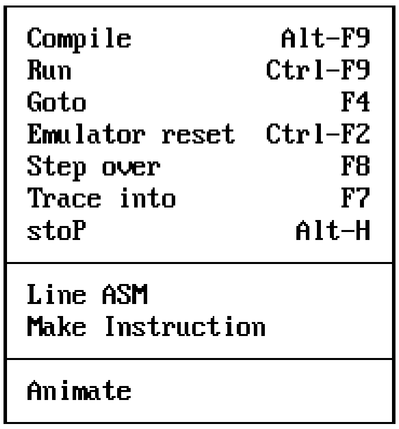


The views menu lets you manage commands and tools editing the emulation memory:

- ***Watch*** - display and editing program variables with respect to user choice
- ***Data Memory*** *(Alt-M)* - display data memory
- ***Breaks*** - displays the events database. Events include all types of breaks (points where program stops) and events for conditional tracing.

- ***Stack*** *(Alt-S)* - displays the stack and highlights the stack pointer position
- ***Program*** *(Alt-P)* - displays the program memory
- ***Trace*** *(Alt-T)* - displays the trace buffer
- ***Disassembler*** - displays the program memory disassembled to instruction set mnemonics
- ***Errors & Messages*** *(Alt-A)* - displays a window for error and message created by the compiler
- ***Clear Error Reference*** *(Alt-C)* - unhihglighting the line where an error was found in the source text

3.2.6 Run

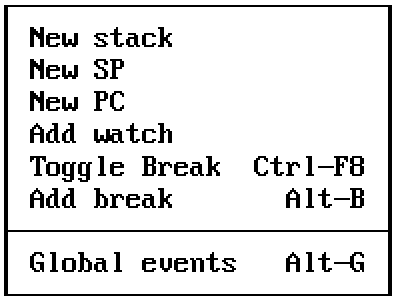


This menu contains commands for processing the source text by the compiler and various types of running an application:

- ***Compile*** *(Alt-F9)* - compilation of the application source text
- ***Run*** *(Ctrl-F9)* - running the program in emulation memory. Running is not inhibited even after errors was found and compilation was unsuccessful. In this case, the last successfully compiled program or random code is used.
- ***Goto*** *(F4)* - this command runs the program to the line where the cursor is in the source text
- ***Emulator reset*** *(Ctrl-F2)* - resets the emulator, i.e. emulated processor is set in accordance to reset specification
- ***Step Over*** *(F8)* - this command executes the next statement (single instruction, subroutine or a macro). It means stepping over the program without branching off into other subroutines and macros which are execute at full speed.
- ***Trace Into*** *(F7)* - stepping the program statement by statement

- ***Line Asm*** - line assembler (compiler). It is usually used for small correction (patch) in the program memory. Caution: in this case the program memory is modified but the source text leaves intact. Therefore, the source text does not correspond to the code memory any more.
- ***Make instruction*** - this command lets you make an instruction out of sequence, i.e. independently on the real contents of the program memory
- ***Animate*** - sets the emulator into the animation mode. Then, the application program goes step by step, while the entire data memory as well as other internal processor registers are completely read after each step.

3.2.7 Debug


```
New stack
New SP
New PC
Add watch
Toggle Break  Ctrl-F8
Add break       Alt-B
---------------------
Global events   Alt-G
```

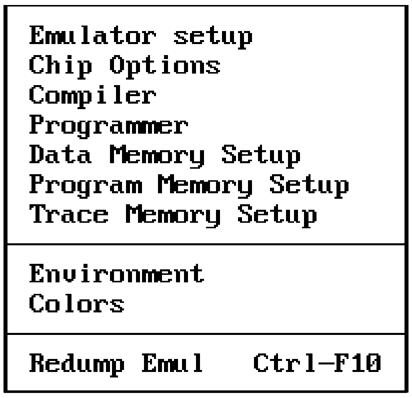
The Debug menu controls all the features of the debugger. You can specify and modify internal processor registers that are not directly accessible by the program:

- ***New Stack*** - modifies the stack
- ***New SP*** - modifies the stack pointer
- ***New PC*** - modifies the program counter
- ***Add Watch*** - adds an item into the watch window
- ***Toggle Break*** *(Ctrl-F8)* - sets a simple address break (breakpoint on selected address in the program memory)
- ***Add Break*** *(Alt-B)* - sets another break or event using the event editor
- ***Global Events*** *(Alt-G)* - sets or modifies global events (specifications of events for breaks)

3.2.8 Option

The Option menu includes all user settings concerning the environment as well as the emulator:

- ***Emulator Setup*** - setting of the emulator parameters, e.g. clock frequency, processor type and so on



- ***Compiler*** - setting of the compiler parameters, output listing format and output file format of the compiler
- ***Chip Options*** - lets set parameters of the emulated processor. These should be an oscillator mode, start-up timer enabled/disabled, watchdog enabled/ disabled and so on. This setting depends on the processor type and family. This setting corresponds largely to the configuration word of the processor (configuration fuse)
- ***Data Memory Setup*** - allows to set parameters for data memory displaying
- ***Program Memory Setup*** - allows to set parameters for program memory displaying
- ***Trace Memory Setup*** - allows to set parameters for trace buffer displaying
- ***Environment*** - user setting of the environment
- ***Colors*** - setting colors of the objects and displaying on the screen
- ***Redump Emul*** - after this command the complete emulation memory and state of the emulator is read

3.3 Source Text, Project and Object

Development of an application using the integrated environment can be accomplished by a few ways. For simple tests you can directly use the compiled source text of the program. In more complicated cases, or for applications with fixed pin assignment (e.g. because of PCB layout needs) and so on, it is recommended to take an opportunity to construct a project.

3.3.1 Arrangement of Directories

When developing an application, you should remember the system of storing information about the application program on the hard disk of the computer. The integrated environment works with two important directories which can be called an installation directory and a work one. The installation directory means the directory where programs necessary for running the environment reside. This directory can be used as a work one, it is true, but it is not recommended because of security of the data. Another reason for it is, if the environment is installed on the computer network, which is used to have the read-only attribute set on all installation directories. The integrated environment use the installation directory for read only. Work directory means any current directory, where it is allowed to read and write data. In this directory the environment always creates a standard configuration project, i.e. a file named DPIC.PRJ and a color palette file DPIC.CFG. The standard project file contains information about the setting of the environment of the previous execution without opening a user project. The color palette file contains color codes for all user projects in this directory.

3.3.2 Source Text and Project

Source text is created using the built-in text editor. The file can be open by the File command in the main menu. If you write a text without opening a project, the file can have any name with respect to MS-DOS convention. If the project is open, the name of the source text file must match the name of the project. After closing the project, a configuration file “name.PRJ” is created. The main project file containing the source text is defined by the name of the project as well as the object file, if this exists.

3.3.3 Object File

The compiler creates the object file after successful finishing the compilation. Besides of instruction operation codes it contains some additional information, e.g. information about the mapping the user variables in data memory or about locations of labels in program memory. If the project is open, the project file has the same name as the source text file (and of course the project).

3.3.4 Project

Because of the environment allows to have more text files open on the desktop not only with source but even with any other text files, the priority of the compilation must be defined. It is done as follows: if the project is open, compilation of the main project file is always called independently on activity of any source text windows. If no project is open, the source text of current window is compiled. For managing projects, the File commands from the main menu can be used.

3.4 Source Text Editor

Source text editor allows to edit a text up to 64 KByte of size. If the size of your source text exceeds this limit (e.g. if you use verbose comments), you can use the Include directive to avoid this situation. This directive lets include multiple files into the source text for compilation purpose. Another reason for separating extensive source texts into more files is a clear arrangement.

3.4.1 Source Text Editing

- ***Cursor movement*** - the cursor moves around the active window area using the arrow keys (by characters) the or *Ctrl* and right/left arrow key (by words). Clicking the left mouse button moves the cursor to the mouse pointer position.
- ***Marking a block*** - the block is marked from keyboard by *Ctrl-KB* (beginning of the block) and *Ctrl-KK* (end of the block), or by dragging a mouse while the left mouse button is pressed.
- ***Unmarking a block*** - the block is unmarked by moving cursor after the block is marked, or by *Ctrl-H* keys.
- ***Move or Copy block*** - text block can be moved or copied via a hidden temporary text buffer called a clipboard. Mark the selected block of text and copy it into the clipboard using hot keys or menu commands. Use *"Edit→Cut" (Shift-Del)* to move or *"Edit→Copy" (Ctrl-Ins)* to copy. Then move the cursor to the new position and insert the block to a text using *"Edit→Paste" (Shift-Ins)* command.
- ***Deleting a text or a block*** - to delete a character, use the *Del* key (deletes a character at a cursor position) or *Backspace* (deletes the character to the left of the cursor). To delete the entire marked block, use *"Edit→Clear" (Ctrl-Del)* command.
- ***To recover the last deleted text*** back to the previous form, use the *"Edit→Undo"* command.
- ***Displaying of the clipboard*** - to display the clipboard, use the *"Edit→Show Clipboard"* command.
- ***Searching the string in text*** - the editor allows to search of text strings with or without case sensitivity or search for whole words only using the *"Edit→Find"* command.
- ***Replace text*** - the *"Edit→Replace"* command searches for a string and replaces it with another string with or without case sensitivity or replaces whole words only, the first occurrence or all occurrences in the text. You can also choose prompting or no prompting option.

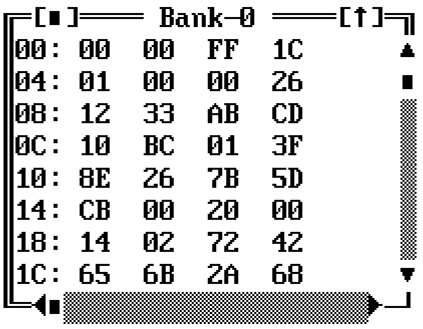
3.4.2 Additional Editor Utilities

- ***Automatic indentation of text*** - the autoindent command provides for automatic indentation of text which affects positioning of the cursor when *Enter* is executed. If it is set (on), the cursor moves to the position according to previous line, otherwise it goes to the first column.
- ***Syntax highlight*** - this option is global for all objects (including editor) of the integrated environment according to user's setting. This option sets displaying of keywords different in colors from another parts of text.
- ***Display the program counter*** - the emulator executes the emulation according to the program. The program counter varies during emulation with respect to it. The editor shows the value of the program counter all the time using the highlighted line of current instruction in source text.
- ***Display breakpoints and traces*** - these options allow to set and display the address break and display a trace point. The trace point is displayed by highlighted character in the first column in the line. If space is the first character of that line, the character is displayed. The breakpoint is displayed by the highlighted line.
- ***Fixed references to errors and breakpoints*** - this is a subsidiary feature of the editor. It allows to edit source text which contains marks where the compiler found errors, and, after deleting or inserting a line the editor accepts the new relative positions of error marks. The same feature is used for breakpoints and points of conditional tracing. See chapter 3.19. in detail.

3.5 Program / Data Memory Editor

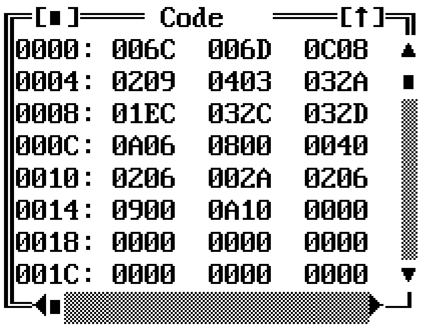
The program and data memory editors are intended for direct editing of memory area of the emulated processor. These are matrix-oriented views. The first column always means the address of the first byte or word in the line. Values of the subsequent memory cells follow. For example: 08: 12 33 AB CD. This means value 12 at location 08, 33 at location 09 and so on.

Data memory editor



```
[■]=== Bank-0 ===[↑]
00: 00  00  FF  1C
04: 01  00  00  26
08: 12  33  AB  CD
0C: 10  BC  01  3F
10: 8E  26  7B  5D
14: CB  00  20  00
18: 14  02  72  42
1C: 65  6B  2A  68
```

Program memory editor



```
[■]=== Code ===[↑]
0000: 006C  006D  0C08
0004: 0209  0403  032A
0008: 01EC  032C  032D
000C: 0A06  0800  0040
0010: 0206  002A  0206
0014: 0900  0A10  0000
0018: 0000  0000  0000
001C: 0000  0000  0000
```

Addresses are listed always in hexadecimal, while values can be displayed in any of five formats as follows: hexadecimal, decimal, binary, octal and ASCII characters. Further, you can select number of columns of the list (1 through 6) and set the size of fixed-length format. All these selections can be done by the *"Option→Data Memory Setup"* or *"Option→Program Memory Setup"*.

- ***moving the cursor*** - editors display two cursors. The former is identical to cursor of the text editor and it is allowed to move it around the editor window using arrow keys. The latter (in the form of a block) appears only where the memory value is allowed to edit. To move the cursor, you can use the left mouse button as well.
- ***moving the block cursor*** - you can move the block cursor to the next or previous memory item. If the cursor is not displayed, it moves to next or previous item relative to the standard cursor. *Tab* or *Ctrl-Right* arrow keys moves the block cursor to the next item, *Ctrl-Left* arrow moves it to the previous one. The first location of the edited memory can be selected by *Ctrl-PgUp*, the last one by *Ctrl-PgDn*. The first item in the line can be edited by *Home*, the last one by the *End* key.
- ***commands for breakpoints and trace points*** - these commands allow to enter and display the Address Break (the point of break on the address) and display the trace point. The trace point is displayed by highlighted character in the first column in the line. The breakpoint is displayed by a highlighted line.
- ***editing of a memory location and an automatic shift*** - the editor allows to edit the selected item using only valid characters defined for the current radix. For example, only characters 0 and 1 are valid for binary radix and 0 through 9 in decimal radix. An automatic cursor shift means that after editing the last character of the item the cursor goes automatically to the next item.

