Dear reader.

Thank you for your interest in BASCOM.

BASCOM was "invented" in 1995. It was intended for personal usage only. I decided to make it public as I found no other tool that was so simple to use. Since that time, a lot of options and extensions were added. Without the help and patience of the many users, BASCOM would not be what it is today: "the best and most affordable tool for fast prototyping".

We hope that BASCOM will contribute in making your work with microprocessors Easy and enjoyable.

The MCS Electronics Team
BASCOM-8051

© 2007 MCS Electronics

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: september 2007 in (whereever you are located)

Publisher
MCS Electronics

Managing Editor
M.C.Alberts

Technical Editors
M.C.Alberts

Cover Designer
B.F.de Graaff

Special thanks to:
All the people who contributed to this document, all the forum members that contributed in a positive way, all beta testers, and all customers.
Table of Contents

Foreword 0

Part I INDEX 13
  1 Table of contents .................................................................................................................. 14
  2 Keyword Reference ............................................................................................................. 17

Part II Installing BASCOM-8051 21

Part III Updates 27

Part IV BASCOM IDE 37
  1 RUNNING BASCOM-8051 ...................................................................................................... 37
  2 BASCOM IDE ..................................................................................................................... 38
  3 File New ............................................................................................................................ 39
  4 File Open .......................................................................................................................... 39
  5 File Close .......................................................................................................................... 40
  6 File Save ........................................................................................................................... 40
  7 File Save As... .................................................................................................................. 40
  8 File Print Preview ............................................................................................................ 40
  9 File Print .......................................................................................................................... 41
 10 File Exit ............................................................................................................................ 42
 11 Edit Undo ........................................................................................................................ 42
 12 Edit Redo ........................................................................................................................ 42
 13 Edit Copy ........................................................................................................................ 42
 14 Edit Cut ............................................................................................................................ 42
 15 Edit Paste ........................................................................................................................ 43
 16 Edit Find .......................................................................................................................... 43
 17 Edit Find Next ................................................................................................................ 43
 18 Edit Replace .................................................................................................................... 44
 19 Edit Goto ........................................................................................................................ 44
 20 Edit Indent Block .......................................................................................................... 45
 21 Edit Unindent Block ...................................................................................................... 45
 22 Editor Keys ...................................................................................................................... 46
 23 Program Compile ............................................................................................................ 47
 24 Program Syntax check .................................................................................................. 47
 25 Program Show Result ................................................................................................... 48
 26 Program Simulate ......................................................................................................... 49
 27 Program Send to chip ................................................................................................... 52
 28 Tools Terminal Emulator ............................................................................................. 53
 29 Tools LCD designer ....................................................................................................... 54

© 2007 MCS Electronics
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part VI</td>
<td>BASCOM Language Reference</td>
</tr>
<tr>
<td>1</td>
<td>BASCOM Statements</td>
</tr>
<tr>
<td>2</td>
<td>#IF</td>
</tr>
<tr>
<td>3</td>
<td>#ELSE</td>
</tr>
<tr>
<td>4</td>
<td>#ENDIF</td>
</tr>
<tr>
<td>5</td>
<td>1WIRE</td>
</tr>
<tr>
<td>6</td>
<td>1WIRECOUNT</td>
</tr>
<tr>
<td>7</td>
<td>1WSEARCHFIRST</td>
</tr>
<tr>
<td>8</td>
<td>1WSEARCHNEXT</td>
</tr>
<tr>
<td>9</td>
<td>$ASM - $END ASM</td>
</tr>
<tr>
<td>10</td>
<td>$BAUD</td>
</tr>
<tr>
<td>11</td>
<td>$BGF</td>
</tr>
<tr>
<td>12</td>
<td>$CRYSTAL</td>
</tr>
<tr>
<td>13</td>
<td>$END</td>
</tr>
<tr>
<td>14</td>
<td>BASCOM Statements</td>
</tr>
<tr>
<td>15</td>
<td>#IF</td>
</tr>
<tr>
<td>16</td>
<td>#ELSE</td>
</tr>
<tr>
<td>17</td>
<td>#ENDIF</td>
</tr>
<tr>
<td>18</td>
<td>1WIRE</td>
</tr>
<tr>
<td>19</td>
<td>1WIRECOUNT</td>
</tr>
<tr>
<td>20</td>
<td>1WSEARCHFIRST</td>
</tr>
<tr>
<td>21</td>
<td>1WSEARCHNEXT</td>
</tr>
<tr>
<td>22</td>
<td>$ASM - $END ASM</td>
</tr>
<tr>
<td>23</td>
<td>$BAUD</td>
</tr>
<tr>
<td>24</td>
<td>$BGF</td>
</tr>
<tr>
<td>25</td>
<td>$CRYSTAL</td>
</tr>
<tr>
<td>26</td>
<td>$END</td>
</tr>
<tr>
<td>Page</td>
<td>Content</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>55</td>
<td>CONFIG BAUD</td>
</tr>
<tr>
<td>56</td>
<td>CONFIG BAUD1</td>
</tr>
<tr>
<td>57</td>
<td>CONFIG DEBOUNCE</td>
</tr>
<tr>
<td>58</td>
<td>CONFIG I2CDELAY</td>
</tr>
<tr>
<td>59</td>
<td>CONFIG GETRC</td>
</tr>
<tr>
<td>60</td>
<td>CONFIG GRAPHLCD</td>
</tr>
<tr>
<td>61</td>
<td>CONFIG LCDPIN</td>
</tr>
<tr>
<td>62</td>
<td>CONFIG LCD</td>
</tr>
<tr>
<td>63</td>
<td>CONFIG LCDBUS</td>
</tr>
<tr>
<td>64</td>
<td>CONFIG MICROHIRE</td>
</tr>
<tr>
<td>65</td>
<td>CONFIG PRINT</td>
</tr>
<tr>
<td>66</td>
<td>CONFIG SCL</td>
</tr>
<tr>
<td>67</td>
<td>CONFIG SDA</td>
</tr>
<tr>
<td>68</td>
<td>CONFIG SERVOS</td>
</tr>
<tr>
<td>69</td>
<td>CONFIG SPI</td>
</tr>
<tr>
<td>70</td>
<td>CONFIG TIMER0, TIMER1</td>
</tr>
<tr>
<td>71</td>
<td>CONFIG WATCHDOG</td>
</tr>
<tr>
<td>72</td>
<td>COUNTER</td>
</tr>
<tr>
<td>73</td>
<td>CPEEK</td>
</tr>
<tr>
<td>74</td>
<td>CURSOR</td>
</tr>
<tr>
<td>75</td>
<td>DATA</td>
</tr>
<tr>
<td>76</td>
<td>DEBOUNCE</td>
</tr>
<tr>
<td>77</td>
<td>DECR</td>
</tr>
<tr>
<td>78</td>
<td>DECLARE</td>
</tr>
<tr>
<td>79</td>
<td>DEF</td>
</tr>
<tr>
<td>80</td>
<td>DEFLCDCHAR</td>
</tr>
<tr>
<td>81</td>
<td>DELAY</td>
</tr>
<tr>
<td>82</td>
<td>DIM</td>
</tr>
<tr>
<td>83</td>
<td>DISABLE</td>
</tr>
<tr>
<td>84</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>85</td>
<td>DO</td>
</tr>
<tr>
<td>86</td>
<td>ELSE</td>
</tr>
<tr>
<td>87</td>
<td>ENABLE</td>
</tr>
<tr>
<td>88</td>
<td>END</td>
</tr>
<tr>
<td>89</td>
<td>END IF</td>
</tr>
<tr>
<td>90</td>
<td>ERASE</td>
</tr>
<tr>
<td>91</td>
<td>EXIT</td>
</tr>
<tr>
<td>92</td>
<td>FOR</td>
</tr>
<tr>
<td>93</td>
<td>FOURTHLINE</td>
</tr>
<tr>
<td>94</td>
<td>FUSING</td>
</tr>
<tr>
<td>95</td>
<td>GET</td>
</tr>
<tr>
<td>96</td>
<td>GETAD</td>
</tr>
<tr>
<td></td>
<td>Function</td>
</tr>
<tr>
<td>---</td>
<td>-------------------</td>
</tr>
<tr>
<td>97</td>
<td>GETAD2051</td>
</tr>
<tr>
<td>98</td>
<td>GETRC</td>
</tr>
<tr>
<td>99</td>
<td>GETRC5</td>
</tr>
<tr>
<td>100</td>
<td>GOSUB</td>
</tr>
<tr>
<td>101</td>
<td>GOTO</td>
</tr>
<tr>
<td>102</td>
<td>HEX</td>
</tr>
<tr>
<td>103</td>
<td>HEXVAL</td>
</tr>
<tr>
<td>104</td>
<td>HIGH</td>
</tr>
<tr>
<td>105</td>
<td>HIGHW</td>
</tr>
<tr>
<td>106</td>
<td>HOME</td>
</tr>
<tr>
<td>107</td>
<td>I2CRECEIVE</td>
</tr>
<tr>
<td>108</td>
<td>I2CSEEND</td>
</tr>
<tr>
<td>109</td>
<td>I2C</td>
</tr>
<tr>
<td>110</td>
<td>IDLE</td>
</tr>
<tr>
<td>111</td>
<td>IF</td>
</tr>
<tr>
<td>112</td>
<td>INCR</td>
</tr>
<tr>
<td>113</td>
<td>INKEY</td>
</tr>
<tr>
<td>114</td>
<td>INP</td>
</tr>
<tr>
<td>115</td>
<td>INPUT</td>
</tr>
<tr>
<td>116</td>
<td>INPUTBIN</td>
</tr>
<tr>
<td>117</td>
<td>INPUTHEX</td>
</tr>
<tr>
<td>118</td>
<td>INSTR</td>
</tr>
<tr>
<td>119</td>
<td>LCASE</td>
</tr>
<tr>
<td>120</td>
<td>LCD</td>
</tr>
<tr>
<td>121</td>
<td>LCDINIT</td>
</tr>
<tr>
<td>122</td>
<td>LCDHEX</td>
</tr>
<tr>
<td>123</td>
<td>LEFT</td>
</tr>
<tr>
<td>124</td>
<td>LEN</td>
</tr>
<tr>
<td>125</td>
<td>LOAD</td>
</tr>
<tr>
<td>126</td>
<td>LOCATE</td>
</tr>
<tr>
<td>127</td>
<td>LOOKUP</td>
</tr>
<tr>
<td>128</td>
<td>LOOKUPSTR</td>
</tr>
<tr>
<td>129</td>
<td>LOW</td>
</tr>
<tr>
<td>130</td>
<td>LOWW</td>
</tr>
<tr>
<td>131</td>
<td>LOWERLINE</td>
</tr>
<tr>
<td>132</td>
<td>MAKEBCD</td>
</tr>
<tr>
<td>133</td>
<td>MAKEDEC</td>
</tr>
<tr>
<td>134</td>
<td>MAKEINT</td>
</tr>
<tr>
<td>135</td>
<td>MAX</td>
</tr>
<tr>
<td>136</td>
<td>MID</td>
</tr>
<tr>
<td>137</td>
<td>MIN</td>
</tr>
<tr>
<td>138</td>
<td>MOD</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
</tr>
</tbody>
</table>

| 139 | MWINIT ................................................................................................................................... 193 |
| 140 | MWREAD ................................................................................................................................... 193 |
| 141 | MWWOPCODE ................................................................................................................................... 194 |
| 142 | MWRITE ................................................................................................................................... 195 |
| 143 | NEXT ................................................................................................................................... 196 |
| 144 | ON interrupt ................................................................................................................................... 196 |
| 145 | ON value ................................................................................................................................... 197 |
| 146 | OPEN ................................................................................................................................... 198 |
| 147 | OUT ................................................................................................................................... 200 |
| 148 | PORT ................................................................................................................................... 201 |
| 149 | PEEK ................................................................................................................................... 202 |
| 150 | POKE ................................................................................................................................... 202 |
| 151 | POWERDOWN ................................................................................................................................... 203 |
| 152 | PRINT ................................................................................................................................... 203 |
| 153 | PRINTBIN ................................................................................................................................... 204 |
| 154 | PRINTHEX ................................................................................................................................... 205 |
| 155 | PRIORITY ................................................................................................................................... 206 |
| 156 | PSET ................................................................................................................................... 207 |
| 157 | PUT ................................................................................................................................... 207 |
| 158 | READ ................................................................................................................................... 208 |
| 159 | READMAGCARD ................................................................................................................................... 209 |
| 160 | REM ................................................................................................................................... 211 |
| 161 | REPLACE ................................................................................................................................... 211 |
| 162 | RESET ................................................................................................................................... 212 |
| 163 | RESTORE ................................................................................................................................... 212 |
| 164 | RETURN ................................................................................................................................... 213 |
| 165 | RIGHT ................................................................................................................................... 214 |
| 166 | RND ................................................................................................................................... 214 |
| 167 | ROTATE ................................................................................................................................... 215 |
| 168 | SELECT ................................................................................................................................... 216 |
| 169 | SET ................................................................................................................................... 216 |
| 170 | SHIFTCURSOR ................................................................................................................................... 217 |
| 171 | SHIFT ................................................................................................................................... 217 |
| 172 | SHIFTIN ................................................................................................................................... 218 |
| 173 | SHIPELCD ................................................................................................................................... 219 |
| 174 | SHOWPIC ................................................................................................................................... 220 |
| 175 | SOUND ................................................................................................................................... 220 |
| 176 | SOUNDEXT ................................................................................................................................... 221 |
| 177 | SPACE ................................................................................................................................... 222 |
| 178 | SPC ................................................................................................................................... 223 |
| 179 | SPIIN ................................................................................................................................... 224 |
| 180 | SPIINIT ................................................................................................................................... 224 |
Part VII Using assembly

1 Using assembly ................................................................. 239
2 Internal registers ............................................................... 246
3 Initialization ........................................................................ 248

Part VIII Additional Hardware

1 Additional Hardware .......................................................... 251
2 Alternative port-pin functions ........................................... 256
3 Hardware - LCD display .................................................... 257
4 Hardware - I2C ................................................................. 258
5 1WIRE INFO .................................................................... 258

Part IX Supported Programmers

1 MCS Flash programmer ..................................................... 264
2 MCS SPI programmer ........................................................ 266
3 Blow IT Flashprogrammer ............................................... 267
4 PG2051 flash programmer ............................................... 267
5 PG302 programmer .......................................................... 268
6 SE512 or SE514 programmer ........................................... 269
7 SE-812 ............................................................................. 270
8 Sample Electronics ISP programmer .................................. 270
9 CYGNAL JTAG Programmer ............................................. 272
10 Futurelec ......................................................................... 272
11 JPK Systems X-programmer ................................................................. 272
12 Peter Averill's TAFE programmer ....................................................... 273
13 STK200/300 ISP Programmer .............................................................. 274
14 Rhombus SCE-51 ............................................................................... 275
15 SE511-SE516 programmer ................................................................. 276

Part X  BASCOM Misc  
1 Error messages .................................................................................. 279
2 Compiler Limits .................................................................................. 282
3 Reserved Words .................................................................................. 283

Part XI  Microprocessor support  
1 Microprocessor support ...................................................................... 288
2 TIMER2 ............................................................................................... 289
3 DATA EEPROM .................................................................................. 292
4 AT898252 WATCHDOG ....................................................................... 292
5 WATCHDOG 80515 .............................................................................. 293
6 INTERRUPTS and PRIORITY 80515 ..................................................... 293
7 INTERRUPTS and PRIORITY 80537 ..................................................... 294
8 ADUC 812 ............................................................................................ 294
9 89C51 ................................................................................................. 297

Part XII  International Resellers  
1 International Resellers ........................................................................ 300

Part XIII  Third party hardware  
1 Third party Hardware .......................................................................... 302

Grifo ....................................................................................................... 302
Rhombus ............................................................................................... 309

Index  

© 2007 MCS Electronics
Part I
1 INDEX

© 1995-2007 MCS Electronics

Help Version 2.0.14.0

See Installing BASCOM-8051 for the installation procedure

MCS Electronics may update this documentation without notice. Products specification and usage may change accordingly. MCS Electronics will not be liable for any miss-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.

© MCS Electronics. All rights reserved.
# 1.1 Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing BASCOM-8051</td>
<td>27</td>
</tr>
<tr>
<td>BASCOM IDE</td>
<td>38</td>
</tr>
<tr>
<td>Running BASCOM 8051</td>
<td>37</td>
</tr>
<tr>
<td>Updates</td>
<td>37</td>
</tr>
</tbody>
</table>

## File

- **File New**
- **File Open**
- **File Close**
- **File Save**
- **File Save As...**
- **File Print**
- **File Print Preview**
- **File Exit**

## Edit

- **Edit Undo**
- **Edit Redo**
- **Edit Copy**
- **Edit Cut**
- **Edit Paste**
- **Edit Find**
- **Edit Find Next**
- **Edit Replace**
- **Editor Keys**
- **Edit Goto**
- **Edit Indent**
- **Edit Unindent**

## Program

- **Compile**
- **Syntax check**
- **Show Result**
- **Simulate**
- **Send to chip**

## Tools

- **Terminal Emulator**
- **LCD designer**
- **Graphic Converter**
- **LIB Manager**
- **Export to RTF**

## Options

- **Compiler Output**
- **Compiler Communication**
- **Compiler I2C**
- **Compiler LCD**
- **Compiler Misc.**
- **Communication**
- **Environment**
- **Hardware Simulator**
- **Programmer**
- **Monitor**
- **Printer**

## Window

- **Cascade**
- **Tile**
- **Arrange Icons**
- **Minimize all**

## Help

- **About**
- **Index**
- **Help on Help**
- **Credits**
- **Forum**
- **Shop**
- **Support**

## BASCOM Statements

- **Language Fundamentals**
- **Internal Registers**
- **Initialization**
Microprocessor support

Reserved Words

Errors

Programmers

Additional hardware

Compiler limits

ASM programming

International Resellers

Available third party hardware

Language Reference

-1-

1WRESET, 1WREAD, 1WRITE, 1WSEARCHFIRST, 1WSEARCHNEXT, 1WIRECOUNT

-COMpiler DIRECtives-

#IF, #ELSE, #ENDIF, $ASM - $END ASM, $INCLUDE, $BAUD, $BGF, $CRYSTAL, $DEFAULT XRAM, $IRAMSTART, $LARGE, $LCD, $MAP, $NOBREAK, $NOINIT, $NONAN, $NONULL, $NORAMCLEAR, $NOSP, $OBJ, $RAMSIZE, $RAMSTART, $REGFILE, $ROMSTART, $SERIALINPUT, $SERIALINPUT2LCD, $SERIALOUTPUT, $SIM

-A-

ABS, ALIAS, ASC, AVG

-B-

BITWAIT, BCD, BREAK

-C-

CALL, CLOSE, CLS, CHR, CONFIG, CONST, COUNTER, CPEEK, CURSOR

-D-

DATA, DEBOUNCE, DECR, DECLARE, DEFINT, DEFBIT, DEFBYTE, DEFLCDCCHAR, DEFWORD, DELAY, DIM, DISABLE, DISPLAY, DO

-E-

ELSE, ENABLE, END, END IF, ERASE, EXIT

-F-

FOR, FOURTHLINE, FUSING
-G-
GET, GETAD, GETAD2051, GETC, GETRC, GETRC5, GOSUB, GOTO

-H-
HEX, HEXVAL, HIGH, HIGHW, HOME

-I-
I2CRECEIVE, I2CSEND, I2CSTART, I2CSTOP, I2CRBYTE, I2CWBYTE, IDLE, IF, INCR, INKEY, INPUT, INPUTB, INPUTBIN, INPUTHEX, INSTR

-L-
LCASE, LCD, LCDINIT, LCDHEX, LEFT, LEN, LOAD, LOCATE, LOOKUP, LOOKUPSTR, LOOP, LOW, LOWW, LOWERLINE

-M-
MAKEDEC, MAKEBCD, MAKEINT, MAX, MID, MIN, MOD

-N-
NEXT

-O-
ON Interrupt, ON Value, OPEN, OUT

-P-
P1, P3, PEEK, POKE, PSET, POWERDOWN, PRINT, PRINTBIN, PRINTHEX, PRIORITY, PUT

-R-
READ, READMAGCARD, REM, REPLACE, RESEt, RESTORE, RETURN, RIGHT, RND, ROTATE

-S-
SELECT, SET, SHIFT, SHIFTCURSOR, SHIFTIN, SHIFTOUT, SHIFTLCD, SHOWPIC, SOUND, SOUNDEXTP, SPACE, SPC, SPIIN, SPIOUT, START, STOP, STOP TIMER, STR, STRING, SUB

-T-
THEN, THIRDLINE, TO

-U-
UCASE, UPPERCASE

-V-
VAL, VARPTR
-W-
WAIT, WAITKEY, WAITMS, WHILe .. WEND

1.2 Keyword Reference

1Wire
1Wire routines allow you to communicate with Dallas 1wire chips.
1WRESET, 1WREAD, 1WRITE, 1WSEARCHFIRST, 1WSEARCHNEXT, 1WIRECOUNT

Conditions
Conditions execute a part of the program depending on the condition
IF, ELSE, END IF, EXIT, DO, LOOP, SELECT, FOR, NEXT, TO, THEN, WHILE .. WEND

Configuration
Configuration command initialize the hardware to the desired state.
CONFIG

Conversion
A conversion routine is a function that converts a number or string.
ASC, BCD, CHR, FUSING, HEX, HEXVAL, HIGH, HIGHW, MAKEDEC, MAKEBCD, MAKEINT, STR, VAL

Delay
Delay routines delay the program for the specified time.
DELAY, WAIT, WAITMS

Directives
Directives are special instructions for the compiler. They can override a setting from the IDE.
#IF, #ELSE, #ENDIF, $ASM - $END ASM, $INCLUDE, $BAUD, $BGF, $CRYSTAL, $DEFAULT XRAM, $IRAMSTART, $LARGE, $LCD, $MAP, $NOBREAK, $NINIT, $NONAN, $NORAMCLEAR, $NOSP, $OBJ, $RAMSIZE, $RAMSTART, $SFILE, $ROMSTART, $SINPUT, $SOUTPUT, $SIM

Graphical LCD
Graphical LCD commands extend the normal text LCD commands.
PSET, SHOWPIC

I2C
I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.

I2CRECEIVE, I2CSEND, I2CSTART, I2CSTOP, I2CRBYTE, I2CWBYTE

**Interrups**

Interrupt related routines.

ON Interrupt, ENABLE, DISABLE, PRIORITY

**IO**

I/O commands are related to the I/O pins of the processor.

ALIAS, BITWAIT, DEBOUNCE, SET, RESET

**Math**

Math functions

ABS, AVG, MAX, MIN, MOD

**Micro**

Micro statements are highly related to the micro processor.

BREAK, P1,P3, IDLE, END, POWERDOWN, START, STOP, STOP TIMER

**Memory**

Memory functions set or read RAM, EEPROM or flash memory.

CPEEK, ERASE, INP, OUT, PEEK, POKE, DIM, READ, RESTORE, DATA, VARPTR

**Remote control**

Remote control statements send or receive IR commands for remote control.

GETRC5

**RS-232**

RS-232 are serial routines that use the UART or emulate a UART.

WAITKEY, PRINT, PRINTBIN, PRINTHEX, PUT, OPEN, SPC, INKEY, INPUT, INPUTBIN, INPUTHEX, GET, CLOSE

**SPI**

SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI.

SPIIN, SPIOUT, SPIINIT

**String**

String routines are used to manipulate strings.
**Text LCD**
Text LCD routines work with the normal text based LCD displays.


**Various**
This section contains all statements that were hard to put into another group

Part II
2 Installing BASCOM-8051

After you have downloaded the software you need to UNZIP the downloaded file. There is only one file named setup.exe. You may run this setup.exe from within the Windows Shell but it is important to notice that when you use the commercial version, you MUST UNZIP the setup.exe since you need to copy the license file to the same directory as setup.exe.

⚠️ You must have Administrator rights in order to be able to run setup.

The opening screen looks like:

![Welcome to the BASCOM-8051 Setup Wizard]

You need to click the Next-button to continue.

A license agreement will be shown. You need to read it and accept the agreement. This is a no-nonsense agreement where you are allowed to install/copy on as many computers as you want, providing that you use only one computer at the same time.
After clicking the 'I accept the agreement' option, you need to click the Next-button again to continue.

The readme.txt file is shown. Basically it tells you to contact support@mcselec.com in case of a problem.

Click the Next-button again to continue with the setup.
You can now select where you want to Install BASCOM-8051. The default is shown below.

![Select Destination Location](image1)

Setup will install BASCOM-8051 into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

```
C:\Program Files\MCS Electronics\BASCOM-8051
```

At least 11.8 MB of free disk space is required.

Click the Next-button again to continue.

You can now select/enter the Program Group name. The default is shown below.

![Select Start Menu Folder](image2)

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

```
MCS Electronics\BASCOM-8051
```

Click the Next-button again to continue.
The files will now be installed.
A screenshot is shown below:

When the files are installed, the installer will install some additional files:

Press the Next-Button to install the additional files. This will go quick in most cases. When you install from CD-ROM the setup will also copy PDF datasheets. The installation will take longer then.
When setup is completely finished it will show the last screen:

Completing the BASCOM-8051 Setup Wizard

To complete the installation of BASCOM-8051, Setup must restart your computer. Would you like to restart now?

☐ Yes, restart the computer now
☐ No, I will restart the computer later

You MUST reboot your PC since it will install a driver needed for the programming.
Part III
3 Updates

The update process is simple.
- Go to the main MCS website at http://www.mcselec.com
- In the left pane under 'Main Menu' you will find a link named 'Registration/Updates'

Notice that the website uses two different accounts: one for the forum/shop and one for the registration/updates. You will see the following screen:
Click the link and select 'Create new account'.

You need to provide a username, password, email and full name. Company name is optional. When you want to receive notifications when updates are available, select this option.

When you filled in the information, click 'Submit Registration'.

After you click submit, you can get various error messages. For example that a username already exists. Press the Back-button in your browser, and correct the error, then try again.

If the registration is successful you will get a message that the registration succeeded.

Now you can login. You will see the following screen:
You need to choose 'Product registration'.
The following screen will be shown:

- Select a product from the list.
- Enter the serial number

⚠️ It is important that you enter a **valid** serial number. Do not try to enter serial numbers from cracked versions. When you enter invalid serial numbers, you will lose support and the ability to update. The valid serial number is shown in the Help, About box.
When the product is selected, the serial number is entered, and you press 'Register product' you will see the following message:

- This does mean that you registered successfully.
- MCS Electronics will validate all registrations once in a few days. When the product is validated you will receive an email. After you receive the email, you can login to the registry again.
- Now you need to select 'Download LIC files'. The following screen will be shown:
At the top you can see which products are registered, and which status they have. When you want to do a FULL SETUP, you need to download the full version.
You do not need to uninstall a previous version. You can install an update into the same directory or a new directory.
You can also order the same update on CD-ROM. You will be directed to the on line shop. Notice that the shop uses a different account/username.

The ZIP file you download contains only one setup.exe. You need to run this executable.
It is also important that you put the license DLL into the same directory as setup.exe.
Setup will copy this file to the bascom application directory. You can also manual copy this file.
The license file is on CD-ROM, diskette, or the media (email) you received it on. It is only supplied once.
Without the file, bascom will not run.

The file is named bsc5132L.DLL for BASCOM-8051 and bscavrl.DLL for BASCOM-AVR.
When you got the license by email, it was zipped and probably had a different extension. Consult the original installation instructions.
The file is only provided once, we can not, and do not provide it again.

See Installing BASCOM for how to do a full install.
It is also possible to do a partial update. For example to update some DAT files, or to update to a beta which is only available as an update.
For partial updates, you need the Update Wiz.
When you do not have the Update Wiz, you can download it. Unzip it to the same directory as BASCOM.

The Update Wiz uses LIC files which you can download. A LIC file is a text file, it is not the LICENSE DLL!
Store the downloaded LIC file in the same directory as the Update Wiz.
When you store the Update Wiz into the same directory as BASCOM, the license DLL already exist there.
When you put the Update Wiz and the LIC files into a separate directory, you need to copy the BASCOM license DLL to this directory too.

When you run the Update Wiz, it will check for a new version and will download this if available. It will then run again.
When the Update Wiz finds a LIC file, it will check if the update/install location is specified. For new downloaded LIC files, the update wiz does not know the update directory, and will ask for the directory you want to update. This can be any (new) directory, but usually is the BASCOM application directory.

After you click Ok, the directory to update is stored in the LIC file. It will not be asked again.

Click the Next button to start the update. It depends on the downloaded LIC files how many products are found. You will get a similar window:
You need to select the product that you want to update. In the sample there is only 1 choice. Press the Next-button to continue.

The Wiz will compare files on the web with your local files in the specified directory.

When it finds packages that are newer, they will be shown in a list. By default they are all selected. You can unselect the packages you do not want to update. Press Next to download the selected packages.
During the download you will see the history file. When all packages are downloaded, they will be installed/unzipped. Press the Next-button to install the downloaded files.

During the installation you will see the progress. When installation is ready, you need to press the Finish-button.

The Wiz can also backup all files it will replace.
4 BASCOM IDE

4.1 RUNNING BASCOM-8051

When you run BASCOM-8051 the following window will appear.

The last saved/closed program will be loaded automatic. When reformatting is enabled, the loaded program will be reformatted too. This is only meaningful for programs written with another editor.

The BASCOM IDE is a so-called multi document application. This means that you can open more than one source file. The operations that you perform are always done on the current document, that is, the window with the focus.

The filename is shown in the caption of the window.

The status bar is separated in four panels.

- line, character position indicator
- modified indicator, to indicate that text has changed
- insert/overwrite indicator
- message panel

Some actions such as programming will make a progress indicator visible.
4.2 BASCOM IDE

Running BASCOM 8051

File
File New
File Open
File Close
File Save
File Save As...
File Print
File Print Preview
File Exit

Edit
Edit Undo
Edit Redo
Edit Copy
Edit Cut
Edit Paste
Edit Find
Edit Find Next
Edit Replace
Editor Keys

Program
Compile
Syntax check
Show Result
Simulate
Send to chip

Tools
Terminal Emulator
LCD designer
Graphic Converter
LIB Manager

Options
4.3 File New

Action
This option creates a new window in which you can write your program. The focus is set to the new window. Depending on the environment settings, the window is normal sized or maximized.

Note that you must save your program before you can compile it. Newly created files will have the name [nonameX] in the window caption. Where X is a number starting with 1 for the first editor window. Before you can compile your program, you must give it a valid name.

4.4 File Open

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

You can specify that BASCOM must reformat the file when it opens the file. See Options Environment options.
This should only be necessary when loading files made with another editor. Since saved/closed files are put in a so called 'recent file list', you can also open a file by selecting it from the File menu.

4.5 File Close

Action
Close current editor window. When changes are made, and they are not saved yet, you will be asked to save your program.

4.6 File Save

Action
With this option, you can quick save your current program to disk. If the program was created with the File New option, you will be asked for a filename first. Use the File Save As option to save the file with another name. Note that the file is saved as an ASCII file.

4.7 File Save As...

Action
With this option, you can save your current program to disk. You can enter a filename before your program is saved. Note that the file is saved as an ASCII file.

4.8 File Print Preview

Action
This will display the print preview window.
By clicking the Setup-button, you can change some printer properties. For margin settings, you must use the Options Printer settings. For a hardcopy, click the Print-button.

See also
Print

4.9 File Print

Action
With this option, you can print the current program.
Note that the current program is the editor window, which has the focus.

See also
Print preview

4.10 File Exit

Action
With this option, you can leave BASCOM.
If you have made changes to your program, you can save them upon leaving BASCOM.
4.11  Edit Undo

**Action**
With this option you can undo the last change you made to your program. By selecting this option again, you can undo the previous change to your program.

**See also**
*Edit Redo*

**Shortcut**
CTRL+Z

4.12  Edit Redo

**Action**
With this option you can redo the last undo action.

**See also**
*Edit Undo*

**Shortcut**
SHIFT+CTRL+Z

4.13  Edit Copy

**Action**
With this option, you can copy selected text into the clipboard. You can select text by dragging the mouse cursor over the text or by Double clicking on a word. Another possibility is to hold the shift key down and pressing the cursor keys. Selected text is shown inverted.

**Shortcut**
CTRL+C and CTRL+INS

4.14  Edit Cut

**Action**
With this option, you can cut selected text into the clipboard. The selected text is copied into the clipboard, and deleted from your program.

**Shortcut**
CTRL+T
4.15 **Edit Paste**

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

**Shortcut**
CTRL+ V and SHIFT + INS

4.16 **Edit Find**

**Action**
With this option, you can search for text in your program. The following dialog window will appear:

![Find Text Dialog](image)

You can choose to search forward or backward. Optional you can search case sensitive and for whole words. Regular expressions are also supported.

**Shortcut**
CTRL+F

4.17 **Edit Find Next**

**Action**
With this option you can search for the next occurrence of the specified text. When you didn't specify a search text, you will be asked for the text to find, with the windows find-dialog.

**See Also**
4.18 Edit Replace

**Action**
With this option, you can replace text in your program. The following replace dialog will appear:

Enter the text to search for and the text to replace with, and press return.

**Shortcut**
CTRL+R

4.19 Edit Goto

**Action**
With this option you can type the line number of the line you want to go to. The following screen will be shown:

The current line number will be shown. You can edit this and press RETURN to jump
to the line number of your choice.

4.20 **Edit Indent Block**

**Action**
Indents a block of selected text.

You need to select at least one line in order to use this option. When you have a structure like:

```plaintext
Do
a=a+1
b=b+1
Loop
```

It is hard to see the structure. You can best indent your code.

```plaintext
Do
  a=a+1
  b=b+1
Loop
```

When you have code that is not indented you can indent it by selecting the two lines within the structure and choose 'Edit Indent Block'.

4.21 **Edit Unindent Block**

**Action**
UnIndents a block of selected text.

You need to select at least one line in order to use this option. When you have a structure like:

```plaintext
Do
a=a+1
b=b+1
Loop
```

It is hard to see the structure. You can best indent your code.

```plaintext
Do
  a=a+1
  b=b+1
Loop
```

When you have code that is not indented you can indent it by selecting the two lines within the structure and choose 'Edit Indent Block'. The Unindent option can be used when the code is too much indented:

```plaintext
Do
  a=a+1
  b=b+1
Loop
```
The sample above show that too much indentation does not make the program readable.

4.22 Editor Keys

The following table lists all editor shortcuts.

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT ARROW</td>
<td>One character to the left</td>
</tr>
<tr>
<td>RIGHT ARROW</td>
<td>One character to the right</td>
</tr>
<tr>
<td>UP ARROW</td>
<td>One line up</td>
</tr>
<tr>
<td>DOWN ARROW</td>
<td>One line down</td>
</tr>
<tr>
<td>HOME</td>
<td>To the beginning of the line</td>
</tr>
<tr>
<td>END</td>
<td>To the end of the line</td>
</tr>
<tr>
<td>PAGE UP</td>
<td>Up one window</td>
</tr>
<tr>
<td>PAGE DOWN</td>
<td>Down one window</td>
</tr>
<tr>
<td>CTRL+LEFT</td>
<td>One word to the left</td>
</tr>
<tr>
<td>CTRL+RIGHT</td>
<td>One word to the right</td>
</tr>
<tr>
<td>CTRL+HOME</td>
<td>To the start of the text</td>
</tr>
<tr>
<td>CTRL+END</td>
<td>To the end of the text</td>
</tr>
<tr>
<td>CTRL+Y</td>
<td>Delete current line</td>
</tr>
<tr>
<td>INS</td>
<td>Toggles insert/overstrike mode</td>
</tr>
<tr>
<td>F1</td>
<td>Help (context sensitive)</td>
</tr>
<tr>
<td>F2</td>
<td>File Simulation</td>
</tr>
<tr>
<td>F3</td>
<td>Find next text</td>
</tr>
<tr>
<td>F4</td>
<td>Send program to chip or run programmer</td>
</tr>
<tr>
<td>F5</td>
<td>Run program (simulator)</td>
</tr>
<tr>
<td>F7</td>
<td>Compile File</td>
</tr>
<tr>
<td>CTRL+F7</td>
<td>Syntax check</td>
</tr>
<tr>
<td>F8</td>
<td>Step through program (simulator)</td>
</tr>
<tr>
<td>SHIFT + F8</td>
<td>Step over code (simulator)</td>
</tr>
<tr>
<td>F9</td>
<td>Toggle breakpoint (simulator)</td>
</tr>
<tr>
<td>F10</td>
<td>Run to cursor (simulator)</td>
</tr>
<tr>
<td>CTRL+J</td>
<td>Pop up code template</td>
</tr>
<tr>
<td>CTRL+G</td>
<td>Goto line</td>
</tr>
<tr>
<td>CTRL+O</td>
<td>Load File</td>
</tr>
<tr>
<td>CTRL+S</td>
<td>Save File</td>
</tr>
<tr>
<td>CTRL+P</td>
<td>Print File</td>
</tr>
<tr>
<td>CTRL+T</td>
<td>Terminal emulator</td>
</tr>
</tbody>
</table>
4.23 Program Compile

**Action**
With this option you can compile your current program. Your program will be saved automatically before it will be compiled.
So if you didn't give it a name, you will be asked for it.
The following files will be created depending on the Option Compiler Settings.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx.BIN</td>
<td>Binary file which can be burned into EPROM.</td>
</tr>
<tr>
<td>xxx.DBG</td>
<td>Debug file which is needed by the simulator.</td>
</tr>
<tr>
<td>xxx.HEX</td>
<td>Intel hexadecimal file.</td>
</tr>
<tr>
<td>xxx.ERR</td>
<td>Error file. (only when errors are found)</td>
</tr>
<tr>
<td>xxx.SIM</td>
<td>Generated by the simulator to store the variable names of the watch window and the breakpoints.</td>
</tr>
<tr>
<td>xxx.PRJ</td>
<td></td>
</tr>
</tbody>
</table>

If an error occurs, you will receive an error message and the compilation will end.
The cursor will be set to the line in which the error occurred. The line will be marked with a red color too. The red marking color will disappear when you compile the program again.

**Shortcut**
F7

4.24 Program Syntax check

**Action**
With this option you can check the syntax of your program. No files are generated with this option.
Shortcut
CTRL+F7

4.25 Program Show Result

Action
Use this option to view the result of the compilation.

![BASCOM 8051 Report window]

See the Options Compiler Output for specifying which files must be created. The files that can be viewed are report and error. Click the Print button to print the selected file. Click the Ok button to return to the editor.

Shortcut
CTRL+W

Information provided in the report:

<table>
<thead>
<tr>
<th>Info</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>Shows the version of the library (the compiler).</td>
</tr>
<tr>
<td>Processor</td>
<td>The type of microprocessor the file is compiled for.</td>
</tr>
<tr>
<td>Report</td>
<td>The name of the source file.</td>
</tr>
<tr>
<td>Date and time</td>
<td>The compilation date and time.</td>
</tr>
<tr>
<td>Comp.time</td>
<td>The start and end time needed for compilation.</td>
</tr>
<tr>
<td>Baud timer</td>
<td>The timer used for the generation of the baud rate.</td>
</tr>
<tr>
<td>Baud rate and frequency</td>
<td>The baud rate selected for the uP and the used crystal. This info is used for RS232 related statements such as PRINT and INPUT. Note that when you use the $crystal and $baud statements the exact baud rate is shown.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ROM start</strong></td>
<td>The starting location of ROM memory.</td>
</tr>
<tr>
<td><strong>RAM start</strong></td>
<td>The starting location of RAM memory.</td>
</tr>
<tr>
<td><strong>LCD mode</strong></td>
<td>4 bit or 8 bit LCD mode.</td>
</tr>
<tr>
<td><strong>Stack start</strong></td>
<td>The starting location of the stack. The space below the stack</td>
</tr>
<tr>
<td></td>
<td>is used for internal variables. The stack grows when calls are</td>
</tr>
<tr>
<td></td>
<td>made by the machine language routines.</td>
</tr>
<tr>
<td><strong>Used ROM</strong></td>
<td>Displays the length of the binary file.</td>
</tr>
<tr>
<td><strong>Variable</strong></td>
<td>The name, type and the location in memory of the used variables</td>
</tr>
</tbody>
</table>

### 4.26 Program Simulate

This option displays the Simulator window in which you can simulate a compiled program. When the source code is saved without compiling, you will be warned that the debug file differs from the source code. You have the option to compile it before you simulate or continue without recompiling.

The simulator window is divided in a few sections.
- Toolbar with speed buttons
- Variable watch/modify window
- Source code window
- Terminal (input/output) window
- Register window
- Status bar

**The margin**

On the left side a margin is visible. This margin can display the following icons:
- a yellow dot, indicating that the line holds executable code
a read dot, indicating that a break line is set. You can only set a breakpoint on a line that has a yellow dot.

a yellow arrow. This arrow shows the line currently executing.

