

BASCOM-AVR



Version 1.0.0.8

Index

Installation »Page 6
 The BASCOM IDE »Page 17
 Running BASCOM-AVR »Page 17
 File New »Page 17
 File Open »Page 17
 File Close »Page 18
 File Save »Page 18
 File Save As »Page 18
 File Print Preview »Page 18
 File Print »Page 19
 File Exit »Page 19

 Edit Undo »Page 19
 Edit Redo »Page 19
 Edit Cut »Page 19
 Edit Copy »Page 19
 Edit Paste »Page 20
 Edit Find »Page 20
 Edit Find Next »Page 20
 Edit Replace »Page 20
 Edit Goto »Page 20
 Edit Toggle Bookmark »Page 20
 Edit Goto Bookmark »Page 21
 Edit Indent Block »Page 21
 Edit Unindent Block »Page 21

 Program Compile »Page 21
 Program Syntax Check »Page 22
 Program Show Result »Page 22
 Program Simulate »Page 23

Program Send to Chip »Page 23

 Tools Terminal Emulator »Page 25
 Tools LCD Designer »Page 26

 Options Compiler »Page 27
 Options Compiler Chip »Page 27
 Options Compiler Output »Page 28
 Options Compiler Communication »Page 29
 Options Compiler I2C,SPI,1WIRE »Page 30
 Options Compiler LCD »Page 31

 Options Communication »Page 32
 Options Environment »Page 33
 Options Simulator »Page 35
 Options Programmer »Page 36
 Editor Keys »Page 39
 BASCOM Developing Order »Page 40
 BASCOM and Memory »Page 40
 BASCOM Error codes »Page 42
 BASCOM and Hardware
 Additional Hardware »Page 44
 AVR Internal Hardware »Page 45
 AVR Internal Hardware TIMER0 »Page 48
 AVR Internal Hardware TIMER1 »Page 49
 AVR Internal Hardware Watchdog timer »Page 50
 AVR Internal Hardware PORT B »Page 50
 AVR Internal Hardware PORT D »Page 52
 AVR Internal Registers »Page 46
 Attaching an LCD display »Page 53
 Using the I2C protocol »Page 54
 Using the 1 Wire protocol »Page 55
 Using the SPI protocol »Page 55
 Power Up »Page 55
 Language Fundamentals »Page 56
 Reserved Words »Page 56
 BASCOM Language Reference
 \$ASM »Page 64
 \$BAUD »Page 64
 \$CRYSTAL »Page 65
 \$DATA »Page 65
 \$DEFAULT »Page 66
 \$EEPROM »Page 67
 \$EXTERNAL »Page 68
 \$INCLUDE »Page 69
 \$LCD »Page 69
 \$LCDRS »Page 72
 \$LCDPUTCTRL »Page 70
 \$LCDPUTDATA »Page 71
 \$LIB »Page 73
 \$REGFILE »Page 75
 \$SERIALINPUT »Page 75
 \$SERIALINPUT2LCD »Page 77
 \$SERIALOUTPUT »Page 78
 \$XRAMSIZE »Page 78
 \$XRAMSTART »Page 79

1WRESET »Page 80
1WREAD »Page 81
1WWRITE »Page 82
ABS »Page 84
ALIAS »Page 83
ASC »Page 85
BAUD »Page 85
BCD »Page 86
BITWAIT »Page 87
BYVAL »Page 88
CALL »Page 88
CHR »Page 90
CLS »Page 91
CLOCKDIVISION »Page 91
CLOSE »Page 92
CONFIG »Page 93
CONFIG TIMER0 »Page 101
CONFIG TIMER1 »Page 103
CONFIG LCD »Page 97
CONFIG LCDBUS »Page 97
CONFIG LCDMODE »Page 98
CONFIG 1WIRE »Page 94
CONFIG SDA »Page 99
CONFIG SCL »Page 100
CONFIG DEBOUNCE »Page 94
CONFIG SPI »Page 101
CONFIG LCDPIN »Page 99
CONFIG WATCHDOG »Page 106
CONFIG PORT »Page 107
COUNTER0 AND COUNTER1 »Page 109
CONST »Page 122
CRYSTAL »Page 111
CPEEK »Page 110
CURSOR »Page 112
DATA »Page 112
DEBOUNCE »Page 114
DECR »Page 116
DECLARE FUNCTION »Page 116
DECLARE SUB »Page 118
DEFXXX »Page 119
DEFLCDCHAR »Page 119
DELAY »Page 120
DIM »Page 120
DISABLE »Page 123
DISPLAY »Page 125
DO-LOOP »Page 125
ELSE »Page 126
ENABLE »Page 126
END »Page 127
EXIT »Page 128
FOR-NEXT »Page 128
FOURTHLINE »Page 129
FUSING »Page 130
GETADC »Page 130
GETRC0 »Page 132
GETRC5 »Page 133

GOSUB »Page 135
GOTO »Page 136
HEX »Page 137
HEXVAL »Page 137
HIGH »Page 138
HOME »Page 138
I2CRECEIVE »Page 139
I2CSEND »Page 139
I2CSTART,I2CSTOP,I2CRBYTE,I2CWBYTE »Page 140
IDLE »Page 141
IF-THEN-ELSE-END IF »Page 142
INCR »Page 143
INKEY »Page 143
INP »Page 144
INPUTBIN »Page 144
INPUTHEX »Page 145
INPUT »Page 146
INSTR »Page 200
LCD »Page 147
LEFT »Page 150
LEN »Page 150
LOAD »Page 151
LOADADR »Page 191
LOCAL »Page 152
LOCATE »Page 154
LOOKUP »Page 154
LOOKUPSTR »Page 155
LOW »Page 156
LOWERLINE »Page 156
LTRIM »Page 151
MAKEBCD »Page 157
MAKEDEC »Page 158
MAKEINT »Page 157
MID »Page 158
ON INTERRUPT »Page 159
ON VALUE »Page 160
OPEN »Page 161
OUT »Page 162
PEEK »Page 163
POKE »Page 163
POWERDOWN »Page 164
PRINT »Page 165
PRINTBIN »Page 166
READ »Page 167
READEEPROM »Page 168
REM »Page 168
RESET »Page 169
RESTORE »Page 170
RETURN »Page 170
RIGHT »Page 171
ROTATE »Page 172
RTRIM »Page 171
SELECT CASE - END SELECT »Page 173
SET »Page 173
SHIFTCURSOR »Page 175
SHIFTIN »Page 175

SHIFTOUT »Page 176
 SHIFLCD »Page 177
 SOUND »Page 177
 SPACE »Page 178
 SPIIN »Page 178
 SPIMOVE »Page 199
 SPIOU »Page 180
 START »Page 180
 STOP »Page 181
 STR »Page 182
 STRING »Page 183
 SUB »Page 184
 SWAP »Page 184
 THIRDLIN »Page 184
 TRIM »Page 185
 UPPERLINE »Page 186
 VAL »Page 186
 VARPTR »Page 187
 WAIT »Page 187
 WAITKEY »Page 188
 WAITMS »Page 188
 WAITUS »Page 189
 WHILE-WEND »Page 189
 WRITEEPROM »Page 190

International Resellers »Page 9

Supported Programmers »Page 193

Assembly Mnemonics »Page 194

Mixing BASIC with assembly »Page 197

If you have questions, remarks or suggestions please let us know.

You can contact us by sending an email to avr@mcselec.com

Our website is at <http://www.mcselec.com>

For info on updates : please read the readme.txt file that is installed into the BASCOM-AVR directory

MCS Electronics may update this documentation without notice.
 Products specification and usage may change accordingly.

MCS Electronics will not be liable for any mis-information or errors found in this document.

All software provided with this product package is provided ' AS IS' without any warranty expressed or implied.

MCS Electronics will not be liable for any damages, costs or loss of profits arising from the usage of this product package.

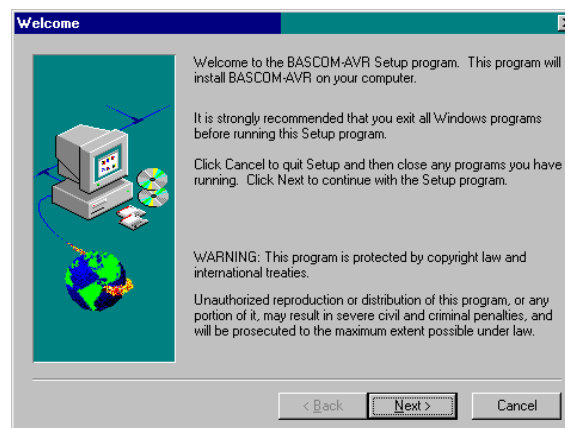
No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without written permission of MCS Electronics.

Copyright MCS Electronics. All rights reserved.

Installation of BASCOM-AVR

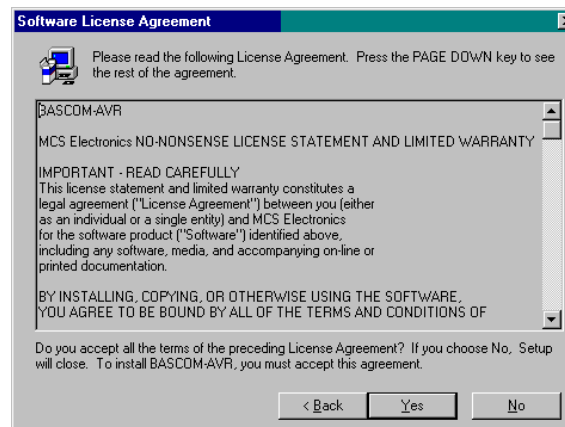
Insert the disk labeled 'disk 1 of 2' and double click the file SETUP.EXE from the Windows explorer.

The following window will appear:



Click on the Next button to continue installation.

The following license info window will appear:

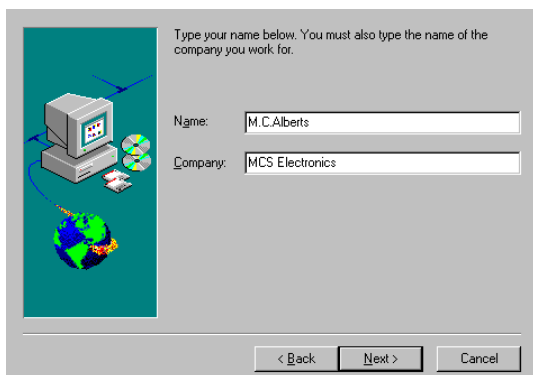


Read the license agreement and click the Yes button when you agree.

A window with additional information is then displayed. This information will be

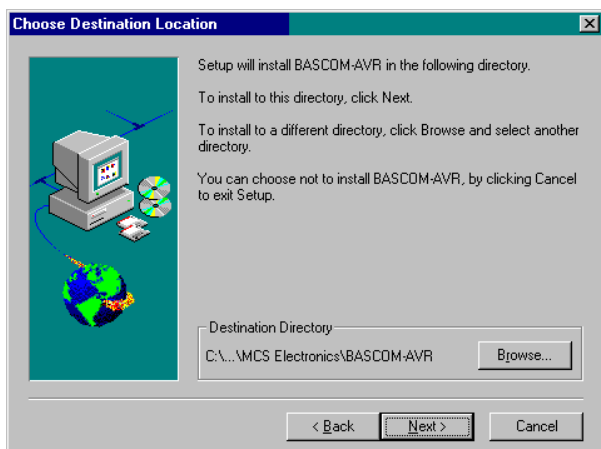
installed as a readme.txt file and contains information on how to get free updates.

After reading the information, click the Next button.
Now the following window appears:



Fill in your name and company name.
Click the Next button to continue.

Now you have the change to select the directory in which BASCOM will be installed.

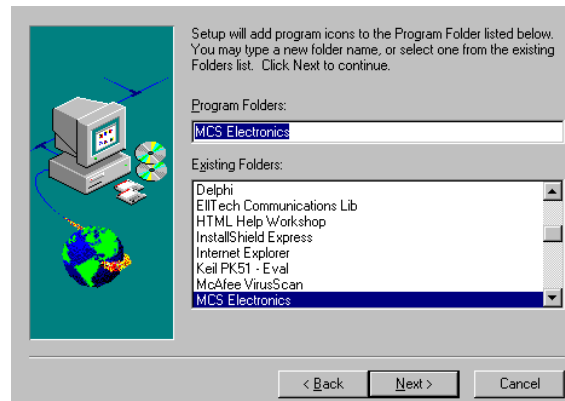


Select the Browse button to change the directory path if required.
By default BASCOM-AVR will be installed into:

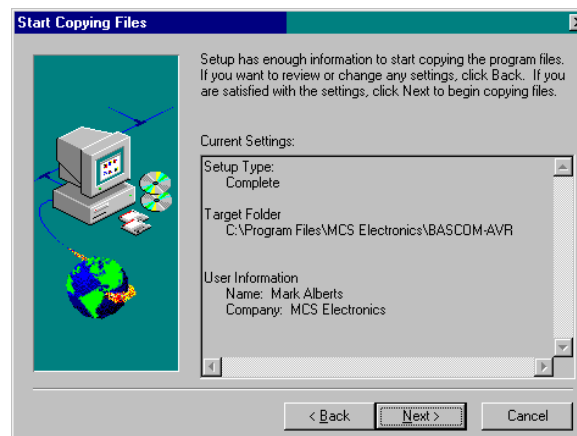
C:\Program Files\MCS Electronics\BASCOM-AVR

After selecting the installation directory, click the Next button.
This time you will be asked in which program group the BASCOM-AVR icon must be placed.

By default, a new program group named MCS Electronics will be made.



After selecting the group, click the Next button to continue.
A summary will be showed. You may go back and change your settings.
Otherwise, click the Next button to complete the installation of BASCOM-AVR.



When the installation is completed you must click the Finish-button, and

restart Windows.

A sub directory named SAMPLES contains all the BASCOSM-AVR sample files.

A sub directory named LIB contains the Library files.

IMPORTANT FOR THE COMMERCIAL VERSION

The license file is not included in the setup. You must copy this file to the **\WINDOWS\SYSTEM** directory.

The license file is named **BSCAVRL.DLL** and can be found on the last installation disk named 'DISK 2 of 2'.

To copy from the Explorer:

Select the file from disk A and drag it into the **\WINDOWS\SYSTEM** directory. When the file is not visible turn the 'Show system Files' option on.

Of course the name of your system directory can be **\W95\SYSTEM** or **\WINNT\SYSTEM** too.

International Resellers

Argentina

DINASTIA SOFT

Oscar H. Gonzalez
Roca 2239 (1714) - Ituzaingo
Buenos Aires
ARGENTINA
Phone: +54-1-4621-0237
Fax: +54-1-4621-0237
Email: dinastiasof@infovia.com.ar
WWW: <http://www.dinastiasoft.com.ar>

Austria

RIBU ELEKTRONIK GMBH

Muehlgasse 18
A-8160 Weiz
AUSTRIA
Phone: 03172-64800
Fax : 03172-64806
Email : office@ribu.at
WWW : <http://www.ribu.at>

Brazil

WF AUTOMAÇÃO INDÚSTRIA COMÉRCIO SERVIÇOS LTDA ME

Miguel Wisintainer
RUA 2 DE SETEMBRO, 733
CEP 89052-000

Australia & USA

DONTRONICS

Don McKenzie
P. O. box 595
Tullamarine 3043
AUSTRALIA
Email : don@dontronics.com
WWW: <http://www.dontronics.com>

Canada

Zanthic Technologies Inc

Steve Letkeman
75 Vintage Meadows Place. S. E.
Medicine Hat, Alberta
T1B4G8 Canada

BLUMENAU.S.C

BRASIL

Email : wf@blusoft.org.br

WWW: <http://www.blusoft.org.br/wf/>

China, Singapore, Malaysia, Croatia, Bosnia, Macedonia Taiwan, Thailand and Hongkong

DIY Electronics (HK) Ltd.

Peter Crowcroft
P. O. Box 88458,
Sham Shui Po, Hong Kong
CHINA
Phone: +85227200255
Fax: +85227250610
Email: peter@kitsrus.com
WWW: <http://kitsrus.com>

Czech & Slovak

LAMIA s.r.o.

Antonin Straka
Porici 20a
Blansko
67801
CZECH REPUBLIC
Phone: +00420-506-418726
Fax: +00420-506-53988

Germany

Elektronikladen Mikrocomputer GmbH

Martin Danne
Wilhelm.-Mellies-Str. 88
D-32758 Detmold
GERMANY
Phone: +495232-8171
Fax: +495232-86197
E-Mail: sales@elektronikladen.de
WWW: www.elektronikladen.de
Vertriebsbüros in Hamburg, Berlin, Leipzig,
Frankfurt, München

Phone: 403-526-8318

Fax: 403-528-9708

Email: zanthic@zanthic.com

WWW: <http://www.zanthic.com>

AX ELEKTRONIKA d.o.o.

Managing director : Jure Mikeln
p. p. 5127
1001 Ljubljana
SLOVENIA
Phone: +386-61-14-914-00, -14-914-05
Fax : +386-61-485-688
Email : jure.mikeln@svet-el.si
WWW: <http://www.svet-el.si>

Germany & Switzerland

Consulting & Distribution

Dr. - Ing. Claus Kuehnel
Muehlenstrasse 9
D-01257 Dresden
GERMANY
Phone: +41.1.785.02.38
Fax: +41.1.785.02.75
Email: info@ckuehnel.ch
WWW: <http://www.ckuehnel.ch>

Hungary

CODIX Ltd, Hungary

Imre Gaspar
Atilla u 1-3,
H-1013 Budapest
HUNGARY
Phone: +3611566330
Fax: +3611564376
Email: codix@mail.matav.hu
WWW: <http://www.hpconline.com/codix>

Italy

Grifo(R)

SalvatoreDamino
Viadell'Artigiano8/6
40016 S.GiorgiodiPianoBO
ITALY
Phone: +39(51)892.052
Fax : +39(51)893.661
Email : tech@grifo.it
WWW: <http://www.grifo.com> (Englisch)
WWW: <http://www.grifo.it> (Italian)

Japan (AVR related)

International Parts & Information Co.,Ltd.

ShujiNonaka
Sengen2-1-6TukubaCity
IbarakiPref.
JAPAN305-0047
Phone: +81-298-50-3113
Fax : +81-298-50-3114
Email sales@ipic.co.jp
WWW <http://www.ipic.co.jp>

Pakistan

ORRIS MICRO SYSTEM

Malik Muhammad Nawaz Awan
15/Y, TARIQBINZIADCOLONY, SAHIWAL.
PAKISTAN
Phone: 0441-66982
Email : oms@brain.net.pk

Poland

RK-SYSTEM

RobertKacprzycki
CHELMONSKIEGO30
05-825GRODZISKMAZ.
POLAND.
Phone: +48227243039
Fax: +48227243037
Email robertk@univcomp.waw.pl

Scandinavia (Sweden, Norway, Denmark)

High Tech Horizon

ChristerJohansson
Asbogatan29C
S-26251Angelholm
SWEDEN
Phone: +46431-410088
Fax : +46431-410088
Email: info@hth.com
WWW: <http://www.hth.com>

Japan (8051 related)

MOMIZI DENSHI

OsamuHamakawa
AOMADANI-NISHI-2-4-8-804
MINO-CITY
JAPAN562-0023
Phone: +81-727-28-7855
Fax: +81-727-28-7855
Email momizi@mb.infoweb.ne.jp

Korea

SAMPLE Electronics Co.

JunghoonKim
306Jeshin43-22ShinkeyYoungsanSeoul
Korea
Postalcode140-090
Phone: 82-2-707-3882
Fax: 82-2-707-3884
Email : info@sample.co.kr
WWW: <http://www.sample.co.kr>

Portugal & Spain

Multidigital, Lda

JoaquimBoavida
P. O. Box 137
4435RioTinto
PORTUGAL
Phone: +351-2-6102217
Fax: +351-2-4862173
Email: info@multidigital.com
WWW: <http://www.multidigital.com>

Sweden

LAWICEL

LarsWictorsson
Klubbagan3
SE-28232TYRINGE
SWEDEN
Phone: +46(0)45159877
Fax : +46(0)45159878
WWW: <http://www.lawicel.com>
Email: info@lawicel.com

Spain

Ibercomp

MiquelZuniga
C/. del Parc, numero 8 (bajos)
E-07014
PalmadeMallorca
SPAIN
Phone: +34(9)71456642
Fax : +34(9)71456758
Email: ibercomp@atlas-iap.es
WWW: <http://www.ibercomp.es>

UK

TECHMAIL SOLUTIONS LTD

DoganIbrahim
14DunveganRoad
Eltham
LondonSE91SA
Phone: 01713435242or01814889689
Fax: 01718216744
Email: dogan@dircon.co.uk
WWW: www.users.dircon.co.uk/~doqan/dogan/

USA

M. Akers Enterprises

MichaelW.Akers
3800VineyardAvenue#E
Pleasanton,CA94566-6734
USA
Phone: +1-925-640-3600
Fax: +1-925-640-3600
Email: info@mwakers.com
WWW: <http://www.mwakers.com>

Turkey

IBD Limited Sirketi

379/1SokakA-BlokNo:2/101
AFASanayiCarsisi-II.Sanayi
35100Bornova-Izmir

Phone: 0090-232-4627477-78
Fax: 0090-232-4627545
E-mail: ibdLtd@superonline.com
E-mail: ibd@ibd-ltd.com
E-mail: sales@ibd-ltd.com
E-mail: faruk@ibd-ltd.com
WWW : www.ibd.com.tr

USA

Techniks, Inc.

FrankCapelle
POBox463
Ringoes,NJ08551
USA
Phone: 908-788-8249
Fax: 908-788-8837
Email: Techniks@techniks.com
WWW: <http://www.techniks.com>

USA

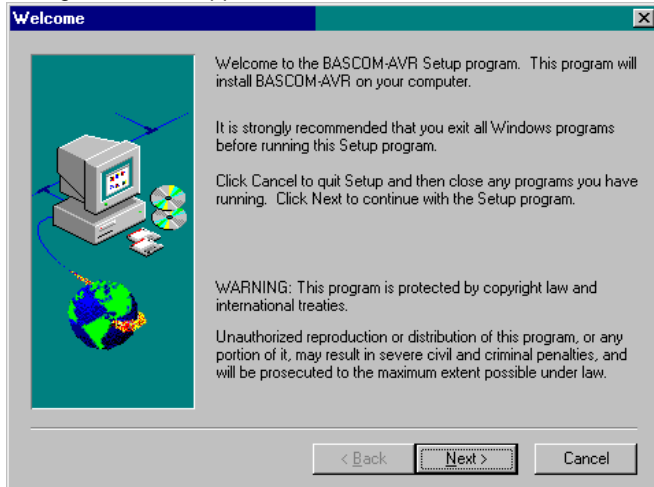
Rhombus

DavidH.Lawrence
1909OldMountainCreekRoad
Greenville,SC29609
USA
Phone: +1-864-233-8330
Fax: +1-864-233-8331
Email: webmaster@rhombusinc.com
WWW: <http://www.rhombusinc.com>

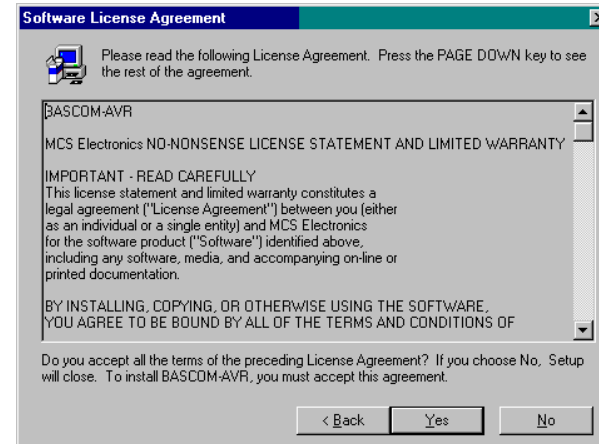
Installation of BASCOM-AVR

Insert the disk labeled 'disk 1 of 2' and double click the file SETUP.EXE from the Windows explorer.

The following window will appear:

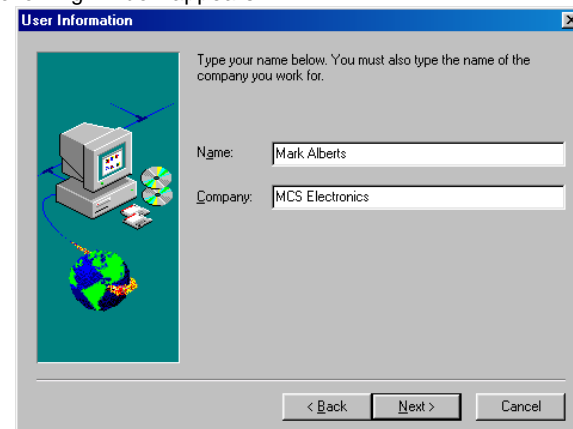


Click on the Next button to continue installation.
The following license info window will appear:



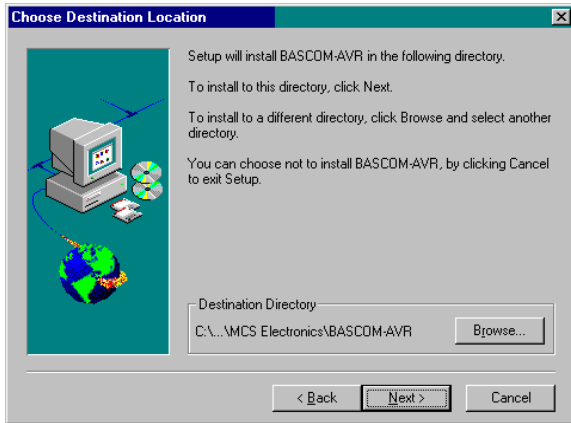
Read the license agreement and click the Yes button when you agree.
A window with additional information is then displayed. This information will be installed as a readme.txt file and contains information on how to get free updates.

After reading the information, click the Next button.
Now the following window appears:



Fill in your name and company name.

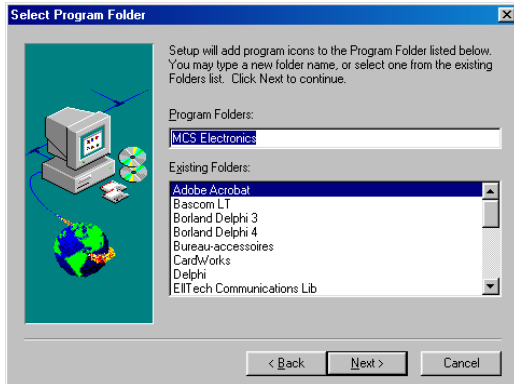
Click the Next button to continue.
Now you have the change to select the directory in which BASCOM will be installed.



Select the Browse button to change the directory path if required.
By default BASCOM-AVR will be installed into:
C:\Program Files\MCS Electronics\BASCOM-AVR
After selecting the installation directory, click the Next button.

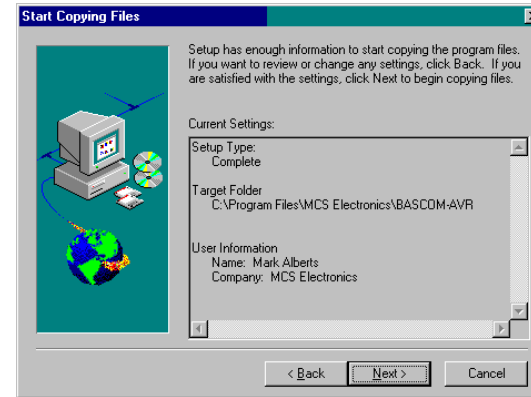
This time you will be asked in which program group the BASCOM-AVR icon must be placed.

By default, a new program group named MCS Electronics will be made.



After selecting the group, click the Next button to continue.

A summary will be showed. You may go back and change your settings. Otherwise, click the Next button to complete the installation of BASCOM-AVR.



When the installation is completed you must click the Finish-button, and restart Windows.

A sub directory named SAMPLES contains all the BASCOM-AVR sample files.

IMPORTANT FOR THE COMMERCIAL VERSION

The license file is not included in the setup. You must copy this file to the **WINDOWS\SYSTEM** directory.
The license file is named **BSCAVRL.DLL** and can be found on the last installation disk named 'DISK 2 of 2'.

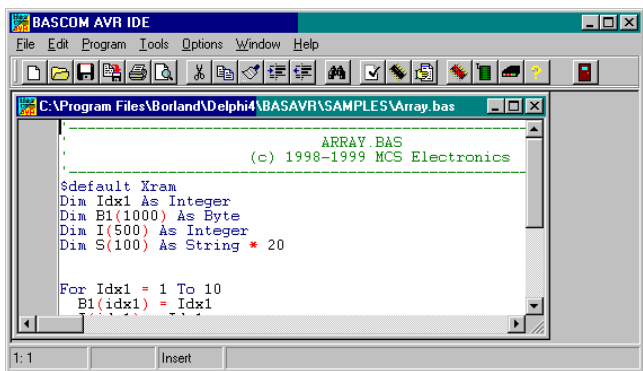
To copy from the Explorer:
Select the file from disk A and drag it into the WINDOWS\SYSTEM directory.

Of course the name of your system directory can be **W95\SYSTEM** or **WINNT\SYSTEM** too.

You also need to **DELETE** the file **windows\system\BASC-AVR.DLL** before you install the commercial version over the DEMO version.

Running BASCOM-AVR

Double-click the BASCOM-AVR icon to run BASCOM.
The following window will appear. (If this is your first run, the edit window will be empty.)




The most-recently opened file will be loaded.

File New

This option creates a new window in which you will write your program.


The focus is set to the new window.

File new shortcut: , CTRL + N

File Open


With this option you can load an existing program from disk.
BASCOSM saves files in standard ASCII format. Therefore, if you want to load a file that was made with another editor be sure that it is saved as an ASCII file.

Note that you can specify that BASCOM must reformat the file when it opens it with the **Options Environment** option. This should only be necessary when loading files made with another editor.

File open shortcut : , CTRL+O

File Close

Close the current program.
When you have made changes to the program, you will be asked to save the program first.

File close shortcut : 

File Save

With this option, you save your current program to disk under the same file name.

If the program was created with the **File New** option, you will be asked to name the file first. Use the **File Save As** option to give the file another name.

Note that the file is saved as an ASCII file.

File save shortcut : , CTRL+S

File Save As

With this option, you can save your current program to disk under a different file name.

Note that the file is saved as an ASCII file.

File save as shortcut : 

File Print Preview

With this option, you can preview the current program before it is printed.


Note that the current program is the program that has the focus.

File print preview shortcut : 

File Print

With this option, you can print the current program.


Note that the current program is the program that has the focus.

File print shortcut :  , CTRL+P

File Exit


With this option, you can leave BASCOM.

If you have made changes to your program, you can save them upon leaving BASCOM.

File exit shortcut : 


Edit Undo

With this option, you can undo the last text manipulation.

Edit Undo shortcut :  , CTRL+Z


Edit Redo

With this option, you can redo the last undo.

Edit Redo shortcut :  , CTRL+SHIFT+Z


Edit Cut

With this option, you can cut selected text into the clipboard.

Edit cut shortcut :  , CTRL+X


Edit Copy

With this option, you can copy selected text into the clipboard.

Edit copy shortcut :  , CTRL+C

Edit Paste

With this option, you can paste text from the clipboard into the current cursor position.

Edit paste shortcut :  , CTRL+V

Edit Find


With this option, you can search for text in your program.

Text at the cursor position will be placed in the find dialog box.

Edit Find shortcut :  , CTRL+F

Edit Find Next

With this option, you can search for the last specified search item.

Edit Find Next shortcut :  , F3

Edit Replace

With this option, you can replace text in your program.

Edit Replace shortcut :  , CTRL+R

Edit Goto

With this option, you can immediately go to a line .

Edit go to line shortcut :  ,CTRL+G

Edit Toggle Bookmark

With this option, you can set/reset a bookmark, so you can jump in your code with the Edit Go to Bookmark option. Shortcut : CTRL+K + x where x can be 1-8

Edit Goto Bookmark

With this option, you can jump to a bookmark. There can be up to 8 bookmarks. Shortcut : CTRL+Q+ x where x can be 1-8

Edit Indent Block

With this option, you can indent a selected block of text.