3.6 Disassembler

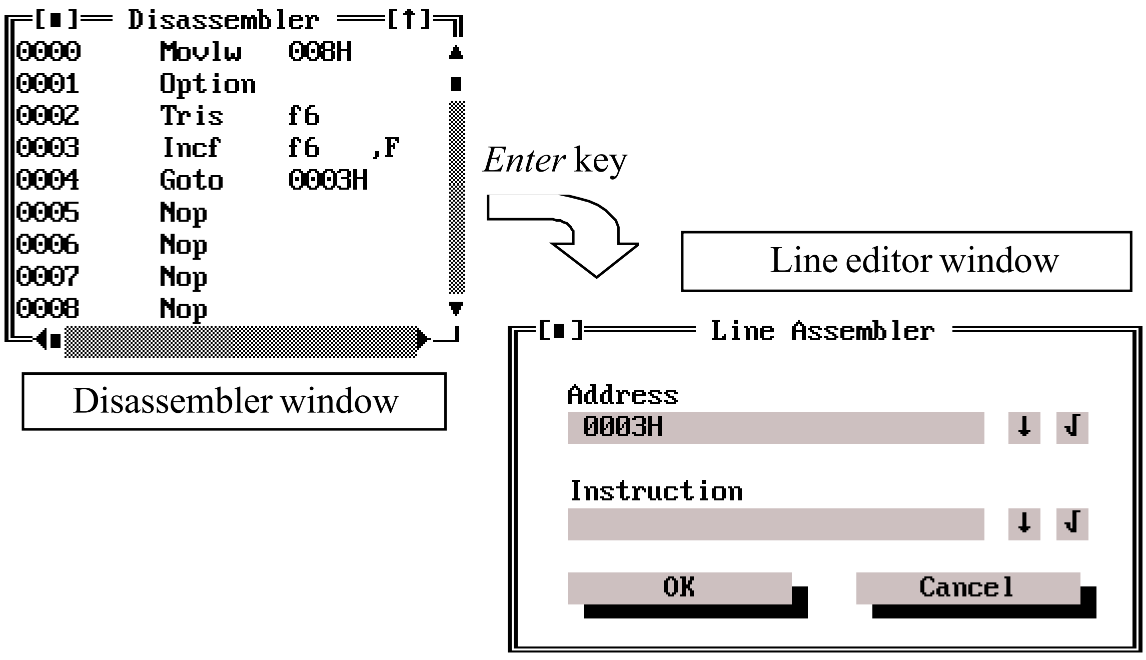
The disassembler is intended for evaluation and displaying the contents of the program memory including the instruction mnemonics. The list is arranged into two columns. The former displays the hex memory address, the latter contains the instruction corresponding to the code on this address.

Using the disassembler:

- ***cursor moving*** - moving the cursor in the disassembler window by the mouse or by the arrow keys is the same as for source text or memory editing.
- ***editing and compilation*** - you can edit the program memory using instruction mnemonics with the help of the disassembler. When editing, move the cursor to the desired position and execute *Enter*. Then the dialog box of a line compiler is

open. *Enter* an address (or leave the address unchanged) and write the instruction in the symbolic mnemonics. Press the OK button in the dialog window. Then, if the entry is able to be evaluated and compiled, the program memory is modified at this location. If there is an error, the message in a dialog box appears and the memory is not changed.

- ***displaying of breakpoints and trace points*** - the breakpoints are displayed by highlighted line the trace points are marked by characters.
- ***displaying of the program counter*** - in the course of emulating the microcontroller goes through instructions and the value of program counter changes. The disassembler displays this value using the highlighted line with respect to the current location of the program counter.
- ***Highlighted syntax*** - this option can be set globally for all objects of the integrated environment according to user's setting. This option sets displaying of keywords different in colors from another parts of text.



```
Disassembler
0000     Movlw   008H
0001     Option
0002     Tris    f6
0003     Incf    f6   ,F
0004     Goto    0003H
0005     Nop
0006     Nop
0007     Nop
0008     Nop
```

```
Line Assembler
Address
0003H
Instruction

OK        Cancel
```

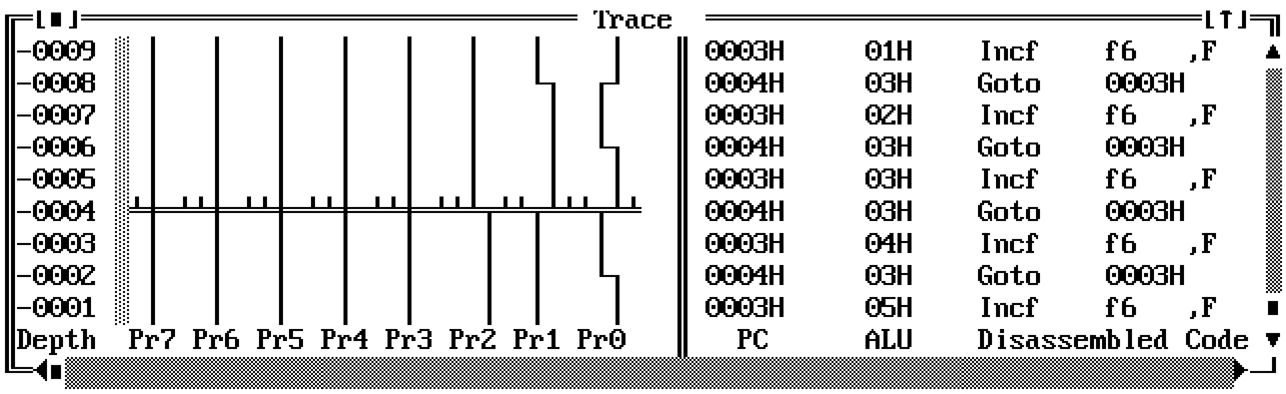
3.7 Trace buffer

Wile debugging, the information about the program flow is stored into the trace buffer. It is possible to store either all program steps or selected program parts only (conditional tracing). The trace buffer list is arranged by the addresses and is listed line by line, the last command at the bottom. The output is user-formattable.

3.7.1 Trace Memory Arrangement

The trace memory word is 32 bits wide. As for information storing, this array is separated in three parts. The initial sixteen bits form an area where the program counter and some subsidiary information about setting trace on/off are stored. The next eight bits of the memory word is intended for ALU result storing. Meaning of the last eight bits is user-selectable. Either values transferred via the internal processor data bus or values sampled on the external logic probe can be stored there. The trace buffer lists these data. Lines are separated into columns as follows:

- ***Depth*** - depth pointer of the trace buffer. It is a number signed with a minus sign ranging from -1 to -4096, which means the history of stored data.
- ***Probes*** - an eight-column array for displaying logical signal samples on the external probe. If the tracing of the internal data bus is set, the columns display the data. The columns can be named by user. In case of internal data bus tracing, the default names Bs7 .. Bs0 are used. Names are entered using the trace buffer list. See chapter 3.17.
- ***PC*** - the program counter value for current history of the line is displayed in this column
- ***ALU*** - the column displays results of the arithmetic/logic unit for current history, i.e. for the currently executed instruction
- ***Disassembler*** - this column displays the currently executed instruction which is reconstructed from the program counter value and the program code.



```
                              Trace
-0009                                         0003H    01H   Incf    f6   ,F
-0008                                         0004H    03H   Goto    0003H
-0007                                         0003H    02H   Incf    f6   ,F
-0006                                         0004H    03H   Goto    0003H
-0005                                         0003H    03H   Incf    f6   ,F
-0004                                         0004H    03H   Goto    0003H
-0003                                         0003H    04H   Incf    f6   ,F
-0002                                         0004H    03H   Goto    0003H
-0001                                         0003H    05H   Incf    f6   ,F
Depth  Pr7 Pr6 Pr5 Pr4 Pr3 Pr2 Pr1 Pr0         PC      ALU   Disassembled Code
```

Moving the cursor in the trace listing can be controlled using arrow keys or the mouse. Move the cursor to the beginning of the line by the *Home* key and to the end of the line by the *End* key. To go to next or previous page of listing use the *PgDn* or *PgUp* keys. The *Ctrl-PgDn* or *Ctrl-PgUp* keys go to the beginning (to the youngest data) or top (to the oldest data) of the trace buffer.

3.7.2 Setting and Selections

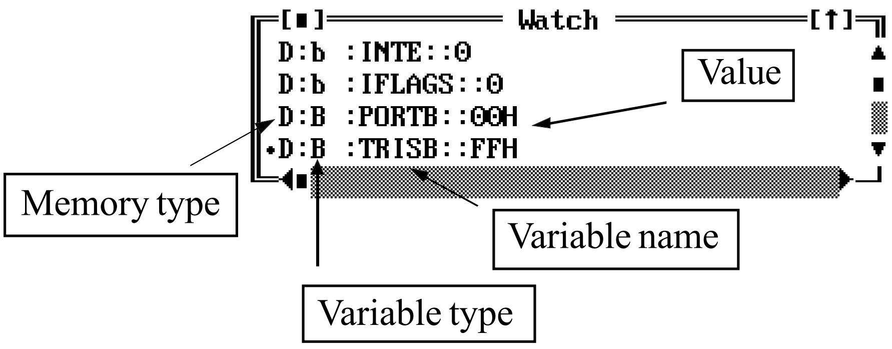
The user can format the trace buffer listing by *"Option→Trace Memory Setup"* from the main menu. Selections like display ([X]) or not display ([ ]) for each columns Probes, ALU and Disassembler are available. For eight columns of Probes you can choose a three-character name for each column, select separate column displaying and numeric or semigraphic displaying.

3.8 Watch (displaying of user-defined variables)

Watch (editor of user-defined variables) is intended for formatting of processor register lists on the screen. The editor allows easy register modification and displaying and modification even registers not accessible in the processor data memory area. Editor of user-defined outputs lets display up to 256 items. More items would not spell a clear view. In spite of this, to allow watching more variables, you can open more editors at a time. The number of them is not limited at all. The only restriction is a memory capacity of the computer.

- ***cursor moving*** - selected item highlighting and a point at the first column mean the cursor in editor of lists. To move it around the window, use arrow keys or the mouse. Further, use the *PgDn* and *PgUp* keys for paging or *Ctrl-PgUp* and *Ctrl-PgDn* for moving the cursor to the first / last record in the listing.
- ***listing format*** - every displayed variable is listed according to the previously defined format, which is partly fixed and partly user-specific. The implicit listing has to do with the first and second items. The first one refers to the memory or register type and can have Virtual, Data or Address values. Virtual means a register generally not accessible in the data memory, e.g. the WREG accumulator (this register is intentionally named WREG to avoid collision with the destination register assignment (W or F) for write instructions). The Data mean registers or variables in the accessible area of the data memory. The second column contains a variable type. One of four designations can appear there:
  - ***B*** (byte) - eight-bit variable, register
  - ***b*** (bit) - a single bit
  - ***BA*** (byte array) - array of bytes
  - ***bA*** (bit array) - array of bits

For details of variable types, see Chapter 4.4.2 concerning extended data types. The last term is Address, which means the address where the variable is mapped in the memory. Another item in the line is a variable or register name. In case that the register address is displayed in the line, the @ character precedes the register name. The last

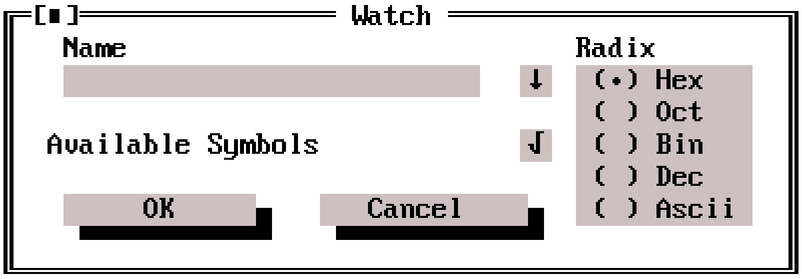


```
[■]════════ Watch ════════[↑]
D:b  :INTE::0
D:b  :IFLAGS::0
D:B  :PORTB::00H
•D:B  :TRISB::FFH
```

item is a register (address) value in the processor emulation memory. Four radixes (hexadecimal, decimal, octal or binary) and ASCII character representation are available for displaying values. The address is always in hexadecimal. In case of bit variables, the comma character and an order of the bit (0 through 7) follows the address. In case of bit array, bits are numbered in ascending order of binary numbers as well as byte addresses.

3.8.1 Formatting and Editing a Value

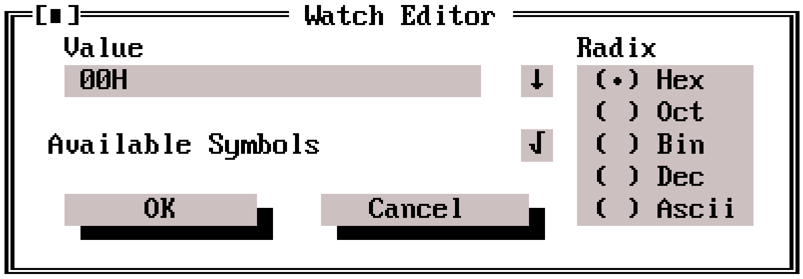
You can add a variable to the listing by the *"Debug→Add Watch"* command from the main menu or by pressing the Ins key. Response of the editor depends on the cursor position. If the cursor is at the end of the list, the variable is appended. If the cursor is at an existing item, the new item is inserted there. Editing is invoked by selection the item by the cursor and the *Enter* key.



```
[■]════════ Watch ════════
 Name                          Radix
                         ↓     (•) Hex
                               ( ) Oct
Available Symbols        √     ( ) Bin
                               ( ) Dec
    OK           Cancel        ( ) Ascii
```