The register window
On the right side the register window is visible. You can change the value of a register by entering a new value.

The variable watch section
The section below the toolbar is the variable watch section. You can add a variable by entering one in an empty cell. You can also add a new variable by selecting it from the source window, and pressing return. You can insert a new variable watch line by pressing the INS-key. You can delete a variable watch line by pressing the CTRL+DEL keys. You can change the value of a variable by setting the focus to the cell with the variables value and then by entering the new value. The variable names are saved and loaded after each simulation session.

The terminal section
The blue window emulates the serial port. So serial output (the PRINT statement for example), is displayed in this window. When serial input is required, you must set the focus to the serial window, before you enter text. The INPUT statement for example, requires serial input.

The source code window
The source code windows shows the source file being simulated. You can start a simulation by pressing F5 or by clicking the run button. When your program runs, you can pause it by clicking the pause button. You can stop the simulation by clicking the stop button. You can also step through the code line by line, by pressing F8, or by clicking the step button. By pressing SHIFT+F8, you can step over code, like GOSUB and CALL.

To pause execution at a certain line, you can set a breakpoint. Just set the cursor on that line and press F9. By pressing F9 again, you can remove the breakpoint. Note that a breakpoint can only be set on a line that contains executable code. This is visible by the yellow dot. Statements like $romstart don't contain executable code and won't have a corresponding yellow dot.

You can also run to a specified line by clicking the run to button.

The status bar
The status bar is also divided into a few sections. These sections from left to right display the following information:
- The value of a variable in the source code window. You can select a variable by moving the mouse cursor over the variable name.
- The status of the simulator (stopped, running or paused)
- The number of clock cycles and the execution time of the executed code. You can reset the value by clicking on this section.
- The stack depth of the program. The stack depth is the deepest level the stack has reached during execution. If it exceeds the available internal memory (128 or 256 bytes), the program will not run correctly in the chip.
The interrupt buttons

The INT0, INT1, T0, T1 and SER buttons can be clicked to generate an interrupt. Because this is a software emulator, no hardware interrupts can be generated. You have to do this yourself by clicking these buttons. TIMER 0 and TIMER 1 are simulated by software. Therefore, they will generate an interrupt automatically if the software enables this. The external gate however isn't simulated so for this occasion you must click the corresponding button. Depending on the chip used, other interrupt buttons can be visible. They have the same purpose as the default interrupt buttons.

Hardware simulator button

By clicking the hardware simulator button a special window will become visible. This window has a LCD simulator, which can simulate custom characters, LED simulation for port 0-3, and a 7-digit LED display simulation. The LCD type can be selected from the menu. Note that the display isn't as fast as it could be, but to assign/display all the dots costs a lot of processor time. The advantage however is that custom characters can be displayed too. The LED's can be switched on or off by clicking on it. The LED type can be set with the CG checkboxes. To select common ground you must set the marker. This will have the effect that all common cathodes are connected to ground and so the LED will be on when the port value will be high. The 7-digit display can be connected to individual port pins. To change the setting you must press the right mouse button to bring up the digit properties window.

Real hardware simulation

Press the real hardware simulation button to enable the hardware simulation. You need additional hardware to use this feature. You can use the MCS flash programmer to simulate one port. An application note can be downloaded that describes the needed hardware. The hardware simulator can simulate port 1 and 3. This way you can test your program in circuit without programming the device. Now only the status reading and setting of the ports is supported. This means that interrupts are not yet supported in hardware simulation.

Update source

The displaying of the variables and the arrow costs a lot of processor time. To simulate faster, you can disable the update of these items. Click the button to enable/disable the update.
**Display memory window**

To display the memory of the internal RAM, you can click the button. By clicking again, you can hide the window.

**Refresh variables**

Normally, variables are only refreshed in step mode (F8), because depending on the used statements, the value would be hard to watch. You can also choose to display the value during program execution. The default is on.

The sections can be made larger or smaller by using the splitters. When you press the right mouse button, a popup menu will be visible. Depending on the place the mouse cursor was at the time you pressed the right-mouse button, different options will be showed.

Extra options that will be come available are:
- Clear breakpoints
- Hide register window
- Hide watch window

To end a session close the windows or just set the focus to an editor window.

### 4.27 Program Send to chip

After you have tested your program you can run one of the supported programmers. You can also press F4 or click on the button.

Some programmers support the auto flash option from the programmers options. When you select this option, the programmer window will not be visible, but the chip will be erased, programmed and verified automatically. The progress will be visible in the IDE-menu bar.

Different serial comport and parallel printer port based programmers are supported. You must select one first with the Options Programmers menu.

- MCS Flashprogrammer
- Blow IT Flashprogrammer
- PG2051
- MCS SPI programmer
- PG302
- JPK Systems X-programmer
- Peter Averill’s TAFE programmer
- SE512 or SE514
- SE-812
- STK200/STK300 ISP programmer
- Sample Electronics simple cable ISP programmer
- RHOMBUS SCE-51 Emulator
- CYGNAL JTAG programmer
4.28 Tools Terminal Emulator

With this option you can start the built in terminal emulator. The following window will appear:

The terminal emulator supports ANSI, TTY, VT100 and VT220 terminal emulation. Information you type and information that the computer board sends, are displayed in the same window.

You must use the same baud rate for the terminal emulator and the program you compile. If you compiled your program with the Compiler Settings at 4800 baud, you must set the Communication Settings also to 4800 baud. The setting for the baud rate is reported in the report file.

The terminal menu has a few options.

**File upload**
This will upload the current program in HEX format to a monitor program. With the Options Monitor settings, you can specify an optional header to be sent before the actual hex file is sent to the monitor. Also a delay in mS can be specified for a optional delays after each line sent.

When an ALTAIR ROM is selected from the Monitor Options, a binary file will be sent to the monitor. The baud rate of the terminal emulator will be used. For an 552 ALTAIR ROM, the terminal baud rate must be set to 115200 baud.

While sending the hex file to the monitor, an extra menu option will be available:

**File Escape**
This will abort the upload to the monitor program.

**File Exit**
This will close the terminal emulator window.
4.29 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.

By clicking, the Clear All button you can clear all points. By clicking the Set All button you can set all points.

With the Options Compiler LCD settings you can choose if the 3 most significant bits must be set high. Some LCD displays require this.

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 number ranging from 0 to 7. When you want to display the custom character you can use the chr() function. LCD chr(0) 'will display custom character 0.

The numbers after the custom character are representing the row values. An empty row is converted to 32 (space) since a zero is used to terminate the bytes.

4.30 Tools Graphic Converter

The Graphic converter is intended to convert BMP files into BASCOM Graphic Files (BGF) that can be used with Graphic LCD displays.

The following dialog box will be shown:
To load a picture click the Load button.
The picture may be 64 pixels high and 240 pixels width.
When the picture is larger it will be adjusted.

You can use your favorite graphic tool to create the bitmaps and use the Graphic converter to convert them into black and white images.

When you click the Save-button the picture will be converted into black and white.
Any non-white color will be converted into black.

The resulting file will have the **BGF** extension.
Press the Ok-button to return to the editor.
The picture can be shown with the **ShowPic** statement.

**4.31 Tools LIB Manager**

With this option you can add and remove ASM routines to the libraries.
The following windows will be displayed:
Select a library first by clicking on it. The Routines list will be refreshed with the contents of the selected library.

By clicking the Add button a dialog box will be shown to select the ASM file that contains the ASM routine(s).

By clicking on the Delete button the selected Routine will be removed from the selected library.

A library is an ASCII file that contains ASM routines.

Each routine must be preceded by the name of the routine between brackets. Each routine must be ended with the [END] line.

A sample routine is shown here:

```
DEC76
; decrease the register pair r6 and r7 with one
; return zero in ACC when r6r7 is zero
_DEC76:
Dec r6                      ; dec LSB
Cjne r6,#255,*+4           ; if it was zero
Dec r7                      ; we need to decrease r7 to
Mov a,r7                    ; result into a
Orl a,r6                    ; OR with r6 to see if it is zero
Ret
[END]
```

The library can be included with the $LIB directive. A routine can be imported with the $EXTERNAL directive.

```
$lib "mylib.lib"
$external _dec76
```

4.32 Tools Triscent Converter

The Triscent Converter will convert a .H file generated by the Triscend program into a triscend.DAT file that can be used by BASCOM.

The triscend.DAT file has an additional section named XBYTE.

```
[XBYTE]
CMAP0_TAR = ff00
CMAP0_ALT = ff01
```

The 3 lines above show the section and 2 entries. The triscend chips are configured by writing to locations where normally XRAM is located. BASCOM handles this automatic for you. So when you assign a value to CMAP0_TAR, the value is written to location &HFF00 where the CMAP0_TAR register is located.

Reading this XRAM SFR will do the reverse.
At www.triscend.com you can find all info you need. Look for the E5 line of chips. These are 8051 compatible chips which can be configured with the Triscend software. You can for example create 3 UARTS, add I2C, SPI, TIMERS etc. So the E5 chip is hardware configurable by software!

After you created your ‘chip’, you create the .H file and this file must be imported with the Tools Triscend Convert option.

There is an evaluation KIT available from triscend. Another pro is that the chips have many pins. So when your design needs a lot of I/O pins, I advise to look at these chips.

4.33 Tools Export to RTF

Action
Exports the current file to an RTF file.

Remarks
RTF files can be used in documents such as Word files. RTF files can also be used to show code with colors on a web page. When your file has the name test.bas, a file with the name test.rtf will be created in the same directory.

4.34 Options Compiler Output

With this option you can specify which files must be created.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Communication</th>
<th>Environment</th>
<th>Hardware simulator</th>
<th>Programmer</th>
<th>Monitor</th>
<th>Pr</th>
</tr>
</thead>
</table>

Output
- Binary file
- Debug File
- HEX file
- Old intel HEX file
- Report file
- Error file
- NOI file

This will generate a ROM-image of the program. Of course you can
also store it in a flashrom.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug file</td>
<td>This option will generate a DBG-file. It is used by the simulator. When you don't use the simulator, you don't need to generate it.</td>
</tr>
<tr>
<td>Hex file</td>
<td>This is an Intel hex-file that is used by most programmers and monitor programs.</td>
</tr>
<tr>
<td>Old Intel hex file</td>
<td>This option will generate an old style Intel hex file and is used by the Elektor monitor. If you choose this option, you must unselect the Hex File option.</td>
</tr>
<tr>
<td>Report file</td>
<td>This file contains info about the program, such as the baudrate, used variables etc.</td>
</tr>
<tr>
<td>Error file</td>
<td>This file is generated when an error occurs. It holds the error descriptions. When there is no error, the file will not be created.</td>
</tr>
</tbody>
</table>

### 4.35 Options Compiler Communication

With this options you can select the used crystal and the baud rate that must be used with serial communications.

![Options Compiler Communication](image)

We advise to use the $BAUD$ and $CRYSTAL$ compiler directives in your program. This way the settings are stored in your source code.

### 4.36 Options Compiler I2C

With this option you can select the port pins that serve as the SDA and SCL line for the I2C statements.
You can also use the CONFIG SDA and CONFIG SCL statements.

### 4.37 Options Compiler LCD

With this option you can select the port pins for the LCD display. This only applies to the LCD statements when used in 4-bit mode and if the LCD display is connected to the port pins. You can also choose the port pins with the CONFIG LCDPIN statement.

In the 4-bit mode, only the highest nibble of the data lines is used. To spare a pin for the R/W pin, reading from the LCD is not supported and you must connect the R/W line to ground. See additional hardware for more info.

You can also use the LCD statements in the data bus mode.
Some LCD displays need the upper 3 bits to be set high. So when you have this kind of display you must select this option. When you select this option the LCD designer will set the upper 3 bits high when the `DEFLCDCHAR` statement is generated.

### 4.38 Options Compiler Misc

With the miscellaneous options you can change the following:

<table>
<thead>
<tr>
<th>Remarks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register file</td>
<td>Select the register file which is suitable for your target uP. The <code>reg51.DAT</code> file is the common file that works for every uP, but doesn’t have hardware specific registers. You can use this file as a base for your own DAT file.</td>
</tr>
<tr>
<td>byte end</td>
<td>Specifies the last location of internal memory that can be used by the compiler for storing variables. For uP’s with 128 bytes of RAM set it to 70 for example. All space after this value is used for the stack. With the simulator you can test if you run out of stack space. For uP’s with 256 bytes of internal RAM, you can use a higher value, <code>F0</code> for example.</td>
</tr>
<tr>
<td>size warning</td>
<td>Select this option to enable the compiler to give a warning message when the code size exceeds the specified size. (decimal)</td>
</tr>
</tbody>
</table>

### 4.39 Options Communication

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

The following window will appear:
Option | Remark
---|---
Comport | The comport of your computer to use.
Baud rate | The baud rate to use.
Parity | The parity to use.
Data bits | The number of data bits to use.
Stop bits | The number of stop bits to use.
Handshake | The handshake to use.
Emulation | The terminal emulation to use.
Font | Click the button to select the font and font color to use.
Backcolor | The background color to use (default blue)
Run emulator modal | Runs the terminal emulator in modal mode so you can use all key combinations that are normally reserved to the IDE.

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.
## 4.40 Options Environment

With this option you can modify the environment options.

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto indent</td>
<td>With auto indent, the cursor will be set to the same left margin as the current line when you press return.</td>
</tr>
<tr>
<td>Don't change case</td>
<td>This option will not change the case of your line when you enabled 'Reformat code'. By default each first characters case is set to uppercase.</td>
</tr>
<tr>
<td>Reformat BAS files</td>
<td>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 don't need to set this option.</td>
</tr>
<tr>
<td>Reformat code</td>
<td>Reformat code when entered in the editor. This will reformat the line after you have set focus to a new line.</td>
</tr>
<tr>
<td>Smart tabs</td>
<td>Will look at the previous line for non spaces to position the cursor.</td>
</tr>
<tr>
<td>Syntax highlight</td>
<td>Enables/disables syntax highlighting</td>
</tr>
<tr>
<td>Show margin</td>
<td>Shows a margin at position 80.</td>
</tr>
<tr>
<td>Comment position</td>
<td>The right position of the comment.</td>
</tr>
<tr>
<td>Tab size</td>
<td>The number of spaces equivalent to one tab.</td>
</tr>
<tr>
<td>Key mapping</td>
<td>Selects the behavior of the editor. Default behaves like Delphi.</td>
</tr>
<tr>
<td>No reformat extension</td>
<td>Specifies file extensions separated by a space where the reformatting is disabled. (for text files or dat files)</td>
</tr>
<tr>
<td>Size of new edit window</td>
<td>Selects the size of the edit window when a file is opened.</td>
</tr>
<tr>
<td>OPTION</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Background color</td>
<td>Background color of the editor</td>
</tr>
<tr>
<td>Keyword color</td>
<td>Color used to highlight keywords(statements)</td>
</tr>
<tr>
<td>Comment color</td>
<td>Color used to highlight comment</td>
</tr>
<tr>
<td>ASM color</td>
<td>Color used to highlight assembly</td>
</tr>
<tr>
<td>HW/ Register color</td>
<td>Color used to highlight special function registers</td>
</tr>
<tr>
<td>Editor font</td>
<td>Font name of the editor</td>
</tr>
<tr>
<td>Bold</td>
<td>Check to display keywords in bold</td>
</tr>
<tr>
<td>Italic</td>
<td>Check to display comment in Italic</td>
</tr>
</tbody>
</table>
### OPTIONS

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool tips</td>
<td>Will enable/disable tool tips.</td>
</tr>
<tr>
<td>Show Toolbar</td>
<td>Will display/hide the toolbar of the IDE.</td>
</tr>
<tr>
<td>Save File As... for new files</td>
<td>When you enable this option you will be prompted to give new files a name before they will be saved with their default name.</td>
</tr>
<tr>
<td>File location</td>
<td>The path to the location of your BAS files. Normally Windows will use My documents as a default.</td>
</tr>
</tbody>
</table>

#### 4.41 Options hardware simulator

This option let you select the address of the LPT connected to the optional hardware simulator.

#### 4.42 Options Programmer

This option let you select the target programmer.

The supported programmers are:

- MCS Flashprogrammer
- Blow IT programmer
- PG2051
- MCS SPI programmer
- PG302
- JPK Systems X-programmer
- Peter Averill’s TAFE programmer
- SE512 or SE514
- SE-812
- CYGNAL
- FutureLec
- SE511-SE516
The auto flash options will automatically program a chip without displaying the programmer window. The auto verify option will verify automatically after each programming. Selecting 'Code + Data' will program both the flash and the EEPROM.

You can select various programmers. On the Parallel-TAB you can select the LPT-address. You can also add or remove an LPT-address. It is only possible to remove address that you added yourself.

The port delay can best be set to 0. In some cases you might want to increase the value. Some programmers have I2C chips on them. For example the MCS Flash programmer. Since different I2C chips exist for the PCF8574, you need to select the checkbox when you use the PCF8574A.

### 4.43 Options Monitor

With the monitor options you can select the monitor you use.

There are only a few monitor programs supported.
- Altair 535/537
- Altair 552
- Monitor hex upload

The Altair monitor needs special instructions and uses binary files. The hex upload feature is meant for monitor programs that work with hex files.

You can upload a file to the target uP from the terminal emulator with the Upload file option.

For hex file based monitors there are 3 additional options:
o monitor prefix, is sent before the hex file
o monitor suffix, is sent after the hex file upload is completed

The prefix and suffix can contain returns or any ASCII character. Use {asc}, to imbed an ASCII character. asc=0-255.

For example @{13} for the prefix, will send @ followed by a return.
  o monitor delay, must be specified in msec’s, and is the delay time for each line sent.

4.44 Options Printer

These options let you select the printer margins.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Margin</td>
<td>The left printer margin in mm</td>
</tr>
<tr>
<td>Top Margin</td>
<td>The top printer margin in mm</td>
</tr>
<tr>
<td>Right Margin</td>
<td>The right printer margin in mm</td>
</tr>
<tr>
<td>Bottom Margin</td>
<td>The bottom printer margin in mm</td>
</tr>
<tr>
<td>Color</td>
<td>Check to print in color</td>
</tr>
<tr>
<td>Wrap Lines</td>
<td>Check when you want long lines to be wrapped. This is convenient when you</td>
</tr>
<tr>
<td></td>
<td>have long lines of source code that would otherwise would not fit on the</td>
</tr>
<tr>
<td></td>
<td>paper.</td>
</tr>
<tr>
<td>Print Header</td>
<td>Check to print a header with file name and page number</td>
</tr>
<tr>
<td>Line Numbers</td>
<td>Check to print line numbers</td>
</tr>
<tr>
<td>Syntax</td>
<td>Check to use syntax highlighting options and colors</td>
</tr>
</tbody>
</table>

4.45 Window cascade

Will cascade all editor windows so they will all be visible.
4.46 **Window Tile**
Window Tile will tile all editor windows.

4.47 **Window arrange icons**
Will arrange all iconized windows.

4.48 **Window minimize all**
Will minimize all editor windows.

4.49 **Help About**
This option shows an about box as displayed below.

![About Box](image)

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. You can compare it with the one from our web site in case you need an update. Click on the Ok-button to return to the editor.

4.50 **Help Index**
Will show the help index of BASCOM.

4.51 **Help on help**
Will bring up help about the Windows help system.
4.52 Help Shop

**Action**
This option will launch your default web browser and will open the MCS Electronics Shop.
We have a number of BASCOM-8051 KIT's and affordable 89Cx051 programmers from Sample Electronics

4.53 Help Forum

**Action**
This option will launch your default web browser and will open the MCS Forum. The forum can be used to talk to other BASCOM users. You can get ideas there, discuss your problems and questions, and you can help other members.

4.54 Help Support

**Action**
This option will launch your default web browser and will open the MCS Support system. The support system can be used to search the knowledge base.

4.55 Help Credits

Will launch this help file and show this topic.

MCS would like to thank the following people who have contributed to BASCOM development:

- Peter Averill from the Victoria University TAFE. Peter designed both the TAFE AT89C2051 programmer and the software to support it.
- Antti from Silicon Studio Ltd. Antti designed the BlowIT ATA89C2051 programmer and software to support it.
- Jakub Jiricek, he designed the SPI-programmer and software to support it.
- Francois du Plessis, he wrote a Windows version of Jacob's SPI-programmer software.
- Henry Arndt (DL2TM), he provided me with the source for his popular Atmel Programmer.
Part V
5 Language fundamentals

5.1 Language fundamentals

Characters from the BASCOM character set are put together to form labels, keywords, variables and operators. These in turn combine to form 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 can be used as parts of hexadecimal numbers.

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

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTER</td>
<td>Terminates input of a line</td>
</tr>
<tr>
<td></td>
<td>Blank ( or space)</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation mark (apostrophe)</td>
</tr>
<tr>
<td>*</td>
<td>Asterisks (multiplication symbol)</td>
</tr>
<tr>
<td>+</td>
<td>Plus sign</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>-</td>
<td>Minus sign</td>
</tr>
<tr>
<td>.</td>
<td>Period (decimal point)</td>
</tr>
</tbody>
</table>
| /         | 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) |

The BASCOM program line

BASCOM program lines have the following syntax:
```
[[line-identifier]] [[statement]] [[statement]] ... [[comment]]
```

© 2007 MCS Electronics
Using Line Identifiers
BASCOM supports one type of line-identifier; alphanumeric line labels:
An alphanumeric line label may be any combination of from 1 to 32 letters and
digits, starting with a letter and ending with a colon.
BASCOM 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.

BASCOM Statements
A BASCOM statement is either "executable" or "nonexecutable".
An executable statement advances the flow of a program's logic by telling the
program what to do next.
Non-executable statements 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 nonexecutable statement used to clarify a program's 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

BASCOM LineLength
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.

Data Types
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.

Elementary Data Types
  • Bit (1/8 byte)
  • Byte (1 byte)
    Bytes are stores 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 -2,147,483,648 to 2,147,483,647.

- **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) or external.

### 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, word, 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

### Variable Names

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**, for a complete list of reserved words).

You can specify a hexadecimal or binary number with the prefix **&H** or **&B**.

- a = &H, 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**

You can also use **DEFINT**, **DEFBIT**, **DEFBYTE** and/or **DEFWORD**.

*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 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 \ 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 \ 4 : remainder = 10 \MOD 4`
- Overflow and division by zero
  - Division by zero, produces an error.
  - At this moment there is no message, so you have to insure yourself that such 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.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Relation Tested</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equality</td>
<td><code>X = Y</code></td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Inequality</td>
<td><code>X &lt;&gt; Y</code></td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td><code>X &lt; Y</code></td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td><code>X &gt; Y</code></td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td><code>X &lt;= Y</code></td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td><code>X &gt;= Y</code></td>
</tr>
</tbody>
</table>
**Logical Operators**

Logical operators perform tests on relations, bit manipulations, or Boolean operators. There are four operators in BASCOM, they are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>Logical complement</td>
</tr>
<tr>
<td>AND</td>
<td>Conjunction</td>
</tr>
<tr>
<td>OR</td>
<td>Disjunction</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive or</td>
</tr>
</tbody>
</table>

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

**Floating point**

Single numbers conform 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:

\[
\begin{align*}
31 & 30 & 23 & 22 & 0 \\
\text{s} & \text{exponent} & \text{mantissa}
\end{align*}
\]

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 vise versa:

Dim I as Integer, S as Single
S = 100.1 'assign the single
I = S 'will convert the single to an integer
Take a look at the single.bas example for more information.

**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 or a word. This means that an array can hold 65535 elements as a maximum. The minimum value is 1 and not zero as in QB.
Arrays can be used on each place where a 'normal' variable is expected but there are a few exceptions. These exceptions are shown in the help topics. **Note that there are no BIT arrays in BASCOM-8051.**

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
    Print a(c) 'print it
Next

**Strings**
Strings can be up to 254 characters long in BASCOM. To save memory you must specify how long each string must be with the DIM statement.

**Dim S As String * 10**
This will reserve space for the string S with a length of 10 bytes. The actual length is 11 bytes because a null(0) is used to terminate the string.

You can concatenate string with the + sign.
Dim S As String * 10, Z As String * 10
S = "test"
Z = S + "abc" + var

In QB you can assign a string with a value and add the original string (or a part of it) too:
S = "test"
S = "a" + s

This will result in the string "atest"
In BASCOM-8051 this is NOT possible because this would require a copy of the string.
In BASCOM the string S is assigned with "a" and on that moment the original string S is destroyed. So you must make a copy of the string yourself in the event you need this functionality.
6 BASCOM Language Reference

6.1 BASCOM Statements

-1-

1WRESET, 1WREAD, 1WRITE
1WSEARCHFIRST, 1WSEARCHNEXT, 1WIRECOUNT

-COMPILER DIRECTIVES-

#IF
#ELSE
#ENDIF

$ASM - $END ASM
$INCLUDE
$BAUD
$BGF
$CRYSTAL
$DEFAULT XRAM
$IRAMSTART
$LARGE
$LCD
$MAP
$NOBREAK
$NOINIT
$NONAN
$NONULL
$NORAMCLEAR
$NOSP
$OBJ
$RAMSIZE
$RAMSTART
$REGFILE
$ROMSTART
$SERIALINPUT
$SERIALINPUT2LCD
$SERIALOUTPUT
$SIM

-A-

ABS
ALIAS
ASC
AVG

-B-

BITWAIT
BCD
BREAK

-C-

CALL
CLOSE
CLS
CHR
CONFIG
CONST
COUNTER
CPEEK
CURSOR

-D-
DATA
DEBOUNCE
DECR
DECLARE
DEFINT
DEFBIT
DEFBYTE
DEFLCDCHAR
DEFWORD
DELAY
DIM
DISABLE
DISPLAY
DO

-E-
ELSE
ENABLE
END
END_IF
ERASE
EXIT

-F-
FOR
FORTHLINE
FUSING

-G-
GET
GETAD
GETAD2051
GETRC
GETRCS
GOSUB
GOTO

-H-
HEX
HEXVAL
HIGH
HIGHW
HOME

-I-
I2CRECEIVE
I2CSEND
I2CSTART
I2CSTOP
I2CRBYTE
I2CWBYTE
IDLE
IF
INCR
INKEY
INP
INPUT
INPUTBIN
INPUTHEX
INSTR
LCASE
LCD
LCDINIT
LCDHEX
LEFT
LEN
LOAD
LOCATE
LOOKUP
LOOKUPSTR
LOOP
LOW
LOWW
LOWERLINE
MAKEDEC
MAKEBCD
MAKEINT
MAX
MID
MIN
MOD
NEXT
ON Interrupt
ON Value
OPEN
OUT
P1,P3
PEEK
POKE
PSET
POWERDOWN
PRINT
PRINTBIN
PRINTEX
PRIORITY
PUT
READ
READMAGCARD
REM
REPLACE
6.2  #IF

**Action**
Conditional compilation directive that tests for a condition.

**Syntax**

```
#IF test
[ #ELSE ]
```
#ENDIF

## Remarks

| test | An expression to test for. The expression may contain defined constants. |

Conditional compilation is used to include parts of your program. This is a convenient way to build different files depending on some constant values. Note that unlike the IF statement, the #IF directive does not expect a THEN. You may nest conditions to 25 levels. The use of #ELSE is optional.

### See Also

#ELSE, #ENDIF

### Example

```bascom
Const DEMO = 1 ' 0 = normal , 1= demo
#If Demo
    Print "Demo program"
#Else
    Print "Full version"
#Endif
```

Since the constant DEMO is assigned with the value 1, the compiler will compile only the line: Print "Demo program". Code between #else and #endif is not compiled! When you change the constant DEMO to 0, the other line will be compiled.

### 6.3 #ELSE

#### Action

Conditional compilation directive that tests for a NOT condition.

#### Syntax

```bascom
#IF test
#ELSE
#ENDIF
```

#### Remarks

| test | An expression to test for. The expression may contain defined constants. |

Conditional compilation is used to include parts of your program. This is a convenient way to build different files depending on some constant values. Note that unlike the IF statement, the #IF directive does not expect a THEN. You may nest conditions to 25 levels. The use of #ELSE is optional. The code between #ELSE and #ENDIF will be compiled when the expression is not true.
See Also

#IF , #ENDIF

Example

CONST DEMO = 1 ' 0 = normal , 1= demo
#IF Demo
   Print "Demo program"
#ELSE
   Print "Full version"
#ENDIF

Since the constant DEMO is assigned with the value 1, the compiler will compile only
the line : Print "Demo program" . Code between #else and #endif is not compiled!
When you change the constant DEMO to 0, the other line will be compiled.

6.4 #ENDIF

Action

Conditional compilation directive that ends a test.

Syntax

#IF test
[ #ELSE ]
#ENDIF

Remarks

<table>
<thead>
<tr>
<th>Test</th>
<th>An expression to test for. The expression may contain defined constants.</th>
</tr>
</thead>
</table>

Conditional compilation is used to include parts of your program. This is a
convenient way to build different files depending on some constant values.
Note that unlike the IF statement, the #IF directive does not expect a THEN.
You may nest conditions to 25 levels.
The use of #ELSE is optional.
Note that #ENDIF must be written as #ENDIF, not as #END IF

See Also

#IF , #ELSE

Example

CONST DEMO = 1 ' 0 = normal , 1= demo
#IF Demo
   Print "Demo program"
#ELSE
   Print "Full version"
#ENDIF

Since the constant DEMO is assigned with the value 1, the compiler will compile only
the line : Print "Demo program" . Code between #else and #endif is not compiled!

When you change the constant DEMO to 0, the other line will be compiled.
6.5 1WIRE

Action
These routines can be used to communicate with Dallas Semiconductors 1Wire-devices.

Syntax 1 for use with the CONFIG 1WIRE statement

1WRESET
1WWRITE var1 [, bytes]
var2 = 1WREAD([ bytes])

Syntax 2 for use with multiple devices/pins

1WRESET pin
1WWRITE var1 [, bytes] pin
var2 = 1WREAD([ bytes] [, pin])
var2 = 1WREAD([ pin])

Pin is the port pin to use with the device such as P1.1

Remarks

<table>
<thead>
<tr>
<th>1WRESET</th>
<th>Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1WWRITE var1</td>
<td>Sends the value of var1 to the bus. Optional is the number of bytes that must be sent. var1 is a numeric variable or constant.</td>
</tr>
<tr>
<td>var2 = 1WREAD()</td>
<td>Reads a byte from the bus and places it into var2. Optional is the number of bytes that must be read. var2 is a number variable.</td>
</tr>
</tbody>
</table>

Example

`'--------------------------------------------------------------'
1WIRE.BAS
'demonstrates 1wreset, 1wwrite and 1wread()'
'pull-up of 4K7 required to VCC from P.1'
'DS2401 serial button connected to P1.1'

Config 1wire = P1.1                      'use this pin
Dim Ar(8) As Byte, A As Byte, I As Byte

1wreset                                'reset the device
Print Err                               'print error 1 if error
1wwrite &H33                            'read ROM command
For I = 1 To 8                          'place into array
  Ar(i) = 1wread()
Next
For I = 1 To 8                          'print output
  Printhex Ar(i);
Next
Print
'linefeed

'You can also use multiple pins`
'alias the pin first
Tsensor Alias P1.2

'the optional argument specifies the pin to use
lwreset Tsensor          'reset
lwwrite &H33 Tsensor     'write
value to Tsensor
lwwrite Ar(1) , 2 Tsensor bytes to Tsensor
A = lwread(tsensor)      'return
byte from Tsensor
Ar(1) = lwread(2 , P1.2) 'read 2
bytes from Tsensor
End

6.6 1WIRECOUNT

Action
This statement returns the number of 1wire devices found on the bus.

Syntax
var2 = 1WIRECOUNT(array )

Remarks
| var2 | A word variable that is assigned with the number if found 1wire devices on the bus. |
| Array | A variable or array that should be at least 8 bytes long. It is used to store the 1wire ID’s while counting. |

The 1wireCount function uses the 1wSearchFirst() and 1wSearchNexy functions internally.

See also
1WIRE 1WSEARCHFIRST 1WSEARCHNEXT

Example
-------------------------------------------------------------------------------------
--------
'       1wirecount.bas
'       (c)1995-2006 MCS Electronics
'       demonstration of using multiple devices
'-------------------------------------------------------------------------------------
--------
'chip we use
$regfile = "89s8252.dat"
'crystal attached
$crystal = 12000000

'baud rate
$baud = 4800

'wait for 500 mili secs
Waitms 500

'the pins we use
'connect a 4K7 resistor from the data pin to VCC

Config 1wire = P1.0

'we need an array of 8 bytes to hold the result
Dim Ar(8) As Byte
'we also need a counter variable and a word variable
Dim I As Byte, W As Word

'some ids of 1wire chips I tested
' 01 51 B5 8D 01 00 00 56
' 01 84 B3 8D 01 00 00 E5

Print "start"
'get the number of connected 1wire device
W = 1wirecount(ar(1))
'print if there was an error and how many sensors are available
Print "ERR " ; Err ; " count " ; W

'now get the data from the first 1wire device on the bus
Ar(1) = 1wsearchfirst()
'print the ID
For I = 1 To 8
    PrintHex Ar(i);
Next
Print

'I assume that there are more than 1 1wire devices
Do
    'get the next device
    Ar(1) = 1wsearchnext()
    For I = 1 To 8
        PrintHex Ar(i);
    Next
    Print
Loop Until Err = 1
'when ERR is 1 it means there are no more devices
' IMPORTANT : 1wsearchfirst and next functions do require that you use
' the SAME array
'In this example this is ar(1)

'once you know the ID, you can address a specific device
End

6.7 1WSEARCHFIRST

Action
This statement reads the first ID from the 1wire bus into a variable array.

Syntax
var2 = 1WSEARCHFIRST()

Remarks

| var2 | A variable or array that should be at least 8 bytes long and that will be assigned with the 8 byte ID from the first 1wire device on the bus. |

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive
function calls to the 1wireSearchNext function to retrieve other ID's on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator.
We advice to use a byte array as shown in the example.

The ERR bit is set when there are no 1wire devices found.

See also
1WIRE, 1WIRECOUNT, 1WSEARCHNEXT

Example

```
'----------------------------------------------------------------------
' 1wirecount.bas
(c) 1995-2006  MCS Electronics

demonstration of using multiple devices

'chip we use
$regfile  = "89s8252.dat"
'crystal attached
$crystal  = 12000000

'baud rate
$baud = 4800

'wait for 500 mili secs
Waitms 500

'the pins we use
'connect a 4K7 resistor from the data pin to VCC
Config 1wire = P1.0

'we need an array of 8 bytes to hold the result
Dim Ar(8) As Byte
'we also need a counter variable and a word variable
Dim I As Byte, W As Word

'some ids of 1wire chips I tested
' 01 51 B5 8D 01 00 00 56
' 01 84 B3 8D 01 00 00 E5

Print "start"
'get the number of connected 1wire device
W = 1wirecount(ar(1))
'print if there was an error and how many sensors are available
Print "ERR " ; Err ; " count " ; W

'now get the data from the first 1wire device on the bus
Ar(1) = 1wsearchfirst()
'print the ID
For I = 1 To 8
  Printhex  Ar(i);
Next
Print

'I assume that there are more than 1 1wire devices
Do
  'get the next device
  Ar(1) = 1wsearchnext()
  For I = 1 To 8
    Printhex  Ar(i);
  Next

```

© 2007 MCS Electronics
Next Print Loop Until Err = 1
' when ERR is 1 it means there are no more devices
' IMPORTANT : 1wsearchfirst and next functions do require that you use
' the SAME array
' In this example this is ar(1)
' once you know the ID, you can address a specific device
End

6.8 1WSEARCHNEXT

Action
This statement reads the next ID from the 1wire bus into a variable array.

Syntax
var2 = 1WSEARCHNEXT( )

Remarks
var2 A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the next 1wire device on the bus.

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive function calls to the 1wireSearchNext function to retrieve other ID’s on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator. I would advice to use a byte array as shown in the example.

The ERR variable is set when there are no more devices found.

See also
1WIRE, 1WSEARCHFIRST, 1WIRECOUNT

Example
'-------------------------------------------------------------------------------
' 1wirecount.bas
'(c) 1995-2006 MCS Electronics
' demonstration of using multiple devices
'-------------------------------------------------------------------------------
'chip we use
$regfile = "89s8252.dat"
'crystal attached
$crystal = 12000000
'baud rate
$baud = 4800

© 2007 MCS Electronics
'wait for 500 mili secs
Waitms 500

'the pins we use
'connect a 4K7 resistor from the data pin to VCC
Config 1wire = P1.0

'we need an array of 8 bytes to hold the result
Dim Ar(8) As Byte
'we also need a counter variable and a word variable
Dim I As Byte, W As Word

'some ids of 1wire chips I tested
' 01 51 B5 8D 01 00 00 56
' 01 84 B3 8D 01 00 00 E5

Print "start"
'get the number of connected 1wire device
W = 1wirecount(ar(1))
'print if there was an error and how many sensors are available
Print "ERR " ; Err ; " count " ; W

'now get the data from the first 1wire device on the bus
Ar(1) = 1wsearchfirst()
'print the ID
For I = 1 To 8
    Printhex Ar(i);
Next
Print

'I assume that there are more than 1 1wire devices
Do
'get the next device
Ar(1) = 1wsearchnext()
For I = 1 To 8
    Printhex Ar(i);
Next
Print
Loop Until Err = 1
'when ERR is 1 it means there are no more devices
' IMPORTANT : 1wsearchfirst and next functions do require that you use
the SAME array
'In this example this is ar(1)

'once you know the ID, you can address a specific device
End

6.9 $ASM - $END 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.
You can also insert ASM code by preceding the line with the ! sign.
See also
ASM programming

Example
Dim c as Byte
$ASM
  Mov r0,#{C} ;address of c
  Mov a,#1
  Mov @r0,a ;store 1 into var c
$END ASM
Print c
End

6.10 $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 must be a numeric constant. |

When you want to use a crystal/baud rate that can’t be selected from the options,
you can use this compiler directive.
You must also use the $CRYSTAL directive.
These statements always work together.

In the generated report you can view which baud rate is actually generated.
But the baud rate is only shown when RS-232 statements are used like PRINT,
INPUT etc.

See also
$CRYSTAL

Example
$baud = 2400
$crystal = 14000000 ' 14 MHz crystal
Print "Hello"
End

6.11 $BGF

Action
Binds a BASCOM Graphic File into the program for use with Graphic LCD displays.
Syntax

$BGF "file"

Remarks

"file" is the name of the BGF file that is included in the program. BMP files can be converted with the Tools Graphic Converter.

See also

SHOWPIC

Example

'------------------------------------------------------------------------------
  (c) 1995-2006 MCS Electronics
  GLCD.BAS
  Sample to show support for T6963C based graphic display
  Only 240*64 display is supported with 30 columns(yet)
  At the moment the display can only be used in PORT mode
  Connection :
  P1.0 - P1.7  to DB0-DB7 of LCD
  P3.2        to FS, font select of LCD can be hard wired too
  P3.5        to CE, chip enable of LCD
  P3.4        to CD, code/data select of LCD
  P3.6        to WR of LCD
  P3.7        to RD of LCD
  A future version will allow external data access too which also uses
  RD and WR
  The display from www.conrad.com needs a negative voltage for the
  contrast.
  I used two 9 V batteries

  configure the LCD display
Config  Graphlcd = 240 * 64 , Port = P1 , Ce = P3.5 , Cd = P3.4 , Cols = 30

  dimension some variables used by the DEMO
Dim  X  As  Byte , Y  As  Byte

  Reset P3.2                                            '8 bit
  wide char is 30 columns

  The following statements are supported:
Cls  'will clear graphic and text
'cls TEXT will clear only the text
'cls GRAPH will clear only the graphic part

  To init the display manual you can use:
'Lcdinit
  'But this should not be needed as it is initialised at start up.

  Locate is supported and you can use 1-8 for the row and 1-30 for the column
Locate 1 , 1
'cursor control is the same as for normal LCD

**Cursor On Blink**

'And to show some text you can use LCD

```bascom
Lcd "Hello world"
```

'Note that the cursor position is not adjusted. You can set it with locate

'Now comes the fun part for using a graphic LCD

'We can display a BMP file. You may use MSPAINT or any other tool that can create a BMP file. With the Graphic converter from the Tools Menu you can convert the file 'into a BGF file. (BASCOM GRAPHICS FILE). The conversion will convert all non white pixels to BLACK.

'To display the BGF file you use the SHOWPIC statement that needs an X and Y parameter

'the third param is the label where the data is stored.

'The position must be divideble by 8 because this is the way the display handles the data

```
Showpic 0 , 0 , Picture1
```

'And we use the PSET known from QB to set or reset a single pixel

'A value of 0 means clear the pixel and 1 means set the pixel

'create a block

```bascom
For X = 0 To 10
  For Y = 0 To 10
    Pset X , Y , 1
  Next
Next
```

'You could remove it too

```bascom
For X = 0 To 10
  For Y = 0 To 10 Step 2
    Pset X , Y , 0
  Next
Next
```

'A simple scope or data logger could be made with PSET !