Edit Indent Block shortcut :  , CTRL+SHIFT+I

Edit Unindent Block

With this option, you can un-indent a block.

Edit Unindent Block shortcut :  , CTRL+SHIFT+U


Program Compile


With this option, you can compile your current program. Your program will be saved automatically before being compiled. The following files will be created depending on the Option Compiler Settings.

File	Description
xxx.BIN	Binary file which can be programmed into the microprocessor
xxx.DBG	Debug file that is needed by the simulator.
xxx.OBJ	Object file for AVR Studio
xxx.HEX	Intel hexadecimal file which is needed by some programmers.
xxx.ERR	Error file. (only when errors are found)
xxx.RPT	Report file.
xxx.EEP	EEPROM image file

If a serious error occurs, you will receive an error message in a dialog box and the compilation will end.

All other errors will be displayed at the bottom above the status bar.

When you click on the line with the error info, you will jump to the line that contains the error. The margin will also display the  sign. At the next compilation, the error window will disappear.

Program compile shortcut :  , F7

Program Syntax Check

With this option, your program is checked for syntax errors. No file will be created except for an error file, if an error is found.

Program syntax check shortcut , CTRL + F7

Program Show Result

Use this option to view the result of the compilation. See the **Options Compiler Output** for specifying which files must be created. The files that can be viewed are report and error.

File show result shortcut :  , CTRL+W

Information provided in the report:


Info	Description
Report	Name of the program
Date and time	The compilation date and time.
Compiler	The version of the compiler.
Processor	The selected target processor.
SRAM	Size of microprocessor SRAM (internal RAM).
EEPROM	Size of microprocessor EEPROM (internal EEPROM).
ROMSIZE	Size of the microprocessor FLASH ROM.
ROMIMAGE	Size of the compiled program.
BAUD	Selected baud rate.
XTAL	Selected XTAL or frequency
BAUD error	The error percentage of the baud rate.
XRAM	Size of external RAM.
Stack start	The location in memory which the hardware stack points to. The HW-stack pointer "grows down".
S-Stacksize	The size of the software stack.
S-Stackstart	The location in memory which the software stack pointer points to. The software stack pointer "grows down".
Framesize	The size of the frame. The frame is used for storing local variables.
Framestart	The location in memory where the frame starts.

- LCD address The address that must be placed on the bus to enable the LCD display E-line.
- LCD RS The address that must be placed on the bus to enable the LCD RS-line
- LCD mode The mode the LCD display is used with. 4 bit mode or 8 bit mode.

Program Simulate

With this option, you can simulate your program.

At this moment there is no internal Simulator. It will be added later. You can simulate your programs with AVR Studio for the time being.

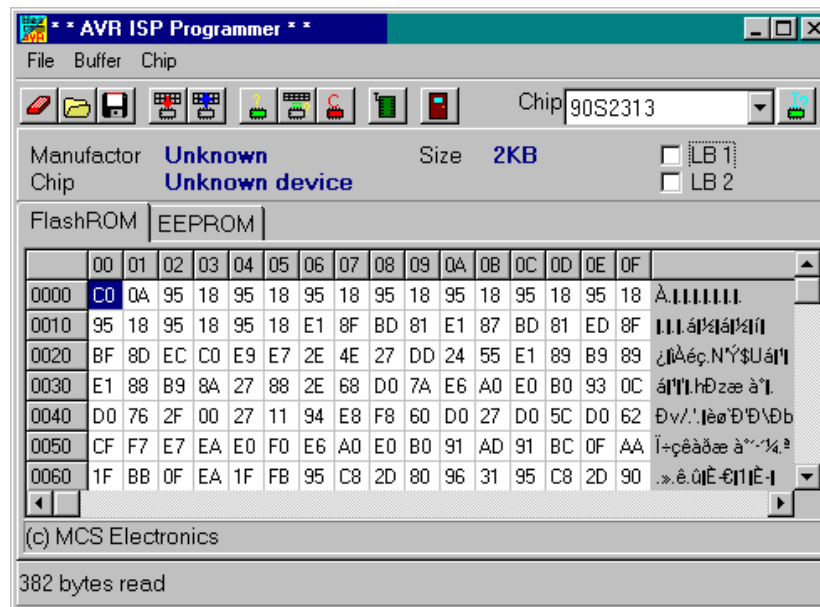
Program Simulate shortcut :  , F2

Program Send to Chip

This option will bring up the selected programmer or will program the chip directly if this option is selected from the Programmer options.

Program send to chip shortcut  , F4

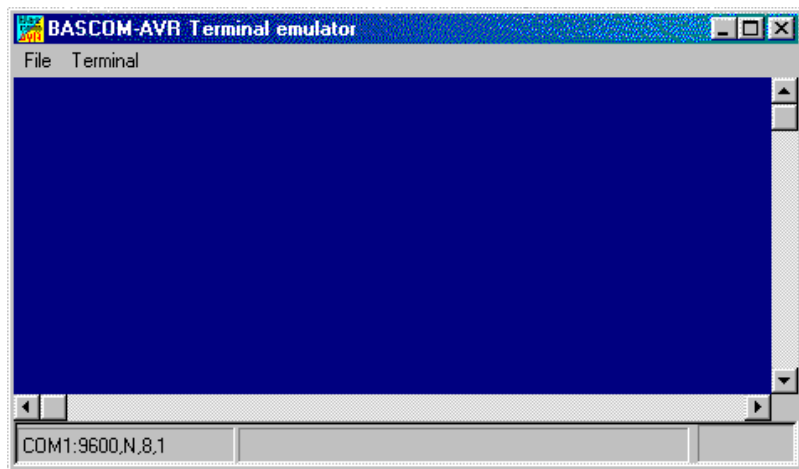
The following window will be shown:



Menuitem	Description
File Exit	Return to editor
Buffer Clear	Clears buffer
Buffer Load from file	Loads a file into the buffer
Buffer Save to file	Saves the buffer content to a file
Chip Identify	Identifies the chip
Write buffer into chip	Programs the buffer into the chip ROM or EEPROM
Read chipcode into buffer	Reads the code or data from the chips code memory or data memory
Chip blank check	Checks if the chip is blank
Chip erase	Erase the content of both the program memory and the data memoty
Chip verify	verifies if the buffer is the same as the chip program or data memory
Chip Set lockbits	Writes the selected lock bits LB1 and/or LB2. Only an erase will reset the lock bits
Chip autoprogram	Erases the chip and programs the chip. After the programming is completed, a verification is performed.
RCEN	Writes a bit to enable the internal oscillator. This RCEN bit is only available on some AVR chips.

Tools Terminal Emulator

With this option you can communicate via the RS-232 interface to the microcomputer. The following window will appear :



Information you type and information that the computer board sends are displayed in the same window.

Note that you must use the same baud rate on both sides of the transmission. If you compiled your program with the Compiler Settings at 4800 baud, you must also set the Communication Settings to 4800 baud. The setting for the baud rate is also reported in the report file.

File Upload

Uploads the current program in HEX format. This option is meant for loading the program into a monitor program.

File Escape

Aborts the upload to the monitor program.

File Exit

Closes terminal emulator.

Terminal Clear

Clears the terminal window.

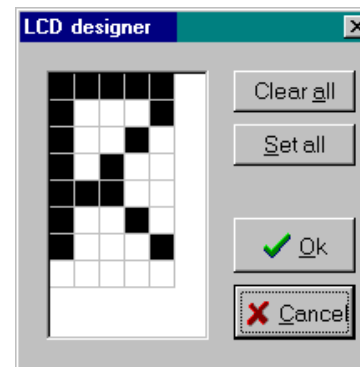
Terminal Open Log

Open or closes a LOG file. When there is no LOG file selected you will be asked to enter or select a filename. All info that is printed to the terminal window is captured into the log file. The menu caption will change into 'Close Log' and when you choose this option the file will be closed.

The terminal emulator has a strange bug that you can't select the menu options by using the keyboard. This is an error in the terminal component and I hope the third party will fix this bug.

Tools LCD Designer

With this option you can design special characters for LCD-displays. The following window will appear:



The LCD-matrix has 7x5 points. The bottom row is reserved for the cursor but can be used.

You can select a point by clicking the left mouse button. If a cell was selected it will be deselected.

Clicking the Set All button will set all points.

Clicking the Clear All button will clear all points.

When you are finished you can press the Ok button : a statement will be inserted in your active program-editor window at the current cursor position. The statement looks like this :

```
Deflcdchar ?,1,2,3,4,5,6,7,8
```

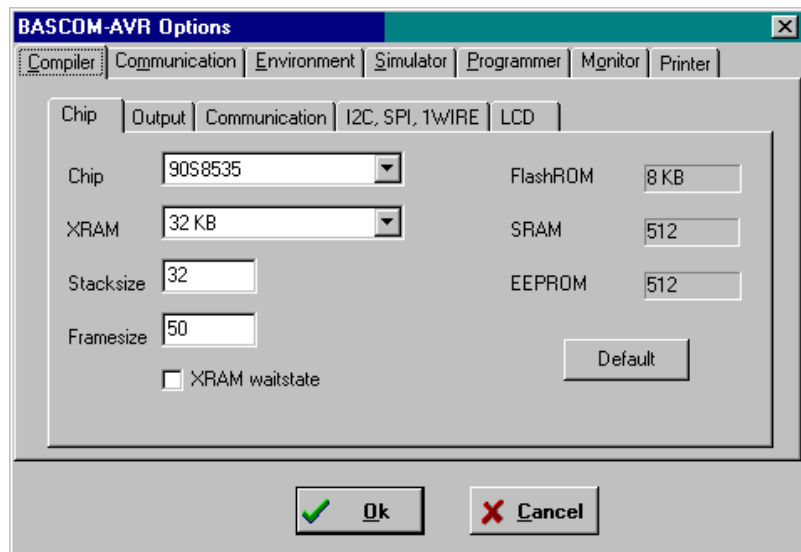
You must replace the ?-sign with a character number ranging from 0-7.

OptionsCompiler

With this option, you can modify the compiler options.
The following TAB pages are available:

- Options Compiler Chip »Page 27
- Options Compiler Output »Page 28
- Options Compiler Communication »Page 29
- Options Compiler I2C , SPI, 1WIRE »Page 30
- Options Compiler LCD »Page 31

OptionsCompilerChip



The following options are available:

Options Compiler Chip

Item	Description
Chip	Selects the target chip. Each chip has a corresponding x.DAT file with specifications of the chip. Note that some DAT files are not available yet.
XRAM	Selects the size of the external RAM.
Stack size	Specifies the size of the software stack.

Each local variable uses 2 bytes. Each variable that is passed in a sub program uses 2 bytes too. So when you have used 10 locals in a SUB and the SUB passes 3 parameters, you need $13 * 2 = 26$ bytes.

Frame size

Specifies the size of the frame.
Each local is stored in a space that is named the frame.
When you have 2 local integers and a string with a length of 10, you need a framesize of $(2*2) + 11 = 15$ bytes.
The internal conversion routines used when you use INPUT num,STR(),VAL() etc, also use the frame. They need a maximum of 12 bytes. So for this example $15+12 = 27$ would be a good value.

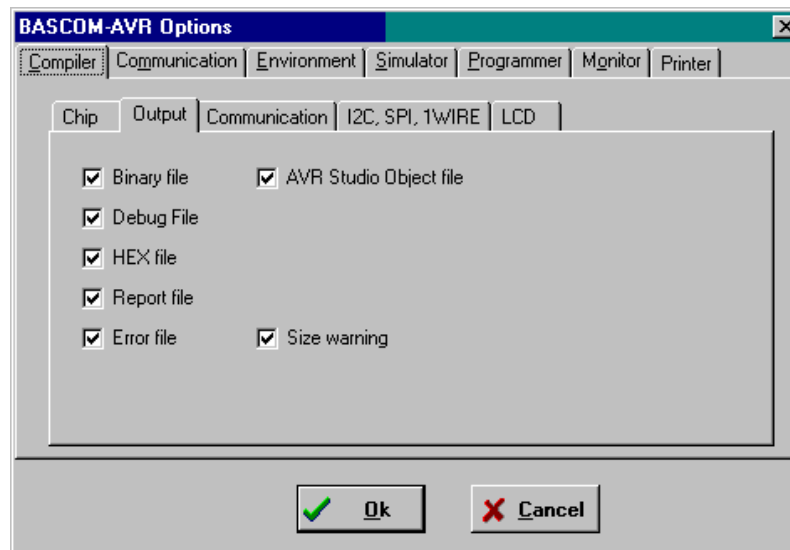
XRAM waitstate

Select to insert a wait state for the external RAM.

Default

Press or click this button to use the current Compiler Chip settings as default for all new projects.

OptionsCompilerOutput

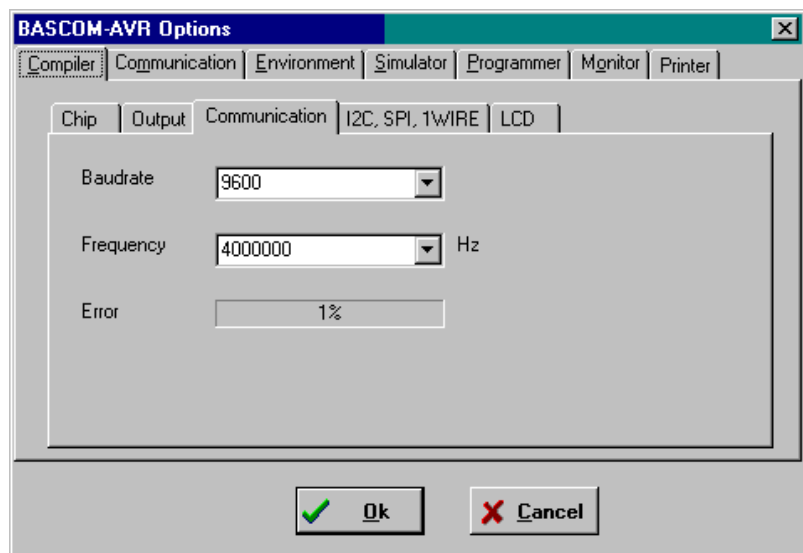


Options Compiler Output

Item	Description
Binary file	Select to generate a binary file. (xxx.bin)
Debug file	Select to generate a debug file (xxx.dbg)

- Hex file Select to generate an Intel HEX file (xxx.hex)
- Report file Select to generate a report file (xxx.rpt)
- Error file Select to generate an error file (xxx.err)
- AVR Studio object file Select to generate an AVR Studio object file (xxx.obj)
- Size warning Select to generate a warning when the code size exceeds the Flash ROM size.

OptionsCompilerCommunication



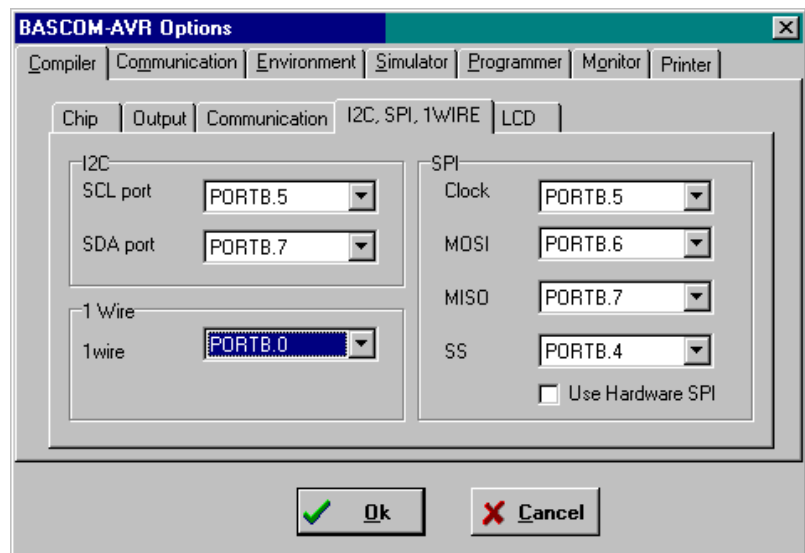
Options Compiler Communication

Item	Description
Baud rate	Selects the baud rate for the serial statements. You can also type in a new baud rate.
Frequency	Select the frequency of the used crystal. You can also type in a new frequency.

The settings for the internal hardware UART are:

- No parity
- 8 data bits
- 1 stop bit

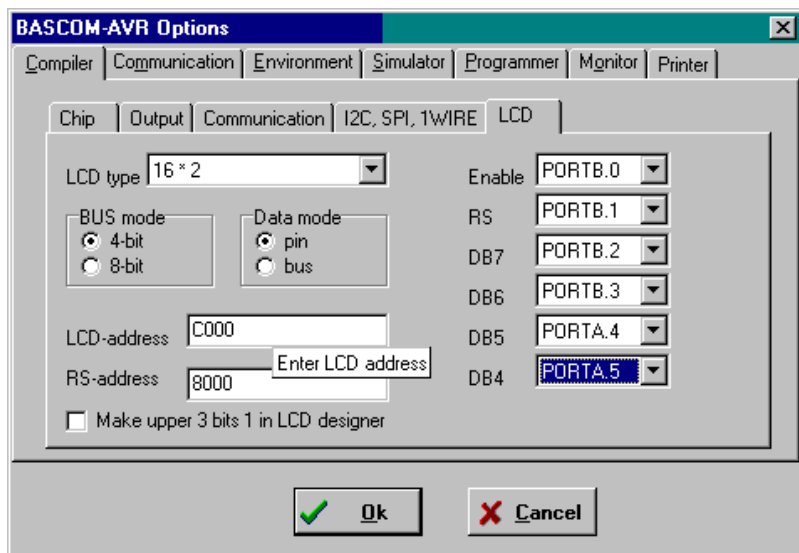
Options Compiler I2C, SPI, 1WIRE



Options Compiler I2C, SPI, 1WIRE

Item	Description
SCL port	Select the port that serves as the SCL-line for the I2C related statements.
SDA port	Select the port that serves as the SDA-line for the I2C related statements.
1WIRE	Select the port that serves as the 1WIRE-line for the 1Wire related statements.
Clock	Select the port that serves as the clock-line for the SPI related statements.
MOSI	Select the port that serves as the MOSI-line for the SPI related statements.
MISO	Select the port that serves as the MISO-line for the SPI related statements.
SS	Select the port that serves as the SS-line for the SPI related statements.
Use hardware SPI	Select to use built-in hardware for SPI, otherwise software emulation of SPI will be used.

Options Compiler LCD

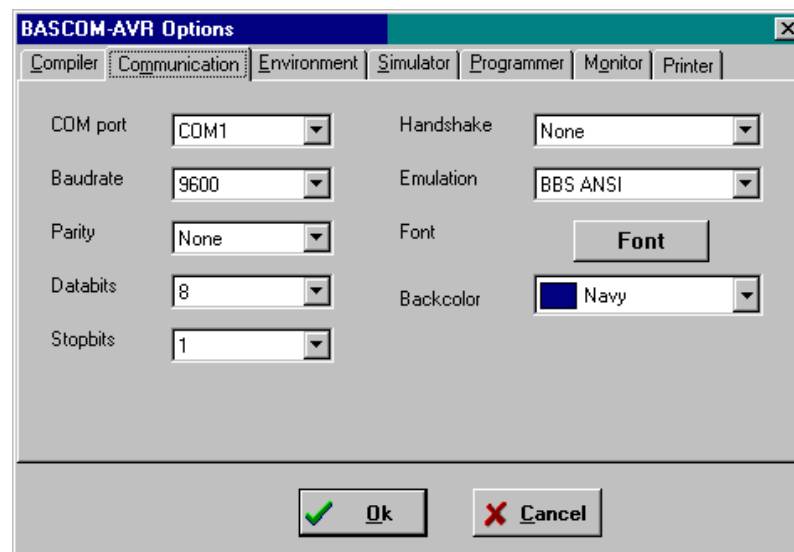


Options Compiler LCD

Item	Description
LCD type	The LCD display used.
Bus mode	The LCD can be operated in BUS mode or in PIN mode. In PIN mode, the data lines of the LCD are connected to the processor pins. In BUS mode the data lines of the LCD are connected to the data lines of the BUS. Select 4 when you have only connect DB4-DB7. When the data mode is 'pin', you should select 4.
Data mode	Select the mode in which the LCD is operating. In PIN mode, individual processor pins can be used to drive the LCD. In BUS mode, the external data bus is used to drive the LCD.
LCD address	In BUS mode you must specify which address will select the enable line of the LCD display. For the STK200, this is C000 = A14 + A15.
RS address	In BUS mode you must specify which address will select the RS line of the LCD display. For the STK200, this is 8000 = A15
Enable	For PIN mode, you must select the processor pin that is connected to the enable line of the LCD display.
RS	For PIN mode, you must select the processor pin that is connected to the RS line of the LCD display.
DB7-DB4	For PIN mode, you must select the processor pins that are connected to the upper four data lines of the LCD display.

Options Communication

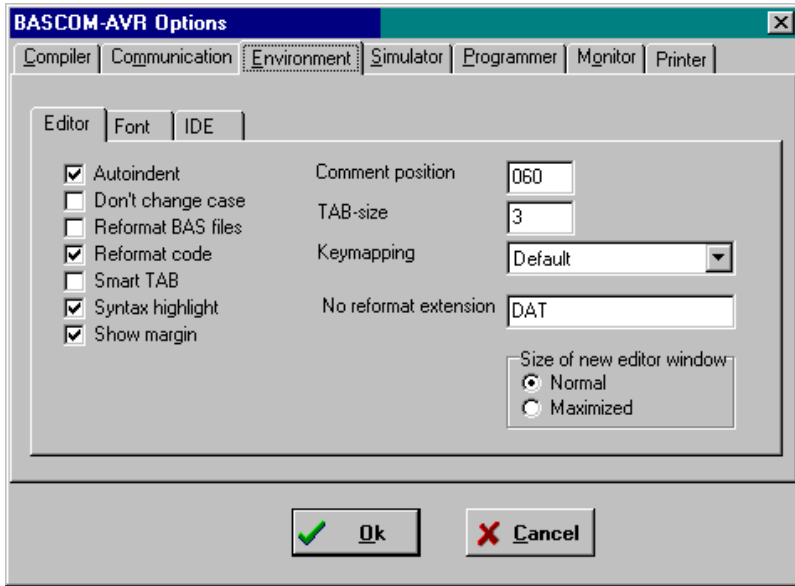
With this option, you can modify the communication settings for the terminal emulator.



Item	Description
Comport	The communication port of your PC that you use for ther terminal emulator.
Baud rate	The baud rate to use.
Parity	Parity, default None.
Data bits	Number of data bits, default 8.
Stop bits	Number of stop bits, default 1.
Handshake	The handshake used, default is none.
Emulation	Emulation used, default BBS ANSI.
Font	Font type and color used by the emulator.
Back color	Background color of the terminal emulator.

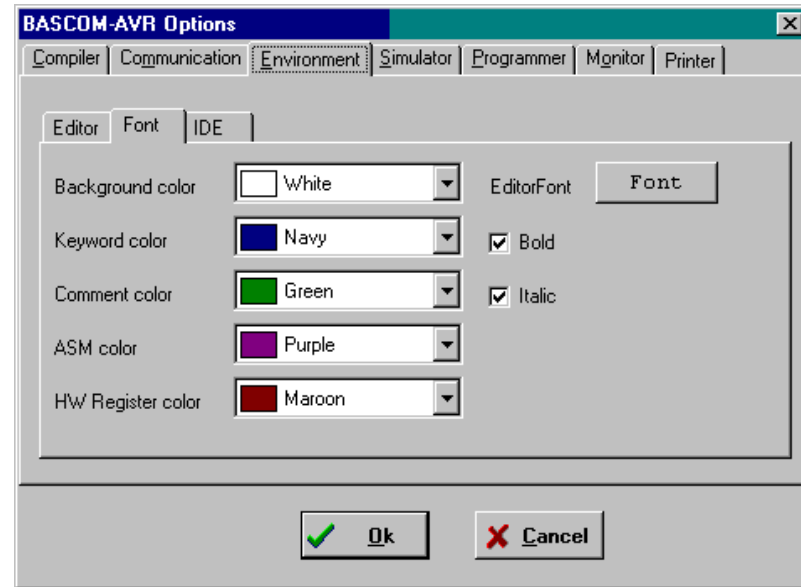
Note that the baud rate of the terminal emulator and the baud rate setting of the compiler options, must be the same in order to work correctly.

OptionsEnvironment

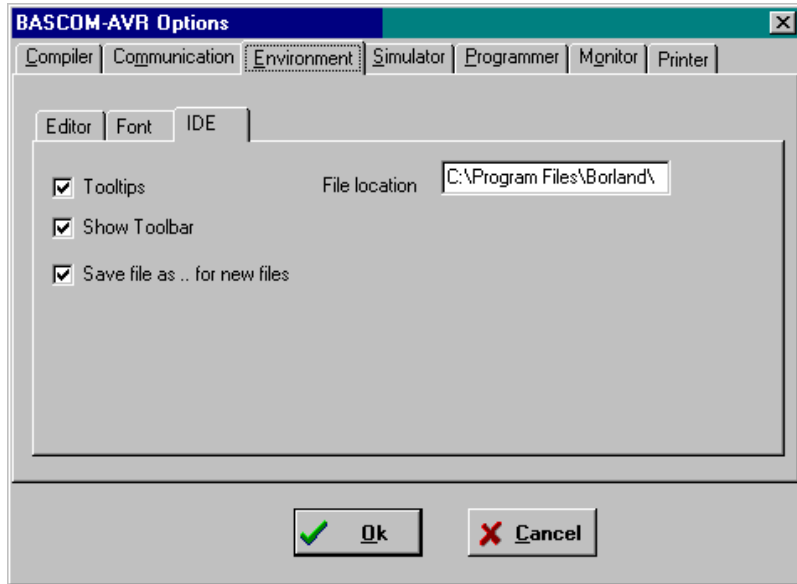


OPTION	DESCRIPTION
Auto Indent	When you press return, the cursor is set to the next line at the current column position
Don't change case	When set, the reformatting won't change the case of the text. Default is that the text is reformatted so every word begins with upper case.
Reformat BAS files	Reformat files when loading them into the editor. This is only necessary when you are loading files that were created with another editor. Normally you won't need to set this option.
Reformat code	Reformat code when entered in the editor.
Smart TAB	When set, a TAB will go to the column where text starts on the previous line.
Syntax highlighting	This options highlights BASCOM statements in the editor.
Show margin	Shows a margin on the right side of the editor.
Comment	The position of the comment. Comment is positioned at the right of your source code.
TAB-size	Number of spaces that are generated for a TAB.
Keypmapping	Choose default, Classic, Brief or Epsilon.
No reformat extension	File extensions separated by a space that will not be reformatted when loaded.

Size of new editor window When a new editor window is created you can select how it will be made. Normal or Maximized (full window)



OPTION	DESCRIPTION
Background color	The background color of the editor window.
Keyword color	The color of the reserved words. Default Navy. The keywords can be displayed in bold too.
Comment color	The color of comment. Default green. Comment can be shown in <i>Italic</i> too.
ASM color	Color to use for ASM statements. Default purple.
HW registers color	The color to use for the hardware registers/ports. Default maroon.
Editor font	Click on this label to select another font for the editor window.



OPTION	DESCRIPTION
Tooltips	Show tooltips.
Show toolbar	Shows the toolbar with the shortcut icons.
Save File As ... for newfiles.	Will display a dialogbox so you can give new files a name when they must be saved. When you dont select this option the default name will be give to the file (nonamex.bas). Where x is a number.
File location	Double click to select a directory where your program files are stored. By default Windows will use the My Documents path.

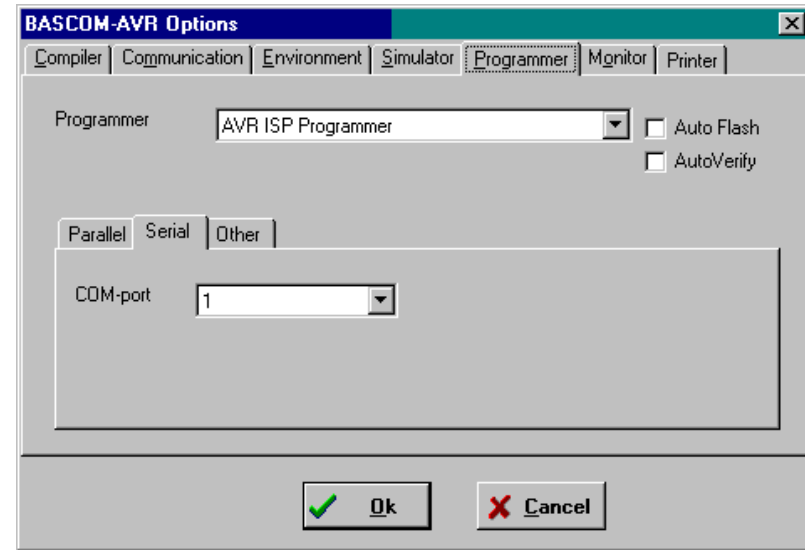
OptionsSimulator

With this option you can modify the simulator settings.

OPTION	DESCRIPTION
Program	The path with the program name of the simulator.
Parameter	The parameter to pass to the program. {FILE}.OBJ will supplie the name of the current program with the extension .OBJ to the simulator.

OptionsProgrammer

With this option you can modify the programmer settings.



OPTION	DESCRIPTION
Programmer	Select one from the list.
Auto flash	Some programmers support auto flash. Pressing F4 will program the chip without showing the programmer window.
Auto verify	Some programmers support verifying. The chip content will be verified after programming.
LPT address	Port address of the LPT that is connected to the programmer.
Send HEX	Only for EPROM Simulator on LPT. Select when a HEX file must be sent instead of the bin file.

OptionsMonitor

With this option you can modify the monitor settings.

OPTION	DESCRIPTION
Upload speed	Selects the baud rate used for uploading

Monitor prefix	String that will be send to the monitor before the upload starts
Monitor suffix	String that us sent to the monitor after the download is completed.
Monitor delay	Time in millions of seconds to wait after a line has been sent to the monitor.
Prefix delay	Time in millions of seconds to wait after a prefix has been sent to the monitor.

OptionsPrinter

With this option you can modify the printer settings.
There are only settings to change the margins of the paper.

OPTION	DESCRIPTION
Left	The left margin.
Right	The right margin.
Top	The top margin.
Bottom	The bottom margin.

WindowCascade

Cascade all open editor windows.

Window Tile

Tile all open editor windows.

Window Arrange Icons

Arrange the icons of the minimized editor windows.

Window Minimize All

Minimize all open editor windows.

Help About

This option shows an about box as showed below.



Your serial number is shown in the about box.
You will need this when you have questions about the product.
The library version is also shown. In this case, it is **1.00**.
You can compare it with the one on our web site in case you need an update.

Click on **Ok** to return to the editor.

Help Index

Shows the BASCOM help file.

When you are in the editor window, the current word will be used as a keyword.

Help on Help

Shows help on how to use the Windows help system.

Help Credits

Shows a form with credits to people I would like to thank for their contributions to BASCOM.