Adding a variable

Editing the value



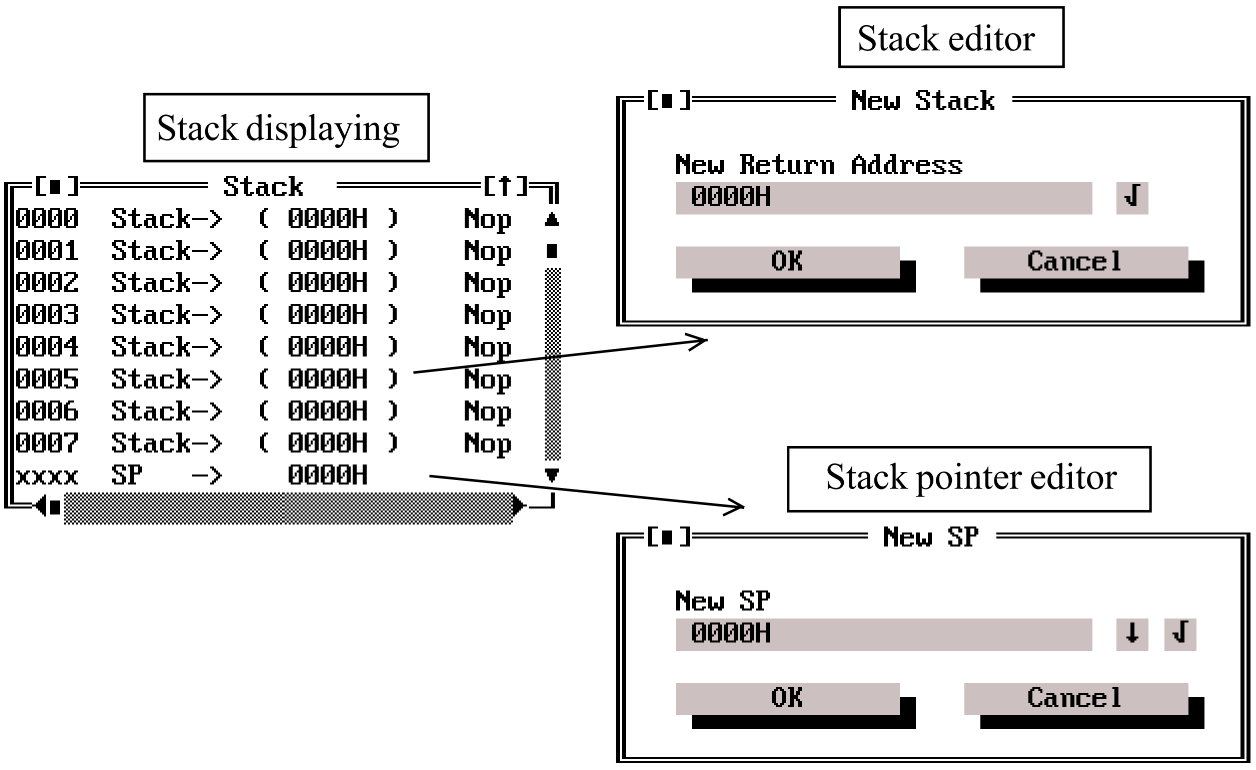
```
[■]════════ Watch Editor ════════
 Value                         Radix
 00H                     ↓     (•) Hex
                               ( ) Oct
Available Symbols        √     ( ) Bin
                               ( ) Dec
    OK           Cancel        ( ) Ascii
```

3.9 Displaying and Editing the Stack

The stack is displayed according to stored returning addresses and current stack pointer value. You can edit returning addresses and the stack pointer value using following commands:

- ***New Stack*** - invokes the stack editor
- ***New SP*** - invokes the stack pointer editor

These commands can be invoked from menu or by cursor and the *Enter* key.



```
Stack
0000  Stack->  ( 0000H )   Nop
0001  Stack->  ( 0000H )   Nop
0002  Stack->  ( 0000H )   Nop
0003  Stack->  ( 0000H )   Nop
0004  Stack->  ( 0000H )   Nop
0005  Stack->  ( 0000H )   Nop
0006  Stack->  ( 0000H )   Nop
0007  Stack->  ( 0000H )   Nop
xxxx  SP    ->   0000H
```

New Return Address: 0000H

New SP: 0000H

3.9.1 Stack value editor

It is a simple line editor with an object for displaying of available symbols. This view includes symbols of labels and constants found during compilation. A return address can be entered using a symbol (label) or an expression which can be evaluated.

3.9.2 Stack Pointer Editor

The editor is intended for current stack pointer modification. The value is entered as a number or using an expression which can be just evaluated.

3.10 Displaying and editing of events

Generally events mean breakpoints (points of breaking the running program) and trace points (points of tracing the program). The event editor allows to specify up to 256 different items specifying conditions for breaking or tracing an application program. Breaks can be watched and modified using the break editor.

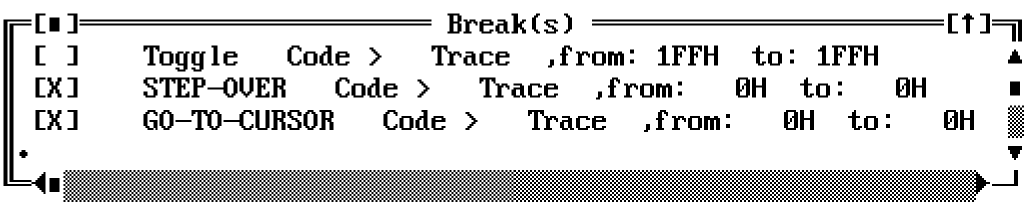
3.10.1 Displaying of Events

- ***positioning the cursor*** - the cursor is represented by a highlighting of the selected item and by a point in the first column of listing. To move the cursor around the window use arrow keys or the mouse. Further, use the *PgDn* and *PgUp* keys for scrolling a page or *Ctrl-PgUp* and *Ctrl-PgDn* for moving the cursor to the first or the last item.
- ***listing format*** - all displayed events are listed in a default format. The listing contains a column for local enable/disable event activity, one for event allocation list, and columns for event effect definition list, event name and event address range.
- ***enable/disable event*** - event effect enable/disable is displayed as follows:

[ ]	-	the event is disabled
[x]	-	the event is locally enabled, but its activity is disabled globally, i.e. the event is inactive
[X]	-	the event is enabled locally as well as globally, i.e. the event is active

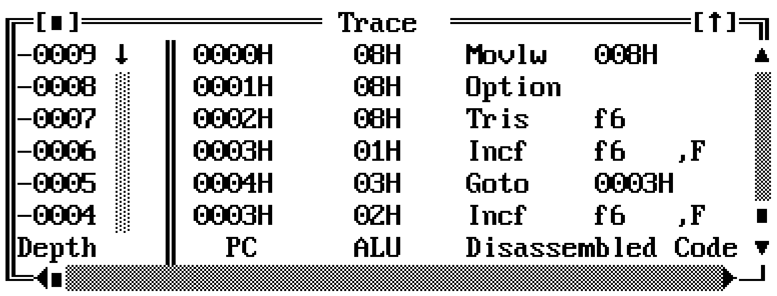
- ***event allocation*** - the column indicates a memory area where the event is allocated by its definition. There are two kinds of the allocation:
  - ■ Code - stop-point on the address or in the specified address memory range
  - ■ Data - stop-point on the address or in the specified data memory range
- ***event type*** - the column indicates an event type. The types can be as follows:

  a) for Code type allocation
  - ❍ *Address* - breakpoint on the address or in the specific address memory range (simple address break with a range selection option)
  - ❍ *Trace* - trace point (program pass through the specified address range is stored)



```
[ ]   Toggle   Code >   Trace  ,from: 1FFH  to: 1FFH
[X]   STEP-OVER   Code >   Trace  ,from:   0H  to:   0H
[X]   GO-TO-CURSOR   Code >   Trace  ,from:   0H  to:   0H
```

Break points



```
-0009   0000H   08H   Movlw   008H
-0008   0001H   08H   Option
-0007   0002H   08H   Tris    f6
-0006   0003H   01H   Incf    f6   ,F
-0005   0004H   03H   Goto    0003H
-0004   0003H   02H   Incf    f6   ,F
Depth   PC      ALU   Disassembled Code
```

Trace points

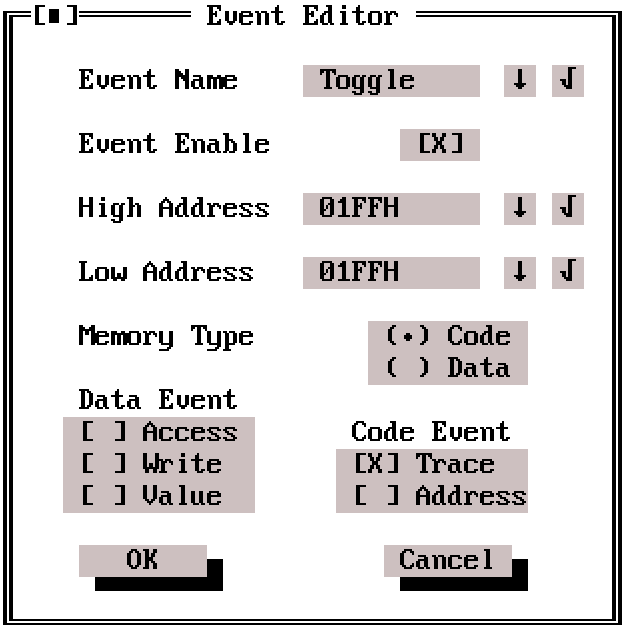
b) for Data type allocation

- *Access* - runing program is broken when the specified register or register in the specified address range is accessed (the access means read or write of any value).
- *Write* - runing program is broken when any value is written into the specific register or registers of the specific address range
- *Value* - runing program is broken in case of write into the specific register or registers of specific address range, if data to be written match the specific value including a bit mask. The emulator lets specify one value and one mask common for all events of this type. The mask and the value are implemented as the bitwise logical AND operation, i.e. only the bits of the value are significant, where the mask has ones. Other bits of the value are ignored. For example, given: value 0xFF and mask 0x1 is specified, and the Value break is set to the STATUS register. Then the break occurs when the carry bit (C) is set to 1.

It is allowed to group event types for the specified allocation type, which is indicated by the & character joining the events. The & character means event activity, e.g. an event is active on the specified address as a breakpoint as well as a trace point (resulting listing is *"Code > Address & Trace"*).

3.10.2 Editing of Events

An event editor is intended for entering and modifying of events. To add a new event definition to the list, invoke the editor by selecting the *"Debug→AddBreak" (Alt-B)* command from main menu or by pressing the *Ins* key in listing window. To edit a previously defined event, press the *Enter* key in the event list window. If the cursor is in the empty line, a new event is entered. If the cursor is on the existing event, this event is modified by the editor.



```
[■] Event Editor

Event Name      Toggle     ↓ √

Event Enable        [X]

High Address    01FFH      ↓ √

Low Address     01FFH      ↓ √

Memory Type         (•) Code
                    ( ) Data
Data Event
[ ] Access        Code Event
[ ] Write         [X] Trace
[ ] Value         [ ] Address

   OK                 Cancel
```

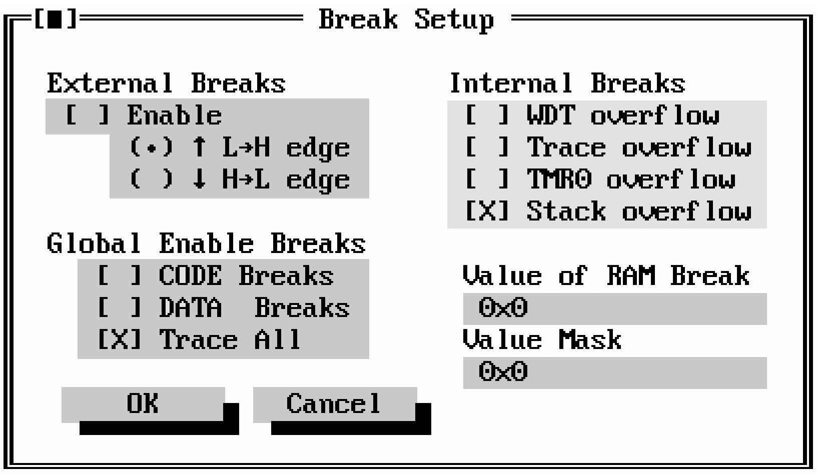
3.11 Global Events

Global events are user selections to affect running a program and set various emulator functions. The following options are available:

- ***External Breaks*** - enable to break a program by an external signal
    - *Enable* - enable/disable of external break signal
        - L-H edge
        - H-L edge - selection of active signal edge
- ***Internal Breaks*** - a group of selections to invoke program break when any of the following internal event of processor occurs:
    - *Wdt overflow* - break on watchdog timer overflow
    - *Trace overflow* - break on trace buffer overflow
    - *TMR0 overflow* - break on timer TMR0 overflow (the original name of TMR0 is RTCC, for compatibility with current Microchip terminology TMR0 is used in this manual).
    - *Stack overflow/underflow* - break on stack overflow/underflow

- ***Global Enable Breaks*** - global activation/deactivation of breakpoints and trace points
  - *Code Breaks* - enable/disable all breakpoints in program memory
  - *Data Breaks* - enable/disable all breakpoints in data memory
  - *Trace All* - enable/disable of conditional tracing (if the conditional tracing is disabled, all instructions are traced)
- ***Value of Ram Breaks, Value Mask*** - editor of value and mask for breakpoints in data memory with respect to the value and the mask (the value and the mask can be specified in decimal or hexadecimal in C-language convention, i.e. 0x10 and so on).

The OK button confirms setting, the Cancel button returns ignoring the changes.



```
[■]                Break Setup
External Breaks             Internal Breaks
[ ] Enable                  [ ] WDT overflow
    (•) ↑ L→H edge          [ ] Trace overflow
    ( ) ↓ H→L edge          [ ] TMR0 overflow
                            [X] Stack overflow
Global Enable Breaks
  [ ] CODE Breaks            Value of RAM Break
  [ ] DATA  Breaks           0x0
  [X] Trace All              Value Mask
                             0x0
   OK        Cancel
```

3.12 Emulator Setting

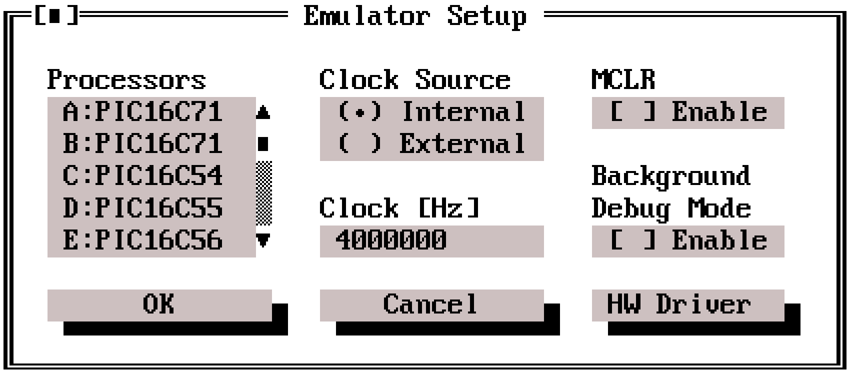
Emulator Setup selections are intended for basic configuration of emulation chip and corresponding setting of the integrated environment.

- ***Processors*** - setting of the type of emulated processor
- ***Clock Source*** - selection of clock signal source for emulated processor
  - *Internal* - selects the internal clock source. The clock frequency is set via the Clock parameter in this case.
  - *External* - the clock signal for the emulation chip is obtained from the hardware of the application via the CLKIN input of the processor.