'We hope to get an AN from an inspired user :-)

```bascom
End
```

'label for the picture

```bascom
Picture1:
```

'$$BGF includes the data from the specified file

```bascom
$bgf "samples\mcs.bgf"
```

### 6.12 **$CRYSTAL**

**Action**

Instruct the compiler to override the crystal frequency options setting.

**Syntax**

```bascom
$CRYSTAL = var
```
Remarks

\[
\text{var} \quad \text{Frequency of the crystal.}
\]
\[
\text{var} : \text{Constant.}
\]

When you want to use an unsupported crystal/baud rate you can use this compiler directive.
When you do, you must also use the corresponding $\text{BAUD}$ directive.
These statements always work together.

See also

$\text{BAUD}$

Example

\[
\begin{align*}
\text{$\text{baud}$} & = 2400 \\
\text{$\text{crystal}$} & = 14000000 & \text{'14 MHz crystal} \\
\text{Print} & \text{ "Hello"} \\
\text{End}
\end{align*}
\]

6.13 $\text{DEFAULT XRAM}$

Action

Compiler directive to handle each dimensioned variable as XRAM variable.

Syntax

$\text{DEFAULT XRAM} | \text{IRAM}$

Remarks

When you are using many XRAM variables it make sense to set this option, so you don't have to type XRAM each time.
To dimension a variable to be stored into IRAM, specify IRAM in that case.

See Also

$\text{DIM}$

Example

\[
\begin{align*}
\text{$\text{default Xram}$} \\
\text{Dim} & \text{ X As Integer} & \text{'will go to XRAM} \\
\text{Dim} & \text{ Z As Iram Integer} & \text{'will be stored in IRAM}
\end{align*}
\]

6.14 $\text{EXTERNAL}$

Action

Compiler directive that instructs the compiler to include the specified assembler routines.
Syntax

$EXTERNAL myrout [, other]

Remarks

The $EXTERNAL directive is used internally by the compiler in order to enable the customizing of the assembler routines by the user. You can use it to include your own assembler routines. At the moment using $EXTERNAL will always include the routine no matter if it is used or not.

See also

$LIB, LIB Manager

Example

$LIB "mylib.lib"
$EXTERNAL _dec76

6.15 $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) 1995-2006 MCS Electronics'
' file: INCLUDE.BAS'
' demo: $INCLUDE'
'--------------------------------------------------------------'
Print "INCLUDE.BAS"
$include "123.bas" 'include file that prints Hello
Print "Back in INCLUDE.BAS"
End
6.16 $IRAMSTART

**Action**
Compiler directive to specify starting internal memory location.

**Syntax**

```
$IRAMSTART = constant
```

**Remarks**

| Constant | A constant with the starting value (0-255) |

**See also**

$NOINIT, $RAMSTART

**Example**

```
$NOINIT
$NOSP
$IRAMSTART = &H60                'first usable memory location
SP = 80
DIM I As Integer
```

6.17 $LARGE

**Action**
Instructs the compiler that LCALL statements must be used.

**Syntax**

```
$LARGE
```

**Remarks**

Internally when a subroutine is called the ACALL statement is used. The ACALL instruction needs only 2 bytes (the LCALL needs 3 bytes). The ACALL statement however can only address routines with a maximal offset of 2048 within the page. AT89C2051 chips will have no problems with that.

When code is generated for another uP, the subroutine being called can be further away and you will receive an error. With the $LARGE statement you instruct the compiler to use the LCALL statement which can address the full 64K address space.

**Example**

```
$LARGE             'I received an error 148 so I need this option
```
6.18 $LIB

**Action**
Compiler directive that instructs the compiler to look for assembler routines in the specified LIB file.

**Syntax**
$LIB "myrout.LIB"

**Remarks**
The $LIB directive is used internally by the compiler in order to enable the customizing of the assembler routines by the user. You can use it to specify your own libraries. You can for example copy the mcs.lib file to a new file named mylib.lib and delete the content of the mcs.lib file. This way the compiler will use your routines. The mcs.lib file must exist in the \LIB subdirectory and that is why you may not delete it. Always make a backup of the mcs.lib file before you change it. It is not encouraged to change the mcs.lib file itself other than making a dummy because updates will contain more asm routines and you have to change everything for each update.

**See also**
$EXTERNAL

**Example**
$LIB "mylib.lib"
$EXTERNAL _dec76

6.19 $LCD

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

**Syntax**
$LCD = [&H]address

**Remarks**
The address where must be written to, to enable the LCD display. The db0-db7 lines of the LCD must be connected to the datelines D0-D7. The RS line of the LCD must be connected to the address line A0.

On systems with external RAM/ROM it makes more sense to attach the LCD to the data bus. With an address decoder you can select the LCD display.
6.20 $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 and the RS of the LCD. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. The RS line of the LCD must be connected to the address line A0 by default. When it is connected to another address line you can specify $LCDRS On systems with external RAM/ROM 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

Example
$lcd = &HA000  'writing to this address will make the E line of the LCD high.
Cls
Lcd "Hello world"
End

$lcdrs = &H8002  'writing to this address will make the RS line of the LCD high.
Cls
Lcd "Elektor"
End
6.21 \$MAP

**Action**
Generates info in the report file with hexadecimal address of each source line.

**Syntax**
\$MAP

**Remarks**
For debugging it can be useful to know at which address a source line begins.

**See also**
NONE

**Example**
\$MAP
Print "Hello"
Print "Test"

Will generate the following section in the report file:

<table>
<thead>
<tr>
<th>Code map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

6.22 \$NOBREAK

**Action**
Instruct the compiler that BREAK statements must not be compiled.

**Syntax**
\$NOBREAK

**Remarks**
With the BREAK statement, you can generate a reserved opcode that is used by the simulator to pause the simulation.
When you want to compile without these opcode's you don't have to remove the BREAK statement: you can use the $NOBREAK statement to achieve the same.

**See also**
BREAK
Example
$nobreak
Break
' this isn't compiled into code so the simulator will not pause
End

6.23 $NOINIT

Action
Instruct the compiler that no initialization must be performed.

Syntax
$NOINIT

Remarks
BASCOM initializes the processor depending on the used statements. When you want to handle this by yourself you can specify this with the compiler directive $NOINIT. The only initialization that is always done is the setting of the stack pointer and the initialization of the LCD display (if LCD statements are used).

When you have selected the Altair as a monitor in the Monitor options, the following code will be generated:
Mov IE,#255
Mov scon,#82
This because the Altair monitor needs this code despite of the $NOINIT. When you do not want that, you have to select HEX Monitor for example.

See also
$NOSP, $NORAMCLEAR

Example
$NOINIT
$NORAMCLEAR
' your program goes here
End

6.24 $NONAN

Action
Compiler directive for changing NAN (not a number) into 0.0

Syntax
$NONAN
**Remarks**
A single can return a NAN when it is not considered to be a number. With the $NONAN directive 0.0 will be returned.

**See also**
NONE

**Example**
NONE

---

**6.25 $NONULL**

**Action**
Compiler directive for changing the behavior of the DATA statements.

**Syntax**

```
$NONULL = value
```

**Remarks**

| value | 0 for default behavior. And -1 for special behavior |

When a string is stored with a DATA statement, a null is added to indicate the string end. In some situations you might not want this. When you write a custom routine to work with a long string for example. With $NONULL = -1, the additional null byte is not added. To switch back to normal mode use a value of 0.

**See also**
NONE

**Example**

```
$nnonull = -1
Lbl:
 Data "test" , "this"
Lbl2: $nonull = 0                     'normal mode
 Data "test" , "this"
```

---

**6.26 $NORAMCLEAR**

**Action**
Instruct the compiler that the internal RAM should not be cleared at start up.

**Syntax**

```
$NORAMCLEAR
```
Remarks
BASCOM clears the internal memory after a reset. When you don’t want this behavior you can use the $NORAMCLEAR compiler directive.

See also
NONE

Example
$NORAMCLEAR
‘your code goes here
End

6.27 $NOSP

Action
Instruct the compiler that the stack pointer must not be set.

Syntax
$NOSP

Remarks
BASCOM initializes the processor depending on the used statements. When you want to handle this by yourself you can specify this with the compiler directive $NOINIT. The only initialization that is always done is the setting of the stack pointer and the initialization of the LCD display (if LCD statements are used). With the $NOSP directive the stack will not be initialized either.

See also
$NOINIT

Example
$NOSP $NOINIT
End

6.28 $OBJ

Action
Includes Intel object code.

Syntax
$OBJ obj

Remarks
obj is the object code to include.
In some cases it can be useful to include object code. This object code can be
generated with other tools.

Example
$OBJ D291  ‘this is equivalent to SET P1.1

6.29 $RAMSIZE

Action
Specifies the size of the external RAM memory.

Syntax
$RAMSIZE = [&H] size

Remarks
<table>
<thead>
<tr>
<th>Size</th>
<th>Size of external RAM memory chip.</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Constant.</td>
</tr>
</tbody>
</table>

See also
$RAMSTART

Example
$ROMSTART = &H4000
$RAMSTART = 0
$RAMSIZE = &H1000
DIM x AS XRAM Byte  ‘specify XRAM to store variable in XRAM

6.30 $RAMTRON

Action
Tell the compiler to use SPI memory as XRAM.

Syntax
$RAMTRON

Remarks
<table>
<thead>
<tr>
<th>address</th>
<th>The (hex)-address where the data is stored. Or the lowest address which enables the RAM chip.</th>
</tr>
</thead>
</table>
You can use this option when you want to run your code in systems with external RAM memory.

Ramtron (www.ramtron.com) sell EEPROM's that are as fast as normal RAM chips. They can be written billions of times. The $ramtron directive will use such as ramtron device as xram device. This only works for the AT89S8252. You only add a ramtron EEPROM to the hardware SPI lines and when you dim a variable as XRAM, the EEPROM will be used to store and retrieve the data.

This is a convenient way to add more memory without adding an address decoder and a RAM chip. Since the EEPROM is housed in a 8 pins chip it will make your design simple.

Note however that it is best practice that writing to such a XRAM variable must not be excessive. The data sheet of the Ramtron chips show that you can write it many times and in effect it will take years until you reach the limit.

Note that $RAMTRON does not need a parameter.

**ASM**

When XRAM is written with Movx @dptr,a , a call will be made to _WriteRamtron. Nothing is destroyed or returned.

When XRAM is read with Movx a,@dptr , a call will be made to _ReadRamtron. Value is returned in ACC as movx a,@dptr would do too.

Both routines are in the mcs.lib file. Both routines call _Wait_Spif to wait for the SPI, SPIF bit.

**Example**

```
'-------------------------------------------------------------------
'    (c) 1995-2006 MCS Electronics
'    RAMTRON.BAS
' This example shos how to use the www.ramtron.com eeprom
' to be used a XRAM
'-------------------------------------------------------------------
'it works only for the 8252
$regfile = "89s8252.dat"

'tell the compiler about ramtron
'THIS SAMPLE WILL NOT SIMULATE beause of the $RAMTON directive
'Suggestion is to add the directive when you simulated your program
$ramtron

'dim some variables
Dim X As Byte , X1 As Byte

'Now dim XRAM. This will be stored in the Ramtron devic
Dim Z(10) As Xram Byte
```

© 2007 MCS Electronics
Wait 1

'I used P1.3 for the CS so the mcs.lib also uses this pin
'I P1.4 could be used too but it needs a change in the mcs.lib
'This sample works actually!
'But since I also have code like *+4 it will not work always
'I need to rewrite that code. Let me know when some routines dont work
'with the $ramtron directive

'fill the data
For X = 1 To 10
  Z(x) = X
Next

'print the data
For X = 1 To 10
  Print Z(x)
Next
End

6.31 $RAMSTART

**Action**
Specifies the location of the external RAM memory.

**Syntax**

```plaintext
$RAMSTART = [&H]address
```

**Remarks**

<table>
<thead>
<tr>
<th>address</th>
<th>The (hex)-address where the data is stored. Or the lowest address which enables the RAM chip. You can use this option when you want to run your code in systems with external RAM memory. Address must be a numeric constant.</th>
</tr>
</thead>
</table>

**See also**

$RAMSIZE

**Example**

```plaintext
$ROMSTART = &H4000
$RAMSTART = 0
$RAMSIZE = &H1000
```
6.32  $REGFILE

**Action**
Instructs the compiler to use the specified register file.

**Syntax**

\[
$REGFILE = "file"
\]

**Remarks**

<table>
<thead>
<tr>
<th>File</th>
<th>The name of the register file to use.</th>
</tr>
</thead>
</table>

The $REGFILE statement must be placed before any other executable statements or compiler directives.

**See also**

NONE

**Example**

'comment is no problem before the $REGFILE statement

$REGFILE = "8052.DAT"  

'use the 8052.DAT file

6.33  $ROMSTART

**Action**
Specifies the location of the ROM memory.

**Syntax**

\[
$ROMSTART = [\&H] address
\]

**Remarks**

<table>
<thead>
<tr>
<th>address</th>
<th>The (hex)-address where the code must start. Default is 0. This value will be used when $ROMSTART is not specified. You can use this option when you want to test the code in RAM. The code must be uploaded and placed into the specified address and can be called from a monitor program. The monitor program must relocate the interrupts to the correct address! When $ROMSTART = &amp;H4000 is specified the monitor program must perform a LJMP instruction. For address 3 this must be &amp;H4003. Otherwise interrupts can not be handled correctly. But that is up to the monitor program.</th>
</tr>
</thead>
</table>
### 6.34 $SERIALINPUT

**Action**
Specifies that serial input must be redirected.

**Syntax**

```
$SERIALINPUT = label
```

**Remarks**

<table>
<thead>
<tr>
<th>Label</th>
<th>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 ACC.</th>
</tr>
</thead>
</table>

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.

**See also**

$SERIALOUTPUT

**Example**

```
$SERIALINPUT = Myinput
'here goes your program
END
!myinput:
;perform the needed actions here
   mov a, sbuf ;serial input buffer to acc
ret
```

### 6.35 $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 and $SERIALOUTPUT statements, but the $SERIALINPUT2LCD is handy when you use a LCD display.

**See also**

- $SERIALINPUT
- $SERIALOUTPUT

**Example**

```plaintext
$serialinput2lcd
Dim V As Byte
Cls
Input "Number ", V
' this will go to the LCD display
```

### 6.36 $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 sent to the serial buffer (SBUF). The character is placed into ACC. |

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.

**See Also**

- $SERIALINPUT

**Example**

```
$SERIALOUTPUT = MyOutput
' here goes your program
END
!myoutput:
;perform the needed actions here
  mov sbuf, a ;serial output buffer (default)
ret
```
6.37 **$SIM**

**Action**
Generates code without the actual waiting loops in order to speed up the simulator.

**Syntax**

```
$SIM
```

**Remarks**
When simulating the WAIT statement, you will experience that it takes a long time to execute. You can also switch off the updating of variables/source which costs time, but an alternative is the $SIM directive.

You must remove the $SIM statement when you want to place your program into a chip/EPROM.

**See also**

BREAK

**Example**

```
$SIM   'don't make code for WAIT and WAITMS
WAIT 2  'the simulator is faster now
```

6.38 **$TIMEOUT**

**Action**
Compiler directive to specify that the TIMEOUT option is used with serial input.

**Syntax**

```
$TIMEOUT
```

**Remarks**

$TIMEOUT will modify the serial input routine so that it enables you to use the TIMEOUT with the INPUT, INPUTBIN, INPUTHEX etc. statements.

**See also**

INPUT, GET

**Example**

```
$TIMEOUT
DIM Name as string * 10
REM Now we can use the TIMEOUT option
INPUT "Name ", name TIMEOUT = 100000 'enable time out
INPUT "Name ", name 'wait until <13> pressed.
```
6.39  $WAIT

**Action**
Will insert a one second delay in the startup code.

**Syntax**
$WAIT

**Remarks**
When using the AT89C8252 ISP facility it is needed that the chip waits 1 second after reset. Otherwise it can occur that the chip can not be programmed serial anymore.
Do not confuse $WAIT with the WAIT statement.
$WAIT is only needed for the AT89C8252!

**See also**
NONE

**Example**
$WAIT 'for at89c8252 only

6.40  ALIAS

**Action**
Indicates that the variable can be referenced with another name.

**Syntax**
newvar ALIAS oldvar

**Remarks**
<table>
<thead>
<tr>
<th>Oldvar</th>
<th>Name of the variable such as P1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newvar</td>
<td>New name of the variable such as direction</td>
</tr>
</tbody>
</table>

Aliasing port pins can give the pin names a more meaningful name.
You can also ALIAS a variable: M ALIAS var.0 for example.

**See also**
CONST
Example
Direction Alias P1.1                      'now you can refer to P1.1
with the variable direction
Set Direction P1.1                      'has the same effect as SET

Dim A As Byte
M Alias A.0
N Alias A.1
Set M
Set N
If M = N Then
  Print "Both bits are set"
End If
End

6.41 ABS

Action
Returns the absolute value of a numeric variable.

Syntax
var = ABS(var2)

Remarks
<table>
<thead>
<tr>
<th>var</th>
<th>Variable that is assigned the absolute value of var2. Var must be a numeric variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var2</td>
<td>The source variable to retrieve the absolute value from. Var2 must be an integer or long.</td>
</tr>
</tbody>
</table>

The absolute value of a number is always positive.

See also
NONE

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

Output
1000
6.42 ASC

**Action**
Convert a string into its ASCII value.

**Syntax**

```plaintext
var = ASC(string)
```

**Remarks**

<table>
<thead>
<tr>
<th>var</th>
<th>Target variable that is assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String variable or constant to retrieve the ASCII value from.</td>
</tr>
</tbody>
</table>

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.

**See also**

CHR

**Example**

```plaintext
Dim A As Byte, S As String * 10
S = "Abc"
A = Asc(s)
Print A
End
```

**Output**

65

6.43 AVG

**Action**
Returns the average value of a byte array.

**Syntax**

```plaintext
var = AVG(ar(1))
```

**Remarks**

<table>
<thead>
<tr>
<th>var</th>
<th>Numeric variable that will be assigned with the lowest value of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar()</td>
<td>The first array element of the array to return the lowest value of.</td>
</tr>
</tbody>
</table>
At the moment AVG() works only with BYTE arrays. Support for other data types will be added too.

See also
MAX, MIN

Example
Dim ar(10) As Byte
Dim bP as Byte
For bP = 1 to 10
    ar(bP) = bP
Next
bP = Avg(ar(1))
Print bP
End

6.44 BAUD

Action
Instruct the compiler to set a new baud rate at run time.

Syntax
BAUD = var

Remarks
Var | The baud rate that you want to use.

var : Constant.

When you want to use a crystal/baud rate that can’t be selected from the options, you can assign this special variable. Do not confuse it with the $BAUD directive!

See also
$CRYSTAL, $BAUD

Example
$BAUD = 2400
$CRYSTAL = 14000000        ' 14 MHz crystal
PRINT "Hello"
BAUD = 9600
Print "Hello"
END
6.45 BCD

**Action**  
Converts a variable into its BCD value.

**Syntax**  
```
PRINT BCD(var)
LCD BCD(var)
```

**Remarks**  
| Var | Variable to convert. This must be a numeric variable or constant. |

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

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

**See also**  
MAKEBCD, MAKEDEC

**Example**  
```
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcd Bcd(a)
End
```

6.46 BITWAIT

**Action**  
Wait until a bit is set or reset.

**Syntax**  
```
BITWAIT x SET | RESET
```

**Remarks**  
| x | Bit variable or internal register like P1.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 P1.0 this doesn't apply.
See also
NONE

Example
Dim A As Bit
Bitwait A , Set                     'wait until bit a is set
Bitwait P1.7 , Reset                'wait until bit 7 of Port 1 is 0.
End

ASM
BITWAIT P1.0 , SET will generate :
Jnb h'91,*+0

BITWAIT P1.0 , RESET will generate :
Jb h'91,*+0

6.47 BREAK

Action
Generates a reserved opcode to pause the simulator.

Syntax
BREAK

Remarks
You can set a breakpoint in the simulator but you can also set a breakpoint from code using the BREAK statement. Be sure to remove the BREAK statements when you debugged your program or use the $NOBREAK directive.

The reserved opcode used is A5.

See also
$NOBREAK

Example
PRINT "Hello"
BREAK                  'the simulator will pause now
End

6.48 CALL

Action
Call and execute a subroutine.
**Syntax**

`CALL Test [(var1, var-n)]`

**Remarks**

<table>
<thead>
<tr>
<th>var1</th>
<th>Any BASCOM variable or constant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var-n</td>
<td>Any BASCOM variable or constant.</td>
</tr>
<tr>
<td>Test</td>
<td>Name of the subroutine. In this case Test</td>
</tr>
</tbody>
</table>

With the CALL statement you can call a procedure or subroutine. As much as 10 parameters can be passed but you can also call a subroutine without parameters.

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,z

**See also**

`DECLARE`, `SUB`

**Example**

```bascom
Dim A As Byte , Bb As Byte
Declare Sub Test(bb As Byte)
A = 65
Call Test(a)                          'call test with parameter A
Test A                                'alternative call
End

Sub Test(bb As Byte)                 'use the same variable as the
declared one
   Lcd Bb                             'put it on the LCD
   Lowerline
   Lcd Bcd(bb)
End Sub
```

**6.49 CHR**

**Action**

Convert a byte, Integer/Word variable or a constant to a character.

**Syntax**

```bascom
PRINT CHR(var)
s = CHR(var)
```
### Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>Byte, Integer/Word variable or numeric constant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>A string variable.</td>
</tr>
</tbody>
</table>

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

### See also

ASC

### Example

```bascom
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcdhex A
Lcd Chr(a)
End
```

### 6.50 CLS

#### Action

Clear the LCD display and set the cursor home.

#### Syntax

```bascom
CLS
```

#### Syntax for graphic LCD

```bascom
CLS TEXT
CLS GRAPH
CLS BOTH
```

#### Remarks

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

### See also

$LCD, LCD

### Example

```bascom
Cls
Lcd "Hello"
Wait 5
Cls
End
```
6.51 CONST

**Action**
Declares a symbolic constant.

**Syntax**
CONST symbol = value

**Remarks**

<table>
<thead>
<tr>
<th>symbol</th>
<th>The name of the symbol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value to assign to the symbol.</td>
</tr>
</tbody>
</table>

Assigned constants consume no program memory. The compiler will replace all occurrences of the symbol with the assigned value. Value may also be an expression that uses other defined constants. The functions that may be used for the expressions are: ASC, ABS, ATN, COS, EXP, FIX, INT, LOG, RND, SGN, SIN, SQR, TAN. Operators are: AND, OR, XOR, +, -, /, \, ^, *, NOT, >, <, =, >=, <=, (, )

**See also**
DIM

**Example**

```bas
'----------------------------------------------------'              (c) 1995-2006 MCS Electronics
'              CONST.BAS'----------------------------------------------------
Dim A As Const 5                        'declare a as a constant
Dim B1 As Const &B1001
Dim S As Single
'Or use the new preferred syntax
Const Cbyte = &HF
Const Cint = -1000
Const Csingle = 1.1
Const Cstring = "test"

S = Csingle
Print S ; " " ; Cstring
Waitms A                                'wait for 5 milliseconds
Print A
Print B1
End
```

6.52 CONFIG

The config statement configures all kind of hardware related statements. Select one of the following topics to learn more about a specific config statement.

CONFIG TIMER0, TIMER1
6.53 CONFIG 1WIRE

Action
Configure the pin to use for 1WIRE statements.

Syntax
CONFIG 1WIRE = pin

Remarks
pin The port pin to use such as P1.0

See also
1WRESET, 1READ, 1WRITE

Example
Config 1WIRE = P1.0 'P1.0 is used for the 1-wire bus
1WRESET 'reset the bus

6.54 CONFIG ADUC812

Action
Configures the ADUC812 microprocessor.

Syntax for ADC
Config ADUC812 = ADCON, MODE = mode, CLOCK = clock, AQUISITION = aq, TIMER2 = tm, EXTRIG = value

Syntax for DAC
**Config ADUC812** = DAC, **MODE** = mode, **RANGE0** = r0, **RANGE1** = r1, **CLEAR0** = clr0, **CLEAR1** = clr1, **SYNC** = sync, **POWER0** = pwr0, **POWER1** = pwr1

### Remarks ADC

<table>
<thead>
<tr>
<th>mode</th>
<th>POWERDOWN, NORMAL, PDNE, STANDBY. PDNE means POWERDOWN if not executing a conversion cycle.</th>
</tr>
</thead>
</table>
| clock | This is a constant that specifies the clock division of the master clock. It may be 1,2,4 or 8.  
An ADC conversion will require 16 ADC clocks in addition to the selected number of acquisition clocks. |
| aq | This is a constant that specifies the time available for the input/track hold amplifier to acquire the input signal.  
It may be in range from 1-4. 1 Acquisition clock is enough for an impedance up to 8K |
| tm2 | The TIMER2 can be ENABLED or DISABLED. When enabled the timer2 overflow serves as a trigger for the AD conversion. |
| value | The external trigger may be ENABLED or DISABLED. When enabled the external pin 23 (CONVST) can start the conversion while it is low. |

### Remarks DAC

<table>
<thead>
<tr>
<th>Mode</th>
<th>The DAC can be in 8 bit mode or 12 bit mode. So the parameter may be 8 or 12. Both DACS are set with this parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>The DAC0 range can be set to VDD or VREF. With VDD the range is from 0-VDD. For VREF it is 0-VREF.</td>
</tr>
<tr>
<td>r1</td>
<td>The DAC1 range can be set to VDD or VREF. With VDD the range is from 0-VDD. For VREF it is 0-VREF</td>
</tr>
<tr>
<td>clr0</td>
<td>This parameter when TRUE will clear the DAC0. This will set the output voltage to 0 V.</td>
</tr>
<tr>
<td>clr1</td>
<td>This parameter when TRUE will clear the DAC1. This will set the output voltage to 0 V</td>
</tr>
<tr>
<td>Sync</td>
<td>May be ENABLED or DISABLED. While enabled the DAC outputs as soon as the DACxL SFR's are written. The user can simultaneously update both DAC's by first updating the DACxL/H SFR's while SYNC is disabled. Both DACs will then update when the SYNC is enabled.</td>
</tr>
<tr>
<td>pwr0</td>
<td>This parameter when ON will power ON the DAC0. When OFF the DAC0 is powered OFF.</td>
</tr>
<tr>
<td>pwr1</td>
<td>This parameter when ON will power ON the DAC1. When OFF the DAC1 is powered OFF</td>
</tr>
</tbody>
</table>
## 6.55 CONFIG BAUD

### Action
Configure the uP to select the intern baud rate generator. This baud rate generator is only available in the 80515, 80517, 80535, 80537 and compatible chips.

### Syntax
```
CONFIG BAUD = baud rate
```

### Remarks

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>Baud rate to use : 4800 or 9600</th>
</tr>
</thead>
</table>

### Example
```
CONFIG BAUD = 9600    'use internal baud generator
Print "Hello"
End
```

## 6.56 CONFIG BAUD1

### Action
Configure the uP to select the internal baud rate generator for serial channel 1. This baud rate generator is only available in the 80517 and 80537.

### Syntax
```
CONFIG BAUD1 = baudrate
```

### Remarks

<table>
<thead>
<tr>
<th>Baudrate</th>
<th>Baud rate to use : 2048 - 37500</th>
</tr>
</thead>
</table>

The 80517 and 80537 have 2 serial ports on board.

### See also
```
CONFIG BAUD
```

### Example
```
CONFIG BAUD1 = 9600    'use internal baud generator
OPEN "Com2:" for Binary as #1
Print #1, "Hello"
Close #1
End
```
6.57 CONFIG DEBOUNCE

**Action**
Configures the delay time for the DEBOUNCE statement.

**Syntax**
```
CONFIG DEBOUNCE = time
```

**Remarks**

<table>
<thead>
<tr>
<th><strong>time</strong></th>
<th>A numeric constant which specifies the delay time in mS.</th>
</tr>
</thead>
</table>

When the debounce time is not configured, 25 mS will be used as a default. Note that the delay time is based on a 12 MHz clock frequency.

**See also**
DEBOUNCE

**Example**
Config Debounce = 25 mS                  '25 mS is the default

6.58 CONFIG I2CDELAY

**Action**
Configures the delay for the I2C clock.

**Syntax**
```
CONFIG I2CDELAY = value
```

**Remarks**

| **Value** | A numeric constant.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 will generate the default clock.</td>
</tr>
<tr>
<td></td>
<td>0 will generate a higher clock and &gt;=2 will generate a lower clock</td>
</tr>
<tr>
<td></td>
<td>frequency.</td>
</tr>
</tbody>
</table>

By default the following delay routine is called with an ACALL:

```
Delay5: 
  Nop 
  Ret 
```

For 12 MHz, there is a 1 MHz system clock. So not counting the other statement, the minimal delay is 4 * 2 = 8 cycles.
The I2Cdelay value will insert the number of specified NOP instructions.
By default the settings are right for all I2C devices and when working with a 12 MHz crystal.
See also
CONFIG_SCL, CONFIG_SDA

Example
CONFIG I2CDELAY = 0                'we need a higher clock

6.59  CONFIG GETRC

Action
Configures the GETRC() charge time.

Syntax
Config GETRC = time

Remarks
| Time  | The time in milli seconds to charge the capacitor |

See also
GETRC

6.60  CONFIG GRAPHLCD

Action
Configures the Graphical LCD display.

Syntax
Config GRAPHLCD = type, PORT = mode, CE = pin, CD = cd, COLS = 30

Remarks
| Type | This must be one of the following :
|      | • 240 * 64
|      | • 240 * 128
| mode | This is the name of the port that is used to put the data on the LCD data pins db0-db7. P1 for example.
| Ce   | The name of the pin that is used to enable the chip on the LCD.
| Cd   | The name of the pin that is used to control the CD pin of the display.
| Cols | The number of columns for use as text display. The current code is written for 30 columns only.

In the sample the following connections were used:
P1.0 to P1.7 to DB0-DB7 of the LCD  
P3.2 to FS, font select of LCD can be hard wired too  
P3.5 to CE, chip enable of LCD  
P3.4 to CD, code/data select of LCD  
P3.6 to WR of LCD, write  
P3.7 to RD of LCD, read

The LCD used from www.conrad.de needs a negative voltage for the contrast.  
Two 9V batteries were used with a pot meter.

The FS (font select) must be set low to use 30 columns and 8x8 fonts.  
It may be connected to ground. This pin is not used by the software routines.  
The current asm code only support 30 columns. You can change it however  
to use 40 columns.

The T6963C displays have both a graphical area and a text area. They can  
be used together. The routines use the XOR mode to display both text and  
graphics layered over each other.

The statements that can be used with the graphical LCD are:  
**CLS** will clear the graphic display and the text display  
**CLS GRAPH** will clear only the graphic part of the display  
**CLS TEXT** will only clear the text part of the display  
**CLS BOTH** is the same as CLS and will clear both text and graphics.

**LOCATE** row, column Will place the cursor at the specified row and  
column  
The row may vary from 1 to 8 and the column from 1 to 30.

**CURSOR** ON/OFF BLINK/NOBLINK can be used the same way as for  
text displays.

**LCD** can also be the same way as for text displays.  
**LCDHEX** can also be used the same way as for text display

New are:  
**SHOWPIC** X, Y, Label where X and Y are the column and row and Label  
is the label where the picture info is placed.

**PSET** X, Y, color Will set or reset a pixel. X can range from 0-239 and Y  
from 9-63. When color is 0 the pixel will turned off. When it is 1 the pixel  
will be set on.

**$BGF** "file.bgf" ‘inserts a BGF file at the current location  
$TIFF is removed from the Help but it still supported this version. $BGF  
should be used however.
Example

´--------------------------------------------------------------------------------
´                                (c) 1995-2006 MCS Electronics
´                                GLCD.BAS
´  Sample to show support for T6963C based graphic display
´  Only 240*64 display is supported with 30 columns (yet)
´  At the moment the display can only be used in PORT mode
´  Connection :
´  P1.0 - P1.7  to DB0-DB7 of LCD
´  P3.2     to FS, font select of LCD can be hard wired too
´  P3.5     to CE, chip enable of LCD
´  P3.4     to CD, code/data select of LCD
´  P3.6     to WR of LCD
´  P3.7     to RD of LCD
`A future version will allow external data access too which also uses RD and WR
`The display from www.conrad.com needs a negative voltage for the contrast.
`I used two 9 V batteries
´--------------------------------------------------------------------------------
´configure the LCD display
Config GraphLcd = 240 * 64 , Port = P1 , Ce = P3.5 , Cd = P3.4 , Cols = 30

`dimension some variables used by the DEMO
Dim X As Byte , Y As Byte

, Reset P3.2  `8 bit wide char is 30 columns

`The following statements are supported:
Cls `will clear graphic and text
`cls TEXT will clear only the text
`cls GRAPH will clear only the graphic part

`To init the display manual you can use:
`Lcdinit
`But this should not be needed as it is initialised at start up.

`Locate is supported and you can use 1-8 for the row and 1-30 for the column
Locate 1 , 1

`cursor control is the same as for normal LCD
Cursor On Blink
'And to show some text you can use LCD
Lcd "Hello world"
'Note that the cursor position is not adjusted. You can set it with locate

'Now comes the fun part for using a graphic LCD
'We can display a BMP file. You may use MSPAINT or any other tool that can create
'a BMP file. With the Graphic converter from the Tools Menu you can convert the file
'into a BGF file. (BASCOM GRAPHICS FILE). The conversion will convert all non white
'pixels to BLACK.

'To display the BGF file you use the SHOWPIC statement that needs an X and Y parameter
'the third param is the label where the data is stored.
'The position must be dividable by 8 because this is the way the display handles the data

Showpic 0 , 0 , Picture1

'And we use the PSET known from QB to set or reset a single pixel
'A value of 0 means clear the pixel and 1 means set the pixel

'create a block
For X = 0 To 10
   For Y = 0 To 10
      Pset X , Y , 1
   Next
Next

'You could remove it too
For X = 0 To 10
   For Y = 0 To 10 Step 2
      Pset X , Y , 0
   Next
Next

'A simple scope or data logger could be made with PSET !
'We hope to get an AN from an inspired user :-)
End

'label for the picture
Picture1:
'$BGF includes the data from the specified file
$bgf "samples\mcs.bgf"
6.61 CONFIG LCDPIN

**Action**
 Override the LCD-options to store the settings in your program.

**Syntax**

```
CONFIG LCDPIN = PIN, DB4= P1.1, DB5=P1.2, DB6=P1.3, DB7=P1.4, E=P1.5, RS=P1.6
```

**Remarks**

P1.1 etc. are just an example in the syntax. The pins of the LCD display that must be connected in PIN mode are:

<table>
<thead>
<tr>
<th>Name</th>
<th>LCD Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB4</td>
<td>DB4</td>
</tr>
<tr>
<td>DB5</td>
<td>DB5</td>
</tr>
<tr>
<td>DB6</td>
<td>DB6</td>
</tr>
<tr>
<td>DB7</td>
<td>DB7</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>RS</td>
<td>RS</td>
</tr>
</tbody>
</table>

The WR line of the display must be connected to GND.

**See also**

CONFIG LCD

**Example**

```
CONFIG LCDPIN = PIN, DB4= P1.1, DB5=P1.2, DB6=P1.3, DB7=P1.4, E=P1.5, RS=P1.6
```

6.62 CONFIG LCD

**Action**
 Configure the LCD display.

**Syntax**

```
CONFIG LCD = LCDtype
```

**Remarks**

<table>
<thead>
<tr>
<th>LCDtype</th>
<th>The type of LCD display used. This can be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 * 4, 40 * 2, 16 * 1, 16 * 1a, 16 * 2, 16 * 4, 16 * 4, 20 * 2 or 20 * 4 or 40 * 4a</td>
<td>Default 16 * 2 is assumed.</td>
</tr>
</tbody>
</table>

The 16 * 1a LCD display is a special one. It is intended for the display that has the memory organized as 2 lines of 8 characters.
The 40 * 4a LCD display is also a special one. It has two ENABLE lines. The CONFIG LCDPIN directive must be used to configure the second E line:
CONFIG LCDPIN = PIN , E1 = Pin, E2 = pin, etc.

To select between E1 and E2 you need to set the B register.
Mov b,#0 'selects E1
Mov b,#1 'selects E2

LCD with a constant will work and also with strings.
To call the low level routines:
Mov a,#2 ; code into acc
Mov B,#0 ; or use Mov b,#1
Acall LCD_CONTROL ; call routine

To send data use the low level routine WRITE_LCD instead of LCD_CONTROL

Most LCD routines will work with the 40*4a display but some will fail. In that case you need to use the low level ASM routines as shown above.

Example
REM Sample for normal displays
CONFIG LCD = 40 * 4
LCD "Hello" 'display on LCD
FOURTHLINE 'select line 4
LCD "4" 'display 4
END

6.63 CONFIG LCDBUS

Action
Configures the LCD databus.

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.

See also
CONFIG LCD
Example
$LCD = &H8000                'address of enable signal
Config LCDBUS = 4            '4 bit mode
LCD "hello"

6.64 CONFIG MICROWIRE

Action
Configures the micro wire pins.

Syntax

Config Microwire = Pin, Cs = P1.1, Din = P1.2, Dout = P1.4, Clock = P1.5, Al = 7

Remarks

<table>
<thead>
<tr>
<th>CS</th>
<th>Chip select</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIN</td>
<td>Data input</td>
</tr>
<tr>
<td>DOUT</td>
<td>Data output</td>
</tr>
<tr>
<td>CLOCK</td>
<td>Pin that generates the Clock</td>
</tr>
<tr>
<td>AL</td>
<td>Address lines. See table below. It depends if you work with bytes or words. In our example we will use the 93C46 and work with bytes. AL will be 7 in this case.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chip</th>
<th>93C46</th>
<th>93C56</th>
<th>93C57</th>
<th>93C66</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data bits</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>AL</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

See also

MWINIT, MWWOPCODE, MWWRITE, MWREAD

Example
NONE

6.65 CONFIG PRINT

Action
Configures the PRINT statement.

Syntax

Config PRINT = pin
Config PRINTMODE = mode

Remarks

<table>
<thead>
<tr>
<th>Pin</th>
<th>The pin to use for the output control such as P3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>The mode of the control pin. SET or RESET.</td>
</tr>
</tbody>
</table>

When you want to control a RS-485 device you need an additional pin to control the buffer direction. When the pin must be high during printing use SET. When it must be low during print use RESET.

Example

Config Print = P3.0  'this pin controls the buffer
Config mode = SET 'during PRINT this pin goes high.
Print "Hello"

6.66 CONFIG SCL

Action

Overrides the SCL pin assignment from the Option Settings.

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.

This statement can not be used to change the pin dynamically during runtime.

See also

CONFIG SDA, CONFIG I2CDelay

Example

CONFIG SCL = P3.5  'P3.5 is the SCL line

6.67 CONFIG SDA

Action

Overrides the SDA pin assignment from the Option Settings.

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.

See also
CONFIG SCL, CONFIG I2CDELAY

Example
CONFIG SDA = P3.7 'P3.7 is the SDA line

6.68 CONFIG SERVOS

Action
Configures the number of servos and their pins.

Syntax
Config SERVOS = number, SERVO1 = P1.1, SERVO2 = P1.2, SERVO3 = P1.4, SERVO4 = P1.5, RELOAD = value

Remarks

<table>
<thead>
<tr>
<th>number</th>
<th>The number of servos you want to use. When you specify 2, you must also add the SERVO1 and SERVO2 parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>servo1</td>
<td>The pin that is attached to servo 1.</td>
</tr>
<tr>
<td>servo2</td>
<td>The pin that is attached to servo 2.</td>
</tr>
<tr>
<td>servo3</td>
<td>The pin that is attached to servo 3.</td>
</tr>
<tr>
<td>servo4</td>
<td>The pin that is attached to servo 4.</td>
</tr>
<tr>
<td>RELOAD</td>
<td>The reload value in uS. Default 100 uS</td>
</tr>
</tbody>
</table>

The CONFIG SERVOS compiler directive will include an interrupt that will execute every 100 uS. The TIMER0 interrupt is enabled and the TIMER0 is started. The number of bytes used by the use of SERVO's is 1 + number of servos.

When you use 2 servo's, it will take 3 bytes of internal memory. TIMER0 can not be used by your program anymore. To change the pulse duration you assign the special reserved variables the number of 100 uS steps:
SERVO1 = 8 '800 uS pulse
SERVO2 = 12 '1200 uS duration

© 2007 MCS Electronics
After 20 mS the pulses will be sent again to the port pins.

The maximum number of servo's is 14. The example shows how to set it up for 4 servo's only.
When you specify RELOAD = 50, 50 uS steps will be used!
When you have a lot of servo's the RELOAD must be higher than when you have less servos. When you have a reload of 10 uS for example it will be impossible for the 8051 to handle more than 1 servo without losing time.
For 2 servo's 20 or 25 should be used for best results.

6.69 **CONFIG SPI**

**Action**

Configures the SPI related statements.

**Syntax**

\[
\text{CONFIG SPI} = \text{SOFT}, \text{DIN} = \text{PIN}, \text{DOUT} = \text{PIN}, \text{CS} = \text{PIN}, \text{CLK} = \text{PIN}, \text{DATA ORDER} = \text{DO}, \text{NOC S} = \text{ON/FF} \\
\text{CONFIG SPI} = \text{HARD}, \text{INTERRUPT} = \text{ON/OFF}, \text{DATA ORDER} = \text{LSB/MSB}, \text{MASTER=}\text{YES/NO}, \text{POLARITY=}\text{HIGH/LOW}, \text{PHASE=}0\text{1}, \text{CLOCKRATE=}4\text{16}64\text{128}
\]

**Remarks**

When you use the software SPI mode you must specify the following information:

<table>
<thead>
<tr>
<th>DIN</th>
<th>Data input. Pin is the pin number to use such as p1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUT</td>
<td>Data output. Pin is the pin number to use such as p1.1</td>
</tr>
<tr>
<td>CS</td>
<td>Chip select. Pin is the pin number to use such as p1.2</td>
</tr>
<tr>
<td>CLK</td>
<td>Clock. Pin is the pin number to use such as p1.3</td>
</tr>
<tr>
<td>NOCS</td>
<td>Option without parameter. Use it to disable the resetting and setting of the CS pin.</td>
</tr>
<tr>
<td>DATA ORDER</td>
<td>Use MSB or LSB. With MSB, MS bit will be sent first. LSB option will send the LS bit first.</td>
</tr>
<tr>
<td>SPIOUTEDGE</td>
<td>Falling or Rising. Falling is the default. The edge specifies if the data will be clocked with a low to high or a high to low edge.</td>
</tr>
</tbody>
</table>

When the NOCS option is used you must reset and set the CS pin yourself. The option is intended when you want to do large transfers between the micro and the SPI device. With the little internal memory you can do that in steps but of course you don't want the CS pin to change after each use of the SPIIN or SPIOUT routine.

When you want to use the hardware SPI that is available in the 89S8252, you must specify the following information:

| INTERRUPT | ON or OFF to enable or disable that the SPI interrupt is set. |
| DATA ORDER | LSB or MSB. Determines which bit is sent first. |
| MASTER    | Yes or No. Set it to Yes for usage with the BASCOM SPI routines. |
POLARITY | High or Low. See the Atmel datasheet
PHASE | 0 or 1.
CLOCKRATE | 4, 16, 64 or 128. This is a division that determines the clock rate. The oscillator clock is divided by the number you specify.
ON | You can turn on/enable SPI by using this option. It sets the enable bit.
OFF | You can turn off the SPI by using this option. It resets the enable bit.

See also
SPIIN $F_{23h}$ SPIOUT $F_{22h}$

Example
Config SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS = P1.2, CLK = P1.3
SPIINIT ' init pins
SPIOUT var , 1 'send 1 byte

6.70 CONFIG TIMER0, TIMER1

Action
Configure TIMER0 or TIMER1.

Syntax
CONFIG TIMERx = COUNTER/TIMER , GATE=INTERNAL/EXTERNAL , MODE=0/3

Remarks
| TIMERx | TIMER0 or TIMER1. COUNTER will configure TIMERx as a COUNTER and TIMER will configure TIMERx as a TIMER. A TIMER has built in clock input and a COUNTER has external clock input. |
| GATE | INTERNAL or EXTERNAL. Specify EXTERNAL to enable gate control with the INT input. |
| MODE | Time/counter mode 0-3. See Hardware for more details. |

So CONFIG TIMER0 = COUNTER, GATE = INTERNAL, MODE=2 will configure TIMER0 as a COUNTER with no external gate control, in mode 2 (auto reload)

When the timer/counter is configured the timer/counter is stopped so you must start it afterwards with the START TIMERx statement.

See the additional statements for other microprocessors that use the CONFIG statement.

Example
CONFIG TIMER0=COUNTER, MODE=1, GATE=INTERNAL
**6.71 CONFIG WATCHDOG**

**Action**
Configures the watchdog timer from the AT89C8252

**Syntax**

```
CONFIG WATCHDOG = time
```

**Remarks**

<table>
<thead>
<tr>
<th>Time</th>
<th>The interval constant in mS the watchdog timer will count to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>16, 32, 64, 128, 256, 512, 1024 and 2048.</td>
<td>Possible settings:</td>
</tr>
</tbody>
</table>

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.

**See also**

START WATCHDOG, STOP WATCHDOG, RESET WATCHDOG

**Example**

```
'-----------------------------------------------------
'(c) 1995-2006 MCS Electronics
' WATCHD.BAS demonstrates the AT89S8252 watchdog timer
' select 89s8252.dat !!!
'-----------------------------------------------------
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 doesn't 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
```

**6.72 COUNTER**

**Action**
Set or retrieve the COUNTER0 or COUNTER1 variable. For 8052 TIMER2 compatible chips, COUNTER2 can be used too.
Syntax

\[
\text{COUNTERX} = \text{var} \\
\text{var} = \text{COUNTERX}
\]

Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>A byte, Integer/Word variable or constant that is assigned to the counter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>counterX</td>
<td>COUNTER0, COUNTER1 or COUNTER2.</td>
</tr>
</tbody>
</table>

Use counterX = 0 to reset the counter.
The counter can count from 0 to 255 in mode 2 (8-bit auto reload).
And to 65535 in mode 1 (16-bit).
In mode 0 the counter can count to 8192. The MSB and 5 bits of the LSB are used in that case. When you assign a constant to a TIMER/COUNTER in mode 0, the bits will be placed in the right place:
\[
\text{COUNTER0} = \&B1\_1111\_1111\_1111\_1111 \ '13 \text{ bits}
\]
Will be translated for mode 0 into \( 1111\_1111\_0001\_1111 \)

The counterX variables are intended to set/retrieve the TIMER/COUNTER registers from BASCOM. COUNTER0 = TL0 and TH0.
So the COUNTERX reserved variable is a 16 bit variable.

To set TLx or THx, you can use : TL0 = 5 for example.

Note that the COUNTERx variable operates on both the TIMERS and COUNTER because the TIMERS and COUNTERS are the same thing except for the mode they are working in. To load a reload value, use the LOAD statement.

⚠️ After access to the counter, the timer/counter is stopped. So when it was running, start it with the statement START COUNTERx

Example

```
'--------------------------------------------------------------
' (c) 1995-2006 MCS Electronics
' file: COUNTER.BAS
' demo: COUNTER
'--------------------------------------------------------------
' Connect the timer input P3.4 to a frequency generator
' *TIMER/COUNTER 1 is used for RS-232 baud rate generator
'
Dim A As Byte, C As Integer
Config Timer0 = Counter, Gate = Internal, Mode = 1
'Timer0 = counter : timer0 operates as a counter
'Gate = Internal : no external gate control
'Mode = 1 : 16-bit counter

Counter0 = 0 'clear counter
Start Counter0 'enable the counter to count
Do
A = Inkey 'set up a loop
'Do
A = Inkey 'check for input
```
C = Counter0  'get counter value
Print C     'print it
Start Counter0  're-start it because it was stopped by accessing the COUNTER
Loop Until A = 27   'until escape is pressed
End

For the next example the ASM code is shown:
COUNTER0 = 1000

Generated code :
Clr TCON.4
Mov tl0,#232
Mov th0,#3

6.73 **CPEEK**

**Action**
Returns a byte stored in code memory.

**Syntax**

```plaintext
var = CPEEK(address)
```

**Remarks**

<table>
<thead>
<tr>
<th>var</th>
<th>Numeric variable that is assigned with the content of the program memory at address</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Numeric variable or constant with the address location</td>
</tr>
</tbody>
</table>

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

**See also**

PEEK, POKE, INP, OUT

**Example**

'-----------------------------------------------------
'       (c) 1995-2006 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 127    'for a 8052 225 could be used
  Break
B1 = Peek(i)       'get byte from internal
memory
    Printhex B1 ; " " ;
    'Poke I , 1                                    'write a value into memory
Next
Print                                             'new line
'be careful when writing into internal memory !!

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

Out &H8000 , 1                                    'write 1 into XRAM at address 8000
    B1 = INP(&H8000)                        'return value from XRAM
Print B1
End

6.74 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. For Graphic LCD displays the state is ON BLINK

See also
DISPLAY

Example
Dim a as byte
A = 255
LCD a
Cursor Off                                      'hide cursor
Wait 1                                          'wait 1 second
Cursor Blink                                    'blink cursor
6.75 DATA

**Action**
Specifies values to be read by subsequent READ statements.

**Syntax**

```
DATA var [, varn]
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Numeric or string constant.</th>
</tr>
</thead>
</table>

To specify a character that cannot be written in the editor such as " you can use $34. The number is the ASCII value of the string. A null will be added so it will be a string of one character!

When you want to store the string data without the ending null you can use the $NONULL directive as shown below:

```
DATA "abcd" 'stored with and ending 0
$NONULL = -1 'from now on store the data without the extra 0
DATA "abed", "edgh"
$NONULL = 0 'and go back to the normal default operation
```

Version 2.09 supports expressions. You must use either expressions or normal constant data on the DATA lines. You may not mix them.

```
DATA INTEGER(15 * constval + x)
```

Where constval is a declare constant (CONST) and x is a CONST too.

The INTEGER() function must be used to indicate that the resulting constant is of the integer type.

Use WORD(), INTEGER(), LONG() or SINGLE() to specify the resulting constant.

**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.

**See also**

READ, RESTORE

**Example**

```
Dim A As Byte, I As Byte, L As Long, S As Xram String * 15
Restore Dta1 'point to data
For A = 1 To 3
    Read I : Print I 'read data and print it
Next

Restore Dta2 'point to data
Read I : Print I ' integer data
Read I : Print I
```
Restore Dta3
Read L : Print L ' long data

Restore Dta4
Read S : Print S ' string data

END

DTA1:
Data 5 , 10 , 100

DTA2:
Data -1% , 1000%
' Integer and Word constants must end with the %-sign.
' (Integer : <0 or >255)

DTA3:
Data 1235678&
'long constants must end with the &-sign

DTA4:
Data "Hello world" , $34
REM You can also mix different constant types on one line
Data "TEST" , 5 , 1000% , -1& , 1.1!

6.76 DEBOUNCE

Action
Debounces a port pin connected to a switch.

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

Remarks
<table>
<thead>
<tr>
<th>Px.y</th>
<th>A port pin like P1.0, to examine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>0 for jumping when Px.y is low, 1 for jumping when Px.y is high</td>
</tr>
<tr>
<td>Label</td>
<td>The label to GOTO when the specified state is detected</td>
</tr>
<tr>
<td>SUB</td>
<td>The label to GOSUB when the specified state is detected</td>
</tr>
</tbody>
</table>

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 will wait 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 uses a different port uses 1 BIT of the internal memory to hold it’s state.

What also should be mentioned is that P2.2-P2.7 and P3 have internal pull up resistors. This can affect the debounce statement. With these port pins, debounce is
best to be used as: **Debounce P1.1, 0, Pr [, sub ]**, as it will not require an external pull up resistor.

**See also**

[CONFIG DEBOUNCE](#)

### Example

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

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

'When P1.0 goes low jump to subroutine Pr
'P1.0 must go high again before it jumps again
'to the label Pr when P1.0 is low

Debounce P1.0 , 1                       'no branch
Debounce P1.0 , 1 , Pr                  'will result in a return
without gosub
End

Pr:
  Print "P1.0 was/is low"
Return
```

### 6.77 DECR

**Action**

Decrement a variable by one.

**Syntax**

```
DECR var
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Variable to be decremented.</th>
</tr>
</thead>
</table>

There are often situations where you want a number to be decreased by 1.
The **DECR** statement is faster than `var = var - 1`.

**See also**

**INCR**

---

**Example**

```bas
Dim A As Byte

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

---

### 6.78 DECLARE

**Action**
Declarates a subroutine.

**Syntax**

```
DECLARE SUB TEST[(var as type)]
```

**Remarks**

<table>
<thead>
<tr>
<th>test</th>
<th>Name of the procedure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>Name of the variable(s). Maximum 10 allowed.</td>
</tr>
<tr>
<td>Type</td>
<td>Type of the variable(s). Bit, Byte, Word, Integer, Long or String.</td>
</tr>
</tbody>
</table>

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

**See also**

**CALL**, **SUB**

---

**Example**

```bas
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

### 6.79 DEF

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

**Syntax**
```
DEFBIT b
DEFBYTE c
DEFINT I
DEFWORD x
```

**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
```

### 6.80 DEFLCDCHAR

**Action**
Define a custom LCD character.

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

**Remarks**
```
<table>
<thead>
<tr>
<th>char</th>
<th>Variable representing the character (0-7).</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1-r8</td>
<td>The row values for the character.</td>
</tr>
</tbody>
</table>
```

*char : Byte, Integer, Word, Long, Constant.*

* r1-r8 : Constant.*

You can use the LCD designer to build the characters.
It is important that after the DEFLCDCHAR statement(s), a CLS follows. The special characters can be printed with the Chr() function.

**See also**
*Edit LCD designer*, *LCD*.

**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
```

### 6.81 DELAY

**Action**
Delay program execution for a short time.

**Syntax**
```
DELAY
```

**Remarks**
Use DELAY to wait for a short time. The delay time is 100 microseconds based on a system frequency of 12 MHz.

**See also**
*WAIT*, *WAITMS*.

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

### 6.82 DIM

**Action**
Dimension a variable.

**Syntax**
```
DIM var AS [XRAM/IRAM] type
```

**Remarks**
<table>
<thead>
<tr>
<th>Var</th>
<th>Any valid variable name such as b1, i or longname. var can also be an array: ar(10) for example.</th>
</tr>
</thead>
</table>
A string variable needs an additional parameter that specifies the length of the string:

Dim s As XRAM String * 10

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

Note that BITS can only be stored in internal memory.

**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. Also the XRAM/IRAM options are not available in QB.

**See Also**

CONST, ERASE

**Example**

'--------------------------------------------------------------
'                     (c) 1995-2006 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 S As Single

'Assign bits
B1 = 1
Set B1 'or

'Assign bytes
A = 12
A = A + 1

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

Print C

'Assign long
L = 12345678

Print L

'Assign single
S = 1234.567

Print S
End

6.83 DISABLE

Action
Disable specified interrupt.

Syntax
DISABLE interrupt

Remarks

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>INTO, INT1, SERIAL, TIMER0, TIMER1 or TIMER2.</th>
</tr>
</thead>
</table>

For other chips: INT2, INT3, INT4, INT5, INT6, INT7, INT8, CAN

By default all interrupts are disabled.
To disable all interrupts specify INTERRUPTS.
To enable the enabling and disabling of individual interrupts use ENABLE INTERRUPTS.

Depending on the chip used, there can be more interrupts.
Look at microprocessor support for more details.

See also
ENABLE

Example

Enable Interrupts 'enable the setting of
interrupts
Enable Timer0 'enable TIMER0
Disable Serial 'disables the serial
interrupt.
Disable Interrupts 'disable all interrupts

6.84 DISPLAY

Action
Turn LCD display on or off.

Syntax
DISPLAY ON / OFF

Remarks
The display is turned on at power up.
See also
CURSOR, LCD

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

6.85 DO

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.

See also
EXIT, WHILE, WEND, FOR, NEXT

Example
Dim A As Byte
Do
   A = A + 1
   Print A
Loop Until A = 10
Print A

   'start the loop
   'increment A
   'print it
   'Repeat loop until A = 10
   'A is still 10 here

6.86 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 CASE

Example
Dim A As Byte
A = 10                                    'let a = 10
If A > 10 Then
   Print "A >10"                          'this will not be printed
Else
   Print "A not greater than 10"          'this will be printed
END IF

6.87 ENABLE

Action
Enable specified interrupt.

Syntax
ENABLE interrupt

Remarks

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>INT0, INT1, SERIAL, TIMER0, TIMER1 or TIMER2</th>
</tr>
</thead>
</table>

For other chips also : INT2, INT3, INT4, INT5, INT6, INT7, INT8, CAN

By default all interrupts are disabled.
To enable the enabling and disabling of interrupts use ENABLE INTERRUPTS.

Other microprocessors can have more interrupts than the 8051/8052. Look at specific microprocessor support for more details.

See also
DISABLE
Example
ENABLE INTERRUPTS 'allow interrupts to be set
ENABLE TIMER1 'enables the TIMER1 interrupt

6.88 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, a never ending loop is generated.

See also
STOP

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

6.89 END IF

Action
End an IF .. THEN structure.

Syntax
END IF

Remarks
You must always end an IF .. THEN structure with an END IF statement.

You can nest IF .. THEN statements.
The use of ELSE is optional.

The editor converts ENDIF to End If when the reformat option is switched on.

See also
IF THEN, ELSE

Example
Dim Nmb As Byte
Again:
  'label
Input " Number " , Nmb
  'ask for number
If Nmb = 10 Then
  'compare
    Print " Number is 10"
      'yes
Else
  'no
    If Nmb > 10 Then
      'is it greater
        Print " Number > 10"
          'yes
    Else
      'no
        Print " Number < 10"
          'print this
    End If
  End If
End If
  'end structure
End If
  'end structure
End 'end program

6.90 ERASE

Action
Erases a variable so memory will be released.

Syntax
ERASE var

Remarks
| var | The name of the variable to erase. |

The variable must be dimensioned before you can erase it.

When you need temporary variables you can erase them after you used them. This way your program uses less memory.

You can only ERASE the last dimensioned variables. So when you DIM 2 variables for local purposes, you must ERASE these variables. The order in which you ERASE them doesn't matter.

For example:
Dim a1 as byte, a2 as byte, a3 as byte, a4 as byte
 'use the vars
ERASE a3 : ERASE a4 'erase the last 2 vars because they were temp vars
Dim a5 as Byte 'Dim new var
Now you can't erase the vars a1 and a2 anymore!

Note that ERASED variables don't show up in the report file nor in the simulator.

Example
Dim A As Byte
  'DIM variable
A = 255
  'assign value
Print A
  'PRINT variable
Erase A
  'ERASE
Dim A As Integer  'DIM again but now as INT
Print A          'PRINT again
REM Note that A uses the same space as the previous ERASED var A so
REM it still holds the value of the previous assigned variable

6.91 EXIT

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

Syntax
EXIT [FOR] [DO] [WHILE] [SUB]

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

See also
FOR [tag], DO [tag], WHILE [tag]

Example
Dim A As Byte, B1 As Byte  'DIM variable
A = 2 : B1 = 1
If A >= B1 Then
   Do
      A = A + 1  'inc a
      If A = 100 Then  'test for a = 100
         Exit Do  'exit the DO..LOOP
      End If
   Loop  'end the DO
End If  'end the IF..THEN

6.92 FOR

Action
Execute a block of statements a number of times.

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

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>The variable counter to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>The starting value of the variable var</td>
</tr>
<tr>
<td>End</td>
<td>The ending value of the variable var</td>
</tr>
<tr>
<td>Value</td>
<td>The value var is increased/decreased with each time NEXT is encountered.</td>
</tr>
</tbody>
</table>

For incremental loops you must use TO.
For decremental loops you must use DOWNTO.
You may use TO for a decremental loop but in that case you must use a negative STEP:
  For a = 10 To 1 STEP -1
You must end a FOR structure with the NEXT statement.
The use of STEP is optional. By default a value of 1 is used.

See also
NEXT, EXIT FOR

Example
Dim Y As Byte, A As Byte, x as byte

  y = 10    'make y 10
  For A = 1 To 10                           'do this 10 times
    For X = Y To 1                           'this one also
      Print X ; A                             'print the values
    Next                                       'next x (count down)
  Next                                        'next a (count up)

Dim S As Single
  For S = 1 To 2 Step 0.1                    'print the values
    Print S
  Next
End

6.93 FOURTHLINE

Action
Reset LCD cursor to the fourth line.

Syntax
FOURTHLINE

Remarks
Only valid for LCD displays with 4 lines.

See also
HOME, UPPERLINE, LOWERLINE, THIRDLINE, LOCATE
Example
Dim a as byte
a = 255
LCD a
Fourthline
LCD a
Upperline
END

6.94 FUSING

Action
Formats a floating point value.

Syntax
var = Fusing( source, mask)

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>The string that is assigned with the result.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>A variable of the type single that must be formatted.</td>
</tr>
</tbody>
</table>
| Mask      | The formatting mask . ###.##
            | The # sign is used to indicate the number of digits before and after the decimal point. Normal rounding is used. |
            | When you don't need rounding the result, use the & sign instead of the # sign after the point. |
            | When you want leading zero's use the 0 character before the point. |

See also
STR

Example
Dim S As Single, Targ As String * 16
' The FUSING() function formats a single into a string in order to 'represent it better without all the digits after the point

'assign single
S = 99.4999
Targ = Fusing(s, ##.)
Print Targ
' with the # mask, you can provide the number of digits before and after 'the point
'the result should be 99.5

'with a 0 before the point, you can indicate how many digits you want
to 'have filled with zeros
Targ = Fusing(s , 000.#)
'the result should be 099.5

'When you dont want that the result is rounded, you can use the &
indicator
Targ = Fusing(s , 000.&&)
'result should be 099.49

'note that if the number of digits you provide is not enough to store
the 'result result is extended automaticly

'Also note that the - sign will use one digit of the mask too
S = -99.12
Targ = Fusing(s , 00.&&)
'result is -99.12

End

6.95 GET

Action
Retrieves a byte from the software UART.

Syntax
GET #channel , var

Remarks

<table>
<thead>
<tr>
<th>Channel</th>
<th>Positive numeric constant that refers to the opened channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>A variable that receives the value from the software UART.</td>
</tr>
</tbody>
</table>

Note that the channel must be opened with the OPEN statement.
Also, note that the CLOSE statement, must be the last in your program. Please see
comment on OPEN statement
An optional TIMEOUT can be specified so that the routine will return when no
character is received.

See also
PUT[07], $TIMEOUT[07]

Example
Dim S As String * 12, I As Byte, A As Byte, Dum As Byte

Open "com3.1:9600" For Output As #1 'p3.1 is normally used for tx so testing is easy
Open "com3.0:9600" For Input As #2 'p3.0 is normally used for RX so testing is easy

S = "test this" 'assign string
Dum = Len(s) 'get length of string
For I = 1 To Dum 'for all characters from left to right
    A = Mid(s, I, 1) 'get character
    Put #1, A 'write it to comport
Next

Do
    Get #2, A 'get character from comport
    Put #1, A 'write it back
    Print A 'use normal channel
Loop

Printbin #1, a 'Printbin is also supported
Inputbin #2, a 'Inputbin is also supported

Close #1 'finally close device
Close #2
End

'To use the TIMEOUT option include (without the remarks):
'$TIMEOUT
' Get #2, A TIMEOUT = 10000 'get character from comport

6.96 GETAD

Action
Retrieves the analog value from channel 0-7. Channel ranges from 0-11 on a 80517 or 80537.

Syntax
var = GETAD(channel, range)

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the A/D value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>The channel to measure</td>
</tr>
<tr>
<td>Range</td>
<td>The internal range selection.</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>0 = 0.5 Volt</td>
<td>192 = 0 - 3.75 Volt</td>
</tr>
<tr>
<td>128 = 0 - 2.5 Volt</td>
<td>64 = 0 - 1.25 Volt</td>
</tr>
<tr>
<td>12 = 3.75 - 5 Volt</td>
<td>200 = 2.5 - 3.75 Volt</td>
</tr>
<tr>
<td>132 = 1.25 - 2.5 Volt</td>
<td></td>
</tr>
</tbody>
</table>

The GETAD() function is only intended for the 80515, 80535, 80517, 80535 and 80552. For the 89C051 use GETAD2051(). It is a microprocessor depended support feature.

See also
GETAD2051

Example
Dim b1 as Byte, Channel as byte, ref as byte
channel=0 'input at P6.0
ref=0 'range from 0 to 5 Volt
b1=getad(channel, ref) 'place A/D into b1

6.97 GETAD2051

Action
Retrieves the analog value from a 89C2051 or 89C4051.

Syntax
var = GETAD2051()

Remarks
| var | The variable that is assigned with the A/D value |

The GETAD2051() function is only intended for the 89C2051 and 89C4051. It uses the analog comparator of the chip.

Connect the hardware as following:
See also

Example

$regfile = "89c2051.dat"
Dim A As Byte
Do
    A = Getad2051()
    A = Lookup(a , Dta)
    Print A
Loop
End

'this table converts the value into a packed BCD value
'this value can be used to display the value on 2 7-segment displays

Dta:
Data 0                 ' 0  0.000
Data 1                 ' 1  0.047
Data 1                 ' 2  0.093
<table>
<thead>
<tr>
<th>Data 2</th>
<th>' 3 0.138</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 2</td>
<td>' 4 0.184</td>
</tr>
<tr>
<td>Data 3</td>
<td>' 5 0.229</td>
</tr>
<tr>
<td>Data 3</td>
<td>' 6 0.273</td>
</tr>
<tr>
<td>Data 3</td>
<td>' 7 0.317</td>
</tr>
<tr>
<td>Data 4</td>
<td>' 8 0.361</td>
</tr>
<tr>
<td>Data 4</td>
<td>' 9 0.404</td>
</tr>
<tr>
<td>Data 5</td>
<td>'10 0.447</td>
</tr>
<tr>
<td>Data 5</td>
<td>'11 0.489</td>
</tr>
<tr>
<td>Data 6</td>
<td>'12 0.531</td>
</tr>
<tr>
<td>Data 6</td>
<td>'13 0.573</td>
</tr>
<tr>
<td>Data 6</td>
<td>'14 0.614</td>
</tr>
<tr>
<td>Data 7</td>
<td>'15 0.655</td>
</tr>
<tr>
<td>Data 7</td>
<td>'16 0.696</td>
</tr>
<tr>
<td>Data 8</td>
<td>'17 0.736</td>
</tr>
<tr>
<td>Data 8</td>
<td>'18 0.776</td>
</tr>
<tr>
<td>Data 8</td>
<td>'19 0.815</td>
</tr>
<tr>
<td>Data 9</td>
<td>'20 0.854</td>
</tr>
<tr>
<td>Data 9</td>
<td>'21 0.893</td>
</tr>
<tr>
<td>Data &amp;H10</td>
<td>'22 0.931</td>
</tr>
<tr>
<td>Data &amp;H10</td>
<td>'23 0.969</td>
</tr>
<tr>
<td>Data &amp;H10</td>
<td>'24 1.006</td>
</tr>
<tr>
<td>Data &amp;H11</td>
<td>'25 1.044</td>
</tr>
<tr>
<td>Data &amp;H11</td>
<td>'26 1.080</td>
</tr>
<tr>
<td>Data &amp;H11</td>
<td>'27 1.117</td>
</tr>
<tr>
<td>Data &amp;H12</td>
<td>'28 1.153</td>
</tr>
<tr>
<td>Data &amp;H12</td>
<td>'29 1.189</td>
</tr>
<tr>
<td>Data &amp;H12</td>
<td>'30 1.224</td>
</tr>
<tr>
<td>Data &amp;H13</td>
<td>'31 1.260</td>
</tr>
<tr>
<td>Data &amp;H13</td>
<td>'32 1.295</td>
</tr>
<tr>
<td>Data &amp;H13</td>
<td>'33 1.329</td>
</tr>
<tr>
<td>Data &amp;H14</td>
<td>'34 1.363</td>
</tr>
<tr>
<td>Data &amp;H14</td>
<td>'35 1.397</td>
</tr>
<tr>
<td>Data &amp;H14</td>
<td>'36 1.431</td>
</tr>
<tr>
<td>Data &amp;H15</td>
<td>'37 1.464</td>
</tr>
<tr>
<td>Data &amp;H15</td>
<td>'38 1.497</td>
</tr>
<tr>
<td>Data &amp;H15</td>
<td>'39 1.530</td>
</tr>
<tr>
<td>Data &amp;H16</td>
<td>'40 1.562</td>
</tr>
<tr>
<td>Data &amp;H16</td>
<td>'41 1.594</td>
</tr>
<tr>
<td>Data &amp;H16</td>
<td>'42 1.626</td>
</tr>
<tr>
<td>Data &amp;H17</td>
<td>'43 1.657</td>
</tr>
<tr>
<td>Data &amp;H17</td>
<td>'44 1.688</td>
</tr>
<tr>
<td>Data &amp;H17</td>
<td>'45 1.719</td>
</tr>
<tr>
<td>Data &amp;H18</td>
<td>'46 1.750</td>
</tr>
</tbody>
</table>
Data &H18  ' 47 1.780
Data &H18  ' 48 1.810
Data &H19  ' 49 1.840
Data &H19  ' 50 1.869
Data &H19  ' 51 1.898
Data &H19  ' 52 1.927
Data &H20  ' 53 1.956
Data &H20  ' 54 1.984
Data &H20  ' 55 2.012
Data &H21  ' 56 2.040
Data &H21  ' 57 2.068
Data &H21  ' 58 2.095
Data &H21  ' 59 2.122
Data &H22  ' 60 2.149
Data &H22  ' 61 2.176
Data &H22  ' 62 2.202
Data &H22  ' 63 2.228
Data &H23  ' 64 2.254
Data &H23  ' 65 2.279
Data &H23  ' 66 2.305
Data &H23  ' 67 2.330
Data &H24  ' 68 2.355
Data &H24  ' 69 2.379
Data &H24  ' 70 2.404
Data &H24  ' 71 2.428
Data &H25  ' 72 2.452
Data &H25  ' 73 2.476
Data &H25  ' 74 2.499
Data &H25  ' 75 2.523
Data &H26  ' 76 2.546
Data &H26  ' 77 2.569
Data &H26  ' 78 2.591

Data &H50  ' 79 5.000
Data &H49  ' 80 4.953
Data &H49  ' 81 4.907
Data &H48  ' 82 4.862
Data &H48  ' 83 4.816
Data &H47  ' 84 4.771
Data &H47  ' 85 4.727
Data &H47  ' 86 4.683
Data &H46  ' 87 4.639
Data &H46  ' 88 4.596
Data &H45  ' 89 4.553
Data &H45  ' 90 4.511
Data &H44  ' 91 4.469
Data &H44  ' 92 4.427
Data &H44  ' 93 4.386
Data &H43  ' 94 4.345
Data &H43  ' 95 4.304
Data &H42  ' 96 4.264
Data &H42  ' 97 4.224
Data &H42  ' 98 4.185
Data &H41  ' 99 4.146
Data &H41  '100 4.107
Data &H40  '101 4.069
Data &H40  '102 4.031
Data &H40  '103 3.994
Data &H39  '104 3.956
Data &H39  '105 3.920
Data &H39  '106 3.883
Data &H38  '107 3.847
Data &H38  '108 3.811
Data &H38  '109 3.776
Data &H37  '110 3.740
Data &H37  '111 3.705
Data &H37  '112 3.671
Data &H36  '113 3.637
Data &H36  '114 3.603
Data &H36  '115 3.569
Data &H35  '116 3.536
Data &H35  '117 3.503
Data &H35  '118 3.470
Data &H34  '119 3.438
Data &H34  '120 3.406
Data &H34  '121 3.374
Data &H33  '122 3.343
Data &H33  '123 3.312
Data &H33  '124 3.281
Data &H32  '125 3.250
Data &H32  '126 3.220
Data &H32  '127 3.190
Data &H31  '128 3.160
Data &H31  '129 3.131
Data &H31  '130 3.102
Data &H31  '131 3.073
Data &H30  '132 3.044
Data &H30  '133 3.016
6.98 GETRC

**Action**
Retrieves the value of a resistor or a capacitor.

**Syntax**

```plaintext
var = GETRC( pin )
```

**Remarks**

<table>
<thead>
<tr>
<th>var</th>
<th>The variable that receives the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>The port pin the R/C is connect to.</td>
</tr>
</tbody>
</table>

GETRC needs a resistor and capacitor in order to work. The capacitor is discharged and the charging time will vary depending on the user resistor/capacitor value.
Uses
This function uses TIMER0.

See also
NONE

Example
'-----------------------------------------------------------------------'  
'     GETRC.BAS'  Retrieve ... the time needed to discharge the capacitor'-----------------------------------------------------------------------'

Config
Timer0 = Timer,
Gate = Internal,
Mode = 1       'the GETRC() functions needs timer 0

Config
Getrc = 10                                             '10mS wait for charging the capacitor. This is the default so for 10 the
CONFIG is not needed

$baud = 9600                              'just my settings
$crystal = 11059200

Dim W As Word                                           'allocate space for variable

Do
'forever
    W = Getrc(p1.7)                         'get RC value
    Print W                                 'print it
    Wait 1                                  'wait a moment
Loop

'return values for cap=10nF .The resistor values where measured with a DVM
'     250 for 10K9
' 198 for 9K02
' 182 for 8K04
' 166 for 7K
' 154 for 6K02
' 138 for 5K04
' 122 for 4K04
' 106 for 3K06
'  86 for 2K16
'  54 for 1K00
'  22 for 198 ohm
'  18 for 150 ohm
'  10 for 104 ohm
'   6 for  1 ohm (minimum)

'As you can see there is a reasonable linearity
'So you can do some math to get the resistor value
'But the function is intended to serve as a rough indication for resistor values
'You can also change the capacitor to get larger values.
'With 10nF, the return value fits into a byte

6.99  GETRC5

Action
Retrieves a RC5 infrared code and sub address.

Syntax
GETRC5(address , command)

Remarks

<table>
<thead>
<tr>
<th>Address</th>
<th>The RC5 sub address received.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>The RC5 command received.</td>
</tr>
</tbody>
</table>

Use a Siemens infrared receiver SFH506-36 and connect it to port pin 3.2 to use this command.
This statement works together with the INT0 interrupt. See the example below on how to use it.
In version 2.09 the command returns the toggle bit in bit position 5 of the address.
You can clear it like: address = address AND &B0001_1111
The toggle bit will toggle after each key press of the remote control.
See Also
NONE

Example
'----------------------------------------------------------------------------
 RC5.BAS (c) 1995-2006 MCS Electronics
 ' connect SFH506-36 IR-receiver to PORT 3.2 (INT0)
 ' choose the correct port from the Compiler I2C TAB. Int0 should
 ' have P3.2 pin
 ' On other chips it may be another pin!
'----------------------------------------------------------------------------

Dim New As Bit
Dim Command As Byte , Subaddress As Byte

Reset Tcon.0
 'triggered by rising edge
On Int0 Receiverc5
Enable Int0
Enable Interrupts
Do
  If New = 1 Then
  'received new code
    Disable Int0
    Print Command ; " " ; Subaddress
    New = 0
  End If
End If
Loop

Receiverc5:
'interrupt routine
'the getrc5 routine uses 30 bytes ! of the stack for measuring
'the interval between the bits
Getrc5 (Subaddress,command)
    New = 1
    'set flag
Return

6.100 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.

See also
GOTO, CALL, RETURN

Example
GOSUB Routine
    'branch to routine
    'after being at 'routine' print this
    'terminate program
END

Routine:
    x = x + 2
    'perform some math
    PRINT X
    'print result
    RETURN
    'return
6.101 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.

**See also**
```
GOSUB
```

**Example**
```
Dim A As Byte

Start:                                    'a label must end with a
    A = A + 1                                 'increment a
If A < 10 Then
    Goto Start                               'is it less than 10?
End If
Print " Ready"                            'do it again
    'close IF
    'that is it
```

6.102 HEX

**Action**
Returns a string representation of a hexadecimal number.

**Syntax**
```
var = HEX(x)
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A numeric variable such as Byte, Integer or Word.</td>
</tr>
</tbody>
</table>

**See also**
```
HEXVAL, VAL, STR
```
Example
Dim A As Byte, S As String * 10
A = 123
S = Hex(a)
Print S
End

6.103 HEXVAL

Action
Convert string representing a hexadecimal number into a numeric variable.

Syntax
var = HEXVAL( x )

Remarks
<table>
<thead>
<tr>
<th>var</th>
<th>The numeric variable that must be assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The hexadecimal string that must be converted.</td>
</tr>
</tbody>
</table>

var : Byte, Integer, Word, Long.
x : String.

The string that must be converted must have a length of 2 bytes, 4 bytes or 8 bytes, for bytes, integers/words and longs respectively.

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.

See also
HEX, VAL, STR

Example
Dim A As Integer, S As String * 15
S = "000A"
A = Hexval(s) : Print A '10
End

6.104 HIGH

Action
Retrieves the most significant byte of a variable.
Syntax

\[ \text{var} = \text{HIGH}(\text{s}) \quad \text{high function gets the upper byte of a word} \]
\[ \text{HIGH}(\text{word}) = \text{byte} \quad \text{high statement set the upper byte of a word} \]

Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the MSB of var S.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The source variable to get the MSB from.</td>
</tr>
<tr>
<td>Word</td>
<td>A word or integer variable that is assigned</td>
</tr>
<tr>
<td>Byte</td>
<td>The value to set to the MSB of the Word/Integer variable</td>
</tr>
</tbody>
</table>

The \text{HIGH()} function returns the MSB of a variable while the \text{HIGH()} statement sets the MSB of a word variable.

See also

\text{LOW}, \text{LOWW}, \text{HIGHW}

Example

Dim I As Integer, Z As Byte
I = &H1001
Z = High(I) ' is 16

6.105 \text{HIGHW}

Action

Retrieves the two most significant bytes of a long.

Syntax

\[ \text{var} = \text{HIGHW}(\text{s}) \]

Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the two MSB of var S. It must be an Integer or Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The source variable to get the MSB from. Must be a long</td>
</tr>
</tbody>
</table>

See also

\text{LOW}, \text{HIGH}, \text{LOWW}

Example

Dim I As Long, Z As Word
I = &H10011001
6.106 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 upper line. You can also specify the first letter of the line like: HOME U

**See also**

CLS, LOCATE, LCD

**Example**

Lowerline
LCD " Hello"
Home Upper
LCD " Upper"

6.107 I2CRECEIVE

**Action**

Receives data from an I2C serial device.

**Syntax**

```
I2CRECEIVE  slave, var
I2CRECEIVE  slave, var ,b2W, b2R
```

**Remarks**

<table>
<thead>
<tr>
<th>slave</th>
<th>A byte, Word/Integer variable or constant with the slave address from the I2C-device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>A byte or integer/word variable that will receive the information from the I2C-device.</td>
</tr>
<tr>
<td>b2W</td>
<td>The number of bytes to write. Be cautious not to specify too many bytes!</td>
</tr>
<tr>
<td>b2R</td>
<td>The number of bytes to receive. Be cautious not to specify too many bytes!</td>
</tr>
</tbody>
</table>
In BASCOM LT you could specify DATA for var, but since arrays are supported now you can specify an array instead of DATA.

This command works only with some additional hardware. See appendix D.

See also

I2CSEND

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 String
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

6.108 I2CSEND

Action
Send data to an I2C-device.

Syntax

I2CSEND slave, var
I2CSEND slave, var , bytes

Remarks

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>slave</td>
<td>The slave address of the I2C-device.</td>
</tr>
<tr>
<td>var</td>
<td>A byte, integer/word or number that holds the value which will be sent to the I2C-device.</td>
</tr>
<tr>
<td>bytes</td>
<td>The number of bytes to send.</td>
</tr>
</tbody>
</table>

This command works only with additional hardware. See appendix D.

See also

I2CRECEIVE

Example

x = 5                                        'assign variable to 5
Dim ax(10) As Byte
slave = &H40                                'slave address of a PCF 8574 I/O IC
bytes = 1 'send 1 byte
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

6.109 I2C

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, 8|9
I2CWBYTE val

Remarks
<table>
<thead>
<tr>
<th>var</th>
<th>A variable that receives the value from the I2C-device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/9</td>
<td>Specify 8 or ACK if there are more bytes to read. (ACK)</td>
</tr>
<tr>
<td></td>
<td>Specify 9 or NACK if it is the last byte to read. (NACK)</td>
</tr>
<tr>
<td>val</td>
<td>A variable or constant to write to the I2C-device.</td>
</tr>
</tbody>
</table>

This command works only with additional hardware. See appendix D.

These functions are provided as an addition to the I2CSEND and I2CRECEIVE functions.

See also
I2CRECEIVE, I2CSEND

Example
'----- Writing and reading a byte to an EEPROM 2404
Dim A As Byte
Const Adresw = 174 'write of 2404
Const Adresr = 175 'read adres of 2404
I2cstart 'generate start
I2cwbyte Adresw 'send slaveadress
I2cwbyte 1 'send adres of EEPROM
I2cubyte 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
I2cubyte Adresw         'write slaveadres
I2cubyte 1               read
I2cstart                'generate repeated start
I2cubyte Adresr         'write slaveadres of EEPROM
I2crcbyte A, 9           'receive value into a. 9 means last byte to receive
I2cstop                'generate stop
Print A                 'print received value
End

6.110 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.

See also
POWERDOWN

Example
IDLE

6.111 IF

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. |

New is the ability to use the one line version of IF:

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

In V 2.00 support for variable bit index is added:

Dim Idx as Byte

For IDX = 0 To 7
    If P3.IDX = 1 Then
        Print "1" ;
    Else
        Print "0" ;
    End if
Next

A new feature in V2 is the ability to use multiple tests:

If a > 10 AND A < 10 OR A = 15 Then NOP End if

It does not work with strings but only numeric conditions.

When you want to test on bytes you can also use the string representation:

Dim X As Byte

If X = "A" then ' normally you need to write :

If X = 65 Then 'so these two lines do the same thing

**See also**

ELSE, END IF

**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 Labell Else Print "A<>10"
Labell:
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"

6.112 INCR

**Action**
Increments a variable by one.

**Syntax**

INCR var

**Remarks**

| Var     | Any numeric variable. |

There are often situations where you want a number to be increased by 1. The **INCR** statement is faster than var = var + 1.

**See also**

DECR

**Example**

Dim A As Integer
Do
    Incr A 'increment a by 1
    Print A 'print a
Loop Until A > 10 than 10

6.113 INKEY

**Action**

Returns the ASCII value of the first character in the serial input buffer.
Syntax
var = INKEY()
var = INKEY(#channel)

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>Byte, Integer, Word, Long or String variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>The channel number of device</td>
</tr>
</tbody>
</table>

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.
See the manual for a design of an RS-232 interface.
The RS-232 interface can be connected to a comport of your computer.

The INKEY() function only works with the hardware UART, not the software UART.

See also
WAITKEY

Example
Dim A As Byte
Do
    A = Inkey()  'look for character
    If A > 0 Then
        Print A    'yes, there was a character in the buffer
            'so print it
    End If
Loop    'loop forever

Example
$regfile = "80517.dat"
Open "COM2:" For Binary As #1  'open serial channel 1 on 80537
Dim St As Byte
St = Inkey(#1)   'get key from com2
If St > 0 Then
    Printbin #1, St  'send to com 2
End If
Close #1
6.114 INP

**Action**
Returns a byte read from a hardware port or external memory location.

**Syntax**
var = INP(address)

**Remarks**
- **var**
  Numeric variable that receives the value.
- **address**
  The address where to read the value from.

The INP statement only works on systems with an uP that can address external memory.

**See also**
OUT, PEEK, POKE

**Example**
```bascom
Dim a As Byte
a = INP(&H8000) 'read value that is placed on databus(d0-d7) at hex address 8000
PRINT a
END
```

6.115 INPUT

**Action**
Allows input from the keyboard during program execution.

**Syntax**
INPUT [" prompt"], var [, varn] [ NOECHO ] [ TIMEOUT = xx]

**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.
- **TIMEOUT**
  Optional delay time. When you specify the delay time, the routine will return when no input data is available after the specified time. No timer is used but a long is used to count down.

The INPUT routine can be used when you have a RS-232 interface on your uP. See the manual for a design of a RS-232 interface. 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. A backspace will remove the last entered character.

**Difference with QB**
In QB you can specify &H with INPUT so QB will recognize that a hexadecimal string is used. BASCOM implements a new statement: INPUTHEX.

**See also**
- INPUTHEX
- PRINT
- $TIMEOUT

**Example**
```bascom
'------------------------------------------------------------------------
'     (c) 1995-2006 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

'------------------------------------------------------------------------
'    When you need that the program times out on waiting for a
'    you need to use the TIMEOUT option.
'    When the charcter is not received within the specified time ERR
'    will be set to 1
'    otherwise ERR will be 0.
'    IMPORTANT : the TIMEOUT variable will use 4 bytes of internal
'    memory
'------------------------------------------------------------------------

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
```

© 2007 MCS Electronics
**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**
**echo** 'supress

**Input** "Enter your name " , S
**Print** "Hello " ; S

**Input** S **Noecho**
**echo** 'without

**Print** S

'unremark next line and remark all lines above for the TIMEOUT option
'this because when you use TIMEOUT once, you need to use it for all
**INPUT** statements

'**Input** "Name " , S **Timeout** = 0
'**Print** Err ; " " ; s

**End**

### 6.116 INPUTBIN

**Action**
Read binary values from the serial port.

**Syntax**

- **INPUTBIN** var1 [,var2]
- **INPUTBIN** #dev, var1 [,var2]

**Remarks**

<table>
<thead>
<tr>
<th>var1</th>
<th>The variable that is assigned with the characters from the serial port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var2</td>
<td>An optional second (or more) variable that is assigned with the</td>
</tr>
<tr>
<td></td>
<td>characters from the serial.</td>
</tr>
<tr>
<td>#dev</td>
<td>Device number. For use with OPEN and CLOSE. Dev is the device number.</td>
</tr>
</tbody>
</table>

The number of bytes to read is depending from 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.

**See also**

PRINTBIN, INPUT, INPUTHEX

**Example**

```
Dim a as Byte, C as Integer
INPUTBIN a, c                'wait for 3 characters
End

'This code only for 80517 and 80537 with dual serial port
Open "COM2:" For Binary As #1   'open serial channel 1
INPUTBIN #1, a
Close #1
```

### 6.117 INPUTHEX

**Action**

Allows input from the keyboard during program execution.

**Syntax**

```
INPUTHEX [" prompt"] , var [, varn] [ NOECHO ] [TIMEOUT=xx]
```

**Remarks**

<table>
<thead>
<tr>
<th>prompt</th>
<th>An optional string constant printed before the prompt character.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var,varn</td>
<td>A numeric variable to accept the input value.</td>
</tr>
<tr>
<td>NOECHO</td>
<td>Disables input echoed back to the Comport.</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Optional delay time. When you specify the delay time, the routine will return when no input data is available after the specified time. No timer is used but 4 bytes are taken from the internal memory to provide a count down timer.</td>
</tr>
</tbody>
</table>

When you use the TIMEOUT option once, you must use it for all INPUT/INPUTHEX statements. Providing zero as the timeout parameter will wait for the longest possible time.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP. See the manual for a design of a RS-232 interface.

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 must be 2 characters long.
If var is an integer/word then the input must be 4 characters long.
If var is a long then the input must be 8 characters long.
**Difference with QB**
In QB you can specify &H with INPUT so QB will recognize that a hexadecimal string is used.
BASCOM implement a new statement : INPUTHEX.

**See also**
INPUT | INPUTBIN | PRINTBIN

**Example**
Dim x As Byte
INPUTHEX " Enter a number ", x 'ask for input

### 6.118 INSTR

**Action**
Returns the position of a sub string in a string.

**Syntax**
```
var = INSTR( start , string , substr )
var = INSTR( string , substr )
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>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.</td>
</tr>
<tr>
<td>String</td>
<td>The string to search.</td>
</tr>
<tr>
<td>Substr</td>
<td>The search string.</td>
</tr>
</tbody>
</table>

At the moment INSTR() works only with internal strings. Support for external strings will be added too.

**Difference with QB**
No constants can be used for the string and sub string.

**See also**
None

**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

6.119 LCASE

Action
Converts a string into lower or upper case.

Syntax
dest = LCASE(source)

Remarks

<table>
<thead>
<tr>
<th>dest</th>
<th>The string variable that will be assigned with the lower case of string source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The source string. The original string will be unchanged.</td>
</tr>
</tbody>
</table>

See also
UCASE

Example

Dim S As String * 12, Z As String * 12
Input "Hello ", S  'assign string
S = Lcase(s)  'convert to lowercase
Print S  'print string
S = Ucase(s)  'convert to upper case
Print S  'print string

6.120 LCD

Action
Send constant or variable to LCD display.

Syntax
LCD x

Remarks
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.

See also

LCDHEX, $LCD, CONFIG LCD

Example

```bascom
'M-------------------------------------------
'   (c) 1995-2006 MCS Electronics
'M-------------------------------------------
' file: LCD.BAS
' demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFT_CURSOR, HOME
'       CURSOR, DISPLAY
'M-------------------------------------------
$sim
Rem The $sim statement will remove long delays for the simulator
Rem It is important to remove this statement when compiling the final file

'Config Lcdpin = Pin , Db4 = P3.1 , Db5 = P3.2 , Db6 = P3.3 , Db7 = P3.4 , E = P3.5 , Rs = P3.6
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 doesnt need the port pins!

Cls
Lcd "Hello world."  'clear the LCD display
Wait 1
Lowerline
Wait 1
Lcd "Shift this."  'display this at the top line
line
Wait 1
For A = 1 To 10
```
Shiftlcd Right  
Wait 1

Next

For A = 1 To 10
Shiftlcd Left  
Wait 1

Next

Locate 2 , 1  
Lcd "*"  
Wait 1

Shiftcursor Right  
Lcd "@"  
Wait 1

Home Upper  
Lcd "Replaced."  
Wait 1

Cursor Off Noblink  
Wait 1

Cursor On Blink  
Wait 1

Display Off  
Wait 1

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  
'first letter also works

Locate 4 , 1 : Lcd "Line 4"

Wait 1

'Now lets build a special character
'the first number is the character number (0-7)
'The other numbers are the row values
'Use the LCD tool to insert this line
Deflcdchar 0 , 31 , 17 , 17 , 17 , 17 , 17 , 31 , 0' replace ? with number (0-7)
Deflcdchar 1 , 16 , 16 , 16 , 16 , 16 , 16 , 16 , 31' 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 --------------
Acc = 1 'value into ACC
Call Write_lcd 'put it on LCD
End

6.121 LCDINIT

Action
Reinitialize the LCD display.

Syntax
LCDINIT

Remarks
When you use any of the LCD display routines the LCD display will be initialized automatically at startup of your program.
The LCD routines demand that the WR of the LCD display is connected to GND.
When in your design the WR pin of the LCD is connected to a PIN of the micro processor, it will be high during the initialization and so the display will not be initialized properly.
The LCDINIT routine allows you to perform initialization after you have set the pin that controls WR of the LCD to 0V.

See also
LCDHEX, $LCD, CONFIG LCD

Example
'---------------------------------------------------------------
' (c) 1995-2006 MCS Electronics
'---------------------------------------------------------------
' file: LCD.BAS
' demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
' CURSOR, DISPLAY
'---------------------------------------------------------------
$sim
Rem The $sim statement will remove long delays for the simulator
Rem It is important to remove this statement when compiling the final file
'Config Lcdpin = Pin , Db4 = P3.1 , Db5 = P3.2 , Db6 = P3.3 , Db7 = P3.4 , E = P3.5 , Rs = P3.6
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 doesnt need the port pins !

'----------------------- these 2 lines can be used when WR is connected to P1.0 for example ---
P1.0 = 0
INITLCD
'-----------------------------------------------
------------------------
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

6.122 LCDHEX

Action
Send variable in hexadecimal format to the LCD display.

Syntax
LCDHEX var

Remarks
var Variable to display.

The same rules apply as for PRINTHEX.

See also
Example
Dim a as byte
a = 255
LCD a
Lowerline
LCDHEX a
End

6.123 LEFT

Action
Return the specified number of leftmost characters in a string.

Syntax
var = LEFT(var1, n)

Remarks
<table>
<thead>
<tr>
<th>var</th>
<th>The string that is assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var1</td>
<td>The sourcestring.</td>
</tr>
<tr>
<td>n</td>
<td>The number of characters to get from the sourcestring.</td>
</tr>
</tbody>
</table>

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

For string operations, all the strings must be of the same type: internal or external.

See Also
RIGHT, MID

Example
Dim S As Xram String * 15, Z As Xram String * 15
S = "ABCDEFG"
Z = Left(s, 5)
Print Z 'ABCDE
End

6.124 LEN

Action
Returns the length of a string.

Syntax
var = LEN(string)
Remarks

<table>
<thead>
<tr>
<th>var</th>
<th>A numeric variable that is assigned with the length of string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>The string to calculate the length of.</td>
</tr>
</tbody>
</table>

Example

Dim S As String * 12
Dim A As Byte
S = "test"
A = Len(s)
Print A ' prints 4

6.125 LOAD

Action

Load specified TIMER with a value for auto reload mode.

Syntax

LOAD TIMER, value

Remarks

<table>
<thead>
<tr>
<th>TIMER</th>
<th>TIMER0, TIMER1 or TIMER2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The variable or value to load.</td>
</tr>
</tbody>
</table>

When you use the ON TIMERx statement with the TIMER/COUNTER in mode 2, you can specify on which interval the interrupt must occur. The value can range from 1 to 255 for TIMER0 and TIMER1. For TIMER2 the range is 1-65535.

The LOAD statement calculates the correct reload value out of the parameter. The formula: TLx = THx = (256-value)
For TIMER2: RCP2L = RCP2H = (65536 - value)

The load statement is not intended to assign/read a value to/from the timers/counters. Use COUNTER instead.

See Additional hardware for more details

Example

LOAD TIMER0, 100 'load TIMER0 with 100

Will generate:
Mov ti0,#h'9C
Mov th0,#h'9C

LOAD TIMER2, 1000
Will generate:
Mov RCAP2L,#24
Mov RCAP2H,#252

6.126 LOCATE

**Action**
Moves the LCD cursor to the specified position.

**Syntax**

```plaintext
LOCATE y, x
```

**Remarks**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Constant or variable with the position. (1-64*)</td>
</tr>
<tr>
<td>Y</td>
<td>Constant or variable with the line (1 - 4*)</td>
</tr>
</tbody>
</table>

* depending on the used display
For Graphical displays X can be in the range from 1-30 and y in the range from 1-8.

**See also**

- `CONFIG LCD`
- `LCD`
- `HOME`
- `CLS`

**Example**

```plaintext
LCD "Hello"
Locate 1,10
LCD "*
```

6.127 LOOKUP

**Action**
Returns a value from a table.

**Syntax**

```plaintext
var = LOOKUP( value, label )
```

**Remarks**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>The returned value</td>
</tr>
<tr>
<td>value</td>
<td>A value with the index of the table</td>
</tr>
<tr>
<td>label</td>
<td>The label where the data starts</td>
</tr>
</tbody>
</table>

**var:** Byte, Integer, Word, Long, Single.
**value:** Byte, Integer, Word, Long, Constant.
See also
LOOKUPSTR

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%

6.128 LOOKUPSTR

Action
Returns a string from a table.

Syntax
var = LOOKUPSTR( value, label [, language , length])

Remarks

<table>
<thead>
<tr>
<th>var</th>
<th>The string returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A value with the index of the table. The index is zero-based. That is, 0 will return the first element of the table.</td>
</tr>
<tr>
<td>label</td>
<td>The label where the data starts</td>
</tr>
<tr>
<td>language</td>
<td>An optional variable that holds a number to identify the language. The first language starts with the number 0.</td>
</tr>
<tr>
<td>length</td>
<td>The length of the data for each language.</td>
</tr>
</tbody>
</table>

value : Byte, Integer, Word, Long, Constant. Range(0-255)

See also
LOOKUP

Example
Dim S As String * 8 , Idx As Byte
Idx = 0 : S = Lookupstr(idx , Sdata)
Print S                                   'will print 'This'
End

Sdata:
Data "This" , "is" , "a test"

Example 2
Dim S As String * 8 , Idx As Byte , Language As Byte
Idx = 0 : Language = 1
S = Lookupstr(idx , Sdata , Language , 17)
Print S                                   ' will print 'Dit '
End

Sdata:
Data "This" , "is" , "a test "  'each language data must have the same length
Data "Dit " , "is" , "een test"  'the length is 17 because strings include a 0 byte

6.129 LOW

Action
Retrieves the least significant byte of a variable.

Syntax
var = LOW( s )

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the LSB of var S.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The source variable to get the LSB from.</td>
</tr>
</tbody>
</table>

See also
HIGH, LOWW, HIGHW

Example
Dim I As Integer , Z As Byte
I = &H1001
Z = Low(I) ' is 1
6.130 LOWW

**Action**
Retrieves the two least significant bytes of a long.

**Syntax**
```
var = LOWW(s)
```

**Remarks**
<table>
<thead>
<tr>
<th>var</th>
<th>The variable that is assigned with the two LSB of var S.</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The source variable to get the LSB’s from.</td>
</tr>
</tbody>
</table>

**See also**
HIGHW, HIGH, LOW

**Example**
Dim L As Integer, Z As Long
L = &H1001
Z = LowW(L)

6.131 LOWERLINE

**Action**
Reset the LCD cursor to the lower line.

**Syntax**
```
LOWERLINE
```

**Remarks**
None

**See also**
UPPERLINE, THIRDLINE, FOURTHLINE, HOME

**Example**
LCD "Test"
LOWERLINE
LCD "Hello"
End
6.132 MAKEBCD

**Action**
Convert a variable into its BCD value.

**Syntax**
var1 = MAKEBCD(var2)

**Remarks**
<table>
<thead>
<tr>
<th>var1</th>
<th>Variable that will be assigned with the converted value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var2</td>
<td>Variable that holds the decimal value.</td>
</tr>
</tbody>
</table>

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.

**See also**
MAKEDEC, BCD()

**Example**
Dim a As Byte
a = 65
LCD a
Lowerline
LCD BCD(a)
a = MakeBCD(a)
LCD " " ; a
End

6.133 MAKEDEC

**Action**
Convert a BCD byte or Integer/Word variable to its DECIMAL value.

**Syntax**
var1 = MAKEDEC(var2)

**Remarks**
<table>
<thead>
<tr>
<th>var1</th>
<th>Variable that will be assigned with the converted value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var2</td>
<td>Variable that holds the BCD value.</td>
</tr>
</tbody>
</table>

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.
See also
MAKEBCD , BCD

Example
Dim a As Byte
a = 65
LCD a
Lowerline
LCD BCD(a)
a = MakeDEC(a)
LCD " " ; a
End

6.134 MAKEINT

Action
Compacts 2 bytes into a word or integer.

Syntax
varn = MAKEINT(LSB , MSB)

Remarks
<table>
<thead>
<tr>
<th>Varn</th>
<th>Variable that will be assigned with the converted value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>Variable or constant with the Least Significant Byte.</td>
</tr>
<tr>
<td>MSB</td>
<td>Variable or constant with the Most Significant Byte.</td>
</tr>
</tbody>
</table>

The equivalent code is:
varn = (256 * MSB) + LSB

See also
MAKEDEC BCD

Example
Dim a As Integer , I As Integer
a = 2
I = MakeINT(a , 1) 'I = (1 * 256) + 2 = 258
End

6.135 MAX

Action
Returns the highest value of an array.

Syntax
var = MAX( ar(1) )
**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Numeric variable that will be assigned with the highest value of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar()</td>
<td>The first array element of the array to return the highest value of.</td>
</tr>
</tbody>
</table>

At the moment MAX() works only with BYTE arrays. Support for other data types will be added too.

**See also**

MIN , AVG

**Example**

Dim ar(10) As Byte  
Dim bP as Byte  
For bP = 1 to 10  
  ar(bP) = bP  
Next  
bP = Max(ar(1))  
Print bP 'should print 10

**6.136 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**

<table>
<thead>
<tr>
<th>Var</th>
<th>The string that is assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var1</td>
<td>The source string.</td>
</tr>
<tr>
<td>St</td>
<td>The starting position.</td>
</tr>
<tr>
<td>l</td>
<td>The number of characters to get/set.</td>
</tr>
</tbody>
</table>

Operations on strings require that all strings are of the same type (internal or external)

**See also**

LEFT , RIGHT

**Example**
Dim S As Xram String * 15, Z As Xram String * 15
S = "ABCDEFG"
Z = Mid(s, 2, 3)
Print Z  'BCD
Z = "12345"
Mid(s, 2, 2) = Z
Print S  'A12DEFG
End

6.137 MIN

Action
Returns the lowest value of an array.

Syntax
var = MIN(ar(1))

Remarks
<table>
<thead>
<tr>
<th>Var</th>
<th>Numeric variable that will be assigned with the lowest value of the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar()</td>
<td>The first array element of the array to return the lowest value of.</td>
</tr>
</tbody>
</table>

At the moment MIN() works only with BYTE arrays. Support for other data types will be added too.

See also
MAX, AVG

Example
Dim ar(10) As Byte
Dim bP as Byte
For bP = 1 to 10
  ar(bP) = bP
Next
bP = Min(ar(1))
Print bP 'should print 1
End

6.138 MOD

Action
Returns the remainder of a division.

Syntax
ret = var1 MOD var2
### Remarks

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ret</td>
<td>The variable that receives the remainder.</td>
</tr>
<tr>
<td>var1</td>
<td>The variable to divide.</td>
</tr>
<tr>
<td>var2</td>
<td>The divisor.</td>
</tr>
</tbody>
</table>

### Example

```fortran
a = 10 MOD 3        'divide 10 through 3
PRINT a                'print remainder (1)
```

#### 6.139 MWINIT

**Action**

Initializes the pins in order to use them with the micro wire statements.

**Syntax**

```fortran
MWINIT
```

**See also**

- CONFIG MICROWIRE
- MWREAD
- MWRITE
- MWWOPCODE

#### 6.140 MWREAD

**Action**

Read a value from the micro wire bus.

**Syntax**

```fortran
MWREAD variable, opcode, address, bytes
```

**Remarks**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>The variable that is assigned with the value retrieved from the micro wire bus.</td>
</tr>
<tr>
<td>Opcode</td>
<td>The opcode to use.</td>
</tr>
<tr>
<td>Address</td>
<td>The address of the device.</td>
</tr>
<tr>
<td>Bytes</td>
<td>Number of bytes to send.</td>
</tr>
</tbody>
</table>

**See also**

- MWRITE
- MWWOPCODE
- MWINIT

**Example**

```
'-----------------------------------------------------------------
' MicroWire test file
```

© 2007 MCS Electronics
please read microwire specs for understanding microwire

'CS - chip select
'DIN - data in
'DOUT - data Out
'CLOCK - Clock
'AL - address lines

93C46  93C56  93C57  93C66

Data bits:  8  16  8  16  8  16  8  16
AL:         7  6  9  8  8  7  9  8

you could use the same pin for DIN and DOUT
we use a 93C46 and send bytes not words so AL is 7
Config Microwire = Pin , Cs = P1.1 , Din = P1.2 , Dout = P1.4 , Clock = P1.5 , Al = 7

init pins
Mwinit

dimension variable used
Dim X As Byte

enable write to eeprom
send startbit, opcode (00) and 11 + address

Mwwopcode opcode, numberOfBits
Mwwopcode &B1001100000 , 10
the mwwopcode can send a command(opcode) to a device

X = 10
write value of X to address 0
opcode is 01
we write 1 byte
Mwwrite var,opcode,address,numberOfBytes
Mwwrite X , &B101 , 0 , 1

Waitms 10
X = 0
read back
Mwread var,opcode,address,numberOfBytes
Mwread X , &B110 , 0 , 1

disable write
send startbit, opcode (00) and 00 + address
Mwwopcode &B1000000000 , 10
End

6.141 MWWOPCODE

Action
Write an opcode to a micro wire device.

Syntax
MWWOPCODE opcode , bits
Remarks

<table>
<thead>
<tr>
<th>Opcode</th>
<th>The opcode that needs to be send to the micro wire device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>The number of bits to send.</td>
</tr>
</tbody>
</table>

Before you can work with micro wire you must send an opcode to enable writing an EEPROM for example.

See also

MWINIT, MWWRITE, MWREAD

Example

'enable write to EEPROM
'Needed bits : startbit (1), opcode (00) and (11) + address
'Mwwopcode opcode, numberOfBits
Mwwopcode &B1001100000 , 10 'send the code

6.142 MWWRITE

Action

Writes a value to the micro wire bus.

Syntax

MWWRITE variable , opcode , address, bytes

Remarks

<table>
<thead>
<tr>
<th>Variable</th>
<th>The variable which's content must be send to the micro wires device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>The opcode to use.</td>
</tr>
<tr>
<td>Address</td>
<td>The address of the device.</td>
</tr>
<tr>
<td>Bytes</td>
<td>Number of bytes to send.</td>
</tr>
</tbody>
</table>

See also

MWINIT, MWREAD, MWWPCODE

Example

'write value of X to address 0
'opcode is 01 and we write one byte
Mwrite X , &B101 , 0 , 1
6.143 NEXT

**Action**

Ends a FOR..NEXT structure.

**Syntax**

`NEXT [var]`

**Remarks**

| Var   | The index variable that is used as a counter when you form the structure with FOR var. Var is optional and not needed. |

You must end each FOR statement with a NEXT statement.

**See also**

FOR

---

6.144 ON interrupt

**Action**

Execute subroutine when specified interrupt occurs.

**Syntax**

`ON interrupt label [NOSAVE]`

**Remarks**

<table>
<thead>
<tr>
<th>interrupt</th>
<th>INT0, INT1, SERIAL, TIMER0, TIMER1 or TIMER2. Chip specific interrupts can be found under microprocessor support.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>The label to jump to if the interrupt occurs.</td>
</tr>
<tr>
<td>NOSAVE</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

You can't use TIMER1 when you are using SERIAL routines such as PRINT because TIMER1 is used as a BAUDRATE generator.

When you use the INT0 or INT1 interrupt you can specify on which condition the interrupt must be triggered.
You can use the Set/Reset statement in combination with the TCON-register for this purpose.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET TCON.0</td>
<td>trigger INT0 by falling edge.</td>
</tr>
<tr>
<td>RESET TCON.0</td>
<td>trigger INT0 by low level.</td>
</tr>
<tr>
<td>SET TCON.2</td>
<td>trigger INT1 by falling edge.</td>
</tr>
<tr>
<td>RESET TCON.2</td>
<td>trigger INT1 by low level.</td>
</tr>
</tbody>
</table>

See [Hardware](#) for more details

### See Also

**ON VALUE**

#### Example

```bascom
ENABLE INTERRUPTS
ENABLE INT0            'enable the interrupt
ON INT0 Label2 nosave  'jump to label2 on INT0
DO                     'endless loop
  LOOP
  END
LABEL2:                 
  PRINT " A hardware interrupt occurred!"   'print message
RETURN
```

### 6.145 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

<table>
<thead>
<tr>
<th>Var</th>
<th>The numeric variable to test. This can also be a SFR such as P1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>label1, label2</td>
<td>The labels to jump to depending on the value of var.</td>
</tr>
</tbody>
</table>

Note that the value is zero based. So when var = 0, the first specified label is jumped/branched.

### See Also
**ON interrupt**

**Example**

```bascom
Dim X As Byte

X = 2 'assign a variable interrupt
On X Gosub Lbl1, Lbl2, Lbl3 'jump to label lbl3
X = 0
On X Goto Lbl1, Lbl2, Lbl3
End

Lbl3:
    Print "lbl3"
Return

Lbl1:
    nop

Lbl2:
    nop

'nop is an ASM statement that does nothing
```

### OPEN

**Action**

Opens and closes a device.

**Syntax**

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

**Remarks**

<table>
<thead>
<tr>
<th>Device</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are 2 hardware devices supported: COM1 and COM2. With the software UART, you must specify the port pin and the baud rate. COM3.0:9600 will use PORT 3.0 at 9600 baud. Optional is <strong>INVERTED</strong> this will use inverted logic so you don't need MAX232 inverters.</td>
<td></td>
</tr>
</tbody>
</table>

| MODE | You can use BINARY, INPUT or OUTPUT for COM1 and COM2, 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. |
Since there are uP's such as the 80537 with 2 serial channels on board, the compiler must know which serial port you want to use. That is why the OPEN statement is implemented. With only 1 serial port on board, you don't need this statement. The statements that support the device are PRINT, PRINTHEX, INPUT and INPUTHEX.

Every opened device must be closed using the CLOSE #channel statement. Of course you must use the same channel number.

The software UART, only supports the GET and PUT statements to retrieve and send data and the PRINTBIN and INPUTBIN statement. The SW UART uses timed loops and interrupts can slow down these loops. So turn interrupts off before you use the SW UART.

COM1: and COM2: are hardware ports, and can be used with PRINT etc. For the software UART it is important that the pin you use is bit addressable. In most cases a PORT is bit addressable but some chips have ports that are not bit addressable. When you use such a port you will get errors like : Error 208, bit variable not found.

Since the OPEN statement doesn't use real file handles like DOS but only serves as a compiler directive, it is important that you must use the CLOSE statement as the last statement in your program.

The following example shows when it will NOT WORK:

```
OPEN "COM2:" FOR BINARY AS #1 'open the port
PRINT #1, "Hello" 'print to serial 1
Gosub Test
PRINT "Hello" 'print to serial 0
CLOSE #1

Test:
    Print #1, "test"
Return
```

Since the compiler frees the handle when it encounters the CLOSE statement, the PRINT #1, "test" code is never executed. To solve this you should put the CLOSE #1 statement under the Return statement.

```
OPEN "COM2:" FOR BINARY AS #1 'open the port
PRINT #1, "Hello" 'print to serial 1
Gosub Test
PRINT "Hello" 'print to serial 0

Test:
    Print #1, "test"
Return
Close #1
```

**See also**

GET, PUT

**Example 1**
'only works with a 80517 or 80537
CONFIG BAUD1 = 9600                        'serial 1 baudrate
OPEN "COM2:" FOR BINARY AS #1            'open the port
PRINT #1, "Hello"                        'print to serial 1
PRINT "Hello"                           'print to serial 0
CLOSE #1                                'close the channel

Example 2
'works with every port pin
Dim A As Byte , S As String * 16 , I As Byte , Dum As Byte
'a software comport is named after the pin you use
'for example P3.0 will be "COM3.0:" (so there is no P)
'for software comports, you must provide the baudrate
'So for 9600 baud, the devicename is "COM3.0:9600"
'When you want to use the pin for sending, you must open the device for OUTPUT
'When you want to use the pin for receiving, you must open the device for INPUT

'At this time only variables can be sent and received with the PUT and GET
'statesments.
'In the feature PRINT etc. will support these software comports.

Open "com3.1:9600" For Output As #1 ' p3.1 is normally used for tx so testing is easy
Open "com3.0:9600,INVERTED" For Input As #2 'p3.0 is normally used for RX
so testing is easy

S = "test this"                        'assign string
Dum = Len(s)                          'get length of string
For I = 1 To Dum                      'for all characters from left to right
    A = Mid(s , I , 1)                'get character
    Put #1, A                         'write it to comport
Next
Do
    Get #2 , A                      'get character from comport
    Put #1, A                       'write it back
    Print A                         'use normal channel
Loop
Close #1                               ' finally close device
Close #2
End

6.147 OUT

Action
Sends a byte to a hardware port or external memory address.

Syntax
OUT address, value
### Remarks

<table>
<thead>
<tr>
<th>address</th>
<th>The address where to send the byte to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The variable or value to send.</td>
</tr>
</tbody>
</table>

⚠️ The OUT statement only works on systems with a uP that can address external memory.

### See also

INP, PEEK, POKE

### Example

```bascom
Dim a as byte
OUT &H8000,1 'send 1 to the databus(d0-d7) at hex address 8000
END
```

Will generate:

```
Mov A,#1
Mov dptr,#h'8000
Movx @dptr,a
```

#### 6.148 PORT

### Action

P1 and P3 are special function registers that are treated as variables.

### Syntax

```
Px = var
var = Px
```

### Remarks

<table>
<thead>
<tr>
<th>X</th>
<th>The number of the port. (1 or 3). <strong>P3.6 can’t be used with an AT89C2051!</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>The variable to retrieve or to set.</td>
</tr>
</tbody>
</table>

Note that other processors can have more ports such as P0, P2, P4 etc. When you select the proper .DAT file you can also use these ports as variables. In fact you can use any SFR as a byte variable in BASCOM.

ACC = 0 'will reset the accumulator for example

See hardware for a more detailed description of the ports.

### Example

```bascom
Dim A As Byte, B1 As Bit
```
A = P1                                    'get value from port 1  
A = A or 2                                'manipulate it   
P1 = A                                    'set port 1 with new value   
P1 = &B10010101                              'use binary notation   
P1 = &HAF                                    'use hex notation    
B1 = P1.1                                 'read pin 1.1  
P1.1 = 0                                  'set it to 0  

6.149 PEEK

Action
Returns a byte stored in internal memory.

Syntax
var = PEEK( address )

Remarks

<table>
<thead>
<tr>
<th>var</th>
<th>Numeric variable that is assigned with the content of the memory location address</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Numeric variable or constant with the address location.(0-255)</td>
</tr>
</tbody>
</table>

See also
POKE, CPEEK, INP, OUT

Example
DIM a As Byte
a = Peek( 0 ) 'return the first byte of the internal memory (r0)
End

6.150 POKE

Action
Write a byte to an internal memory location.

Syntax
POKE address , value

Remarks

<table>
<thead>
<tr>
<th>address</th>
<th>Numeric variable with the address of the memory location to set. (0-255)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value to assign. (0-255)</td>
</tr>
</tbody>
</table>
Be careful with the POKE statement because you can change variables with it, which can cause your program to function incorrect.

**See also**
*PEEK*, *CPEEK*, *INP*, *OUT*

**Example**
```
POKE 127, 1                 'write 1 to address 127
End
```

### 6.151 POWERDOWN

**Action**
Put processor into power down mode.

**Syntax**
```
POWERDOWN
```

**Remarks**
The power down mode stops the system clock completely. The only way to reactivate the micro controller is by system reset.

**See also**
*IDLE*

**Example**
```
POWERDOWN
```

### 6.152 PRINT

**Action**
Send output to the RS-232 port.

**Syntax**
```
PRINT var ; " constant"
```

**Remarks**
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. See the manual for a design of an RS-232 interface.
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.

See also
PRINTHEX, INPUT, OPEN, CLOSE, SPC

Example
'--------------------------------------------------------------
' file: PRINT.BAS
' demo: PRINT, PRINTHEX
'--------------------------------------------------------------
Dim A As Byte, B1 As Byte, C As Integer
A = 1
Print "print variable a " ; A
Print
Print "Text to print."                  'constant to print

B1 = 10
PRINTHEX B1                           'print in hexa notation
C = &HA000                             'assign value to c%
PRINTHEX C                             'print in hex notation
Print C                                 'print in decimal notation

C = -32000                             
Print C
PRINTHEX C                             
Rem Note That Integers Range From -32767 To 32768
End

6.153 PRINTBIN

Action
Print binary content of a variable to the serial port.

Syntax
PRINTBIN var [,;varn]
PRINTBIN #dev, var ; [,;varn]

Remarks
var The variable which value is sent to the serial port.
Optional variables to send separated by a ;.
Device number for use with OPEN and CLOSE

PRINTBIN is equivalent to PRINT CHR(var); but whole arrays can be printed this way.

When you use a Long for example, 4 bytes are printed.

**See also**
- INPUTBIN
- PRINT
- PRINTHEX
- INPUTHEX

**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

'This code only for 80517/80537 with dual serial port
Open "COM2:" For Binary As #1  'open serial channel 1
PRINTBIN #1, a(1) ; a(2) ; a(3)  'note that the channel is separated by a , and the vars by ;
Close #1

**6.154 PRINTHEX**

**Action**
Sends a variable in hexadecimal format to the serial port.

**Syntax**

```
PRINTHEX var
```

**Remarks**

| var | The variable to print. |

The same rules apply to PRINTHEX as PRINT.

The PRINTHEX routine can be used when you have a RS-232 interface on your uP. See the manual for a design of an RS-232 interface. 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.

**See also**
- PRINT
- INPUTHEX
- SPC
Example
Dim x As Byte
INPUT x 'ask for var
PRINT x 'print it in decimal format
PRINTHEX "Hex " ; x 'print it in hex format

6.155 PRIORITY

Action
Sets the priority level of the interrupts.

Syntax
PRIORITY SET / RESET interrupt

Remarks

<table>
<thead>
<tr>
<th>SET</th>
<th>Bring the priority level of the interrupt to a higher level.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>Bring the priority level of the interrupt to a lower level.</td>
</tr>
<tr>
<td>RESET</td>
<td>The interrupt to set or reset.</td>
</tr>
</tbody>
</table>

The interrupts are: INT0, INT1, SERIAL, TIMER0, TIMER1 and TIMER2.

Interrupt INT0 always has the highest priority.
When more interrupts occur at the same time the following order is used to handle the interrupts.

Note that other microprocessors can have additional/other interrupt setting.
Read microprocessor support to check the additions.

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT0</td>
<td>1 (highest)</td>
</tr>
<tr>
<td>TIMER0</td>
<td>2</td>
</tr>
<tr>
<td>INT1</td>
<td>3</td>
</tr>
<tr>
<td>TIMER1</td>
<td>4</td>
</tr>
<tr>
<td>SERIAL</td>
<td>5 (lowest)</td>
</tr>
</tbody>
</table>

Example
PRIORITY SET SERIAL 'serial int highest level
ENABLE SERIAL 'enable serial int
ENABLE TIMER0 'enable timer0 int
ENABLE INTERRUPTS 'activate interrupt handler
ON SERIAL label 'branch to label if serial int occur
DO 'loop for ever
LOOP
Label:                           'start label
PRINT " Serial int occurred."       'print message
RETURN                            'return from interrupt

6.156 PSET

Action
Sets or resets a single pixel.

Syntax
PSET  X, Y, value

Remarks

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>The X location of the pixel. In range from 0-239.</td>
</tr>
<tr>
<td>Y</td>
<td>The Y location of the pixel. In range from 0-63.</td>
</tr>
<tr>
<td>value</td>
<td>The value for the pixel. 0 will clear the pixel. 1 Will set the pixel.</td>
</tr>
</tbody>
</table>

The PSET is handy to create a simple data logger or oscilloscope.

See also
CONFIG GRAPHLCD

Example
Dim X as Byte, Y as Byte
For X = 0 To 10
    For Y = 0 To 10
        Pset X, Y, 1                        'make a nice block
    Next
Next
End

6.157 PUT

Action
Sends a byte to the software UART.

Syntax
PUT  #channel , var

Remarks

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>Positive numeric constant that refers to the opened channel.</td>
</tr>
<tr>
<td>var</td>
<td>A variable or constant who's value is sent to the the software UART.</td>
</tr>
</tbody>
</table>
See also
GET, PRINT, INPUT, OPEN

Example
Open "com3.1:9600" For Output As #1 'p3.1 is normally used for
tx so testing is easy
Open "com3.0:9600" For Input As #2 'p3.0 is normally used for
RX so testing is easy

S = "test this" 'assign string
Dum = Len(s) 'get length of string
For I = 1 To Dum 'for all characters from left to right
    A = Mid(s, I, 1) 'get character
    Put #1, A 'write it to comport
Next

Do
    Get #2, A 'get character from comport
    Put #1, A 'write it back
    Print A 'use normal channel
Loop

Close #1 'finally close device
Close #2
End

6.158 READ

Action
Reads those values and assigns them to variables.

Syntax
READ var

Remarks

| var | Variable that is assigned data value. |

Difference with QB
It is important that the variable is of the same type as the stored data.

See also
DATA, RESTORE
Example
Dim A As Byte, I As Byte, C As Integer, S As XRAM 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"

6.159 READMAGCARD

Action
Reads data from a magnetic card reader.

Syntax
READMAGCARD var , bytes , code, timeout

Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>A byte array large enough to store the data from the magnetic card reader.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes</td>
<td>The number of bytes read from the card.</td>
</tr>
<tr>
<td>Shifts</td>
<td>The coding used. Must be 5 or 7. In version 2.03 only 5 is supported.</td>
</tr>
<tr>
<td>Timeout</td>
<td>A LONG variable or constant that the routine will wait for a card. Err will be set when no card is detected within Timeout.</td>
</tr>
</tbody>
</table>

There can be 3 tracks on a magnetic card.
Track 1 stores the data in 7 bit including the parity bit. This is handy to store alpha numeric data.
On track 2 and 3 the data is stored with 5 bit coding.
The ReadMagCard routine works with ISO7811-2 5 and 7 bit decoding.
The returned numbers for 5 bit coding are:

<table>
<thead>
<tr>
<th>Returned number</th>
<th>ISO characterT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
See also
None

Calls
_Read_Magcard_Code5

Example
'[DIM used variables]
Dim X(40) As Byte , I As Byte , Bts As Byte

'[ALIAS the pins used]
_mcs Alias P1.1
_mclock Alias P1.2
_mdata Alias P1.0

Do
    Print "Slide magcard through reader"
    Readmagcard X(1) , Bts , 5, 10000 'call routine
    ' ^ may be 5 or 7. 7 bit coding not implemented yet
    Print "Error " ; Err '1 if error occurred
    Print ; " " ; Bts ; " bytes read" 'show number of bytes read
    Print Err
    For I = 1 To Bts
        Print X(I) ; " "; 'show number
    Next
    Print
Loop
End

6.160 REM

**Action**
Instruct the compiler that comment will follow.

**Syntax**
REM or '

**Remarks**
You can comment your program for clarity.
You can use REM or ' followed by your comment.
All statements after REM or ' are treated as comment so you cannot use statements after a REM statement.

It is also possible to use block comments:
'(~ start block comment
print "This will not be compiled
'~ end block comment

Note that the starting ' sign will ensure compatibility with QB
Each block must be closed with a ')

**Example**
REM TEST.BAS version 1.00
PRINT a " this is comment : PRINT " hello"
     ^--- this will not be executed!

6.161 REPLACE

**Action**
Replace all occurrences of a single character in a string.

**Syntax**
REPLACE string , old , new

**Remarks**
<table>
<thead>
<tr>
<th>string</th>
<th>The source string to change.</th>
</tr>
</thead>
<tbody>
<tr>
<td>old</td>
<td>A string constant or byte that specifies the character to replace.</td>
</tr>
<tr>
<td>new</td>
<td>The new character. Also a string constant or a byte.</td>
</tr>
</tbody>
</table>

**Example**
Dim S as String * 12
s = "Hello"
REPLACE s , "e" , "a"        ' now we got some dutch :-) 
Print s                        ' should print Hallo

6.162  RESET

Action
Reset a bit of a PORT (P1.x, P3.x) or an internal bit/byte/integer/word/long variable.

Syntax
RESET bit
RESET var.x

Remarks
| bit   | Can be a P1.x, P3.x or any bitvariable where x=0-7.
| var   | Can be a byte, integer or word variable.
| x     | Constant of variable to reset.(0-7) for bytes and (0-15) for Integer/Word. 0-31 for a LONG.

See also
SET

Example
Dim b1 as bit, b2 as byte, I as Integer
RESET P1.3       'reset bit 3 of port 1
RESET b1                   'bitvariable
RESET b2.0       'reset bit 0 of bytevariable b2
RESET I.15       'reset MS bit from I

6.163  RESTORE

Action
Allows READ to reread values in specified DATA statements.

Syntax
RESTORE label

Remarks
| Label | The label of a DATA statement.

See also
DATA, READ
**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)

### 6.164 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.

**See also**

GOSUB

**Example**

```
Dim Result As Byte, Y As Byte
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
```
6.165 RIGHT

**Action**
Return a specified number of rightmost characters in a string.

**Syntax**

```
var = RIGHT(var1, st)
```

**Remarks**

<table>
<thead>
<tr>
<th>var</th>
<th>The string that is assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var1</td>
<td>The source string.</td>
</tr>
<tr>
<td>st</td>
<td>The starting position.</td>
</tr>
</tbody>
</table>

All strings must be of the same data type, internal or external.

**See also**

LEFT, MID

**Example**

```
Dim s As XRAM String * 15, z As XRAM String * 15
s = "ABCDEFG"
z = Right(s, 2)
Print z                                 'FG
End
```

6.166 RND

**Action**
Returns a random number.

**Syntax**

```
var = RND(limit)
```

**Remarks**

<table>
<thead>
<tr>
<th>Limit</th>
<th>The maximum number that will be assigned to the random number.</th>
</tr>
</thead>
</table>

The RND() function uses 2 internal bytes to store the value of the random seed.

⚠️ It is important to understand that the RND() function is a math function. Every time you reset the micro, it will produce the same sequence. Only when you vary the variables with for example a timer, temperature reading, or a clock, you can make a more random value.

**See also**
Example
'---------------------------------------------------------'                     (c) 1995-2006 MCS Electronics'            ... W As WordDo  'get a random number and limit it to be maximum 100  W = Rnd(100)  Print WLoopEnd

6.167 ROTATE

Action
Shifts all bits one place to the left or right.

Syntax
ROTATE var , LEFT/RIGHT [ , shifts]

Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>Byte, Integer/Word or Long variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shifts</td>
<td>The number of shifts to perform.</td>
</tr>
</tbody>
</table>

Note that the behavior of ROTATE is just like the ASM RL or RR mnemonic. It works for integer, words, single and longs also. All bits in the variable are preserved so for a byte after 8 rotations, the value will be the same.

See also
SHIFTIN\[\], SHIFTOUT\[\], SHIFT\[\]

Calls
 Rotate_LEFT or _ROTATE_RIGHT

Example
Dim a as Byte
a = 128
ROTATE a, LEFT , 2
Print a                '1
6.168 SELECT

**Action**
Executes one of several statement blocks depending on the value of a variable.

**Syntax**
SELECT CASE var
  CASE test1 : statements
  [CASE test2 : statements ]
  CASE ELSE : statements
END SELECT

**Remarks**
<table>
<thead>
<tr>
<th>var</th>
<th>Variable to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>Value to test for.</td>
</tr>
<tr>
<td>Test2</td>
<td>Value to test for.</td>
</tr>
</tbody>
</table>

**See also**
IF THEN

**Example**
Dim b2 as byte
SELECT CASE b2
  CASE 2 : PRINT "2"
  CASE 4 : PRINT "4"
  CASE IS >5 : PRINT ">5" 'a test requires the IS keyword
  CASE 10 TO 20 : PRINT "range 10 to 20"
  CASE ELSE
END SELECT
END

6.169 SET

**Action**
Set a bit of a PORT(P1.x,P3.x) or a bit/byte/integer/word/long variable.

**Syntax**
SET bit
SET var.x

**Remarks**
<table>
<thead>
<tr>
<th>Bit</th>
<th>P1.x, P3.x or a Bit variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>A byte, integer, word or long variable.</td>
</tr>
</tbody>
</table>
### 6.170 SHIFTCURSOR

**Action**
Shift the cursor of the LCD display left or right by one position.

**Syntax**

```
SHIFTCURSOR LEFT | RIGHT
```

**See also**

SHIFTLCD, LCD, CLS, LOCATE, HOME

**Example**

```
LCD "Hello"
SHIFTCURSOR LEFT
End
```

### 6.171 SHIFT

**Action**
Shifts all bits one place to the left or right.

**Syntax**

```
SHIFT var , LEFT/RIGHT [ , shifts]
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>Byte, Integer/Word or Long variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shifts</td>
<td>The number of shifts to perform.</td>
</tr>
</tbody>
</table>

The SHIFT statements shifts all bits to the left or right and so for a byte after 8 shifts, the byte will be zero.

**See also**

RESET
See also
SHIFTIN, SHIFTOUT, ROTATE

Example
Dim a as Word
a = 128
SHIFT a, LEFT, 1
Print a '256

6.172 SHIFTIN

Action
Shifts a bit stream in or out a variable.

Syntax
SHIFTIN pin, pclock, var, option [PRE]
SHIFTOUT pin, pclock, var, option

Remarks

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>The portpin which serves as input/output.</td>
</tr>
<tr>
<td>pclock</td>
<td>The portpin which generates the clock.</td>
</tr>
<tr>
<td>var</td>
<td>The variable that is assigned.</td>
</tr>
</tbody>
</table>
| 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 |
|   | For the SHIFTIN statement you can add 4 to the parameter to use the external clock signal for shifting. |
| PRE | Add this additional parameter (no comma) to sample the input pin before the clock signal is generated. |

It depends on the type of the variable, how many shifts will occur. When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur.

See also
NONE

Example
Dim a as byte
SHIFTIN P1.0, P1.1, a, 0
SHIFTOUT P1.2, P1.1, a, 0
For the SHIFTIN example the following code is generated:
Setb P1.1
Mov R0,#h'21
Mov r2,#h'01
__UNQLBL1:
Mov r3,#8
__UNQLBL2:
Clr P1.1
Nop
Nop
Mov c,P1.0
Rlc a
Setb P1.1
Nop
Nop
Djnz r3,__UNQLBL2
Mov @r0,a
Dec r0
Djnz r2,__UNQLBL1

Of course, it depends on the parameter, which code will be generated. To shift with an external clock signal:
SHIFTIN P1.0, P1.1 , a , 4 'add 4 for external clock

Generated code:
Mov R0,#h'21
Mov r2,#h'01
__UNQLBL1:
Mov r3,#8
__UNQLBL2:
Jnb P1.1,*+0
Mov c,P1.0
Rlc a
Jb P1.1,*+0
Djnz r3,__UNQLBL2
Mov @r0,a
Dec r0
Djnz r2,__UNQLBL1

6.173 SHIFTLCD

Action
Shift the LCD display left or right by one position.

Syntax
SHIFTLCD LEFT / RIGHT

Remarks
NONE
See also

SHIFTCURSOR, CLS, LCD, HOME, LOCATE

Example

LCD "Very long text"
SHIFTLCD LEFT
Wait 1
SHIFTLCD RIGHT
End

6.174 SHOWPIC

Action
Shows a BGF file on the graphic display

Syntax

SHOWPIC x, y, label

Remarks

Showpic can display a converted BMP file. The BMP must be converted into a BGF file with the Tools Graphic Converter. The X and Y parameters specify where the picture must be displayed. X and Y must be 0 or a multiple of 8. The picture height and width must also be an multiple of 8. The label tells the compiler where the graphic data is located. It points to a label where you put the graphic data with the $BGF directive.

See also

$BGF, CONFIG GRAPHLCD, PSET

Example

CLS GRAPH
ShowPic 0,0, label
End

Label:
$BGF "mypic.bgf"

6.175 SOUND

Action
Sends pulses to a port pin.

Syntax

SOUND pin, duration, frequency [NOINT]
Remarks

<table>
<thead>
<tr>
<th>Pin</th>
<th>Any I/O pin such as P1.0 etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>The number of pulses to send. Byte, integer/word or constant. (1-32768).</td>
</tr>
<tr>
<td>Frequency</td>
<td>The time the pin is pulled low and high.</td>
</tr>
<tr>
<td>NOINT</td>
<td>An option to disable interrupts during the sound statement.</td>
</tr>
</tbody>
</table>

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 NOINT will clear the global interrupts so no interrupts can occur during the sound statement. When the sound statement has completed the interrupt register is restored.

The port pin is switched high and low for frequency µS. The pin will be in the low state when the sound statement ends. This loop is executed duration times.

See also
SOUNDEXT

Example

SOUND P1.1, 10000, 10                'BEEP
End

6.176 SOUNDEXT

Action

Sends pulses to a port pin.

Syntax

SOUND pin, duration, frequency [,NOINT]

Remarks

<table>
<thead>
<tr>
<th>Pin</th>
<th>Any I/O pin such as P1.0 etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>This is an integer, word or constant that specifies how long the sound is generated. A bigger value will result in a longer duration of the sound.</td>
</tr>
<tr>
<td>Frequency</td>
<td>This is an integer, word, or constant that that will be used to generate the frequency. A higher value will result in a higher frequency. A very low value might result in a sound that can not be heard.</td>
</tr>
<tr>
<td>NOINT</td>
<td>An option to disable interrupts during the sound statement.</td>
</tr>
</tbody>
</table>

The SOUNDEXT should be used instead of the SOUND statement. It has a wider range.
When you connect a speaker or a buzzer to a port pin (see hardware), you can use the SOUNDEXT statement to generate some tones. The NOINT will clear the global interrupts so no interrupts can occur during the sound statement. When the sound statement has completed the interrupt register is restored.

The SoundExt routine will create the sound as following:
- The port pin is set LOW
- The specified frequency is inverted
- The inverted value is decreased
- The port pin is set HIGH
- The inverted value is restored and decreased again

The actions are executed for DURATION times.

When the statement is ready, it will leave the pin in the HIGH state. The time the pin is low is exact the same time as the pin is high. So the created pulse width is 50%.

Since loops are used, the frequency is relatively to the processor speed. The width range of the frequency will ensure that you can create hearable tones width a variety of oscillator values. When you want to create tones that are independent of the processor speed, you need to use a timer.

See also
SOUND

Example
$regfile = "89s4051.dat"
$crystal = 8000000
Dim X As Word
X = 0
Do
   X = X - 10
   Soundext P3.4, 500, X
Waitms 1
Loop

6.177 SPACE

Action
Returns a string of spaces.

Syntax
var = SPACE(x)

Remarks
<table>
<thead>
<tr>
<th>X</th>
<th>The number of spaces.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var</td>
<td>The string that is assigned.</td>
</tr>
</tbody>
</table>

Using 0 for x, will result in a string of 255 bytes because there is no check for a zero
length assign.

See also
STRING, SPC

Example
Dim s as XRAM String * 15, z as XRAM String * 15
s = Space(5)
Print " {" ;s ; " }"
Dim A as Byte
A = 3
S = Space(a)

Generated code for last 2 lines:
; ----------- library routine -----------
_SStr_String:
Mov @r1,a
Inc r1
Djnz r2, _SStr_String
Clr a
Mov @r1,a
Ret
; -------------------------------
Mov R1,#h'22 ; location of string
Mov R2, #h'21 ; number of spaces
Mov a,#32
Acall _SStr_String

6.178 SPC

Action
Prints spaces to the serial port or LCD display.

Syntax
PRINT SPC(x)

Remarks
x The number of spaces to print. Range from 1 - 255.

Use SPACE() function to assign spaces to a string.
SPC() can only be used in combination with PRINT and LCD.

See Also
SPACE

Example
Dim s as XRAM String * 15, z as XRAM String * 15
s = "Hello"
Print " {" ; s ; SPC(3) ; "}"

### 6.179 SPIIN

**Action**
Reads a value from the SPI-bus.

**Syntax**

```
SPIIN var, bytes
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the value read from the SPI-bus.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>The number of bytes to read.</td>
</tr>
</tbody>
</table>

**See also**
- SPIOUT
- CONFIG SPI
- SPIINIT

**Example**

```
Dim a(10) as byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SPIIN
SPIIN a(1) , 4                'read 4 bytes
```

### 6.180 SPIINIT

**Action**
Initializes the pins of the SPI-bus.

**Syntax**

```
SPIINIT
```

**Remarks**

The pins used for the SPI bus must be set to the proper logical level before you can use the SPI commands.

**See also**
- SPIOUT
- CONFIG SPI
- SPIIN

**Example**

```
Dim a(10) as byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SPIINIT
```
SPIIN a(1), 4                    'read 4 bytes

6.181 SPIOUT

Action
Sends a value of a variable to the SPI-bus.

Syntax
SPIOUT var , bytes

Remarks
<table>
<thead>
<tr>
<th>var</th>
<th>The variable's content must be send to the SPI-bus.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytes</td>
<td>The number of bytes to send.</td>
</tr>
</tbody>
</table>

See also
SPIIN, CONFIG SPI, SPIINIT

Example
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SPIINIT                              ' init SPI pins
Dim a(10) as Byte , X As Byte
SPIOUT a(1) , 5                   'send 5 bytes
SPIOUT X , 1                      'send 1 byte

6.182 START

Action
Start the specified timer/counter.

Syntax
START timer

Remarks
| timer | TIMER0, TIMER1, TIMER2, COUNTER0 or COUNTER1. |

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.

See also
STOP TIMERx
Example
ON TIMER0 label2
LOAD TIMER0, 100
START TIMER0
DO 'start loop
LOOP 'loop forever
label2: 'perform an action here
RETURN

6.183 STOP

Action
Stop program execution.

Syntax
STOP

Remarks
END can also be used to terminate a program.

When an END or STOP statement is encountered a never ending loop is generated.

See Also
STOP TIMER, START

Example
PRINT var 'print something
STOP 'that's it

6.184 STOP Timer

Action
Stop the specified timer/counter.

Syntax
STOP timer

Remarks
timer | TIMER0, TIMER1, TIMER2, COUNTER0 or COUNTER1.

You can stop a timer when you don't want an interrupt to occur.
TIMER0 and COUNTER0 are the same.

See also
START TIMERx, STOP

Example
'--------------------------------------------------------------'
'(c) 1995-2006 MCS Electronics
'--------------------------------------------------------------'
' file: TIMER0.BAS
' demo: ON TIMER0
' *TIMER1 is used for RS-232 baudrate generator
'--------------------------------------------------------------'
Dim Count As Byte, Gt As Byte

Config Timer0 = Timer , Gate = Internal , Mode = 2
'Timer0 = counter : timer0 operates as a counter
'Gate = Internal : no external gate control
'Mode = 2 : 8-bit auto reload (default)

On Timer0 Timer_0_int
Load Timer0 , 100               'when the timer reaches 100 an
interrupt will occur
Enable Interrupts                'enable the use of interrupts
Enable Timer0                     'enable the timer

Rem Setting Of Priority
Priority Set Timer0              'highest priority
Start Timer0                     'start the timer

Count = 0                                      'reset counter
Do
Input "Number " , Gt
Print "You entered : " ; Gt
Loop Until Gt = 1
until users enters 1
Stop Timer0
End

Rem The Interrupt Handler For The Timer0 Interrupt
Timer_0_int:
   Inc Count
   If Count = 250 Then
      Print "Timer0 Interrupt occured"
      Count = 0
   End If
6.185 STR

**Action**
Returns a string representation of a number.

**Syntax**

```plaintext
var = STR( x )
```

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>A string variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A numeric variable.</td>
</tr>
</tbody>
</table>

The string must be big enough to store the string.

**See also**

- VAL
- HEX
- HEXVAL

**Difference with QB**

In QB STR() returns a string with a leading space. This behaviour is not in BASCOM.

**Example**

```plaintext
Dim a as Byte, S as XRAM String * 10
a = 123
s = Str(a)
Print s
End
```

6.186 STRING

**Action**

Returns a string consisting of m repetitions of the character with ASCII code n.

**Syntax**

```plaintext
var = STRING(m ,n )
```
Remarks

<table>
<thead>
<tr>
<th>Var</th>
<th>The string that is assigned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>The ASCII-code that is assigned to the string.</td>
</tr>
<tr>
<td>M</td>
<td>The number of characters to assign.</td>
</tr>
</tbody>
</table>

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. When you need this let me know.

See also

SPACE

Example

Dim s as XRAM String * 15
s = String(5,65)
Print s                'AAAAA
End

6.187 SUB

Action

Defines a Sub procedure.

Syntax

```
SUB Name[(var1)]
```

Remarks

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the sub procedure, can be any non reserved word.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1</td>
<td>The name of the parameter.</td>
</tr>
</tbody>
</table>

You must end each subroutine with the END SUB statement.

You must Declare Sub procedures before the SUB statement.
The parameter names and types must be the same in both the declaration and the Sub procedure.

Parameters are global to the application.
That is the used parameters must be dimensioned with the DIM statement.
Therefore, the variables can be used by the program and sub procedures.
The following examples will illustrate this:

```
Dim a as byte, b1 as byte, c as byte    'dim used variables
Declare Sub Test(a as byte)             'declare subroutine
a = 1 : b1 = 2: c = 3                   'assign variables
Print a ; b1 ; c                        'print them
```
See also
CALL, DECLARE

Example
NONE

6.188 SWAP

Action
Exchange two variables of the same type.

Syntax
SWAP var1, var2

Remarks

<table>
<thead>
<tr>
<th>var1</th>
<th>A variable of type bit, byte, integer or word.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var2</td>
<td>A variable of the same type as var1.</td>
</tr>
</tbody>
</table>

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

6.189 THIRDLINE

Action
Reset LCD cursor to the third line.

Syntax
THIRDLINE

Remarks
See also
UPPERLINE, LOWERLINE, FOURTHLINE

Example
Dim a as byte
a = 255
LCD a
Thirdline
LCD a
Upperline
End

6.190 UCASE

Action
Converts a string into upper case.

Syntax
dest = \textbf{UCASE}( source )

Remarks
\begin{tabular}{|l|p{14cm}|}
\hline
\textbf{dest} & The string variable that will be assigned with the upper case of string \textbf{SOURCE}. \\
\hline
\textbf{source} & The source string. The original string will be unchanged. \\
\hline
\end{tabular}

See also
\textbf{LCASE}

Example
\texttt{Dim S As String * 12 , Z As String * 12}
\texttt{Input "Hello " , S} \quad \texttt{\textasciitilde assign string}
\texttt{S = Lcase(s)} \quad \texttt{\textasciitilde convert to lowercase}
\texttt{Print S} \quad \texttt{\textasciitilde print string}

\texttt{S = Ucase(s)} \quad \texttt{\textasciitilde convert to upper case}
\texttt{Print S} \quad \texttt{\textasciitilde print string}
6.191 UPPERLINE

**Action**
Reset LCD cursor to the upper line.

**Syntax**
UPPERLINE

**Remarks**
NONE

**See also**
LOWERLINE, THIRDLINE, FOURTHLINE

**Example**
Dim a as byte
a = 255
LCD a
Lowerline
LCD a
Upperline
End

6.192 VAL

**Action**
Converts a string representation of a number into a number.

**Syntax**
var = Val( s )

**Remarks**

<table>
<thead>
<tr>
<th>Var</th>
<th>A numeric variable that is assigned with the value of s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Variable of the string type.</td>
</tr>
</tbody>
</table>


**See also**
STR, HEXVAL

**Example**
Dim a as byte, s As XRAM string * 10
s = "123"
a = Val(s) 'convert string
6.193 VARPTR

**Action**
Retrieves the memory-address of a variable.

**Syntax**
```
var = VARPTR( var2 )
```

**Remarks**
<table>
<thead>
<tr>
<th>Var</th>
<th>The variable that is assigned with the address of var2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>var2</td>
<td>A variable to retrieve the address from.</td>
</tr>
</tbody>
</table>

**See also**
PEEK, POKE

**Example**
```
Dim I As Integer, B1 As Byte
B1 = Varptr(I)
```

Generated code:
```
Mov h'23, #h'21
```

6.194 WAIT

**Action**
Suspends program execution for a given time.

**Syntax**
```
WAIT seconds
```

**Remarks**
<table>
<thead>
<tr>
<th>seconds</th>
<th>The number of seconds to wait.</th>
</tr>
</thead>
</table>

The delay time is based on the used X-tal (frequency). When you use interrupts the delay can be extended.

**See also**
DELAY, WAITMS, WAITMSE
Example
WAIT 3                'wait for three seconds
Print "*"

6.195 WAITKEY

Action
Wait until a character is received in the serial buffer.

Syntax
var = WAITKEY()
var = WAITKEY(#channel)

Remarks
Var
Variable that is assigned with the ASCII value of the serial buffer.

channel
The channel number of the device


See also
INKEY

Example
Dim A As Byte
A = Waitkey                'wait for character
Print A

Example
Dim A As Byte
Open "COM2:" For Binary As #1 'open serial chan.1 COM2 of 80517/80537
Dim St As Byte
St = Inkey(#1)              'get key
St = Inkey()                        'get key from COM1 (the default)

6.196 WAITMS

Action
Suspends program execution for a given time in mS.

Syntax
WAITMS mS

Remarks
mS
The number of milliseconds to wait. (1-255)
The delay time is based on the used X-tal (frequency). The use of interrupts can slow down 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.

See also
DELAY, WAIT, WAITMSE

Example
WAITMS 10
Print "**"

6.197 WAITMSE

Action
Suspends program execution for a given time in mS.

Syntax
WAITMS mS

Remarks

The delay time is based on the used X-tal (frequency). So it is important that you provide the right $CRYSTAL value.

The use of interrupts can slow down this routine. For a real precise delay you should use a timer. The WAITMS statement can only delay for 255 mS. That is why the WAITMSE statement was added, it can give a longer delay.

See also
DELAY, WAIT, WAITMS

Example
WAITMSE 1000
Print "**"

6.198 WATCHDOG

Action
Start and stop the watchdog timer.

Syntax
**START WATCHDOG** 'will start the watchdog timer.
**STOP WATCHDOG** 'will stop the watchdog timer.
**RESET WATCHDOG** 'will reset the watchdog timer.

**Remarks**
The AT89S8252 has a built in watchdog timer. A watchdog timer is a timer that will reset the uP when it reaches a certain value. So during program execution this WD-timer must be reset before it exceeds its maximum value. This is used to be sure a program is running correct. When a program crashes or sits in an endless loop it will not reset the WD-timer so an automatic reset will occur resulting in a restart. You need to configure the reset time with CONFIG WATCHDOG.

**CONFIG WATCHDOG** = value

| value | The time in mS it takes the WD will overflow, causing a reset. Possible values are: 16,32,64,128,256,512,1024 or 2048 |

**See Also**
CONFIG WATCHDOG

**Example**
DIM A AS INTEGER
CONFIG WATCHDOG = 2048 'after 2 seconds a reset will occur
START WATCHDOG 'start the WD
DO
  PRINT a
  a = a + 1 'notice the reset
  REM RESET WATCHDOG 'delete the REM to run properly
LOOP
END

6.199 **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. BASCOM then returns to the WHILE statement and checks condition. If it is still true, the process is repeated. If it is not true, execution resumes with the statement following the WEND
statement.

See also

*DO .. LOOP*, *FOR .. NEXT*

Example

```bascom
Dim A As Byte
While A <= 10
  Print A
  Incr A
Wend
```
7 Using assembly

7.1 Using assembly

**In line assembly**

Assembler statements are recognized by the compiler. The only exception is SWAP because this is a valid BASIC statement. You must precede this ASM-statement with the !-sign so the compiler knows that you mean the ASM SWAP statement.

Note that for the ACC register, A is used in mnemonics. (Except for bit operations)

Example:

- Mov a, #10 'ok
- Mov acc,#10 'also ok but generates 1 more byte
- Setb acc.0 'ok
- Setb a.0 'NOT OK

You can also include an assembler file with the `$INCLUDE FILE.ASM` statement.

The assembler is based on the standard Intel mnemonics. The following codes are used to describe the mnemonics:

<table>
<thead>
<tr>
<th>Rn</th>
<th>working register R0-R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>128 internal RAM locations, any IO port, control or status register. For example: P1, P3, ACC</td>
</tr>
<tr>
<td>@Ri</td>
<td>indirect internal RAM location addressed by register R0 or R1</td>
</tr>
<tr>
<td>#data</td>
<td>8-bit constant included in instruction</td>
</tr>
<tr>
<td>#data16</td>
<td>16-bit constant included in instruction</td>
</tr>
<tr>
<td>Bit</td>
<td>128 software flags, any IO pin, control or status bit For example: ACC.0, P1.0, P1.1</td>
</tr>
</tbody>
</table>

**Boolean variable manipulation**

| CLR C   | clear carry flag |
| CLR bit | clear direct bit |
| SETB C  | set carry flag  |
| SETB bit| set direct bit  |
| CPL C   | complement carry flag |
| CPL bit | complement direct bit |
| ANL C, bit | AND direct bit to carry flag |
| ORL C, bit | OR direct bit to carry flag |
| MOV C, bit | Move direct bit to carry flag |

**Program and machine control**
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCALL addr16</td>
<td>long subroutine call</td>
</tr>
<tr>
<td>RET</td>
<td>return from subroutine</td>
</tr>
<tr>
<td>RETI</td>
<td>return from interrupt</td>
</tr>
<tr>
<td>LJMP addr16</td>
<td>long jump</td>
</tr>
<tr>
<td>SJMP rel</td>
<td>short jump (relative address)</td>
</tr>
<tr>
<td>JMP @A+DPTR</td>
<td>jump indirect relative to the DPTR</td>
</tr>
<tr>
<td>JZ rel</td>
<td>jump if accu is zero</td>
</tr>
<tr>
<td>JNZ rel</td>
<td>jump if accu is not zero</td>
</tr>
<tr>
<td>JC rel</td>
<td>jump if carry flag is set</td>
</tr>
<tr>
<td>JNC rel</td>
<td>jump if carry flag is not set</td>
</tr>
<tr>
<td>JB bit,rel</td>
<td>jump if direct bit is set</td>
</tr>
<tr>
<td>JNB bit,rel</td>
<td>jump if direct bit is not set</td>
</tr>
<tr>
<td>JBC bit,rel</td>
<td>jump if direct bit is set &amp; clear bit</td>
</tr>
<tr>
<td>CJNE A,direct,rel</td>
<td>compare direct to A &amp; jump of not equal</td>
</tr>
<tr>
<td>CJNE A,#data,rel</td>
<td>comp. I’mmed. to A &amp; jump if not equal</td>
</tr>
<tr>
<td>CJNE Rn,#data,rel</td>
<td>comp. I’mmed. to reg. &amp; jump if not equal</td>
</tr>
<tr>
<td>CJNE @Ri,#data,rel</td>
<td>comp. I’mmed. to ind. &amp; jump if not equal</td>
</tr>
<tr>
<td>DJNZ Rn,rel</td>
<td>decrement register &amp; jump if not zero</td>
</tr>
<tr>
<td>DJNZ direct,rel</td>
<td>decrement direct &amp; jump if not zero</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
</tbody>
</table>

**Arithmetic operations**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Rn</td>
<td>add register to accu</td>
</tr>
<tr>
<td>ADD A,direct</td>
<td>add register byte to accu</td>
</tr>
<tr>
<td>ADD A,@Ri</td>
<td>add indirect RAM to accu</td>
</tr>
<tr>
<td>ADD A,#data</td>
<td>add immediate data to accu</td>
</tr>
<tr>
<td>ADDC A,Rn</td>
<td>add register to accu with carry</td>
</tr>
<tr>
<td>ADDC A,direct</td>
<td>add direct byte to accu with carry flag</td>
</tr>
<tr>
<td>ADDC A,@Ri</td>
<td>add indirect RAM to accu with carry flag</td>
</tr>
<tr>
<td>ADDC A,#data</td>
<td>add immediate data to accu with carry flag</td>
</tr>
<tr>
<td>SUBB A,Rn</td>
<td>subtract register from A with borrow</td>
</tr>
<tr>
<td>SUBB A,direct</td>
<td>subtract direct byte from A with borrow</td>
</tr>
<tr>
<td>SUBB A,@Ri</td>
<td>subtract indirect RAM from A with borrow</td>
</tr>
<tr>
<td>SUBB A,#data</td>
<td>subtract immediate data from A with borrow</td>
</tr>
<tr>
<td>INC A</td>
<td>increment accumulator</td>
</tr>
<tr>
<td>Assembly Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>INC Rn</td>
<td>increment register</td>
</tr>
<tr>
<td>INC direct</td>
<td>increment direct byte</td>
</tr>
<tr>
<td>INC@Ri</td>
<td>increment indirect RAM</td>
</tr>
<tr>
<td>DEC A</td>
<td>decrement accumulator</td>
</tr>
<tr>
<td>DEC Rn</td>
<td>decrement register</td>
</tr>
<tr>
<td>DEC direct</td>
<td>decrement direct byte</td>
</tr>
<tr>
<td>DEC@Ri</td>
<td>decrement indirect RAM</td>
</tr>
<tr>
<td>INC DPTR</td>
<td>increment datapointer</td>
</tr>
<tr>
<td>MUL AB</td>
<td>multiply A &amp; B</td>
</tr>
<tr>
<td>DIV AB</td>
<td>divide A by B</td>
</tr>
<tr>
<td>DA A</td>
<td>decimal adjust accu</td>
</tr>
</tbody>
</table>

**Logical operations**

<table>
<thead>
<tr>
<th>Assembly Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL A,Rn</td>
<td>AND register to accu</td>
</tr>
<tr>
<td>ANL A,direct</td>
<td>AND direct byte to accu</td>
</tr>
<tr>
<td>ANL A,@Ri</td>
<td>AND indirect RAM to accu</td>
</tr>
<tr>
<td>ANL A,#data</td>
<td>AND immediate data to accu</td>
</tr>
<tr>
<td>ANL direct,A</td>
<td>AND accu to direct byte</td>
</tr>
<tr>
<td>ANL direct,#data</td>
<td>AND immediate data to direct byte</td>
</tr>
<tr>
<td>ORL A,Rn</td>
<td>OR register to accu</td>
</tr>
<tr>
<td>ORL A,direct</td>
<td>OR direct byte to accu</td>
</tr>
<tr>
<td>ORL A,@Ri</td>
<td>OR indirect RAM to accu</td>
</tr>
<tr>
<td>ORL A,#data</td>
<td>OR immediate data to accu</td>
</tr>
<tr>
<td>ORL direct,A</td>
<td>ORL accu to direct byte</td>
</tr>
<tr>
<td>ORL direct,#data</td>
<td>ORL immediate data to direct byte</td>
</tr>
<tr>
<td>XRL A,Rn</td>
<td>exclusive OR register to accu</td>
</tr>
<tr>
<td>XRL A,direct</td>
<td>exclusive OR direct byte to accu</td>
</tr>
<tr>
<td>XRL A,@Ri</td>
<td>exclusive OR indirect RAM to accu</td>
</tr>
<tr>
<td>XRL A,#data</td>
<td>exclusive OR immediate data to accu</td>
</tr>
<tr>
<td>XRL direct,A</td>
<td>exclusive OR accu to direct byte</td>
</tr>
<tr>
<td>XRL direct,#data</td>
<td>exclusive OR immediate data to direct byte</td>
</tr>
<tr>
<td>CLR A</td>
<td>clear accu</td>
</tr>
<tr>
<td>CPL A</td>
<td>complement accu</td>
</tr>
<tr>
<td>RL A</td>
<td>rotate accu left</td>
</tr>
<tr>
<td>RLC A</td>
<td>rotate A left through the carry flag</td>
</tr>
<tr>
<td>RR A</td>
<td>rotate accu right</td>
</tr>
<tr>
<td>RRC A</td>
<td>rotate accu right through the carry flag</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SWAP A</td>
<td>swap nibbles within the accu</td>
</tr>
</tbody>
</table>

**Data transfer**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rn</td>
<td>move register to accu</td>
</tr>
<tr>
<td>MOV A,direct</td>
<td>move direct byte to accu</td>
</tr>
<tr>
<td>MOV A,@Ri</td>
<td>move indirect RAM to accu</td>
</tr>
<tr>
<td>MOV A,#data</td>
<td>move immediate data to accu</td>
</tr>
<tr>
<td>MOV Rn,A</td>
<td>move accu to register</td>
</tr>
<tr>
<td>MOV Rn,direct</td>
<td>move direct byte to register</td>
</tr>
<tr>
<td>MOV Rn,#data</td>
<td>move immediate data to register</td>
</tr>
<tr>
<td>MOV direct,A</td>
<td>move accu to direct byte</td>
</tr>
<tr>
<td>MOV direct,Rn</td>
<td>move register to direct byte</td>
</tr>
<tr>
<td>MOV direct,direct</td>
<td>move direct byte to direct</td>
</tr>
<tr>
<td>MOV direct,@Ri</td>
<td>move indirect RAM to direct byte</td>
</tr>
<tr>
<td>MOV direct,#data</td>
<td>move immediate data to direct byte</td>
</tr>
<tr>
<td>MOV@Ri,A</td>
<td>move accu to indirect RAM</td>
</tr>
<tr>
<td>MOV@Ri,direct</td>
<td>move direct byte to indirect RAM</td>
</tr>
<tr>
<td>MOV@Ri,#data</td>
<td>move immediate to indirect RAM</td>
</tr>
<tr>
<td>MOV DPTR,#data16</td>
<td>load datapointer with a 16-bit constant</td>
</tr>
<tr>
<td>MOVC A,@A+DPTR</td>
<td>move code byte relative to DPTR to A</td>
</tr>
<tr>
<td>MOVC A,@A+PC</td>
<td>move code byte relative to PC to A</td>
</tr>
<tr>
<td>MOVX A,@Ri</td>
<td>move external RAM (8-bit) to A</td>
</tr>
<tr>
<td>MOVX A,@DPTR</td>
<td>move external RAM (16 bit) to A</td>
</tr>
<tr>
<td>MOVX@Ri,A</td>
<td>move A to external RAM (8-bit)</td>
</tr>
<tr>
<td>MOVX@DPTR,A</td>
<td>move A to external RAM (16-bit)</td>
</tr>
<tr>
<td>PUSH direct</td>
<td>push direct byte onto stack</td>
</tr>
<tr>
<td>POP direct</td>
<td>pop direct byte from stack</td>
</tr>
<tr>
<td>XCH A,Rn</td>
<td>exchange register with accu</td>
</tr>
<tr>
<td>XCH A,direct</td>
<td>exchange direct byte with accu</td>
</tr>
<tr>
<td>XCH A,@Ri</td>
<td>exchange indirect RAM with A</td>
</tr>
<tr>
<td>XCHD A,@Ri</td>
<td>exchange low-order digit ind. RAM w. A</td>
</tr>
</tbody>
</table>

**How to access labels from ASM.**

Each label in BASCOM is changed into a period followed by the label name.

**Example:**

GOTO Test

Test:
generated ASM code:
LJMP .Test
.Test:

When you are using ASM-labels you can also precede them with the !-Sign so the label won't be converted.
Jb P1.0, Test ; no period
!test : ; indicate ASM label

Or you can include the period in the labelname.
Another good alternative is to use the $ASM $END ASM directives.

Example:
$Asm
  mov a,#1
  test:
    sjmp test
$End Asm

How variables are stored.
BIT variables are stored in bytes. These bytes are stored from 20hex -2Fhex thus allowing 16 * 8 = 128 bit variables. You can access a bit variable as follows:

Dim var As Bit  'dim variable
SETB {var} ; set bit
CLR {var} ; clear bit
Print var ; print value
End

Or you can use the BASIC statement SET and RESET which do the same thing.

BYTE variables are stored after the BIT variables. Starting at address 20 hex + (used bytes for bit vars).

INTEGER/WORD variables are stored with the LSB at the lowest memory position. LONG variables are stored with the LSB at the lowest memory position too.

You can access variables by surrounding the variable with {}. To refer to the MSB of an Integer/Word use var+1. To refer to the MSB of a Long use var+3. The following example shows how to access the variables from ASM

Dim t as Byte, c as Integer
CLR a ; clear register a
MOV {t}, a ; clear variable t
INC {t} ; t=t + 1
MOV {c}, {t} ; c = t
MOV {c+0}, {t} ; LSB of C = t (you don't have to enter the +0)
MOV {t+1}, {t} ; MSB of C = t
MOV {c}, #10 ; assign value

You can also change SFRs from BASIC. P1 = 12  'this is obvious
ACC = 5 ; this is ok too
B = 3  'B is a SFR too
MUL AB 'acc = acc * b
Print acc

EXTERNAL variables are stored similar.
Strings are stored with a terminating zero.

Example:

$RAMSTART = 0
Dim s As String * 10 'reserve 10 bytes + 1 for string terminator
s = "abcde" 'assign string constant to string
ram location 0 = a 'first memory location
ram location 1 = b
ram location 2 = c
ram location 3 = d
ram location 4 = e
ram location 5 = #0

External variables must be accessed somewhat different.

Dim T as XRAM Byte
mov dptr,#\{T\} ; address of T to datapointer
mov a,#65 ; place A into acc
movx @dptr,a ; move to external memory
Print T ; print it from basic

Dim T1 as XRAM Integer
mov dptr,#\{T1\} ; set datapointer
mov a,#65 ; place A into acc (LSB)
movx @dptr,a ; move to external memory
inc dptr ; move datapointer
mov a,#1 ; 1 to MSB
movx @dptr,a ; move to external memory

Print T1 ; print it from basic

Helper routines
There are two ASM helper routines that can make it a bit easier:
PLACEVALUE var , SFR
PLACEADRES var, SFR

PLACEVALUE assigns the variable, var, to the specified register, SFR.
Placevalue 1, A will generate:
Mov a,#1

Dim x as Byte
Placevalue x , R0 will generate:
Mov a, h'3A ; in this example only of course

Where it is becoming handy is with arrays:
Placevalue a(x), RO will generate:

Mov r0,#h'3A
Mov a,@r0
RI a
Add a,#h'1F
Mov R0,a
Mov a,@r0

These are all examples, the generated code will differ with the type of variables used.
You can only assign 1 SFR with the PLACEVALUE statement.
This is where PLACEADRES comes around the corner.
Placeadres , places a variables address into a register.

Placeadres ar(x),A
Placeadres z , R0

When external variables are used, you don't need to specify a register because DPTR is always assigned.

Dim X as xram Integer
PLACEADRES x , dptr or PLACEADRES x
Will generate :
Mov dptr,#2

Or with arrays :
PLACEADRES ar(x)

Mov dptr,#2
Mov r0,#h'37
Mov a,@r0
Mov r2,a
Inc r0
Mov a,@r0
Mov r3,a
Mov r1,#1
Acall _AddIndex

Of course these are also examples, the generated code depends on the types and if they are internal or external variables.

**Hexdecimal notation**
You can also use hexadecimal notation.
Example : Mov a,#h'AA
Or use the BASIC notation :
Mov a,#&HAA

**Binary notation**
You can also use binary notation.
Example : Mov a,#&B10001000

**Jumping with offset**
You can specify an offset instead of a labelname when jumping.
Jb P1.0 , *+12 ;jump forward
Jb P1.0 , *+12 ;jump back
Jnb P1.0 , *+0 ;loop until P1.0 becomes high
This also applies to the other instructions where can be jumped to a label like SJMP, LJMP DJNZ etc.

**Internal buffer for string conversion**
The string conversion routines used for PRINT num, STR() and VAL(), use an internal buffer of 16 bytes. This has the advantage that no stack handling is needed but the disadvantage that a fixed space is used.
Of course you can use this buffer. It can be referenced with **___TMP_S1**
So when you need a temp string, you can use this buffer.
Note that this buffer is only available with the mentioned statements!

**Example:**
Dim s as single
s = 1.1
Print s                      ; now the buffer is needed
___TMP_S1 = "Use this space"
Print ___TMP_S1

**Comment**
The ; sign can be used or the BASIC comment sign '
Mov a,#1 ; comment
Mov a,#2  'comment

7.2  **Internal registers**
You can manipulate the register values directly from BASIC.
They are also reserved words. The internal registers are:

**BIT addressable registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCON</td>
<td>Timer/counter control</td>
</tr>
<tr>
<td>P1</td>
<td>Port 0 latch</td>
</tr>
<tr>
<td>SCON</td>
<td>Serial port control</td>
</tr>
<tr>
<td>IE</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>P3</td>
<td>Port 3 latch</td>
</tr>
<tr>
<td>IP</td>
<td>Interrupt priority control</td>
</tr>
<tr>
<td>PSW</td>
<td>Program status word</td>
</tr>
<tr>
<td>ACC</td>
<td>Accumulator</td>
</tr>
<tr>
<td>B</td>
<td>B register</td>
</tr>
</tbody>
</table>

**BYTE addressable register**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>DPL</td>
<td>Data pointer low word</td>
</tr>
<tr>
<td>DPH</td>
<td>Data pointer high word</td>
</tr>
<tr>
<td>PCON</td>
<td>Power control</td>
</tr>
<tr>
<td>TMOD</td>
<td>Timer/counter mode control</td>
</tr>
</tbody>
</table>
The registers and their addresses are defined in the REG51.DAT file which is placed in the BASCOM application directory. You can use an other file for other uPs. You can select the appropriate register file with the Options Compiler settings.

Take care when you are directly manipulating registers! The ACC and B register are frequently used by BASCOM. Also the SP register is better to be left alone. Altering SP will certainly crash your application!

Bit addressable registers can be used with the `SET` and `RESET` statements and as bit-variables. Byte addressable registers can be used as byte variables. `P1 = 40` will place a value of 40 into port 40.

Please 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 B as Byte` because B is an internal register. You can however manipulate the register with the `B = value` statement.

Making your own register file is very simple:
- copy the 8052.DAT file to a new DAT file for example myproc.DAT
- DOS c:\bascom copy 8052.dat myproc.dat
- edit the registerfile with BASCOM

A register file has a few sections. The following example shows only a few items under each section.

The [BIT] section contains all SFR's which are bit addressable. A bit addressable SFR ends with 0 or 8. After the SFR name you can write the hexadecimal address. An optional initial value for the simulator can also be specified. Separate the values by a comma.

The [BYTE] section contains all the other SFR's.

The [MISC] section has a few items:
- **up**: here you can enter a short name for the uP.
- **IRAM**: the amount of available internal memory (128 or 256 bytes)
- **org**: the hexadecimal address where the code can start. This is 3 bytes after the last interrupt entry address, because the last interrupt will have a LJMP to an ISR and a LJMP needs 3 bytes.
- **I_**xxx**: where xxx is the name of the additional interrupt. The name must be no
longer than 6 characters. As you can see in the example below the last interrupt T2 has an entry address of 73 (hex). So the org is set to 73+3 = 76 (hex). You only need to specify the additional interrupts. The interrupts for INT0, INT1, TIMER0, TIMER1 and SERIAL are already handled by the compiler.

- CLOCKDIV : The division factor of the oscillator. By default this is 12 and when you don't specify it, 12 will be used. Some micro processors have a division factor of 6 or 4.

**EXAMPLE**

**[BIT]**
ACC = E0
B = F0

**[BYTE]**
ADCH = C6
ADCON = C5
CTCON = EB

**[MISC]**
up = 80552
I_TIMER2 = 2B
I_CT0 = 33
I_CT1 = 3B
I_CT2 = 43
I_CT3 = 4B
I_ADC = 53
I_CM0 = 5B
I_CM1 = 63
I_CM2 = 6B
I_T2 = 73
org = 76
IRAM = 256
CLOCKDIV = 12

## 7.3 Initialization

BASCOM initializes the processor depending on the used statements. When you want to handle this by yourself you can specify this by the meta command $NOINIT.

The only initialization that is always done is the setting of the stack pointer and the initialization of the LCD display (if LCD related statements are used).

You can use the $NOSP statement when you don't want the stack pointer to be set.

All data used for variables like the internal RAM or external RAM, is in an unknown condition at startup. This means that you can not assume that a variables is 0.

For example:

Dim a as byte
Print a
End

When you run the code, 'a' can contain any value. When you want to be sure the variable is 0, assign it with 0. During a reset, the memory content might be the
same as before the reset, but again, there is no guarantee.
Part VIII
8 Additional Hardware

8.1 Additional Hardware

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

- **I2CSEND** and **I2CRECEIVE** and other I2C related statements.
- **LCD, LCDHEX, DISPLAY** and other related LCD statements.
- 1 WIRE bus explanation.

More about connecting a LCD display.
More about the I2C bus.

**Hardware related commands**

The uP must be connected to a crystal. The frequency of the crystal can range from 0 to 24 Mhz for most chips. The frequency is divided by 12 internally. So with a 12 Mhz crystal the processor is clocked with 1 Mhz. Because almost each instruction takes, 1 clock cycle to execute the processor can handle 1 MIPS.

When RS-232 statements such as INKEY, PRINT and INPUT are used, TIMER1 is connected to the system clock. So TIMER1 can't be used for other purposes such as ON TIMER1 anymore. When no RS-232 related statements are used you can use TIMER1.

The Baud rate is generated by dividing the system clock. When a crystal of 11.0592 Mhz is used, the Baud rate can be generated very accurately. Other crystals can be used too but the generated baud rate will never be exactly 2400 or 4800 baud and higher baud rates are almost impossible. The exact baud rate is shown in the report file.

**Clock**

The clock frequency is the system frequency divided by 12. With a 12 Mhz crystal this means that every microsecond the register is incremented.

**Timers and Counters**

The 8051 has two 16-bit timers named TIMER0 and TIMER1. Below the internal representation of timer0 is shown. TIMER0 and TIMER1 are almost identical so you can read TIMER1 for TIMER0.
Each counter register has two SFRs associated with it. For TIMER0 the SFRs are TL0 and TH0. TL0 is the lowest byte of TIMER0 and TH0 is the highest byte of TIMER0. These two registers make the timers 16-bit wide.

The timer can operate as a timer or as a counter.

**A timer uses the system clock divided by 12 as the source of its input pulses.**
**So it increments periodical.**

**A counter uses external pulses to increment its count.**
The external pulses are received at alternative pin P3.4 for TIMER0 and P3.5 for TIMER1.
The timer/counter can be controlled by the run-bit TR0.

You can stop a timer/counter with the statement `STOP TIMER0` or `STOP COUNTER0`.
You can start a timer/counter with the statement `START TIMER0` or `START TIMER1`.

The timer/counter can also be controlled with the alternative pin P3.2.
This pin is labeled for its alternative INTO-input but it can be used to control the timer.
When GATE is reset the timer/counter is enabled.
When GATE is set the timer/counter is enabled if INTO is active(low). (provided that the timer is started)

The timer/counter can operate in four modes:
- **mode 0 : 13-bit counter.**
  An interrupt is generated when the counter overflows. So it takes 8192 pulses to generate the next interrupt.
- **mode 1 : 16-bit counter.**
Mode 1 is similar to mode 0. It implements a 16-bit counter. It takes 65536 input pulses to generate the next interrupt.

- **mode 2**: 8-bit auto reload.
  TL0 serves as an 8-bit timer/counter. When the timer/counter overflows the number stored in TH0 is copied into TL0 and the count continues. An interrupt is generated each time the counter overflows and a reload is performed.

- **mode 3**: TIMER1 is inactive and holds its count. (TIMER1). For TIMER0 in timer mode two 8-bit timers are available and in counter mode one 8-bit timer is available. See a datasheet for more details.

The timer/counter can be configured with the **CONFIG** statement.

**CONFIG** TIMER0= COUNTER/TIMER, GATE=INTERNAL/EXTERNAL, MODE=0-3

The first argument is the timer/counter you want to configure, TIMER0 in this case. GATE specifies if external timer control with the INT0 pin is enabled. MODE specifies the timer/counter mode (0-3).

So **CONFIG TIMER0 = COUNTER, GATE = INTERNAL, MODE=2** will configure TIMER0 as a COUNTER with no external gate control, in mode 2 (auto reload). When the timer/counter is configured the timer/counter is stopped so you must start it afterwards with the **START TIMER0** statement.

The **ON TIMERx** statement can be used to respond to a timer/counter interrupt when the timer overflows.

When the timer/counter is used in mode 2 (auto reload) the reload value can be specified with the **LOAD TIMERx, value** statement. Because it is an 8-bit register a maximum time of 255 uS can be achieved.

So for a period of 10 uS you must supply a value of (256-10) is 246. To make things easier you can assign the value directly: LOAD TIMERx, 250 will internally be transformed into 256-250=6. This saves you the trouble of calculating the correct value.

The **COUNTER0** and **COUNTER1** variables hold the values of timer/counter 0 and 1. You can also set the timer/counter contents with the **COUNTER0 = value** statement.

Please note that with the **LOAD** statement, you can only load a byte value into the timer/counter. Because the statement is meant for timer/counter mode 2.

Also note that you can assign a value to the timer/counter with the **COUNTER0/COUNTER1** variables. You cannot use the **TIMER0/TIMER1** in its place but it does the same thing: assigning/retrieving the timer/counter.

Port 3 is a unique port because it has alternative functions. That is you can use it as a port like P3.1 = 1 or SET P3.1 or you can make use of the double function of this port.

<table>
<thead>
<tr>
<th>Port</th>
<th>Alternative function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RxD receive data for RS-232</td>
</tr>
<tr>
<td>P3.1</td>
<td>TxD    transmit data for RS-232</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 interrupt 0 input/timer 0 gate control</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 interrupt 1 input/timer 1 gate control</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 timer 0 input or counter input</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 timer 1 input or counter input</td>
</tr>
<tr>
<td>P3.6</td>
<td>-</td>
</tr>
<tr>
<td>P3.7</td>
<td>-</td>
</tr>
</tbody>
</table>

When you make use of the PRINT, INPUT and other RS-232 related statements P3.0 and P3.1 are used for the RS-232 interface.

When you make use of the INT0/INT1 interrupts, you must connect an interrupt source to the corresponding pins. A switch for example. The INTx interrupt can occur on the falling edge of a signal or when the signal is low.

Use the following statements to specify the trigger:

<table>
<thead>
<tr>
<th>SET TCON.0</th>
<th>Falling edge generates interrupt for INT0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET TCON.0</td>
<td>Low signal generates interrupt for INT0.</td>
</tr>
<tr>
<td>SET TCON.2</td>
<td>Falling edge generates interrupt for INT1.</td>
</tr>
<tr>
<td>RESET TCON.2</td>
<td>Low signal generates interrupt for INT1.</td>
</tr>
</tbody>
</table>

When TCON.x is RESET the interrupts keep on occurring while the input is low. When TCON.x is SET the interrupt only occurs on the falling edge.

To test if a hardware interrupt is generated you can test the TCON.1 and TCON.3 flags. These flags are set by hardware when an external interrupt edge is detected. They are reset by the RETURN statement of the interrupt service routine or subroutine.

TCON.1 must be tested for INT0 and TCON.3 must be tested for INT1.

Some uPs have an additional timer named TIMER2. It depends on the used chip which features TIMER2 has.

**Ports and Power Up**

**Port 1** is an 8-bit bi-directional I/O port. Port pins P1.2 to P1.7 provide internal pull-ups. P1.0 and P1.1 requires external pull-ups. P1.0 and P1.1 also serve as the positive input(AINO) and the negative input(AIN1), respectively, of the on-chip precision analog comparator.

The port 1 output buffers can sink 20 mA and can drive LED displays directly. When 1s are written to Port 1 pins, they can be used as inputs. When pins P1.2 to P1.7 are used as inputs and are externally pulled low, they will source current because of the internal pullups.

**Port 3** pins P3.0 to P3.5, P3.7 are seven bi-directional I/O pins with internal pull-ups. P3.6 is hard wired as an input to the output of the on-chip comparator and is not accessible as a general purpose I/O pin.
The port3 output buffers can sink 20 mA. When 1’s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. Port 3 pins that are externally being pulled low will source current because of the pullups. Port 3 also serves the functions of various special features of the AT89C2051 as listed below.

<table>
<thead>
<tr>
<th>Port</th>
<th>Alternative function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RxD receive data for RS-232</td>
</tr>
<tr>
<td>P3.1</td>
<td>TxD transmit data for RS-232</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 interrupt 0 input/timer 0 gate control</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 interrupt 1 input/timer 1 gate control</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 timer 0 input or counter input</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 timer 1 input or counter input</td>
</tr>
<tr>
<td>P3.7</td>
<td>-</td>
</tr>
</tbody>
</table>

Writing to a Port
P1 = 255 will write the value 255 to the port 1, setting all the pins to 1 so all pins can be used as inputs.

P1 = 0 will write the value 0 to port 1, setting all pins to zero.

Reading from a Port
byte = P1 will read the value from port 1 and will assign the value to variable byte.

Setting individual pins of a Port
You can also set individual pins of the ports in BASCOM.

SET P1.0 will set pin P1.0 high. P1.0 = 1 will also set pin P1.0 high.

RESET P1.0 will set pin P1.0 low. P1.0 = 0 will also set pin P1.0 low.

At power up both ports are high and can be used as inputs. Individual bits can be set to use a port both as input/output. For example: P1 = &B00001111 , will set a value of 15 to port 1. P1.0 to P1.3 can be used as inputs because they are set high.

How to interface the port pins
The schematic above shows how to connect a LED as an output, a speaker as an output and a switch as an input device.

8.2 Alternative port-pin functions

The AT89S8252 ports have alternative functions. The following table shows the alternative functions.

<table>
<thead>
<tr>
<th>Port pin</th>
<th>Alternate function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.0</td>
<td>T2 external count input to timer.counter 2, clock out</td>
</tr>
<tr>
<td>P1.1</td>
<td>T2EX timer/counter 2 capture/reload trigger and direction flag</td>
</tr>
<tr>
<td>P1.4</td>
<td>/SS Slave port select input</td>
</tr>
<tr>
<td>P1.5</td>
<td>MOSI Master data output, slave data input pin for SPI channel</td>
</tr>
<tr>
<td>P1.6</td>
<td>MISO Master data input, slave data output pin for SPI channel</td>
</tr>
<tr>
<td>P1.7</td>
<td>SCK Master clock output, slave clock input pin for SPI channel</td>
</tr>
<tr>
<td>P3.0</td>
<td>RxD serial input port</td>
</tr>
<tr>
<td>P3.1</td>
<td>TxD serial output port</td>
</tr>
<tr>
<td>P3.2</td>
<td>/INT0 external interrupt 0</td>
</tr>
<tr>
<td>P3.3</td>
<td>/INT1 external interrupt 1</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 timer 0 external input</td>
</tr>
</tbody>
</table>
T1 timer 1 external input
/WR external data memory write strobe
/RD external data memory read strobe

/ means active low

8.3 Hardware - LCD display

The LCD display can be connected as follows:

<table>
<thead>
<tr>
<th>LCD-DISPLAY</th>
<th>PORT</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB7</td>
<td>P1.7</td>
<td>14</td>
</tr>
<tr>
<td>DB6</td>
<td>P1.6</td>
<td>13</td>
</tr>
<tr>
<td>DB5</td>
<td>P1.5</td>
<td>12</td>
</tr>
<tr>
<td>DB4</td>
<td>P1.4</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>P1.3</td>
<td>6</td>
</tr>
<tr>
<td>RS</td>
<td>P1.2</td>
<td>4</td>
</tr>
<tr>
<td>RW</td>
<td>Ground</td>
<td>5</td>
</tr>
<tr>
<td>Vss</td>
<td>Ground</td>
<td>1</td>
</tr>
<tr>
<td>Vdd</td>
<td>+5 Volt</td>
<td>2</td>
</tr>
<tr>
<td>Vo</td>
<td>0-5 Volt</td>
<td>3</td>
</tr>
</tbody>
</table>

This leaves P1.1 and P1.0 and P3 for other purposes.

You can change the LCD pin layout from the Options LCD menu.
You can select the display used with the CONFIG LCD statement.

The LCD display operates in 4-bit mode.
See the $LCD statement for operation in 8-bit mode.

BASCOM supports a lot of statements to control the LCD display.
For those who want to have more control the example below shows how to do so.

Acc = 5
Call Lcd_control
Acc = 65
Call Write_lcd

'load register A with value
'it is a control value to control the display
'load with new value (letter A)
'write it to the LCD display

Note that lcd_control and write_lcd are assembler subroutines which can be called from BASCOM.

See manufacture details from your LCD display for the correct assignment.
8.4 Hardware - I2C

The design below shows how to implement an I2C-bus. Note that you can select which port pins you want to use for the I2C interface with the compiler settings. You can also select the SDA and SCL pin with the `CONFIG SDA` and `CONFIG SCL` statement.

8.5 1WIRE INFO

The following information is written by Göte Haluza, thanks!

Dallas Semiconductor (DS) 1wire. This is a brief description of DS 1wirebus when used in combination with BASCOM. For more detailed explanations about the 1wirebus, please go to http://www.dalsemi.com/techbriefs/tb1.html. Using BASCOM, makes the world a lot easier. This paper will approach the subject from a "BASCOM-user-point-of-view".

1wire-net is a serial communication protocol, used by DS devices. The bus could be implemented in two basic ways:

**With 2 wires**, then DQ and ground is used on the device. Power is supplied on the DQ line, which is +5V, and used to charge a capacitor in the DS device. This power is used by the device for its internal needs during communication, which makes DQ go low for periods of time. This bus is called the **1wirebus**.

**With 3 wires**, when +5V is supplied to the VDD line of the device, and DQ + ground as above. This bus is called the **2wirebus**.

So, the ground line is "not counted" by DS. But hereafter we use DS naming conventions.
How it works. (1wire)
The normal state of the bus is DQ=high. Through DQ the device gets its power, and performs the tasks it is designed for.

When the host (your micro controller (uC)) wants something to happen with the 1w-bus, it issues a reset-command. That is a very simple electric function that happens then; the DQ goes active low for a time (480uS on original DS 1w-bus). This put the DS-devices in reset mode; then (they) send a presence pulse, and then (they) listen to the host.

The presence pulse is simply an active low, this time issued by the device(s).

Now, the host cannot know what is on the bus, it is only aware of that at least 1 DS device is attached on the bus.

All communication on the 1w-bus is initialized by the host, and issued by time-slots of active-low on a normally high line (DQ), issued by the device, which is sending at the moment. The device(s) internal capacitor supplies its power needs during the low-time.

How you work with 1w-bus
Thereafter, you can read a device, and write to it. If you know you only have 1 sensor attached, or if you want to address all sensors, you can start with a "Skip Rom" - command. This means; take no notice about the Ids of the sensors - skip that part of the communication.

When you made a 1w-reset, all devices of the bus are listening. If you chose to address only one of them, the rest of them will not listen again before you have made a new 1w-reset on the bus.

I do not describe BASCOM commands in this text - they are pretty much self-explaining. But the uC has to write the commands to the bus - and thereafter read the answer. What you have to write as a command depends on devices you are using - and what you want to do with it. Every DS chip has a datasheet, which you can find at http://www.dalsemi.com/datasheets/pdfindex.html. There you can find out all about the actual devices command structure.

There are some things to have in mind when deciding which of the bus-types to use.
The commands, from BASCOM, are the same in both cases. So this is not a problem.

The +5V power-supply on the VDD when using a 2wire-bus has to be from separate power supply, according to DS. But it still works with taking the power from the same source as for the processor, directly on the stabilising transistor. I have not got it to work taking power directly from the processor pin.

Some devices consume some more power during special operations. The DS1820 consumes a lot of power during the operation "Convert Temperature". Because the sensors knows how they are powered (it is also possible to get this information from the devices) some operations, as "Convert T" takes different amount of time for the
sensor to execute. The command "Convert T" as example, takes ~200mS on 2wire, but ~700mS on 1wire. This has to be considered during programming.

And that power also has to be supplied somehow.
If you use 2wire, you don't have to read further in this part. You can simultaneously "Convert T" on all the devices you attach on the bus. And save time. This command is the most power-consuming command, possible to execute on several devices, I am aware of.

If you use 1wire, there are things to think about. It is about not consuming more power than you feed. And how to feed power? That depends on the devices (their consumption) and what you are doing with them (their consumption in a specific operation).

Short, not-so-accurate description of power needs, not reflecting on cable lengths
Only the processor pin as power supplier, will work < 5 sensors. (AVR, 1w-functions use an internal pull-up. 8051 not yet tested). Don't even think of simultaneous commands on multiple sensors.

With +5V through a 4K7 resistor, to the DQ-line, 70 sensors are tested. But, take care, cause issuing "Convert T" simultaneously, would cause that to give false readings. About ~15 sensors is the maximum amount of usable devices, which simultaneously performs some action. This approach DS refers to as "pull-up resistor".

With this in mind, bus up to 70 devices has been successfully powered this way.

The resistor mentioned, 4K7, could be of smaller value. DS says minimum 1K5, I have tested down to 500 ohm - below that the bus is not usable any more. (AVR). Lowering the resistor feeds more power - and makes the bus more noise-resistant. But, the resistor minimum value is naturally also depending on the uC-pin electric capabilities. Stay at 4K7 - which is standard recommendation.

DS recommends yet another approach, called "strong pull-up" which (short) works via a MOS-FET transistor, feeding the DQ lines with enough power, still on 1wire, during power-consuming tasks. This is not tested, but should naturally work. Cause this functionality is really a limited one; BASCOM has no special support for that. But anyway, we tell you about it, just in case you wonder. Strong pull-up has to use one uC pin extra - to drive the MOS-FET.

Cable lengths (this section is only for some limited understanding)
For short runs up to 30 meters, cable selection for use on the 1W bus is less critical. Even flat modular phone cable works with limited numbers of 1-Wire devices. However, the longer the 1W bus, the more pronounced cable effects become, and therefore the greater importance placed on cable selection.

For longer distances, DS recommends twisted-pair-cable (CAT5).

DS standard examples show 100 meters cable lengths, so they say, that's no
Additional Hardware

Noise and CRC
The longer cable and the noisier environment, the more false readings will be made. The devices are equipped with a CRC-generator - the LSByte of the sending is always a checksum. Look in program examples to learn how to re-calculate this checksum in your uC. AND, if you notice that there are false readings - do something about your cables. (Shield, lower resistor)

Transfer speed
On the original 1w-bus, DS says the transfer speed is about 14Kbits/second. And, if that was not enough, some devices has an overdrive option. That multiplies the speed by 10. This is issued by making the communication-time-slots smaller (from 60 uS to 6uS) which naturally will make the devices more sensitive, and CRC-error probably occur more often. BUT, if that is not an issue, ~140Kbit is a reachable speed to the devices. So, whatever you thought before, it is FAST.

The BASCOM scanning of the bus is finds about 50 devices/second, and reading a specific sensors value to a uC should be about 13 devices/second.

Topology
Of the 1w-net - that is an issue we will not cover so much. Star-net, bus-net? It seems like you can mix that. It is a bus-net, but not so sensitive about that.

The benefit of the 1w-bus
Each device is individual - and you can communicate with it over the media of 2 wires. Still, you can address one individual device, if you like. Get its value. There are 64 ^ 2 unique identifications-numbers.
Naturally, if lot of cables are unwanted, this is a big benefit. And you only occupy 1 processor pin.

DS supplies with different types of devices, which all are made for interfacing an uC directly. No extra hardware. There are sensors, so you can get knowledge about the real world, and there are also potentiometers and relays, so you can do something about it. On the very same bus.

And the Ibutton approach from DS (ever heard of it?) is based on 1wire technology. Maybe something to pick up.

BASCOM let you use an uC with 1wire-devices so easy, that (since now) also has to count as a benefit - maybe one of the largest. ;-)
Göte Haluza
System engineer
Part IX
9 Supported Programmers

9.1 MCS Flash programmer

There are different models of the MCS Flash programmer, but all of them are compatible with the driver software.

The MCS Flash programmer is a parallel printer port based programmer. It can only program the ATMEL 89C1051, AT89C2051 and AT89C4051. Select the correct printer port address before you run the programmer. Be sure to switch on the power supply before running BASCOM.

When you run the programmer, the buffer will be filled automatically with your program data. The programmer works with binary files. The following menu options are available.

**Exit**
Exit the programmer.

**Buffer clear**
Clear the buffer. That is, fill it with zero bytes.

**Buffer Read from disk**
Load a file into the buffer. By default the current program.BIN file is selected. Select a file with the file selection dialog box and press the Ok-button.

**Buffer Write to disk**
Write the content of the buffer to a file. Note that the file size is 1024, 2048 or 4096 bytes depending on the chip type.
Buffer read from chip
Read the content of the FLASGROM into the buffer.
If the lock bits are set all bytes will return FF.

Buffer Write to chip
Program the chip with the content of the buffer.
The chip is erased before the buffer is written to the chip.

Buffer Verify
Compares the content of the buffer with the content of the chip.

Buffer program chip
Erases the chip, writes the buffer to the chip and finally verifies the buffer with the chip.

Chip get type
Retrieves the chip type. AT89C1051, AT89C2051 or AT89C4051.

Chip Erase
Erases the chip. Lock bits are also erased.

Chip Set lockbit 1
When LB1 is set the chip can not be programmed anymore.

Chip Set lockbit 2
When LB2 is set the chip can not be programmed nor can it be verified (read)
Use LB1 and LB2 together for securing your program.

Options LPT1 .. LPT3
Select the printer port the programmer is connected to.

Option Port delay
Because computers become faster every day and the hardware is run by software a
delay can be specified for very fast computers.
A value of 5 is used on a 486DX266. You must increase the value on faster
computers if problems occur.
The default is 0, and for best results, 0 should be used.
9.2 MCS SPI programmer

The MCS SPI programmer is a parallel printer port based SPI-programmer. It is a modified design of Jakub Jiricek's SPI-programmer. (two LED's were added)

The programmer can program the AT89S52 which has an extra 2048 bytes built in EEPROM for storing data and the AT89S53.

The nice thing about SPI-programmable chips is that the chip can be programmed in circuit. You only must design your application so that the SPI-port pins will not be pulled low.

The following menu options are available:

**File exit**
Will exit the programmer.

**Write code**
Will program the chip with the current programs binary image.

**Write data**
Will ask for a file and will write the data to the EEPROM.

**Verify code**
Will verify the programs binary image with the chip content.

**Verify data**
Will verify a file with the chips EEPROM content.

**Read code**
Will ask for a filename and will write the chip content to the file.

**Read data**
Will ask for a filename and will write the EEPROM content to the file.

**Chip reset**
Will reset the chip.

**Chip erase**
Will erase the chip.

**Chip set lockbits**
Will set the selected lock bits.

The following use feedback was received:
**I have at last found my problem with the SPI flash programmer designed by Jakub Jiricek.**

My PC's LPT port was set to NORMAL mode in the BIOS. Symptoms include normal reset pulse but very slow progress bar with eventual failure to verify.

Correct programmer operation was achieved by changing to EPP (enhanced
parallel port) mode in BIOS. I can only assume that the s/w must be using one line in bi-directional mode. Of course, this "fix" may only apply to my PC.

⚠ Not recommended for new programmers.

9.3 Blow IT Flashprogrammer

The Blow IT flashprogrammer is a parallel printer port based programmer and can only erase, and program a chip. The programmer works only with the AT89C1051 to AT89C4051 chips.

The programmer uses the same interface as the MCS Flashprogrammer, but doesn't support all the features due it's design. So for a description read the MCS Flashprogrammer help.

⚠ Not recommended for new programmers.

9.4 PG2051 flash programmer

The PG2051 is a serial comport based programmer and can program AT89C1051 and ATC2051 chips only. A nice feature is that the programmer can serve as an simulator too. The programmer works with Intel HEX files only.

The following menu options are available:

File Exit
This will exit the programmer.

Buffer read from disk
This allows you to load a binary file from disk. The current projects binary file is always loaded automatic.

Buffer write to disk
This option can be used to save the buffer to disk.

Buffer download
With this option you send the programs' hex file to the programmer/simulator. After it is sent, you can program the chip or simulate the program.

Buffer retrieve
Use this option to load the chip content into the buffer.

Buffer verify
This option will verify the buffer with the chip content.

Buffer autoprogram
This option will erase the chip, download the buffer, program the chip and finally verifies the chip.

Chip get type
To identify the chip you can select this option.
The radio-button 89C1051 or 89C2051 will be set.

**Chip set lockbit 1**  
Set lockbit 1 so the chip can not be programmed anymore.

**Chip set lockbit 2**  
Set lockbit 2 so the chip can not be programmed or verified/read anymore.

**Chip erase**  
Erases the chip.

**Chip program**  
Will program the chip with the downloaded buffer.

**Chip simulate**  
Will simulate the programmed program. This saves swapping the chip in and out of the target application.

9.5 **PG302 programmer**

The PG302 is a serial comport based programmer.  
The programmer can program a wide variety of chips with additional adapters.  
The BASCOM interface is designed to look similar with the original PG302 driver software.

You must select the target chip from the device list.  
Some chips will enable the memory radio buttons. For example the AT89S8252.  
You can select the memory-area with the radio buttons in these cases.

**Blank check**  
Will perform a blank check on the chip. That is, every memory location will be checked if it is equal to 255 (hex FF), indicating an un-programmed byte.

**Erase**  
Will erase the chip. All memory locations will be set to 255.

**Program**  
Will program the chip with the current program.  
If EEPROM-memory is selected, you will be asked for a filename.

**Verify**  
Will verify the current program with the target chip.

**Read**  
Will read the target chip and saves the result to a file.

**Set lockbit**  
Will set the selected lock bits.  
You must select the lock bits first. The lock bits to set depend on the selected target chip.
Auto erase
When this checkbox is selected, the target chip will be erased before it will be programmed.

Auto verify
When this checkbox is selected, the result will be verified after each programming.

9.6 SE512 or SE514 programmer
The SE512 and SE514 are parallel printer port based programmers. The nice thing about these programmers is that they can simulate the application too. This has the advantage that no device swapping is needed until your application works like you want. The SE512 can program the AT89C1051 to AT89C4051. The SE514 can program larger chips too.

Buffer clear
Will clear the buffer.

Buffer load from file
With this option you can load a file into the buffer. By default the current program is loaded into the buffer.

Buffer save to file
With this option you can save the buffer to a binary file.

Chip Write buffer into chip
With this option you program the chip.

Chip Read chipcode into buffer
This option will read the target device its memory into the buffer.

Chip Blank check
Performs a blank check on the target device. A chip is considered blank if every memory location contains 255 (FF hex)

Chip Erase
Will erase the target chip.

Chip verify
Will verify the buffer with the chipcontent.

Chip autoprogram
Will erase, program and verify the chip.

Note that the targetchip will be detected automatic. When the targetchip can't be detected, the menu options will not work.
9.7 SE-812

The SE-812 from Sample Electronics is a programmer for the aduc812. The programmer is well suited for in circuit programming.

Since it is a serial programmer that operates via the COM port, the programming is done with the terminal emulator. When you select the SE812 from the programmer options there will be an additional menu in the terminal emulator.

- Erase chip. This option will erase both the code flash and the EEPROM.
- Erase code flash. This option will erase only the code flash memory.
- Program chip. This will program the chip with the current program.
- Auto program. This will erase the chip and program the chip.

The programmer works only with version 2.00 of the boot loader.

9.8 Sample Electronics ISP 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. And only a few resistors are needed. The operation is the same a for the STK200/300 programmer.

What you need is a DB25 centronics male connector, a flat cable and a connector that can be connected to the target MCU board.

The connections to make are as following:

<table>
<thead>
<tr>
<th>DB25 pin</th>
<th>Target MCU pin(AT89S8252)</th>
<th>DT104</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, D0</td>
<td>MOSI, pin 6</td>
<td>J5, pin 4</td>
</tr>
<tr>
<td>4, D2</td>
<td>RESET, pin 9</td>
<td>J5, pin 8</td>
</tr>
<tr>
<td>5, D3</td>
<td>CLOCK, pin 8</td>
<td>J5, pin 6</td>
</tr>
<tr>
<td>11, BUSY</td>
<td>MISO, pin 7</td>
<td>J5, pin 5</td>
</tr>
<tr>
<td>18-25,GND</td>
<td>GROUND</td>
<td>J5, pin 1</td>
</tr>
</tbody>
</table>

The MCU pin numbers are shown for an 8252!

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 an 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!
The following picture shows the connections to make. Both a setup for the DT104 and stand alone PCB are shown.

I received the following useful information:

Hi Mark,

I have been having spurious success with the simple cable programmer from Sample Electronics for the AVR series.

After resorting to hooking up the CRO I have figured it out (I think). When trying to identify the chip, no response on the MISO pin indicates that the Programming Enable command has not been correctly received by the target. The SCK line Mark/Space times were okay but it looked a bit sad with a slow rise time but a rapid fall time. So I initially tried to improve the rise time with a pullup. No change ie still could not identify chip. I was about to add some buffers when I came across an Atmel app note for their serial programmer

"During this first phase of the programming cycle, keeping the SCK line free from pulses is critical, as pulses will cause the target AVR to loose synchronisation with the programmer. When synchronisation is lost, the only means of regaining synchronisation is to release the RESET line for more than 100ms."

I have added a 100pF cap from SCK to GND and works first time every time now. The SCK rise time is still sad but there must have been enough noise to corrupt the initial command despite using a 600mm shielded cable.

This may be useful to your users.
9.9 CYGNAL JTAG Programmer

The CYGNAL JTAG programmer comes with the CYGNAL development kit and is also available from www.sample.co.kr. All tests were performed with the programmer/evaluation board from Sample Electronics.

The Cygnal JTAG programmer is controlled by a COM port. You need to select a free COM port of your PC that is connected to the programmer.

When you program the cygnal chip BASCOM will erase and program the chip.

9.10 Futurelec

The Futurelec programmer from www.futurlec.com is an ISP programmer for the AT89S8252. All tests are performed with the AT89S8252 board from Futurelec Electronics.

9.11 JPK Systems X-programmer

The JPK Systems X-programmer is a serial comport based SPI-programmer. It is fully optical isolated and so an ideal device for industrial equipment. It supports AVR chips too, but these aren't supported in BASCOM of course so there is only support for the 89S8252 and the 89S53.

Since it is serial based, the support is placed in the terminal emulator. After selecting the JPK programmer, there will be additional menu options available in the terminal emulator. All these options can be found under the JPK menu. The transfer between the PC and the programmer is implemented with the X-modem CRC protocol.

Select device
Use this option to select the target device. You can choose between the 89S8252 and the 89S53.

Erase
Erase the target chip.

Read code
Will read the codememory from the chip. You will be asked for a filename first.

Program chip
Supported Programmers

Will program the targetchip with the current program.

**Set lockbits**
Will set the lockbits of the targetchip. All lockbits will be set.

**Read EEPROM**
Will save the EEPROM data into a file. This only applies to the AT89S8252.

**Write EEPROM**
Will program the EEPROM with a file. This only applies to the AT89S8252.

Of course all commands can be typed manually too, but you must set the terminal emulator communication settings to 2400N82 in that case.

### 9.12 Peter Averill's TAFE programmer

The TAFE flashprogrammer is a parallel printer port based programmer and can be build with the DT004 and DT206 SimmSticks from Dontronics. The programmer can program only AT89C1051 to AT89C4051 chips.

Peter also has schematics available on the web so you can build your own PCB. The programmer supports all the usual features except the 'read signature' feature. That's is why you have to select the used chip yourself from the mnu.

The programmer uses the same interface as the MCS Flashprogrammer, so for a description read the [MCS Flashprogrammer](#) help.

I got some feedback from a user that had problems with his programmer. he added 5K1 pullup resistors to +5V. This is shown in the picture below. The dots (11) must each have a resistor of 5K1 to +5V.

![Schematic Diagram](image-url)
9.13 STK200/300 ISP Programmer

The STK200 and STK300 are AR starter kits from Atmel. They come with a parallel printer port programmer dongle for in system programming of the chips. This dongle can be used to program the 89S8252 or 89S53.

For those who don't have this kit and the programmer the following schematic shows how to make your own programmer:

![Schematic Diagram]

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!
MCS also sells a compatible dongle.
The following screen will pop up when you have selected this programmer:
You must select the chip you use. By selecting the FlashROM TAB or the EEPROM TAB you can write that info to the chip. When the chip does not have EEPROM memory, the EEPROM TAB will not be visible.

When the chip such as the 89S8253, 89S2051 or 89S4051 has USER data, an additional TAB will be shown.

This is intended to read/write the user data.

When you select auto Flash, pressing F4 from the IDE will program the chip automatic and the window will not be displayed.

When Code + Data is selected from the programmer options both the Code and the EEPROM data are programmed.

The STK200/300 is recommended for new programmers.

### 9.14 Rhombus SCE-51

Rhombus developed the SCE-51. A powerful small 8051 micro processor board with on board RAM and FLASHROM and bootloader.

In addition the board serves as an in circuit emulator.

Transferring your program to RAM goes very fast. Faster than loading it into the traditional FLASHROM. So during debugging it is well suited for debugging large applications.

When you select the SCE-51, the following window will appear when you press F4.
The filename is automatic filled.
The original SCE-51 software from Rhombus has much more options and BASCOM only supports programming to RAM and FLASH.

You must select the target memory before you click the Program button.

By clicking the Erase button you can erase the memory.
During programming a status bar will be shown.

The baud rate is fixed to 19200 baud. Support for 115200 baud will be added later.

9.15 **SE511-SE516 programmer**

The SE511-SE516 can be used for the SE511 and SE516 programmers from Sample Electronics.
These programmers are serial programmers. They require a COM port.
When you launch the programmer, the current program will be loaded into the memory. You can also use the LOAD button to load a program into the buffer.

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>This button will reset the programmer and will determine the used chip.</td>
</tr>
<tr>
<td>Load</td>
<td>Load a binary or Intel HEX file into the buffer</td>
</tr>
<tr>
<td>Save</td>
<td>Save the current buffer to file</td>
</tr>
<tr>
<td>Read</td>
<td>Read the chip flash content</td>
</tr>
<tr>
<td>Blank</td>
<td>Test if a chip is blank</td>
</tr>
<tr>
<td>Erase</td>
<td>Erase (blank) a chip</td>
</tr>
<tr>
<td>Write</td>
<td>Write(program) the buffer into the chip</td>
</tr>
<tr>
<td>Verify</td>
<td>Verify if the buffer is the same as the chip content</td>
</tr>
<tr>
<td>Lock Bits</td>
<td>Write the selected lock bits</td>
</tr>
</tbody>
</table>
Part X
10 BASCOM Misc

10.1 Error messages

The following table lists all errors that can occur.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BASIC source file not found</td>
</tr>
<tr>
<td>2</td>
<td>Code does not fit into FLASHROM</td>
</tr>
<tr>
<td>3</td>
<td>Unknown statement</td>
</tr>
<tr>
<td>4</td>
<td>Extension expected</td>
</tr>
<tr>
<td>5</td>
<td>Wrong variable or variable not dimensioned</td>
</tr>
<tr>
<td>6</td>
<td>Two parameters expected</td>
</tr>
<tr>
<td>7</td>
<td>No more space for BIT</td>
</tr>
<tr>
<td>8</td>
<td>No more space for BYTE</td>
</tr>
<tr>
<td>9</td>
<td>No more space for INTEGER/WORD</td>
</tr>
<tr>
<td>10</td>
<td>Wrong type (BIT,BYTE or INTEGER/WORD) expected</td>
</tr>
<tr>
<td>11</td>
<td>AS expected by DIM</td>
</tr>
<tr>
<td>12</td>
<td>, expected</td>
</tr>
<tr>
<td>13</td>
<td>Unknown interrupt</td>
</tr>
<tr>
<td>14</td>
<td>IF THEN expected</td>
</tr>
<tr>
<td>15</td>
<td>FOR, DO or WHILE expected</td>
</tr>
<tr>
<td>16</td>
<td>Wrong number of parameters</td>
</tr>
</tbody>
</table>
| 17 | Illegal compare (=,>,<,<>),<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>,<,>
<table>
<thead>
<tr>
<th>Line</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>139</td>
<td>9 parameters expected</td>
</tr>
<tr>
<td>140</td>
<td>COUNTER0 or COUNTER1 expected.</td>
</tr>
<tr>
<td>141</td>
<td>= expected.</td>
</tr>
<tr>
<td>142</td>
<td>Maximum of 128 aliases statements allowed</td>
</tr>
<tr>
<td>143</td>
<td>Duplicate label</td>
</tr>
<tr>
<td>144</td>
<td>Value does not fit into byte</td>
</tr>
<tr>
<td>145</td>
<td>No more space for external BYTE</td>
</tr>
<tr>
<td>146</td>
<td>No more space for external INTEGER/WORD</td>
</tr>
<tr>
<td>147</td>
<td>No more space for STRING</td>
</tr>
<tr>
<td>148</td>
<td>Call outside 2048 page range. Use $LARGE to compile this program.</td>
</tr>
<tr>
<td>150</td>
<td>Unsupported LCD display</td>
</tr>
<tr>
<td>151</td>
<td>Unsupported mode</td>
</tr>
<tr>
<td>152</td>
<td>Variable not found or dimensioned</td>
</tr>
<tr>
<td>153</td>
<td>Wrong type (BYTE,INTEGER/WORD, LONG or STRING) expected</td>
</tr>
<tr>
<td>154</td>
<td>; expected</td>
</tr>
<tr>
<td>155</td>
<td>SELECT CASE expected</td>
</tr>
<tr>
<td>156</td>
<td>Numeric variable expected</td>
</tr>
<tr>
<td>157</td>
<td>(external) LONG expected</td>
</tr>
<tr>
<td>158</td>
<td>Value does not fit into Integer</td>
</tr>
<tr>
<td>159</td>
<td>Value does not fit into Word</td>
</tr>
<tr>
<td>160</td>
<td>Value does not fit into Long</td>
</tr>
<tr>
<td>161</td>
<td>* xxx (xxx=length) expected</td>
</tr>
<tr>
<td>162</td>
<td>Variable expected</td>
</tr>
<tr>
<td>163</td>
<td>Small string expected.</td>
</tr>
<tr>
<td>164</td>
<td>Variable not DIMensioned</td>
</tr>
<tr>
<td>166</td>
<td>Three parameters expected</td>
</tr>
<tr>
<td>167</td>
<td>1 or 0 expected</td>
</tr>
<tr>
<td>168</td>
<td>4 or 8 expected</td>
</tr>
<tr>
<td>170</td>
<td>Wrong value for WATCHDOG</td>
</tr>
<tr>
<td>171</td>
<td>Wrong parameter for I2C</td>
</tr>
<tr>
<td>172</td>
<td>Byte,Integer or Long expected</td>
</tr>
<tr>
<td>173</td>
<td>Variable expected</td>
</tr>
<tr>
<td>174</td>
<td>Integer or Long expected</td>
</tr>
<tr>
<td>175</td>
<td>Value does not fit into bit</td>
</tr>
<tr>
<td>176</td>
<td>Variables must be of the same type</td>
</tr>
<tr>
<td>Page</td>
<td>Message</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>177</td>
<td>Illegal operation</td>
</tr>
<tr>
<td>178</td>
<td>Value doesn't fit</td>
</tr>
<tr>
<td>179</td>
<td>Not supported</td>
</tr>
<tr>
<td>180</td>
<td>Illegal operation in PlaceValue</td>
</tr>
<tr>
<td>181</td>
<td>Constant or Internal byte or integer expected for index</td>
</tr>
<tr>
<td>182</td>
<td>Invalid device</td>
</tr>
<tr>
<td>183</td>
<td>Channel not opened</td>
</tr>
<tr>
<td>184</td>
<td>Device already open</td>
</tr>
<tr>
<td>185</td>
<td>Device was not open</td>
</tr>
<tr>
<td>186</td>
<td>Value does not fit into byte</td>
</tr>
<tr>
<td>187</td>
<td>IF ... THEN not allowed on same line as CASE</td>
</tr>
<tr>
<td>188</td>
<td>END IF expected</td>
</tr>
<tr>
<td>189</td>
<td>CONST expected</td>
</tr>
<tr>
<td>190</td>
<td>Channel expected (#x)</td>
</tr>
<tr>
<td>191</td>
<td>ALIAS already used</td>
</tr>
<tr>
<td>192</td>
<td>Word or Integer expected</td>
</tr>
<tr>
<td>193</td>
<td>CONST already defined</td>
</tr>
<tr>
<td>194</td>
<td>= expected</td>
</tr>
<tr>
<td>195</td>
<td>TO expected</td>
</tr>
<tr>
<td>196</td>
<td>Jump out of address range</td>
</tr>
<tr>
<td>197</td>
<td>RNDDATA variable not dimensioned</td>
</tr>
<tr>
<td>198</td>
<td>) expected</td>
</tr>
<tr>
<td>199</td>
<td>( expected</td>
</tr>
<tr>
<td>206</td>
<td>Library file not found</td>
</tr>
<tr>
<td>207</td>
<td>Library file already registered</td>
</tr>
<tr>
<td>208</td>
<td>) expected</td>
</tr>
<tr>
<td>209</td>
<td>( expected</td>
</tr>
<tr>
<td>210</td>
<td>LEFT or RIGHT expected</td>
</tr>
<tr>
<td>211</td>
<td>External routine not found</td>
</tr>
<tr>
<td>212</td>
<td>Valid number must be in range from 1-16</td>
</tr>
<tr>
<td>213</td>
<td>Numeric constant expected</td>
</tr>
<tr>
<td>214</td>
<td>No SUB found.</td>
</tr>
<tr>
<td>215</td>
<td>Already in SUB</td>
</tr>
<tr>
<td>216</td>
<td>Wrong mode</td>
</tr>
<tr>
<td>217</td>
<td>NOINT expected</td>
</tr>
<tr>
<td>218</td>
<td>+ must be between {}</td>
</tr>
<tr>
<td>219</td>
<td>Address &gt;127, use indirect addressing</td>
</tr>
</tbody>
</table>
10.2 Compiler Limits

There are some limitations to the compiler:
You can perform only one calculation in a formula.

\[
a = a \times b_1 \quad \text{Good} \\
a = a \times b_1 + c \quad \text{False}
\]

<table>
<thead>
<tr>
<th>Maximum allowed labels</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum allowed variable names</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum number of INTEGER/WORD variables</td>
<td>10*</td>
</tr>
<tr>
<td>Maximum number of BYTE variables</td>
<td>20*</td>
</tr>
<tr>
<td>Maximum number of BIT variables</td>
<td>120*</td>
</tr>
<tr>
<td>Maximum number of STRING variables</td>
<td>Up to available external memory</td>
</tr>
<tr>
<td>Maximum number of ALIAS statements</td>
<td>128</td>
</tr>
</tbody>
</table>

*Depending on the used statements and the used variables of the other types.

A maximum of 32 bytes is used internally. This depends on the used statements. The stack uses some space too. So it depends on the used statements how much variables you can use. In the worst case \((32+16+8) = 56\) bytes are used.

You can find out by viewing the report file how much bytes are used by your program.

When you have a micro such as the 89S8252 with 256 bytes of internal memory, you can have more variables.

8 used bit vars will use 1 byte;
1 used byte will use 1 byte;
1 used integer/word will use 2 bytes;
1 used long will use 4 bytes;
1 used single will use 4 bytes;
1 string with a length of 10 bytes will use 11 bytes.

Maximum nesting:

\[
\begin{array}{|c|c|}
\hline
\text{FOR .. NEXT} & 50 \\
\text{IF .. THEN} & 50 \\
\text{DO .. LOOP} & 50 \\
\text{WHILE .. WEND} & 50 \\
\text{SELECT .. CASE} & 25 \\
\hline
\end{array}
\]
10.3 Reserved Words

The following table shows the reserved BASCOM statements. **Red** keywords can only be used on systems, which can address external RAM memory.

!  
;  
$INCLUDE  
$NOINIT  
$NOSP  
$NOBREAK  
$BAUD  
$BGF  
$DEFAULT  
$CRYSTAL  
$LARGE  
$LCD  
$ROMSTART  
$RAMSIZE  
$RAMSTART  
$SERIALINPUT  
$SERIALOUTPUT  
$SIM  
1WRESET  
1WREAD  
1WWRITE  
ACK  
ALIAS  
ABS()  
AND  
AS  
ASC()  
BAUD  
BCD()  
BIT  
BITWAIT  
BLINK  
BOOLEAN  
BREAK  
BYTE  
CALL  
CASE  
CLS  
CHR()  
CONFIG  
CONST  
COUNTER  
COUNTER0  
COUNTER1  
CPEEK()  
CURSOR  
DATA  
DEC  
DECLARE  
DEFBIT  
DEBYTE
DEFLCDCHAR
DEFINT
DEFWORD
DELAY
DIM
DISABLE
DISPLAY
DO
DOWNTO
ELSE
ELSEIF
ENABLE
END
ERR
EXIT
EXTERNAL
FOR
FOURTH
FOURTHLINE
GATE
GETAD
GOSUB
GOTO
HEXVAL()
HIGH()
HIGHW()
HOME
I2CRECEIVE
I2CSEND
I2CSTART
I2CSTOP
I2CRBYTE
I2CWBYTE
IDLE
IF
INC
INKEY
INP()
INPUT
INPUTHEX
INT0
INT1
INTEGER
INTERNAL
IS
LCD
LCDHEX
LEFT
LEFT()
LOAD
LOCATE
LONG
LOOKUP
LOOP
LOW()
LOWW()
LOWER
LOWERLINE
MAKEBCD()
MAKEDEC()
MAKEINT()
MID()
MOD
MODE
NACK
NEXT
NOBLINK
NOSAVE
NOT
OFF
ON
OR
OUT
PO-0-P6
PEEK()
POKE
POWERDOWN
PSET
PRINT
PRINTHEX
PRIORITY
READ
READEEPROM
REM
RESET
RESTORE
RETURN
RIGHT
RIGHT()
RND()
ROTATE
SELECT
SERIAL
SET
SHIFT
SHIFTLCD
SHIFTCURSOR
SHIFTIN
SHIFTOUT
SHOWPIC
SOUND
SPACE()
START
STEP
STR()
STRING()
STOP
STOP TIMER
SUB
SWAP
THEN
THIRD
THIRDLINE
TIMEOUT
TIMERO
TIMER1
TO
UNTIL
UPPER
UPPERLINE
VAL()
WAIT
WAITKEY
WAITMS
WATCHDOG
WRITEEEPROM
WEND
WHILE
WORD
XOR
XRAM

The internal registers are also reserved words (variables)

TCON
P1
SCON
IE
P3
IP
PSW
ACC
B
SP
DPL
DPH
PCON
TMOD
TL0
TL1
TH0
TH1
SBUF

Note that you can change the internal registers with the Register File settings from the Options menu.
Part XI
11 Microprocessor support

11.1 Microprocessor support

Some microprocessors have additional features compared to the AT89C2051/8051.

8032/8052/AT89S8252

TIMER2

AT89S8252

WATCHDOG
DATA EEPROM
Alternative port-pin functions

80515, 80535, 80517, 80535

GETAD
WATCHDOG
BAUDRATE GENERATOR
INTERRUPTS and PRIORITY

80517, 80537

GETAD
WATCHDOG
BAUDRATE GENERATOR
BAUDRATE GENERATOR1
INTERRUPTS and PRIORITY

89C51+

WATCHDOG
PRIORITY

ADUC812

CONFIG ADUC812
Using the DAC that also contains an example
The additional interrupts are:
ADCI, I2CSPI and PSMI

To enable them:
ENABLE ADCI, ENABLE I2CSPI, ENABLE PSMI

To disable them:
DISABLE ADCI, DISABLE I2CSPI, DISABLE PSMI

To set the priority to the highest level in addition to the normal priority interrupt sources:
PRIORITY SET|RESET ADCI
PRIORITY SET|RESET I2CSPI

80552
GETAD(channel, prm) where channel is the channel and the prm is a parameter that may be 0 for software trigger only or 32 (dec) for trigger by rising edge on STADC too.

To use the **PWM** of the 80552:

```vbnet
Dim Pwp As Byte, Pwa as Byte, Pwb as Byte
Pwp = 200      'set output frequency (0 - 255)
Pwa = 50        'set channel 0 (a) pulse width (0 - 255)
Pwb = 0         'set channel 1 (b) pulse width (0 - 255)
Do
  Gosub Pwm
Loop
Pwm:
$asm
  MOV  PWMP , {Pwp}
  MOV  PWM0 , {Pwa}
  MOV  PWM1 , {Pwb}
$end asm
Return
```

### 11.2 TIMER2

Some microprocessors have an additional timer on board: TIMER2. This section describes the 8032 compatible TIMER2 and is not compatible with the TIMER2 found in the 80C535 and others. TIMER2 is a 16-bit timer/counter which can operate as either an event timer or an event counter. TIMER2 has three main operating modes: capture, auto-reload (up or down counting), and baud rate generator.

When using the TIMER2 interrupt, you must reset the interrupt bit that caused the interrupt yourself in the ISR handler.

**Capture mode**

In the capture mode there are two options:

- 16-bit timer/counter which upon overflowing sets bit TF2, the TIMER2 overflow bit. This bit can be used to generate an interrupt.

Counter mode:

```
CONFIG TIMER2 = COUNTER, GATE = INTERNAL, MODE = 1
```

Timer mode:

```
CONFIG TIMER2 = TIMER, GATE = INTERNAL, MODE = 1
```

- As above but with the added future that a 1 to 0 transition on at external input T2EX causes the current values in the TIMER2 registers TL2 and TH2 to be captured into the capture registers RCAP2L and RCAP2H.

Counter mode:

```
CONFIG TIMER2 = COUNTER, GATE = EXTERNAL, MODE = 1
```
Timer mode:
CONFIG TIMER2=TIMER,GATE=EXTERNAL,MODE=1
In addition the transition at T2EX causes bit EXF2 in T2CON to be set and EXF2 like TF2 can generate an interrupt.

The TIMER2 interrupt routine can interrogate TF2 and EXF2 to determine which event caused the interrupt.
(there is no reload value in this mode. Even when a capture event occurs from T2EX the counter keeps on counting T2EX pin transitions or osc/12 pulses)

**Auto reload mode**
In the 16-bit auto reload mode, TIMER2 can be configured as a timer or counter which can be programmed to count up or down. The counting direction is determined by bit DCEN.
TIMER2 will default to counting up to &HFFFFF and sets the TF2 overflow flag bit upon overflow. This causes the TIMER2 registers to be reloaded with the 16-bit value in RCAP2L and RCAP2H.
The values in RCAP2L and RCAP2H are preset by software means.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

Timer mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

If EXEN2=1 then a 16-bit reload can be triggered either by an overflow or by a 1 to 0 transition at input T2EX. This transition also sets the EXF2 bit. The TIMER2 interrupt, if enabled, can be generated when either TF2 or EXF2 are 1.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=EXTERNAL,MODE=0

Timer mode:
CONFIG TIMER2=TIMER,GATE=EXTERNAL,MODE=0
TIMER2 can also count up or down. This mode allows pin T2EX to control the direction of count. When a logic 1 is applied at pin T2EX TIMER2 will count up.
TIMER2 will overflow at &HFFFFF and sets the TF2 flag, which can then generate an interrupt, if the interrupt is enabled. This timer overflow also causes the 16-bit value in RCAP2L en RCAP2H to be reloaded in to the timer registers TL2 and TH2.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP

Timer mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP

A logic 0 applied at pin T2EX causes TIMER2 to count down. The timer will under flow when TL2 and TH2 become equal to the value stored in RCAP2L and RCAP2H. TIMER2 under flows sets the TF2 flag and causes &HFFFFF to be reloaded into the timer registers TL2 and TH2.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,
DIRECTION=DOWN

Timer mode:
The external flag TF2 toggles when TIMER2 underflows or overflows. The EXF2 flag does not generate an interrupt in counter UP/DOWN mode.

**Baud rate generator**
This mode can be used to generate a baud rate for the serial port. TIMER1 can be used for an other task this way.

CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=2

**Receive only**
This mode can be used to generate the baudrate for the receiver only. TIMER1 can be used for the transmission with another baudrate.

CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=3

Note that TIMER1 must be setup from assembler this way.

**Transmit only**
This mode can be used to generate the baud rate for transmitter only. TIMER1 can be used for the reception with another baudrate.

CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=4

Note that TIMER1 must be setup from assembler this way.

**Clock output**
Some 8052 deviants have the ability to generate a 50% duty cycle clock on P1.0.

CONFIG TIMER2=TIMER,MODE=5

The output frequency = \( \frac{f_{OSC}}{4} / (65536 - \text{CAPTURE}) \)

Use \( \text{CAPTURE} = \text{value} \) to set the capture register.

**How to determine what caused the interrupt**
You can test the bit T2CON.7 to see if an overflow caused the interrupt. You can test bit T2CON.6 whether either a reload or capture is caused by a negative transition on T2EX.

Timer2_ISR:
If T2CON.7 = 1 Then
  Print "Timer overflowed"
  Reset T2con.7
Else
  If T2CON.6 = 1 Then
    Print "External transition"
    Reset t2con.6
  End if
End If
Return
11.3 DATA EEPROM

The AT89S8252 has a built in 2Kbytes flash EEPROM. You can use this to store data. Two statements are provided: WRITEEEPROM and READEEPROM.

**WRITEEEPROM** var [,, address ]

<table>
<thead>
<tr>
<th>var</th>
<th>Any BASCOM variable name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>The address of the EEPROM where to write the data to.</td>
</tr>
<tr>
<td></td>
<td>Ranges from 0 to 2047.</td>
</tr>
<tr>
<td></td>
<td>When you omit the address the address will be assigned automatically. You can view the assigned address in the report file.</td>
</tr>
</tbody>
</table>

**READEEPROM** var [,, address ]

<table>
<thead>
<tr>
<th>var</th>
<th>Any BASCOM variable name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>The address of the EEPROM where to read the data from.</td>
</tr>
<tr>
<td></td>
<td>Ranges from 0 to 2047.</td>
</tr>
<tr>
<td></td>
<td>You can omit the address when you have written a value before with the WRITEEEPROM var statement.</td>
</tr>
<tr>
<td></td>
<td>Because in that case the compiler knows about the address because it is assigned by the compiler.</td>
</tr>
</tbody>
</table>

**Example**

```bascom
Dim S As String * 15, S2 As String * 10
S = "Hello", S2 = "test"

Dim L As Long
L = 12345678
Writeeeprom S
Writeeeprom S2 'write strings
Writeeeprom L 'write long

S = "", S2 = "", L = 0 'clear variables
Readeeprom L : Print L
Readeeprom S : Print S
Readeeprom S2 : Print S2
End
```

11.4 AT898252 WATCHDOG

The AT89S8252 has a built in watchdog timer. A watchdog timer is a timer that will reset the uP when it reaches a certain value. So during program execution this WD-timer must be reset before it exceeds its maximum value. This is used to be sure a program is running correct. When a program crashes or sits in an endless loop it will not reset the WD-timer so an automatic reset will occur resulting in a restart.

**START WATCHDOG** will start the watchdog timer.
**STOP WATCHDOG** will stop the watchdog timer.
**RESET WATCHDOG** will reset the watchdog timer.
See also

CONFIG WATCHDOG

Example

'-----------------------------------------------------
' (c) 1998 MCS Electronics
' WATCHD.BAS demonstrates the AT89S8252 watchdog timer
' select 89s8252.dat !!!
'-----------------------------------------------------
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

11.5 WATCHDOG 80515

The 80515 and 80535 both have a WD-timer. This is a 16 bit timer that can't be stopped!
It will reset the system after 65535 uS at 12MHz.

START WATCHDOG  'start the WD-timer.
RESET WATCHDOG  'will reset the WD-timer.

11.6 INTERRUPTS and PRIORITY 80515

The 80515, 80535, 80517 and 80537 have more interrupt sources and priority is handled different compared to the 8051.

Enable interrupts:
ENABLE AD 'AD converter
ENABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
ENABLE TIMER2EX 'timer2 external reload

Disable interrupts:
DISABLE AD 'AD converter
DISABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
DISABLE TIMER2EX 'timer2 external reload

Selecting of priority:
PRIORITY SET|RESET  source , level
level can be 0,1,2 or 3.(0=lowest,3=highest)

The source can be :
INTO/ADC
TIMER0/INT2
INTO/INT3
Note that only one of the pairs must be selected.
PRIORITY SET INT4,3 'will set INT4 to the highest priority.
When two ints occur with the same priority the first source in the list will be handled first. So when both TIMER1 and INT4 have the same priority, TIMER1 will be serviced first. Look at a datasheet for more details.

11.7 INTERRUPTS and PRIORITY 80537

The 80517 and 80537 have more interrupts and priority is handled different compared to the 8051.

Enable interrupts:
ENABLE AD 'AD converter
ENABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
ENABLE TIMER2EX 'timer2 external reload
ENABLE CTF 'compare timer interrupt
ENABLE SERIAL1 'serial1 interrupt

Disable interrupts:
DISABLE AD 'AD converter
DISABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
DISABLE TIMER2EX 'timer2 external reload
DISABLE CTF 'compare timer interrupt
DISABLE SERIAL1 'serial1 interrupt

Selecting of priority:
PRIORITY SET|RESET source , level
level can be 0,1,2 or 3.(0=lowest,3=highest)

source can be :
INTO/ADC/Serial1
TIMER0/INT2
INTO/INT3
TIMER1/CTF/INT4
Serial/INT5
TIMER2/INT6

Note that only one of the TRIPLE-pairs must be selected.
PRIORITY SET INT4,3 'will set INT4 to the highest priority.
When two ints occur with the same priority the first source in the list will be handled first. So when both TIMER1 and INT4 have the same priority, TIMER1 will be serviced first.
Look at a datasheet for more details.

11.8 ADUC 812

The 812 has 2 DACS named DAC0 and DAC1.

You can use the CONFIG ADUC812 statement to set the DAC behaviour.
The DAC can be powered on or off.
DAC0.POWEROFF will power off the DAC0
DAC1.POWERON will power on the DAC1

To force the output of the DAC to 0 volt use:
DAC0.CLEAR

To let it output the voltage use:
DAC0.NORMAL

The DAC values can be written with the following statements:
DAC0.value = 1024 'or a variable
DAC1.value = word

The sync bit is reset and to sync the DAC with the supplied values use:

DAC.SYNC
Note that the SYNC method operates on both DACs and so there is no 0 or 1 specified!

All the previous methods shown can work with 0 for DAC0 or 1 for DAC1.

See the audc812.bas example:

'-------------------------------------------------------------------------------------
' ADCU812.bas (c) 2000 MCS Electronics
' Note that the support for this chip is untested
' Any feedback appreciated!
'-------------------------------------------------------------------------------------
'Use this dat file
$regfile = "812.dat"

'configure ADC
Config Audc812 = Adcon , Mode = Normal , Clock = 1 , Acquisition = 1 , Timer2 = Disabled , Extrig = Disabled

'configure DACS
Config Audc812 = Dac , Mode = 12 , Range1 = Vref , Range0 = Vref , Clear0 = False , Sync = Enabled , Power0 = On , Power1 = Off

Declare Sub Write_ebyte
Declare Sub Read_ebyte

'dim variables
Dim Wdac As Word
Dim Adc As Word

Dim Eeadr As Word, Eebyte As Byte, Page As Word
'get value from adc channel 0
'note that simulator will halt until you make the adcon2 bit 4 zero.
Adc = Getad(0)

'enable dac0 by powering it on
Dac0.poweron

'0V to output of dac0
Dac0.clear

'put voltage into dacs
Dac0.value = 12
Dac1.value = 500

'dac0 was 0V but must work normal now
Dac0.normal

'and after setting the value(s) the dacs must be updated with the sync method
Dac.sync

'the EEPROM is accessed via pages
'each page is 4 bytes
'to write 1 byte you need to write the whole 4 byte page
'assign eeadr with the address
'and eebyte with the value to write
Eeadr = 100 : Eebyte = 5 : Call Write_ebyte

Eeadr = 100 : Call Read_ebyte
Print Eebyte
End

Sub Write_ebyte
    Page = Eeadr \ 4                'page
    mov edarl, {page}             ; page address
    mov econ, #1                  ; read 4 current bytes
    mov econ, #5                  ; erase page
    Waitms 20                     'wait 20 msecs
    Page = Page * 4
    Page = Eeadr - Page
    If Page = 0 Then
mov edata1,{ebyte} ; data register to write
Elseif Page = 1 Then
    mov edata2,{ebyte} ; data register to write
Elseif Page = 2 Then
    mov edata3,{ebyte} ; data register to write
Else
    'must be 3
    mov edata4,{ebyte} ; data register to write
End If
mov econ,#2 ; write registers
End Sub

Sub Read_ebyte
    Page = Eadr \ 4
    'page
    mov edarl,{page} ; page address
    mov econ,#1 ; read 4 current bytes
    Page = Page * 4
    Page = Eadr - Page
    If Page = 0 Then
        mov {EEbyte},edata1 ; data register to read
    Elseif Page = 1 Then
        mov {ebyte},edata2 ; data register to read
    Elseif Page = 2 Then
        mov {ebyte},edata3 ; data register to read
    Else
        'must be 3
        mov {ebyte},edata4 ; data register to read
    End If
mov econ,#2 ; write registers
End Sub
End

11.9 89C51

The 89C51 has an additional PCA interrupt. The priority mechanism is also different compared to a normal 8051. You can set a level in the range from 0-3. PRIORITY SET|RESET source , level
level can be 0, 1, 2 or 3. (0=lowest, 3=highest)

The source can be:
INT0
TIMER0
INT1
TIMER1
SERIAL
TIMER2
PCA
PRIORITY SET INT0,3 'will set INT0 to the highest priority. Look at a datasheet for more details.

The WATCHDOG can be started with the statement:
START WATCHDOG.
RESET WATCHDOG must be used in your program to reset the WD-timer. When it reaches 16384 the chip will be reset. The input to the WD-timer is the XTAL frequency!
Part XII
12 International Resellers

12.1 International Resellers

The list with resellers is updated frequently. Please look at the resellers list at the MCS website:

http://www.mcselec.com/index.php?option=com_contact&catid=82&Itemid=59
Part XIII
13 Third party hardware

13.1 Third party Hardware

There is a lot of third party hardware available. Below you find links to some of the available hardware

Grifo, boards for BASCOM-AVR, BASCOM-8051 and BASCOM-LT

Rhombus SCE-51, small 8051 board and in circuit emulator

13.1.1 Grifo

EXAMPLES
BASCOM - BASIC

The content of this page is provided by Grifo.

As following you can find a wide range of demo programs. The programs have been realized to be used on a well-known hardware, as the KS1-AVR or the DEB-01, etc. in order to avoid any doubts about the interpretation of the results.

The demo programs are well documented in order to allow a fast approach for anybody. In addition to that, being the same demos written in different languages, it is possible to get an efficient comparison both for Quality and Speed terms.

INDEX

BASIC

• Examples - \BASCOM-LT
SHORT PROGRAM DESCRIPTION

x_AD11
This program monitors one analogic channel out of eleven, managed by IC12 (TLC2543), visualization of the channel is in hexadecimal format, through T1 and T2 the channel to convert is selected, T1 increments while T2 decrements.
The display shows first the channel being converted, then the 12 bits wide hexadecimal value of the channel converted: Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.
For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

x_AD4
This program monitors one analogue channel out of four, managed by IC12 (PCF8591), visualization of the channel is in hexadecimal format, through T1 the channel to convert is selected: Whenever a key is pressed, an acoustic signal is emitted.
Display DY1 shows the channel to convert, while displays DY3 and DY4 show the converted value in HEX.
Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.
For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

x_DA
This program monitors one D/A converter channel on IC2 (PCF8591), key T1 increments the value, while key T2 decrements the value which is shown in hexadecimal format the 7 segments displays.
Whenever a key is pressed, an acoustic signal is emitted.
Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.
For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

x_REE
This program allows to read a serial EEPROM on IC4 (max 24c08), with addresses ranging from &H400 to &H7ff, addresses from &H0 to &H0FF are taken by IC7 (RTC PCF8583) while addresses from &H100 to &H3FF are free space.
At start the program shows the address where to write, through keys T1 and T2 the value in incremented or decremented.
Through key T3 the address is accepted and the value read at such address is shown.
Whenever a key is pressed, an acoustic signal is emitted.
Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.
For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

**x_WEE**

This program allows to write to a serial EEPROM on IC4 (max 24c08), with addresses ranging from &H400 to &H7ff, addresses from &H00 to &H0FF are taken by IC7 (RTC PCF8583) while addresses from &H100 to &H3FF are free space.

At start the program shows the address where to write, through keys T1 and T2 the value in incremented or decremented.

Through key T3 the address is accepted, then the value to write is selected through T1 and T2, as last press key T3 to write.

Whenever a key is pressed, an acoustic signal is emitted.

After the operation is terminated the selected address and the written data are shown one after the other.

Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.

For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

**x_LCD**

This program allows to manage an alphanumeric LCD featuring a number rows and columns definable by **User**.

The display must be connected to **CN5** following the connections shown in the diagram of **K51-AVR page 4 of 4**.

Before compiling select in menu **Option/Compiler/Misc/**: Byte End **5F**, Register File **REGS1.DAT**

In menu **Option/ LCD select:**

Db4 = P1.5 , Db5 = P1.6 , Db6 = P1.7 , Db7 = P1.2 , E = P1.4 , Rs = P1.3

**x_PPI**

This program shows, in hexadecimal format, the status of the eight lines connected to IC1 (PCF8547A9).

Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.

For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

**x_PPO**

This program activates sequentially one at a time all the 8 lines connected to IC1 (PCF8574A).

Before compiling set in menu Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.

For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.
**x_PPO2**

This program turns on in sequence the 16 TTL lines available on connector **CN3**.

Before compiling select in menu **Option/Compiler/Misc/ : Byte End 5F; Register File 8052.DAT**

**x_RTC**

This program allows you to show the RTC or Real Time Clock on IC7 (PCF8583) to the four 7 segments displays: To set the RTC values keys T2 and T3 are used, in detail key T2 increments the hours and T3 increments the minutes.

Whenever one of the two keys is pressed the seconds are reset.

Key T1 switches between visualization of seconds and hours.

Whenever a key is pressed, an acoustic signal is emitted.

Date and eventual alarm are not managed: Before compiling set in menu **Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.**

For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

**x_TER**

This program reads the temperature measured by IC3 (DS1621) and shows it in centigrade degrees with values ranging from -55 to +125.

Before compiling set in menu **Option/Compiler/Misc: Byte End 5F; Register File 89c1051.DAT or 8052.DAT.**

For use with 8xC51/52 modify the source where the pins used are described replacing pins for 89c1051 with pins for 8xc51.

**BASCOM Examples for boards**

**KND_08 - KND_44 - KAD_08**

**KND_08**

This program allows to manage the board resources of KND 08 card through a menu, using 2 TTL lines driven by a family 51 micro controller.

This program is managed through a RS 232 serial line, so it is essential to connect a free COM port of the PC to connector CN2 of K51-AVR.

To configure the BASCOM 8051 terminal in menu Options/Communication select the COM port and set Baud Rate to 19200, parity to none, data bits to 8, stop bits to 1.

The board used to drive KND 08 is K51-AVR, connections are:

K51-AVR ............... KND 08
L1 (pin4 CN6) ----> SC (pin2 CN1 KND08)
L2 (pin5 CN6) ----> SD (pin1 CN1 KND08)
Supply both the boards.
Before compiling in menu Option/Compiler/Misc set Byte End(Hex) = 60.

**KND_44**

This program allows to manage the board resources of KND 44 card through a menu, using 2 TTL lines driven by a family 51 micro controller.

This program is managed through a RS 232 serial line, so it is essential to connect a free COM port of the PC to connector CN2 of K51-AVR.

To configure the BASCOM 8051 terminal in menu Options/Communication select the COM port and set Baud Rate to 19200, parity to none, databits to 8, stopbits to 1.

The board used to drive KND 44 is K51-AVR, connections are:

- K51-AVR ............... KND 44
- L1 (pin4 CN6) ----> SC (pin2 CN1 KND44)
- L2 (pin5 CN6) ----> SD (pin1 CN1 KND44)

Supply both the boards.
Before compiling in menu Option/Compiler/Misc set Byte End(Hex) = 60.

**KAD_08**

This program manages a sliding alphanumeric message on eight 14-segments displays, installed on KAD 08 board, through 2 TTL signals driveb by a micro controller of family 51.

The master board is K51-AVR which must be connected to KAD 08 as follows:

- K51-AVR ........... KAD 08
- L1 (pin4 CN6) ----> SC (pin2 CN1 KAD08)
- L2 (pin5 CN6) ----> SD (pin1 CN1 KAD08)

Supply both the boards.
Before compiling in menu Option/Compiler/Misc set Byte End(Hex) = 70.

**KAD_08_2**

This program allows to manage the resources on the KAD 08 board through a menu and 2 TTL lines driven by a micro controller of the 51 family.

This program is controlled through the RS 232 serial line so it is essential to connect a free COM port on the PC to the connector CN2 of K51-AVR.

Configure the BASCOM 8051 terminal using menu Option/Communication, select the COM port and set baud rate to 19200, parity to none, data bits to 8 and stop bits to 1.

The master board is K51-AVR which must be connected to KAD 08 as follows:

- K51-AVR ........... KAD 08
- L1 (pin4 CN6) ----> SC (pin2 CN1 KAD08)
- L2 (pin5 CN6) ----> SD (pin1 CN1 KAD08)

Supply both the boards.
Before compiling in menu Option/Compiler/Misc set Byte End(Hex) = 50.
EXAMPLEs
BASCOM-LT
for K51-AVR.

K51-AVR
DEMO_AD11, DEMO_AD4, DEMO_DA, DEMO_REE, DEMO_WEE, DEMO_LCD,
DEMO_PPI, DEMO_PPI1, DEMO_PPO, DEMO_PPO1, DEMO_PPO2,
DEMO_RTC, DEMO_TER.

BASCOM-LT Examples for boards
KND_08 - KND_44 - KAD_08

KND_08, KND_44, KAD_08

EXAMPLEs
BASCOM-8051
for K51-AVR.

K51-AVR
51_AD11, 51_AD4, 51_DA, 51_REE, 51_WEE, 51_PPI, 51_PPO, 51_RTC,
51_TER

GPC® F2
F2_AD11, F2_AD4, F2_DA, F2_REE, F2_WEE, F2_PPI, F2_PPO, F2_RTC,
F2_TER

BASCOM-8051 Examples for boards
KND_08 - KND_44 - KAD_08
KND_08, KND_44, KAD_08, KAD_08_2

GPC® F2
F2_KND_08, F2_KND_44, F2_KAD_08, F2_KND_08_2

EXAMPLEs
BASCOM-AVR
for K51-AVR.

K51-AVR
DEMO_AD11, DEMO_AD4, DEMO_DA, DEMO_REE, DEMO_WEE, DEMO_PPI,
DEMO_PPO, DEMO_RTC, DEMO_TER

BASCOM-AVR Examples for boards
KND_08 - KND_44 - KAD_08

KND_08, KND_44, KAD_08

Page up-dated at June 7st, 2000

GRIFO®
Via dell'Artigiano, 8/6
40016 San Giorgio di Piano
Bologna ITALY

Tel: +39 051 892.052 (4 lines)
FAX: +39 051 893.661

E-mail:
for commercial communications sales@grifo.it
for technical communications tech@grifo.it
for general communications grifo@grifo.it
13.1.2 Rhombus

Rhombus developed the SCE-51. A powerful small 8051 micro processor board with on board RAM and FLASHROM and bootloader.
In addition the board serves as an in circuit emulator.

Transferring your program to RAM goes very fast. Faster than loading it into the traditional FLASHROM. So during debugging it is well suited for debugging large applications.

There are many possibilities with this board and you have to look at www.rhombusinc.com for all the details.

A picture of the board is included here:
Since the help file must be kept small, the quality of the picture is poor.
A bootloader is integrated into BASCOM. Select the Rhombus SCE-51 programmer to enable it.
Index

- # -

#ELSE 81
#ENDIF 82
#if 80

- $ -

$BAUD 89
$BGF 89
$CRYSTAL 91
$DEFAULT XRAM 92
$END ASM 88
$EXTERNAL 92
$INCLUDE 93
$IRAMSTART 94
$LARGE 94
$LCD 95, 97
$LCDRS 96
$LIB 95
$NOBREAK 97
$NOINIT 98
$NONAN 98
$NONULL 99, 136
$NORAMCLEAR 99
$NOSP 100
$OBJ 100
$RAMSIZE 101
$RAMSTART 103
$RAMTRON 101
$REGFILE 104
$ROMSTART 104
$SERIALINPUT 105
$SERIALINPUT2LCD 105
$SERIALOUTPUT 106
$SIM 107
$TIMEOUT 107
$WAIT 108

- 1 -

1WIRE 258
1WIRECOUNT 84
1WREAD 83
1WRESET 83
1WSEARCHFIRST 85
1WSEARCHNEXT 87
1WRITE 83

- 8 -

8032_Alternative port-pin functions 256
8032_TIMER2 289
80515_WATCHDOG 293
89C51 297

- A -

ABS 109
Additional Hardware 251
ALIAS 108
ASC 110
AT898252_WATCHDOG 292
AVG 110

- B -

BASCOM 38
BASCOM statements 77
BAUD 111
BCD 112
BITWAIT 112
Blow IT Flashprogrammer 267
BREAK 113
BROWSE001e 144

- C -

CALL 113
CASE 216
CHR 114
CLOSE 198
CLS 115
Compiler Limits 282
CONFIG 116
CONFIG 1WIRE 117
CONFIG ADUC812 117
CONFIG BAUD 119
CONFIG BAUD1 119
CONFIG DEBOUNCE 120
CONFIG GETRC 121
CONFIG GRAPHLCD 121
CONFIG I2CDELAY 120
CONFIG LCD  125
CONFIG LCDBUS  126
CONFIG LCDPIN  125
CONFIG MICROWIRE  127
CONFIG PRINT  127
CONFIG SCL  128
CONFIG SDA  128
CONFIG SERVOS  129
CONFIG SPI  130
CONFIG TIMER0
  TIMER1  131
CONFIG WATCHDOG  132
CONST  116
COUNTER  132
CPEEK  134
Credits  68
CURSOR  135
CYGNAL JTAG Programmer  272

- D -

DATA  136
DATA EEPROM  292
DEBOUNCE  137
DECLARE  139
DECLR  138
DEF  140
DECR  140
DEFBYTE  140
DEFFINT  140
DEFLCDCHAR  140
DELAY  141
DIM  141
DISABLE  143
DISPLAY  143

- E -

Edit Copy  42
Edit Cut  42
Edit Find  43
Edit Find Next  43
Edit Goto  44
Edit Indent Block  45
Edit Paste  43
Edit Redo  42
Edit Replace  44
Edit Undo  42
Edit Unindent Block  45

Editor Keys  46
ELSE  144
ENABLE  145
END  146
ENDIF  146
ERASE  147
Error messages  279
EXIT  148

- F -

File Close  40
File Compile  47
File End  41
File Exit  41
File New  39
File Open  39
File Print  41
File Print Preview  40
File Save  40
File Save As...  40
File Simulate  49
File Transmit  53
FOR  148
FOURTHLINE  149
FUSING  150
Futurelec  272

- G -

GET  151, 198
GETAD  152
GETAD2051  153
GETRC  158
GETRC5  160
GOSUB  162
GOTO  163
Grifo  302

- H -

Hardware - I2C  258
Hardware - LCD display  257
Help About  67
Help Credits  68
Help Forum  68
Help index  67
Help on help  67
Help Shop  68

© 2007 MCS Electronics
Options View Report 48
OUT 200

- P -

PEEK 202
Peter Averill's TAFE programmer 273
PG2051 flash programmer 267
PG302 programmer 268
POKE 202
PORT 201
POWERDOWN 203
PRINT 203
PRINTBIN 204
PRINTHEX 205
PRIORITY 206
PSET 207
PUT 198, 207

- R -

READ 208
READMAGCARD 209
REM 211
REPLACE 211
Reserved Words 283
RESET 212
RESTORE 212
RETURN 213
Rhombus 309
Rhombus SCE-51 275
RIGHT 214
RND 214
ROTATE 215
RUNNING BASCOM-8051 37

- S -

Sample Electronics ISP programmer 270
SE 512 or SE514 programmer 269
SE511-SE516 programmer 276
SE-812 270
SELECT 216
Send to chip 52
SET 216
SHIFT 217
SHIFTCURSOR 217
SHIIFTIN 218
SHIFTLCD 219
SHOWPIC 220
SOUND 220
SPACE 222
SPC 223
SPIIN 224
SPIINIT 224
SPIOUT 225
START 225
STK200/300 ISP Programmer 274
STOP 226
STOP TIMER 226
STR 228
STRING 228
SUB 229
SWAP 230
Syntax check 47

- T -

Table of contents 14
THIRDLINE 230
Tool LIB Manager 55
Tool Triscient Converter 56
Tools Export to RTF 57
Tools Graphic Converter 54

- U -

UCASE 231
UPPERLINE 232
Using assembly 239
Using the DAC 294

- V -

VAL 232
VARPTR 233

- W -

WAIT 233
WAITKEY 234
WAITMS 234
WATCHDOG 235
WEND 236
WHILE 236
WHILE.. WEND 236
Window arrange icons 67
Window cascade 66
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window tile</td>
<td>67</td>
</tr>
<tr>
<td>Windows minimize all</td>
<td>67</td>
</tr>
</tbody>
</table>