BASCOSM Editor Keys

Key	Action
LEFT ARROW	One character to the left
RIGHT ARROW	One character to the right
UP ARROW	One line up
DOWN ARROW	One line down
HOME	To the beginning of a line
END	To the end of a line
PAGE UP	Up one window
PAGE DOWN	Down one window
CTRL+LEFT	One word to the left
CTRL+RIGHT	One word to the right
CTRL+HOME	To the start of the text
CTRL+END	To the end of the text
CTRL+ Y	Delete current line
INS	Toggles insert/overstrike mode
F1	Help (context sensitive)
F3	Find next text
F4	Send to chip (run flash programmer)
F5	Run
F7	Compile File
F8	Step
F9	Set breakpoint
F10	Run to
CTRL+F7	Syntax Check
CTRL+F	Find text
CTRL+G	Go to line
CTRL+K+x	Toggle bookmark. X can be 1-8
CTRL+L	LCD Designer
CTRL+M	File Simulation
CTRL+N	New File
CTRL+O	Load File
CTRL+P	Print File
CTRL+Q+x	Go to Bookmark. X can be 1-8
CTRL+R	Replace text
CTRL+S	Save File
CTRL+T	Terminal emulator
CTRL+P	Compiler Options

CTRL+W	Show result of compilation
CTRL+X	Cut selected text to clipboard
CTRL+Z	Undo last modification
SHIFT+CTRL+Z	Redo last undo
CTRL+INS	Copy selected text to clipboard
SHIFT+INS	Copy text from clipboard to editor
CTRL+SHIFT+J	Indent Block
CTRL+SHIFT+U	Unindent Block
Select text	Hold the SHIFT key down and use the cursor keys to select text. or keep the left mouse key pressed and tag the cursor over the text to select.

DevelopingOrder

- Start BASCOM;
- Open a file or create a new one;
- Check the chip settings, baud rate and frequency settings for the target system;
- Compile the file;
- If an error occurs fix it and recompile (F7);
- Run the simulator (AVR Studio at the moment);
- Program the chip;

Memoryusage

Every variable uses memory. This memory is also called SRAM. The available memory depends on the chip.

A special kind of memory are the registers in the AVR. Registers 0-31 have addresses 0-31.

Almost all registers are used by the compiler or might be used in the future. Which registers are used depends on the statements you used.

This brings us back to the SRAM.

No SRAM is used by the compiler other than the space needed for the software stack and frame.

Each 8 used bits occupy one byte.

Each byte occupies one byte.

Each integer/word occupies two bytes.

Each Long or Single occupies four bytes.

Each String occupies at least 2 bytes.

A string with a length of 10. occupies 11 bytes. The extra byte is needed to indicate the end of the string.

Use bits or bytes where you can to save memory. (not allowed for negative values)

The software stack is used to store the addresses of LOCAL variables and for variables that are passed to SUB routines.

Each LOCAL variable and passed variable to a SUB, uses two bytes to store the address. So when you have a SUB routine in your program that passes 10 variables, you need $10 * 2 = 20$ bytes. When you use 2 LOCAL variables in the SUB program that receives the 10 variables, you need an additional $2 * 2 = 4$ bytes.

The software stack size can be calculated by taking the maximum number of parameters in a SUB routine, adding the number of LOCAL variables and multiplying the result by 2. To be safe, add 4 more bytes for internally-used LOCAL variables.

LOCAL variables are stored in a place that is named the frame.

When you have a LOCAL STRING with a size of 40 bytes, and a LOCAL LONG, you need $41 + 4$ bytes = 45 bytes of frame space.

The report will show the result of both calculations.

When you use conversion routines such as STR(), VAL() etc. that convert from numeric to string and vice versa, you also need a frame. It should be 16 bytes in that case.

Note that the use of the INPUT statement with a numeric variable, or the use of the PRINT/LCD statement with a numeric variable, will also force you to reserve 16 bytes of frame space. This because these routines use the internal numeric<->string conversion routines.

XRAM

You can easily add external memory to a 8515. Then XRAM will become available.(extended memory).

When you add a 32KB RAM, the first address will be 0.

But because the XRAM can only start after the SRAM, which is &H0260, the lower memory locations of the XRAM will not be used.

ERAM

Most AVR chips have internal EEPROM on board.

This EEPROM can be used to store and retrieve data.

In BASCOM, this dataspace is called ERAM.

An important difference is that an ERAM variable can be written for a maximum of 100.000 times. So only assign an ERAM variable when it is needed and not in a loop.

Constant code usage

Constants are stored in a constant table.

Each used constant in your program will end up in the constant table.

For example:

```
Print "ABCD"
```

```
Print "ABCD"
```

This example will only store one constant (ABCD).

```
Print "ABCD"
```

```
Print "ABC"
```

In this example, two constants will be stored because the strings differ.

Error Codes

The following table lists errors that can occur.

Error	Description
1	Unknown statement
2	Unknown structure EXIT statement
3	WHILE expected
4	No more space for IRAM BIT
5	No more space for BIT
6	. expected in filename
7	IF THEN expected
8	BASIC source file not found
9	Maximum 128 aliases allowed
10	Unknown LCD type
11	INPUT, OUTPUT, 0 or 1 expected
12	Unknown CONFIG parameter
13	CONST already specified
14	Only IRAM bytes supported
15	Wrong data type
16	Unknown Definition
17	9 parameters expected
18	BIT only allowed with IRAM or SRAM
19	STRING length expected (DIM S AS STRING * 12 ,for example)
20	Unknown DATA TYPE
21	Out of IRAM space
22	Out of SRAM space
23	Out of XRAM space
24	Out of EPROM space
25	Variable already dimensioned
26	AS expected
27	parameter expected
28	IF THEN expected
29	SELECT CASE expected
30	BIT's are GLOBAL and can not be erased

31 Invalid data type
 32 Variable not dimensioned
 33 GLOBAL variable can not be ERASED
 34 Invalid number of parameters
 35 3 parameters expected
 36 THEN expected
 37 Invalid comparison operator
 38 Operation not possible on BITS
 39 FOR expected
 40 Variable can not be used with RESET
 41 Variable can not be used with SET
 42 Numeric parameter expected
 43 File not found
 44 2 variables expected
 45 DO expected
 46 Assignment error
 47 UNTIL expected
 50 Value doesn't fit into INTEGER
 51 Value doesn't fit into WORD
 52 Value doesn't fit into LONG
 60 Duplicate label
 61 Label not found
 62 SUB or FUNCTION expected first
 63 Integer or Long expected for ABS()
 64 , expected
 65 device was not OPEN
 66 device already OPENED
 68 channel expected
 70 BAUD rate not possible
 71 Different parameter type passed then declared
 72 Getclass error. This is an internal error.
 73 Printing this FUNCTION not yet supported
 74 3 parameters expected
 80 Code does not fit into target chip
 81 Use HEX(var) instead of PRINTHEX
 82 Use HEX(var) instead of LCDHEX
 85 Unknown interrupt source
 86 Invalid parameter for TIMER configuration
 87 ALIAS already used
 88 0 or 1 expected
 89 Out of range : must be 1-4
 90 Address out of bounds
 91 INPUT, OUTPUT, BINARY, or RANDOM expected
 92 LEFT or RIGHT expected
 93 Variable not dimensioned
 94 Too many bits specified

95 Falling or rising expected for edge
 96 Prescale value must be 1,8,64,256 or 1024
 97 SUB or FUNCTION must be DECLARED first
 98 SET or RESET expected
 99 TYPE expected
 100 No array support for IRAM variables
 101 Can't find HW-register
 102 Error in internal routine
 103 = expected
 104 LoadReg error
 105 StoreBit error
 106 Unknown register
 107 LoadnumValue error
 108 Unknown directive in device file
 109 = expected in include file for .EQU
 110 Include file not found
 111 SUB or FUNCTION not DECLARED
 112 SUB/FUNCTION name expected
 113 SUB/FUNCTION already DECLARED
 114 LOCAL only allowed in SUB or FUNCTION
 115 #channel expected
 116 Invalid register file
 117 Unknown interrupt
 200 .DEF not found
 201 Low Pointer register expected
 202 .EQU not found, probably using functions that are not supported by the selected chip
 203 Error in LD or LDD statement
 204 Error in ST or STD statement
 205 } expected
 10000 DEMO/BETA only supports 1024 bytes of code

AdditionalHardware

Of course just running a program on the chip is not enough. You will probably attach all kind of electronics to the processor ports.

BASC0M supports a lot of hardware and so has lots of hardware related statements.

Before explaining about programming the additional hardware, it might be better to talk about the chip.

The AVR internal hardware »Page 45

Attaching an LCD display »Page 53
 Using the I2C protocol »Page 54
 Using the 1WIRE protocol »Page 55
 Using the SPI protocol »Page 55

You can attach additional hardware to the ports of the microprocessor.
 The following statements will become available:

I2CSEND »Page 139 and I2CRECEIVE »Page 139 and other I2C related statements.
 CLS, »Page 91 LCD, »Page 147 DISPLAY »Page 125 and other related LCD-statements.

1WRESET »Page 80 , 1WWRITE »Page 82 and 1WREAD »Page 81

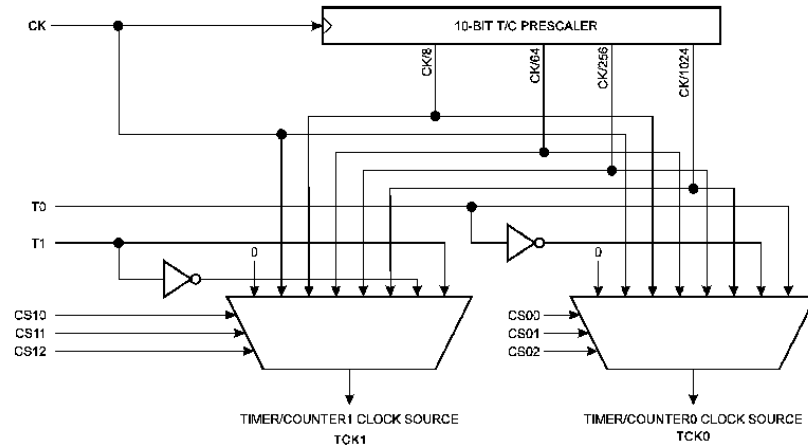
AVR Internal Hardware

The AVR chips all have internal hardware that can be used.
 For the description we have used the 8515 so some described hardware will not be available when you select a 2313 for example.

Timer/Counters

The AT90S8515 provides two general purpose Timer/Counters - one 8-bit T/C and one 16-bit T/C. The Timer/Counters have individual pre-scaling selection from the same 10-bit pre-scaling timer. Both Timer/Counters can either be used as a timer with an internal clock time base or as a counter with an external pin connection which triggers the counting.

Prescaler for Timer/Counter0 and 1



More about TIMERO »Page 48
 More about TIMER1 »Page 49

The WATCHDOG Timer. »Page 50

Almost all AVR chips have the ports B and D. The 40 pin devices also have ports A and C that also can be used for addressing an external RAM chip. Since all ports are identical but the PORT B and PORT D have alternative functions, only these ports are described.

PORT B »Page 50

PORT D »Page 52

AVR Internal Registers

You can manipulate the register values directly from BASIC. They are also reserved words. The internal registers for the AVR90S8515 are :

Addr.	Register
\$3F	SREG I T H S V N Z C
\$3E	SPH SP15 SP14 SP13 SP12 SP11 SP10 SP9 SP8
\$3D	SPL SP7 SP6 SP5 SP4 SP3 SP2 SP1 SP0
\$3C	Reserved
\$3B	GIMSK INT1 INTO - - - - -
\$3A	GIFR INTF1 INTF0
\$39	TIMSK TOIE1 OCIE1A OCIE1B - TICIE1 - TOIE0 -
\$38	TIFR TOV1 OCF1A OCF1B -ICF1 -TOV0 -

\$37	Reserved
\$36	Reserved
\$35	MCUCR SRE SRW SE SM ISC11 ISC10 ISC01 ISC00
\$34	Reserved
\$33	TCCR0 - - - - CS02 CS01 CS00
\$32	TCNT0 Timer/Counter0 (8 Bit)
\$31	Reserved
\$30	Reserved
\$2F	TCCR1A COM1A1 COM1A0 COM1B1 COM1B0 - -PWM11 PWM10
\$2E	TCCR1B ICNC1 ICES1 - - CTC1 CS12 CS11 CS10
\$2D	TCNT1H Timer/Counter1 - Counter Register High Byte
\$2C	TCNT1L Timer/Counter1 - Counter Register Low Byte
\$2B	OCR1AH Timer/Counter1 - Output Compare Register A High Byte
\$2A	OCR1AL Timer/Counter1 - Output Compare Register A Low Byte
\$29	OCR1BH Timer/Counter1 - Output Compare Register B High Byte
\$28	OCR1BL Timer/Counter1 - Output Compare Register B Low Byte
\$27	Reserved
\$26	Reserved
\$25	ICR1H Timer/Counter1 - Input Capture Register High Byte
\$24	ICR1L Timer/Counter1 - Input Capture Register Low Byte
\$23	Reserved
\$22	Reserved
\$21	WDTCSR - - - WDTOE WDE WDP2 WDP1 WDP0
\$20	Reserved
\$1F	Reserved - - - - - EEAR8
\$1E	EEARL EEPROM Address Register Low Byte
\$1D	EEDR EEPROM Data Register
\$1C	EECR - - - - - EEMWE EEWE EERE
\$1B	PORTA PORTA7 PORTA6 PORTA5 PORTA4 PORTA3 PORTA2 PORTA1 PORTA0
\$1A	DDRA DDA7 DDA6 DDA5 DDA4 DDA3 DDA2 DDA1 DDA0
\$19	PINA PINA7 PINA6 PINA5 PINA4 PINA3 PINA2 PINA1 PINA0
\$18	PORTB PORTB7 PORTB6 PORTB5 PORTB4 PORTB3 PORTB2 PORTB1 PORTB0
\$17	DDRB DDB7 DDB6 DDB5 DDB4 DDB3 DDB2 DDB1 DDB0
\$16	PINB PINB7 PINB6 PINB5 PINB4 PINB3 PINB2 PINB1 PINB0
\$15	PORTC PORTC7 PORTC6 PORTC5 PORTC4 PORTC3 PORTC2 PORTC1 PORTC0
\$14	DDRC DDC7 DDC6 DDC5 DDC4 DDC3 DDC2 DDC1 DDC0
\$13	PINC PINC7 PINC6 PINC5 PINC4 PINC3 PINC2 PINC1 PINC0
\$12	PORTD PORTD7 PORTD6 PORTD5 PORTD4 PORTD3 PORTD2 PORTD1 PORTD0
\$11	DDRD DDD7 DDD6 DDD5 DDD4 DDD3 DDD2 DDD1 DDD0
\$10	PIND PIND7 PIND6 PIND5 PIND4 PIND3 PIND2 PIND1 PIND0
\$0F	SPDR SPI Data Register
\$0E	SPSR SPIF WCOL - - - - -
\$0D	SPCR SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
\$0C	UDR UART I/O Data Register
\$0B	USR RXC TXC UDRE FE OR - - -
\$0A	UCR RXCIE TXCIE UDRIE RXEN TXEN CHR9 RXB8 TXB8
\$09	UBRR UART Baud Rate Register
\$08	ACSR ACD - ACO AC1 ACIE ACIC ACIS1 ACIS0
\$00	Reserved

The registers and their addresses are defined in the xxx.DAT files which are placed in the BASCOM-AVR application directory.

The registers can be used as normal byte variables.
 PORTB = 40 will place a value of 40 into port B.

Note that internal registers are reserved words. This means that they can't be dimensioned as BASCOM variables!

So you can't use the statement **DIM SREG As Byte** because **SREG** is an internal register.
 You can however manipulate the register with the **SREG = value** statement.

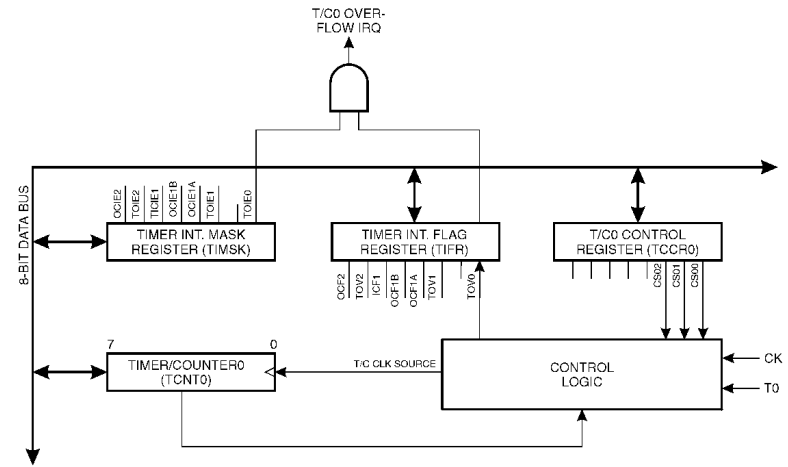
AVR Internal Hardware TIMERO

The 8-Bit Timer/Counter0

The 8-bit Timer/Counter0 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped.
 The overflow status flag is found in the Timer/Counter Interrupt Flag Register - TIFR. Control signals are found in the Timer/Counter0 Control Register - TCCR0. The interrupt enable/disable settings for Timer/Counter0 are found in the Timer/Counter Interrupt Mask Register - TIMSK.

When Timer/Counter0 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one internal CPU clock period. The external clock signal is sampled on the rising edge of the internal CPU clock.

Timer/Counter0 Block Diagram



The 8-bit Timer/Counter0 features both a high resolution and a high accuracy usage with the lower pre-scaling opportunities. Similarly, the high pre-scaling opportunities make the Timer/Counter0 useful for lower speed functions or exact timing functions with infrequent actions.

AVR Internal Hardware TIMER1

The 16-Bit Timer/Counter1 (8515 other timers may be different)

The 16-bit Timer/Counter1 can select clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped. The different status flags (overflow, compare match and capture event) and control signals are found in the Timer/Counter1 Control Registers - TCCR1A and TCCR1B.

The interrupt enable/disable settings for Timer/Counter1 are found in the Timer/Counter Interrupt Mask Register - TIMSK.

When Timer/Counter1 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one internal CPU clock period. The external clock signal is sampled on the rising edge of the internal CPU clock.

The 16-bit Timer/Counter1 features both a high resolution and a high accuracy usage with the lower prescaling opportunities. Similarly, the high prescaling opportunities make the Timer/Counter1 useful for lower speed functions or exact timing functions with infrequent actions.

The Timer/Counter1 supports two Output Compare functions using the Output Compare Register 1 A and B - OCR1A and OCR1B as the data sources to be compared to the Timer/Counter1 contents.

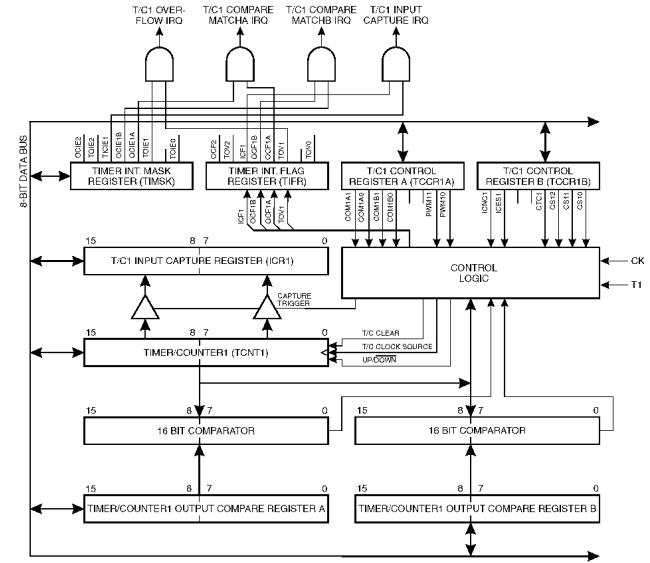
The Output Compare functions include optional clearing of the counter on compareA match, and actions on the Output Compare pins on both compare matches.

Timer/Counter1 can also be used as a 8, 9 or 10-bit Pulse With Modulator. In this mode the counter and the OCR1A/OCR1B registers serve as a dual glitch-free stand-alone PWM with centered pulses.

The Input Capture function of Timer/Counter1 provides a capture of the Timer/Counter1 contents to the Input Capture Register - ICR1, triggered by an external event on the Input Capture Pin - ICP. The actual capture event

settings are defined by the Timer/Counter1 Control Register -TCCR1B. In addition, the Analog Comparator can be set to trigger the Input Capture.

Timer/Counter1 Block Diagram



AVR Internal Hardware Watchdog timer

The Watchdog Timer

The Watchdog Timer is clocked from a separate on-chip oscillator which runs at 1MHz. This is the typical value at VCC = 5V.

By controlling the Watchdog Timer pre-scaler, the Watchdog reset interval can be adjusted from 16K to 2,048K cycles (nominally 16 - 2048 ms). The RESET WATCHDOG - instruction resets the Watchdog Timer.

Eight different clock cycle periods can be selected to determine the reset period.

If the reset period expires without another Watchdog reset, the AT90Sxxxx resets and executes from the reset vector.

AVR Internal Hardware Port B

Port B

Port B is an 8-bit bi-directional I/O port. Three data memory address locations are allocated for the Port B, one each for the Data Register - PORTB, \$18(\$38), Data Direction Register - DDRB, \$17(\$37) and the Port B Input Pins - PINB, \$16(\$36). The Port B Input Pins address is read only, while the Data Register and the Data Direction Register are read/write.

All port pins have individually selectable pull-up resistors. The Port B output buffers can sink 20mA and thus drive LED displays directly. When pins PB0 to PB7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated.

The Port B pins with alternate functions are shown in the following table:

When the pins are used for the alternate function the DDRB and PORTB register has to be set according to the alternate function description.

Port B Pins Alternate Functions

Port	Pin	Alternate Functions
PORTB.0	T0	(Timer/Counter 0 external counter input)
PORTB.1	T1	(Timer/Counter 1 external counter input)
PORTB.2	AIN0	(Analog comparator positive input)
PORTB.3	AIN1	(Analog comparator negative input)
PORTB.4	SS	(SPI Slave Select input)
PORTB.5	MOSI	(SPI Bus Master Output/Slave Input)
PORTB.6	MISO	(SPI Bus Master Input/Slave Output)
PORTB.7	SCK	(SPI Bus Serial Clock)

The Port B Input Pins address - PINB - is not a register, and this address enables access to the physical value on each Port B pin. When reading PORTB, the PORTB Data Latch is read, and when reading PINB, the logical values present on the pins are read.

PortB As General Digital I/O

All 8 bits in port B are equal when used as digital I/O pins. PORTB.X, General I/O pin: The DDBn bit in the DDRB register selects the direction of this pin, if DDBn is set (one), PBn is configured as an output pin. If DDBn is cleared (zero), PBn is configured as an input pin. If PORTBn is set (one) when the pin configured as an input pin, the MOS pull up resistor is activated.

To switch the pull up resistor off, the PORTBn has to be cleared (zero) or the pin has to be configured as an output pin.

DDBn Effects on Port B Pins

DDBn	PORTBn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	PBn will source current if ext. pulled low.

1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	Push-Pull One Output

AVR Internal Hardware Port D**Port D***Port D Pins Alternate Functions*

Port	Pin	Alternate Function
PORTD.0	RDX	(UART Input line)
PORTD.1	TDX	(UART Output line)
PORTD.2	INT0	(External interrupt 0 input)
PORTD.3	INT1	(External interrupt 1 input)
PORTD.5	OC1A	(Timer/Counter1 Output compareA match output)
PORTD.6	WR	(Write strobe to external memory)
PORTD.7	RD	(Read strobe to external memory)

RD - PORTD, Bit 7

RD is the external data memory read control strobe.

WR - PORTD, Bit 6

WR is the external data memory write control strobe.

OC1- PORTD, Bit 5

Output compare match output: The PD5 pin can serve as an external output when the Timer/Counter1 compare matches.

The PD5 pin has to be configured as an out-put (DDD5 set (one)) to serve this function. See the Timer/Counter1 description for further details, and how to enable the output. The OC1 pin is also the output pin for the PWM mode timer function.

INT1 - PORTD, Bit 3

External Interrupt source 1: The PD3 pin can serve as an external interrupt source to the MCU. See the interrupt description for further details, and how to enable the source

INT0 - PORTD, Bit 2

INT0, External Interrupt source 0: The PD2 pin can serve as an external interrupt source to the MCU. See the interrupt description for further details, and how to enable the source.

TXD - PORTD, Bit 1

Transmit Data (Data output pin for the UART). When the UART transmitter is enabled, this pin is configured as an output regardless of the value of DDRD1.

RXD - PORTD, Bit 0

Receive Data (Data input pin for the UART). When the UART receiver is enabled this pin is configured as an output regardless of the value of DDRD0. When the UART forces this pin to be an input, a logical one in PORTD0 will turn on the internal pull-up.

When pins TXD and RXD are not used for RS-232 they can be used as an input or output pin.

No PRINT, INPUT or other RS-232 statement may be used in that case.

The UCR register will by default not set bits 3 and 4 that enable the TXD and RXD pins for RS-232 communication. It is however reported that this not works for all chips. In this case you must clear the bits in the UCR register with the following statements:

```
RESET UCR.3
RESET UCR.4
```

Attaching an LCD Display

A LCD display can be connected with two methods.

- By wiring the LCD-pins to the processor port pins. This is the pin mode. The advantage is that you can choose the pins and that they don't have to be on the same port. This can make your PCB design simple. The disadvantage is that more code is needed.
- By attaching the LCD-data pins to the data bus. This is convenient when you have an external RAM chip and will adds little code.

The LCD-display can be connected in PIN mode as follows:

LCD-DISPLAY	PORT	PIN
DB7	PORTB.7	14
DB6	PORTB.6	13
DB5	PORTB.5	12
DB4	PORTB.4	11
E	PORTB.3	6
RS	PORTB.2	4
RW	Ground	5
Vss	Ground	1
Vdd	+5 Volt	2
Vo	0-5 Volt	3

This leaves PORTB.1 and PORTB.0 and PORTD for other purposes. You can change these settings from the Options LCD »Page 31 menu.

BASC0M supports many statements to control the LCD-display. For those who want to have more control the example below shows how to use the internal routines.

```
$ASM
Ldi _temp1, 5           'load register R24 with value
Rcall _Lcd_control      'it is a control value to control the display
Ldi _temp1,65          'load register with new value (letter A)
Rcall _Write_lcd        'write it to the LCD-display
$END ASM
```

Note that _lcd_control and _write_lcd are assembler subroutines which can be called from BASCOM.

See the manufacturer's details from your LCD display for the correct assignment.

Using the I2C protocol

The I2C protocol is a 2-wire protocol designed by Philips. Of course you also need power and ground so it really needs 4 wires.

The I2C protocol was invented for making designs of TV PCB's more simple. But with the availability of many I2C chips, it is ideal for the hobbyist too.

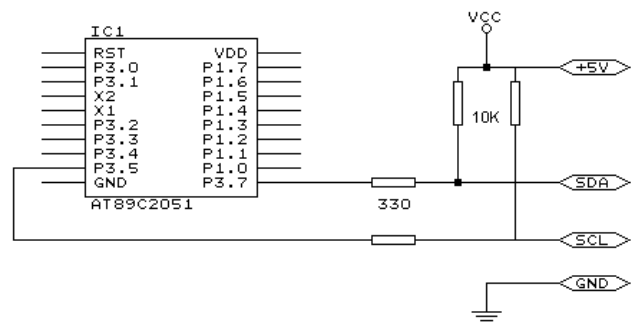
The PCF8574 is a nice chip - it is an I/O extender with 8 pins that you can use either as input or output.

The design below shows how to implement an I2C-bus.

R1 and R2 are 330 ohm resistors.

R3 and R4 are 10 kilo-ohm resistors. For 5V, 4K7 is a good value in combination with AVR chips.

You can select which port pins you want to use for the I2C interface with the compiler settings.



Using the 1 WIRE protocol

The 1 wire protocol was invented by Dallas Semiconductors and needs only 1 wire for the communication. You also need power and ground of course.

This topic is not finished at this stage.

Using the SPI protocol

This topic is not finished yet.

Power Up

At power up all ports are in Tri-state and can serve as input pins. When you want to use the ports (pins) as output, you must set the data direction first with the statement : CONFIG PORTB = OUTPUT

Individual bits can also be set to be uses as input or output. For example : DDRB = &B00001111 , will set a value of 15 to the data direction register of PORTB. PORTB.0 to PORTB.3 (the lower 5 bits) can be used as outputs because they are set low. The upper four bits (PORTB.4 to PORTB.7), can be used for input because they are set low.

You can also set the direction of a port pin with the statement : CONFIG PINB.0 = OUTPUT | INPUT

ReservedWords

The following table shows the reserved BASCOM statements or characters.

^	!	;	\$BAUD	\$CRYSTAL
\$DATA	\$DEFAULT	\$END	\$EEPROM	\$INCLUDE
\$LCD	\$LCDRS	\$LCDPUTCTRL	\$LCDPUTDATA	\$REGFILE
\$SERIALINPUT	\$SERIALINPUT2LCD	\$SERIALOUTPUT	\$XRAMSIZE	\$XRAMSTART
1WRESET	1WREAD	1WWRITE	ACK	ABS()
ALIAS	AND	AS	ASC()	AT
BAUD	BCD()	BIT	BITWAIT	BLINK
BOOLEAN	BYTE	BYVAL	CALL	CAPTURE1
CASE	CHR()	CLS	CLOSE	COMPARE1A
COMPARE1B	CONFIG	CONST	COUNTER	COUNTER0
COUNTER1	COUNTER2	CPEEK()	CRYSTAL	CURSOR
DATA	DEBOUNCE	DECR	DECLARE	DEFBIT
DEFBYTE	DEFLNG	DEFWORD	DEGNSG	DEFLCDCHAR
DEFINT	DEFWORD	DELAY	DIM	DISABLE
DISPLAY	DO	DOWNTO	ELSE	ELSEIF
ENABLE	END	ERAM	ERASE	ERR
EXIT	EXTERNAL	FOR	FOURTH	FOURTHLINE
FUNCTION	GATE	GETAD()	GETRCS5()	GOSUB
GOTO	HEXVAL()	HIGH()	HOME	I2C RECEIVE
I2CSEND	I2CSTART	I2CSTOP	I2CRBYTE	I2CWBYTE
IDLE	IF	INCR	INKEY	INP()
INPUT	INPUTBIN	INPUTHEX	INT0	INT1
INTEGER	INTERNAL	INSTR	IS	LCASE()
LCD	LEFT	LEFT()	LEN()	LOAD
LOCAL	LOCATE	LONG	LOOKUP()	LOOKUPSTR()
LOOP	LTRIM()	LOW()	LOWER	LOWERLINE
MAKEBCD()	MAKEDEC()	MAKEINT()	MID()	MOD
MODE	NACK	NEXT	NOBLINK	NOSAVE
NOT	OFF	ON	OR	OUT
OUTPUT	PEEK()	POKE	PORTA	PORTB
PORTC	PORTD	POWERDOWN	PRINT	PRINTBIN
PWM1A	PWM1B	READ	RADEEPROM	REM
RESET	RESTORE	RETURN	RIGHT	RIGHT()
ROTATE	RTRIM()	SELECT	SERIAL	SET
SHIFT	SHIFTLCD	SHIFTCURSOR	SHIFTIN	SHIFTOUT
SOUND	SPACE()	SPIINIT	SPIIN	SPIMOVE
SPIOUT	START	STEP	STR()	STRING()
STOP	STOPTIMER	SUB	SWAP	THEN
THIRD	THIRDLINE	TIMER0	TIMER1	TIMER2
TO	TRIM()	UCASE()	UNTIL	UPPER
UPPERLINE	VAL()	VARPTR()	WAIT	WAITKEY()
WAITMS	WAITUS	WATCHDOG	WRITEEEPROM	WEND
WHILE	WORD	XOR	XRAM	

LanguageFundamentals

Characters from the BASCOM character set are put together to form labels, keywords, variables and operators.

These in turn are combined to form the statements that make up a program.

This chapter describes the character set and the format of BASCOM program lines. In particular, it discusses:

- The specific characters in the character set and the special meanings of some characters.
- The format of a line in a BASCOM program.
- Line labels.
- Program line length.

Character Set

The BASCOM BASIC character set consists of alphabetic characters, numeric characters, and special characters.

The alphabetic characters in BASCOM are the uppercase letters (A-Z) and lowercase letters (az) of the alphabet.

The BASCOM numeric characters are the digits 0-9.

The letters A-H can be used as parts of hexadecimal numbers.

The following characters have special meanings in BASCOM statements and expressions:

Character	Name
ENTER	Terminates input of a line
	Blank (or space)
'	Single quotation mark (apostrophe)
*	Asterisks (multiplication symbol)
+	Plus sign
,	Comma
-	Minus sign
.	Period (decimal point)
/	Slash (division symbol) will be handled as \
:	Colon
"	Double quotation mark
;	Semicolon
<	Less than
=	Equal sign (assignment symbol or relational operator)
>	Greater than
\	Backslash (integer/word division symbol)
^	Exponent

The BASCOM program line

BASCOSM program lines have the following syntax:

[[*line-identifier*]] [[*statement*]] [[:*statement*]] ... [[*comment*]]

Using Line Identifiers

BASCOSM support one type of line-identifier; alphanumeric line labels:

An alphabetic line label may be any combination of from 1 to 32 letters and digits, starting with a letter and ending with a colon.

BASCOSM keywords are not permitted.

The following are valid alphanumeric line labels:

Alpha:

ScreenSUB:

Test3A:

Case is not significant. The following line labels are equivalent:

alpha:

Alpha:

ALPHA:

Line labels may begin in any column, as long as they are the first characters other than blanks on the line.

Blanks are not allowed between an alphabetic label and the colon following it.

A line can have only one label.

BASCOSM Statements

A BASCOM statement is either "executable" or "non-executable".

An executable statement advances the flow of a programs logic by telling the program what to do next.

Non executable statement perform tasks such as allocating storage for variables, declaring and defining variable types.

The following BASCOM statements are examples of non-executable statements:

- REM or (starts a comment)
- DIM

A "comment" is a non-executable statement used to clarify a programs operation and purpose.

A comment is introduced by the REM statement or a single quote character(').

The following lines are equivalent:

```
PRINT " Quantity remaining" : REM Print report label.
PRINT " Quantity remaining" ' Print report label.
```

More than one BASCOM statement can be placed on a line, but colons(:) must separate statements, as illustrated below.

```
FOR I = 1 TO 5 : PRINT " Gday, mate." : NEXT I
```

BASCOMLineLength

If you enter your programs using the built-in editor, you are not limited to any line length, although it is advised to shorten your lines to 80 characters for clarity.

DataTypes

Every variable in BASCOM has a data type that determines what can be stored in the variable. The next section summarizes the elementary data types.

ElementaryDataTypes

- Bit (1/8 byte). A bit can hold only the value 0 or 1.
A group of 8 bits is called a byte.
- Byte (1 byte).
Bytes are stored as unsigned 8-bit binary numbers ranging in value from 0 to 255.
- Integer (two bytes).
Integers are stored as signed sixteen-bit binary numbers ranging in value from -32,768 to +32,767.
- Word (two bytes).
Words are stored as unsigned sixteen-bit binary numbers ranging in value from 0 to 65535.
- Long (four bytes).
Longs are stored as signed 32-bit binary numbers ranging in value from -2147483648 to 2147483647.
- Single.
Singles are stored as signed 32 bit binary numbers.
- String (up to 254 bytes).
Strings are stored as bytes and are terminated with a 0-byte.
A string dimensioned with a length of 10 bytes will occupy 11 bytes.

Variables can be stored internal (default) , external or in EEPROM.

Variables

A variable is a name that refers to an object--a particular number.

A numeric variable, can be assigned only a numeric value (either integer,

byte, long, single or bit).

The following list shows some examples of variable assignments:

- A constant value:
A = 5
C = 1.1
- The value of another numeric variable:
abc = def
k = g
- The value obtained by combining other variables, constants, and operators:
Temp = a + 5
Temp = C + 5
- The value obtained by calling a function:
Temp =Asc(S)

VariableNames

A BASCOM variable name may contain up to 32 characters. The characters allowed in a variable name are letters and numbers. The first character in a variable name must be a letter.

A variable name cannot be a reserved word, but embedded reserved words are allowed.

For example, the following statement is illegal because AND is a reserved word.

```
AND = 8
```

However, the following statement is legal:

```
ToAND = 8
```

Reserved words include all BASCOM commands, statements, function names, internal registers and operator names. (see BASCOM Reserved Words »Page 56 , for a complete list of reserved words).

You can specify a hexadecimal or binary number with the prefix &H or &B. a = &HA , a = &B1010 and a = 10 are all the same.

Before assigning a variable, you must tell the compiler about it with the DIM statement.

Dim b1 As Bit, I as Integer, k as Byte , s As String * 10

The STRING type needs an additional parameter to specify the length.

You can also use DEFINT, DEFBIT, DEFBYTE ,DEFWORD ,DEFLNG or DEFSNG.

For example DEFINT c tells the compiler that all variables that are not dimensioned and that are beginning with the character c are of the Integer type.

Expressions and Operators

This chapter discusses how to combine, modify, compare, or get information about expressions by using the operators available in BASCOM.

Anytime you do a calculation you are using expressions and operators. This chapter describes how expressions are formed and concludes by describing the following kind of operators:

- Arithmetic operators, used to perform calculations.
- Relational operators, used to compare numeric or string values.
- Logical operators, used to test conditions or manipulate individual bits.
- Functional operators, used to supplement simple operators.

Expressions and Operators

An expression can be a numeric constant, a variable, or a single value obtained by combining constants, variables, and other expressions with operators.

Operators perform mathematical or logical operations on values.

The operators provided by BASCOM can be divided into four categories, as follows:

1. Arithmetic
2. Relational
3. Logical
4. Functional

Arithmetic

Arithmetic operators are +, -, *, \, / and ^.

- Integer

Integer division is denoted by the backslash (\).

Example: $Z = X \setminus Y$

- Modulo Arithmetic

Modulo arithmetic is denoted by the modulus operator **MOD**.

Modulo arithmetic provides the remainder, rather than the quotient, of an integer division.

Example: $X = 10 \setminus 4$: remainder = $10 \text{ MOD } 4$

- Overflow and division by zero

Division by zero, produces an error.

At the moment no message is produced, so you have to make sure yourself that this won't happen.

Relational Operators

Relational operators are used to compare two values as shown in the table below.

The result can be used to make a decision regarding program flow.

Operator	Relation Tested	Expression
=	Equality	$X = Y$
<>	Inequality	$X <> Y$
<	Less than	$X < Y$
>	Greater than	$X > Y$
<=	Less than or equal to	$X <= Y$
>=	Greater than or equal to	$X >= Y$

Logical Operators

Logical operators perform tests on relations, bit manipulations, or Boolean operators.

There four operators in BASCOM are :

Operator	Meaning
NOT	Logical complement
AND	Conjunction
OR	Disjunction
XOR	Exclusive or

It is possible to use logical operators to test bytes for a particular bit pattern.

For example the AND operator can be used to mask all but one of the bits of a status byte, while OR can be used to merge two bytes to create a particular binary value.

Example

A = 63 And 19

```
PRINT A
A = 10 Or 9
PRINT A
```

Output
16
11

Floatingpoint (ASM code used is supplied by Jack Tidwell)
Single numbers conforming to the IEEE binary floating point standard.
An eight bit exponent and 24 bit mantissa are supported.
Using four bytes the format is shown below:

```
31 30 _____ 23 22 _____ 0
s  exponent  mantissa
```

The exponent is biased by 128. Above 128 are positive exponents and below are negative. The sign bit is 0 for positive numbers and 1 for negative. The mantissa is stored in hidden bit normalized format so that 24 bits of precision can be obtained.

All mathematical operations are supported by the single.

You can also convert a single to an integer or word or vice versa:

```
Dim I as Integer, S as Single
S = 100.1 'assign the single
I = S 'will convert the single to an integer
```

Arrays

An array is a set of sequentially indexed elements having the same type.
Each element of an array has a unique index number that identifies it.
Changes made to an element of an array do not affect the other elements.

The index must be a numeric constant, a byte, an integer, word or long.
The maximum number of elements is 65535.

The first element of an array is always one. This means that elements are 1-based.

Arrays can be used on each place where a 'normal' variable is expected.

Example:

```
Dim a(10) as byte 'make an array named a, with 10 elements (1 to 10)
Dim c as Integer
For C = 1 To 10
  a(c) = c 'assign array element
  Printa(c) 'print it
Next
a(c + 1) = a 'you can add an offset to the index too
```

\$ASM

Action

Start of inline assembly code block.

Syntax

\$ASM

Remarks

Use \$ASM together with \$END ASM to insert a block of assembler code in your BASIC code.

Example

```
Dim c as Byte
Loadadr x,x 'load address of variable C into register X
$ASM
  Ldi R24,1 'load register R24 with the constant 1
  St X,R24 ;store 1 into var c
$END ASM
Print c
End
```

\$BAUD

Action

Instruct the compiler to override the baud rate setting from the options menu.

Syntax

\$BAUD = var

Remarks

Var The baud rate that you want to use.

var : Constant.

The baud rate is selectable from the Compiler Settings »Page 29. It is stored in a configuration file. The \$BAUD statement is provided for compatibility with BASCOM-8051.

In the generated report, you can view which baud rate is actually generated.

Seealso

\$CRYSTAL »Page65 , BAUD »Page 85

Example

```
$BAUD = 2400
$CRYSTAL = 14000000      ' 14 MHz crystal
Print "Hello"
'Now change the baudrate in a program
BAUD = 9600 '
Print "Did you change the terminal emulator baud rate too?"
END
```

\$CRYSTAL

Action

Instruct the compiler to override the crystal frequency options setting.

Syntax

\$CRYSTAL = var

Remarks

var Frequency of the crystal.

var : Constant.

The frequency is selectable from the Compiler Settings »Page 29. It is stored in a configuration file. The \$CRYSTAL statement is provided for compatibility with BASCOM-8051.

Seealso

\$BAUD »Page 64 BAUD »Page85

Example

```
$BAUD = 2400
$CRYSTAL = 14000000
PRINT "Hello"
END
```

\$DATA

Action

Instruct the compiler to store the data in the DATA lines following the \$DATA directive, in code memory.

Syntax

\$DATA

Remarks

The AVR has built-in EEPROM. With the WRITEEEPROM and

READEEPROM statements, you can write and read to the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.

A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory or the EEP file and therefore two compiler directives were added.

\$EEPROM and \$DATA.

\$EEPROM tells the compiler that the DATA lines following the compiler directive, must be stored in the EEP file.

To switch back to the default behaviour of the DATA lines, you must use the \$DATA directive.

Seealso

\$EEPROM »Page 67

ASM

Example

```
Dim B As Byte
Restore Lbl1      'point to code data
Read B
Print B
Restore Lbl2
Read B
Print B
End

Lbl1:
DATA 100

$EEPROM          'the following DATA lines data will go to the EEP
                 'file

DATA 200

$DATA           'switch back to normal
Lbl2:
DATA 300
```

\$DEFAULT

Action

Set the default for data types dimensioning to the specified type.

Syntax

\$DEFAULT = var

Remarks

Var SRAM, XRAM, ERAM

Each variable that is dimensioned will be stored into SRAM, the internal memory of the chip. You can override it by specifying the data type.

Dim B As XRAM Byte , will store the data into external memory.

When you want all your variables to be stored in XRAM for example, you can use the statement : \$DEFAULT XRAM

Each Dim statement will place the variable in XRAM than.

To switch back to the default behaviour, use \$END \$DEFAULT

Seealso

ASM

Example

```
$DEFAULT XRAM
Dim A As Byte, b As Byte, C As Byte
'a,b and c will be stored into XRAM
```

```
$DEFAULT SRAM
Dim D As Byte
'D will be stored in internal memory, SRAM
```

\$EEPROM

Action

Instruct the compiler to store the data in the DATA lines following the \$DATA directive in an EEP file.

Syntax

\$EEPROM

Remarks

The AVR has build in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write and read to the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.

A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory or the EEP file and therefore two compiler directives were added. \$EEPROM and \$DATA.

\$EEPROM tells the compiler that the DATA lines following the compiler directive, must be stored in the EEP file.

To switch back to the default behaviour of the DATA lines, you must use the \$DATA directive.

Seealso

\$DATA »Page 65

ASM

Example

```
Dim B As Byte
Restore Lbl          'point to code data
Read B
Print B
Restore Lbl2
Read B
Print B
End
```

```
Lbl:
DATA 100
```

```
$EEPROM          'the following DATA lines data will go to the EEP
                  'file
```

```
DATA 200
```

```
$DATA          'switch back to normal
Lbl2:
DATA 300
```

\$EXTERNAL

Action

Instruct the compiler to include ASM routines form a library.

Syntax

\$EXTERNAL Myroutine [, myroutine2]

Remarks

You can place ASM routines in a library file. With the \$EXTERNAL directive you tell the compiler which routines must be included in your program.

An automatic search will be added later so the \$EXTERNAL directive will not be needed any longer.

Seealso

\$LIB »Page 73

Example

```
Dim B As Byte
$LIB "Mylib.LIB"
$EXTERNAL TestAsm
Rcall TestAsm
End
```

\$INCLUDE

Action

Includes an ASCII file in the program at the current position.

Syntax

\$INCLUDE file

Remarks

File Name of the ASCII file, which must contain valid BASCOM statements. This option can be used if you make use of the same routines in Many programs. You can write modules and include them into your program. If there are changes to make you only have to change the module file, not all your BASCOM programs. You can only include ASCII files!

Example

```
'-----
'                               (c) 1997-2000 MCS Electronics
'-----
' file: INCLUDE.BAS
' demo: $INCLUDE
'-----
Print "INCLUDE.BAS"
$include c:\bascom\123.bas      'include file that prints Hello
Print "Back in INCLUDE.BAS"
End
```

\$LCD

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

\$LCD = [&H] address

Remarks

Address

The address where must be written to, to enable the LCD display and the RS line of the LCD display. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7) The RS line of the LCD can be configured with the LCDRS statement.

On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

Seealso

\$LCDRS »Page 72

Example

```
REM We use a STK200 board so use the following addresses
$LCD = &HC000      'writing to this address will make the E-line of
                  'the LCD 'high and the RS-line of the LCD high.
$LCDRS = &H8000    'writing to this address will make the E-line of
                  'the LCD 'high.

Cls
LCD "Hello world"
```

\$LCDPUTCTRL

Action

Specifies that LCD control output must be redirected.

Syntax

\$LCDPUTCTRL = label

Remarks

Label

The name of the assembler routine that must be called when a control byte is printed with the LCD statement. The character must be placed in R24/_temp1.

With the redirection of the LCD statement, you can use your own routines.

Seealso

\$SERIALPUTDATA »Page 71

Example

```
'define chip to use
$regfile = "8535def.dat"

'define used crystal
$crystal = 4000000

'dimension used variables
Dim S As String * 10
Dim W As Long

'inform the compiler which routine must be called to get serial
'characters
$LCDPUTDATA = Myoutput
$LCDPUTCTRL = MyoutputCtrl
'make a never ending loop
Do
    LCD "test"
Loop

End

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and
'restore
'all registers so we can use all BASIC statements
'$LCDPUTDATA requires that the character is passed in R24

Myoutput:
    Pushall                'save all registers
    'your code here
    Popall                 'restore registers
    Return

MyoutputCtrl:
    Pushall                'save all registers
    'your code here
    Popall                 'restore registers
    Return
```

\$LCDPUTDATA

Action

Specifies that LCD data output must be redirected.

Syntax

\$LCDPUTDATA = label

Remarks

Label The name of the assembler routine that must be called when a character is printed with the LCD statement. The character must be placed in R24/_temp1.

With the redirection of the LCD statement, you can use your own routines.

Seealso

\$SERIALPUTCTRL »Page 70

Example

```
'define chip to use
$regfile = "8535def.dat"

'define used crystal
$crystal = 4000000

'dimension used variables
Dim S As String * 10
Dim W As Long

'inform the compiler which routine must be called to get serial
'characters
$LCDPUTDATA = Myoutput

'make a never ending loop
Do
    LCD "test"
Loop

End

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and
'restore
'all registers so we can use all BASIC statements
'$LCDPUTDATA requires that the character is passed in R24

Myoutput:
    Pushall                'save all registers
    'your code here
    Popall                 'restore registers
    Return

MyoutputCtrl:
    Pushall                'save all registers
    'your code here
    Popall                 'restore registers
    Return
```

\$LCDRS

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

\$LCDRS = [&H] *address*

Remarks

Address

The address where must be written to, to enable the LCD display.

The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7)

On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

See also

\$LCD »Page 69

Example

```
REM We use a STK200 board so use the following addresses
$LCD = &HC000    'writing to this address will make the E-line of
                 the LCD 'high and the RS-line of the LCD high.
$LCDRS = &H8000  'writing to this address will make the E-line of
                 the LCD 'high.
```

```
ClS
LCD "Hello world"
```

\$LIB

Action

Informs the compiler about the use libraries.

Syntax

\$LIB "libname1" [, "libname2"]

Remarks

Libname is the name of the library that holds ASM routines that are used by your program. More filenames can be specified by separating the names by a comma.

The libraries will be searched when you specify the routines to use with the \$EXTERNAL directive.

The search order is the same as the order you specify the library names.

The MCS.LIB will be searched last and is always included so you don't need to specify it with the \$LIB directive.

Because the MCS.LIB is searched last you can include duplicate routines in

your own LIB. Now these routines will be used instead of the ones from the default MCS.LIB library. This is a good way when you want to enhance the MCS.LIB routines. Just copy the MCS.LIB to a new file and make the changes in this new file. When we make changes to the library your changes will be preserved.

Creating your own LIB file

A library file is a simple ASCII file. It can be created with the BASCOM editor, notepad or any other ASCII editor.

The file must include the following header information. It is not used yet but will be later.

copyright = Your name

www = optional location where people can find the latest source

email = your email address

comment = AVR compiler library

libversion = the version of the library in the format : 1.00

date = date of last modification

statement = A statement with copyright and usage information

The routine must start with the name in brackets and must end with the [END].

The following ASM routine example is from the MCS.LIB library.

[_ClockDiv]

```
; MEGA chips only
;_templ holds the division in the range from 0-129
; 0 will set the division to 1
_ClockDiv:
Cpi _templ,0          ; is it zero?
Breq _ClockDivX      ; yes so turn of the division
Subi _templ,2         ; subtract 2
Com _templ            ;complement
Clr _temp2
Out XDIV,_temp2      ; enable write by writing zeros
_ClockDivX:
Out XDIV,_templ      ; write new division
Ret                  ;return
```

[END]

See also

\$EXTERNAL »Page 68

Example

```
'define chip to use
$regfile = "8535def.dat"

'define used crystal
```

```

$crystal = 4000000

'dimension used variables
Dim S As String * 10
Dim W As Byte

$LIB "MYLIB.LIB" , "MCS.LIB"
$EXTERNAL _ShiftL1 , _ShiftL2

Shift W , LEFT , 2      'uses _shiftL1

```

\$REGFILE

Action

Instruct the compiler to use the specified register file instead of the selected dat file.

Syntax

\$REGFILE = var

Remarks

Var The name of the register file. The register files are stored in the BASCOM-AVR application directory and all end with the DAT extension.
The register file holds information about the chip such as the internal registers and interrupt addresses.

The \$REGFILE statement overrides the setting from the Options menu. The settings are stored in a <project>.CFG file and the directive is added for compatibility with BASCOM-8051

The \$REGFILE directive must be the first statement in your program.

Seealso

ASM

Example

```
$REGFILE = "8515DEF.DAT"
```

\$SERIALINPUT

Action

Specifies that serial input must be redirected.

Syntax

\$SERIALINPUT = label

Remarks

Label The name of the assembler routine that must be called when a character is needed from the INPUT routine. The character must be returned in R24/_temp1.

With the redirection of the INPUT command, you can use your own routines. This way you can use other devices as input devices.

Note that the INPUT statement is terminated when a RETURN code (13) is received.

Seealso

\$SERIALOUTPUT »Page 78

Example

```

-----
'                                     $myserialinput.bas
'                                     (c) 2000 MCS Electronics
'      demonstrates $SERIALINPUT redirection of serial input
-----
'define chip to use
$regfile = "8535def.dat"

'define used crystal
$crystal = 4000000

'dimension used variables
Dim S As String * 10
Dim W As Long

'inform the compiler which routine must be called to get serial
'characters
$serialinput = Myinput

'make a never ending loop
Do
  'ask for name
  Input "name " , S
  Print S
  'error is set on time out
  Print "Error " ; Err
Loop

End

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and

```

```
'restore
'all registers so we can use all BASIC statements
'$SERIALINPUT requires that the character is passed back in R24

Myinput:
  Pushall                'save all registers
  W = 0                  'reset counter

Myinput1:
  Incr W                 'increase counter
  Sbis USR, 7            ' Wait for character
  Rjmp myinput2          'no charac waiting so check again
  Popall                 'we got something
  Err = 0                'reset error
  In _templ, UDR         ' Read character from UART
  Return                 'end of routine

Myinput2:
  If W > 1000000 Then    'with 4 MHz ca 10 sec delay
    rjmp Myinput_exit    'waited too long
  Else
    Goto Myinput1        'try again
  End If

Myinput_exit:
  Popall                 'restore registers
  Err = 1                'set error variable
  ldi R24, 13            'fake enter so INPUT will end
Return
```

\$SERIALINPUT2LCD

Action

This compiler directive will redirect all serial input to the LCD display instead of echoing to the serial port.

Syntax

```
$SERIALINPUT2LCD
```

Remarks

You can also write your own custom input or output driver with the `$SERIALINPUT` »Page 75 and `$SERIALOUTPUT` statements, but the `$SERIALINPUT2LCD` is handy when you use a LCD display.

Seealso

`$SERIALINPUT` »Page 75 , `$SERIALOUTPUT` »Page 78

Example

```
$SERIALINPUT2LCD
Dim v as Byte
CLS
INPUT "Number ", v      'this will go to the LCD display
```

\$SERIALOUTPUT

Action

Specifies that serial output must be redirected.

Syntax

```
$SERIALOUTPUT = label
```

Remarks

label The name of the assembler routine that must be called when a character is send to the serial buffer (UDR).
The character is placed into R24/_temp1.

With the redirection of the PRINT and other serial output related commands, you can use your own routines.

This way you can use other devices as output devices.

Seealso

`$SERIALINPUT` »Page 75 , `$SERIALINPUT2LCD` »Page 77

Example

```
$SERIALOUTPUT = MyOutput
'your program goes here
END
myoutput:
'perform the needed actions here
  Ldi _templ,65 ;serial output buffer (default)
ret
```

\$XRAMSIZE

Action

Specifies the size of the external RAM memory.

Syntax

```
$XRAMSIZE = [&H] size
```

Remarks

size Size of external RAM memory chip.
size : Constant.

The size of the chip can be selected from the Options Compiler Chip »Page

27 menu.

The \$XRAMSIZE overrides this setting.

Seealso

\$XRAMSTART »Page 79

Example

```
$XRAMSTART = &H300
$RAMSIZE = &H1000
DIM x AS XRAM Byte      'specify XRAM to store variable in XRAM
```

\$XRAMSTART

Action

Specifies the location of the external RAM memory.

Syntax

\$XRAMSTART = [&H] *address*

Remarks

address The (hex)-address where the data is stored. Or the lowest address that enables the RAM chip. You can use this option when you want to run your code in systems with external RAM memory.

address : Constant.

By default the extended RAM will start after the internal memory so the lower addresses of the external RAM can't be used to store information.

When you want to protect an area of the chip, you can specify a higher address for the compiler to store the data. For example, you can specify &H400. The first dimensioned variable will be placed in address &H400 and not in &H260.

Seealso

\$XRAMSIZE »Page 78

Example

```
$XRAMSTART = &H400
$XRAMSIZE = &H1000
Dim B As Byte
```

1WRESET

Action

This statement brings the 1wire pin to the correct state, and sends a reset to the bus.

Syntax

1WRESET
1WRESET [*pin*]

Remarks

1WRESET Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred
pin An optional parameter. This is the port pin that is used for the 1wire statements. For example PORTB.1

New is support for multi 1-wire devices on different pins.

To use this you must specify the port pin that is used for the communication. The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement.

The syntax for additional 1-wire devices is :

1WRESET *pin*
1WWRITE *var/constant* [,bytes] [*pin*]
var = **1WREAD**(*pin*) , for reading one byte
var = **1WREAD**(bytes, *pin*) ,for reading multiple bytes

Seealso

1WREAD »Page 81 , 1WWRITE »Page 82

Asm

Calls: `_resetsd` or for the additional devices : `_resetsd_pin`. For example :
`_resetsd_portb.1`

Input: -

Output: - ERR is set when an error occurred.

Example

```
'-----
'                               1WIRE.BAS
' Demonstrates lwreset, lwwrite and lwread()
' pullup of 4K7 required to VCC from PORTB.1
' DS2401 serial button connected to PORTB.1
'-----
Config lwire = PORTB.1          'use this pin
Dim Ar(8) As Byte , A As Byte , I As Byte

lwreset                        'reset the bus
Print Err                      'print error 1 if error
```

```

1wwrite &H33                'read ROM command
For I = 1 To 8
  Ar(I) = lwwrite()          'read byte
Next
'or ar(1) = lwwrite(8)       'read 8 bytes
For I = 1 To 8
  Print hex(Ar(I));         'print output
Next
Print                        'linefeed
End

```

1WREAD

Action

This statement reads data from the 1wire bus into a variable.

Syntax

var2 = 1WREAD([bytes])

var2 = 1WREAD(pin) or **var2 = 1WREAD(pin [, bytes])**

Remarks

var2 =	Reads a byte from the bus and places it into var2.
1WREAD([bytes])	Optional, the number of bytes to read can be specified.
pin	An optional parameter. This is the port pin that is used for the 1wire statements. For example PORTB.1

New is support for multi 1-wire devices on different pins. To use this you must specify the port pin that is used for the communication. The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement.

The syntax for additional 1-wire devices is :

1WRESET pin

1WWRITE var/constant [,bytes] [pin]

var = 1WREAD(**pin**) , for reading one byte

var = 1WREAD(bytes, **pin**) ,for reading multiple bytes

Asm

Calls: _reads for 1 byte and _ReadDSbytes for mutple bytes.

Input: R18 number of bytes, X address of variable

Output:

Seealso

1WWRITE »Page 82 , 1WRESET »Page 80

Example

```

-----
'                               1WIRE.BAS
' Demonstrates lwreset, lwwrite and lwread()
' pullup of 4K7 required to VCC from PORTB.1
' DS2401 serial button connected to PORTB.1
-----
Config lwire = PORTB.1          'use this pin
Dim Ar(8) As Byte , A As Byte , I As Byte

lwreset                        'reset the bus
Print Err                      'print error 1 if error
lwwrite &H33                   'read ROM command
For I = 1 To 8
  Ar(I) = lwread()             'read byte
Next
'or ar(1) = lwread(8)          'read 8 bytes
For I = 1 To 8
  Print hex(Ar(I));           'print output
Next
Print                          'linefeed
End

```

1WWRITE

Action

This statement writes a variable to the 1wire bus.

Syntax

1WWRITE var1 [, bytes]

1WWRITE var1 [, bytes] [pin]

Remarks

var1	Sends the value of var1 to the bus. The number of bytes can be specified too but this is optional.
pin	An optional parameter. This is the port pin that is used for the 1wire statements. For example PORTB.1

New is support for multi 1-wire devices on different pins.

To use this you must specify the port pin that is used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement.

The syntax for additional 1-wire devices is :

1WRESET pin

1WWRITE var/constant [,bytes] [pin]

var = 1WREAD(**pin**) , for reading one byte
var = 1WREAD(bytes, **pin**) ,for reading multiple bytes

Seealso

1WREAD »Page 81 , 1WRESET »Page 80

Asm

Calls: `_writeds` for 1 byte or `_WriteDSBytes` for multiple bytes

Input: R18 number of bytes, X address of variable

Output: -

Example

```

-----
'                               1WIRE.BAS
' Demonstrates lwire, lwwrite and lwiread()
' pullup of 4K7 required to VCC from PORTB.1
' DS2401 serial button connected to PORTB.1
-----
Config lwire = PORTB.1           'use this pin
Dim Ar(8) As Byte , A As Byte , I As Byte

lwreset                          'reset the bus
Print Err                        'print error 1 if error
lwwrite &H33                      'read ROM command
For I = 1 To 8
  Ar(I) = lwiread()              'read byte
Next
'or ar(1) = lwiread(8)          'read 8 bytes
For I = 1 To 8
  Print hex(Ar(I));             'print output
Next
Print                             'linefeed
End

```

ALIAS

Action

Indicates that the variable can be referenced with another name.

Syntax

`newvar ALIAS oldvar`

Remarks

`oldvar` Name of the variable such as PORTB.1
`newvar` New name of the variable such as direction

Aliasing port pins can give the pin names a more meaningful name.

Seealso

CONST »Page 122

Example

```

direction ALIAS PORTB.1 'now you can refer to PORTB.1 with the
variable direction
SET direction           'has the same effect as SET PORTB.1
END

```

ABS()

Action

Returns the absolute value of a numeric signed variable.

Syntax

`var = ABS(var2)`

Remarks

`var` Variable that is assigned the absolute value of var2.
`Var2` The source variable to retrieve the absolute value from.

var : Byte, Integer, Word, Long.
var2 : Integer, Long.

The absolute value of a number is always positive.

Seealso

-

Difference with QB

You can not use numeric constants since the absolute value is obvious for numeric constants.

Does not work with Singles.

Asm

Calls: `_abs16` for an Integer and `_abs32` for a Long
Input: R16-R17 for an Integer and R16-R19 for a Long
Output: R16-R17 for an Integer and R16-R19 for a Long

Example

```

Dim a as Integer, c as Integer
a = -1000
c = Abs(a)
Print c

```

End

ASC

Action

Convert a string into its ASCII value.

Syntax

var = ASC(string)

Remarks

var Target variable that is assigned.
String String variable or constant from which to retrieve the ASCII value.

var : Byte, Integer, Word, Long.
string : String, Constant.

Note that only the first character of the string will be used.
 When the string is empty, a zero will be returned.

Seealso

CHR »Page 90

Asm

Calls: -
 Input:
 Output: `_temp1=R24`

Example

```
Dim a as byte, s as String * 10
s = "ABC"
a = Asc(s)
Print a           'will print 65
End
```

BAUD

Action

Changes the baud rate for the hardware UART.

Syntax

BAUD = var

Remarks

Var The baud rate that you want to use.

Do not confuse the BAUD statement with the \$BAUD compiler directive.
 \$BAUD overrides the compiler setting for the baud rate and BAUD will change the current baud rate.

Seealso

\$CRYSTAL »Page 65 , \$BAUD »Page 64

Asm

Calls: -
 Input: -
 Output: -
 Code : `Ldi _temp1, baud`
 `Out UBRR, _temp1`

Example

```
$BAUD = 2400
$CRYSTAL = 14000000 ' 14 MHz crystal
Print "Hello"
'Now change the baudrate in a program
BAUD = 9600 '
Print "Did you change the terminal emulator baud rate too?"
END
```

BCD

Action

Converts a variable stored in BCD format into a string.

Syntax

PRINT BCD(var)
LCD BCD(var)

Remarks

Var Variable to convert.

var1 : Byte, Integer, Word, Long, Constant.

When you want to use an I2C clock device which stores its values as BCD values you can use this function to print the value correctly.
 BCD() displays values with a leading zero.

The BCD() function is intended for the PRINT/LCD statements.
 Use the MAKEBCD function to convert variables from decimal to BCD.

Use the MAKEDEC function to convert variables from BCD to decimal.

Seealso

MAKEDEC »Page 158 , MAKEBCD »Page 157

Asm

Calls: _BcdStr

Input: X hold address of variable

Output: R0 with number of bytes, frame with data.

Example

```
Dim a as byte
a = 65
LCD a
Lowerline
LCD BCD(a)
End
```

BITWAIT

Action

Wait until a bit is set or reset.

Syntax

BITWAIT x SET/RESET

Remarks

X Bit variable or internal register like PORTB.x , where x ranges from 0-7.

When using bit variables be sure that they are set/reset by software.

When you use internal registers that can be set/reset by hardware such as PORTB.0 this doesn't apply.

Seealso

-

Asm

Calls: -

Input: -

Output: -

Code : shown for address 0-31

label1:

Sbic PINB.0,label2

Rjmp label1

Label2:

Example

```
Dim a as bit
BITWAIT a , SET 'wait until bit a is set
BITWAIT PORTB.7, RESET 'wait until bit 7 of Port B is 0.
End
```

BYVAL

Action

Specifies that a variable is passed by value.

Syntax

Sub Test(BYVAL var)

Remarks

Var Variable name

The default for passing variables to SUBS and FUNCTIONS, is by reference , BYREF. When you pass a variable by reference, the address is passed to the SUB or FUNCTION. When you pass a variable by Value, a temp variable is created on the frame and the address of the copy is passed.

When you pass by reference, changes to the variable will be made to the calling variable.

When you pass by value, changes to the variable will be made to the copy so the original value will not be changed.

By default passing by reference is used.

Seealso

CALL »Page 88 , DECLARE »Page 118 , SUB »Page 184 , FUNCTION »Page 116

ASM

Example

```
Declare Sub Test(Byval X As Byte, Byref Y As Byte, Z As Byte)
```

CALL

Action

Call and execute a subroutine.

Syntax

CALL Test [(var1, var-n)]

Remarks

Var1 Any BASCOM variable or constant.

Var-n Any BASCOM variable or constant.

Test Name of the subroutine. In this case Test.

You can call sub routines with or without passing parameters.

It is important that the SUB routine is DECLARED before you make the CALL to the subroutine. Of course the number of declared parameters must match the number of passed parameters.

It is also important that when you pass constants to a SUB routine, you must DECLARE these parameters with the BYVAL argument.

With the CALL statement, you can call a procedure or subroutine.
For example: **Call Test2**

The call statement enables you to implement your own statements.

You don't have to use the CALL statement:

Test2 will also call subroutine test2

When you don't supply the CALL statement, you must leave out the parenthesis.

So Call Routine(x,y,z) must be written as Routine x,y,x

Unlike normal SUB programs called with the GOSUB statement, the CALL statement enables you to pass variables to a SUB routine.

See also

DECLARE »Page 118 , SUB »Page 184 , EXIT »Page 128 , FUNCTION »Page 116 , LOCAL »Page 152

Example

```
Dim A As Byte, B As Byte      'dimension some variables
Declare Sub Test(b1 As Byte, BYVAL b2 As Byte) 'declare the SUB
program
a = 65                        'assign a value to variable
A
Call test (a , 5)            'call test with parameter A and
constant
test a , 5                   'alternative call
Print A                      'now print the new value
End
```

```
SUB Test(b1 as byte, BYVAL b2 as byte) 'use the same variable
names as 'the declared
one !!!
LCD b1                               'put it on the LCD
Lowerline
LCD BCD(b2)
B1 = 10                               'reassign the variable
B2 = 15                               'reassign the variable
End SUB
```

One important thing to notice is that you can change b2 but that the change will not be reflected to the calling program!

Variable A is changed however.

This is the difference between the BYVAL and BYREF argument in the DECLARE ration of the SUB program.

When you use BYVAL, this means that you will pass the argument by its value. A copy of the variable is made and passed to the SUB program. So the SUB program can use the value and modify it, but the change will not be reflected to the calling parameter. It would be impossible too when you pass a numeric constant for example.

If you do not specify BYVAL, BYREF will be used by default and you will pass the address of the variable. So when you reassign B1 in the above example, you are actually changing parameter A.

CHR

Action

Convert a numeric variable or a constant to a character.

Syntax

PRINT CHR(var)

s = CHR(var)

Remarks

Var Numeric variable or numeric constant.

S A string variable.

When you want to print a character to the screen or the LCD display, you must convert it with the CHR() function.

When you use PRINT numvar, the value will be printed.

When you use PRINT Chr(numvar), the ASCII character itself will be printed.

The Chr() function is handy in combination with the LCD custom characters

where you can redefine characters 0-7 of the ASCII table.

Seealso

ASC() »Page 85

Example

```
Dim a As Byte           'dim variable
a = 65                  'assign variable
LCD a                   'print value (65)
Lowerline
LCD HEX(a)              'print hex value (41)
LCD Chr(a)              'print ASCII character 65 (A)
End
```

CLS

Action

Clear the LCD display and set the cursor to home.

Syntax

CLS

Remarks

Clearing the LCD display does not clear the CG-RAM in which the custom characters are stored.

Seealso

\$LCD »Page 69, LCD »Page 147, SHIFTLCD »Page 177, SHIFTCURSOR »Page 175, SHIFTLCD »Page 177

Example

```
Cls                     'Clear LCD display
LCD "Hello"             'show this famous text
End
```

Action

Will set the system clock division available in the MEGA chips.

Syntax

CLOCKDIVISON =var

Remarks

var Variable or numeric constant that sets the clock division.

Valid values are from 2-129.

A value of 0 will disable the division.

On the MEGA 103 and 603 the system clock frequency can be divided so you can save power for instance. A value of 0 will disable the clock divider. The divider can divide from 2 to 127. So the other valid values are from 2 - 127. Some routines that rely on the system clock will not work proper anymore when you use the divider. WAITMS for example will take twice the time when you use a value of 2.

Seealso

POWERSAVE »Page 164

Example

```
$BAUD = 2400
Clockdivision = 2
END
```

CLOSE

Action

Opens and closes a device.

Syntax

OPEN "device" for MODE As #channel
CLOSE #channel

Remarks

device The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits.

The format for COM1 is : COM1:speed, where the speed is optional and will override the compiler settings for the speed.

The format for the software UART is: COMpin:speed,8,N,stop bits[,INVERTED]

Where pin is the name of the PORT-pin.

Speed must be specified and stopbits can be 1 or 2.

An optional parameter ,**INVERTED** can be specified to use inverted RS-

232.

Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.

MODE You can use BINARY or RANDOM for COM1, but for the software UART pins, you must specify INPUT or OUTPUT.

channel The number of the channel to open. Must be a positive constant >0.

The statements that support the device are PRINT , INPUT and INPUTHEX.

Every opened device must be closed using the CLOSE #channel statement.

Of course, you must use the same channel number.

The best place for the CLOSE statement is at the end of your program.

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

See also

OPEN »Page 161 , PRINT »Page 165

Example

```

'-----
'                               OPEN.BAS
'   demonstrates software UART
'-----

Dim B As Byte

'open channel for output and use inverted logic

Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 , "serial output"
Close #1

'Now open a pin for input and use inverted logic
Open "comd.2:9600,8,n,1,inverted" For Input As #2
Input #2 , B
Close #2

'use normal hardware UART for printing
Print B

End

```

CONFIG

The CONFIG statement is used to configure the hardware devices.

CONFIG TIMER0 »Page 101

CONFIG TIMER1 »Page 103

CONFIG LCD »Page 97

CONFIG LCDBUS »Page 97

CONFIG LCDMODE »Page 98

CONFIG 1WIRE »Page 94

CONFIG SDA »Page 99

CONFIG SCL »Page 100

CONFIG DEBOUNCE »Page 94

CONFIG SPI »Page 101

CONFIG LCDPIN »Page 99

CONFIG WATCHDOG »Page 106

CONFIG PORT »Page 107

CONFIG KBD »Page 96

CONFIG I2CDELAY »Page 95

CONFIG INTx »Page 96

CONFIG WAITSUART »Page 106

CONFIG 1WIRE

Action

Configure the pin to use for 1WIRE statements and override the compiler setting.

Syntax

CONFIG 1WIRE =pin

Remarks

Pin The port pin to use such as PORTB.0

The CONFIG 1WIRE statement, only overrides the compiler setting.

You can have only one pin for the 1WIRE statements because the idea is that you can attach multiple 1WIRE devices to the 1WIRE bus.

See also

1WRESET »Page 80 , 1WREAD »Page 81 , 1WWRITE »Page 82

Example

```

Config 1WIRE = PORTB.0 'PORTB.0 is used for the 1-wire bus
1WRESET                'reset the bus

```

CONFIGDEBOUNCE

Action

Configures the delay time for the DEBOUNCE statement.

Syntax

CONFIG DEBOUNCE = time

Remarks

Time A numeric constant which specifies the delay time in mS.

When debounce time is not configured, 25 mS will be used as a default.

Seealso

DEBOUNCE

Example

```

CONFIG DEBOUNCE = 30 'when the config statement is not used a
default
                        ' of 25mS will be used

'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1)
Debounce Pind.0 , 0 , Pr , Sub
Debounce Pind.0 , 0 , Pr , Sub
'
'      ^----- label to branch to
'      ^----- Branch when PIND.0 goes low(0)
'      ^----- Examine PIND.0

'When Pind.0 goes low jump to subroutine Pr
'Pind.0 must go high again before it jumps again
'to the label Pr when Pind.0 is low

Debounce Pind.0 , 1 , Pr 'no branch
Debounce Pind.0 , 1 , Pr 'will result in a return without gosub
End

Pr:
Print "PIND.0 was/is low"
Return

```

CONFIGI2CDELAY

Action

Compiler directive that overrides the internal I2C delay routine.

Syntax

CONFIG I2CDELAY = value

Remarks

value A numeric value in the range of 1-255.
A higher value means a slower I2C clock.

For the I2C routines the clock rate is calculated depending on the used crystal. In order to make it work for all I2C devices the slow mode is used. When you have faster I2C devices you can specify a low value.

Seealso

CONFIG SCL »Page 100 , CONFIG SDA »Page 99

Example

```

CONFIG SDA = PORTB.7 'PORTB.7 is the SDA line
CONFIG I2CDELAY = 5
See I2C example for more details.

```

CONFIGINTx

Action

Configures the way the interrupts 4-7 will be triggered.

Syntax

CONFIG INTx = state

Where X can be 4 to 7.

Remarks

state

- LOW LEVEL to generate an interrupt while the pin is held low. Holding the pin low will generate an interrupt over and over again.
- FALLING to generate an interrupt on the falling edge.
- RISING to generate an interrupt on the rising edge..

Example

```

'-----
Config INT4 = LOW LEVEL

End

```

CONFIGKBD

Action

Configure the GETKBD() function and tell which port to use.

Syntax

CONFIG KBD = PORTx

Remarks

PORTx The name of the PORT to use such as PORTB or PORTD.

Seealso

GETKBD »Page 131

CONFIGLCD

Action

Configure the LCD display and override the compiler setting.

Syntax

CONFIG LCD = LCDtype

Remarks

LCDtype The type of LCD display used. This can be :
40 * 4, 16 * 1, 16 * 2, 16 * 4, 16 * 4, 20 * 2 or 20 * 4 or 16 * 1a
Default 16 * 2 is assumed.

When you have a 16 * 2 display, you don't have to use this statement.
The 16 * 1a is special. It is used for 2 * 8 displays that have the address of line 2, starting at location &H8.

Example

```
CONFIG LCD = 40 * 4
LCD "Hello"      'display on LCD
FOURTHLINE      'select line 4
LCD "4"         'display 4
END
```

CONFIGLCDBUS

Action

Configures the LCD data bus and overrides the compiler setting.

Syntax

CONFIG LCDBUS = constant

Remarks

Constant 4 for 4-bit operation, 8 for 8-bit mode (default)

Use this statement together with the \$LCD = address statement.
When you use the LCD display in the bus mode the default is to connect all the data lines. With the 4-bit mode, you only have to connect data lines d7-d4.

Seealso

CONFIG LCD »Page 97

Example

```
$LCD = &HC000      'address of enable and RS signal
$LCDRS = &H800     'address of enable signal
Config LCDBUS = 4 '4 bit mode
LCD "hello"
```

CONFIGLCDMODE

Action

Configures the LCD operation mode and overrides the compiler setting.

Syntax

CONFIG LCDMODE = type

Remarks

Type **PORT** will drive the LCD in 4-bit port mode and is the default.
In PORT mode you can choose different PIN's from different PORT's to connect to the lower 4 data lines of the LCD display. The RS and E can also be connected to a user selectable pin. This is very flexible since you can use pins that are not used by your design and makes the board layout simple. On the other hand, more software is necessary to drive the pins.

BUS will drive the LCD in bus mode and in this mode is meant when you have external RAM and so have an address and data bus on your system. The RS and E line of the LCD display can be connected to an address decoder. Simply writing to an external memory location select the LCD and the data is sent to the LCD display. This means the data-lines of the LCD display are fixed to the data-bus lines.

Use \$LCD »Page 69 = address and \$LCDRS »Page 72 = address, to specify the addresses that will enable the E and RS lines.

See also

CONFIG LCD »Page 97 , \$LCD »Page 69 , \$LCDRS »Page 72

Example

```
Config LCDMODE = PORT    'the report will show the settings
Config LCDBUS = 4        '4 bit mode
LCD "hello"
```

CONFIGLCDPIN

Action

Override the LCD-PIN select options.

Syntax

```
CONFIG LCDPIN = PIN , DB4= PN,DB5=PN, DB6=PN, DB7=PN, E=PN,
RS=PN
```

Remarks

PN The name of the PORT pin such as PORTB.2 for example.
DUM Actually a dummy you can leave out as long as you don't forget to include the = sign.

You can override the PIN selection from the Compiler Settings with this statement, so a second configuration lets you not choose more pins for a second LCD display.

See also

CONFIG LCD »Page 97

Example

```
CONFIG LCDPIN = PIN ,DB4= PORTB.1,DB5=PORTB.2,DB6=PORTB.3,
DB7=PORTB.4,E=PORTB.5,RS=PORTB.6
```

The above example must be typed on one line.

CONFIGSDA

Action

Overrides the SDA pin assignment from the Option Compiler Settings »Page 30.

Syntax

```
CONFIG SDA = pin
```

Remarks

Pin The port pin to which the I2C-SDA line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SDA pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project.CFG file.

See also

CONFIG SCL »Page 100 , CONFIG I2CDELAY »Page 95

Example

```
CONFIG SDA = PORTB.7        'PORTB.7 is the SDA line
See I2C example for more details.
```

CONFIGSCL

Action

Overrides the SCL pin assignment from the Option Compiler Settings »Page 30.

Syntax

```
CONFIG SCL = pin
```

Remarks

Pin The port pin to which the I2C-SCL line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SCL pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. Of course BASCOM-AVR also stores the settings in a project.CFG file.

See also

CONFIG SDA »Page 99 , CONFIG I2CDELAY »Page 95

Example

```
CONFIG SCL = PORTB.5        'PORTB.5 is the SCL line
```

CONFIG SPI

Action

Configures the SPI related statements.

Syntax

```
CONFIG SPI = SOFT, DIN = PIN, DOUT = PIN , SS = PIN, CLK = PIN
```

Remarks

SPI **SOFT** for software emulation of SPI, this lets you choose the PINS to use.

HARD for the internal SPI hardware, that will use fixed pins.
Not implemented yet.

DIN Data input or MISO. Pin is the pin number to use such as PORTB.0
DOUT Data output or MOSI. Pin is the pin number to use such as PORTB.1
SS Slave Select. Pin is the pin number to use such as PORTB.2
CLK Clock. Pin is the pin number to use such as PORTB.3

Seealso

SPIIN »Page 178, SPIOUT »Page 180, SPIINIT »Page 179

Example

```
Config SPI = SOFT, DIN = PORTB.0 , DOUT = PORTB.1, SS = PORTB.2, CLK
= PORTB.3
Dim var As Byte
SPIINIT 'Init SPI state and pins.
SPIOUT var , 1 'send 1 byte
```

CONFIG TIMER0

Action

Configure TIMER0.

Syntax

```
CONFIG TIMER0 = COUNTER ,EDGE=RISING/FALLING
CONFIG TIMER0 = TIMER ,PRESCALE= 1|8|64|256|1024
```

Remarks

TIMER0 is a 8 bit counter. See the hardware description of TIMER0.

When configured as a COUNTER:

EDGE You can select whether the TIMER will count on the falling or rising edge.

When configured as a TIMER:

PRESCALE The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter.
Valid values are 1 , 8, 64, 256 or 1024

When you use the CONFIG TIMER0 statement, the mode is remembered by the compiler and the TCCR0 register is set.

When you use the STOP TIMER0 statement, the TIMER is stopped.
When you use the START TIMER0 statement, the TIMER TCCR0 register is loaded with the last value that was configured with the CONFIG TIMER0 statement.

So before using the START »Page 180 and STOP »Page 181 TIMER0 statements, use the CONFIG statement first.

Example

```
'First you must configure the timer to operate as a counter or as a
timer
'Lets configure it as a COUNTER now
'You must also specify if it will count on a rising or falling edge
```

```
Config Timer0 = Counter , Edge = Rising
'Config Timer0 = Counter , Edge = falling
'unremark the line above to use timer0 to count on falling edge
```

```
'To get/set the value from the timer access the timer/counter
register
'let's reset it to 0
Tcnt0 = 0
```

```
Do
  Print Tcnt0
Loop Until Tcnt0 >= 10
'when 10 pulses are counter the loop is exited
```

```
'Now configure it as a TIMER
'The TIMER can have the system clock as an input or the system clock
divided
'by 8,64,256 or 1024
'The prescale parameter accpets 1,8,64,256 or 1024
Config Timer0 = Timer , Prescale = 1
```

```
'The TIMER is started now automatically
'You can STOP the timer with the following statement :
Stop Timer0
```

```
'Now the timer is stopped
'To START it again in the last configured mode, use :
Start Timer0

'Again you can access the value with the tcnt0 register
Print Tcnt0

'when the timer overflows, a flag named TOV0 in register TIFR is set
'You can use this to execute an ISR
'To reset the flag manual in non ISR mode you must write a 1 to the
bit position
'in TIFR:
Set Tifr.1

'The following code shows how to use the TIMER0 in interrupt mode
'The code is block remarked with '( en ')

'(

'Configure the timer to use the clock divided by 1024
Config Timer0 = Timer , Prescale = 1024

'Define the ISR handler
On Ovfo Tim0_isr
'you may also use TIMER0 for OVFO, it is the same

Do
    'your program goes here
Loop

'the following code is executed when the timer rolls over
Tim0_isr:
    Print "***";
Return

')

End
```

CONFIGTIMER1

Action

Configure TIMER1.

Syntax

```
CONFIG TIMER1 = COUNTER ,EDGE=RISING/FALLING ,NOICE
CANCEL=0/1, CAPTURE EDGE =RISING/FALLING
CONFIG TIMER1 = TIMER ,PRESCALE= 1|8|64|256|1024
CONFIG TIMER1 = PWM ,PWM = 8 ,COMPARE A PWM= CLEAR
UP/CLEAR DOWN/DISCONNECT ,COMPARE B PWM = (see A)
```

Remarks

The TIMER1 is a 16 bit counter. See the hardware description of TIMER1.

When configured as a COUNTER:

EDGE You can select whether the TIMER will count on the falling or rising edge.

CAPTURE EDGE You can choose to capture the TIMER registers to the INPUT CAPTURE registers
With the CAPTURE EDGE = FALLING/RISING, you can specify to capture on the falling or rising edge of pin ICP

NOICE CANCELING To allow noise canceling you can provide a value of 1.

When configured as a TIMER:

PRESCALE The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter.
Valid values are 1 , 8, 64, 256 or 1024

The TIMER1 also has two compare registers A and B

When the timer value matches a compare register, an action can be performed

COMPARE A The action can be:
SET will set the OC1X pin
CLEAR will clear the OC1X pin
TOGGLE will toggle the OC1X pin
DISCONNECT will disconnect the TIMER from output pin OC1X

And the TIMER can be used in PWM mode

You have the choice between 8, 9 or 10 bit PWM mode

Also you can specify if the counter must count UP or down after a match to the compare registers

Note that there are two compare registers A and B

PWM Can be 8, 9 or 10.
COMPARE A PWM PWM compare mode. Can be CLEAR UP or CLEAR DOWN

Example

```
-----
'                                     TIMER1.BAS
'-----

Dim W As Word

'The TIMER1 is a versatile 16 bit TIMER.
'This example shows how to configure the TIMER

'First like TIMER0 , it can be set to act as a TIMER or COUNTER
```

```
'Lets configure it as a TIMER that means that it will count and that
'the input is provided by the internal clock.
'The internal clock can be divided by 1,8,64,256 or 1024
Config Timer1 = Timer , Prescale = 1024

'You can read or write to the timer with the COUNTER1 or TIMER1
variable
W = Timer1
Timer1 = W

'To use it as a COUNTER, you can choose on which edge it is triggered
Config Timer1 = Counter , Edge = Falling
'Config Timer1 = Counter , Edge = Rising

'Also you can choose to capture the TIMER registers to the INPUT
CAPTURE registers
'With the CAPTURE EDGE = , you can specify to capture on the falling
or rising edge of pin ICP
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'To allow noise canceling you can also provide :
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling ,
Noise Canceling = 1

'to read the input capture register :
W = Capture1
'to write to the capture register :
Capture1 = W

'The TIMER also has two compare registers A and B
'When the timer value matches a compare register, an action can be
performed
Config Timer1 = Counter , Edge = Falling , Compare A = Set , Compare
B = Toggle
'SET , will set the OC1X pin
'CLEAR, will clear the OC1X pin
'TOGGLE, will toggle the OC1X pin
'DISCONNECT, will disconnect the TIMER from output pin OC1X

'To read write the compare registers, you can use the COMPARE1A and
COMPARE1B variables
Compare1a = W
W = Compare1a

'And the TIMER can be used in PWM mode
'You have the choice between 8,9 or 10 bit PWM mode
'Also you can specify if the counter must count UP or down after a
match
'to the compare registers
'Note that there are two compare registers A and B
Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B
Pwm = Clear Down

'to set the PWM registers, just assign a value to the compare A and B
registers
Compare1a = 100
```

```
Compare1b = 200

'Or for better reading :
Pwm1a = 100
Pwm1b = 200

End
```

CONFIGWAITSUART

Action

Compiler directive that specifies that software UART waits after sending the last byte.

Syntax

CONFIG WAITSUART =value

Remarks

value A numeric value in the range of 1-255.
A higher value means a longer delay in mS.

When the software UART routine are used in combination with serial LCD displays it can be convenient to specify a delay so the display can process the data.

Seealso

Example

See OPEN »Page 161 example for more details.

CONFIGWATCHDOG

Action

Configures the watchdog timer.

Syntax

CONFIG WATCHDOG = time

Remarks

Time The interval constant in mS the watchdog timer will count to before it will reset your program.

Possible settings :

16 , 32, 64 , 128 , 256 , 512 , 1024 and 2048.

When the WD is started, a reset will occur after the specified number of mS. With 2048, a reset will occur after 2 seconds, so you need to reset the WD in your programs periodically with the **RESET WATCHDOG** statement.

Seealso

START WATCHDOG »Page 180, STOP WATCHDOG »Page 181, RESET WATCHDOG »Page 169

Example

```

-----
'                                     (c) 2000 MCS Electronics
' WATCHD.BAS demonstrates the watchdog timer
-----
Config Watchdog = 2048                'reset after 2048 mSec
Start Watchdog                        'start the watchdog timer
Dim I As Word
For I = 1 To 10000
    Print I                            'print value
    ' Reset Watchdog
    'you will notice that the for next doesnt finish because of the
    reset
    'when you unmark the RESET WATCHDOG statement it will finish
    because the
    'wd-timer is reset before it reaches 2048 msec
Next
End

```

CONFIG PORT

Action

Sets the port or a port pin to the right data direction.

Syntax

CONFIG PORTx =state

CONFIG PINx.y = state

Remarks

state A constant that can be INPUT or OUTPUT. INPUT will set the data direction register to input for port X. OUTPUT will set the data direction to output for port X. You can also use a number for state. &B0001111,

will set the upper nibble to input and the lower nibble to output.

You can also set one port pin with the CONFIG PIN = state, statement.

Again, you can use INPUT, OUTPUT or a number. In this case the number can be only zero or one.

state : Constant.

The best way to set the data direction for more than 1 pin, is to use the CONFIG PORT, statement and not multiple lines with CONFIG PIN statements.

Example

```

-----
'                                     (c) 2000 MCS Electronics
-----
' file: PORT.BAS
' demo: PortB and PortD
-----
Dim A As Byte , Count As Byte

'Use port B for OUTPUT
Config Portb = Output
A = Portb                                'get
inputvalue of port 1
A = A And Portb
A = Pinb

Print A                                  'print
it

Portb = 10                               'set
port1 to 10
Portb = Portb And 2

Set Portb.0                              'set
bit 0 of port 1 to 1
Bitwait Portb.0 , Set                    'wait
until bit is set(1)

Incr Portb

Count = 0
Do
    Incr Count
    Portb = 1
    For A = 1 To 8
        Rotate Portb , Left              'rotate
    bits left
    Next
    'the following 2 lines do the same as the previous loop
    Portb = 1

```

```

Rotate Portb , Left , 8
Loop Until Count = 10
Print "Ready"

'note that the AVR port pins have a data direction register
'when you want to use a pin as an input it must be set low first
'you can do this by writing zeros to the DDRx:
'DDRB = &B11110000 'this will set portb1.0, portb.1, portb.2 and
portb.3 to use as inputs.

'So : when you want to use a pin as an input set it low first in the
DDRx!
' and when you want to use the pin as output, write a 1 first

End

```

COUNTER0 and COUNTER1

Action

Set or retrieve the internal 16 bit hardware register.

Syntax

COUNTER0 = <i>var</i>	TIMER0 can also be used
<i>var</i> = COUNTER0	
COUNTER1 = <i>var</i>	TIMER1 can also be used
<i>var</i> = COUNTER1	
CAPTURE1 = <i>var</i>	TIMER1 capture register
<i>var</i> = CAPTURE1	
COMPARE1A = <i>var</i>	TIMER1 COMPARE A register
<i>var</i> = COMPARE1A	
COMARE1B = <i>var</i>	TIMER1 COMPARE B register
<i>var</i> = COMPARE1B	
PWM1A = <i>var</i>	TIMER1 COMPAREA register. (Is used for PWM)
<i>var</i> = PWM1A	
PWM1B = <i>var</i>	TIMER1 COMPARE B register. (Is used for PWM)
<i>var</i> = PRM1B	

Remarks

Var A byte, Integer/Word variable or constant that is assigned to the register or is read from the register.

Because the above 16 bit register pairs must be accessed somewhat differently than you may expect, they are implemented as variables. The exception is TIMER0/COUNTER0, this is a normal 8 bit register and is

supplied for compatibility with the syntax.

When the CPU reads the low byte of the register, the data of the low byte is sent to the CPU and the data of the high byte is placed in a temp register. When the CPU reads the data in the high byte, the CPU receives the data in the temp register.

When the CPU writes to the high byte of the register pair, the written data is placed in a temp register. Next when the CPU writes the low byte, this byte of data is combined with the byte data in the temp register and all 16 bits are written to the register pairs. So the MSB must be accessed first.

All of the above is handled automatically by BASCOM when accessing the above registers.

Note that the available registers may vary from chip to chip. The BASCOM documentation used the 8515 to describe the different hardware registers.

CPEEK

Action

Returns a byte stored in code memory.

Syntax

var = CPEEK(*address*)

Remarks

Var Numeric variable that is assigned with the content of the program memory at *address*

Address Numeric variable or constant with the address location

There is no CPOKE statement because you can not write into program memory.

Seealso

PEEK »Page 163, POKE »Page 163, INP »Page 144, OUT »Page 162

Example

```

-----
(c) 1998-2000 MCS Electronics
PEEK.BAS
demonstrates PEEK, POKE, CPEEK, INP and OUT
-----

```

```

Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31
    B1 = Peek(i)
memory
    Print Hex(b1) ; " ";
    'Poke I , 1
memory(register)
Next
Print
'be careful when writing into internal memory !!

'now dump a part of the code-memory(program)
For I = 0 To 255
    B1 = Cpeek(i)
memory
    Print Hex(b1) ; " ";
Next
'note that you can not write into code memory!!

Out &H8000 , 1
8000
B1 = Inp(&H8000)
Print B1

```

CRYSTAL

Action

Special byte variable that can be used with software UART routine to change the baudrate during runtime.

Syntax

CRYSTAL = var

Remarks

With the software UART you can generate good baudrates. But chips such as the ATtiny22 have an internal 1 MHz clock. The clock frequency can change during runtime by influence of temperature or voltage. The crystal variable can be changed during runtime to change the baudrate. You must DIM »Page 120ension this special variable before you can use it!

Some values for 1 MHz internal clock :

Crystal = 66 'for 2400 baud

Crystal = 31 'for 4800 baud

Crystal = 14 'for 9600 baud

Seealso

OPEN »Page 161 , CLOSE »Page 161

Example

```

Dim crystal As Byte, B as byte
Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 , "serial output"
crystal = 16
Print #1, "serial output"
Close #1
End

```

CURSOR

Action

Set the LCD Cursor State.

Syntax

CURSOR ON / OFF BLINK / NOBLINK

Remarks

You can use both the ON or OFF and BLINK or NOBLINK parameters. At power up the cursor state is ON and NOBLINK.

Seealso

DISPLAY »Page 125 , LCD »Page 147

Example

```

Dim a As Byte
a = 255
LCD a
CURSOR OFF 'hide cursor
Wait 1 'wait 1 second
CURSOR BLINK 'blink cursor
End

```

DATA

Action

Specifies constant values to be read by subsequent READ statements.

Syntax

DATA var [, varn]

Remarks

Var Numeric or string constant.

The DATA related statements use the internal registers pair R8 and R9 to

store the data pointer.

To store a " sign on the data line, you can use :

```
DATA $34
```

The \$-sign tells the compiler that the ASCII value will follow of the character.

You can also use this to store special characters that can't be written by the editor such as chr(7)

Because the DATA statements allows you to generate an EEP file to store in EEPROM, the \$DATA »Page 65 and \$EEPROM »Page 67 directives have been added. Read the description of these directives to learn more about the DATA statement.

The DATA statements must not be accessed by the flow of your program because the DATA statements are converted to the byte representation of the DATA.

When your program flow enters the DATA lines, unpredictable results will occur.

So as in QB, the DATA statement is best be placed at the end of your program or in a place that program flow will no enter.

For example this is fine:

```
Print "Hello"
Goto jump
DATA "test"
```

Jump:

'because we jump over the data lines there is no problem.

The following example will case some problems:

```
Dim S As String * 10
Print "Hello"
Restore lbl
Read S
DATA "test"
Print S
```

Difference with QB

Integer and Word constants must end with the % -sign.

Long constants must end with the &-sign.

Single constants must end with the !-sign.

Seealso

READ »Page 167 , RESTORE »Page 170 , \$DATA »Page 65 , \$EEPROM »Page 67

Example

```
-----
'                                READDATA.BAS
'                                Copyright 2000 MCS Electronics
'-----

Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long
Restore Dta1                                'point to stored data
For Count = 1 To 3                          'for number of data items
    Read B1 : Print Count ; " " ; B1
Next

Restore Dta2                                'point to stored data
For Count = 1 To 2                          'for number of data items
    Read A : Print Count ; " " ; A
Next

Restore Dta3
Read S : Print S
Read S : Print S

Restore Dta4
Read L : Print L                            'long
type

End

Dta1:
Data &B10 , &HFF , 10

Dta2:
Data 1000% , -1%

Dta3:
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement

Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is
used
'for the READ statement
```

DEBOUNCE

Action

Debounce a port pin connected to a switch.

Syntax

DEBOUNCE Px.y , state , label [, SUB]

Remarks

Px.y A port pin like PINB.0 , to examine.
state 0 for jumping when PINx.y is low , 1 for jumping when PINx.y is high
label The label to GOTO when the specified state is detected
SUB The label to GOSUB when the specified state is detected

When you specify the optional parameter SUB, a GOSUB to label is performed instead of a GOTO.

The DEBOUNCE statements wait for a port pin to get high(1) or low(0). When it does it waits 25 mS and checks again (eliminating bounce of a switch)

When the condition is still true and there was no branch before, it branches to the label.

When DEBOUNCE is executed again, the state of the switch must have gone back in the original position before it can perform another branch.

Each DEBOUNCE statement which use a different port uses 1 BIT of the internal memory to hold its state.

Seealso

CONFIG DEBOUNCE »Page 94

Example

```

-----
'
'          DEBOUN.BAS
'          Demonstrates DEBOUNCE
'
-----
CONFIG DEBOUNCE = 30 'when the config statement is not used a
default of 25mS will be used

'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1)
Debounce Pind.0 , 0 , Pr , Sub
Debounce Pind.0 , 0 , Pr , Sub
'
'          ^----- label to branch to
'          ^----- Branch when P1.0 goes low(0)
'          ^----- Examine P1.0

'When Pind.0 goes low jump to subroutine Pr
'Pind.0 must go high again before it jumps again
'to the label Pr when Pind.0 is low

Debounce Pind.0 , 1 , Pr          'no
branch
Debounce Pind.0 , 1 , Pr          'will

```

```

result in a return without gosub
End

```

```

Pr:
  Print "PIND.0 was/is low"
Return

```

DECR

Action

Decrements a variable by one.

Syntax

DECR var

Remarks

Var Variable to decrement.

var : Byte, Integer, Word, Long, Single.

There are often situations where you want a number to be decreased by 1. The Decr statement is provided for compatibility with BASCOM-8051.

Seealso

INCR »Page 143

Example

```

-----
'
'          (c) 1997,1998 MCS Electronics
'
-----
'   file:  DEC.R.BAS
'   demo:  DEC.R
'
-----
Dim A As Byte

A = 5          'assign value to a
Decr A        'dec (by one)
Print A      'print it
End

```

DECLARE FUNCTION

Action

Declares a user function.

Syntax

DECLARE FUNCTION TEST[([BYREF/BYVAL] var as type)] As type

Remarks

test Name of the function.
Var Name of the variable(s).
Type Type of the variable(s) and of the result. Byte,Word/Integer, Long or String.

When BYREF or BYVAL is not provided, the parameter will be passed by reference.

Use BYREF to pass a variable by reference with its address.

Use BYVAL to pass a copy of the variable.

See the CALL »Page 88 statement for more details.

You must declare each function before writing the function or calling the function.

Seealso

CALL »Page 88, SUB »Page 184 , FUNCTION »Page 116

Example

```

-----
'
'                               (c) 2000 MCS Electronics
'                               Demonstration of user function
'
-----

'A user function must be declare before it can be used.
'A function must return a type
Declare Function Myfunction(byval I As Integer , S As String) As Integer
'The byval paramter will pass the parameter by value so the original value
'will not be changed by the function

Dim K As Integer
Dim Z As String * 10
Dim T As Integer
'assign the values
K = 5
Z = "123"

T = Myfunction(k , Z)
Print T
End

Function Myfunction(byval I As Integer , S As String) As Integer
'you can use local variables in subs and functions
Local P As Integer

P = I

'because I is passed by value, altering will not change the

```

```

original
'variable named k
I = 10

P = Val(s) + I

'finally assign result
'Note that the same data type must be used !
'So when declared as an Integer function, the result can only be
'assigned with an Integer in this case.
Myfunction = P
End Function

```

DECLARE SUB

Action

Declares a subroutine.

Syntax

DECLARE SUB TEST[([BYREF/BYVAL] var as type)]

Remarks

test Name of the procedure.
Var Name of the variable(s).
Type Type of the variable(s). Byte, Word/Integer, Long or String.
When BYREF or BYVAL is not provided, the parameter will be passed by reference.
Use BYREF to pass a variable by reference with its address.
Use BYVAL to pass a copy of the variable.
See the CALL »Page 88 statement for more details.

You must declare each sub before writing or calling the sub procedure.

Seealso

CALL »Page 88, SUB »Page 184

Example

```

Dim a As Byte, b1 As Byte, c As Byte
Declare Sub Test(a As Byte)
a = 1 : b1 = 2: c = 3

Print a ; b1 ; c

Call Test(b1)
Print a ;b1 ; c
End

Sub Test(a as byte)
Print a ; b1 ; c

```

End Sub

DEFxxx

Action

Declares all variables that are not dimensioned of the DefXXX type.

Syntax

```
DEFBIT b          Define BIT
DEFBYTE c Define BYTE
DEFINT I          Define INTEGER
DEFWORD x Define WORD
DEFLNG I          Define LONG
DEFSNG s Define SINGLE
```

Difference with QB

QB allows you to specify a range like DEFINT A - D. BASCOM doesn't support this.

Example

```
Defbit b : DefInt c      'default type for bit and integers
Set b1                'set bit to 1
c = 10                 'let c = 10
```

DEFLCDCHAR

Action

Define a custom LCD character.

Syntax

```
DEFLCDCHAR char,r1,r2,r3,r4,r5,r6,r7,r8
```

Remarks

char Constant representing the character (0-7).
r1-r8 The row values for the character.

You can use the LCD designer »Page 26 to build the characters.

It is important that a CLS follows the DEFLCDCHAR statement(s).

Special characters can be printed with the Chr »Page 90() function.

Seealso

Tools LCD designer »Page 26

Example

```
DefLCDchar 0,1,2,3,4,5,6,7,8 'define special character
Cls                          'select LCD DATA RAM
LCD Chr(0)                   'show the character
End
```

DELAY

Action

Delay program execution for a short time.

Syntax

```
DELAY
```

Remarks

Use DELAY to wait for a short time.
The delay time is ca. 1000 microseconds.

Seealso

WAIT , WAITMS

Example

```
P1 = 5                'write 5 to port 1
DELAY                'wait for hardware to be ready
```

DIM

Action

Dimension a variable.

Syntax

```
DIM var AS [XRAM/IRAM] type [AT location]
```

Remarks

var Any valid variable name such as b1, i or longname. var can also be an array : ar(10) for example.

type Bit, Byte, Word, Integer, Long, Single or String
XRAM Specify XRAM to store variable into external memory
SRAM Specify SRAM to store variable into internal memory (default)

ERAM Specify ERAM to store the variable into EEPROM

A string variable needs an additional length parameter:

```
Dim s As XRAM String * 10
```

In this case, the string can have a maximum length of 10 characters.

Note that BITS can only be stored in internal memory.

The optional AT parameter lets you specify where in memory the variable must be stored. When the memory location already is occupied, the first free memory location will be used.

Difference with QB

In QB you don't need to dimension each variable before you use it. In BASCOM you must dimension each variable before you use it. This makes for safer code.

In addition, the XRAM/SRAM/ERAM options are not available in QB.

See Also

CONST »Page 122, LOCAL »Page 152

Example

```

-----
'                                     (c) 2000 MCS Electronics
-----
'   file: DIM.BAS
'   demo: DIM
-----
Dim B1 As Bit                               'bit
can be 0 or 1
Dim A As Byte                               'byte
range from 0-255
Dim C As Integer
'integer range from -32767 - +32768
Dim L As Long
Dim W As Word
Dim S As String * 10                         'length
can be up to 10 characters

'new feature : you can specify the address of the variable
Dim K As Integer At 120
'the next dimensioned variable will be placed after variable s
Dim Kk As Integer

'Assign bits
B1 = 1                                     'or
Set B1                                     'use set

'Assign bytes
A = 12

```

```

A = A + 1

'Assign integer
C = -12
C = C + 100
Print C

W = 50000
Print W

'Assign long
L = 12345678
Print L

'Assign string
S = "Hello world"
Print S

End

```

CONST

Action

Declares a symbolic constant.

Syntax

```

CONST symbol = numconst
CONST symbol = stringconst

```

Remarks

Symbol	The name of the symbol.
Numconst	The numeric value to assign to the symbol.
Stringconst	The string to assign to the symbol

Assigned constants consume no program memory because they only serve as a reference to the compiler.

The compiler will replace all occurrences of the symbol with the assigned value.

See also

ALIAS »Page 83

Difference with BASCOM-8051

In BASCOM-8051 only numeric constants can be used. The syntax is also different

Example

DISPLAY

Action

Turn LCD display on or off.

Syntax

DISPLAY ON / OFF

Remarks

The display is turned on at power up.

See also

LCD »Page 147

Example

```
Dim a as byte
a = 255
LCD a
DISPLAY OFF
Wait 1
DISPLAY ON
End
```

DO-LOOP

Action

Repeat a block of statements until condition is true.

Syntax

DO

statements

LOOP [UNTIL expression]

Remarks

You can exit a DO..LOOP with the EXIT DO statement.

The DO-LOOP is always performed at least once.

See also

EXIT »Page 128 , WHILE-WEND »Page 189 , FOR-NEXT »Page 128

Example

```
Dim A As Byte
DO
    A = A + 1
    PRINT A
LOOP UNTIL A = 10
'start the loop
'increment A
'print it
'Repeat loop until A = 10
```

Print A

ELSE

Action

Executed if the IF-THEN expression is false.

Syntax

ELSE

Remarks

You don't have to use the ELSE statement in an IF THEN .. END IF structure.

You can use the ELSEIF statement to test for another condition.

```
IF a = 1 THEN
...
ELSEIF a = 2 THEN
..
ELSEIF b1 > a THEN
...
ELSE
...
END IF
```

See also

IF , END IF , SELECT

Example

```
A = 10
IF A > 10 THEN
    PRINT " A >10"
ELSE
    PRINT " A not greater than 10"
END IF
'let a = 10
'make a decision
'this will not be printed
'alternative
'this will be printed
```

ENABLE

Action

Enable specified interrupt.

Syntax

ENABLE interrupt

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt
COMPARE1A,OC1A	TIMER1 OUTPUT COMPARE A interrupt
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt
SPI	SPI interrupt
URXC	Serial RX complete interrupt
UDRE	Serial data register empty interrupt
UTXC	Serial TX complete interrupt
SERIAL	Disables URXC, UDRE and UTXC
ACI	Analog comparator interrupt
ADC	A/D converter interrupt

By default all interrupts are disabled.

To enable the enabling and disabling of interrupts use ENABLE INTERRUPTS.

Other chips might have additional interrupt sources such as INT2, INT3 etc.

Seealso

DISABLE »Page 123

Example

```
ENABLE INTERRUPTS      'allow interrupts to be set
ENABLE TIMER1          'enables the TIMER1 interrupt
```

END

Action

Terminate program execution.

Syntax

END

Remarks

STOP can also be used to terminate a program.

When an END or STOP statement is encountered, all interrupts are disabled and a never-ending loop is generated.

Seealso

STOP »Page 181

Example

```
PRINT "Hello"          'print this
END                    'end program execution
```

EXIT

Action

Exit a FOR..NEXT, DO..LOOP, WHILE ..WEND, SUB..END SUB or FUNCTION..END FUNCTION.

Syntax

```
EXIT FOR
EXIT DO
EXIT WHILE
EXIT SUB
EXIT FUNCTION
```

Remarks

With the EXIT ... statement you can exit a structure at any time.

Example

```
IF a >= b1 THEN        'some silly code
DO                      'begin a DO..LOOP
  A = A + 1             'incr a
  IF A = 100 THEN      'test for a = 100
    EXIT DO            'exit the DO..LOOP
  END IF               'end the IF..THEN
LOOP                   'end the DO
END IF                  'end the IF..THEN
```

FOR-NEXT

Action

Execute a block of statements a number of times.

Syntax

```
FOR var = start TO/DOWNTO end [STEP value]
```

Remarks

var The variable counter to use
start The starting value of the variable var
end The ending value of the variable var
value The value var is increased/decreased with each time NEXT is

encountered.

For incremental loops, you must use TO.

For decremental loops, you must use DOWNTO.

You must end a FOR structure with the NEXT statement.

The use of STEP is optional. By default, a value of 1 is used.

Seealso

EXIT FOR »Page 128

Example

```
'-----
'                                     (c) 2000 MCS Electronics
'-----
'   file: FOR_NEXT.BAS
'   demo: FOR, NEXT
'-----

Dim A As Byte , B1 As Byte , C As Integer

For A = 1 To 10 Step 2
  Print "This is A " ; A
Next A

Print "Now lets try DOWNTO"
For C = 10 Downto -5
  Print "This is C " ; C
Next

Print "You can also nest FOR..NEXT statements."
For A = 1 To 10
  Print "This is A " ; A
  For B1 = 1 To 10
    Print "This is B1 " ; B1
  Next
Next
' note that you do not have
to specify the parameter
Next A

End
```

FOURTHLINE

Action

Set LCD cursor to the start of the fourth line.

Syntax

FOURTHLINE

Remarks

Only valid for LCD displays with 4 lines.

Seealso

HOME »Page 138 , UPPERLINE »Page 186 , LOWERLINE »Page 156 ,

THIRDLINE »Page 184 ,LOCATE »Page 154

Example

```
Dim a as byte
a = 255
LCD a
Fourthline
LCD a
Upperline
END
```

FUSING

This statement is for the prof edition only and will be described later.

GETADC

Action

Retrieves the analog value from channel 0-7.

Syntax

var = GETADC(channel)

Remarks

var The variable that is assigned with the A/D value
channel The channel to measure

The GETADC() function is only intended for the AVR90S8535 or other chips that have a built-in A/D converter.

Seealso

Example

```
'-----
'                                     ADC.BAS
'   demonstration of GETADC() function for 8535 micro
'-----
'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function
```

```
'The prescaler divides the internal clock by 2,4,8,15,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End
```

GETKBD

Action

Scans a 4x4 matrix keyboard and return the value of the key pressed.

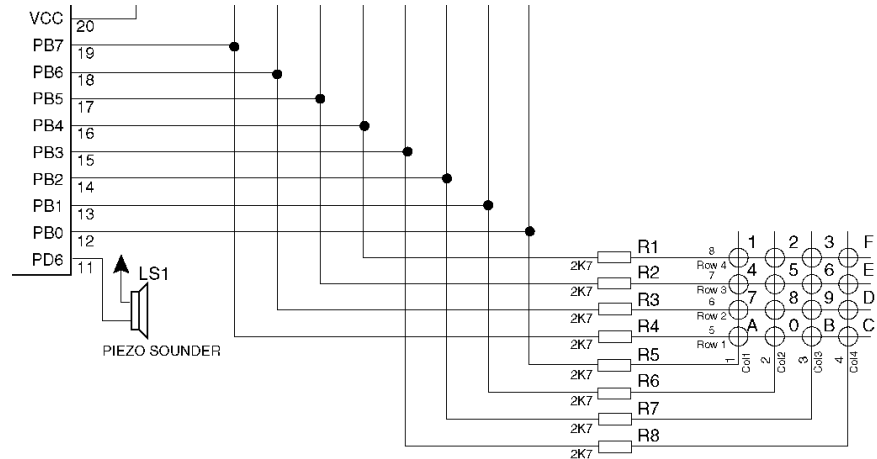
Syntax

var = GETKBD()

Remarks

var The variable that is assigned with the value read from the keyboard

The GETKBD() function can be attached to a port of the uP. You can define the port with the CONFIG KBD statement. A schematic for PORTB is shown below



Note that the port pins can be used for other tasks as well.

Seealso

CONFIG KBD

Example

```
CONFIG KBD = PORTB
Dim B As Byte
Do
  B = KETKBD()
  Print B
Loop
End
```

GETRC0

This function has not been tested yet!

Action

Retrieves the value of a resistor or a capacitor.

Syntax

var = GETRC0(pin)

Remarks

Var The variable that is assigned with the value.

Pin The port pin for the R/C is connection.

Seealso

Example

GETRC5

Action

Retrieves the RC5 remote code from a IR transmitter.

Syntax

GETRC5(address, command)

Uses

TIMER0

Remarks

address The RC5 address
 command The RC5 command.

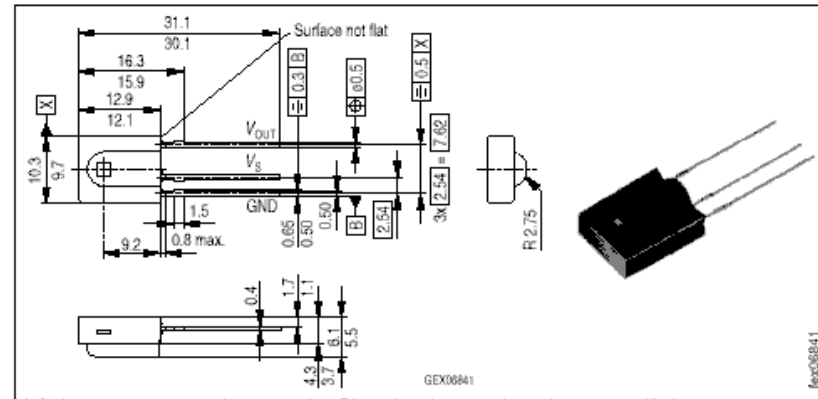
This statement used the AVR 410 application note. Since a timer is needed for accurate delays and background processing TIMER0 is used by this statement.

Also the interrupt of TIMER0 is used by this statement. TIMER0 can be used by your application since the values are preserved by the statement but a delay can occur. The interrupt can not be reused.

The SFH506-36 is used from Siemens. Other types can be used as well.

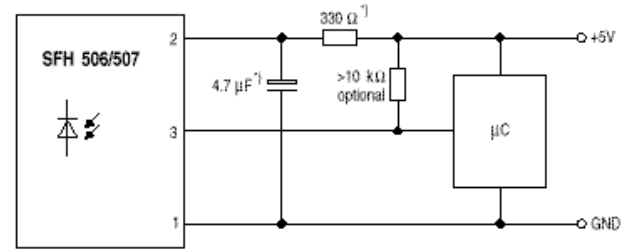
IR-Empfänger/Demodulator-Baustein IR-Receiver/Demodulator Device

SFH 506



Maße in mm, wenn nicht anders angegeben/Dimensions in mm, unless otherwise specified.

For a good operation use the following values for the filter.



¹ only necessary to suppress power supply disturbances

Most audio and video systems are equipped with an infra-red remote control. The RC5 code is a 14-bit word bi-phase coded signal.

The two first bits are start bits, always having the value 1.

The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.

Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on.

The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

Seealso

Example

```

-----
'
'          RC5.BAS
'          (c) 2000 MCS Electronics
'          based on Atmel AVR410 application note
-----
'This example shows how to decode RC5 remote control signals
'with a SFH506-35 IR receiver.

'Connect to input to PIND.2 for this example
'The GETRC5 function uses TIMER0 and the TIMER0 interrupt.
'The TIMER0 settings are restored however so only the interrupt can
not
'be used anymore for other tasks

'tell the compiler which pin we want to use for the reciever input

Config Rc5 = Pind.2

'the interrupt routine is inserted automatic but we need to make it
occur
'so enable the interrupts
Enable Interrupts

'reserve space for variables
Dim Address As Byte , Command As Byte

Do
'now check if a key on the remote is pressed
'Note that at startup all pins are set for INPUT
'so we dont set the direction here
'If the pins is used for other input just unremark the next line
'Config Pind.2 = Input
Getrc5(address , Command)

'we check for the TV address and that is 0
If Address = 0 Then
'clear the toggle bit
'the toggle bit toggles on each new received command
Command = Command And &B10111111
Print Address ; " " ; Command
End If
Loop
End

```

GOSUB

Action

Branch to and execute subroutine.

Syntax

GOSUB label

Remarks

Label The name of the label where to branch to.

With GOSUB, your program jumps to the specified label, and continues execution at that label.

When it encounters a RETURN statement, program execution will continue after the GOSUB statement.

Seealso

GOTO »Page 136 , CALL »Page 88 , RETURN »Page 170

Example

```

GOSUB Routine           'branch to routine
Print "Hello"          'after being at 'routine' print this
END                    'terminate program

Routine:                'this is a subroutine
    x = x + 2           'perform some math
    PRINT X             'print result
RETURN                 'return

```

GOTO

Action

Jump to the specified label.

Syntax

GOTO label

Remarks

Labels can be up to 32 characters long.

When you use duplicate labels, the compiler will give you a warning.

Seealso

GOSUB »Page 135

Example

```

Start:                  'a label must end with a colon
A = A + 1               'increment a
IF A < 10 THEN          'is it less than 10?
    GOTO Start          'do it again
END IF                  'close IF
PRINT " Ready"         'that is it

```

HEX

Action

Returns a string representation of a hexadecimal number.

Syntax

```
var = Hex( x )
```

Remarks

var A string variable.
X A numeric variable of data type Byte, Integer, Word or Long.

Seealso

HEXVAL »Page 137

Example

```
Dim a as Byte, S as String * 10
a = 123
s = Hex(a)
Print s
Print Hex(a)
End
```

HEXVAL

Action

Convert string representing a hexadecimal number into a numeric variable.

Syntax

```
var = HEXVAL( x )
```

Remarks

var The numeric variable that must be assigned.
X The hexadecimal string that must be converted.

Difference with QB

In QB you can use the VAL() function to convert hexadecimal strings. But since that would require an extra test for the leading &H signs that are required in QB, a separate function was designed.

Seealso

HEX »Page 137, VAL »Page186, STR »Page 182

Example

```
Dim a as Integer, s as string * 15
s = "A"
a = Hexval(s) : Print a
End
```

HIGH

Action

Retrieves the most significant byte of a variable.

Syntax

```
var = HIGH( s )
```

Remarks

Var The variable that is assigned with the MSB of var S.
S The source variable to get the MSB from.

Seealso

LOW »Page 156

Example

```
Dim I As Integer, Z As Byte
I = &H1001
Z = High(I) ' is 16
```

HOME

Action

Place the cursor at the specified line at location 1.

Syntax

HOME UPPER / LOWER /THIRD / FOURTH

Remarks

If only HOME is used than the cursor will be set to the upperline. You can also specify the first letter of the line like: HOME U

Seealso

CLS »Page 91 , LOCATE »Page 154

Example

```
Lowerline
LCD "Hello"
Home Upper
LCD "Upper"
```

I2CRECEIVE

Action

Receives data from an I2C serial device.

Syntax

```
I2CRECEIVE slave, var
I2CRECEIVE slave, var ,b2W, b2R
```

Remarks

- Slave** A byte, Word/Integer variable or constant with the slave address from the I2C-device.
- Var** A byte or integer/word variable that will receive the information from the I2C-device.
- b2W** The number of bytes to write.
Be cautious not to specify too many bytes!
- b2R** The number of bytes to receive.
Be cautious not to specify too many bytes!

You can specify the base address of the slave chip because the read/write bit is set/reset by the software.

Seealso

I2CSEND »Page 139

Example

```
x = 0 'reset variable
slave = &H40 'slave address of a PCF 8574 I/O IC
I2CRECEIVE slave, x 'get the value
PRINT x 'print it
```

```
Dim buf(10) as Byte
buf(1) = 1 : buf(2) = 2
I2CRECEIVE slave, buf(), 2, 1'send two bytes and receive one byte
Print buf(1) 'print the received byte
```

I2CSEND

Action

Send data to an I2C-device.

Syntax

```
I2CSEND slave, var
I2CSEND slave, var , bytes
```

Remarks

- slave** The slave address off the I2C-device.
- var** A byte, integer/word or numbers that holds the value, which will be, send to the I2C-device.
- bytes** The number of bytes to send.

Seealso

I2CRECEIVE »Page 139

Example

```
x = 5 'assign variable to 5
Dim ax(10) As Byte
Const slave = &H40 'slave address of a PCF 8574 I/O IC
I2CSEND slave, x 'send the value or

For a = 1 to 10
    ax(a) = a 'Fill dataspace
Next
bytes = 10
I2CSEND slave,ax(),bytes
END
```

I2CSTART, I2CSTOP, I2CRBYTE, I2CWBYTE

Action

- I2CSTART generates an I2C start condition.
- I2CSTOP generates an I2C stop condition.
- I2CRBYTE receives one byte from an I2C-device.
- I2CWBYTE sends one byte to an I2C-device.

Syntax

```
I2CSTART
I2CSTOP
I2CRBYTE var, ack/nack
I2CWBYTE val
```

Remarks

- var** A variable that receives the value from the I2C-device.
- ack/nack** Specify ACK if there are more bytes to read.
Specify NACK if it is the last byte to read.

val A variable or constant to write to the I2C-device.

These statements are provided as an addition to the I2CSEND and I2CRECEIVE functions.

Seealso

I2CRECEIVE »Page 139 , I2CSEND »Page 139

Example

```
----- Writing and reading a byte to an EEPROM 2404 -----
--
DIM a As Byte
Const adresW = 174      'write of 2404
Const adresR = 175      'read address of 2404
I2CSTART                'generate start
I2CWBYTE  adresW  'send slave address
I2CWBYTE  1        'send address of EEPROM
I2CWBYTE  3        'send a value
I2CSTOP              'generate stop
WaitMS 10           'wait 10 mS because that is the time that
the chip needs to write the data

-----now read the value back into the var a -----
-----
I2CSTART                'generate start
I2CWBYTE  adresW  'write slave address
I2CWBYTE  1        'write adres of EEPROM to read
I2CSTART                'generate repeated start
I2CWBYTE  adresR  'write slave address of EEPROM
I2CRBYTE  a,nack  'receive value into a. nack means last byte to
receive
I2CSTOP                'generate stop
PRINT a                'print received value
END
```

IDLE

Action

Put the processor into the idle mode.

Syntax

IDLE

Remarks

In the idle mode, the system clock is removed from the CPU but not from the interrupt logic, the serial port or the timers/counters.

The idle mode is terminated either when an interrupt is received or upon system reset through the RESET pin.

Seealso

POWERDOWN »Page 164

Example

IDLE

IF-THEN-ELSE-ENDIF

Action

Allows conditional execution or branching, based on the evaluation of a Boolean expression.

Syntax

IF expression **THEN**

[**ELSEIF** expression **THEN**]

[**ELSE**]

END IF

Remarks

Expression Any expression that evaluates to true or false.

The one line version of IF can be used :
IF expression **THEN** statement [**ELSE** statement]
 The use of [ELSE] is optional.

Also new is the ability to test on bits :
IF var.bit = 1 **THEN**

Seealso

ELSE »Page 126

Example

```
DIM A AS INTEGER
A = 10
IF A = 10 THEN                                'test expression
    PRINT "This part is executed." 'this will be printed
ELSE
    PRINT "This will never be executed." 'this not
END IF
IF A = 10 THEN PRINT "New in BASCOM"
IF A = 10 THEN GOTO LABEL1 ELSE PRINT "A<>10"
LABEL1:
```

```
REM The following example shows enhanced use of IF THEN
IF A.15 = 1 THEN 'test for bit
```

```

PRINT "BIT 15 IS SET"
END IF
REM the following example shows the 1 line use of IF THEN [ELSE]
IF A.15 = 0 THEN PRINT "BIT 15 is cleared" ELSE PRINT "BIT 15 is set"

```

INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

Var Any numeric variable.

Seealso

DECR »Page 116

Example

```

DO 'start loop
  INCR a 'increment a by 1
  PRINT a 'print a
LOOP UNTIL a > 10 'repeat until a is greater than 10

```

INKEY

Action

Returns the ASCII value of the first character in the serial input buffer.

Syntax

var = **INKEY**

Remarks

var Byte, Integer, Word, Long or String variable.

If there is no character waiting, a zero will be returned.

The INKEY routine can be used when you have a RS-232 interface on your uP.

The RS-232 interface can be connected to a comport of your computer.

Seealso

WAITKEY »Page 188

Example

```

DO 'start loop
  A = INKEY() 'look for character
  IF A > 0 THEN 'is variable > 0?
    PRINT A 'yes , so print it
  END IF
LOOP 'loop forever

```

INP

Action

Returns a byte read from a hardware port or any internal memory location.

Syntax

var = **INP**(address)

Remarks

var Numeric variable that receives the value.
address The address where to read the value from. (0- &HFFFF)

The PEEK() function will read only the lowest 32 memory locations (registers). The INP() function can read from any memory location since the AVR has a linear memory model.

Seealso

OUT »Page 162

Example

```

Dim a As Byte
a = INP(&H8000) 'read value that is placed on databus(d0-d7) at
               'hex address 8000
PRINT a
END

```

INPUTBIN

Action

Read binary values from the serial port.

Syntax

INPUTBIN var1 [,var2]
INPUTBIN #channel , var1 [,var2]

Remarks

var1 The variable that is assigned with the characters from the serial port.

var2 An optional second (or more) variable that is assigned with the characters from the serial.

The channel is for use with the software UART routine and must be use with OPEN and CLOSE. »Page 161

The number of bytes to read depends on the variable you use. When you use a byte variable, 1 character is read from the serial port. An integer will wait for 2 characters and an array will wait until the whole array is filled.

Note that the INPUTBIN statement doesn't wait for a <RETURN> but just for the number of bytes.

Seealso

PRINTBIN »Page 166

Example

```
Dim a as Byte, C as Integer
INPUTBIN a, c      'wait for 3 characters
End
```

INPUTHEX

Action

Allows input from the keyboard during program execution.

Syntax

```
INPUTHEX [" prompt" ], var [, varn ] [ NOECHO ]
```

Remarks

prompt character. An optional string constant printed before the prompt character.

Var,varn A numeric variable to accept the input value.

NOECHO Disables input echoed back to the Comport.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP.

The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as input device. You can also use the build in terminal emulator.

If **var** is a byte then the input can be maximum 2 characters long.
If **var** is an integer/word then the input can be maximum 4 characters long.
If **var** is a long then the input can be maximum 8 characters long.

Difference with QB

In QB you can specify &H with INPUT so QB will recognise that a hexadecimal string is being used.

BASCOSM implements a new statement: INPUTHEX.

Seealso

INPUT »Page 146

Example

```
Dim x As Byte
INPUTHEX "Enter a number " , x      'ask for input
```

INPUT

Action

Allows input from the keyboard during program execution.

Syntax

```
INPUT [" prompt" ], var [, varn ] [ NOECHO ]
```

Remarks

prompt An optional string constant printed before the prompt character.

Var,varn A variable to accept the input value or a string.

NOECHO Disables input echoed back to the Comport.

The INPUT routine can be used when you have an RS-232 interface on your uP.

The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as an input device.

You can also use the built-in terminal emulator.

Difference with QB

In QB you can specify &H with INPUT so QB will recognise that a hexadecimal string is being used.

BASCOSM implements a new statement : INPUTHEX.

Seealso

INPUTHEX »Page 145 , PRINT »Page 165

Example

```

-----
'                               (c) 1997,1998 MCS Electronics
-----
'   file: INPUT.BAS
'   demo: INPUT, INPUTHEX
-----
'To use another baudrate and crystalfrequency use the
'metastatements $BAUD = and $CRYSTAL =
$baud = 1200                'try 1200 baud for example
$crystal = 12000000         '12 MHz

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15       'only for uP with XRAM
support

Input "Use this to ask a question " , V
Input B1                   'leave out for no question

Input "Enter integer " , C
Print C

Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho             'suppress echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho             'without echo
Print S
End

```

LCD

Action

Send constant or variable to LCD display.

Syntax

LCD x

Remarks

X Variable or constant to display.

More variables can be displayed separated by the ; -sign
LCD a ; b1 ; "constant"
The LCD statement behaves just like the PRINT statement.

Seealso

\$LCD »Page 69 , \$LCDRS »Page 72 , CONFIG LCD »Page 97

Example

```

-----
'                               (c) 2000 MCS Electronics
-----
'   file: LCD.BAS
'   demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'           CURSOR, DISPLAY
-----
'note : tested in bus mode with 4-bit

Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Rem with the config lcdpin statement you can override the compiler
settings

Dim A As Byte
Config Lcd = 16 * 2                'configure lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses
over 2 lines

'$LCD = address will turn LCD into 8-bit databus mode
'           use this with uP with external RAM and/or ROM
'           because it aint need the port pins !

Cls                               'clear the LCD display
Lcd "Hello world."                'display this at the top line
Wait 1
Lowerline                         'select the lower line
Wait 1
Lcd "Shift this."                 'display this at the lower
line
Wait 1
For A = 1 To 10
  Shiftlcd Right                   'shift the text to the right
  Wait 1                           'wait a moment
Next

For A = 1 To 10
  Shiftlcd Left                    'shift the text to the left
  Wait 1                           'wait a moment

```

```

Next

Locate 2 , 1          'set cursor position
Lcd "***"           'display this
Wait 1               'wait a moment

Shiftcursor Right    'shift the cursor
Lcd "@"             'display this
Wait 1               'wait a moment

Home Upper           'select line 1 and return
home
Lcd "Replaced."      'replace the text
Wait 1               'wait a moment

Cursor Off Noblink   'hide cursor
Wait 1               'wait a moment
Cursor On Blink      'show cursor
Wait 1               'wait a moment
Display Off           'turn display off
Wait 1               'wait a moment
Display On           'turn display on
'-----NEW support for 4-line LCD-----
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third           'goto home on line three
Home Fourth
Home F               'first letter also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 242 , 234 , 228      '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240  '
replace ? with number (0-7)
Cls                  'select data RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1)  'print the special character

'----- Now use an internal routine -----
_templ = 1          'value
into ACC
rCall _write_lcd    'put
it on LCD
End

```

LEFT

Action

Return the specified number of leftmost characters in a string.

Syntax

```
var = Left(var1 , n )
```

Remarks

Var The string that is assigned.
Var1 The source string.
n The number of characters to get from the source string.

Seealso

RIGHT »Page 171 , MID »Page 158

Example

```

Dim s As XRAM String * 15, z As String * 15
s = "ABCDEFGG"
z = Left(s,5)
Print z                'ABCDE
End

```

LEN

Action

Returns the length of a string.

Syntax

```
var = LEN( string )
```

Remarks

var A numeric variable that is assigned with the length of string.
string The string to calculate the length of.
Strings can be maximum 254 bytes long.

Example

```

Dim S As String * 12
Dim A As Byte
S = "test"
A = Len(s)
Print A ' prints 4
Print Len(s)

```

LTRIM

Action

Returns a copy of string with leading blanks removed

Syntax

```
var = LTRIM( org )
```

Remarks

var String that receives the result.
org The string to remove the leading spaces from

Seealso

RTRIM »Page 171 , TRIM »Page 185

ASM

Example

```
Dim S As String * 6
S = "  AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

LOAD

Action

Load specified TIMER with a reload value.

Syntax

```
LOAD TIMER , value
```

Remarks

TIMER TIMER0
Value The variable or value to load.

The TIMER0 does not have a reload mode. But when you want the timer to generate an interrupt after 10 ticks for example, you can use the RELOAD statement.

It will do the calculation. (256-value)

So LOAD TIMER0, 10 will load the TIMER0 with a value of 246 so that it will overflow after 10 ticks.

LOCAL

Action

Dimensions a variable LOCAL to the function or sub program.

Syntax

```
LOCAL var As Type
```

Remarks

var The name of the variable
type The data type of the variable.
There can be only LOCAL variables of the type BYTE, INTEGER, WORD, LONG, SINGLE or STRING.

A LOCAL variable is a temporary variable that is stored on the frame. When the SUB or FUNCTION is terminated, the memory will be released back to the frame.

BIT variables are not possible because they are GLOBAL to the system.

The AT , ERAM, SRAM, XRAM directives can not be used with a local DIM statement. Also arrays are not possible with LOCAL's.

Seealso

DIM »Page 120

ASM

Example

```
'-----
'                                     (c) 2000 MCS Electronics
'                                     DECLARE.BAS
' Note that the usage of SUBS works different in BASCOM-8051
'-----
' First the SUB programs must be declared

'Try a SUB without parameters
Declare Sub Test2

'SUB with variable that cant be changed(A) and
'a variable that can be changed(B1), by the sub program
'When BYVAL is specified, the value is passed to the subprogram
'When BYREF is specified or nothing is specified, the address is
passed to
'the subprogram

Declare Sub Test(byval A As Byte , B1 As Byte)
```

```

'All variable types that can be passed
'Notice that BIT variables cant be passed.
'BIT variables are GLOBAL to the application
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long
, S As String)

Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String *
10      'dim used variables
Dim Ar(10) As Byte

Call Test2          'call
sub
Test2              'or use
without CALL
'Note that when calling a sub without the statement CALL, the
enclosing parentheses must be left out
Bb = 1
Call Test(1 , Bb)  'call
sub with parameters
Print Bb          'print
value that is changed

'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S

'now pass an array
'note that it must be passed by reference
Test 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
End

'End your code with the subprograms
'Note that the same variables and names must be used as the declared
ones

Sub Test(byval A As Byte , B1 As Byte)          'start
sub
    Print A ; " " ; B1          'print passed variables
    B1 = 3                      'change
value
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub

Sub Test2          'sub without parameters
    Print "No parameters"
End Sub

Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
    Local X As Byte

    X = 5          'assign
local

```

```

B = X
I = -1
W = 40000
L = 20000
S = "test"
End Sub

```

LOCATE

Action

Moves the LCD cursor to the specified position.

Syntax

LOCATE y , x

Remarks

X Constant or variable with the position. (1-64*)
y Constant or variable with the line (1 - 4*)

* Depending on the used display

Seealso

CONFIG LCD »Page 97 , LCD »Page 147, HOME »Page 138 ,CLS »Page 91

Example

```

LCD "Hello"
Locate 1,10
LCD "*"

```

LOOKUP

Action

Returns a value from a table.

Syntax

var =LOOKUP(value, label)

Remarks

Var The returned value
Value A value with the index of the table
Label The label where the data starts

Seealso

LOOKUPSTR »Page 155

Example

```
DIM b1 As Byte , I as Integer
b1 = Lookup(1, dta)
Print b1 ' Prints 2 (zero based)
```

```
I = Lookup(0,DTA2)
End
```

```
DTA:
DATA 1,2,3,4,5
```

```
DTA2: 'integer data
DATA 1000% , 2000%
```

LOOKUPSTR

Action

Returns a string from a table.

Syntax

`var =LOOKUPSTR(value, label)`

Remarks

var The string returned
value A value with the index of the table. The index is zero-based. That is, 0 will return the first element of the table.
Label The label where the data starts

Seealso

LOOKUP »Page 154

Example

```
Dim s as string, idx as Byte
idx = 0 : s = LookupStr(idx,Sdata)
Print s 'will print 'This'
End
```

```
Sdata:
Data "This" , "is" ,"a test"
```

LOW

Action

Retrieves the least significant byte of a variable.

Syntax

`var = LOW(s)`

Remarks

Var The variable that is assigned with the LSB of var S.
S The source variable to get the LSB from.

Seealso

HIGH »Page 138

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = Low(I) ' is 1
```

LOWERLINE

Action

Reset the LCD cursor to the lowerline.

Syntax

`LOWERLINE`

Remarks

-

Seealso

UPPERLINE, »Page 186 THIRDLINE »Page 184, FOURTHLINE »Page 129, HOME »Page 138

Example

```
LCD "Test"
LOWERLINE
LCD "Hello"
End
```

MAKEBCD

Action

Convert a variable into its BCD value.

Syntax

```
var1 = MAKEBCD(var2)
```

Remarks

var1 Variable that will be assigned with the converted value.
Var2 Variable that holds the decimal value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from decimal to BCD.

For printing the bcd value of a variable, you can use the BCD() function which converts a BCD number into a BCD string.

Seealso

MAKEDEC »Page 158, BCD »Page 86

Example

```
Dim a As Byte
a = 65
LCD a
Lowerline
LCD BCD(a)
a = MakeBCD(a)
LCD " " ; a
End
```

MAKEINT

Action

Compact two bytes into a word or integer.

Syntax

```
varn = MAKEINT(LSB, MSB)
```

Remarks

Varn Variable that will be assigned with the converted value.
LSB Variable or constant with the LS Byte.
MSB Variable or constant with the MS Byte.

The equivalent code is:

```
varn = (256 * MSB) + LSB
```

Seealso

LOW »Page 156, HIGH »Page 138

Example

```
Dim a As Integer, I As Integer
a = 2
I = MakeINT(a, 1) 'I = (1 * 256) + 2 = 258
End
```

MAKEDEC

Action

Convert a BCD byte or Integer/Word variable to its DECIMAL value.

Syntax

```
var1 = MAKEDEC(var2)
```

Remarks

var1 Variable that will be assigned with the converted value.
var2 Variable that holds the BCD value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from BCD to decimal.

Seealso

MAKEBCD »Page 157

Example

```
Dim a As Byte
a = 65
LCD a
Lowerline
LCD BCD(a)
a = MakeDEC(a)
LCD " " ; a
End
```

MID

Action

The MID function returns part of a string (a sub string).
The MID statement replaces part of a string variable with another string.

Syntax

```
var = MID(var1 ,st [, l] )
MID(var ,st [, l] ) = var1
```

Remarks

var	The string that is assigned.
Var1	The source string.
st	The starting position.
l	The number of characters to get/set.

Seealso

LEFT »Page 150, RIGHT »Page 171

Example

```
Dim s As XRAM String * 15, z As XRAM String * 15
s = "ABCDEFGH"
z = Mid(s,2,3)
Print z           'BCD
z="12345"
Mid(s,2,2) = z
Print s           'A12DEFGH
End
```

ON INTERRUPT

Action

Execute subroutine when specified interrupt occurs.

Syntax

```
ON interrupt label [NOSAVE]
```

Remarks

Interrupt INT0, INT1, INT2, INT3, INT4, INT5, TIMER0, TIMER1, TIMER2, ADC, EEPROM, CAPTURE1, COMPARE1A, COMPARE1B, COMPARE1. Or you can use the AVR name convention :

OC2, OVF2, ICP1, OC1A, OC1B, OVF1, OVF0, SPI, URXC, UDRE, UTXC, ADCC, ERDY and ACI.

Label The label to jump to if the interrupt occurs.

NOSAVE When you specify NOSAVE, no registers are saved and restored in the interrupt routine. So when you use this option be sure to save and restore used registers.

You must return from the interrupt routine with the RETURN statement. You may have only one RETURN statement in your interrupt routine because the compiler restores the registers and generates a RETI instruction when it encounters a RETURN statement in the ISR.

Example

```
ENABLE INTERRUPTS
ENABLE INT0           'enable the interrupt
ON INT0 Label2 nosave 'jump to label2 on INT0
DO                   'endless loop
LOOP
END
```

ON VALUE

Action

Branch to one of several specified labels, depending on the value of a variable.

Syntax

```
ON var [GOTO] [GOSUB] label1 [, label2 ]
```

Remarks

var	The numeric variable to test. This can also be a SFR such as PORTB.
label1, label2	The labels to jump to depending on the value of <i>var</i> .

Note that the value is zero based. So when var = 0, the first specified label is jumped/branched.

Example

```
x = 2           'assign a variable interrupt
ON x GOSUB lb11, lb12, lb13 'jump to label lb13
x=0
ON x GOTO lb11, lb12, lb13
END

lb13:
PRINT " lb13"
RETURN

Lb11:

Lb12:
```

OPEN

Action

Opens a device.

Syntax

OPEN *device* for MODE As *#channel*
CLOSE *#channel*

Remarks

device The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device.

With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data.

So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use.

COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stopbits.

The format for COM1 is : COM1:speed, where the speed is optional and will override the compiler settings for the speed.

The format for the software UART is:

COMpin:speed,8,N,stopbits[,INVERTED]

Where pin is the name of the PORT-pin.

Speed must be specified and stop bits can be 1 or 2.

An optional parameter ,INVERTED can be specified to use inverted RS-232.

Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.

MODE You can use BINARY or RANDOM for COM1, but for the software UART pins, you must specify INPUT or OUTPUT.

channel The number of the channel to open. Must be a positive constant >0.

The statements that support the device are PRINT »Page 165 , INPUT »Page 146 and INPUTHEX »Page 145

Every opened device must be closed using the CLOSE *#channel* statement. Of course, you must use the same channel number.

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

Seealso

CLOSE »Page 92

Example

```

-----
'
'                                OPEN.BAS
'    demonstrates software UART
'
-----

Dim B As Byte

'open channel for output and use inverted logic

Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 , "serial output"
Close #1

'Now open a pin for input and use inverted logic
Open "comd.2:9600,8,n,1,inverted" For Input As #2
Input #2 , B
Close #2

'use normal hardware UART for printing
Print B

End

```

OUT

Action

Sends a byte to a hardware port or external memory address.

Syntax

OUT address, value

Remarks

address

The address where to send the byte to.

value

The variable or value to send.

The OUT statement can write a value to any AVR memory location.

Seealso

INP »Page 144

Example

```

Dim a as byte
OUT &H8000,1          'send 1 to the databus(d0-d7) at hex address
8000
END

```

PEEK

Action

Returns the content of a register.

Syntax

var = PEEK(address)

Remarks

Var Numeric variable that is assigned with the content of the memory location **address**

Address Numeric variable or constant with the address location.(0-31)

Peek() will read the content of a register.

Inp() can read any memory location

See also

POKE »Page 163 , CPEEK »Page 110 , INP »Page 144 , OUT »Page 162

Example

```
DIM A As Byte
a = Peek( 0 )      'return the first byte of the internal memory (r0)
End
```

POKE

Action

Write a byte to an internal register.

Syntax

POKE address , value

Remarks

address Numeric variable with the address of the memory location to set. (0-31)

value Value to assign. (0-255)

See also

PEEK »Page 163 , CPEEK »Page 110 , INP »Page 144 , OUT »Page 162

Example

```
POKE 1, 1          'write 1 to R1
End
```

POPALL

Action

Restores all registers that might be used by BASCOM.

Syntax

POPALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background. That is why Pushall saves all registers and POPALL restores all registers.

See also

PUSHALL »Page 166

POWERDOWN

Action

Put processor into power down mode.

Syntax

POWERDOWN

Remarks

In the power down mode, the external oscillator is stopped. Only a reset can wake up the MCU. You can force the watchdog timer to generate this reset.

See also

IDLE »Page 141 , POWERSAVE »Page 164

Example

```
POWERDOWN
```

POWERSAVE

Action

Put processor into power save mode.

Syntax

POWERSAVE

Remarks

The POWERSAVE mode is only available on the 8535.

Seealso

IDLE »Page 141, POWERDOWN »Page 164

Example

POWERDOWN

PRINT

Action

Send output to the RS-232 port.

Syntax

PRINT var ; " constant"

Remarks

var The variable or constant to print.

You can use a semicolon (;) to print more than one variable at one line. When you end a line with a semicolon, no linefeed will be added.

The PRINT routine can be used when you have a RS-232 interface on your uP.

The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator as an output device. You can also use the build in terminal emulator.

Seealso

INPUT »Page 146 ,OPEN »Page 161 , CLOSE »Page 92

Example

```

'-----
'                               (c) 2000 MCS Electronics
'-----
'   file: PRINT.BAS
'   demo: PRINT
'-----
Dim A As Byte , B1 As Byte , C As Integer
A = 1
Print "print variable a " ; A
Print                               'new line
Print "Text to print."             'constant to print

B1 = 10

```

```

Print Hex(B1)                'print in hexa notation
C = &HA000                  'assign value to c%
Print Hex(C)                'print in hex notation
Print C                      'print in decimal notation

C = -32000
Print C
Print Hex(C)
Rem Note That Integers Range From -32767 To 32768
End

```

PRINTBIN

Action

Print binary content of a variable to the serial port.

Syntax

PRINTBIN var [,varn]
PRINTBIN #channel , var [, varn]

Remarks

var The variable which value is send to the serial port.
varn Optional variables to send.

The channel is optional and for use with OPEN »Page 161 and CLOSE »Page 92 statements.

PRINTBIN is equivalent to PRINT CHR(var); but whole arrays can be printed thisway.

When you use a Long for example, 4 bytes are printed.

Seealso

INPUTBIN »Page 144

Example

```

Dim a(10) as Byte, c as Byte
For c = 1 To 10
    a(c) = a          'fill array
Next
PRINTBIN a(1)       'print content

```

PUSHALL

Action

Saves all registers that might be used by BASCOM.

Syntax

PUSHALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background. That is why Pushall saves all registers. Use POPALL to restore the registers.

Seealso

POPALL »Page 164

READ

Action

Reads those values and assigns them to variables.

Syntax

READ var

Remarks

Var Variable that is assigned data value.

It is best to place the DATA »Page 112 lines at the end of your program.

Difference with QB

It is important that the variable is of the same type as the stored data.

Seealso

DATA »Page 112 , RESTORE »Page 170

Example

```
Dim A As Byte, I As Byte, C As Integer, S As String * 10
RESTORE dta
FOR a = 1 TO 3
  READ i : PRINT i
NEXT
RESTORE DTA2
READ C : PRINT C
READ C : PRINT C
Restore dta3 : Read s : Print s
END
```

```
dta:
Data 5,10,15
dta2:
Data 1000%, -2000%
dta3:
Data "hello"
```

READEEPROM

Action

Reads the content from the DATA EEPROM and stores it into a variable.

Syntax

READEEPROM var , address

Remarks

var The name of the variable that must be stored
address The address in the EEPROM where the data must be read from.

This statement is provided for compatibility with BASCOM-8051.

You can also use :

Dim V as Eram Byte 'store in EEPROM

Dim B As Byte 'normal variable

B = 10

V = B 'store variable in EEPROM

B = V 'read from EEPROM

When you use the assignment version, the datatypes must be equal!

According to a datasheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset.

Seealso

WRITEEEPROM »Page 190

ASM

Example

```
Dim B As Byte
WriteEEPROM B ,0 'store at first position
ReadEEPROM B, 0 'read byte back
```

REM

Action

Instruct the compiler that comment will follow.

Syntax

REM or '

Remarks

You can and should comment your program for clarity and your later sanity. You can use REM or ' followed by your comment.

All statements after REM or ' are treated as comments so you cannot use statements on the same line after a REM statement.

Block comments can be used:

```
'( start block comment
print "This will not be compiled
') end block comment
```

Note that the starting ' sign will ensure compatibility with QB

Example

```
REM TEST.BAS version 1.00
PRINT a ' " this is comment : PRINT " hello"
      ^--- this will not be executed!
```

RESET

Action

Reset a bit to zero.

Syntax

RESET bit

RESET var.x

Remarks

bit Can be a SFR such as PORTB.x, or any bit variable where x=0-7.
var Can be a byte, integer word or long variable.
x Constant of variable to reset.(0-7) for bytes and (0-15) for Integer/Word. For longs(0-31)

Seealso

SET »Page 173

Example

```
Dim b1 as bit, b2 as byte, I as Integer
RESET PORTB.3 'reset bit 3 of port B
RESET b1 'bitvariable
```

```
RESET b2.0 'reset bit 0 of bytevariable b2
RESET I.15 'reset MS bit from I
```

RESTORE

Action

Allows READ to reread values in specified DATA statements by setting data pointer to beginning of data statement.

Syntax

RESTORE label

Remarks

label The label of a DATA statement.

Seealso

DATA »Page 112 , READ »Page 167

Example

```
DIM a AS BYTE, I AS BYTE
RESTORE dta
FOR a = 1 TO 3
  READ a : PRINT a
NEXT
RESTORE DTA2
READ I : PRINT I
READ I : PRINT I
END

DTA1:
Data 5, 10, 100

DTA2:
Data -1%, 1000%
Integers must end with the %-sign. (Integer : <0 or >255)
```

RETURN

Action

Return from a subroutine.

Syntax

RETURN

Remarks

Subroutines must be ended with a related RETURN statement.
Interrupt subroutines must also be terminated with the Return statement.

Seealso

GOSUB »Page 135

Example

```
GOSUB Pr          'jump to subroutine
PRINT result     'print result
END              'program ends

Pr:              'start subroutine with
label
    result = 5 * y          'do something stupid
    result = result + 100  'add something to it
RETURN           'return
```

RIGHT

Action

Return a specified number of rightmost characters in a string.

Syntax

var = **RIGHT**(var1 ,st)

Remarks

var The string that is assigned.
Var1 The source string.
st The starting position.

Seealso

LEFT »Page 150 , MID »Page 158

Example

```
Dim s As String * 15, z As String * 15
s = "ABCDEFGH"
z = Right(s,2)
Print z          'FG
End
```

RTRIM

Action

Returns a copy of a string with trailing blanks removed

Syntax

var = **RTRIM**(org)

Remarks

var String that is assigned with the result.
org The string to remove the trailing spaces from

Seealso

TRIM »Page 185 , LTRIM »Page 151

ASM

Example

```
Dim S As String * 6
S = "  AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

ROTATE

Action

Rotate all bits one place to the left or right.

Syntax

ROTATE var , **LEFT/RIGHT** [, shifts]

Remarks

Var Byte, Integer/Word or Long variable.
Shifts The number of shifts to perform.

The ROTATE statement rotates all the bits in the variable to the left or right. All bits are preserved so no bits will be shifted out of the variable. This means that after rotating a byte variable with a value of 1, eight times the variable will be unchanged. When you want to shift out the MS bit or LS bit, use the SHIFT statement.

Seealso

SHIFT »Page 174 , SHIFIN »Page 175 , SHIFOUT »Page 176

Example

```
Dim a as Byte
```

```
a = 128
ROTATE a, LEFT , 2
Print a      '2
```

SELECT-CASE-END SELECT

Action

Executes one of several statement blocks depending on the value of an expression.

Syntax

```
SELECT CASE var
  CASE test1 : statements
  [CASE test2 : statements ]
  CASE ELSE :statements
END SELECT
```

Remarks

var Variable. to test
Test1 Value to test for.
Test2 Value to test for.

Seealso

-

Example

```
Dim b2 as byte
SELECT CASE b2
  CASE 2 : PRINT "2"
  CASE 4 : PRINT "4"
  CASE IS >5 : PRINT ">5"
  CASE ELSE
END SELECT
END
```

'set bit 1 of port 1

'a test requires the IS keyword

SET

Action

Set a bit to one.

Syntax

```
SET bit
SET var.x
```

Remarks

Bit Bitvariable.
Var A byte, integer, word or long variable.
X Bit of variable (0-7) to set. (0-15 for Integer/Word) and (0-31) for Long

Seealso

RESET »Page 169

Example

```
Dim b1 as Bit, b2 as byte, c as Word, L as Long
SET PORTB.1 'set bit 1 of port B
SET b1        'bit variable
SET b2.1      'set bit 1 of var b2
SET C.15      'set highest bit of Word
SET L.31      'set MS bit of LONG
```

SHIFT

Action

Shift all bits one place to the left or right.

Syntax

```
SHIFT var , LEFT/RIGHT [ , shifts]
```

Remarks

Var Byte, Integer/Word or Long variable.
Shifts The number of shifts to perform.

The SHIFT statement rotates all the bits in the variable to the left or right.

When shifting LEFT the most significant bit, will be shifted out of the variable. The LS bit becomes zero. Shifting a variable to the left, multiplies the variable with a value of two.

When shifting to the RIGHT, the least significant bit will be shifted out of the variable. The MS bit becomes zero. Shifting a variable to the right, divides the variable by two.

Seealso

ROTATE »Page 172 , SHIF TIN »Page 175 , SHIF TOUT »Page 176

Example

```
Dim a as Byte
a = 128
SHIFT a, LEFT , 2
Print a '0
```

SHIFTCURSOR

Action

Shift the cursor of the LCD display left or right by one position.

Syntax

SHIFTCURSOR LEFT / RIGHT

Seealso

SHIFTLCD »Page 177

Example

```
LCD "Hello"
SHIFTCURSOR LEFT
End
```

SHIF TIN

Action

Shifts a bitstream into a variable.

Syntax

SHIF TIN pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as an input.
Pclock	The port pin which generates the clock.
Var	The variable that is assigned.
Option	Option can be : 0 – MSB shifted in/out first when clock goes low 1 – MSB shifted in/out first when clock goes high 2 – LSB shifted in/out first when clock goes low 3 – LSB shifted in/out first when clock goes high Adding 4 to the parameter ses an external clock signal for the shifting.
Bits	Optional number of bits to shift in.
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.
When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur.

Seealso

SHIF TOUT »Page 176 , SHIF T »Page 174

Example

```
Dim a as byte
SHIF TIN PORTB.0, PORTB.1, A, 4, 4,10 'shiftin 4 bits and use external
clock
SHIF T A, RIGHT,4 'adjust

SHIF TIN PORTB.0, PORTB.1 , A 'read 8 bits
```

SHIF TOUT

Action

Shifts a bitstream out of a variable into a port pin .

Syntax

SHIF TOUT pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as an input.
Pclock	The port pin which generates the clock.
Var	The variable that is assigned.
Option	Option can be : 0 – MSB shifted in/out first when clock goes low 1 – MSB shifted in/out first when clock goes high 2 – LSB shifted in/out first when clock goes low 3 – LSB shifted in/out first when clock goes high
Bits	Optional number of bits to shift in.
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.
When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur.

Seealso

SHIF TIN »Page 175 , SHIF T »Page 174

Example

```
Dim a as byte
```

```
SHIFTOUT PORTB.0, PORTB.1, A, 4, 4,10 'shiftout 4 bits
SHIFTIN PORTB.0, PORTB.1 , A 'shiftout 8 bits
```

SHIFTLCD

Action

Shift the LCD display left or right by one position.

Syntax

SHIFTLCD LEFT / RIGHT

Remarks

-

Seealso

SHIFTCURSOR »Page 175

Example

```
LCD "Very long text"
SHIFTLCD LEFT
Wait 1
SHIFTLCD RIGHT
End
```

SOUND

Action

Sends pulses to a port pin.

Syntax

SOUND pin, duration, pulses

Remarks

Pin	Any I/O pin such as PORTB.0 etc.
Duration	The number of pulses to send. Byte, integer/word or constant.
Pulses	The time the pin is pulled low and high. This is the value for a loop counter.

When you connect a speaker or a buzzer to a port pin (see hardware) , you can use the SOUND statement to generate some tones.

The port pin is switched high and low for *pulses* times.
This loop is executed *duration* times.

The SOUND statement is not intended to generate accurate frequencies. Use a TIMER to do that.

Seealso

-

Example

```
SOUND PORTB.1 , 10000, 10 'BEEP
End
```

SPACE

Action

Returns a string that consists of spaces.

Syntax

var = **SPACE**(x)

Remarks

X The number of spaces.

Var The string that is assigned.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

Seealso

STRING »Page 183

Example

```
Dim s as String * 15, z as String * 15
s = Space(5)
Print " { " ; s ; " } "

Dim A as Byte
A = 3
S = Space(a)
```

SPIIN

Action

Reads a value from the SPI-bus.

Syntax

SPIIN var, bytes

Remarks

var The variable which receives the value read from the SPI-bus.
bytes The number of bytes to read.

Seealso

SPIOUT, »Page 180 SPIINIT, »Page 179 CONFIG SPI »Page101 , SPIMOVE »Page 199

Example

```
Dim a(10) as byte
CONFIG SPI = SOFT, DIN = PORTB.0, DOUT = PORTB.1, CS=PORTB.2, CLK =
PORTB.3
INITSPI
SPIIN a(1) , 4          'read 4 bytes
```

SPIINIT

Action

Sets the SPI pins in the right mode.

Syntax

SPIINIT

Remarks

After the configuration of the SPI pins, you must initialize the SPI pins to set them for the right data direction. When the pins are not used by other hardware/software, you only need to use SPIINIT once.

When other routines change the state of the SPI pins, use SPIINIT again before using SPIIN and SPIOUT.

Seealso

SPIIN »Page 178 , SPIOUT »Page 180

ASM

Calls `_init_spi`

Example

```
Dim a(10) as byte
CONFIG SPI = SOFT, DIN = PORTB.0, DOUT = PORTB.1, CS=PORTB.2, CLK =
PORTB.3
INITSPI
SPIIN a(1) , 4          'read 4 bytes
```

SPIOUT

Action

Sends a value of a variable to the SPI-bus.

Syntax

SPIOUT **var** , **bytes**

Remarks

var The variable whose content must be send to the SPI-bus.
bytes The number of bytes to send.

Seealso

SPIIN »Page 178 , SPIINIT »Page 179 , CONFIG SPI »Page 101 , SPIMOVE »Page 199

Example

```
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
INITSPI
Dim a(10) as Byte , X As Byte
SPIOUT a(1) , 5      'send 5 bytes
SPIOUT X , 1         'send 1 byte
```

START

Action

Start the specified device.

Syntax

START **device**

Remarks

Device TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power) or ADC(A/D converter power)

You must start a timer/counter in order for an interrupt to occur (when the external gate is disabled).

TIMER0 and COUNTER0 are the same device.

The AC and ADC parameters will switch power to the device and thus enabling it to work.

Seealso

STOP »Page 181

Example

```

-----
'          ADC.BAS
' demonstration of GETADC() function for 8535 micro
-----
'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,15,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End

```

Action

Stop the specified device. Or stop the program

Syntax

STOP device

STOP

Remarks

Device TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG,
AC (Analog comparator power) or ADC(A/D converter power)

The single STOP statement will end your program.

The STOP statement with one of the above parameters, will stop the specified device.

TIMER0 and COUNTER0 are the same device.

The AC and ADC parameters will switch power off the device to disable it and thus save power.

Seealso

START »Page 180

Example

```

-----
'          ADC.BAS
' demonstration of GETADC() function for 8535 micro
-----
'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,15,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

'With STOP ADC, you can remove the power from the chip
'Stop Adc

Dim W As Word , Channel As Byte

Channel = 0
'now read A/D value from channel 0
Do
  W = Getadc(channel)
  Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
Loop
End

```

STR

Action

Returns a string representation of a number.

Syntax

var = Str(x)

Remarks

var A string variable.
X A numeric variable.

The string must be big enough to store the result.

See also

VAL »Page 186 , HEX »Page 137 , HEXVAL »Page 137

Difference with QB

In QB STR() returns a string with a leading space. BASCOM does not.

Example

```
Dim a as Byte, S as XRAM String * 10
a = 123
s = Str(a)
Print s
End
```

STRING

Action

Returns a string consisting of m repetitions of the character with ASCII Code n.

Syntax

var = **STRING**(m ,n)

Remarks

var The string that is assigned.
n The ASCII-code that is assigned to the string.
m The number of characters to assign.
 Since a string is terminated by a 0 byte, you can't use 0 for n.
 Using 0 for m will result in a string of 255 bytes, because there is no check on a length assign of 0.

See also

SPACE »Page 178

Example

```
Dim s as String * 15
s = String(5,65)
Print s                               'AAAAA
End
```

SUB

Action

Defines a Sub procedure.

Syntax

SUB **Name**[(var1)]

Remarks

Name Name of the sub procedure, can be any non-reserved word.
var1 The name of the parameter.

You must end each subroutine with the END SUB statement.
 You can copy the DECLARE SUB line and remove the DECLARE statement.
 This ensures that you have the right parameters.

See the DECLARE SUB »Page 118 topic for more details.

SWAP

Action

Exchange two variables of the same type.

Syntax

SWAP var1 , var2

Remarks

var1 A variable of type bit, byte, integer, word, long or string.
var2 A variable of the same type as var1.

After the swap, var1 will hold the value of var2 and var2 will hold the value of var1.

Example

```
Dim a as integer,b1 as integer
a = 1 : b1 = 2                       'assign two integers
SWAP a, b1                           'swap them
PRINT a ; b1                         'prints 21
```

THIRDLINE

Action

Reset LCD cursor to the third line.

Syntax

THIRDLINE

Remarks

-

Seealso

UPPERLINE »Page 186 , LOWERLINE »Page 156 , FOURTHLINE »Page 129

Example

```
Dim a as byte
a = 255
LCD a
Thirdline
LCD a
Upperline
End
```

TRIM

Action

Returns a copy of a string with leading and trailing blanks removed

Syntax

```
var = TRIM( org )
```

Remarks

var	String that receives the result.
org	The string to remove the spaces from

Seealso

RTRIM »Page 171 , LTRIM »Page 151

ASM

Example

```
Dim S As String * 6
S = " AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

UPPERLINE

Action

Reset LCD cursor to the upperline.

Syntax

UPPERLINE

Remarks

-

Seealso

LOWERLINE »Page 156 , THIRDLINE »Page 184 , FOURTHLINE »Page 129

Example

```
Dim a as byte
a = 255
LCD a
Lowerline
LCD a
Upperline
End
```

VAL

Action

Converts a string representation of a number into a number.

Syntax

```
var = Val( s )
```

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type.

Seealso

STR »Page 182

Example

```
Dim a as byte, s As String * 10
s = "123"
a = Val(s) 'convert string
Print a
```

End

VARPTR

Action

Retrieves the memory-address of a variable.

Syntax

```
var = VARPTR( var2 )
```

Remarks

Var The variable that receives the address of var2.
Var2 A variable to retrieve the address from.

Seealso

-

Example

```
Dim B As Xram Byte At &H300 , I As Integer , W As Word
W = VarpTR(b)
Print Hex(w)      'Print &H0300
End
```

WAIT

Action

Suspends program execution for a given time.

Syntax

```
WAIT seconds
```

Remarks

seconds The number of seconds to wait.

No accurate timing is possible with this command.
 When you use interrupts, the delay may be extended.

Seealso

DELAY »Page 120 , WAITMS »Page 188

Example

```
WAIT 3          'wait for three seconds
```

```
Print ""
```

WAITKEY

Action

Wait until a character is received in the serial buffer.

Syntax

```
var = WAITKEY
```

Remarks

var Variable that receives the ASCII value of the serial buffer.

Seealso

INKEY »Page 143

Example

```
Dim A As Byte
A = Waitkey()      'wait for character
Print A
```

WAITMS

Action

Suspends program execution for a given time in mS.

Syntax

```
WAITMS mS
```

Remarks

ms The number of milliseconds to wait. (1-255)

No accurate timing is possible with this command.
 In addition, the use of interrupts can slow this routine.
 This statement is provided for the I2C statements.
 When you write to an EEPROM you must wait for 10 mS after the write instruction.

Seealso

DELAY »Page 120 , WAIT »Page 187 , WAITUS »Page 189

Example

```
WAITMS 10       'wait for 10 mS
Print ""
```

WAITUS

Action

Suspends program execution for a given time in uS.

Syntax

```
WAITUS uS
```

Remarks

uS The number of micriseconds to wait. (1-255)
This must be a constant. No variable!

No accurate timing is possible with this command.
In addition, the use of interrupts can slow this routine.

Seealso

DELAY »Page 120 , WAIT »Page187 , WAITUS »Page189

Example

```
WAITUS 10           'wait for 10 uS
Print "**"
```

WHILE-WEND

Action

Executes a series of statements in a loop, as long as a given condition is true.

Syntax

```
WHILE condition
    statements
WEND
```

Remarks

If the condition is true then any intervening statements are executed until the WEND statement is encountered.

BASC0M then returns to the WHILE statement and checks the **condition**.

If it is still true, the process is repeated.

If it is not true, execution resumes with the statement following the WEND statement.

So in contrast with the DO-LOOP structure, a WHILE-WEND condition is tested first so that if the condition fails, the statements in the WHILE-WEND structure are never executed.

Seealso

DO-LOOP »Page 125

Example

```
WHILE a <= 10           'if a is smaller or equal to 10
    PRINT a           'print variable a
    INCR a
WEND
```

WRITEEEPROM

Action

Write a variables content to the DATA EEPROM.

Syntax

```
WRITEEEPROM var , address
```

Remarks

var The name of the variable that must be stored
address The address in the EEPROM where the variable
 must be stored.

This statement is provided for compatibility with BASCOM-8051.

You can also use :

```
Dim V as Eram Byte 'store in EEPROM
Dim B As Byte       'normal variable
B = 10
```

```
V = B               'store variable in EEPROM
```

When you use the assignment version, the data types must be the same!

According to a datasheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset.

Seealso

READEEPROM »Page 168

ASM

Example

```
Dim B As Byte
WriteEEPROM B ,0 'store at first position
ReadEEPROM B, 0 'read byte back
```

LOADADR

Action

Loads the address of a variable into a register pair.

Syntax

```
LOADADR var , reg
```

Remarks

var A variable which address must be loaded into the register pair X, Y or Z.

reg The register X, Y or Z.

The LOADADR statement serves as a assembly helper routine.

Example

```
Dim S As String * 12
Dim A As Byte
$ASM
  loadadr S , X 'load address into R26 and R27
  ld _templ, X 'load value of location R26/R27 into R24(_templ)
$END ASM
```

Changes compared to BASCOM-8051

The design goal was to make BASCOM-AVR compatible with BASCOM-8051.

The standard edition is intended as a replacement for BASCOM-LT.

The professional edition is intended as a replacement for BASCOM-8051.

For the AVR compilers I had to remove some statements.

New statements are also added. And some statements were changed.

They need specific attention, but the changes to the syntax will be made available to BASCOM-8051 too in the future.

Statements that were removed

STATEMENT	DESCRIPTION
\$LARGE	Not needed anymore.
\$ROMSTART	Code always starts at address 0 for the AVR.
\$LCDHEX	Use LCD Hex(var) instead.
\$NOINIT	Not needed anymore
\$NOSP	Not needed anymore
\$NOBREAK	Can't be used anymore because there is no object code that can be used for it.

\$SIM	Removed because there is no simulator yet.
\$OBJ	Removed.
BREAK	Can't be used anymore because there is no object code that can be used for it.
PRIORITY	AVR does not allow setting priority of interrupts
PRINTHEX	You can use Print Hex(var) now
LCDHEX	You can use Lcd Hex(var) now

Statements that were added

STATEMENT	DESCRIPTION
FUNCTION	Now you can define your own user FUNCTIONS.
LOCAL	You can have LOCAL variables in SUB routines or FUNCTIONS.
^	New math statement. Var = 2 ^ 3 will return 2*2*2
SHIFT	Because ROTATE was changed, I added the SHIFT statement. SHIFT works just like ROTATE, but when shifted left, the LS BIT is cleared and the carry doesn't go to the LS BIT.
LTRIM	LTRIM, trims the leftmost spaces of a string.
RTRIM	RTRIM, trims the rightmost spaces of a string.
TRIM	TRIM, trims both the leftmost and rightmost spaces of a string.

Statements that behave differently

STATEMENT	DESCRIPTION
ROTATE	Rotate now behaves like the ASM rotate, this means that the carry will go to the most significant bit of a variable or the least significant bit of a variable.
CONST	String were added to the CONST statement. I also changed it to be compatible with QB.
DECLARE	BYVAL has been added since real subprograms are now supported.
DIM	You can now specify the location in memory of the variable.
GETRC	Dim v as byte AT 100, will use memory location 100. Is named GETRC0 now to indicate that it works with TIMER0.

ISP programmer

BASC0M supports the STK200 ISP programmer from Kanda.

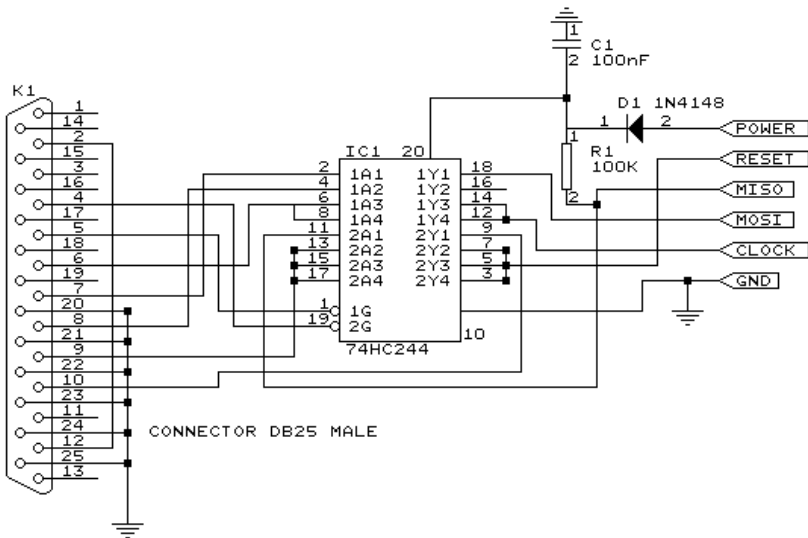
This is a very reliable parallel printer port programmer.

The STK200 ISP programmer is included in the STK200 starter kit.

All programs were tested with the STK200.

For those who don't have this kit and the programmer the following schematic shows how to make your own programmer:

The dongle has a chip with no identification but since the schematic is all over the web, I have included it. Kanda also sells a very cheap separate programmer dongle. So I suggest you buy this one!



Supported Programmers

BASCOM supports the following programmers

AVR ICP910 based on the AVR910.ASM application note

STK200 ISP programmer »Page 192 from Atmel/Kanda

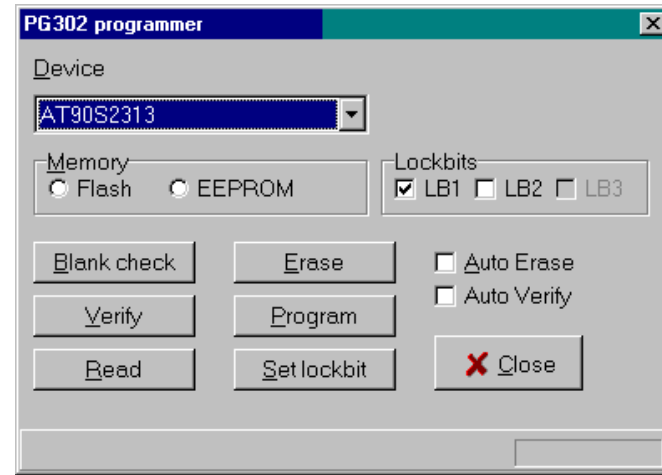
The PG302 programmer »Page 194 from Iguana Labs


The simple cable programmer »Page 198 from Sample Electronics.

Eddie McMullen's SPI programmer.

PG302 programmer

The PG302 is a serial programmer. It works and looks exactly as the original PG302 software.



Select the programmer from The Option Programmer menu or right click on the  button to show the Option Programmer »Page 36 menu.

Assembler mnemonics

BASCOM supports the mnemonics as defined by Atmel.

The Assembler accepts mnemonic instructions from the instruction set.

A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Mnemonics	Operands	Description	Operation	Flags	Clock
ARITHMETI					
CAND					
LOGIC					
INSTRUCTI					
ONS					
ADD	Rd,Rr	AddwithoutCarry	Rd = Rd + Rr	Z,C,N,V,H	1
ADC	Rd,Rr	AddwithCarry	Rd = Rd + Rr + C	Z,C,N,V,H	1
SUB	Rd,Rr	SubtractwithoutCarry	Rd = Rd - Rr	Z,C,N,V,H	1
SUBI	Rd,K	SubtractImmediate	Rd = Rd - K	Z,C,N,V,H	1

SBC	Rd,Rr	Subtract with Carry	$Rd = Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd,K	Subtract Immediate with Carry	$Rd = Rd - K - C$	Z,C,N,V,H	1
AND	Rd,Rr	Logical AND	$Rd = Rd \cdot Rr$	Z,N,V	1
ANDI	Rd,K	Logical AND with Immediate	$Rd = Rd \cdot K$	Z,N,V	1
OR	Rd,Rr	Logical OR	$Rd = Rd \vee Rr$	Z,N,V	1
ORl	Rd,K	Logical OR with Immediate	$Rd = Rd \vee K$	Z,N,V	1
EOR	Rd,Rr	Exclusive OR	$Rd = Rd \hat{\wedge} Rr$	Z,N,V	1
COM	Rd	Ones Complement	$Rd = \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Twos Complement	$Rd = \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd = Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd = Rd \cdot (\$FFh - K)$	Z,N,V	1
INC	Rd	Increment	$Rd = Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd = Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd = Rd \cdot Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd = Rd \hat{\wedge} Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd = \$FF$	None	1
ADIW	RdL,K	Add Immediate to Word	$RdH:RdL = RdH:RdL + K$	None	1
SBIW	RdL,K	Subtract Immediate from Word	$RdH:RdL = RdH:RdL - K$	None	1
MUL	Rd,Rr	Multiply Unsigned	$R1, R0 = Rd * Rr$	C	2
BRANCH INSTRUCTONS					
RJMP	k	Relative Jump	$PC = PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC = Z$	None	2
JMP	k	Jump	$PC = k$	None	3
RCALL	k	Relative Call Subroutine	$PC = PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC = Z$	None	3
CALL	k	Call Subroutine	$PC = k$	None	4
RET		Subroutine Return	$PC = STACK$	None	4
RETI		Interrupt Return	$PC = STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) $PC = PC + 2$ or 3	None	1/2
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,H	1
SBRC	Rr,b	Skip if Bit in Register Cleared	If (Rr(b)=0) $PC = PC + 2$ or 3	None	1/2
SBRS	Rr,b	Skip if Bit in Register Set	If (Rr(b)=1) $PC = PC + 2$ or 3	None	1/2
SBIC	P,b	Skip if Bit in I/O Register Cleared	If (I/O(P,b)=0) $PC = PC + 2$ or 3	None	2/3
SBIS	P,b	Skip if Bit in I/O Register Set	If (I/O(P,b)=1) $PC = PC + 2$ or 3	None	2/3
BRBS	s,k	Branch if Status Flag Set	if (SREG(s)=1) then $PC = PC + k + 1$	None	1/2
BRBC	s,k	Branch if Status Flag Cleared	if (SREG(s)=0) then $PC = PC + k + 1$	None	1/2
BREQ	k	Branch if Equal	if (Z=1) then $PC = PC + k + 1$	None	1/2
BRNE	k	Branch if Not Equal	if (Z=0) then $PC = PC + k + 1$	None	1/2
BRCS	k	Branch if Carry Set	if (C=1) then $PC = PC + k + 1$	None	1/2
BRCC	k	Branch if Carry Cleared	if (C=0) then $PC = PC + k + 1$	None	1/2
BRSH	k	Branch if Same or Higher	if (C=0) then $PC = PC + k + 1$	None	1/2
BRLO	k	Branch if Lower	if (C=1) then $PC = PC + k + 1$	None	1/2
BRMI	k	Branch if Minus	if (N=1) then $PC = PC + k + 1$	None	1/2
BRPL	k	Branch if Plus	if (N=0) then $PC = PC + k + 1$	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N V=0) then $PC = PC + k + 1$	None	1/2
BRLT	k	Branch if Less Than, Signed	if (N V=1) then $PC = PC + k + 1$	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H=1) then $PC = PC + k + 1$	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H=0) then $PC = PC + k + 1$	None	1/2
BRTH	k	Branch if T Flag Set	if (T=1) then $PC = PC + k + 1$	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T=0) then $PC = PC + k + 1$	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V=1) then $PC = PC + k + 1$	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V=0) then $PC = PC + k + 1$	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I=1) then $PC = PC + k + 1$	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I=0) then $PC = PC + k + 1$	None	1/2
DATA TRANSFER INSTRUCTONS					

MOV	Rd,Rr	Copy Register	$Rd = Rr$	None	1
LDI	Rd,K	Load Immediate	$Rd = K$	None	1
LDS	Rd,k	Load Direct	$Rd = (k)$	None	3
LD	Rd,X	Load Indirect	$Rd = (X)$	None	2
LD	Rd,X+	Load Indirect and Post-Increment	$Rd = (X), X = X + 1$	None	2
LD	Rd,-X	Load Indirect and Pre-Decrement	$X = X - 1, Rd = (X)$	None	2
LD	Rd,Y	Load Indirect	$Rd = (Y)$	None	2
LD	Rd,Y+	Load Indirect and Post-Increment	$Rd = (Y), Y = Y + 1$	None	2
LD	Rd,-Y	Load Indirect and Pre-Decrement	$Y = Y - 1, Rd = (Y)$	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	$Rd = (Y + q)$	None	2
LD	Rd,Z	Load Indirect	$Rd = (Z)$	None	2
LD	Rd,Z+	Load Indirect and Post-Increment	$Rd = (Z), Z = Z + 1$	None	2
LD	Rd,-Z	Load Indirect and Pre-Decrement	$Z = Z - 1, Rd = (Z)$	None	2
LDD	Rd,Z+q	Load Indirect with Displacement	$Rd = (Z + q)$	None	2
STS	k,Rr	Store Direct	$(k) = Rr$	None	3
ST	X,Rr	Store Indirect	$(X) = Rr$	None	2
ST	X+,Rr	Store Indirect and Post-Increment	$(X) = Rr, X = X + 1$	None	2
ST	-X,Rr	Store Indirect and Pre-Decrement	$X = X - 1, (X) = Rr$	None	2
ST	Y,Rr	Store Indirect	$(Y) = Rr$	None	2
ST	Y+,Rr	Store Indirect and Post-Increment	$(Y) = Rr, Y = Y + 1$	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	$Y = Y - 1, (Y) = Rr$	None	2
STD	Y+q,Rr	Store Indirect with Displacement	$(Y + q) = Rr$	None	2
ST	Z,Rr	Store Indirect	$(Z) = Rr$	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	$(Z) = Rr, Z = Z + 1$	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	$Z = Z - 1, (Z) = Rr$	None	2
STD	Z+q,Rr	Store Indirect with Displacement	$(Z + q) = Rr$	None	2
LPM		Load Program Memory	$R0 = (Z)$	None	3
IN	Rd,P	InPort	$Rd = P$	None	1
OUT	P,Rr	OutPort	$P = Rr$	None	1
PUSH	Rr	Push Register on Stack	$STACK = Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd = STACK$	None	2
BITAND BIT-TEST INSTRUCTONS					
LSL	Rd	Logical Shift Left	$Rd(n+1) = Rd(n), Rd(0) = 0, C = Rd(7)$	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	$Rd(n) = Rd(n+1), Rd(7) = 0, C = Rd(0)$	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) = C, Rd(n+1) = Rd(n), C = Rd(7)$	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) = C, Rd(n) = Rd(n+1), C = Rd(0)$	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) = Rd(n+1), n=0..6$	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftrightarrow Rd(7..4)$	None	1
BSET	s	Flag Set	$SREG(s) = 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) = 0$	SREG(s)	1
SBI	P,b	Set Bit in I/O Register	$I/O(P,b) = 1$	None	2
CBI	P,b	Clear Bit in I/O Register	$I/O(P,b) = 0$	None	2
BST	Rr,b	Bit Store from Register to T	$T = Rr(b)$	T	1
BLD	Rd,b	Bit load from T to Register	$Rd(b) = T$	None	1
SEC		Set Carry	$C = 1$	C	1
CLC		Clear Carry	$C = 0$	C	1
SEN		Set Negative Flag	$N = 1$	N	1
CLN		Clear Negative Flag	$N = 0$	N	1
SEZ		Set Zero Flag	$Z = 1$	Z	1
CLZ		Clear Zero Flag	$Z = 0$	Z	1
SEI		Global Interrupt Enable	$I = 1$	I	1
CLI		Global Interrupt Disable	$I = 0$	I	1
SES		Set Signed Test Flag	$S = 1$	S	1
CLS		Clear Signed Test Flag	$S = 0$	S	1
SEV		Set Twos Complement Overflow	$V = 1$	V	1
CLV		Clear Twos Complement Overflow	$V = 0$	V	1
SET		Set T in SREG	$T = 1$	T	1
CLT		Clear T in SREG	$T = 0$	T	1

SHE	SetHalfCarryFlaginSREG	H = 1	H	1
CLH	ClearHalfCarryFlaginSREG	H = 0	H	1
NOP	NoOperation		None	1
SLEEP	Sleep		None	1
WDR	WatchdogReset		None	1

*) Not available in base-line microcontrollers

The Assembler is not case sensitive.

The operands have the following forms:

Rd: R0-R31 or R16-R31 (depending on instruction)

Rr: R0-R31

b: Constant (0-7)

s: Constant (0-7)

P: Constant (0-31/63)

K: Constant (0-255)

k: Constant, value range depending on instruction.

q: Constant (0-63)

Rd: R24, R26, R28, R30. For ADIW and SBIW instructions

Mixing ASM and BASIC

BASC0M allows you to mix BASIC with assembly.

This can be very usefull in some situations when you need full control of the generated code.

Almost all assembly mnemonics are recognized by the compiler. The exceptions are : SUB, SWAP and OUT. These are BASIC reserved words and have priority over the ASM mnemonics. To use these mnemonics precede them with the ! - sign.

For example :

```
Dim a As Byte At &H60 'A is stored at location &H60
Ldi R27 , $00         'Load R27 with MSB of address
Ldi R26 , $60         'Load R26 with LSB of address
Ld R1, X              'load memory location $60 into R1
!SWAP R1              'swapnibbles
```

As you can see the SWAP mnemonic is preceded by a ! sign.

Another option is to use the assembler block directives:

```
$ASM
Ldi R27 , $00         'Load R27 with MSB of address
Ldi R26 , $60         'Load R26 with LSB of address
Ld R1, X              'load memory location $60 into R1
SWAP R1              'swapnibbles
$END ASM
```

A special assembler helper function is provided to load the address in to the

register X or Z. Y can not be used because it is used as the stack pointer.

```
Dim A As Byte        'reserve space
LOADADR a, X        'load address of variable named A into registerpair
X
```

This has the same effect as :

```
Ldi R26 , $60 'for example !
```

```
Ldi R27, $00 'for example !
```

Some registers are used by BASCOM

R4 and R5 are used to point to the stack frame

R6 is used to store some bit variables:

R6 bit 0 = flag for int/word conversion

R6 bit 1 = temp bit space used for swapping bits

R6 bit 2 = error bit (ERR variable)

R6 bit 3 = show/noshow flag when using INPUT statement

R8 and R9 are used as a datapointer for the READ statement.

All other registers are used depending on the used statements.

To Load the address of a variable you must enclose them in brackets.

Dim B As Bit

Lds R16, {B} 'will replace {B} with the address of variable B

To refer to the bitnumber you must precede the variable name by BIT.

Sbrs R16 , **BIT.B** 'notice the point!

Since this was the first dimensioned bit the bitnumber is 7. Bits are stored in bytes and the first dimensioned bit goes in the LS bit.

Sample Electronics cable programmer

The simple cable programmer was submitted by Sample Electronics. They produce professional programmers too. This simple programmer you can make yourself within a 10 minutes.

What you need is a DB25 centronics male connector, a flatcable and a connector that can be connected to the target MCU board.

The connections to make are as following:

DB25 pin	Target MCU pin(AT90S8535)	DT104
2, D0	MOSI, pin 6	J5, pin 4
4, D2	RESET, pin 9	J5, pin 8
5, D3	CLOCK, pin 8	J5, pin 6

11, BUSY	MISO, pin 7	J5, pin 5
18-25,GND	GROUND	J5, pin 1

The MCU pin numbers are shown for an 8535!

Note that 18-25 means pins 18,19,20,21,22,23,24 and 25
 You can use a small resistor of 100 ohm in series with the D0, D2 and D3 line in order not to short circuit your LPT port in the event the MCU pins are high.
 But it was tested without these resistors and my PC still works :-)

Tip : when testing programmers etc. on the LPT it is best to buy a I/O card for your PC that has a LPT port. This way you dont destroy your LPT port that is on the motherboard in the event you make a mistake!

SPIMOVE

Action

Sends and receives value or a variable to the SPI-bus.

Syntax

```
var = SPIMOVE( byte )
```

Remarks

var	The variable that is assigned with the received byte from the SPI-bus.
byte	The variable or constant whose content must be send to the SPI-bus.

Seealso

SPIIN »Page 178 , SPIINIT »Page 179 , CONFIG SPI »Page 101

Example

```
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
INITSPI
Dim a(10) as Byte , X As Byte
SPIOUT a(1) , 5 'send 5 bytes
SPIOUT X , 1 'send 1 byte
A(1) = SpiMove(5) ' move 5 to SPI and store result in a(1)
End
```

Action

Returns the position of a substring in a string.

Syntax

```
var = INSTR( start , string , substr )
var = INSTR( string , substr )
```

Remarks

Var	Numeric variable that will be assigned with the position of the substring in the string. Returns 0 when the substring is not found.
Start	An optional numeric parameter that can be assigned with the first position where must be searched in the string. By default (when not used) the whole string is searched starting from position 1.
String	The string to search.
Substr	The search string.

No constant can be used for string it must be a string.
 Only substr can be either a string or a constant.

Seealso

Example

```
Dim S As String * 10 , Z as String * 5
Dim bP as Byte
s = "This is a test"
Z = "is"
bP = Instr(s,z) : Print bP 'should print 3
bP = Instr(4,s,z) : Print bP 'should print 6
End
```

Index

—\$—

\$ASM, 64
 \$BAUD, 64
 \$CRYSTAL, 65
 \$DATA, 65
 \$DEFAULT, 66
 \$EEPROM, 67
 \$EXTERNAL, 68
 \$INCLUDE, 69
 \$LCD, 69
 \$LCDPUTCTRL, 70
 \$LCDPUTDATA, 71
 \$LCDRS, 72
 \$LIB, 73
 \$REGFILE, 75
 \$SERIALINPUT, 75
 \$SERIALINPUT2LCD, 77
 \$SERIALOUTPUT, 78
 \$XRAMSIZE, 78
 \$XRAMSTART, 79

—1—

1WREAD, 81
 1WRESET, 80
 1WWRITE, 82

—A—

A word of thank, 13
 ABS, 84
 AdditionalHardware, 44
 ALIAS, 83
 ASC, 85
 Assembler mnemonics, 194
 Attaching an LCDDisplay, 53
 AVRInternalHardware, 45
 AVRInternalHardwarePortB, 50
 AVRInternalHardwarePortD, 52
 AVRInternalHardwareTIMER1, 49
 AVRInternalHardwareWatchdogtimer, 50
 AVRInternalRegisters, 46

—B—

BASCOMEditorKeys, 39
 BAUD, 85
 BCD, 86
 BITWAIT, 87
 BYVAL, 88

—C—

CALL, 88
 CASE, 173
 ChangescomparedtoBASCOM-8051, 191
 CHR, 90
 CLOCKDIVISION, 91
 CLOSE, 92
 CLS, 91
 CONFIG, 93
 CONFIG1WIRE, 94
 CONFIGDEBOUNCE, 94
 CONFIGI2CDELAY, 95
 CONFIGINTx, 96
 CONFIGKBD, 96
 CONFIGLCD, 97
 CONFIGLCDBUS, 97
 CONFIGLCDMODE, 98
 CONFIGLCDPIN, 99
 CONFIGPORT, 107
 CONFIGSCL, 100
 CONFIGSDA, 99
 CONFIGSPI, 101
 CONFIGTIMER0, 101
 CONFIGTIMER1, 103
 CONFIGWAITSUART, 106
 CONFIGWATCHDOG, 106
 CONST, 122
 Constants, 40
 COUNTER0andCOUNTER1, 109
 CPEEK, 110
 CRYSTAL, 111
 CURSOR, 112

—D—

DATA, 112
 DEBOUNCE, 114
 DECLAREFUNCTION, 116
 DECLARESUB, 118
 DECR, 116
 DEFBIT, 119
 DEFINT, 119
 DEFLCDCHAR, 119
 DEFLNG, 119
 DEFSNG, 119
 DEFWORD, 119
 DEFxxx, 119
 DELAY, 120
 DevelopingOrder, 40
 DIM, 120
 DISABLE, 123
 DISPLAY, 125
 DO, 125
 DOWNT0, 128

—E—

EditCopy, 19
 EditCut, 19
 EditFind, 20
 EditFindNext, 20
 EditGoto, 20
 EditGotoBookmark, 21
 EditIndentBlock, 21
 EditPaste, 20
 EditRedo, 19
 EditReplace, 20
 EditToggleBookmark, 20
 EditUndo, 19
 EditUnindentBlock, 21
 ELSE, 126, 142
 ENABLE, 126
 END, 127
 ENDF, 142
 ENDSELECT, 173
 ERAM, 40
 ErrorCodes, 42
 EXIT, 128

—F—

FileClose, 18
 FileExit, 19
 FileNew, 17
 FileOpen, 17
 FilePrint, 19
 FilePrintPreview, 18
 FileSave, 18
 FileSaveAs, 18
 FOR, 128
 FOR-NEXT, 128
 FOURTHLINE, 129
 FUSING, 130

—G—

GETAD, 130
 GETKBD, 131
 GETRC0, 132
 GETRC5, 133
 GOSUB, 135
 GOTO, 136

—H—

HelpAbout, 38
 HelpCredits, 39
 HelpIndex, 38
 Help on Help, 38
 HEX, 137
 HEXVAL, 137
 HIGH, 138

HOME, 138

—I—

I2CRBYTE, 140
 I2CRECEIVE, 139
 I2CSEND, 139
 I2CSTART, 140
 I2CSTOP, 140
 I2CWBYTE, 140
 I2START, I2CSTOP, I2CRBYTE, I2CWBYTE, 140
 IDLE, 141
 IF, 142
 IF-THEN-ELSE-ENDIF, 142
 INCR, 143
 Index, 1
 INKEY, 143
 INP, 144
 INPUT, 146
 INPUTBIN, 144
 INPUTHEX, 145
 Installation, 6
 INSTR, 200
 ISPprogrammer, 192

—L—

LanguageFundamentals, 56
 LCD, 147
 LEFT, 150
 LEN, 150
 LOAD, 151
 LOADADR, 191
 LOCAL, 152
 LOCATE, 154
 LOOKUP, 154
 LOOKUPSTR, 155
 LOOP, 125
 LOW, 156
 LOWERLINE, 156
 LTRIM, 151

—M—

MAKEBCD, 157
 MAKEDEC, 158
 MAKEINT, 157
 Memoryusage, 40
 MID, 158
 MixingASMandBASIC, 197

—N—

NEXT, 128

O

ONINTERRUPT,159
 ONVALUE,160
 OPEN,161
 OptionsCommunication,32
 OptionsCompiler,27,30
 OptionsCompiler1WIRE,30
 OptionsCompilerChip,27
 OptionsCompilerCommunication,29
 OptionsCompilerI2C,30
 OptionsCompilerLCD,31
 OptionsCompilerOutput,28
 OptionsCompilerSPI,30
 OptionsEnvironment,33
 OptionsMonitor,36
 OptionsPrinter,37
 OptionsProgrammer,36
 OptionsSimulator,35
 OUT,162

P

PEEK,163
 PG302programmer,194
 POKE,163
 POPALL,164
 PowerUp,55
 POWERDOWN,164
 POWERSAVE,164
 PRINT,165
 PRINTBIN,166
 ProgramCompile,21
 ProgramSendtoChip,23
 ProgramShowResult,22
 ProgramSimulate,23
 ProgramSyntaxCheck,22
 PUSHALL,166

R

READ,167
 READEEPROM,168
 REM,168
 Resellers,9
 ReservedWords,56
 RESET,169
 RESTORE,170
 RETURN,170
 RIGHT,171
 ROTATE,172
 RTRIM,171
 RunningBASC0M-AVR,17

S

SampleElectronicscableprogrammer,198

SELECT,173
 SELECT-CASE-ENDSELECT,173
 SET,173
 SETUP,6
 SHIFT,174
 SHIFTCURSOR,175
 SHIFTLCD,177
 SHIFTOUT,176
 SOUND,177
 SPACE,178
 SPIIN,178
 SPIINIT,179
 SPIMOVE,199
 SPIOUT,180
 START,180
 STEP,128
 STOP,181
 STR,182
 STRING,183
 SUB,184
 SupportedProgrammers,193
 SWAP,184

T

THEN,142
 THIRDLINe,184
 TIMER0,48
 ToolsLCDDesigner,26
 ToolsTerminalEmulator,25
 TRIM,185

U

UPPERLINE,186
 Usingthe1WIREprotocol,55
 UsingtheI2Cprotocol,54
 UsingtheSPIprotocol,55

V

VAL,186
 VARPTR,187

W

WAIT,187
 WAITKEY,188
 WAITMS,188
 WAITUS,189
 WEND,189
 WHILE,189
 WHILE-WEND,189
 WindowArrangeIcons,37
 WindowMinimizeAll,37

X

WindowTile,37
 WindowsCascade,37
 WRITEEEPROM,190

XRAM,40