    Default: Internal

Notes and recommendations:

1. Use the external clock source only if it is quite necessary for the application. Because of a response time, the minimal frequency 25 kHz is allowed. A great deal of internal operations of the emulation chip are synchronous with the clock of the emulated processor, and needs more clock cycles.
2. If parameter Internal is set, the OSCI/CLKIN input of the emulated processor is ignored, otherwise this pin serves as an input of active clock from the external source of an application. In both cases, all crystals, ceramic resonators and RCs are not accepted.



```
Processors        Clock Source      MCLR
 A:PIC16C71        (•) Internal      [ ] Enable
 B:PIC16C71        ( ) External
 C:PIC16C54                         Background
 D:PIC16C55       Clock [Hz]        Debug Mode
 E:PIC16C56        4000000           [ ] Enable
```

- ***MCLR Enable*** - enables reset of emulated processor from the -MCLR pin or when power supply of emulated processor falls under approximately 2.5 V.

  Default: Disabled
- ***Clock*** - setting the clock frequency of the internal clock source. The frequency can be set from 25 kHz to 20 MHz or up to maximal clock frequency of the emulated processor. The specific value is rounded to the nearest value that the frequency generator is able to generate (the step is 40 Hz). Worst case difference between the desired and rounded value for the minimal frequency 25 kHz is less than 0.1%, and falls for higher frequencies proportionally and becomes quite negligible in comparison with the accuracy of common crystal oscillator. The specified value (not the rounded one) is always displayed.
- ***Background Debug Mode***
  - *Enable* - reading and editing the data memory of the emulator in real time, i.e. in full operation of emulated processor
  - *Disable* - reading and editing the data memory of the emulator in the Halted mode only

Note: It is allowed to edit the microcontroller data memory in *Background Debug Mode*. If you edit the memory, consider, that the emulation chip is much faster than the editing, and the improper editing can abort the application program. In this mode an automatic cyclic data memory reading is also in progress.

- ***Hw Driver*** - pressing this button invokes a window with the description of the emulator hardware driver. Features of the driver and restrictions of selected processor emulation, if there are any, are included.

The OK button confirms the setting, the Cancel returns ignoring the changes.

3.13 Chip Options

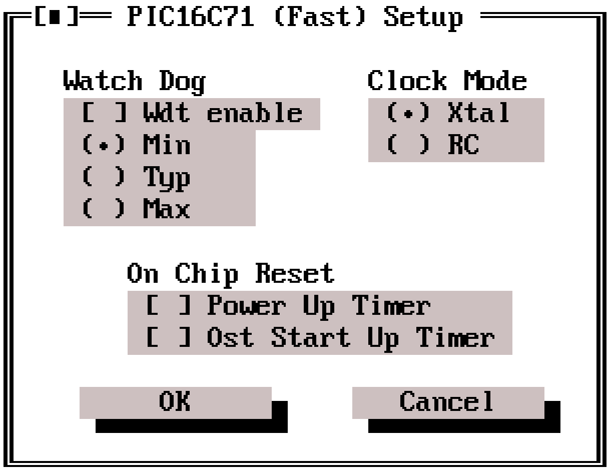
This is a dialog for setting of special functions of emulated processors. Contents of the dialog varies according to the selected processor. The most frequently used items are:

- ***Watch dog***
  - *WDT Enable* - enable of the watchdog timer (WDT). Even in case that WDT is enabled, it is possible to step the program without aborting the operation.

    Default: Disabled
  - *Min, Typ, Max* - three selections for WDT overflow time (useful for tolerance analysis in programs using the watchdog). Minimal, typical and maximal values can be set in accordance to the manufacturer's databook specification. For example, given: PIC16C5x values are 9, 18 and 30 ms.

    Default: typical
- ***Clock Mode*** - setting of the OSC2/CLKOUT output mode
  - *Xtal* - the CLKOUT pin outputs the frequency selected by the Clock option
  - *RC mode* - the CLKOUT pin outputs the clock which has 1/4 the frequency of processor clock according to the specified behavior of PICs in RC mode.

    Default: Xtal



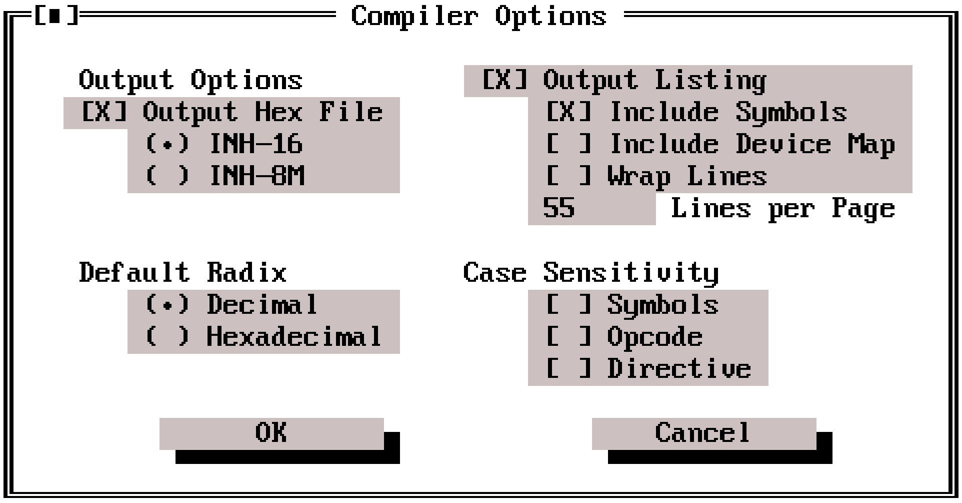
```
[■]== PIC16C71 (Fast) Setup

  Watch Dog              Clock Mode
   [ ] Wdt enable         (•) Xtal
   (•) Min                ( ) RC
   ( ) Typ
   ( ) Max

      On Chip Reset
       [ ] Power Up Timer
       [ ] Ost Start Up Timer

        OK                  Cancel
```

- ***On Chip Reset***
  - *Power Up Timer* - enable/disable of Power Up Reset timer
  - *Osc Start Up Timer* - enable/disable of the oscillator start-up timer

3.14 Setting of the Compiler Parameters

This setting influences a compiler run, especially user selections for output files formatting.

- ***Output Options***
  - *Output HEX File* - output file generation in Intel Hex format
    - INH-16 - output file is stored in Intel Hex-16 format.
    - INH-8M - output file is stored in Intel Hex-8M format.
- ***Output Listing*** - enables generation of output file with a text compilation listing
  - *Include Symbols* - listing does/does not contain the symbol table
  - *Device Map* - listing does/does not contain a package of the specified processor
  - *Wrap Lines* - the compiler wraps lines if the line length exceeds 80 characters
  - *Lines Per Page* - setting the number of lines per page of listing
- ***Default Radix*** - selection of default radix format to interpret numbers without radix specification.
  - *Decimal* - all numbers without radix specification will be interpreted as decimal
  - *Hexadecimal* - all numbers without radix specification will be interpreted as hexadecimal



```
[■]══════════════ Compiler Options ══════════════

  Output Options              [X] Output Listing
  [X] Output Hex File             [X] Include Symbols
      (•) INH-16                  [ ] Include Device Map
      ( ) INH-8M                  [ ] Wrap Lines
                                  55       Lines per Page

  Default Radix               Case Sensitivity
      (•) Decimal                 [ ] Symbols
      ( ) Hexadecimal             [ ] Opcode
                                  [ ] Directive

          OK                          Cancel
```

- ***Case Sensitivity*** - the compiler will be case-sensitive for the following:
  - *Symbols* - symbols, i.e. variable names, labels and so on
  - *Opcode* - instruction mnemonics
  - *Directive* - compiler directive

  The OK button confirms the setting, Cancel ignores the changes of setting.

Notes:

a) Mnemonics of instruction as well as compiler directives for case sensitivity is always written with the initial capital letter. Other letters are small. In case of symbols, the case is identified according to the first occurrence in the source text.

b) Processor type which a compilation is processed for, is a parameter of the compiler as well. This is parsed to the compiler from the emulator setup.

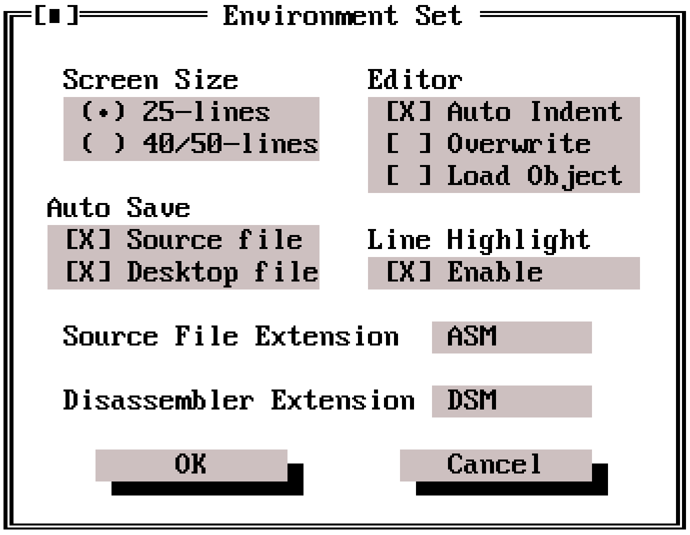
3.15 Setting of the Environment Parameters

Setting of the environment parameters is intended for user configuration of a screen layout and system behavior in special cases. This setting is always stored with a project (if it is open) or to the DPIC. PRJ file in work directory.

- ***Screen Size***
  - *25, 40/50 lines* - setting of the number of lines on the screen
- ***Editor*** - setting parameters of the source text editor which the editor will be invoked with
  - *Auto Indent* - automatic indentation the beginning of the line according to the previous line
  - *OverWrite* - overwrite mode switch
  - *Load Object* - if no project is open, the environment attempts to find the compiled "object file" corresponding to the name of the open file and install it into the emulation memory.
- ***Auto Save***
  - *Source File* - all source files are automatically saved after finishing a job in the integrated environment.
  - *Desktop File* - the environment setting including the layout of windows placed on the screen will automatically be saved.
- ***Line Highlight*** - all editor types display groups of symbols and marks of source file in specified colors. Additionally, it is a simple syntax checking of instruction mnemonics, defined operands and so on.

- ***Source File Extension*** - a three-letter extension for source text files
- ***Disassembler Extension*** - a three-letter extension for disassembled files

The OK button confirms the setting, Cancel ignores the changes of setting.



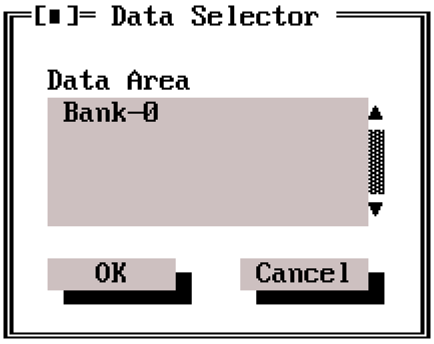
```
Environment Set

Screen Size              Editor
 (•) 25-lines             [X] Auto Indent
 ( ) 40/50-lines          [ ] Overwrite
                          [ ] Load Object
Auto Save
 [X] Source file          Line Highlight
 [X] Desktop file         [X] Enable

Source File Extension    ASM

Disassembler Extension   DSM

        OK                    Cancel
```

3.16 Setting of the Memory Listing Parameters

The *"Option→Program Memory Setup"* opens the Data Selector window, where you can select a register bank and after pressing the *Enter* key open a window of the data memory list editor. Similarly, set the listing of the program memory by the *"Option→Program Memory Setup"*.

3.16.1 Data selector

The Data Selector window includes the Data Area list with a scroll bar and a cursor. Use the cursor to select a bank (area) of data memory and press *Enter* to invoke the editor for modifying the memory. The selection can be cancelled using the Cancel button.



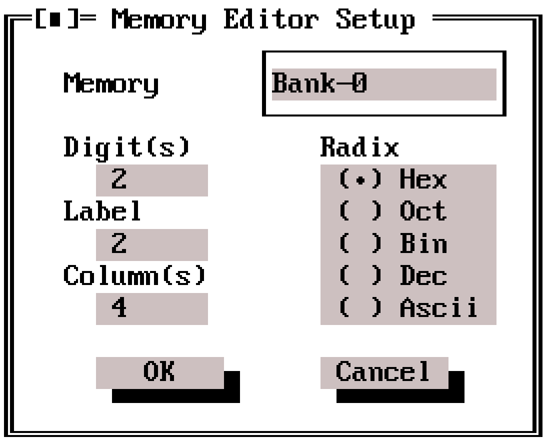
```
Data Selector

Data Area
 Bank-0

   OK        Cancel
```

3.16.2 Editor for the Setting of the Listing

You can format the listing using the following parameters:

- ***Digit(s)*** - a line editor for setting of the number of digits for a memory cell. Range 1 through 16 is valid. If the number exceeds the specific length, the number is truncated from the left, i.e. from the most significant orders. If the number is shorter, initial zeros are added to keep the specific format.
- ***Label*** - setting of the number of the decimal positions for address displaying. Range 1 through 8 is valid. The address is displayed in hexadecimal, initial zeros may precede.
- ***Columns*** - setting a number of columns of listing ranging 1 through 5
- ***Radix*** - setting of the radix to display memory. This radix is also used by the listing editor.

The OK button confirms the setting, Cancel ignores the changes of setting.


[■]= Memory Editor Setup

Memory: Bank-0

Digit(s): 2

Label: 2

Column(s): 4

Radix: (•) Hex ( ) Oct ( ) Bin ( ) Dec ( ) Ascii

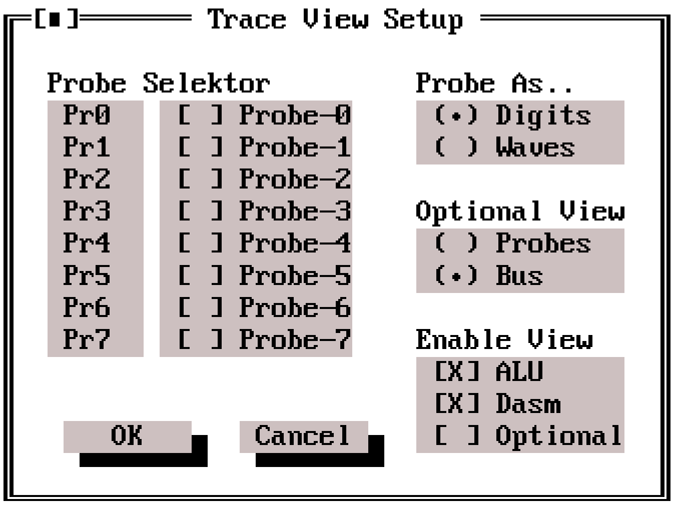
3.17 *Setting of the Trace Buffer Listing*

- ***Probe selector***
  - *Probe* - lets enter a three-letter name for each displayed signal
  - *Selector* - lets to select a probe displayed in listing
- ***Probe As..*** - the switch of the probe display type allows to select semigraphic or numeric displaying of the logical value
- ***Optional View*** - switches internal/external probes. The internal probes are always connected to internal processor data bus in Q1 cycle. For the instructions using File registers the input register value appears there. For literal operations, a literal copy is there. The external probe leads to a connector. This allows to connect it

to any application with the power supply of the same range as it is specified for the emulated processor. The switching level is approximately 1.6 V. The probe state is latched into the trace memory in late Q1 cycle, which is the moment when processor pins are also read.

- ***Enable View*** - enables displaying of trace memory blocks in a window
  - ❍ *ALU* - displaying of the ALU output
  - ❍ *Dasm* - displaying of a disassembled program
  - ❍ *Optional* - displaying of selected probe type, i.e. either external or internal, according to the Option and View setting.

The OK button confirms the setting, Cancel ignores the changes of setting.



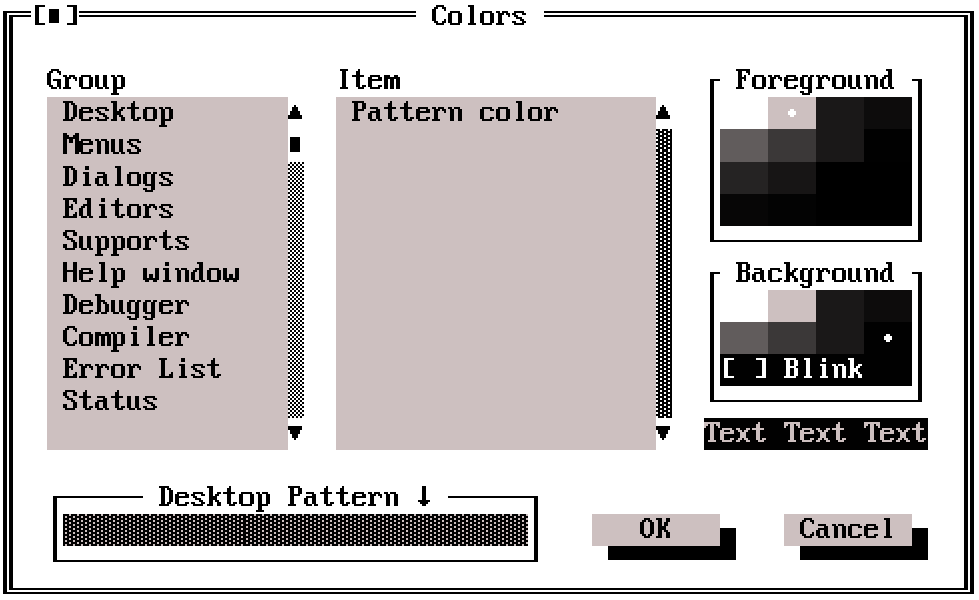
```
[■]═══════ Trace View Setup ═══════

Probe Selektor              Probe As..
 Pr0   [ ] Probe-0           (•) Digits
 Pr1   [ ] Probe-1           ( ) Waves
 Pr2   [ ] Probe-2
 Pr3   [ ] Probe-3          Optional View
 Pr4   [ ] Probe-4           ( ) Probes
 Pr5   [ ] Probe-5           (•) Bus
 Pr6   [ ] Probe-6
 Pr7   [ ] Probe-7          Enable View
                             [X] ALU
                             [X] Dasm
   OK        Cancel          [ ] Optional
```

3.18 Coloring the Environment

This selection allows to set colors and other parameters of objects on the screen.

- ***Group*** - allows to select main object groups
- ***Item*** - set for selection a subset of objects from the Group set
- ***Foreground*** - text color setting for specific object
- ***Background*** - background color setting for specific object
  - ❍ *Blink* - setting of the blink attribute for text of specific object
- ***Desktop Pattern*** - setting of the pattern of the work area on the screen

Note: Desktop pattern allows to select a character displayed on the work area. After selecting this option in dialog window, use the down arrow key to open a window with extended ASCII chart. Select desired character and press *Enter* then. After closing the dialog window for color setting the screen is filled by these characters to make a field.



3.19 *Accessory Elements on the Screen*

Work area on the screen outside a menu has also accessory objects of displaying, which specify integrated environment and emulation system.

3.19.1 Status

Status is located on the top bar together with the main menu. It displays an emulator condition, program counter value (if defined) and emulated processor type. Emulator conditions can be as follows:

- ***Undefined*** - undefined condition during environment initialization, when a contact to emulator is not established yet, or if emulator sent undefined data.
- ***Error*** - this condition comes on if an error in communication with the emulator is detected or if a user requirement is not successfully finished, e.g. unsuccessful memory verification.
- ***Running*** - indicates a real program run
- ***Halted*** - emulator is halted
- ***Break*** - emulator is stopped on the break condition
- ***Step*** - a program step is in progress
- ***Download*** - emulator configuration is in progress
- ***Verify*** - emulation memory verification is in progress
- ***Busy*** - emulator is busy waiting for the end of specified operation

3.19.2 Program Manager

Program manager displaying is located in the middle of the lower bar of the work area. It consists of two parts separated by a line. On the left a project name is displayed, on the right is a name of object file currently present in emulation memory. If neither project nor object is open, the “None” condition is displayed.

3.19.3 Available Memory Indicator

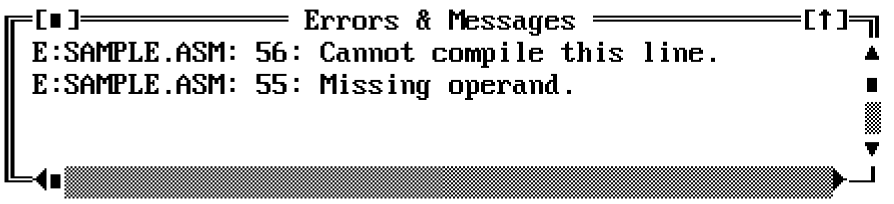
On the right bottom corner memory available to the integrated environment is displayed in bytes.

3.2 Window for Error Messages of the Compiler

A window for error messages of the compiler is an object for easy searching lines with errors generated by the compiler. The window is automatically open if any error in source text is found in any passage of compilation. After opening the window, errors are displayed in the following format:

line number, file, error type

By pressing the Enter key, source text editor is activated and the line where an error is detected becomes the current one. To move around errors use the *Alt-F8* (forward) and *Alt-F7* (backward) hotkeys. If the compilation consists of more source files and at least two of them contains errors, it is possible to select and tag desired error by the *Space* key. Then, errors will be browsed only in the file where the tagged errors are.



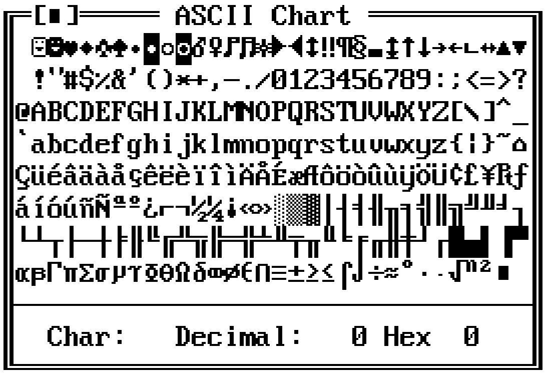
```
E:SAMPLE.ASM: 56: Cannot compile this line.
E:SAMPLE.ASM: 55: Missing operand.
```

It is allowed to correct an error just after it is found without loss of other errors. If you delete a line with an error, the error is deleted from the error listing as well.

During compilation, some mismatches or other problems can be found, which need a warning to the user. In this case the error window is not automatically open. If you want to open it, press the *Alt-A* keys in source text editor.

3.21 ASCII Chart

Using the *“≡→ASCII”* command you can open a window where the ASCII table is displayed. On the bottom there is a bar where the current character (selected by the cursor) is displayed including its numeric representation in decimal and hexadecimal.

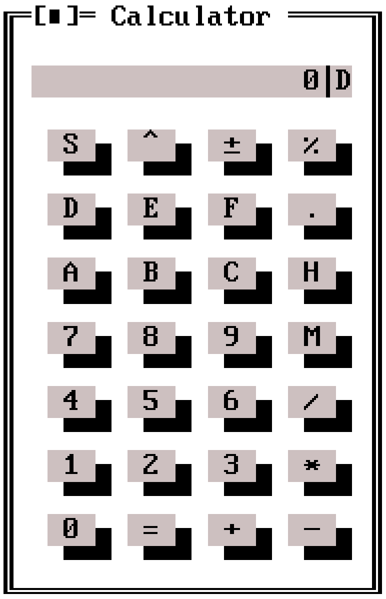


```
[■]═══ ASCII Chart ═══
 ☺☻♥♦♣♠•◘○◙♂♀♪♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~⌂
ÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜ¢£¥₧ƒ
áíóúñÑªº¿⌐¬½¼¡«»░▒▓│┤╡╢╖╕╣║╗╝╜╛┐
└┴┬├─┼╞╟╚╔╩╦╠═╬╧╨╤╥╙╘╒╓╫╪┘┌█▄▌▐▀
αßΓπΣσµτΦΘΩδ∞φε∩≡±≥≤⌠⌡÷≈°∙·√ⁿ²■

  Char:   Decimal:   0 Hex  0
```

3.22 Calculator

Using the *“≡→Calculator”* a window with a simple calculator is open. It operates with whole numbers ranging -32768 to 32767. The calculator is able to work in decimal, hexadecimal and binary radix. The following operations are available:

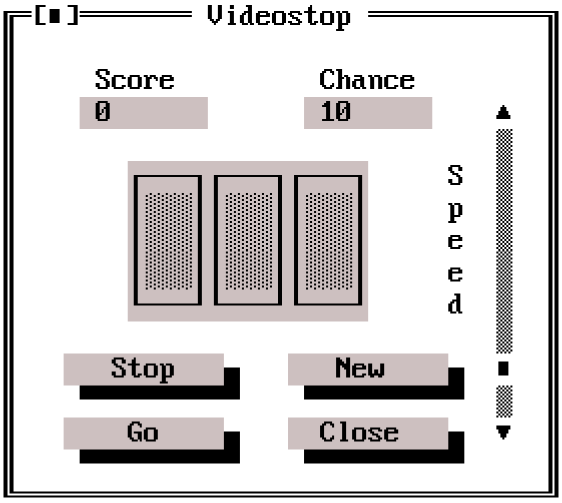
error deleting	-	the S key
sign change	-	the _ key
second complement	-	the ^ key
adding	-	the + key
subtracting	-	the - key
integer division	-	the / key
modulus	-	the M key
multiplication	-	the * key
evaluation	-	the Enter key
set decimal	-	the . key
set hexadecimal	-	the H key
set binary	-	the % key



Using keys corresponding to other buttons you can enter numeric operands for calculation. Only the 0 and 1 keys in binary, 0 to 9 in decimal and 0 to 9 and A to F in hexadecimal are valid.

3.23 Videostop

If your application is out of order, Videostop gets you completely down. It is a simple game for rest time. The goal is to stop casting the dice when at least two dice has the same number. Ten tries for it is available in one game. If a try is successful, ten points are added. If all the three of dice match, a hundred points bonus is added. If a try is unsuccessful, ten points are subtracted. Using the scroll bar you can set the casting speed. The first press of the Go button the dice are revealed. Another pressing this button starts casting. The Stop button stops casting. The New button resets a game. The Close button ends the game and closes the Videostop window.



3.24 Information

The Information Window gives the basic information about the hardware and software version.

3.25 Error Messages of the IDE

<u>Symbol not exist</u>

The specific symbol is unknown and then unaccessible. Error can be caused by a typing error or by the fact that a successful compilation did not precede a symbol specification.

<u>Syntax error</u>

An error in expression is found. Thus, the expression can be evaluated. Error can be caused by a typing error in symbol name, incorrect brace entry of an array name, unknown operand and so on.

Out of symbol range

Number of symbols which can be displayed in Watch window is exceeded. Open another window for desired symbol.

Not accessible symbol

The specific symbol is not accessible because of incomplete definition. The error occurs when editing a constant or so.

Insufficient memory

Specified memory block is unsuccessfully allocated. The error can occur when work memory structures are allocated.

Error of breakpoint type

Illegal breakpoint or trace point is found. The error can occur when an event is defined.

Out of break memory

Maximal number of event definitions exceeded. A particular solution of this problem can be using wider address range for event definitions.

Not accessible break(s)

The breakpoint is not accessible, e.g. the range exceeds the legal memory area of emulated processor.

Break out of range

Event definition contains an item which makes the event inaccessible by the hardware, especially if address of specified break is out of memory implemented in specific processor.

Unable to edit address

An attempt to edit on the address where current situation does not allowed it.

Value has been unchanged

An unsuccessful attempt to change emulation memory

Access denied

Specific activity is refused by emulator hardware or firmware.

Disk full

Disk capacity exceeded. File was not successfully saved.

Unable to open *.DEF file

Processor definition file can not be open. The environment recovers from this error by default setting. Allow to read the definition file to avoid the error.

*.DEF file read error

An error in reading of the definition file. The file is corrupted. Reinstall the file from installation diskettes.

*.DEF file not found

Processor definition file is not found in home directory of integrated environment.

Could not open project file

Project definition file can not be open. The file is not found or read error occurred.

Could not create project file

Unsuccessful creation of a project file. Either no free space is on the disk or an internal error of environment is found.

Project not opened

An attempt to close a project without opening any one.

Unable to load object file

Object file is not found or read error is detected.

Can not compile this line

The line assembler is not able to compile the line

Download file not found

The initialization file of emulator is not found. Corresponding files with the BIN extensions must be in home directory of the emulator.

Download error

Error in emulator initialization. It can be caused by hardware breakdown or by incompatibility of configuration file.

Unable to find help file

The help file is not found. This file with the HLP extension must reside in the home directory of the integrated environment.

Incompatible object format

An attempt to open a project with a structure not corresponding to the project format of integrated environment.

Recompile the program first

Inconsistency in creation of an object file and source file. Recompile the program.

No line to PC reference exists

Currently processed line has no reference to address in program memory. This error may be caused by an attempt to enter switching breakpoint after unsuccessful compilation or to enter a breakpoint to the line which has no representation in operation code, e.g. a comment line.

Incompatible object format

Structure of the object file format does not correspond to an object of integrated environment.

Unable to evaluate expression

Evaluation of specified expression is not possible, because of incorrect syntax or no existence of the symbol.

Error in Emul Init Procedure

Error in installation of emulated processor driver. It is an internal error caused by driver and hardware incompatibility or by corrupted PC parallel port, the cable or the emulator.

3.26 Information Messages of the IDE

Save current desktop?, Save current project?

If automatic desktop saving is disabled, after finishing a project job the user is asked to confirm saving of the current desktop.

Source has been changed - Rebuild?

Source text does not correspond to the file the compilation was created from. No references to lines exist or these references do not correspond to the current condition. If the source text is not compiled, the line corresponding to program counter value will not be highlighted in the source file while stepping a program.

Load object file?

If no project is open and you enter a command to open a file with the extension corresponding to the source file in the environment setting, you are asked to confirm the change of the emulation memory. If the change is not confirmed, the file is open but no references to source text lines are installed with all resulting consequences.

Emulator not found. Run Demo?

If communication with the emulator was not established, the situation can be solved either by confirmation (i.e. the demo simulator is invoked) or by refusal (i.e. the environment will repeat an attempt to establish communication). If no emulator is available, the environment can be invoked just in demo mode using the /DEMO parameter in command line.

Project Exists - Overwrite?

An ask to confirm the Save Project as.. command if a project of the specific name exists (and then it would be overwritten).

PC Overflow

An incorrect overflow of the processor program counter occurred because of an error in developed application program. This situation occurs if a PC value is changed from a maximum to zero while running the program, i.e. the user program “strayed”. You can suppress the message by disabling this in the Global Events option.

TMR0 Overflow

The counter/timer overflowed. The message can be suppressed by disabling this in the Global Events option.

WatchDog Overflow

The watchdog timer overflowed and the processor is reset. The message can be suppressed by disabling this in the Global Events option.

Stack Overflow

The stack overflowed or underflowed. The error occurs if number of nested procedures exceeds the processor stack depth or if a return from a procedure was executed without corresponding procedure call. Application programs can use this effect intentionally. In this case, you can suppress the error message in the Global Events option and use a break on the stack overflow condition. For standard programs, it is recommended to check the stack overflow/underflow to avoid errors of this type.

Trace Overflow

The emulation was stopped after trace buffer overflow. The trace buffer is full and then any instruction causes a loss of one history item written into the trace buffer. Stopping and a message can be suppressed by disabling this in the Global Events option.

External Pin Break

The emulator stopped because of an edge detection on the external probe. The message can be suppressed by disabling this in the Global Events option.

External Reset Occurred

Processor reset from an external source. The external reset source means any reset of emulated processor in other way than from a computer keyboard.

4. Compiler of the Assembler Language

This chapter describes the assembler programming language for microcontroller PIC16C5x and PIC16Cxx series.

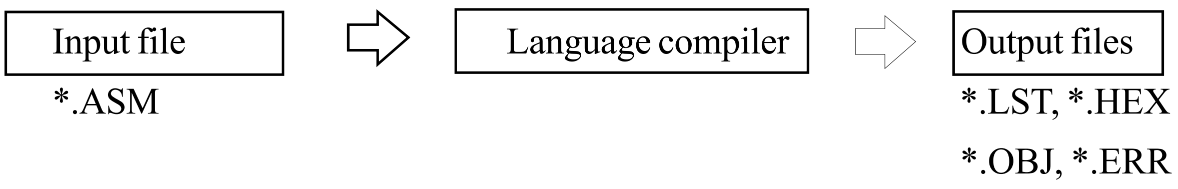
The language compiler is a program which compiles a source text of program (written with respect to rules of the specific language) to binary code able to be loaded into a processor and then run.

An input of the compiler is a file containing source text of the program (usually a file with the "ASM" extension).

Source text compilation goes in two passages. The first passage is a preparing (preprocessing) one, when syntax is checked, macroinstructions are expanded, symbols are defined and files are included. Run of this passage can be controlled using compiler directives either directly written into the source text or set via the dialog box of the integrated environment. After the preprocessing passage is done, real compilation of the program source text to binary code follows. The compiler outputs the following files:

- a file called "listing" containing a complete compilation protocol, numbered source file lines, marked errors, table of symbols and their references. The extension of this file is "LST".
- a file of binary code for the specified processor in the INHX-8M or INHX-16 format (it is selected by compiler setting in the dialog of the integrated environment). The extension of this file is "HEX".
- If errors occur during compilation, an error file is created. It contains file and line specification where errors was found and a brief description of the error.
- a file called "object" which contains information important to the integrated environment, e.g. names and values of defined symbols and references to lines of source text (needed for stepping). The extension of this file is "OBJ".

Block diagram of compiler operation:



4.1 Program Source Text

The program source text consists of program lines. Generally a program line contains four parts separated by spaces or tabs from each other. The format of a program line is as follows:

- ***label*** - a symbolic reference to specific program line to specify destination of jumps for program branching. Label is not compulsory and is usually used if you need to jump to this line from another part of the program. The label name must match conditions laid down to symbols (see symbols).
- ***command*** - mnemonic abbreviation of instruction from the instruction set of specific processor (see instruction set) or a directive (see compiler directives).
- ***operands*** - operands of an instruction or a directive of compiler. Number of operands depends on the instruction type (see instruction set) or a directive (see compiler directives). Operands are separated with a comma from each other.
- ***comment*** - any text initiated with the semicolon. Comments can contain any characters and are fully ignored by the compiler.

4.2 Constants

Constants mean any numeric or text information in program, which are known during compilation (i.e. not contents of registers which are not defined before running a program). Compiler arithmetics converts internally numeric constants to 16-bit unsigned integers (ranging 0 to 65535) and text constants to array of 16-bit numbers.

4.2.1 Numeric Constants

Numeric constants serve for example to specifying of instruction operands (register numbers, bit positions, literals). They can be entered in various radix or in ASCII. The following radix are available: binary (base 2), octal (base 8), decimal (base 10), and hexadecimal (base 16). You can specify a default radix (decimal or hexadecimal) to interpret numbers without radix specification. Pay attention to entries which can be misinterpreted depending on specified default radix (e.g. 1D means 1 in decimal default radix while it does 29 in hexadecimal). Radix specification is not case sensitive (e.g. 22h=22H).

4.2.2 Text Constants

Text constant is a special type of constant. You can imagine it as an array of numeric values assigned to specified ASCII characters. You can not use text constants

Radix	Entry	Example	Interpretation in default radix	
			decimal	hexadecimal
binary	B'<binary number>'	B'10'	2	2
	<binary number>B	1B	**1**	**1B (27)**
octal	O'<octal number>'	O'100'	64	40 (64)
	Q'<octal number>'	Q'100'	64	40 (64)
	<octal number>O	100O	64	40 (64)
	<octal number>Q	100Q	64	40 (64)
	\<octal number>	\100	64	40 (64)
decimal	<decimal number>	50	**50**	**50 (80)**
	D'<decimal number>'	D'100'	100	64 (100)
	<decimal number>D	1D	**1**	**1D (29)**
	.<decimal number>	.100	100	64 (100)
hexadecimal	H'<hexadecimal number>'	H'10'	16	10 (16)
	<hexadecimal number>H	10H	16	10 (16)
	0x<hexadecimal number>	0x10	16	10 (16)
ASCII	A'<ASCII character>'	A'd'	100	64 (100)
	'<ASCII character>'	'd'	100	64 (100)

everywhere but in the following cases only:

- *table definition* (the TABLE directive)
- *definition of program memory contents* (DB and DW directives)
- *definition of constant array of bytes* (the CONST BYTE [] directive)

Text constants are defined as ASCII characters enclosed in double quotes. (E.g. “This is a text”). If you need to insert a non-printable character or any numeric values, you can type hex codes in the \x<hex number> format. (E.g. “text1\x20text2” inserts the ASCII character of code 20 (space) between text1 and text2.

4.3 Expressions

Expression means mathematical or logical operations performed with constants. For example, “3+4” is an expression representing mathematical addition of the constants 3 and 4. The result is the constant 7. To construct mathematical and logical expressions, you can use operations in the following table. If an expression consists of more mathematical or logical operations, rules of precedence of operations is specified as follows: expressions within parenthesis are always evaluated first, then operations of precedence 2, then operations of precedence 3 and so on. If more operations have the same precedence, the expression is evaluated from left to right. In case of logical operations (labeled by the letter “L” in the table), the result of expression evaluation is either zero value (the expression value is false, e.g. 1>2) or the 1 value (the expression value is true, e.g. 2!=3).

Priority	Operator	Description	Example
1.	()	Parentheses	(7+8)/3
2.	!	Not	!(p1<10)
	~	Complement	~8
	+	Unary plus (positive number)	+3
	-	Unary minus (negative number)	-4
3.	*	Multiplication	2*5
	/	Division	5/3
	%	Modulus	5%3
4.	+	Addition	2+2
	-	Subtraction	6-2
5.	<<	Left shift	5<<2
	>>	Right shift	4>>1
6. (L)	<	Less than	x<y
	<=	Less than or equal	x<=y
	>	Great than	x>y
	>=	Great than or equal	x>=y
7. (L)	==	Logical equal	x==y
	!=	Logical not equal	x!=y
8.	&	Inclusive AND	x&y
9.	^	Exclusive OR	x^y
10.	|	Inclusive OR	x|y
11. (L)	&&	Logical AND	x&&y
12. (L)	||	Logical OR	x&&y

4.4 Symbols

Symbols in programs mean the following:

- ***symbolic constant names*** (defined by the EQU and SET directives)
- ***data type names*** (defined by the BIT, BYTE, CONST BYTE directives)
- ***labels*** (see program line)
- ***macroinstruction names*** (see macroinstructions)

Every symbol has its own name and a symbol definition. The syntax depends on a symbol type.

Symbol name must be unique in the entire program and must match the following conditions to be correctly interpreted:

1. symbol must begin with a letter 'A'-'Z', 'a'-'z'
2. symbol may contain only letters 'A'-'Z', 'a'-'z', numbers '0'-'9' and the underscore '_' character. Thus, separators (space, tab) must not be used.

4.4.1 Symbolic Names of Constant

You can use the EQU or SET directives to define a symbolic name of constant. Both of the directives have the same syntax and assign a constant value to symbolic name. Instead of a value an expression can be used and then the resulting value is assigned to the symbolic name.

```
        <symbol name> EQU <value>
        <symbol name> SET <value>
```

For example, given:

```
        Data1   SET     13h
        Data2   EQU     10h
        Data3   EQU     (1+Data1)*Data2
```

The difference between EQU and SET is only in possibility to change a symbol value. If a symbol value is defined by the SET directive, it is allowed to alter the value by another SET or by EQU directives. But if you attempt to change a value assumed by the EQU directive either by SET or EQU directives, the compiler reports an error.

4.4.2 Data Types

Data types are implemented in program language especially for easier work with processor registers. They allow quite unique addressing bits or bytes. Definition of a data type consists from a symbol name (matching the conditions laid down to a symbol name), a keyword specifying the data type and either an information about a location in memory area (address, bit, bank) or a value.

Besides of basic types, you can define an extended type (array). Arrays have a fixed number of components of one type. The size must be defined. Components are numbered from 0 and accessible via indexes written just after a symbol name in square brackets. A symbol name without an index means the first component, i.e. the component with the index 0. The Byte type is intended for instructions using registers, the Bit type for bit-oriented instructions and the constant type is suitable for instructions operating with literals.

The compiler recognizes and accepts the following data types:

- ***Byte*** - one register (byte) in the data memory. The definition contains an address and a register bank. See the following example.

```
        <symbol name> BYTE @<address>,<bank>
```

- ***Bit*** - one bit of a register in the data memory. The definition contains an address, bit number (bit position in the register, 0=LSb, 7=MSb) and register bank number. If a bit number is omitted, zero is assumed. See the following example.

```
<symbol name> BIT @<address>,<bit>,<bank>
```

- ***Byte Array*** - an array of registers (bytes) in the data memory. The definition contains a size of the array (number of bytes of the array), the address and the bank which specify the address of the first component of the array. Following components of the array (bytes) are allocated upwards. See the following example.

```
<symbol name> BYTE[size] @<address>,<bit>,<bank>
```

- ***Bit array*** - an array of bits in the data memory. The definition contains a size of the array (number of bits of the array), the address, the bit number (bit position in the register, 0=LSb, 7=MSb) and register bank number. If a bit number is omitted, zero is assumed. See the following example.

```
<symbol name> BIT @<address>,<bit>,<bank>
```

- ***Const Byte*** - an 8-bit constant (byte) equivalent to a symbol defined by the EQU directive. The definition contains a value assigned to the symbol. You can enter an expression instead of the value and then the resulting value is assigned to the symbol. See the following example.

```
<symbol name> CONST BYTE = <value>
```

- ***Const Byte Array*** - an array of 8-bit constants (bytes). The definition contains a size of the array and values separated by commas. If the size is entered by a number, it is automatically evaluated from the length of the array. You can enter the values as numeric constants, expressions, text constants or previously defined arrays of constant bytes.

```
<symbol name> CONST BYTE[size] = <value>,{,<value>...}
```

The following example illustrates syntax and allocation of various data types. No bank is specified and that is why the default bank 0 is assumed.

```
count   BYTE    @8
state   BIT     @9
array   BYTE[3] @0Bh
bits    BIT[10] @9,2
number  CONST BYTE = 1+2*3
data    CONST BYTE[] = "text", number, @state, 22h, 3*5
```

A constant symbol is not resident in the data memory. This is a thing of the compiler only. If a constant is used in program, the compiler just replaces the symbol by its numeric value. Values of the constant and the constant arrays are: number=7, data=data[0]=116 (A"t"), data[1]=101 (A"e"), data[2]=120 (A"x"), data[3]=116 (A"t"), data[4]=7, data[5]=9 (address of the "state" symbol), data[6]=34 (22h), data[7]=15.

Addre-ss	7	6	5	4	3	2	1	0
08h	number							
09h	bits[5]	bits[4]	bits[3]	bits[2]	bits[1]	bits[0]		state
0Ah					bits[9]	bits[8]	bits[7]	bits[6]
0Bh	array[0]							
0Ch	array[1]							
0Dh	array[2]							

Caution: The byte or bit types and the array of them mean contents of the register or a bit value which are not known during compilation (e.g. the "CLRF count" operation clears register 8 but does not clear the "count" symbol. That is why these data types can be used only with appropriate operations intended for them. To get an address of a byte or a bit or the array of bytes or bits, the "@" character (which means an address) must precede the symbol name.

4.4.3 Labels

A label is entered as the first item in a command line (see command line). You can emphasize the fact that it is a label by a colon followed the label name. But this does not affect the label name at all, because the colon is ignored.

For example, given:
```
    read1        MOVLW 10h  ; defines a label read1
```

For example, if you want to define more labels for a single address, it is allowed to use another way of definition called a standalone label. If you define a standalone label, neither a command nor a directive, only a comment is allowed in this line. Besides of this, a standalone label must start in column 1 or must be followed by the colon.

For example, given:
```
    ...
    label1:
    label2 CLRF     8h      ; defines two labels
```

4.5 Instruction Set of the PIC Processors

The instruction set is defined by the manufacturer of the processor, i.e. by Microchip in this case. Each processor family has its own instruction set. Instructions are usually called by names abbreviating the operation (mnemotechnic, symbolic instruction names).

The compiler compiles programs for 16C5x and 16Cxx families. Both of them are similar to one another and their instruction sets are the same (mnemotechnic instruction names) except of some differences. But binary opcodes are quite different. The 16C5x family uses 12-bit wide path, while 16Cxx does 14-bit wide one. Generally, the instruction set can be separated into groups as follows:

- ***byte-oriented instructions***
- ***bit-oriented instructions***
- ***literal instructions***
- ***control instructions***

4.5.1 Byte-oriented Operations

The byte-oriented instructions work with a data register. They allow to move data between a data register and the work register W, modify the register value (increment, decrement, rotation,...) and arithmetic and logic operations with a value in the W register.

An instruction operand is a register number which can be specified just by a constant or by the data type of byte.

Most of instruction of this category needs another operand to specify where the result is stored. In case of 0, the result is stored to the W register, in case of 1 the destination register is specified by the first operand. The compiler has predefined two symbols for this purpose (W=0 and F=1). If the destination is omitted, the register specified by the first operand is used.

For example, given:
```
    CLRF     8h    ; clear register 8h
    ; increments register 10
    INCF     10h,1
    INCF     10h,F
    INCF     10h
    ; store incremented value of register 10 to W register
    INCF     10h,0
    INCF     10h,W
    ; if you use the definition from the example of data types,
    ; then:
    MOVWF    array[1] ; copies W register to register 12
```

4.5.2 Bit-oriented Operations

The bit-oriented instructions operate with bits in data registers of the processor. They allow to set a bit to 1 or 0 or to test logical value of specified bit and branch a program depending on the result.

This kind of instruction uses the bit data type or one component of a bit array. These types fully specify the bit.

If you do not specify the operand using the bit type, two operands must be used. The first one specifies the register, the second one does the position of the bit in the register (0-7).

```
For example, given:
    BCF      8,2      ; clears bit 2 of register 8
    ; if you use the definition from the example of data types,
    ; then:
    BSF      bits[2]  ; set bit 4 of register 9
```

4.5.3 Literal Operations

The literal instructions are intended for modification of the work register W and for program branching and subroutine calling. The jump (GOTO), call (CALL) and return from subroutine (RETLW) belongs to them as well.

These instructions have a constant (defined in any way) as an operand.

```
Example: MOVLW   10h   ; writes 10h to the work register W
```

4.5.4 Control Operations

The control instructions are particularly intended for control of the processor operation (switch to “sleep” mode, reset of the “Watchdog” timer,...). The control instructions except of the TRIS instruction have no operand since they need no additional information. The TRIS instruction sets pins of the processor as input or output. The port register number is a parameter in this case.

4.5.5 PIC16C5x Instruction Set Summary

ADDWF - ADD W and F

Syntax:	ADDWF f,d	
Operation:	(W+f) → d	
Description:	Add the contents of the W register and register f. The result is stored in the W register (if d=0) or in register f (if d=1).	
Cycles:	1	Affects: C, DC, Z

ANDLW - AND Literal and W

Syntax:	ANDLW k	
Operation:	(k&W) → W	
Description:	And the contents of the W register and the literal k. The result is stored in the W register.	
Cycles:	1	Affects: Z

ANDWF - AND W with F

Syntax:	ANDWF f,d	
Operation:	(W & f) → f,d	
Description:	And the contents of the W register and register f. The result is stored in the W register (if d=0) or in register f (if d=1).	
Cycles:	1	Affects: Z

BCF - Bit Clear F

Syntax:	BCF f,b	
Operation:	0 → f(b)	
Description:	Bit b in register f is cleared	
Cycles:	1	Affects: -

BSF - Bit Set F

Syntax:	BSF f,b	
Operation:	1 → f(b)	
Description:	Bit b in register f is set	
Cycles:	1	Affects: -

BTFSC - Bit Test F, Skip if Clear

Syntax:	BTFSC f,b	
Operation:	skip if f(b)=0	
Description:	If bit b in register f is 0 then the next instruction is skipped	
Cycles:	1 (2 - if skip)	Affects: -

BTFSS - Bit Test F, Skip if Set

Syntax:	BTFSS f,b
Operation:	skip if f(b)=1
Description:	If bit b in register f is 1 then the next instruction is skipped
Cycles:	1 (2 - if skip) Affects: -

CALL - subroutine CALL

Syntax:	CALL k
Operation:	PC+1 → TOS; k → PC<7:0>; 0 → PC<8>; PA2,PA1,PA0 → PC<11:9>
Description:	Return address (PC+1) is pushed in stack. Constant k is loaded into lower eight PC bits. The upper PC bits are loaded from program page pre-select bits. Thus, the program continues from the new address.
Cycles:	2 Affects: -

CLRF - CLeaR F

Syntax:	CLRF f,d
Operation:	00h → f
Description:	Register f is cleared
Cycles:	1 Affects: Z

CLRW - CLeaR W

Syntax:	CLRW
Operation:	00h → W
Description:	The W register is cleared
Cycles:	1 Affects: Z

CLRWDT - CLear WatchDog Timer

Syntax:	CLRWDT
Operation:	00h → WDT, 0 → prescaler
Description:	The watchdog timer and the prescaler is reset.
Cycles:	1 Affects: 1 → TO, 1 → PD

COMF - COMplement F

Syntax:	COMF f,d
Operation:	/f → d
Description:	The contents of register f are complemented. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: Z

DECF - DECrement F

Syntax:	DECF f,d
Operation:	(f-1) → d
Description:	Register f is decremented. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: Z

DECFSZ - DECrement F and Skip if Zero

Syntax:	DECFSZ f,d
Operation:	(f-1) → d, skip if the result is zero
Description:	Register f is decremented. The result is stored in the W register (if d=0) or in register f (if d=1). If the result is zero, the next instruction is skipped.
Cycles:	1 (2 - if skip) Affects: Z

GOTO - GO TO address (unconditional jump)

Syntax:	GOTO k
Operation:	k → PC<8:0>, PA2,PA1,PA0 → PC<11:9>
Description:	The nine bit constant is loaded into lower part of PC. The upper PC bits are loaded from program page pre-select bits in the STATUS register. Thus, the program continues from the new address.
Cycles:	2 Affects: -

INCF - INCrement F

Syntax:	INCF f,d
Operation:	(f+1) → d
Description:	Register f is incremented. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: Z

INCFSZ - INCrement F and Skip if Zero

Syntax:	INCFSZ f,d
Operation:	(f+1) → d, skip if the result is zero
Description:	Register f is incremented. The result is stored in the W register (if d=0) or in register f (if d=1). If the result is zero, the next instruction is skipped.
Cycles:	1(2 - if skip) Affects: Z

IORLW - Inclusive OR Literal with W

Syntax:	IORLW k
Operation:	(W .or. k) → W
Description:	Or the contents of the W register and the literal k. The result is stored in the W register.
Cycles:	1 Affects: Z

IORWF - Inclusive OR W with F

Syntax:	IORWF f,d
Operation:	(W .or. f) $\rightarrow$ f,d
Description:	Or the contents of the W register and register f. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: Z

MOVF - MOVe F

Syntax:	MOVF f,d
Operation:	(f) $\rightarrow$ d
Description:	The contents of register f are moved into the W register (if d=0) or back into register f (if d=1)
Cycles:	1 Affects: Z

MOVWF - MOVe W to F

Syntax:	MOVWF f,d
Operation:	W $\rightarrow$ d
Description:	The contents of the W register is moved to the W register.
Cycles:	1 Affects: -

NOP - No OPeration

Syntax:	NOP
Operation:	No operation
Description:	No operation
Cycles:	1 Affects: -

OPTION - load OPTION register

Syntax:	OPTION
Operation:	W $\rightarrow$ OPTION
Description:	The W register is copied to the OPTION register
Cycles:	1 Affects: -

RETLW - RETurn Literal to W

Syntax:	RETLW k
Operation:	k $\rightarrow$ W, TOS $\rightarrow$ PC
Description:	Return from subroutine. PC is popped from the stack and the literal k is loaded into the W register.
Cycles:	1 Affects: -

RLF - Rotate Left F through carry

Syntax:	RLF f,d
Operation:	f<n> $\rightarrow$ d<n+1>, f<7> $\rightarrow$ C, C $\rightarrow$ d<0>
Description:	The contents of register f are rotated one bit to the left through the carry bit. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: -

RRF - Rotate Right F through carry

Syntax:	RRF f,d
Operation:	f<n> $\rightarrow$ d<n-1>, f<0> $\rightarrow$ C, C $\rightarrow$ d<7>
Description:	The contents of register f are rotated one bit to the right through the carry bit. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: -

SLEEP - SLEEP

Syntax:	SLEEP
Operation:	0 $\rightarrow$ PD, 1 $\rightarrow$ TO, 00h $\rightarrow$ WDT, 0 $\rightarrow$ prescaler
Description:	Reset the ‘power-down’ and set ‘time-out’ bits in the STATUS register and reset the watchdog timer and the prescaler. The processor is put into the SLEEP mode. The oscillator is stopped.
Cycles:	1 Affects: TO, PD

SUBWF - SUBtract W from F

Syntax:	SUBWF f,d
Operation:	(f-W) $\rightarrow$ d
Description:	Subtract the W register from register f. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: C, DC,Z

SWAPF - SWAP F

Syntax:	SWAPF f,d
Operation:	f<0:3> $\rightarrow$ d<4:7>, f<4:7> $\rightarrow$ d<0:3>
Description:	The upper and lower nibbles of register f are exchanged. The result is stored in the W register (if d=0) or in register f (if d=1).
Cycles:	1 Affects: -

TRIS - load TRIS register

Syntax:	TRIS f	
Operation:	W TRIS register f	
Description:	The contents of the W register is stored to the specified output driver control register (f=5, 6, or 7)	
Cycles:	1	Affects: -

XORLW - Exclusive OR Literal with W

Syntax:	XORLW k	
Operation:	(W .xor. k) $\rightarrow$ W	
Description:	Xor the contents of the W register and the literal k. The result is stored in the W register.	
Cycles:	1	Affects: Z

XORWF - Exclusive OR W with F

Syntax:	XORWF f,d	
Operation:	(W .xor. f) $\rightarrow$ d	
Description:	Xor the contents of the W register and register f. The result is stored in the W register (if d=0) or in register f (if d=1).	
Cycles:	1	Affects: Z

4.5.6 PIC16Cxx Instruction Set Summary

PIC16Cxx instruction set is a superset of the PIC16C5x one. Thus, only additional instructions and the instructions which differ from above are described as follows:

ADDLW - ADD Literal and W

Syntax:	ADDLW k	
Operation:	(k+W) $\rightarrow$ W	
Description:	Add the contents of the W register and the literal k. The result is stored in the W register.	
Cycles:	1	Affects: Z

CALL - subroutine CALL

Syntax:	CALL k	
Operation:	PC+1 $\rightarrow$ TOS; k $\rightarrow$ PC<10:0>; PCLATH<4:3> $\rightarrow$ PC<12:11>	
Description:	Return address (PC+1) is pushed in the stack. The 11-bit constant k is loaded into lower eleven PC bits. The upper PC bits are loaded from the PCLATH register. Thus, program continues from the new address.	
Cycles:	2	Affects: -

GOTO - GO TO address (unconditional jump)

Syntax:	GOTO k
Operation:	k → PC<10:0>, PCLATH<4:3> → PC<12:11>
Description:	The eleven bit constant is loaded into lower part of PC. The upper PC bits are loaded from the PCLATH (f3) register. Thus, the program continues from the new address.
Cycles:	2 Affects: -

RETFIE - RETurn From IntErrupt

Syntax:	RETFIE
Operation:	TOS → PC, 1 → GIE
Description:	Return from interrupt. PC is popped from the stack and interrupt is enabled by setting the GIE flag (Global Interrupt Enable).
Cycles:	2 Affects: -

RETURN - RETURN from subroutine

Syntax:	RETURN
Operation:	TOS → PC
Description:	Return from interrupt. PC is popped from the stack.
Cycles:	2 Affects: -

SUBLW - SUB Literal and W

Syntax:	SUBLW k
Operation:	(k-W) → W
Description:	Subtract the contents of the W register from the literal k. The result is stored in the W register.
Cycles:	1 Affects: Z

4.6 Compiler Directives

Compiler directives are intended for compilation parameters setting. These are not compiled into processor binary code. The only exception are the DB, DW and TABLE instructions, which store data to program memory. Some directives (e.g. directives for case sensitivity, number of lines per page of listing, disabling creation of specified output files, ...) are set in dialog windows of the integrated environment. Others (e.g. including a file, setting address for compilation, conditional assembly, macro definition, ...) are intended to be written directly into the program source text. These directives are discussed in this chapter. Most of the directives allow to be entered in two ways (see directive syntax). In case of using the '#' character (an equivalent of syntax of directives used by higher programming languages, e.g. the C one), it is not permitted to define a label in this line.

4.6.1 Directive Overview

BANK	selection of data memory register bank
BIT	bit or array of bit data type definition
BYTE	byte or array of byte data type definition
CONST BYTE	constant or array of constant definition
DB	store data into the program memory
DW	store data into the program memory
ELSE	conditional compilation switch
END	end of program
ENDIF	end of conditional compilation
ENDM	end of macro definition
EQU	assign a value to a symbol
IF	begin of conditional compilation
INCLUDE	include a file
LOCAL	local label definition within a macro
MACRO	begin of macro definition
ORG	setting of an address for compilation
PRAGMA	setting of compiler parameters (disable warning generation)
SET	assign a value to a symbol
TABLE	table of value definition

4.6.2 BANK Directive

The BANK directive selects a default data memory bank. If you define data types (e.g. bit or byte) without a bank specification, the default bank will be used. If no BANK directive is used at all, the bank 0 is assumed.

```
Syntax:     #BANK   <bank number>

Example:    #BANK   0   ; sets default bank 0
```

4.6.3 BIT Directive

The BIT directive is intended for definition of bit or bit array data types. The definition consists of a symbol name, the keyword BIT (in case of bit array BIT[]) and a unique bit specification (register address, bit position, bank). In case of bit array it defines the first component of the array. If a bank is omitted, the default bank is used. If a bit position is omitted, bit 0 (LSB) is assumed. If a bank is specified, the bit position must not be omitted. In case of a bit array, the array size (number of components of the array) must be specified. Components of the array are numbered (starting from 0) by an index in square brackets following the array name. If no index is specified, zero is assumed. The array components (bits) are allocated from the specified position upwards to MSB and higher addresses (see data types).

Syntax:
```
<symbol> BIT  @<address> {<bit>, {<bank>}}
<symbol> BIT  [<size>] @<address> {<bit>, {<bank>}}
```

Example:
```
littlebit  BIT @8,1     ; bit 1 in register 8 array1
BIT[3] @8,2             ; 3-bit array
```

4.6.4 BYTE Directive

The BYTE directive is intended for definition of byte or byte array data types. Definition consists of a symbol name, the keyword BYTE (BYTE[] in case of a byte array) and unique byte specification (register address, bank). In case of byte array it defines the first component (byte) of the array. If a bank is omitted, the default bank is used. In case of byte array, the array size (number of components of the array) must be specified. Components of the array are numbered (starting from 0) by an index in square brackets following the array name. If no index is specified, 0 is assumed. The array components (bytes) are allocated from the specified position upwards to higher addresses (see data types).

Syntax:
```
<symbol> BYTE  @<address>, {<bank>}
<symbol> BYTE  [<size>] @<address>, {<bank>}}
```

Example:
```
byte1   BYTE @9     ; bit 1 in register 8 array1
BYTE[3] @.10        ; 2-byte array
```

4.6.5 CONST BYTE Directive

The CONST BYTE directive defines a constant (similar to the EQU and SET directives) or an array of constant (CONST BYTE[]). The definition consists from a symbol name, the keyword CONST BYTE (CONST BYTE[] in case of a byte array) and a value. The value can be specified as numeric constant, text constant or an expression. In case of an array, the array size can be omitted because it is automatically evaluated from the number of specified values. Components of the array (bytes) are numbered (starting from 0) by an index in square brackets following the array name. If no index is specified, 0 is assumed.

Syntax:
```
<symbol> CONST BYTE = <value>
<symbol> CONST BYTE[{size}] = <values>
```

Example:
```
const1   CONST BYTE = 10h
const2   CONST BYTE[] = ‘abcd’, 0x20, const1
```

4.6.6 DB Directive

The DB directive stores an 8-bit literal (byte) into the program memory on current address. The value can be specified by a numeric constant, text constant, array of constant or an expression.

```
Syntax:     {<label>} DB  <value> {<value>, ...}

Example:    ; storing a text into program memory starting 100h
        ORG   100h
        DB    "program version 1.0"
```

4.6.7 DW Directive

The DW directive stores a constant (ranging up to the instruction width of the specific processor family) into current address in program memory. The value can be specified by a numeric constant, text constant, array of constants or an expression.

```
Syntax:     {<label>}  DW  <value>, {<value>, ...}

Example:    ; specifies configuration fuses of 16C84
            ORG  2007h
            DW   0xXXXX
```

4.6.8 END Directive

The END directive defines an end of the program. All lines followed this directive are ignored by the compiler.

```
Syntax1:    {<label>} END
Syntax2:    #END

Example:    END
```

4.6.9 EQU Directive

The EQU directive defines a symbol value. The value can be specified as a numeric constant or an expression. If the specified symbol already exists, the compiler outputs an error.

```
Syntax:     <symbol>  EQU  <value>

Example:    const1    EQU  20+6*2
            const2    EQU  const1*3
```

4.6.10 IF - ELSE - ENDIF Directives

The IF - ELSE - ENDIF directives define a block of conditional assembly. The definition begins with the IF directive followed by an expression which can be evaluated during compilation. Thus, only constants and previously defined symbols can be used. One of two alternatives comes after evaluation:

1. The expression is evaluated as true (the result is a number different from zero). Then, only <code TRUE> is compiled while <code FALSE> is ignored by the compiler.
2. The expression is evaluated as false (the result is zero). Then, only <code FALSE> is compiled while <code TRUE> is ignored by the compiler.

The ELSE separates the <code TRUE> from <code FALSE>. The ENDIF directive ends the conditionally compiled block.

```
Syntax: #IF  <expression>       or: IF <expression>
            <code TRUE>                 <code TRUE>
        #ELSE                       ELSE
            <code FALSE>                <code FALSE>
        #ENDIF                      ENDIF
```

4.6.11 INCLUDE Directive

The INCLUDE directive inserts a file in program source text. It is usually used for including symbol definitions, macros, libraries and so on. The name of the file to be included is an operand of the directive. The name must be enclosed with parenthesis. Up to nine levels of nesting is permitted.

```
Syntax1:    {<label>} INCLUDE "<file name>"
Syntax2:    #INCLUDE "<file name>"

Example:    INCLUDE "PICREG.EQU"
```

4.6.12 MACRO - LOCAL - ENDM Directives

The MACRO - LOCAL - ENDM directives define macroinstructions. The macroinstruction is a block of instructions which usually makes an operational unit (e.g. a delay generator) or a part which is often used in the program. Macro begins with the definition, where a macro name and arguments are specified. The arguments are intended for parsing values to instructions of the macro body. Up to 10 arguments are permitted. Macro can call another macro, but only up to five levels of nesting is permitted. The LOCAL directive declares that the specified data elements are to be considered in local text to the macro. Macro ends with the ENDM directive.

```
Macro definition:

Syntax1:    #MACRO  <macro name> {<arguments>}
            {#LOCAL <loc.l.>}
            <instructions>
            #ENDM

Syntax2:    <macro name> MACRO  {<arguments>}
            {LOCAL <loc.l.>}
            <instructions>
            #ENDM
```

Example:

```
; macro my1 with two arguments par_a and par_b which stores a
; constant (represented by the par_a parameter) into the data
; register (represented by the par_b parameter):
#MACRO my1 par_a par_b
    Movlw par_a
    Movwf par_b,F
#ENDM
```

Macro call:

```
Syntax:     {<label>} <macro name> {<arguments>}
```

Then the code:

```
            my1 11h, 10h
```

will be expanded to:

```
            Mowlw 11h
            Movwf 10h,F
```

The compiler allows automatic using of specified macro library for source text compilation. In the directory where files of integrated environment reside is the STDPIC.MCR file with built-in basic macroinstructions to maintain compatibility with other products. This file is read before the compilation and then all macros included in it are defined during compilation and available for a user. User can add his own macros there, but it is recommended to create another macro file and include it by the INCLUDE directive into the source text.

4.6.13 ORG Directive

The ORG directive sets current address of the program memory. Then, next commands will be compiled from the specified memory location. This allows to set any address of the program memory. In case of the 16C84 processor, it is allowed to write data directly into the data EEPROM memory which is mapped from address 2100h.

```
Syntax:     {<label>} ORG <value>

Example:    ORG 1FFh        ; 16C54 reset vector Goto start
            ORG 0 start
```

4.6.14 PRAGMA Directive

The PRAGMA directive is intended to set various parameters of the assembler. At present only the optional disable of compiler warning generation is implemented. This directive does not affect warnings if macro libraries is not found. The warning is always generated in this case.

```
Syntax:     #PRAGMA warn-
```

4.6.15 SET Directive

The SET directive defines (like the EQU directive) a symbol value. The value can be specified as a numeric constant or an expression. You can change the assignment of the value defined by the SET directive as often as you like throughout the program by another SET or EQU directive.

```
Syntax:     <symbol>  SET  <value>

Example:    const1    SET  20+6*2
            const2    SET  const1*3  ; pre-defining a value
```

4.6.16 TABLE Directive

The TABLE directive defines a table of values in the program memory. During compilation, this directive expands using the RETLW instructions so that the return codes correspond to the table. The values can be specified as numeric constants, text constants or expressions.

```
Syntax1:    {<label>}  TABLE  <values>
Syntax2:    #TABLE <values>
```

```
Example:    ADDWF 2,1
            TABLE 0xB7, 0x83, 0x11, 0x50
```

The code will be expanded into the following instructions:

```
            ADDWF 2,1
            RETLW 0xB7
            RETLW 0x83
            RETLW 0x11
            RETLW 0x50
```

4.7 Error Messages and Warnings of the Compiler

Can not open input file "<file name>"

Specific file was not found. Make sure that the file exists.

Can not open include file "<file name>"

The INCLUDE directive attempts to insert a file which can not be opened. Make sure whether the file name (and the path, if there is any) specification is correct without a typing error and the file really exists.

Can not open output file "<file name>"

Specified file can not be created and saved. Either the disc is full or write-protected. In case of a computer network, perhaps you have not enough rights to access the current directory.

Can not close file "<file name>"

The specified file was open but can not be closed. Perhaps severe failure of the recording medium occurred or there is not space enough to save the file.

Can not compile this line

The compiler does not understand the line of the source text. Check the syntax of the instruction or directive.

Maximum number of included files or macros exceeded

Maximum number of included files was exceeded. Maybe the file is included into itself in a recurrent way and it leads to runaway. Check included files in all included files.

Unmatched parenthesis

A number of left parenthesis does not match a number of right parenthesis in an expression. Check the syntax of the expression.

Can not evaluate expression '<expression>'

The expression can not be evaluated. Check a syntax of the expression, mathematical symbols and constants.

Can not add new label '<label>' to symbol table

Multiple definition of the label in a single program. Rename the label. In case of the label within a macro, you can define this label as local (the LOCAL directive) or rename it in the macro definition.

Can not add new symbol '<symbol>' to symbol table

Multiple definition of the symbol. Rename the symbol.

Unknown data type

Unknown data type.

Unknown data type or expression

Unknown data type or expression.

Code placed on address '<address>' replaced by another one

Program was overwritten on address '<address>'.

Keyword 'const' must be followed by the keyword 'byte'

The 'CONST' keyword must be followed by the 'BYTE' keyword.

Insufficient memory to add symbol '<symbol>' to symbol table

The memory available to symbols is full. That is why the symbol can not be added to the symbol table. Reduce a number of unused symbols if there are any.

Duplicate definition of symbol '<symbol>'

An attempt to change a value of the symbol.

Illegal bank number

Illegal number of data memory bank is specified.

Address out of data memory

Register address exceeds the data memory of selected processor

Command allowed for 16Cxx family only

This instruction is defined for the 16Cxx family only

Address out of program memory

The address exceeds the program memory.

Bit number should be in range 0-7. Number truncated!

A number of bits must be from 0 to 7. The number is truncated to three lower bits.

File register number must be 0-#. Number truncated!

A register number must be from 0 to X. The number is truncated.

File register number must be 0-47.

Number truncated! A register number must be from 0 to 47. The number is truncated.

Call is possible to low 256 bytes of page only

In the 16C5x family it is allowed to call only subroutines beginning in the lower half-page of the program memory.

Literal value truncated to 8 bits

A constant higher than 256 is truncated to 8 bits.

Unknown data type

Unknown data type.

Can not evaluate address

The compiler can not evaluate an address of a data type. Check whether the address specification has the initial “@” character.

Missing address

No address is specified for a data type.

Can not evaluate bit number

An illegal bit number is specified for a data type of bit or bit array. Check the syntax and typing error.

Can not evaluate bank number

An illegal bank number is specified for a data type. Check the syntax and typing error.

Bad constant syntax, ‘=’ expected

The syntax of CONST BYTE data type definition is illegal. The ‘=’ character followed by a value is expected.

Illegal hexadecimal number

The syntax of a hexadecimal number is illegal. The number must begin with a digit, not a character. For example, the expression FFh is illegal, the right one is 0FFh.

‘]’ expected

The right square bracket to close an index of an array is missing.

Syntax error

The syntax of a command or directive is illegal.

Bad number of real macro parameters. Check macro definition

A number of parameters parsing to a macro does not match the number of parameters specified in the macro definition.

Directive ‘<directive>’ is not supported

The specific directive is not implemented in the compiler.

Unimplemented register

The specific register number is out of the range of the data RAM.

Label is not allowed here

It is not permitted to place a label in this line.

Missing symbol name

A symbol name is missing.

Can not open macro file

The macro definition file can not be opened. Does it really exist?

Can not close macro file

The macro definition file can not be closed.

Duplicate macro definition '<macro>'

The specific macro has already been defined. Rename the macro.

Number of macro parameters exceeds limit

A macro can only have up to 10 parameters. Number of specific parameters exceeds this limit.

Missing #endm directive

A macro definition is not terminated.

Missing macro name

A macro name is not specified in the macro definition.

Number of macro local labels exceeds limit

The limit number of local labels in a macro (up to 10) is exceeded.

Number of macros exceeds limit

Number of macros exceeds the specific limit.

Missing operand

An operand of an instruction or directive is missing.

Invalid TRIS argument

An operand of the TRIS instruction is illegal.

Jump to different page of program memory

Jump to an address out of the current page of the program memory.

This is a *Demo* version

The demo version of the compiler is installed. The restriction is in the maximal number of lines to be compiled.

Too many errors & warnings

There are too many errors in the program source text.

Unknown error

An unknown error of the compiler.

Notes: