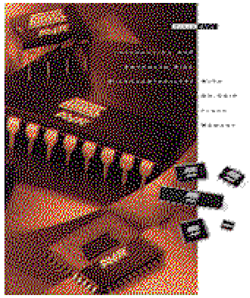
Index

BASCOM-AVR



Version 1.11.7.4

BASCOM and Hardware (not a jump link)

BASCOM Language Reference (not a jump link)

$ASM	$BAUD	$BAUD1	$BGE
$CRYSTAL	$DATA	$DBG	$DEFAULT
$EEPROMHEX	$EXTERNAL	$INCLUDE	$LCD
$INC			
$LCDPUTCTRL	$LCDPUTDATA	$LCDVFO	$LIB
$NOINIT	$NORAMCLEAR	$REGFILE	$ROMSTART
$SERIALINPUT1	$SERIALINPUT2LCD	$SERIALOUTPUT	$SERIALOUTPUT1
$TINY	$WAITSTATE	$XRAMSIZE	$XRAMSTART
1WRESET	1WREAD	1WWRITE	1WSEARCHFIRST
1WVERIFY	1WIRECOUNT		
ABS	ACOS	ALIAS	ASC
ATN	ATN2	BASE64DEC	
BAUD	BCD	BIN	BINVAL
BITWAIT	BYVAL	BSAVE	BLOAD
CALL	CIRCLE	CHECKSUM	CHR
CLOCKDIVISION	CLOSE	CPEEKH	CONFIG
CLOSESOCKET			
CONFIG ADC	CONFIG BCCARD	CONFIG CLOCK	CONFIG COM1
CONFIG DATE	CONFIG PS2EMU	CONFIG ATEMU	CONFIG I2CSLAVE
CONFIG GRAPHLCD	CONFIG KEYBOARD	CONFIG TIMER0	CONFIG TIMER1
CONFIG LCDBUS	CONFIG LCDMODE	CONFIG 1WIRE	CONFIG SERIALIN
CONFIG SERIALOUT	CONFIG SERIALOUT1	CONFIG SDA	CONFIG SCL
CONFIG SPI	CONFIG LCDPIN	CONFIG WATCHDOG	CONFIG PORT
CONFIG TCPIP			
CONST	COS	COSH	CRC8
CRYSTAL	CPEEK	CURSOR	DATE
DATA	DATE$	DBG	DEBOUNCE
DAYOFWEEK	DAYOFYEAR	DriveGetIdentity	DriveWriteSector
DECLARE FUNCTION	DECLARE SUB	DEFXXX	DEFLCDCHAR
DELAY	DIM	DISABLE	DISPLAY
DTMFOUT	DISKFREE	DIR	DriveReset
ECHO	ELSE	ENABLE	END
EXP	EOF		
FIX	FORMAT	FOR-NEXT	FOURTHLINE
FUSING	FLUSH	FREEFILE	FILEATTR

FILETIME	FILEDATETIME		
GETADC	GETKBD	GETATKBD	GETRC
GLCDCMD only for SED	GLCDDATA only for SED	GOSUB	GOTO
GET			
HEX	HEXVAL	HIGH	HIGHW
I2CINIT	I2CRECEIVE	I2CSEND	I2CSTART,I2CSTOP,I2CRBYTE,I2CWBYTE
IF-THEN-ELSE-END IF	INCR	INITLCD	INKEY
INPUTBIN	INPUTHEX	INPUT	INSTR
ISCHARWAITING	INITFILESYSTEM	IP2STR	
KILL			
LCASE	LCD	LCDAT only for SED series	LEFT
LINE	LOAD	LOADADR	LOADLABEL
LCDCONTRAST			
LOCATE	LOG	LOG10	LOOKDOWN
LOC	LOF		
LOOKUPSTR	LOW	LOWERLINE	LTRIM
MAKEBCD	MAKEDEC	MAKEINT	MID
MIN			
ON INTERRUPT	ON VALUE	OPEN	OUT
PEEK	POKE	POPALL	POWER
POWERSAVE	PRINT	PRINTBIN	PSET
PS2MOUSEXY	PUT		
PULSEOUT	PUSHALL	RAD2DEG	RC5SEND
READ	READEEPROM	READMAGCARD	REM
RESTORE	RETURN	RIGHT	RND
ROUND	RTRIM		
SENDSCAN	SENDSCANKBD		
SELECT CASE - END SELECT	SET	SETFONT onlyfor SED series	SERIN
SECOFDAY	SECELAPSED	SYSDAY	SYSSEC
SETTCP			
SGN	SHIFT	SHIFTCURSOR	SHIFTIN
SHIFTLCD	SHOWPIC	SHOWPICE	SIN
SONYSEND	SOUND	SPACE	SPC
SPIINIT	SPIMOVE	SPIOUT	SQR
STCHECK	STOP	STR	STRING
SOCKETSTAT	SOCKETCONNECT	SOCKETLISTEN	SEEK

TIME	TCPWRITE	TCPWRITESTR	TCPREAD
SWAP	TAN	TANH	THIRDLINE
TOGGLE	TRIM	UCASE	UPPERLINE
UDPREAD	UDPWRITE	UDPWRITESTR	
VARPTR	WAIT	WAITKEY	WAITMS
WRITE			
WHILE-WEND	WRITEEEPROM	X10SEND	X10DETECT

International Resellers

Supported Programmers

Assembly Mnemonics

Mixing BASIC with assembly

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

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

Our website is located at **http://www.mcselec.com**

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

Installation of BASCOM -AVR

When you downloaded the ZIP files from our website you need to UNZIP them all.

The first file will unzip the file named SETUP.EXE

The second will unzip the file named SETUP.W02

The third will unzip the file named SETUP.W03

The fourth will unzip into SETUP.W04.

The files can also come on diskettes. In that case there are no zip files and you can continue without unzipping.

And finally the files can be on a CD-ROM. In that case the files are unzipped already too.

The commercial edition comes with a license file in the form of a dll. This file is always on the disk where the file SETUP.EXE is located. When explorer does not show this file, you must set the option in explorer to view system files(because a DLL is a system file).

Some resellers might distribute the DLL file in a zipped file. Or the file might have the extension of a number like 123. In this case you must rename the number into DLL.

Make sure the DLL is in the same directory as the SETUP.EXE file.

When you are using the DEMO you don't need to worry about the license file.

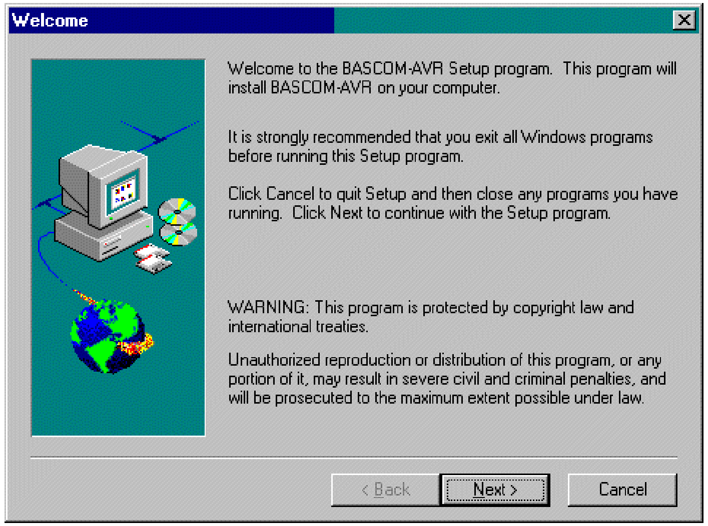
When you are installing on a NT machine like NT4 or W2000, you need to have Administrator rights.

After installing BASCOM you need to run BASCOM once as an administrator too. After that you may run BASCOM as any other user.

Now run the SETUP.EXE by double clicking on it in explorer. Or from the DOS command prompt.

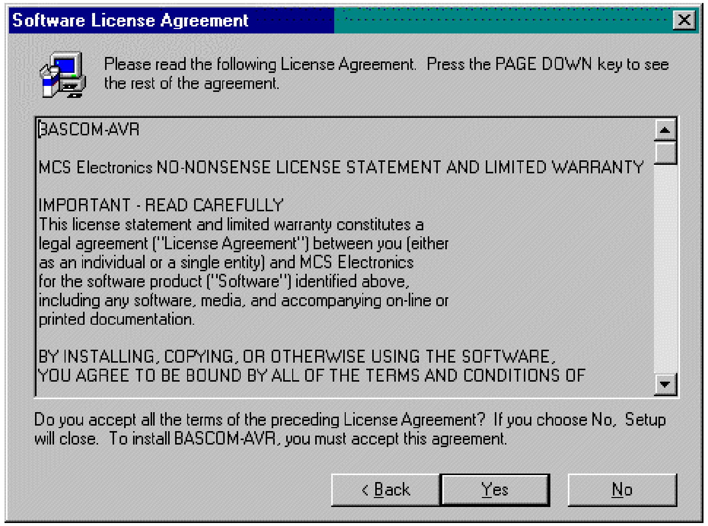
The following window will appear:

(screen shots may differ a bit)



Click on the Next button to continue installation.

The following license info window will appear:

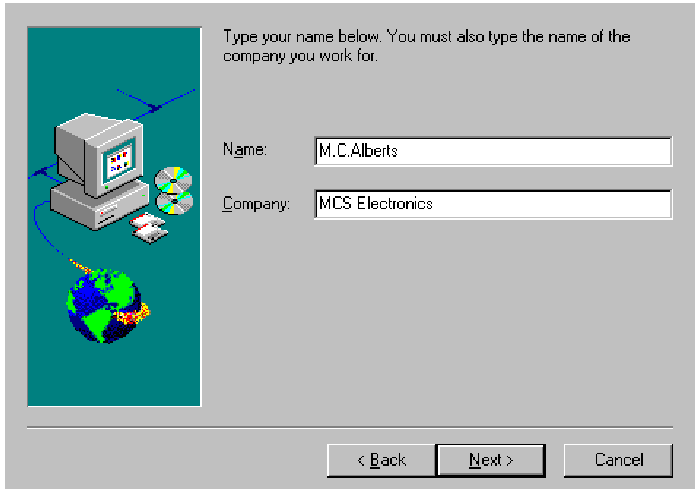


Read the license agreement and click the Yes button when you agree.

A window with additional information is then displayed. This information will be installed as a readme.txt file and contains information on how to get free updates. It also contains the password needed to unzip updates.

After reading the information, click the Next button.

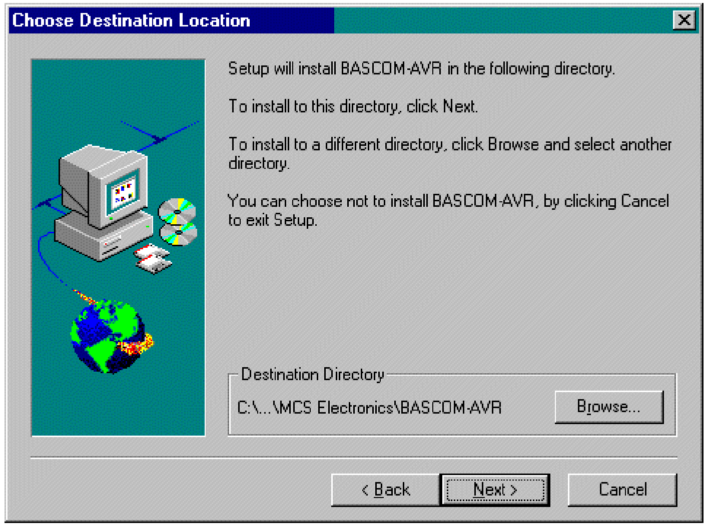
Now the following window appears:



Fill in your name and company name.

Click the Next button to continue.

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



Select the Browse button to change the directory path if required.

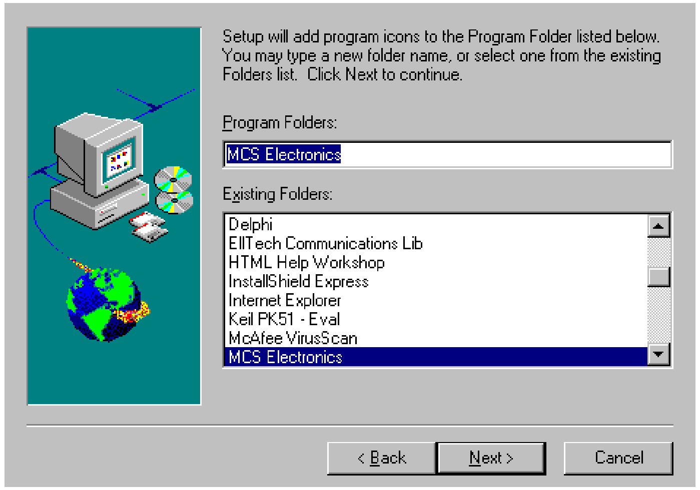
By default BASCOM-AVR will be installed into:

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

After selecting the installation directory, click the Next button.

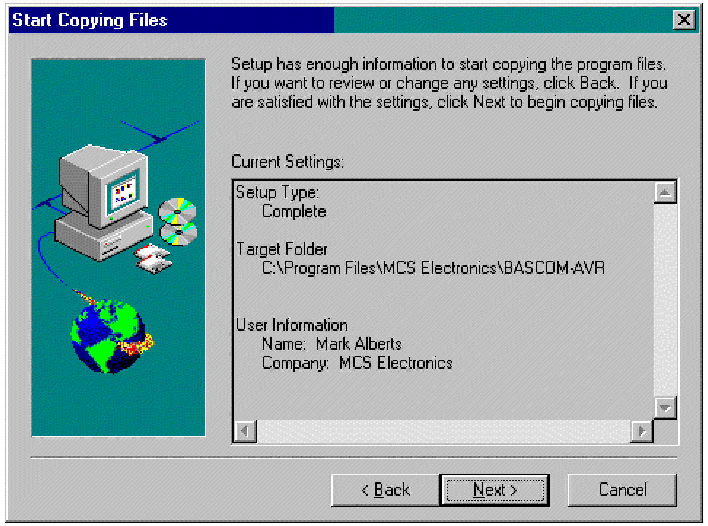
This time you will be asked in which program group the BASCOM-AVR icon must be placed.

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



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

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



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

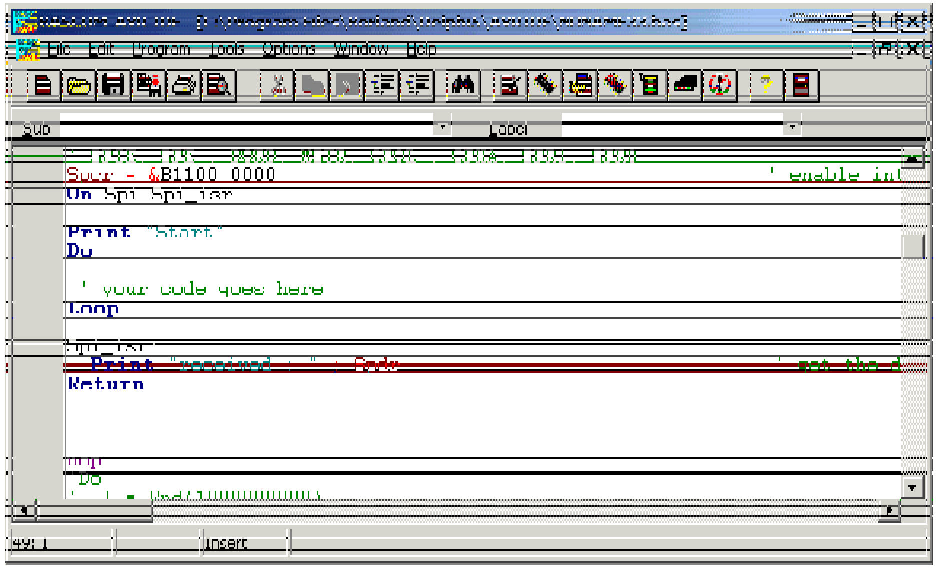
A sub directory named SAMPLES contains all the BASCOM-AVR sample files.

A sub directory named LIB contains the Library files.

Running BASCOM-AVR

Double-click the BASCOM-AVR icon to run BASCOM.

The following window will appear. (If this is your first run, the edit window will be empty.)



The most-recently opened file will be loaded.

FileNew

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

The focus is set to the new window.

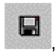
File new shortcut: , CTRL + N

File Open

With this option you can load an existing program from disk.

BASCOM saves files in standard ASCII format. Therefore, if you want to load a file that was made with another editor be sure that it is saved as an ASCII file.

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

File open shortcut : , CTRL+O

File Close

Close the current program.

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

File close shortcut : 

File Save

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

If the program was created with the **F**ile **N**ew option, you will be asked to name the file first. Use the **F**ile **S**ave **A**s option to give the file another name.

Note that the file is saved as an ASCII file.

File save shortcut : , CTRL+S

File Save As

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

Note that the file is saved as an ASCII file.

File save as shortcut :

FilePrintPreview

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

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

File print preview shortcut :

FilePrint

With this option, you can print the current program.

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

File print shortcut : , CTRL+P

File Exit

With this option, you can leave BASCOM.

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

File exit shortcut :

Edit Undo

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

Edit Undo shortcut : , CTRL+Z

Edit Redo

With this option, you can redo the last undo.

Edit Redo shortcut : , CTRL+SHIFT+Z

Edit Cut

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

Edit cut shortcut : , CTRL+X

Edit Copy

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

Edit copy shortcut : , CTRL+C

Edit Paste

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

Edit paste shortcut : , CTRL+V

Edit Find

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

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

Edit Find shortcut : , CTRL+F

Edit Find Next

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

Edit Find Next shortcut : , F3

EditReplace

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

Edit Replace shortcut : , CTRL+R

Edit Goto

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

Edit go to line shortcut : ,CTRL+G

Edit Toggle Bookmark

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

Edit Goto Bookmark

With this option, you can jump to a bookmark.

There can be up to 8 bookmarks. Shortcut : CTRL+Q+ x where x can be 1-8

Edit Indent Block

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

Edit Indent Block shortcut : , CTRL+SHIFT+I

Edit Unindent Block

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

Edit Unindent Block shortcut : , CTRL+SHIFT+U

Program Compile

With this option, you can compile your current program.

Your program will be saved automatically before being compiled.

The following files will be created depending on the Option Compiler Settings.

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

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

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

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

At the next compilation, the error window will disappear.

Program compile shortcut: , F7

Program Syntax Check

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

Program syntax check shortcut , CTRL + F7

Program Show Result

Use this option to view the result of the compilation.

See the [Options Compiler Output](#) for specifying which files must be created.

The files that can be viewed are report and error.

File show result shortcut : ,CTRL+W

Informationprovided in the report:

Info	Description
Report	Name of the program
Date and time	The compilation date and time.
Compiler	The version of the compiler.
Processor	The selected target processor.
SRAM	Size of microprocessor SRAM (internal RAM).
EEPROM	Size of microprocessor EEPROM (internal EEPROM).
ROMSIZE	Size of the microprocessor FLASH ROM.
ROMIMAGE	Size of the compiled program.
BAUD	Selected baud rate.
XTAL	Selected XTAL or frequency.
BAUD error	The error percentage of the baud rate.
XRAM	Size of external RAM if available.
Stack start	The location in memory, where the hardware stack points to. The HW-stack pointer grows down.
S-Stacksize	The size of the software stack.
S-Stackstart	The location in memory where the software stack pointer points to. The software stack pointer grows down.
Framesize	The size of the frame. The frame is used for storing local variables.
Framestart	The location in memory where the frame starts.
LCD address	The address that must be placed on the bus to enable the LCD display E-line.
LCD RS	The address that must be placed on the bus to enable the LCD RS-line
LCD mode	The mode the LCD display is used with. 4 bit mode or 8 bit mode.
LCD DB7-DB4	The port pins used for controlling the LCD in pin mode.
LCD E	The port pin used to control the LCD enable line.
LCD RS	The port pin used to control the LCD RS line.
Variable	The variable name and address in memory
Constant	Constants name and value Some internal constants are : _CHIP : number that identifies the selected chip _RAMSIZE : size of SRAM _ERAMSIZE : size of EEPROM _XTAL : value of crystal _BUILD : number that identifies the version of the compiler

Program Simulate

With this option, you can simulate your program.

You can simulate your programs with AVR Studio or any other Simulator available or you can use the build in Simulator.

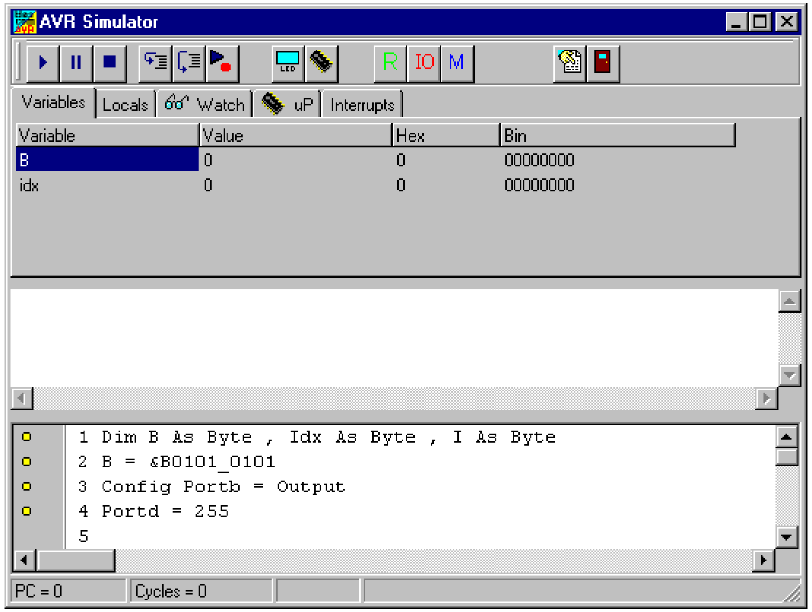
Which one will be used when you press F2 depends on the selection you made in the Options Simulator TAB.

Program Simulate shortcut : , F2

To use the build in Simulator the files DBG and OBJ must be selected from the Options Compiler Output TAB.

The OBJ file is the same file that is used with the AVR Studio simulator.

The DBG file contains info about variables and many other info needed to simulate a program.



The Sim window is divided into a few sections:

The Toolbar

The toolbar contains the buttons you can press to start an action.

This starts a simulation. It is the RUN button. The simulation will pause when you press the pause button. You can also press F5.

This is the pause button. Pressing this button will pause simulation.

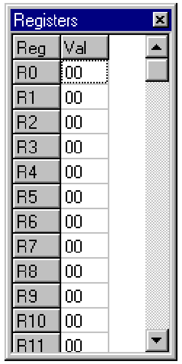
This is the STOP button. Pressing this button will stop simulation and you can't continue. This because all variables are reset. You need to press this button when you want to simulate your program again.

This is sthe STEP button. Pressing this button(or F8) will execute one code line of your BASIC program. After the line is executed the simulator will be in the pause state.

This is the STEP OVER button. It has the same effect as the STEP button but sub programs are executed and there is no step into the SUB program. You can also press SHIFT+F8

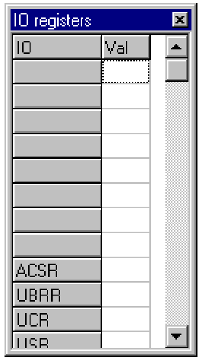
This is the RUN TO button. The simulator will RUN to the current line. The line must contain executable code.

This button will show the register window.



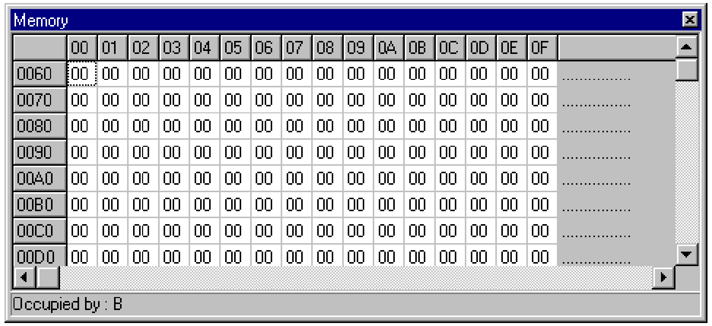
The values are show in hexadecimal format. To change a value click the cell of the Val column and type the new value.

This is the IO button and will show the IO registers.



The IO window works the same like the Register window. Blank rows indicate that there is no IO-register assigned to that address.(The blank rows might be deleted later.)

Pressing this button shows the Memory window.



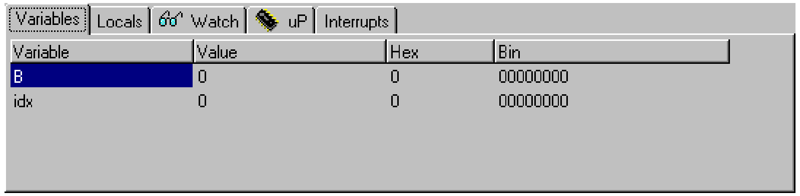
The values can be changed the same way like in the Register window.

When you move from cell to cell you can view in the status bar which variable is stored in the address.

The refresh variables button will refresh all variables during a run(F5). When you use the hardware simulator, the LEDS will only update their state when you have enabled this option. Note that using this option will slow down simulation.

Under the toolbar section there is a TAB with the pages:

VARIABLES



You can add variables by double clicking in the Variable-column. A list will pop up from which you can select the variable.

To watch an array variable you can type the name of the variable with the index.

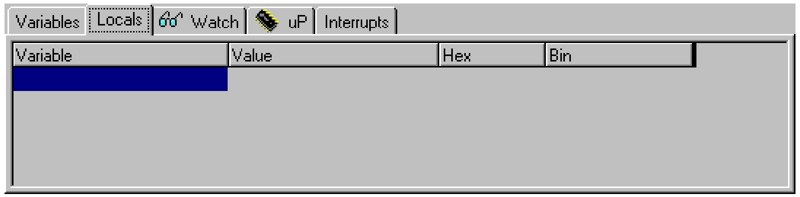
During simulation you can change the values of the variables in the Value-column, Hex-column or Bin-column. You must press ENTER to store the change.

To delete a row you can press CTRL+DEL.

To enter more variables, press the DOWN-key so a new row will become visible.

It is also possible to select a variable by selecting it from the code window, and then pressing enter.

LOCALS



The LOCAL window show the variables in a SUB or FUNCTION. LOCAL variables are also shown. You can not add variables.

Changing the value of the variables works the same as for the Variable TAB.

WATCH

The Watch-TAB can be used to enter an expression that will be evaluated during simulation. When the expression is true the simulation is paused.

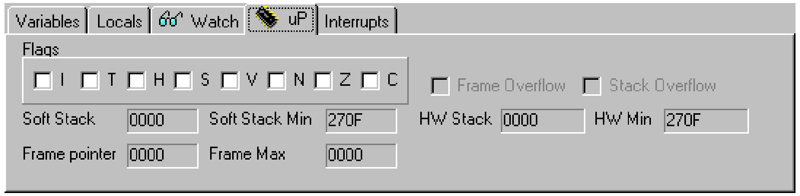
Type the expression in the text-field and press the Add-button.

When you press the Modify-button the current selected expression from the list is modified with the typed value.

To delete an expression you must select the expression from the list and press the Remove-button.

When the expression becomes true the expression that matches will be selected and the Watch-TAB will be shown.

UP



This TAB shows the status of the microprocessor SREG register.

The flags can be changed by clicking their checkboxes.

The software stack , hardware stack and frame pointer values are also shown. The minimum or maximum value during simulation is shown. When one of the data is entering another one there is a case of stack/frame overflow.

This will be signaled with a pause and a checkbox.

The snapshot button will create a snapshot of the registers and HW registers.

So it will create a copy of the memory once you press the snapshot button.

You will notice that the snapshot button will change into 'Stop'

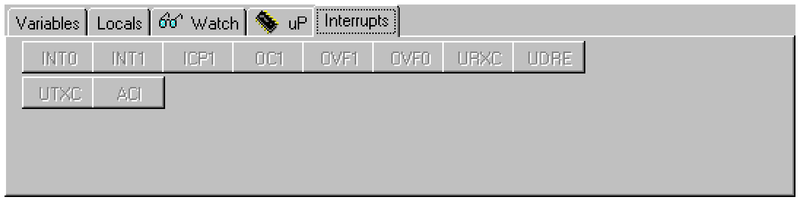
Now execute some code by pressing F8 and press the Snapshot button again.

A window will pop up that will show all modified address locations.

This can help at determining which registers a statement uses.

When you write an ISR with the NOSAVE option you can determine the used registers and save only the modified registers.

INTERRUPTS



This TAB shows the interrupt sources. When no ISR's are programmed all buttons will be disabled.

By clicking a button the corresponding ISR is executed.

The pulse generator can be used to supply pulses to the timer when used in counter mode.

Select the pin from the pull down box. Depending on the chip one or more pins are available. Most chips have 2counter so 2 input pins.

Select the number of pulse and the delay time and press the Pulse-button to generate the pulses.

The delay time is needed since other tasks must be processed as well.

The option ‘Enable timers’ must be selected when you want to simulate timers/counters.

TERMINAL Section

Under the TAB window you will find the terminal emulator window.

When you use PRINT, the output will be shown in this window.

When you use INPUT in your program, you must set the focus to the terminal window and type the needed value.

You can also print the values directly to the COM port.

Check the Terminal option to enable this feature.

The terminal emulator settings will be used for the baud rate and COM port.

SOURCE Section

Under the Terminal section you find the Source Window.

It contains the program you simulate. All lines that contain executable code have a yellow point in the left margin.

You can set a breakpoint on these lines by pressing F9.

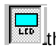
By moving the mouse cursor over a variable name the value is shown in the status bar.

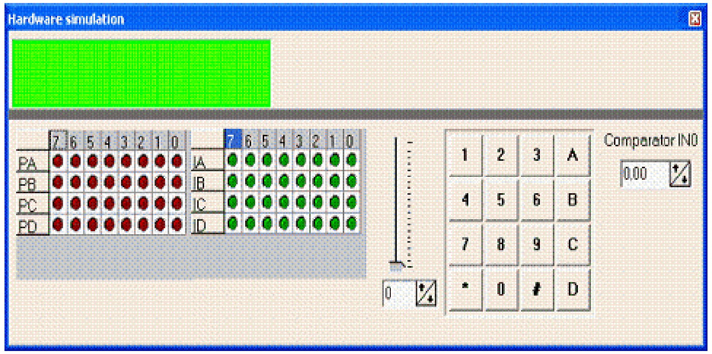
When you select a variable and press ENTER it will be added to the Variable window.

When you want to use the keys (F8 for stepping for example) the focus must be set to the Source Window.

A blue arrow will show the line that will be executed next.

The hardware simulator.

By pressing the hardware simulation button  the windows shown below will be displayed.



The top section is a virtual LCD display. It works for display code in PIN mode and bus mode. For bus mode only 8-bit bus mode is supported by the simulator.

The LED bars below are a visual indication of the ports.

By clicking a LED it will toggle.

PA means PORTA, PB means PORTB etc.

IA means PINA, IB means PINB etc.

It depends on the kind of microprocessor you have selected, which ports will be shown.

Right beside the PIN leds, there is a track bar. This bar can be used to simulate the input voltage of the ADC converter. Note that not all chips have an ADC converter. You can set a value for each channel.

Beside the trackbar is a numeric keypad. This keypad can be used to simulate the GETKBD() function.

When you simulate the getkbd() it is important that you press/click the keyboard button before running the getkbd() line !!!

With the Comparator simulator, you can specify the logic level on IN0.

Enable Real Hardware Simulation

By clicking the  button you can simulate the ports in circuit!

In order to get it work you must compile the basmon.bas file.

When compiled program a chip.

Lets say you have the DT006 simmstick. And you are using a 2313 AVR chip.

Open the basmon.bas file and change the line with $REGFILE = "xxx" into $REGFILE = "2313def.dat"

Now compile the program. Program the chip. It is best to set the lock bits so the monitor does not get overwritten when you accidentally press F4.

The real hardware simulation only works when the target micro system has a serial port. Most have and so does the DT006.

Connect a cable between the COM port of your PC and the DT006. You probably already have one connected. Normally it is used to send data to the terminal emulator with PRINT.

The monitor program is compile with 19200 baud. The Options Communication settings must be set to the same baud rate!

The same settings for the monitor program are used as for the Terminal emulator. So select the COM port and the baud rate of 19200.

Power up the DT006. It probably was since you created the basmon program and stored it in the 2313.

When you press the real hardware simulation button now the simulator will send and receive data when a port, pin or ddr register is changed.

This allows you to simulate an attached LCD display for example. Or something simpler, the LED. In the SAMPLE dir you will find a program DT006. You can compile thie program and press F2.

When you step through the program the LED's will change!

All statements can be simulated this way but the have to be static. Which means that 1wire will not work because it depends on timing. I2C has a static bus and that will work.

It is important that when you finish your simulation sessions that you click the button again to disable the Real hardware simulation.

When the program hangs it probably means that something wend wrong in the communication. The only way to escape is to press the Real hardware simulation again.

I think the simulation is a cost effective way to test attached hardware.

The refresh variables button will refresh all variables during a run(F5). When you use the hardware simulator, the LEDS will only update their state when you have enabled this option. Note that using this option will slow down simulation.

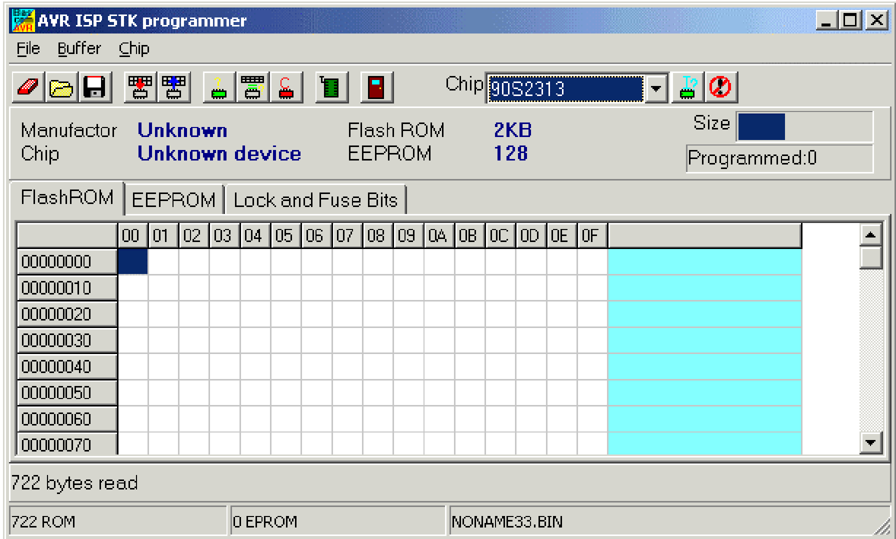
Program Send to Chip

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

Program send to chip shortcut , F4

Menu item	Description
File Exit	Return to editor
File, Test	With this option you can set the logic level to the LPT pins. This is only intended for the Sample Electronics programmer.
Buffer Clear	Clears buffer
Buffer Load from file	Loads a file into the buffer
Buffer Save to file	Saves the buffer content to a file
ChipIdentify	Identifies the chip
Write buffer into chip	Programs the buffer into the chip ROM or EEPROM
Read chipcode into buffer	Reads the code or data from the chips code memory or data memory
Chip blank check	Checks if the chip is blank
Chip erase	Erase the content of both the program memory and the data memory
Chip verify	Verifies if the buffer is the s ame as the chip program or data memory
Chip Set lockbits	Writes the selected lock bits LB1 and/or LB2. Only an erase will reset the lock bits
Chip autoprogram	Erases the chip and programs the chip. After the programming is completed, verification is performed.
RCEN	Writes a bit to enable the internal oscillator. This RCEN bit is only available on some AVR chips.

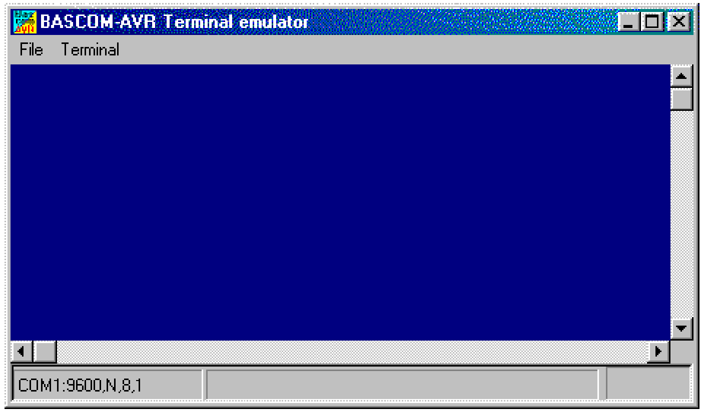
The following window will be shown:



Note that a chip must be ERASED before it can be programmed.

Tools Terminal Emulator

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



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

Note that you must use the same baud rate on both sides of the transmission. If you compiled your program with the Compiler Settings at 4800 baud, you must also set the Communication Settings to 4800 baud.

The setting for the baud rate is also reported in the report file.

File Upload

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

File Escape

Aborts the upload to the monitor program.

File Exit

Closes terminal emulator.

Terminal Clear

Clears the terminal window.

Terminal Open Log

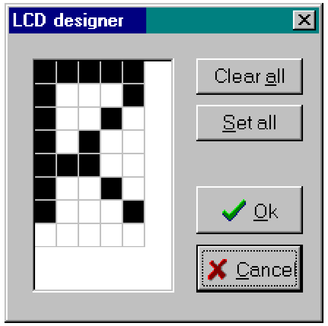
Open or closes a LOG file. When there is no LOG file selected you will be asked to enter or select a filename. All info that is printed to the terminal window is captured into the log file. The menu caption will change into 'Close Log' and when you choose this option the file will be closed.

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

Tools LCD Designer

With this option you can design special characters for LCD-displays.

The following window will appear:



The LCD-matrix has 7x5 points. The bottom row is reserved for the cursor but can be used.
You can select a point by clicking the left mouse button. If a cell was selected it will be deselected.

Clicking the Set All button will set all points.

Clicking the Clear All button will clear all points.

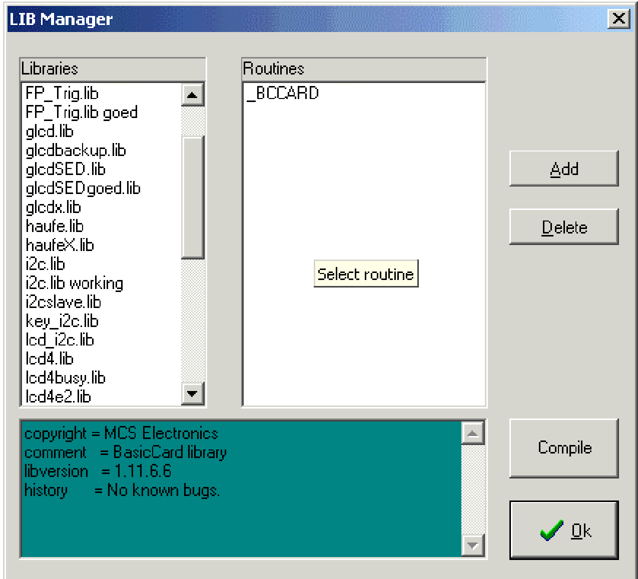
When you are finished you can press the Ok button : a statement will be inserted in your active program-editor window at the current cursor position. The statement looks like this :

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

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

Tools LIB Manager

With this option the following window will appear:



The Libraries are shown in the left pane. When you select one the routines that are in the library will be shown in the right pane.

By selecting a routine you can DELETE it.

By clicking the ADD button you can add an ASM routine to the library.

The COMPILE button will compile the lib into a LBX file. When an error occurs you will get an error. By watching the content of the generated lbx file you can determine the error.

A compiled LBX file does not contain comment and a huge amount of mnemonics is compiled into object code. This object code is inserted at compile time of the main BASIC program. And this results in faster compilation.

The DEMO version comes with the compiled MCS.LIB file and is named MCS.LBX. The ASM source is included with the commercial edition.

With the ability to create LBX files you can create add on packages for BASCOM and sell them. The LBX files could be distributed for free and the ASM source could be sold.

Some examples :

- ?? - MODBUS crc routine for the modbus slave program.
- ?? - Glcd.lib contains the graphical LCD asm code

Commercial packages available from MCS:

- ?? - I2CSLAVE library

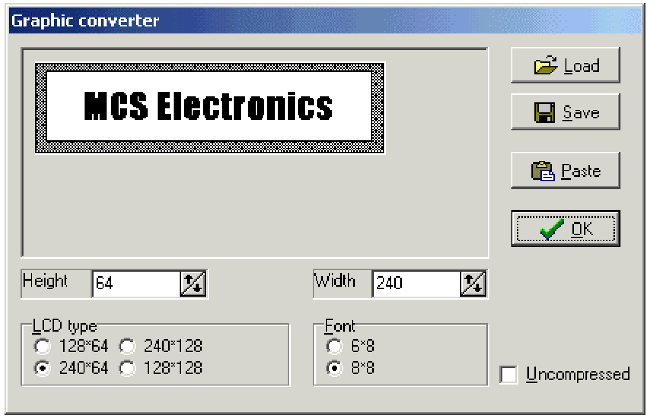
?? - BCCARD for communication with www.basiccard.com chipcards

See [$LIB](#) for writing your own libraries

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 can be maximum128 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.

You can also paste a picture from the clipboard by clicking the Paste button.

Press the Ok-button to return to the editor.

The picture can be shown with the [ShowPic](#) statement or the ShowpicE statement.

The BGF files are RLE encoded to save space.

When you use your own drawing routine you can also save the pictures uncompressed by setting the Uncompressed checkbox. The resulting BGF files can not be shown with the showpic or showpicE statements anymore in that case!

The BGF format is made up as following:

?? - first byte is the height of the picture

?? - second byte is the width of the picture

?? - for each row, all pixels are scanned from left to right in steps of 6 or 8 depending on the font size. The resulting byte in stored with RLE compression

RLE used is : byte value, AA(hex), repeats.
So a sequence of 5, AA, 10 means that a byte with the value of 5 must be repeated 16 times (hex notation used)

Tools Stack Analyzer

The Stack analyzer helps to determine the proper stack size.

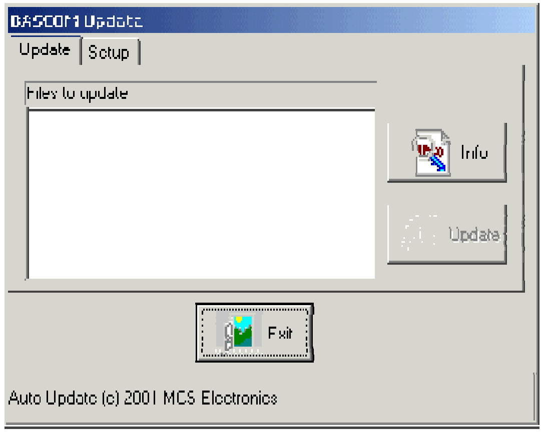
See $DBG for the proper usage of this option.

Tools Auto Update

The auto update feature allows you to update BASCOM automatic.

Before choosing this option you must close all editor windows.

The auto update window looks like :



You must first fill in the setting by clicking the Setup TAB.

The setup options are described below.

To check if there is a newer version of BASCOM available press the Info button.

You must have a connection to the internet in order to use this option.

The program will check the file update.ver on the internet and will compare the files on your system.

The compare is based on the file size. The files are compared with the zipped files in your BASCOM directory. There fore it is important that once you have updated the system, you do not move or delete the downloaded zip files.

When the zip files are not in the BASCOM\ZIPS directory or the file size is different, the file will be added to the white window. A checkbox is set to indicate that you need to download it.

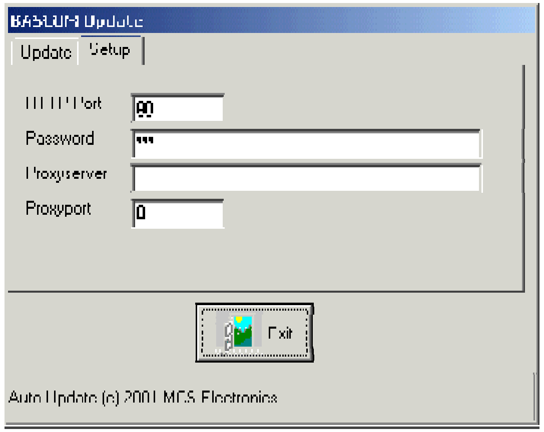
Once Auto update knows which files are needed you may click the checkbox to skip the file for downloading.

When ready press the Update button to start the update process.

All selected files will be downloaded and after the last file is downloaded the application will end. The application mcsunzip.exe is started automatic and this application unzips the downloaded files.

After the files are unzipped the mcsunzip.exe will start BASCOM again.

The Setup tab looks like:



Item	Description
HTTP Port	This should be set to 80
Password	The password is the password that you can find in the BASCOM application dir in the readme.txt file. Copy it from there and paste it with CTRL+V. The –s shown as part of the password should not be copied.
Proxyserver	This is the name or IP address of the proxy server. Home users typically don't have a proxy server. Ask your administrator for the TCP address.
Proxyport	This is the proxy port number. Set it to 0 when you don't have a proxy server.

Privacy notice

The auto update process only checks the MCS electronics website for the update.ver file. And will download the needed zipped files. The application will **not** send any information from your PC to either the MCS website or any other location.

It is good practice to use a firewall so you can check all in and out going data from your applications when you are connected to the Internet.

When you do not have an internet connection on your BASCOM PC.

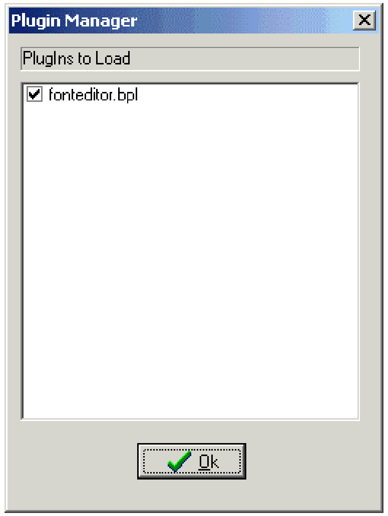
Follow this procedure to get the update:

- ?? Download the zip files manual from the website www.mcselec.com/download/avr or www.mcselec.com/download/avr/beta
- ?? Copy the zip files to the BASCOM PC in a subdirectory named ZIPS
- ?? ZIPS must be a subdir in the BASCOM application directory
- ?? Create a file with notepad in the BASCOM application folder named uzp.lst
- ?? Add all the ZIP file names to this files and save and close the uzp.lst file
- ?? Run BASCOM and select Tools, Auto Update
- ?? Select the Password-TAB
- ?? Enter the password. The password can be found in the readme.txt file that is located in the BASCOM application folder
- ?? As an alternative for typing in the password, use copy(CTRL+C) & paste(CTRL+V) so you do not make type errors
- ?? Exit BASCOM and run the program named mcsunzip.exe

?? This program will unzip the zip files and will start BASCOM when ready

Tools Plugin Manager

The Plugin Manager allows you to specify which Plug-ins needs to be loaded the next time you start BASCOM.



Just select the plug ins you want to load/use by setting the checkbox.

The plug ins will be loaded under the Tools Menu.

You need to add a button to the toolbar by pressing the right mouse button while the mouse cursor is pointed to the toolbar.

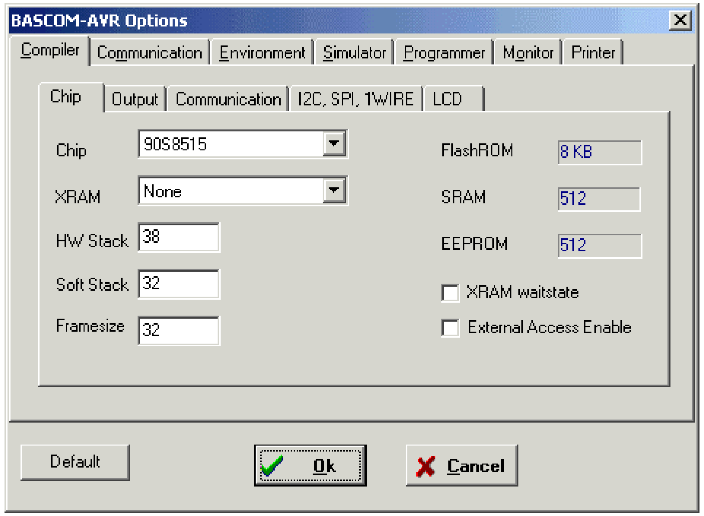
When you want to write your own plug ins, contact plugin@mcselec.com

OptionsCompiler

With this option, you can modify the compiler options.

The following TAB pages are available:

Options Compiler Chip



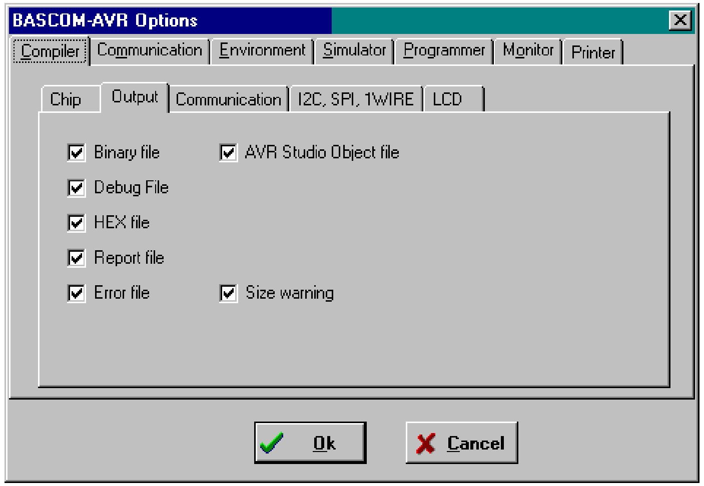
The following options are available:

Options Compiler Chip

Item	Description
Chip	Selects the target chip. Each chip has a corresponding x.DAT file with specifications of the chip. Note that some DAT files are not available yet.
XRAM	Selects the size of the external RAM. KB means Kilo Bytes. For 32 KB you need a 62256 STATIC RAM chip.
HW Stack	The amount of bytes available for the hard ware stack. When you use GOSUB or CALL, you are using 2 bytes of HW stack space. When you nest 2 GOSUB's you are using 4 bytes (2*2). Most statements need HW stack too. An interrupt needs 32 by tes.
Soft Stack	Specifies the size of the software stack. Each local variable uses 2 bytes. Each variable that is passed to a sub program uses 2 bytes too. So when you have used 10 locals in a SUB and the SUB passes 3 parameters, you need 13 * 2 = 26 bytes.
Frame size	Specifies the size of the frame. Each local is stored in a space that is named the frame space. When you have 2 local integers and a string with a length of 10, you need a frame size of (2*2) + 11 = 15 bytes. The internal conversion routines used when you use INPUT num,STR(),VAL() etc, also use the frame. They need a maximum of 16 bytes. So for this example 15+16 = 31 would be a good value.
XRAM waitstate	Select to insert a wait state for the external RAM.
External Access enable	Select this option to allow external access of the micro. The 8515 for example can use port A and C to control a RAM chip.

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

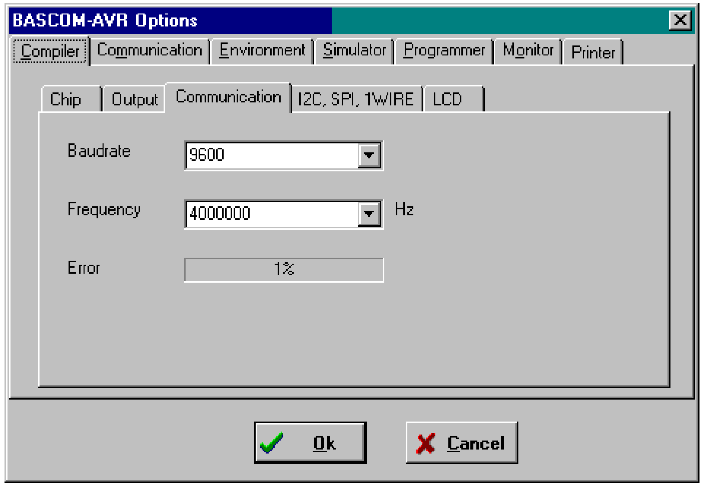
OptionsCompilerOutput



Options Compiler Output

Item	Description
Binary file	Select to generate a binary file. (xxx.bin)
Debug file	Select to generate a debug file (xxx.dbg)
Hex file	Select to generate an Intel HEX file (xxx.hex)
Reportfile	Select to generate a report file (xxx.rpt)
Error file	Select to generate an error file (xxx.err)
AVR Studio object file	Select to generate an AVR Studio object file (xxx.obj)
Size warning	Select to generate a warning when the code size exceeds the Flash ROM size.
Swap words	This option will swap the bytes of the object code words. Useful for some programmers. Should be disabled for most programmers. Don't use it with the internal supported programmers.
Optimize code	This options does additional optimization of the generated code. Since it takes more time it is an option.
Show internal variables	Internal variables are used. Most of them refer to a register. Like _TEMP1 = R24. This option shows these variables in the report.

Options Compiler Communication



Options Compiler Communication

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

The settings for the internal hardware UART are:

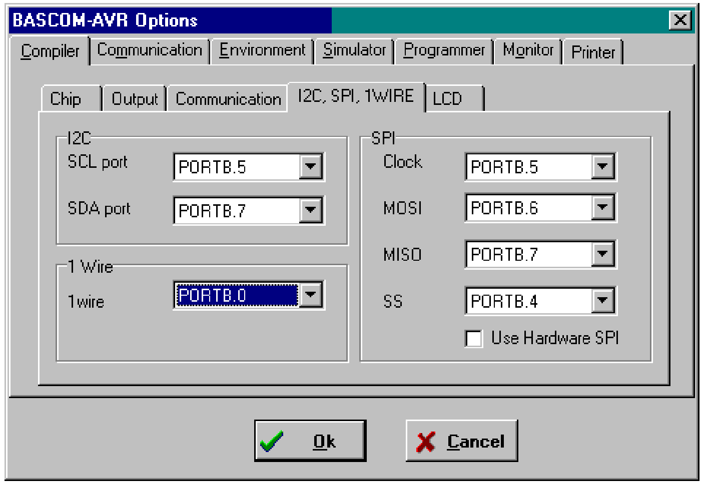
No parity

8 data bits

1 stop bit

Note that these settings must match the settings of the terminal emulator. In the simulator the output is always shown correct since the baud rate is not taken in consideration during simulation. With real hardware when you print data at 9600 baud, the terminal emulator will show weird characters when not set to the same baud rate, in this example, to 9600 baud.

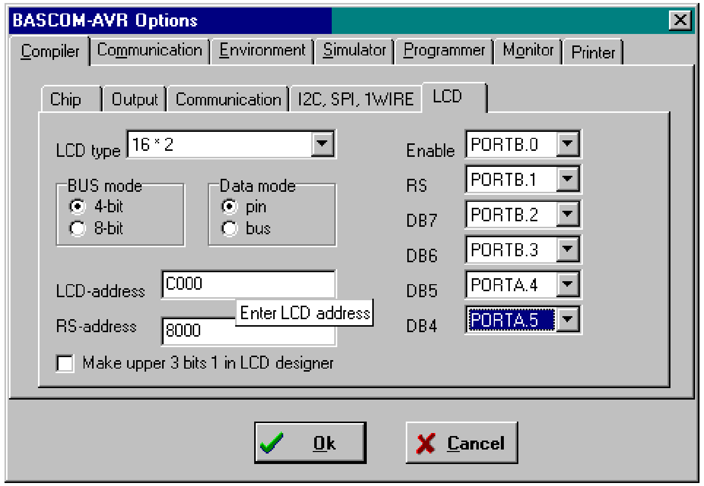
Options Compiler I2C, SPI, 1WIRE



Options Compiler I2C, SPI, 1WIRE

Item	Description
SCL port	Select the port that serves as the SCL-line for the I2C related statements.
SDA port	Select the port that serves as the SDA-line for the I2C related statements.
1WIRE	Select the port that serves as the 1WIRE-line for the 1Wire related statements.
Clock	Select the port that serves as the clock-line for the SPI related statements.
MOSI	Select the port that serves as the MOSI-line for the SPI related statements.
MISO	Select the port that serves as the MISO-line for the SPI related statements.
SS	Select the port that serves as the SS-line for the SPI related statements.
Use hardware SPI	Select to use built-in hardware for SPI, otherwise software emulation of SPI will be used. The 2313 does not have internal HW SPI so can only be used with software spi mode.

Options Compiler LCD

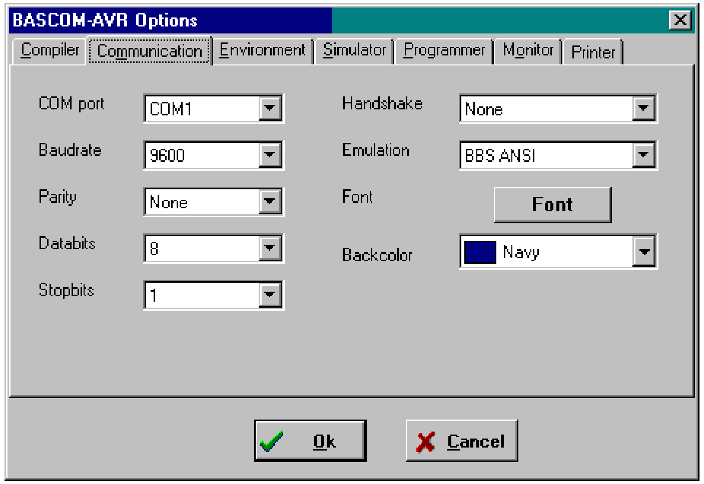


Options Compiler LCD

Item	Description
LCD type	The LCD display used.
Bus mode	The LCD can be operated in BUS mode or in PIN mode. In PIN mode, the data lines of the LCD are connected to the processor pins. In BUS mode the data lines of the LCD are connected to the data lines of the BUS. Select 4 when you have only connect DB4-DB7. When the data mode is 'pin' , you should select 4.
Data mode	Select the mode in which the LCD is operating. In PIN mode, individual processor pins can be used to drive the LCD. In BUS mode, the external data bus is used to drive the LCD.
LCD address	In BUS mode you must specify which address will select the enable line of the LCD display. For the STK200, this is C000 = A14 + A15.
RS address	In BUS mode you must specify which address will select the RS line of the LCD display. For the STK200, this is 8000 = A15
Enable	For PIN mode, you must select the processor pin that is connected to the enable line of the LCD display.
RS	For PIN mode, you must select the processor pin that is connected to the RS line of the LCD display.
DB7-DB4	For PIN mode, you must select the processor pins that are connected to the upper four data lines of the LCD display.
Make upper 3 bits high in LCD designer	Some displays require that for setting custom characters, the upper 3 bits must be 1. Should not be used by default.

OptionsCommunication

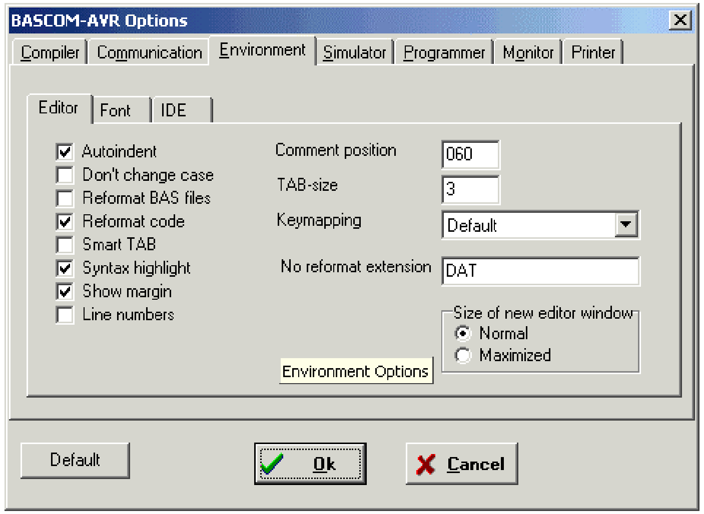
With this option, you can modify the communication settings for the terminal emulator.



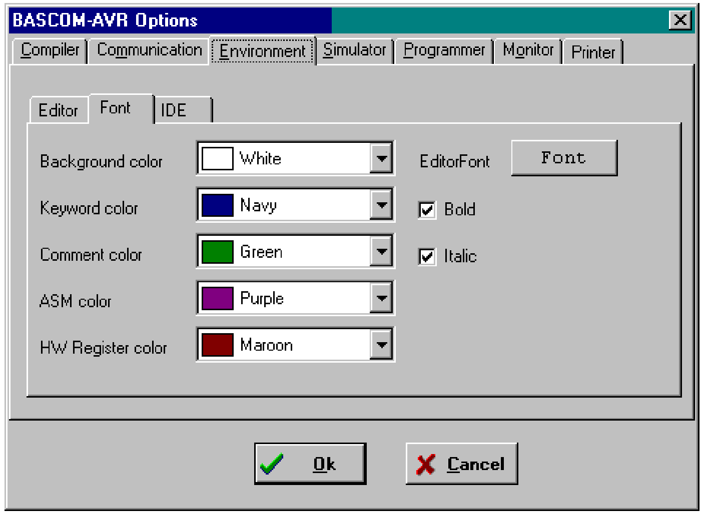
Item	Description
Comport	The communication port of your PC that you use for ther terminal emulator.
Baud rate	The baud rate to use.
Parity	Parity, default None.
Data bits	Number of data bits, default 8.
Stop bits	Number of stop bits, default 1.
Handshake	The handshake used, default is none.
Emulation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

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

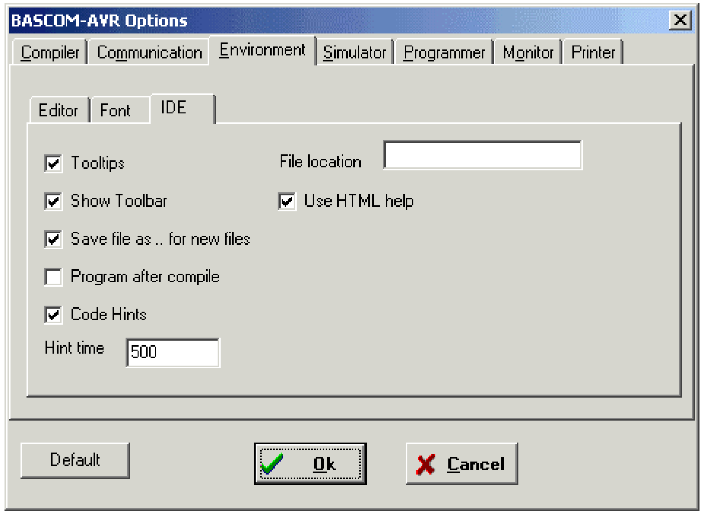
Options Environment



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



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



OPTION	DESCRIPTION
Tooltips	Show tool tips.
Show toolbar	Shows the toolbar with the shortcut icons.
Save File As ... for new files.	Will display a dialogbox so you can give new files a name when they must be saved. When you dont select this option the default name will be give to the file (nonamex.bas). Where x is a number.
Program after Compile	This option will run the programmer after the program is compiled with success.
File location	Double click to select a directory where your program files are stored. By default Windows will use the My Documents path.
Use HTML Help	HTML help is available for download and when your OS supports HTML help, you can turn this option on. W98,W98SE,W98ME and W2000 support HTML Help.
Auto backup	Check this option to make periodic backups. When checked you can specify the backup time in minutes.

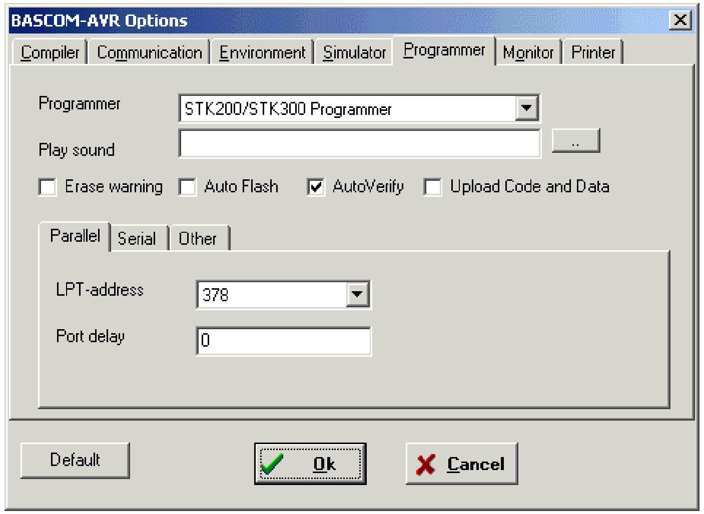
OptionsSimulator

With this option you can modify the simulator settings.

OPTION	DESCRIPTION
Use integrated simulator	Set this option to use BASCOM's simulator. You can also use AVR Studio by clearing this option.
Run simulator after compilation	Run the selected simulator after a successful compilation.
Program	The path with the program name of the simulator.
Parameter	The parameter to pass to the program. {FILE}.OBJ will supply the name of the current program with the extension .OBJ to the simulator.

Options Programmer

With this option you can modify the programmer settings.



OPTION	DESCRIPTION
Programmer	Select one from the list.
Play sound	Name of a WAV file to be played when programming is finished. Press the ..-button to select the file.
Erase Warning	Set this option when you want a confirmation when the chip is erased.
Auto flash	Some programmers support auto flash. Pressing F4 will program the chip without showing the programmer window.
Auto verify	Some programmers support verifying. The chip content will be verified after programming.
Upload code and data	Set this option to program both the FLASH memory and the EEPROM memory
	Parallel printer port programmers
LPT address	Port address of the LPT that is connected to the programmer.
	Serial port programmer
COM port	The com port the programmer is connected to.
STK500 EXE	The path of stk500.exe. This is the full file location to the files stk500.exe that comes with the STK500.
	Other
Use HEX	Select when a HEX file must be sent instead of the bin file.
Program	The program to execute. This is your programmer software.
Parameter	The optional parameter that the program might need. Use {FILE} to insert the binary filename(file.bin) and {EEPROM} to insert the filename of the generated EEP file. When 'Use Hex' is checked the filename (file.hex) will be inserted for {FILE}. In all cases a binary file will be inserted for {EEPROM} with the extension .EEP

OptionsMonitor

With this option you can modify the monitor settings.

OPTION	DESCRIPTION
Upload speed	Selects the baud rate used for uploading
Monitor prefix	String that will be send to the monitor before the upload starts
Monitor suffix	String that us sent to the monitor after the download is completed.
Monitor delay	Time in millions of seconds to wait after a line has been sent to the monitor.
Prefix delay	Time in millions of seconds to wait after a prefix has been sent to the monitor.

OptionsPrinter

With this option you can modify the printer settings.

There are only settings to change the margins of the paper.

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

WindowCascade

Cascade all open editor windows.

WindowTile

Tile all open editor windows.

Window Arrange Icons

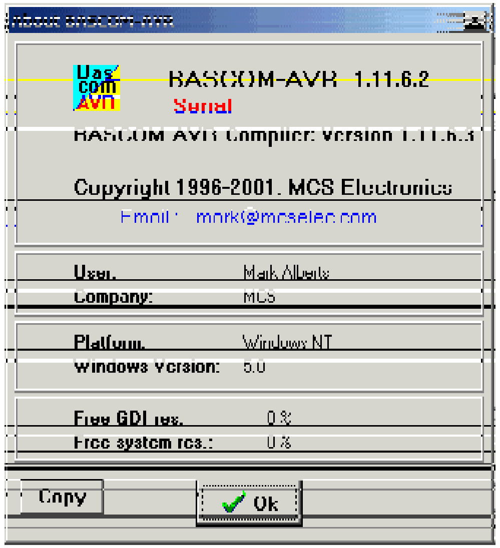
Arrange the icons of the minimized editor windows.

Window Minimize All

Minimize all open editor windows.

Help About

This option shows an about box as showed below.



Your serial number is shown in the about box.

You will need this when you have questions about the product.

The library version is also shown. In this case, it is **1.00**.

You can compare it with the one on our web site in case you need an update.

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

Help Index

Shows the BASCOM help file.

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

Help on Help

Shows help on how to use the Windows help system.

Help Credits

I would like to thank the following people for their contributions to BASCOM:

- ?? Dr.-Ing. Claus Kuehnel for his book 'AVR RISC' , that helped me a lot when I began to study the AVR chips. Check his website at http://www.ckuehnel.ch
- ?? Atmel, who gave permission to use the AVR picture in the start up screen. And for the great tech support. Check their website at http://www.atmel.com
- ?? Brian Dickens, who did most of the Beta testing. He also checked the documentation on grammar and spelling errors.
- ?? Eddie McMullen, who provided me with source code for the new parallel printer port SPI programmer. Check his website with 8051 and AVR related hardware at http://www.eedevl.com
- ?? Jack Tidwell. I used his FP unit. It is the best one I found and it saved lots of work.
- ?? Josef Franz Vögel. He wrote the complete trig FP library.

BASCOM Editor Keys

Key	Action
LEFT ARROW	One character to the left
RIGHT ARROW	One character to the right
UP ARROW	One line up
DOWN ARROW	One line down
HOME	To the beginning of a line
END	To the end of a line
PAGE UP	Up one window
PAGE DOWN	Down one window
CTRL+LEFT	One word to the left
CTRL+RIGHT	One word to the right
CTRL+HOME	To the start of the text
CTRL+END	To the end of the text
CTRL+ Y	Delete current line
INS	Toggles insert/overstrike mode
F1	Help (context sensitive)
F2	Run simulator
F3	Find next text
F4	Send to chip (run flash programmer)
F5	Run
F7	Compile File
F8	Step
F9	Set breakpoint
F10	Run to
CTRL+F7	Syntax Check
CTRL+F	Find text
CTRL+G	Go to line
CTRL+K+x	Toggle bookmark. X can be 1-8
CTRL+L	LCD Designer
CTRL+M	File Simulation
CTRL+N	New File
CTRL+O	Load File
CTRL+P	Print File
CTRL+Q+x	Go to Bookmark. X can be 1-8
CTRL+R	Replace text
CTRL+S	Save File
CTRL+T	Terminal emulator
CTRL+P	Compiler Options
CTRL+W	Show result of compilation
CTRL+X	Cut selected text to clipboard
CTRL+Z	Undo last modification
SHIFT+CTRL+Z	Redo last undo
CTRL+INS	Copy selected text to clipboard
SHIFT+INS	Copy text from clipboard to editor
CTRL+SHIFT+J	Indent Block
CTRL+SHIFT+U	Unindent Block
Select text	Hold the SHIFT key down and use the cursor keys to select text. or keep the left mouse key pressed and tag the cursor over the text to select.

Developing Order

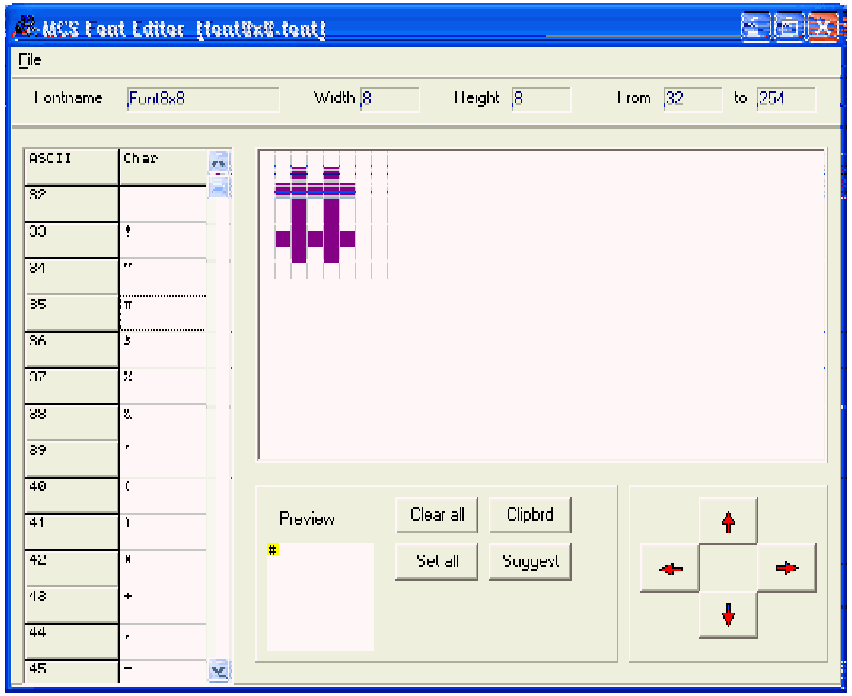
- Start BASCOM;
- Open a file or create a new one;
- Check the chip settings, baud rate and frequency settings for the target system;
- Save the file;
- Com pile the file;
- If an error occurs fix it and recompile (F7);
- Run the simulator;
- Program the chip(F4);

FontEditor

The Font Editor is a Plugin that is intended to create Fonts that can be used with the SED Graphical display.

When you have installed the Font Editor , a menu opion becomes available under the Tools menu : Font Editor.

When you choose this option the following window will appear:



You can open an existing Font file, or Save a modifiedfile.

The font files are installed into the Samples directorie.

You can copy an image form the clipboard and you can move the image up , down, left and right.

When you select a new character, the current character is saved. The suggest option will draw an image of the current selected character.

When you keep the left mouse button pressed, you can set the pixels in the grid. When you keep the right mouse button pressed, you can clear the pixels in the grid.

When you choose the option to create a new Font, you can provide the name of the font, the height of the font in pixels and thw width of the font in pixels.

The Max ASCII is the last ASCII character value you want to use. Each character will occupy space. So it is important that you do not choose a value that is too high and will not be used.

When you display normal text, the maximum number is 127 so it does not make sense to specify a value of 255.

PinOut

This plugin is based on the PinOut Viewer from Karl Jan Skontorp.

You can download the Karl Jan's program from www.mcselec.com/download/appnotes/avr-pins.zip

This program contains all the pictures from the AVR chips.

The PinOut Plugin uses the same pictures , or you can add your own pictures.

When the plugin is selected it will show asmall window :

After you have choosen a picture from the list it will be displayed.

AdditionalHardware

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

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

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

The AVR internal hardware
Attaching an LCD display
Using the I2C protocol
Using the 1WIRE protocol
Using the SPI protocol

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.

CLS, LCD, DISPLAY and other related LCD-statements.

1WRESET, 1WWRITE and 1WREAD

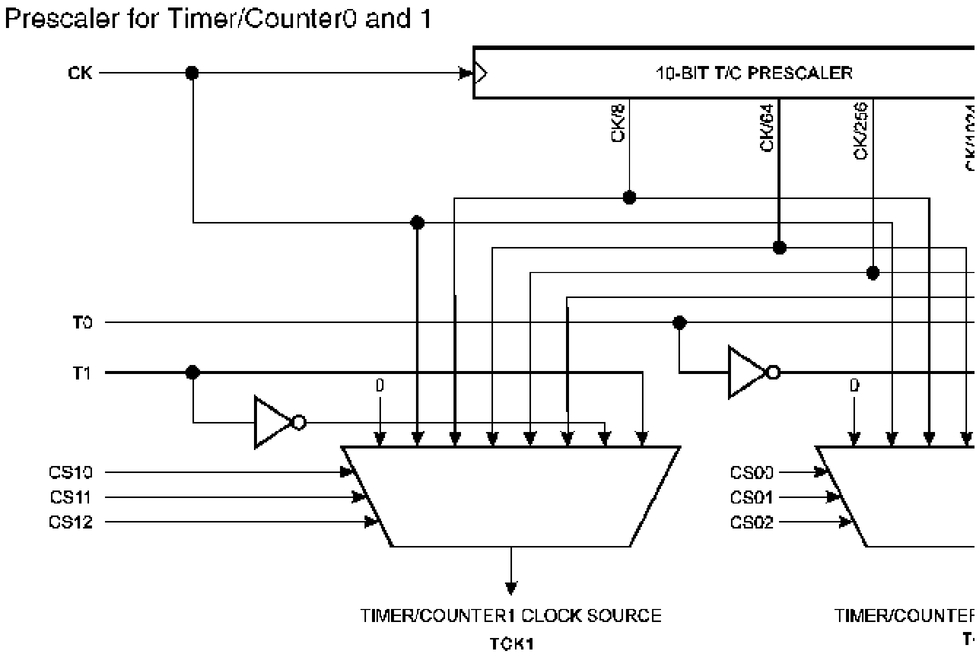
AVR Internal Hardware

The AVR chips all have internal hardware that can be used.

For the description we have used the 8515 so some described hardware will not be available when you select a 2313 for example.

Timer / Counters

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



More about TIMER0
More about TIMER1

The WATCHDOG Timer.

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

PORT B

PORT D

AVR Internal Registers

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

Addr.	Register
$3F	SREG I T H S V N Z C
$3E	SPH SP15 SP14 SP13 SP12 SP11 SP10 SP9 SP8
$3D	SPL SP7 SP6 SP5 SP4 SP3 SP2 SP1 SP0
$3C	Reserved
$3B	GIMSK INT1 INT0- - - - - -
$3A	GIFR INTF1 INTF0
$39	TIMSK TOIE1 OCIE1A OCIE1B - TICIE1 - TOIE0-
$38	TIFR TOV1 OCF1A OCF1B -ICF1 -TOV0-
$37	Reserved
$36	Reserved
$35	MCUCR SRE SRW SE SM ISC11 ISC10 ISC01 ISC00
$34	Reserved
$33	TCCR0 - - - - -CS02 CS01 CS00
$32	TCN T0 Timer/Counter0 (8 Bit)
$31	Reserved
$30	Reserved
$2F	TCCR1A COM1A1 COM1A0 COM1B1 COM1B0 - -PWM11 PWM10
$2E	TCCR1B ICNC1 ICES1 - - CTC1 CS12 CS11 CS10
$2D	TCNT1H Timer/Counter1 - Counter Register High Byte
$2C	TCNT1L Timer/Counter1 - Counter R egister Low Byte
$2B	OCR1AH Timer/Counter1 - Output Compare Register A High Byte
$2A	OCR1AL Timer/Counter1 - Output Compare Register A Low Byte
$29	OCR1BH Timer/Counter1 - Output Compare Register B High Byte
$28	OCR1BL Timer/Counter1 - Output Compare Register B Low Byte
$27	Reserved
$26	Reserved
$25	ICR1H Timer/Counter1 - Input Capture Register High Byte
$24	ICR1L Timer/Counter1 - Input Capture Register Low Byte
$23	Reserved

$22	Reserved
$21	WDTCR - - - WDTOE WDE WDP2 WDP1 WDP0
$20	Reserved
$1F	Reserved - - - - - - - EEAR8
$1E	EEARL EEPROM Address Register Low Byte
$1D	EEDR EEPROM Data Register
$1C	EECR - - - - - EEMWE EEWE EERE
$1B	PORTA PORTA7 PORTA6 PORTA5 PORTA4 PORTA3 PORTA2 PORTA1 PORTA0
$1A	DDRA DDA7 DDA6 DDA5 DDA4 DDA3 DDA2 DDA1 DDA0
$19	PINA PINA7 PINA6 PINA5 PINA4 PINA3 PINA2 PINA1 PINA0
$18	PORTB PORTB7 PORTB6 PORTB5 PORTB4 PORTB3 PORTB2 PORTB1 PORTB0
$17	DDRB DDB7 DDB6 DDB5 DDB4 DDB3 DDB2 DDB1 DDB0
$16	PINB PINB7 PINB6 PINB5 PINB4 PINB3 PINB2 PINB1 PINB0
$15	PORTC PORTC7 PORTC6 PORTC5 PORTC4 PORTC3 PORTC2 PORTC1 PORTC0
$14	DDRC DDC7 DDC6 DDC5 DDC4 DDC3 DDC2 DDC1 DDC0
$13	PINC PINC7 PINC6 PINC5 PINC4 PINC3 PINC2 PINC1 PINC0
$12	PORTD PORTD7 PORTD6 PORTD5 PORTD4 PORTD3 PORTD2 PORTD1 PORTD0
$11	DDRD DDD7 DDD6 DDD5 DDD4 DDD3 DDD2 DDD1 DDD0
$10	PIND PIND7 PIND6 PIND5 PIND4 PIND3 PIND2 PIND1 PIND0
$0F	SPDR SPI Data Register
$0E	SPSR SPIF WCOL - - - - - -
$0D	SPCR SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
$0C	UDR UART I/O Data Register
$0B	USR RXC TXC UDRE FE OR - - -
$0A	UCR RXCIE TXCIE UDRIE RXEN TXEN CHR9 RXB8 TXB8
$09	UBRR UART Baud Rate Register
$08	ACSR ACD - ACO ACI ACIE ACIC ACIS1 ACIS0
$00	Reserved

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

The registers can be used as normal byte variables.

PORTB = 40 will place a value of 40 into port B.

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

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

AVR Internal Hardware TIMER0

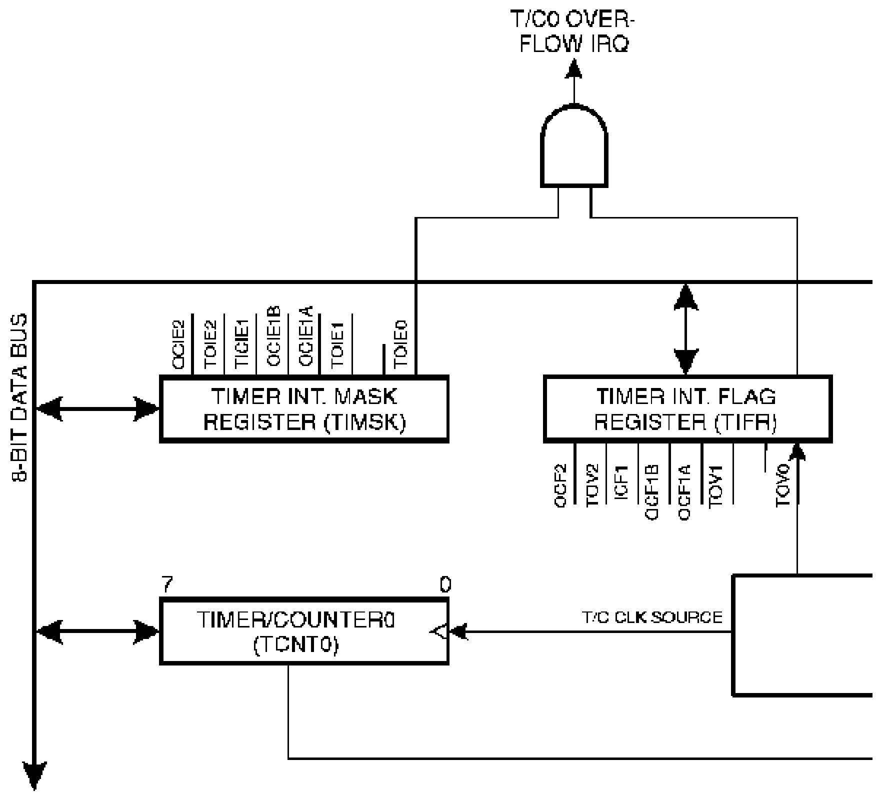
The 8Bit Timer/Counter0

The 8-bit Timer/Counter0 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped.

The overflow status flag is found in the Timer/Counter Interrupt Flag Register - TIFR. Control signals are found in the Timer/Counter0 Control Register - TCCR0. The interrupt enable/disable settings for Timer/Counter0 are found in the Timer/Counter Interrupt Mask Register - TIMSK.

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

Timer/Counter0 Block Diagram



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

AVR Internal Hardware TIMER1

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

The 16-bit Timer/Counter1 can select clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped.

The different status flags (overflow, compare match and capture event) and control signals are found in the Timer/Counter1 Control Registers - TCCR1A and TCCR1B.

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

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

The external clock signal is sampled on the rising edge of the internal CPU clock.

The 16-bit Timer/Counter1 features both a high resolution and a high accuracy usage with the lower prescaling opportunities.

Similarly, the high prescaling opportunities make the Timer/Counter1 useful for lower speed functions or exact timing functions with infrequent actions.

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

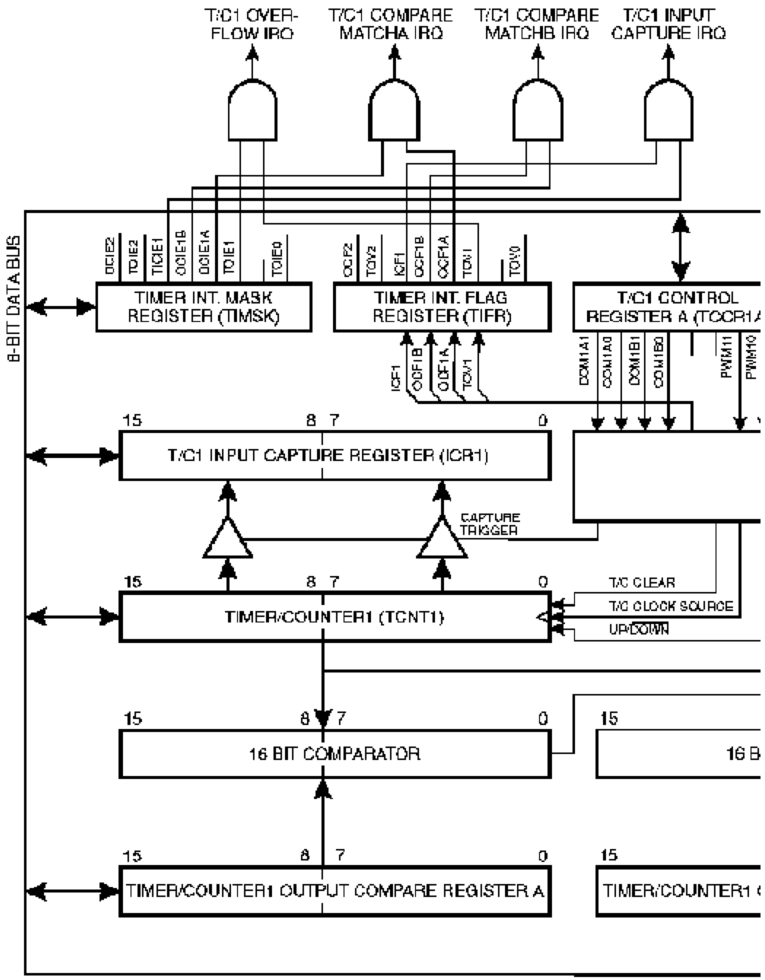
The Output Compare functions include optional clearing of the counter on compareA match, and actions on the Output Compare pins on both compare matches.

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

The Input Capture function of Timer/Counter1 provides a capture of the Timer/Counter1 contents to the Input Capture Register - ICR1, triggered by an external event on the Input Capture Pin - ICP. The actual capture event settings are defined by the Timer/Counter1 Control Register -TCCR1B.

In addition, the Analog Comparator can be set to trigger the Input Capture.

Timer/Counter1 Block Diagram



AVR Internal Hardware Watchdog timer

The Watchdog Timer

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

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

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

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

AVR Internal Hardware Port B

Port B

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

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

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

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

Port B Pins Alternate Functions

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

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

PortB As General Digital I/O

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

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

DDBn Effects on Port B Pins

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

AVR Internal Hardware Port D

Port D

Port D Pins Alternate Functions

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

RD - PORTD, Bit 7

RD is the external data memory read control strobe.

WR - PORTD, Bit 6

WR is the external data memory write control strobe.

OC1- PORTD, Bit 5

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

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

INT1 - PORTD, Bit 3

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

INT0 - PORTD, Bit 2

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

TXD - PORTD, Bit 1

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

RXD - PORTD, Bit 0

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

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

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

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

RESET UCR.3

RESET UCR.4

Adding XRAM

Some AVR chips like the 8515 for example can be extended with external RAM memory.

On these chips Port A serves as a Multiplexed Address/Data input/output.
Port C also serves as Address output when using external SRAM.

The maximum size of a XRAM chip can be 64Kbytes.

The STK200 has a 62256 ram chip (32K x 8 bit).

Here is some info from the BASCOM userlist :

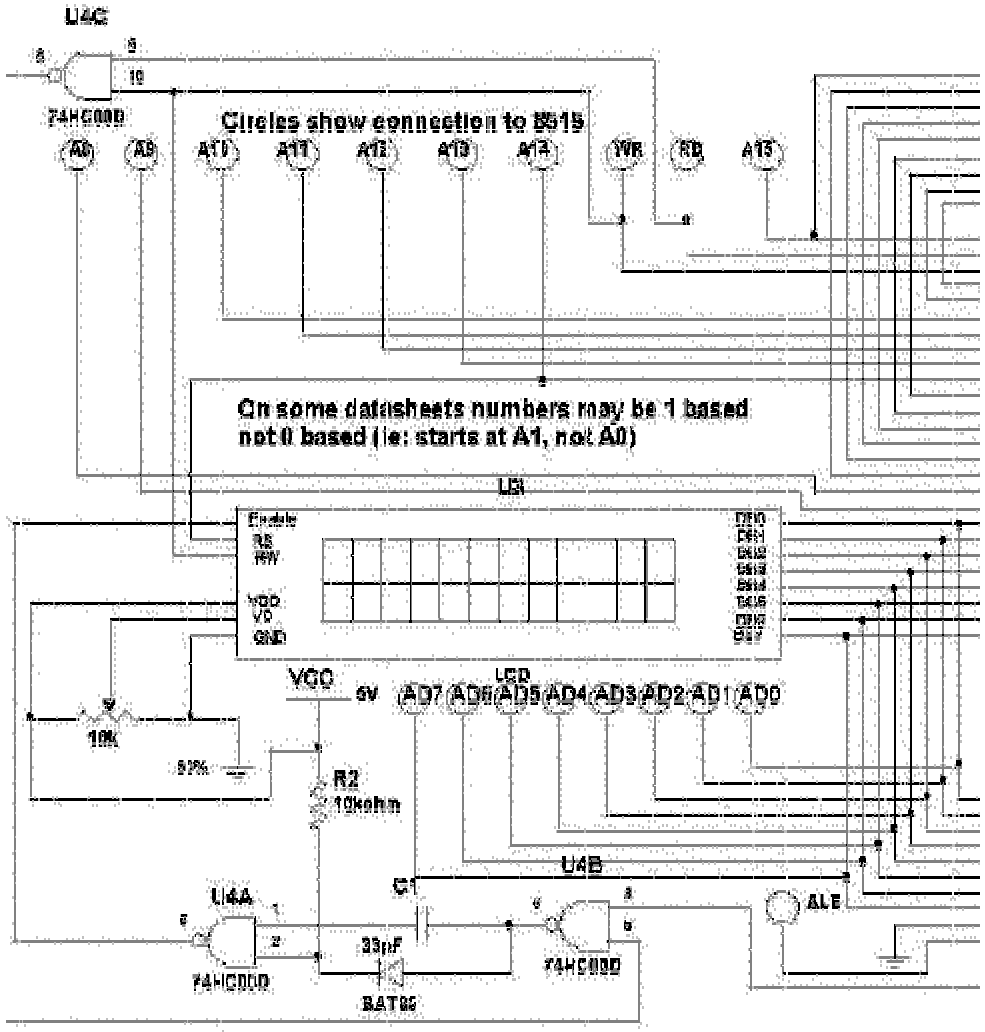
If you do go with the external ram , be careful of the clock speed.
Using a 4Mhz crystal , will require a SRAM with 70nS access time
or better. Also the data latch (74HC573) will have to be from a faster
family such as a 74FHC573 if you go beyond 4Mhz.

You can also program an extra wait state, which slow it down a bit.

Here you find a pdf file showing STK200 schematics:
http://www.avr -forum.com/Stk200_schematic.pdf

If you use 32kRAM, then connect the /CS signal to A15 which give
to the range of &H0000 to &H7FFF, if you use a 64kRAM, then
tie /CS to GND, so the RAM is selected all the time.

Thanks to Colin O'Flynn for creating this circuit :



Attaching an LCD Display

A LCD display can be connected with two methods.

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

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

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

This leaves PORTB.1 and PORTB.0 and PORTD for other purposes.

You can change these settings from the Options LCD menu.

BASCOM supports many statements to control the LCD-display.

For those who want to have more control the example below shows how to use the internal routines.

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

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

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

Memory usage

Every variable uses memory. This memory is also called SRAM.

The available memory depends on the chip.

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

Almost all registers are used by the compiler or might be used in the future.

Which registers are used depends on the statements you used.

This brings us back to the SRAM.

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

Some statements might use some SRAM. When this is the case it is mentioned in the help topic of that statement.

Each 8 used bits occupy one byte.

Each byte occupies one byte.

Each integer/word occupies two bytes.

Each Long or Single occupies four bytes.

Each String occupies at least 2 byes.

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

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

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

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

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

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

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

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

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

XRAM

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

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

But because the XRAM can only start after the SRAM, which is **&H** 0260, the lower memory locations of the XRAM will not be used.

ERAM

Most AVR chips have internal EEPROM on board.

This EEPROM can be used to store and retrieve data.

In BASCOM, this data space is called ERAM.

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

Constant code usage

Constants are stored in a constant table.

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

For example:

Print "ABCD"

Print "ABCD"

This example will only store one constant (ABCD).

Print "ABCD"

Print "ABC"

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

Using the I2C protocol

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

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

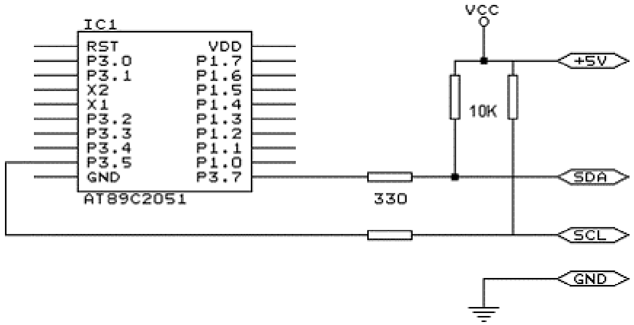
The PCF8574 is a nice chip - it is an I/O extender with 8 pins that you can use either as input or output.

The design below shows how to implement an I2C-bus. The circuit shown is for the 8051 micro the AT89C2051 which is pin compatible with the AT90S2313. It also works for the AVR.

R1 and R2 are 330 ohm resistors.

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

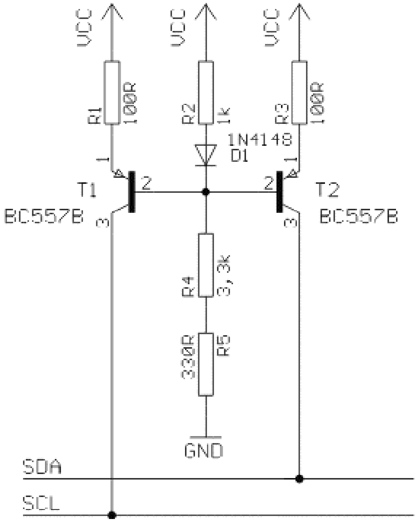
You can select which port pins you want to use for the I2C interface with the compiler settings.



The following information was submitted by Detlef Queck.

"Many people have over and over problems width I2C(TWI) Termination, use 4,7k or 10 k pullup? How long can the SCL,SDA Line width used pullup's go etc.etc.

You can bring this incredible situation's down. Here is a Schematic for an active Termination of I2C and TWI. We have this Schematic used for over 10 years, and have no problem's with it. The I2C (TWI) Line's can be up to 80cm(400KHz) without any problem when the Terminator is at the end of the Lines."



Using the 1 WIRE protocol

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

This topic is written by Göte Haluza. He tested the new 1wire search routines and is building a weather station. 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 1w-bus, please go to http://www.dalsemi.com. 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 devices(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 stabilizing 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 do 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 simultaneously 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 <u>some</u> limitation 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 problem. They also show examples with 300m cabling, and I think I have seen something with 600-meter bus (but I cant find it again).

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 will 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) that also has to count as a benefit - maybe one of the largest. ;-)

The disadvantages of the 1w-bus

So far as I know, DS is the only manufacturer of sensors for the bus. Some people think their devices are expensive. And, until now, it was really difficult to communicate with the devices. Particularly when using the benefit of several devices on one bus. Still some people say that the 1w-bus is slow - but I don't think so.

Göte Haluza
System engineer

Using the SPI protocol

General description of the SPI

The SPI allows high-speed synchronous data transfer between the AVR and peripheral devices or between several AVR devices. On most parts the SPI has a second purpose where it is used for In System Programming (ISP).

The interconnection between two SPI devices always happens between a master device and a slave device. Compared to some peripheral devices like sensors which can only run in slave mode, the SPI of the AVR can be configured for both master and slave mode.

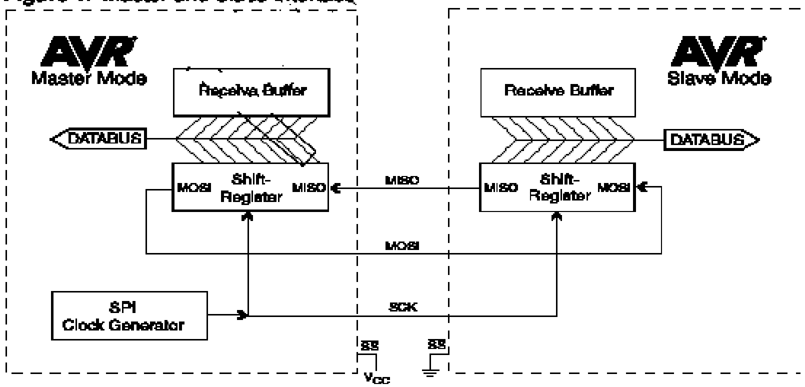
The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR).

Special considerations about the /SS pin have to be taken into account. This will be described later in the section "Multi Slave Systems - /SS pin Functionality".

The master is the active part in this system and has to provide the clock signal a serial data transmission is based on. The slave is not capable of generating the clock signal and thus can not get active on its own.

The slave just sends and receives data if the master generates the necessary clock signal. The master however generates the clock signal only while sending data. That means that the master has to send data to the slave to read data from the slave.

Figure 1. Master and Slave Interface



Data transmission between Master and Slave

The interaction between a master and a slave AVR is shown in Figure 1. Two identical SPI units are displayed. The left unit is configured as master while the right unit is configured as slave. The MISO, MOSI and SCK lines are connected with the corresponding lines of the other part.

The mode in which a part is running determines if they are input or output signal lines. Because a bit is shifted from the master to the slave and from the slave to the master simultaneously in one clock cycle both 8-bit shift registers can be considered as one 16-bit circular shift register. This means that after eight SCK clock pulses the data between master and slave will be exchanged.

The system is single buffered in the transmit direction and double buffered in the receive direction. This influences the data handling in the following ways:

1. New bytes to be sent can not be written to the data register (SPDR) / shift register before the entire shift cycle is completed.

2. Received bytes are written to the Receive Buffer immediately after the transmission is completed.

3. The Receive Buffer has to be read before the next transmission is completed or data will be lost.

4. Reading the SPDR will return the data of the Receive Buffer.

After a transfer is completed the SPI Interrupt Flag (SPIF) will be set in the SPI Status Register (SPSR). This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the SPI Interrupt Enable (SPIE) bit in the SPCR enables the interrupt of the SPI while setting the I bit in the SREG enables the global interrupts.

Pins of the SPI

The SPI consists of four different signal lines. These lines are the shift clock (SCK), the Master Out Slave In line (MOSI), the Master In Slave Out line (MISO) and the active low Slave Select line (/SS). When the SPI is enabled, the data direction of the MOSI, MISO, SCK and /SS pins are overridden according to the following table.

Table 1. SPI Pin Overrides

Pin Direction Overrides	Master SPI Mode Direction Overrides	Slave SPI Modes
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
SS	User Defined	Input

This table shows that just the input pins are automatically configured. The output pins have to be initialized manually by software. The reason for this is to avoid damages e.g. through driver contention.

Multi Slave Systems - /SS pin Functionality

The Slave Select (/SS) pin plays a central role in the SPI configuration. Depending on the mode the part is running in and the configuration of this pin, it can be used to activate or deactivate the devices. The /SS pin can be compared with a chip select pin which has some extra features. In master mode, the /SS pin must be held high to ensure master SPI operation if this pin is configured as an input pin. A low level will switch the SPI into slave mode and the hardware of the SPI will perform the following actions:

1. The master bit (MSTR) in the SPI Control Register (SPCR) is cleared and the SPI system becomes a slave. The direction of the pins will be switched according to Table 1.

2. The SPI Interrupt Flag (SPIF) in the SPI Status Register (SPSR) will be set. If the SPI interrupt and the global interrupts are enabled the interrupt routine will be executed. This can be useful in systems with more than one master to avoid that two masters are accessing the SPI bus at the same time. If the /SS pin is configured as output pin it can be used as a general purpose output pin which does not affect the SPI system.

Note: In cases where the AVR is configured for master mode and it can not be ensured that the /SS pin will stay high between two transmissions, the status of the MSTR bit has to be checked before a new byte is written. Once the MSTR bit has been cleared by a low level on the /SS line, it must be set by the application to re-enable SPI master mode.

In slave mode the /SS pin is always an input. When /SS is held low, the SPI is activated and MISO becomes output if configured so by the user. All other pins are inputs. When /SS is driven high, all pins are inputs, and the SPI is pas-sive, which means that it will not receive incoming data.

Table 2 shows an overview of the /SS Pin Functionality.

Note: In slave mode, the SPI logic will be reset once the /SS pin is brought high. If the /SS pin is brought high during a transmission, the SPI will stop sending and receiving immediately and both data received and data sent must be considered as lost.

TABLE 2. Overview of SS pin.

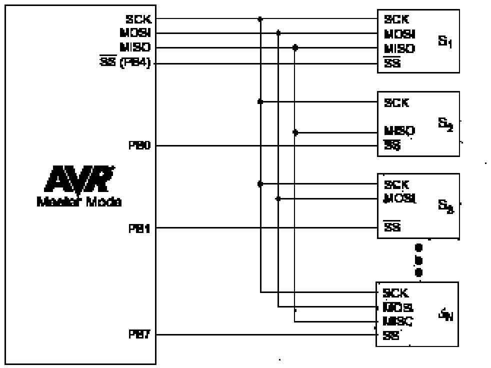
Mode	/SS Config	/SS Pin level	Description
Slave	Always input	High	Slave deactivated

		Low	Slave activated
Master	Input	High	Master activated
		Low	Master deactivated
	Output	High	Master activated
		Low	

As shown in Table 2, the /SS pin in slave mode is always an input pin. A low level activates the SPI of the device while a high level causes its deactivation. A Single Master Multiple Slave System with an AVR configured in master mode and /SS configured as output pin is shown in Figure 2. The amount of slaves, which can be connected to this

AVR is only limited by the number of I/O pins to generate the slave select signals.

Figure 2. Multi Slave System



The ability to connect several devices to the same SPI-bus is based on the fact that only one master and only one slave is active at the same time. The MISO, MOSI and SCK lines of all the other slaves are tristated (configured as input pins of a high impedance with no pullup resistors enabled). A false implementation (e.g. if two slaves are activated at the same time) can cause a driver contention which can lead to a CMOS latchup state and must be avoided. Resistances of 1 to 10 k ohms in series with the pins of the SPI can be used to prevent the system from latching up. However this affects the maximum usable data rate, depending on the loading capacitance on the SPI pins.

Unidirectional SPI devices require just the clock line and one of the data lines. If the device is using the MISO line or the MOSI line depends on its purpose. Simple sensors for instance are just sending data (see S2 in Figure 2), while an external DAC usually just receives data (see S3 in Figure 2).

SPI Timing

The SPI has four modes of operation, 0 through 3. These modes essentially control the way data is clocked in or out of an SPI device. The configuration is done by two bits in the SPI control register (SPCR). The clock polarity is speci-fied by the CPOL control bit, which selects an active high or active low clock. The clock phase (CPHA) control bit selects one of the two fundamentally different transfer formats. To ensure a proper communication between master and slave both devices have to run in the same mode. This can require a reconfiguration of the master to match the requirements of different peripheral slaves.

The settings of CPOL and CPHA specify the different SPI modes, shown in Table 3. Because this is no standard and specified different in other literature, the configuration of the SPI has to be done carefully.

Table 3. SPI Mode configuration

SPI Mode	CPOL	CPHA	Shift SCK edge	Capture SCK edge
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

The clock polarity has no significant effect on the transfer format. Switching this bit causes the clock signal to be inverted (active high becomes active low and idle low

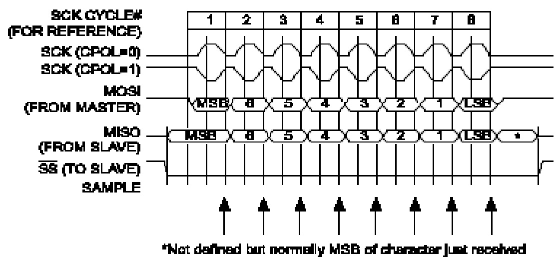
becomes idle high). The settings of the clock phase, how-ever, selects one of the two different transfer timings, which are described closer in the next two chapters. Since the MOSI and MISO lines of the master and the slave are directly connected to each other, the diagrams show the timing of both devices, master and slave. The /SS line is

the slave select input of the slave. The /SS pin of the master is not shown in the diagrams. It has to be inactive by a high level on this pin (if configured as input pin) or by configuring it as an output pin.

A.) CPHA = 0 and CPOL = 0 (Mode 0) and CPHA = 0 and CPOL = 1 (Mode 1)

The timing of a SPI transfer where CPHA is zero is shown in Figure 3. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.

Figure 3. SPI Transfer Format with CPHA = 0



When the SPI is configured as a slave, the transmission starts with the falling edge of the /SS line. This activates the SPI of the slave and the MSB of the byte stored in its data register (SPDR) is output on the MISO line. The actual transfer is started by a software write to the SPDR of the master. This causes the clock signal to be generated. In cases where the CPHA equals zero, the SCK signal remains zero for the first half of the first SCK cycle. This ensures that the data is stable on the input lines of both the master and the slave. The data on the input lines is read with the edge of the SCK line from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one). The edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one) causes the data to be shifted one bit further so that the next bit is output on the MOSI and MISO lines.

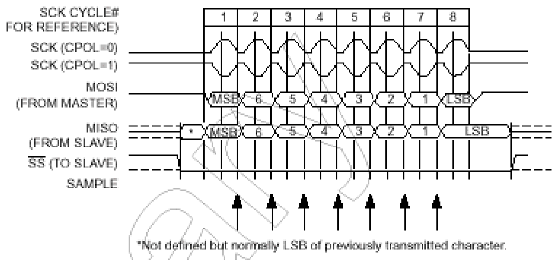
After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to thereceive buffer.

B.) CPHA = 1 and CPOL = 0 (Mode 2) and CPHA = 1 and

CPOL = 1 (Mode 3)

The timing of a SPI transfer where CPHA is one is shown in Figure 4. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.

Figure 4. SPI Transfer Format with CPHA = 1



Like in the previous cases the falling edge of the /SS lines selects and activates the slave. Compared to the previous cases, where CPHA equals zero, the transmission is not started and the MSB is not output by the slave at this stage. The actual transfer is started by a software write to the SPDR of the master what causes the clock signal to be generated. The first edge of the SCK signal from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one) causes both the master and the slave to output the MSB of the byte in the SPDR.

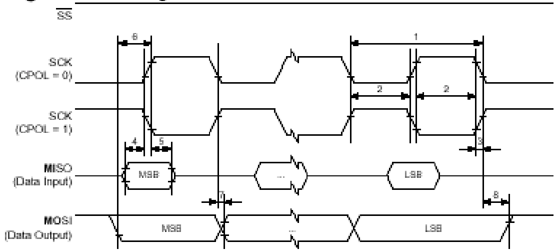
As shown in Figure 4, there is no delay of half a SCK-cycle like in Mode 0 and 1. The SCK line changes its level immediately at the beginning of the first SCK-cycle. The data on the input lines is read with the edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one).

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

Considerations for high speed transmissions

Parts which run at higher system clock frequencies and SPI modules capable of running at speed grades up to half the system clock require a more specific timing to match the needs of both the sender and receiver. The following two diagrams show the timing of the AVR in master and in slave mode for the SPI Modes 0 and 1. The exact values of the displayed times vary between the different pars and are not an issue in this application note. However the function-ality of all parts is in principle the same so that the following considerations apply to all parts.

Figure 5. Timing Master Mode



The minimum timing of the clock signal is given by the times "1" and "2". The value "1" specifies the SCK period while the value "2" specifies the high / low times of the

clock signal. The maximum rise and fall time of the SCK signal is specified by the time "3". These are the first timings of the AVR to check if they match the requirements of

the slave.

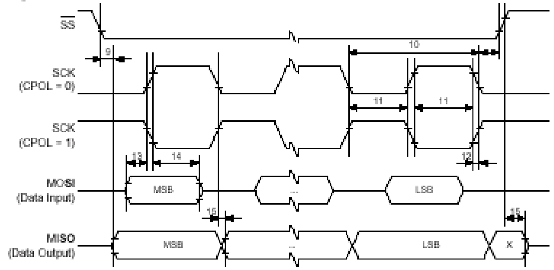
The Setup time "4" and Hold time "5" are important times because they specify the requirements the AVR has on the interface of the slave. These times determine how long before the clock edge the slave has to have valid output data ready and how long after the clock edge this data has to be valid.

If the Setup and Hold time are long enough the slave suits to the requirements of the AVR but does the AVR suit to the requirements of the slave?

The time "6" (Out to SCK) specifies the minimum time the AVR has valid output data ready before the clock edge occurs. This time can be compared to the Setup time "4" of the slave.

The time "7" (SCK to Out) specifies the maximum time after which the AVR outputs the next data bit while the time "8" (SCK to Out high) the minimum time specifies during which the last data bit is valid on the MOSI line after the SCK was set back to its idle state.

Figure 6. Timing Slave Mode



In principle the timings are the same in slave mode like previously described in master mode. Because of the switching of the roles between master and slave the requirements on the timing are inverted as well. The minimum times of the master mode are now maximum times and vice versa.

SPI Transmission Conflicts

A write collision occurs if the SPDR is written while a transfer is in progress. Since this register is just single buffered in the transmit direction, writing to SPDR causes data to be written directly into the SPI shift register. Because this write operation would corrupt the data of the current transfer, a write-collision error in generated by setting the WCOL bit in the SPSR. The write operation will not be executed in this case and the transfer continues undisturbed. A write collision is generally a slave error because a slave has no control over when a master will initiate a transfer. A master, however, knows when a transfer is in progress. Thus a master should not generate write collision errors, although the SPI logic can detect these errors in a master as well as in a slave mode.

When you set the SPI option from the Options, Compiler, SPI menu SPCR will be set to 01010100 which means ; enable SPI, master mode, CPOL = 1

When you want to control the various options with the hardware SPI you can use the CONFIG SPI statement.

Power Up

At power up all ports are in Tri-state and can serve as input pins.

When you want to use the ports (pins) as output, you must set the data direction first with the statement : CONFIG PORTB = OUTPUT

Individual bits can also be set to be used as input or output.

For example : DDRB = **&B**00001111 , will set a value of 15 to the data direction register of PORTB.

PORTB.0 to PORTB.3 (the lower 4 bits) can be used as outputs because they are set high. The upper four bits (PORTB.4 to PORTB.7), can be used for input because they are set low.

You can also set the direction of a port pin with the statement :

CONFIG PINB.0 = OUTPUT | INPUT

The internal RAM is cleared at power up or when a reset occurs. Use $NORAMCLEAR to disable this feature.

Supported Programmers

BASCOM supports the following programmers

AVR ICP910 based on the AVR910.ASM application note

STK200 ISP programmer from Atmel/Kanda

The PG302 programmer from Iguana Labs

The simple cable programmer from Sample Electronics.

Eddie McMullen's SPI programmer.

KITSRUS KIT122 Programmer

MCS Universal Interface Programmer

STK500 programmer and Extended STK500 programmer.

Lawicel BootLoader

ISP programmer

BASCOM supports the STK200 and STK200+ and STK300 ISP programmer from Kanda.

This is a very reliable parallel printer port programmer.

The STK200 ISP programmer is included in the STK200 starter kit.

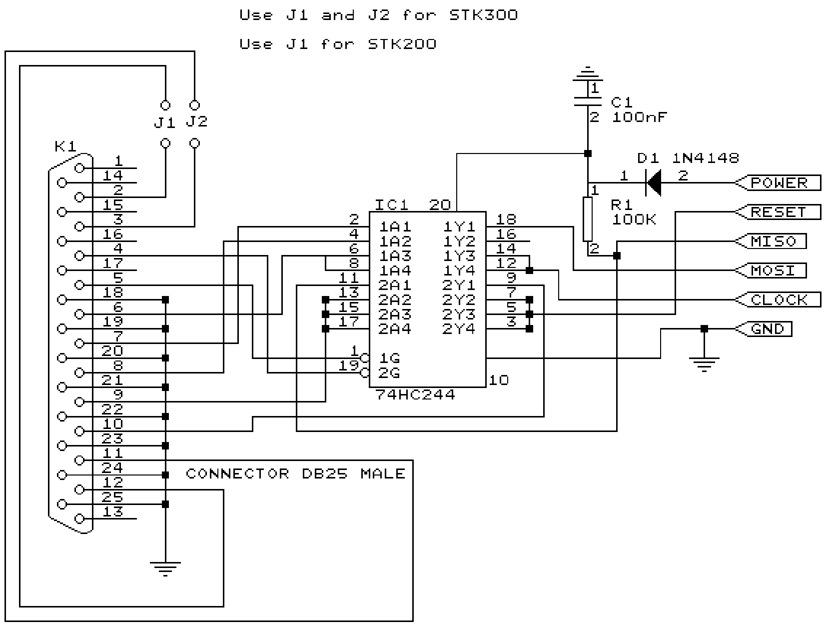
All programs were tested with the STK200.

For those who don't have this kit and the programmer the following schematic shows how to make your own programmer:

The dongle has a chip with no identification but since the schematic is all over the web, I have included it. Kanda also sells a very cheap separate programmer dongle. So I suggest you buy this one!

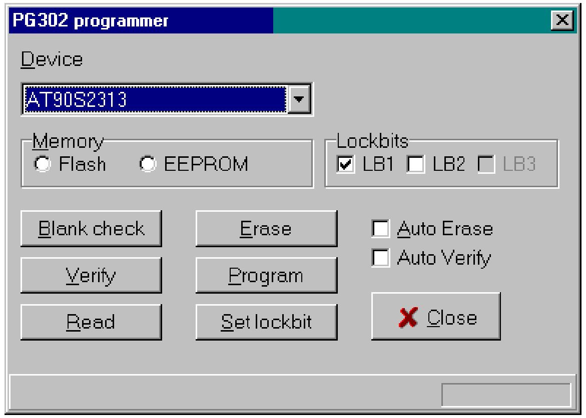
Here is a great tip received from a user :

If the parallel port is disconnected from the interface and left floating, the '244 latch outputs will waver, causing your microcontroller to randomly reset during operation. The simple addition of a 100K pull-up resisistor between pin 1 and 20 of the latch, and another between pin 19 and 20, will eliminate this problem. You'll then have HIGH-Z on the latch outputs when the cable is disconnected (as well as when it's connected and you aren't programming), so you can use the MOSI etc. pins for I/O.



PG302 programmer

The PG302 is a serial programmer. It works and looks exactly as the original PG302 software.



Select the programmer from The Option Programmer menu or right click on the  button to show the [Option Programmer](#) menu.

Sample Electronics cable programmer

Sample Electronics submitted the simple cable programmer.
They produce professional programmers too. This simple programmer you can make yourself within 10 minutes.

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:

DB25 pin	Target MCU pin(AT90S8535)	Target MCU M103/M128	Target MCU pin 8515	DT104
2, D0	MOSI, pin 6	PE.0, 2	MOSI, 6	J5, pin 4
4, D2	RESET, pin 9	RESET, 20	RESET, 9	J5, pin 8
5, D3	CLOCK, pin 8	PB.1,11	CLOCK, 8	J5, pin 6
11, BUSY	MISO, pin 7	PE.1, 3	MISO, 7	J5, pin 5
18-25,GND	GROUND	GROUND	GND,20	J5, pin 1

The MCU pin numbers are shown for an 8535! And 8515

Note that 18-25 means pins 18,19,20,21,22,23,24 and 25
You can use a small resistor of 100-220 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 don't 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 pull-up. 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

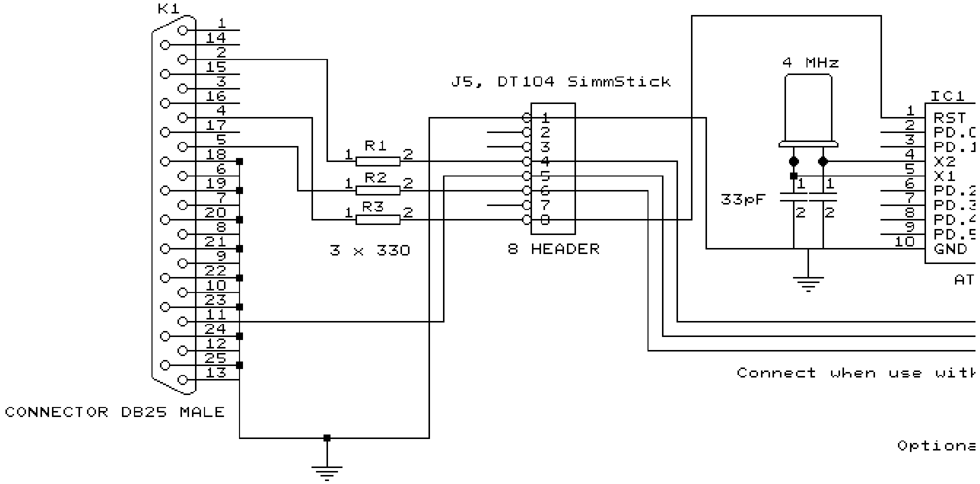
"During this first phase of the programming cycle, keeping the SCK line free from pulses is critical, as pulses will cause the target AVR to loose synchronization with the programmer. When synchronization is lost, the only means of regaining synchronization is to release the RESET line for more than 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.

Regards,

Mark Hayne



KITSRUS Programmer

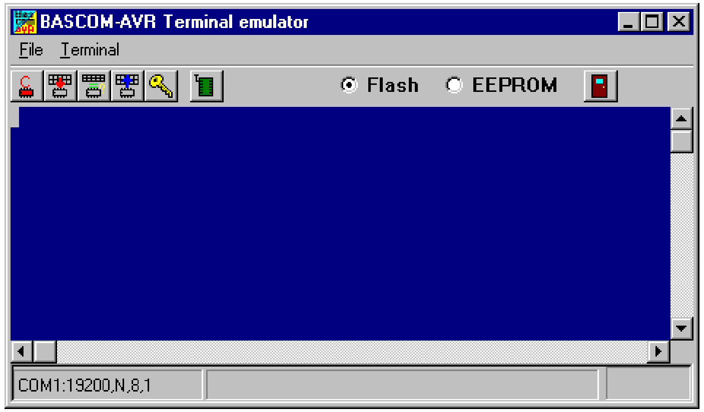
The K122 is a KIT from KITSRUS. (www.kitsrus.com)

The programmer supports the most popular 20 and 40 pins AVR chips.

On the Programmer Options tab you must select this programmer and the COM port it is connected to.

On the Monitor Options tab you must specify the upload speed of 9600, Monitor delay of 1 and Prefix delay 1.

When you press the Program button the Terminal Emulator screen will pop up:



A special toolbar is now visible.

You must press the Program enable button to enable the programmer.

When you enable the programmer the right baud rate will be set.

When you are finished you must press the Enable button again to disable it.

This way you can have a micro connected to your COM port that works with a different BAUD rate.

There is an option to select between FLASH and EEPROM.

The prompt will show the current mode which is set to FLASH by default.

The buttons on the toolbar allow you to :

ERASE, PROGRAM, VERIFY, DUMP and set the LOCK BITS.

When DUMP is selected you will be asked for a file name.

When the DUMP is ready you must CLOSE the LOGFILE where the data is stored. This can be done to select the CLOSE LOGFILE option form the menu.

MCS Universal Interface Programmer

The MCS Universal Interface programmer allows you to customize the pins that are used for the ISP interface. The file prog.settings stores the various interfaces.

The content :

```
;how to use this file to add support for other programmers
;first create a section like [newprog]
; then enter the entries:
; BASE= $hexaddress
; MOSI= address in form of BASE[+offset] , bit [,inverted]
; CLOCK= same as MOSI
; RESET=same as MOSI
; MISO=same as MOSI
; The bit is a numer that must be written to set the bit
; for example 128 to set bit 7
; Optional is ,INVERTED to specify that inverse logic is used
; When 128 is specified for the bit, NOT 128 will be written(127)


[FUTURELEC]
;tested and ok
BASE=$378

MOSI=BASE+2,1,inverted
CLOCK=BASE,1
RESET=BASE,2
MISO=BASE+1,64

[sample]
;tested and ok
BASE=$378

MOSI=BASE,1
CLOCK=BASE,8
RESET=BASE,4
MISO=BASE+1,128,INVERTED

[stk200]
;tested and ok
BASE=$378

MOSI=BASE,32
CLOCK=BASE,16
RESET=BASE,128
MISO=BASE+1,64
```

Four programmers are supported : Futurelec, Sample and STK200/STK300 and WinAVR/ SP12.

To add your own programmer open the file with notepad and add a new section name. For the example I will use stk200 that is already in the file.

[stk200]

The LPT base address must be specified. For LPT1 this is in most cases $378. $ means hexadecimal.

The pins that are needed are MOSI, CLOCK, RESET and MISO.

Add the pin name MOSI =

After the pin name add the address of the register. For the STK200 the data lines are used so BASE must be specified. After the address of the register, specify the bit number value to set the pin high. Pin 0 will be 1, pin 1 would be 2, pin 2 would be 4 etc. D5 is used for the stk so we specify 32.

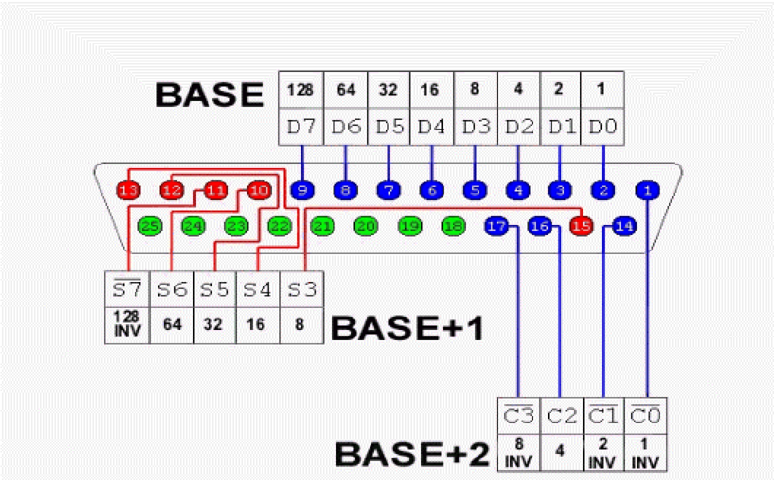
When the value is set by writing a logic 0, also specify, INVERTED.

After you have specified all pins, save the file and restart BASCOM.

Select the Universal Programmer Interface and select the entry you created.

After you have selected an entry save your settings and exit BASCOM. At the next startup of BASCOM, the settings will be used.

The following picture shows the LPT connector and the relation of the pins to the LPT registers.



Always add your entry to the bottom of the file and email the settings to avr@mcselec.com so it can be added to BASCOM.

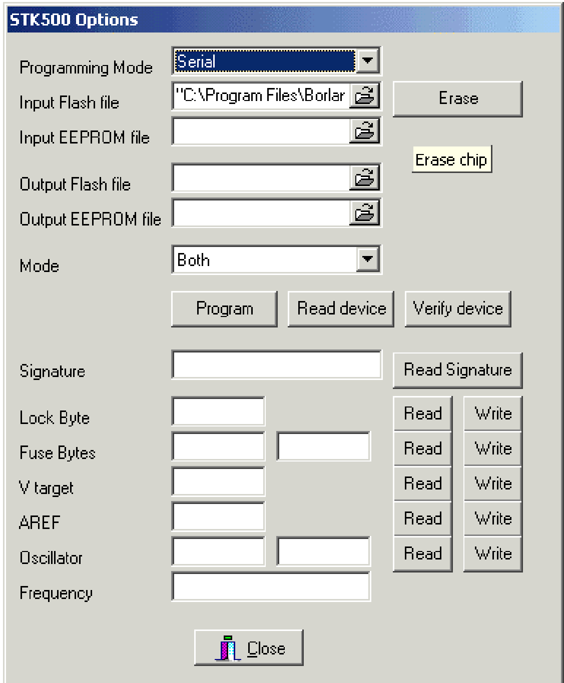
STK500Programmer

When you select the STK500 programmer, BASCOM will run the files named stk500.exe that is installed with AVR Studio.

That is why you have to specify the file location of the stk500.exe

The normal STK500 support will erase, and program the flash.

The extended STK500 support will show the following window:



Option	Description
Programming mode	Serial or parallel. Some options require the parallel mode.
Input Flash File	The program HEX file. It is loaded automatic
Input EEPROM File	The program EEP file. It is loaded automatic when it exists
Output Flash File	The name of the output flash file. Only needed when you want to read a device.
Output EEPROM File	The name of the output EEPROM file. Only needed when you want to read the EEPROM from a device
Mode	Both will work on the FLASH and EEPROM, Flash will only work on the FLASH ROM and EEPROM will only work on the EEPROM. Use both when you want to program both a program and an

	EEPROM (EEP) file.
Erase	Erase chip. Must be done before programming the chip.
Program	Program the chip
Read device	Read the flash and or EEPROM content and store in the specified files.
Verify device	Verify the chip content with the files.
Read signature	Read the signature bytes that identify the chip.
Read/Write Lock Byte	Read and write the lock byte. Hex notation!
Read/Write Fuse Bytes	Read and write the fuse bytes. Hex notation!
Read/Write Vtarget	Read or set the Vtarget voltage
Read/Write Aref	Read or set the Aref voltage
Read/Write oscillator	Read or write the oscillator settings
Read/Write frequency	Read or set the board frequency

All options will set the command line parameters. A file named stk500.cmd will be created by the compiler with the proper syntax.

This file will be executed and the result is stored in the stk500.log file.

Lawicel BootLoader

The Lawicel Bootloader must be used with the StAVeR. The StAVeR contains a bootloader so you only need a serial interface, no parallel programmer or other programmers.

You can also use Hyperterminal.

When you have selected the Lawicel Bootloader from the Options, Programmer, the following window will appear when you press F4.

As the window suggests, press the reset button on the activity board or StAVeR, and the chip will be programmed. This is visible by a second wind that will be shown during programming.

When the programming succeeds, both windows will be closed.

When an error occurs, you will get an error message and you can clock the Cancel button in order to return to the Editor.

AVR ISP Programmer

The AVRISP programmer is AVR ICP910 based on the AVR910.ASM application note.

The old ICP910 does not support Mega chips. Only a modified version of the AVR910.ASM supports Universal commands so all chips can be programmed.

The new AVRISP from Atmel that can be used with AVR Studio, is not compatible!

When you do not want to use the default baud rate that AVR910 is using, you can edit the file bascavr.ini from the Windows directory.

Add the section [AVRISP]

Then add : COM=19200,n,8,1

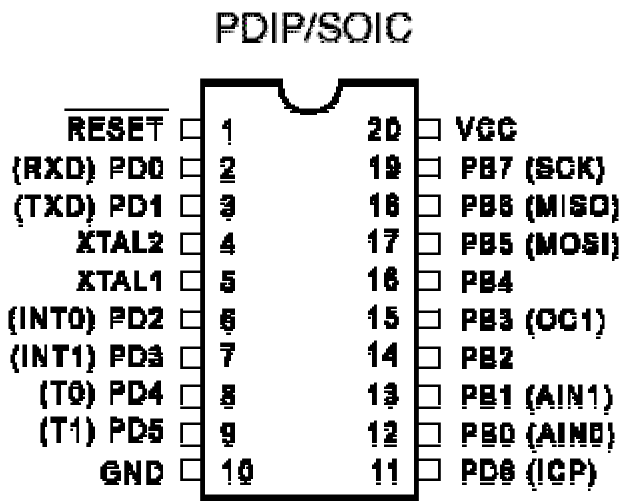
This is the default. When you made your own dongle, you can increase the baud rate

You need to save the file and restart BASCOM before the settings will be in effect.

AT90S2313

This page is intended to show user comments about the chip.

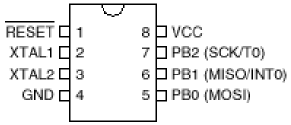
Your comment is welcome.



AT90S2323

This page is intended to show user comments about the chip.

Your comment is welcome.

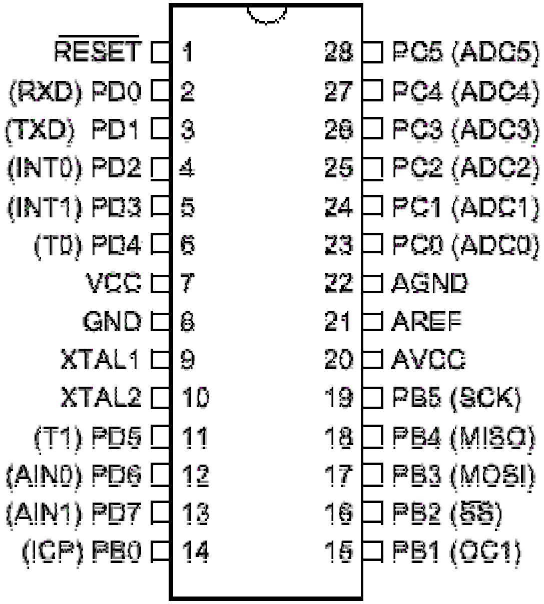


AT90S/LS2323

AT90S2333

This page is intended to show user comments about the chip.

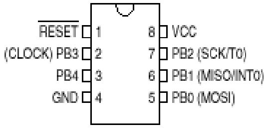
Your comment is welcome.



AT90S2343

This page is intended to show user comments about the chip.

Your comment is welcome.



AT90S/LS2343

[tip from Martin Verschuren]

When using the AT90S2343 with BASCOM-AVR 1.11.6.4 and the STK200. Programming must be done with jumper ext-clk.

The BASCOM build in programmer will detect a Tiny22, which seems to have the same ID string as the 2343 (Atmel source) so no wonder.

By using the internal clock RCEN=0, then the jumper of the STK200 must be on int.clk after programming.

Don't leave this away, some AT90S2343 will not correctly startup.

In your own project notice that you have to pullup the clk pin(2) at power up else it won't work. (I just looked for it for a day to get this problem solved:-)

Note : the at90s2343 and tiny22 have the same chip ID. In BASCOM you need to choose the tiny22 even if you use the 2343.

I note from MCS : only the AT23LS43-1 has the internal oscillator programmed by default! All other 2343 chips need an external clock signal. Tip: use a AT90S2313 and connect X2 to the clock input of the 2343.

[tip from David Chambers]
Using the AT90S2343 with BASCOM 1.11.7.3 the DT006 hardware there are no problems with programming the chip ie no special jumper conditions to enable programming. However it is best to remove links connecting ports to the DT006 LED's before programming. If access to PB3 and PB4 is desired then jumpers J11 & J12 must be installed with pins 2 and 3 linked in both cases. Note that PB3 and PB4 are each connected to a momentary pushbutton on the DT006 board. These can be used to check contact closure functions, so bear this in mind when writing code for contact monitoring.

The current ATMEL data sheet specifies that all versions –1, -4 and –10 are supplied with a fuse bit set for the internal clock that operates at approximately 1Mhz. If using the internal clock make sure to enter 1000000 under Options\Complier\Communication\frequency.

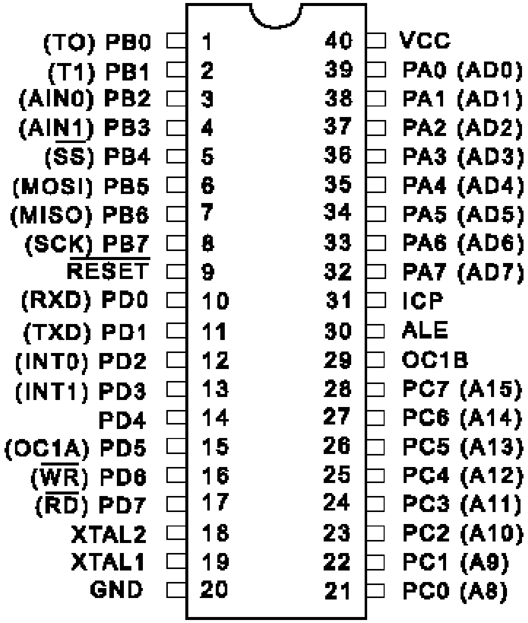
A great little chip with minimal external components. Only the resistor and capacitor required for RESET during power up.

Note that the LED's on the DT006 are not connected to the same programmed port pins when changing the chip type. This is because the special functions assigned ports varies between the 8pin, 20 pin and 28 pin products eg the MOSI, MISI and SCK functions are assigned to PB0, PB1 and PB2 for an 8 pin processor and PB5, PB6 and PB7 for a 20 pin processor. The result is that for a given program the LED's that respond are different.

AT90S4414

This page is intended to show user comments about the chip.

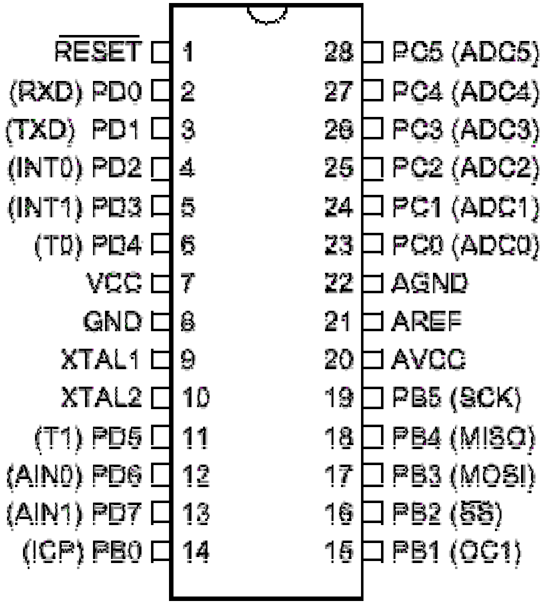
Your comment is welcome.



AT90S4433

This page is intended to show user comments about the chip.

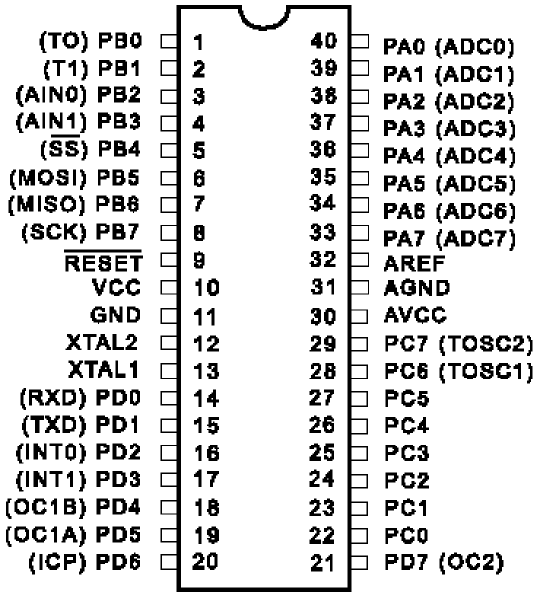
Your comment is welcome.



AT90S4434

This page is intended to show user comments about the chip.

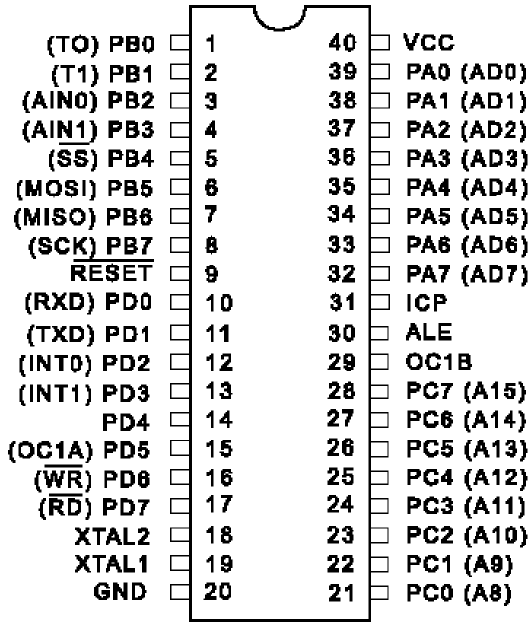
Your comment is welcome.



AT90S8515

This page is intended to show user comments about the chip.

Your comment is welcome.



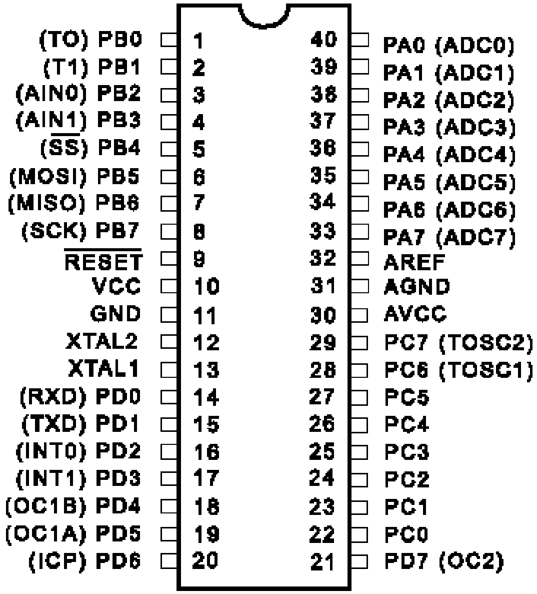
MEGA8515

MEGA8535

AT90S8535

This page is intended to show user comments about the chip.

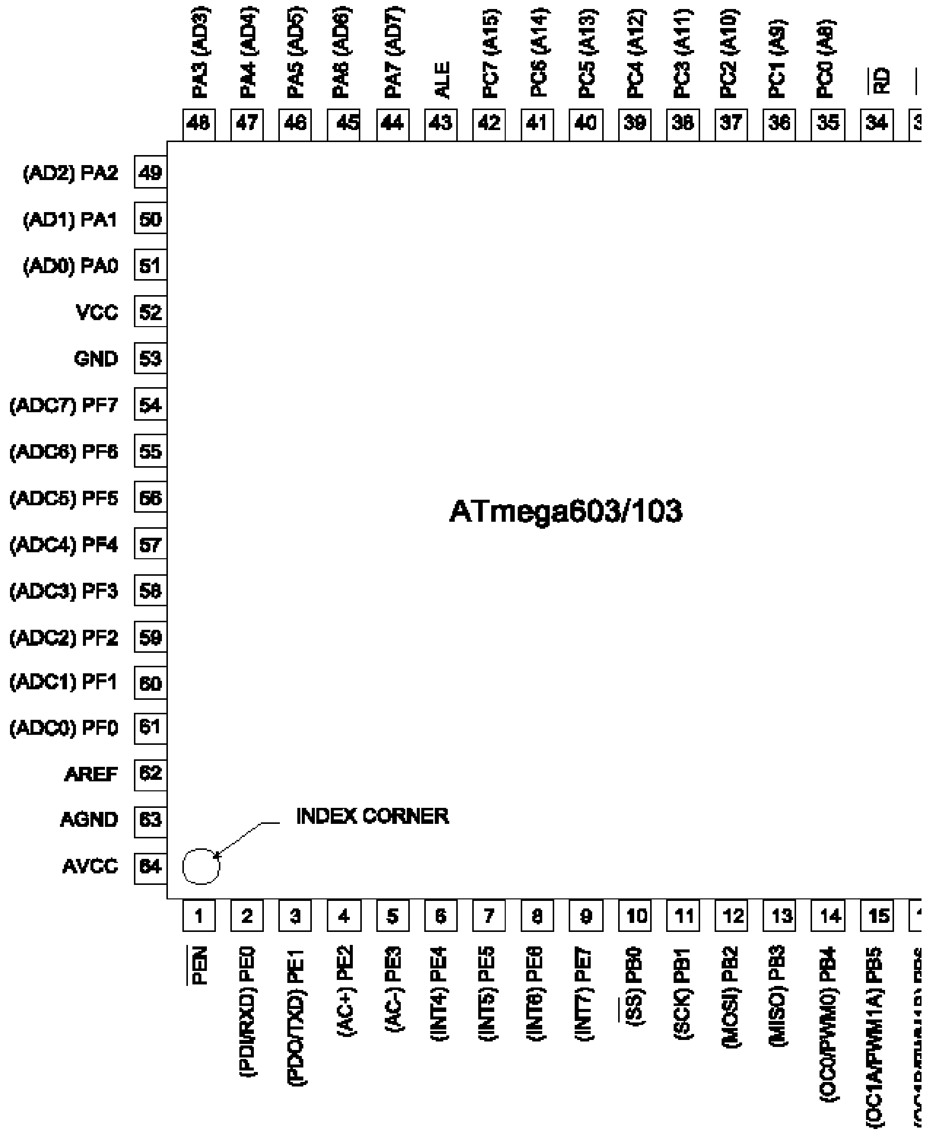
Your comment is welcome.



MEGA603

This page is intended to show user comments about the chip.

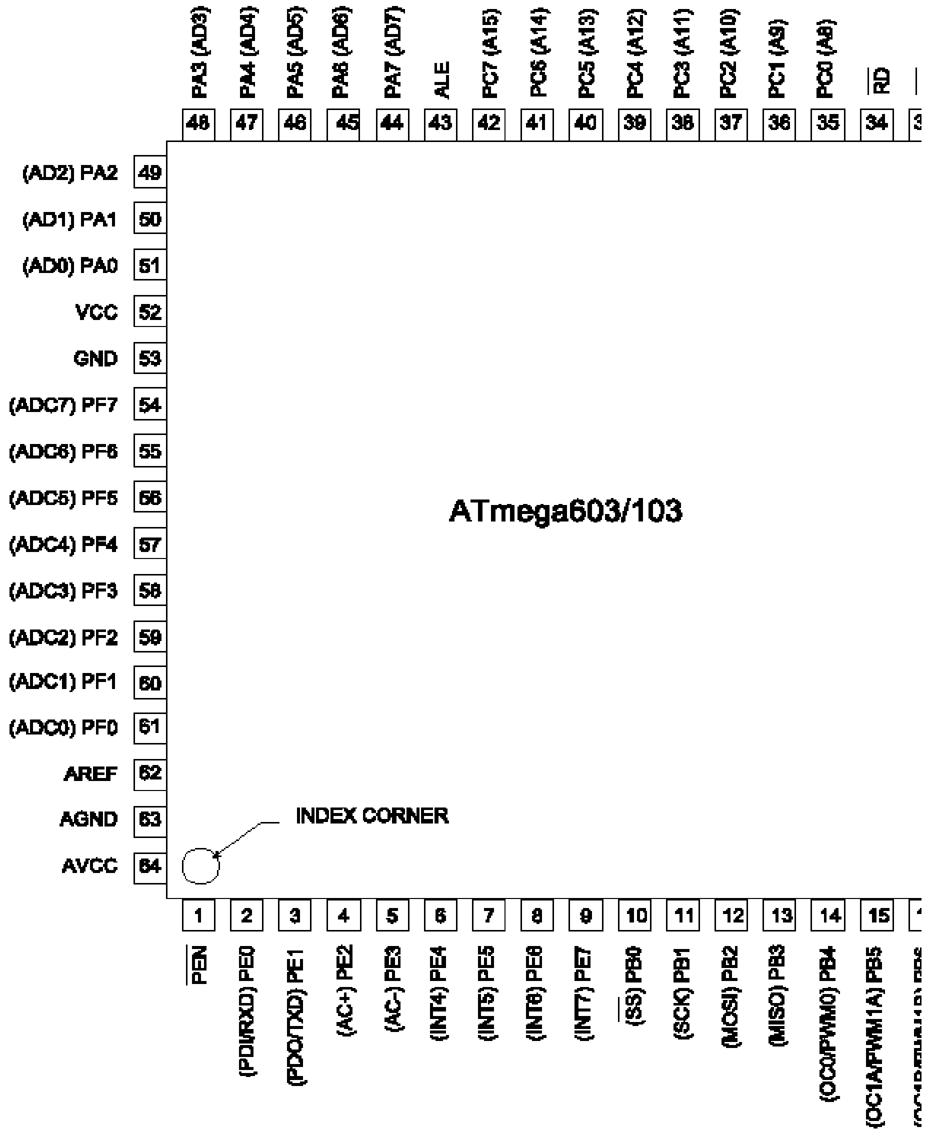
Your comment is welcome.



MEGA103

This page is intended to show user comments about the chip.

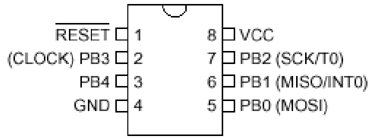
Your comment is welcome.



ATtiny22

This page is intended to show user comments about the chip.

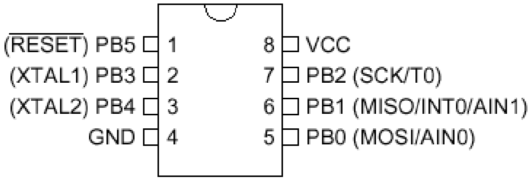
Your comment is welcome.



ATtiny12

This page is intended to show user comments about the chip.

Your comment is welcome.

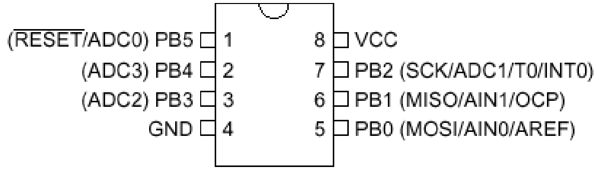


ATtiny15

This page is intended to show user comments about the chip.

Your comment is welcome.

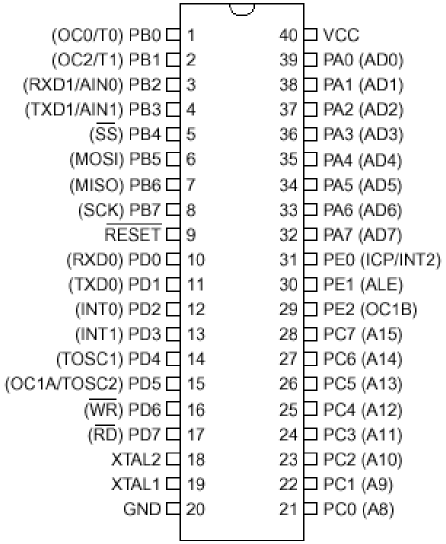
PDIP/SOIC



M161

This page is intended to show user comments about the chip.

Your comment is welcome.



M162

This page is intended to show user comments about the chip.

Your comment is welcome.

The M162 has a clock-16 divider enabled by default. See the M162.bas sample file

M163

The M163 by default uses the internal clock running at 1 MHz

When you have problems with timing set the right fuse bit A987= 0101. This will solve this problem.

I have just found a small difference in PortB when using the Mega163 in place of a 8535. The difference is in regard to PortB.4 - PortB.7 when not used as a SPI

interface. The four upper bits of PortB are shared with the hardware SPI unit.

If the SPI is configured in SLAVE mode (DEFAULT) the MOSI , SCK , /SS

Are configured as inputs, Regardless of the DDRB setting !

The /SS (slave select) pin also has restrictions on it when using it as a general input.- see data sheet ATmega163 - p57.

This sample allows you to use the upper nibble of PortB as outputs.

Portb = &B0000_0000

DDRB = &B1111_0000 'set upper bits for output.

Spcr = &B0001_0000 ' set SPI to Master and Disable.

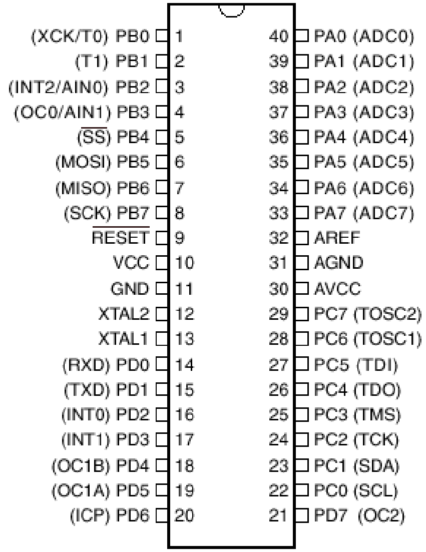
If The SPCR register is not set for Master, you cannot set the pins for

Output.

M323

This page is intended to show user comments about the chip.

Your comment is welcome.

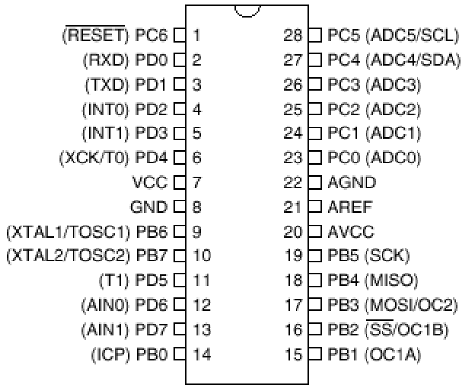


The JTAG interface is enabled by default. This means that portC.2-portC.5 pins can not be used. Program the JTAG fuse bit to disable the JTAG interface.

M8

This page is intended to show user comments about the chip.

Your comment is welcome.



M16

This page is intended to show user comments about the chip.

Your comment is welcome.

M64

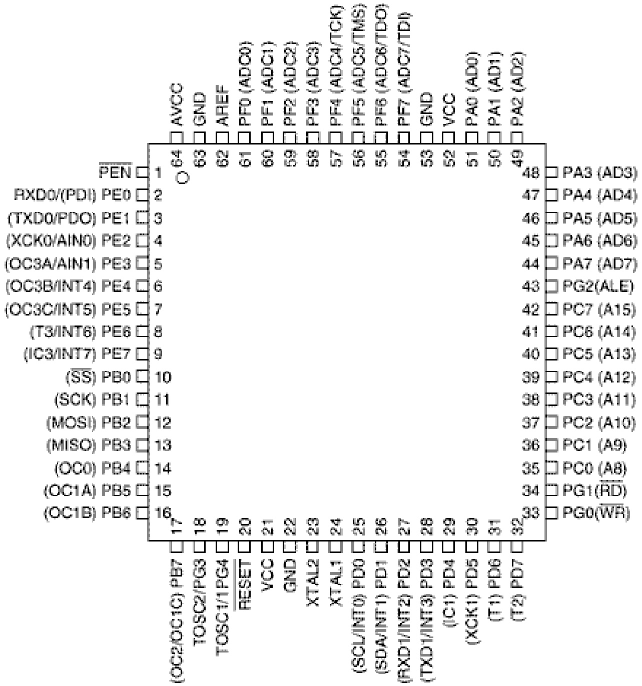
This page is intended to show user comments about the chip.

Your comment is welcome.

M128

This page is intended to show user comments about the chip.

Your comment is welcome.



AT90S1200

This chip is not really supported because it does not have internal memory. You can write your program using asm and program the chip via bascom.

AT86RF401

This page is intended to show user comments about the chip.

Your comment is welcome.

Changes compared to BASCOM-8051

The design goal was to make BASCOM-AVR compatible with BASCOM-8051.

For the AVR compiler I had to remove some statements.

New statements are also added. And some statements were changed.

They need specific attention, but the changes to the syntax will be made available to BASCOM-8051 too in the future.

Statements that were removed

STATEMENT	DESCRIPTION
$LARGE	Not needed anymore.
$ROMSTART	Code always starts at address 0 for the AVR. Added again in 1.11.6.2
$LCDHEX	Use LCD Hex(var) instead.
$NOINIT	Not needed anymore. Added in 1.11.6.2
$NOSP	Not needed anymore
$NOBREAK	Can't be used anymore because there is no object code that can be used for it.
$OBJ	Removed.
BREAK	Can't be used anymore because there is no object code that can be used for it.
PRIORITY	AVR does no allow setting priority of interrupts
PRINTHEX	You can use Print Hex(var) now
LCDHEX	You can use Lcd Hex(var) now

Statements that were added

STATEMENT	DESCRIPTION
FUNCTION	You can define your own user FUNCTIONS.
LOCAL	You can have LOCAL variables in SUB routines or FUNCTIONS.
^	New math statement. Var = 2 ^ 3 will return 2*2*2
SHIFT	Because ROTATE was changed, I added the SHIFT statement. SHIFT works just like ROTATE, but when shifted left, the LS BIT is cleared and the carry doesn't go to the LS BIT.
LTRIM	LTRIM, trims the leftmost spaces of a string.
RTRIM	RTRIM, trims the rightmost spaces of a string.
TRIM	TRIM, trims both the leftmost and rightmost spaces of a string.

Statements that behave differently

STATEMENT	DESCRIPTION
ROTATE	Rotate now behaves like the ASM rotate, this means that the carry will go to the most significant bit of a variable or the least significant bit of a variable.
CONST	String were added to the CONST statement. I also changed it to be compatible with QB.
DECLARE	BYVAL has been added since real subprograms are now supported.
DIM	You can now specify the location in memory of the variable. Dim v as byte AT 100, will use memory location 100.

Language Fundamentals

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

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

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

????The specific characters in the character set and the special meanings of some characters.

????The format of a line in a BASCOM program.

????Line labels.

????Program line length.

Character Set

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

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

The BASCOM numeric characters are the digits 0-9.

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

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

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

The BASCOM program line

BASCOM program lines have the following syntax:

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

Using Line Identifiers

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

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

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. When there is a label on the line, no other identifiers may be used on the same line. So the label is the sole identifier on a line.

BASCOM Statements

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

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

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

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

? REM or (starts a comment)

? DIM

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

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

The following lines are equivalent:

PRINT " Quantity remaining" : REM Print report label.

PRINT " Quantity remaining" ' Print report label.

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

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

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). A bit can hold only the value 0 or 1.

A group of 8 bits is called a 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 -2147483648 to 2147483647.

????Single.
Singles are stored as signed 32 bit binary numbers. Ranging in value from
1.5 x 10^–45 to 3.4 x 10^38

????String (up to 254 bytes).
Strings are stored as bytes and are terminated with a 0-byte.
A string dimensioned with a length of 10 bytes will occupy 11 bytes.

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

Variables

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

A numeric variable, can be assigned only a numeric value (either integer, byte, long, single or bit).

The following list shows some examples of variable assignments:

????A constant value:
A = 5
C = 1.1

????The value of another numeric variable:
abc = def
k = g

????The value obtained by combining other variables, constants, and operators: Temp = a + 5
Temp = C + 5

????The value obtained by calling a function:
Temp = Asc(S)

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 , a = **&B**1010 and a = 10 are all the same.

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

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

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

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

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

Expressions and Operators

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

Anytime you do a calculation you are using expressions and operators.

This chapter describes how expressions are formed and concludes by describing the following kind of operators:

? Arithmetic operators, used to perform calculations.

? Relational operators, used to compare numeric or string values.

? Logical operators, used to test conditions or manipulate individual bits.

? Functional operators, used to supplement simple operators.

Expressions and Operators

An expression can be a numeric constant, a variable, or a single value

obtained by combining constants, variables, and other expressions with operators.

Operators perform mathematical or logical operations on values.

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

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

Arithmetic

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

? Integer

Integer division is denoted by the backslash (\).
Example: Z = X\ 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 the moment no message is produced, so you have to make sure yourself that this won't happen.

Relational Operators

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

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

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

Logical Operators

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

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

It is possible to use logical operators to test bytes for a particular bit pattern.
For example the AND operator can be used to mask all but one of the bits of a status byte, while OR can be used to merge two bytes to create a particular binary value.

Example

```
A = 63 And 19
PRINT A
A = 10 Or 9
PRINT A
```

Output
19
11

Floating point (ASM code used is supplied by Jack Tidwell)

Single numbers conforming to the IEEE binary floating point standard.
An eight bit exponent and 24 bit mantissa are supported.
Using four bytes the format is shown below:

```
31 30________23 22______________________________0
s exponent mantissa
```

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

All mathematical operations are supported by the single.
You can also convert a single to an integer or word or vise versa:

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

Here is a fragment from the Microsoft knowledge base about FP:

Floating-point mathematics is a complex topic that confuses many programmers. The tutorial below should help you recognize programming situations where floating-point errors are likely to occur and how to avoid them. It should also allow you to recognize cases that are caused by inherent floating-point math limitations as opposed to actual compiler bugs.

Decimal and Binary Number Systems

Normally, we count things in base 10. The base is completely arbitrary. The only reason that people have traditionally used base 10 is that they have 10 fingers, which have made handy counting tools.

The number 532.25 in decimal (base 10) means the following:

(5 * 10^2) + (3 * 10^1) + (2 * 10^0) + (2 * 10^1) + (5 * 10^-2)
500 + 30 + 2 + 2/10 + 5/100
____
= 532.25

In the binary number system (base 2), each column represents a power of 2 instead of 10. For example, the number 101.01 means the following:

(1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^1) + (1 * 2^-2)
4 + 0 + 1 + 0 + 1/4
____
= 5.25 Decimal

How Integers Are Represented in PCs

Because there is no fractional part to an integer, its machine representation is much simpler than it is for floating-point values. Normal integers on personal computers (PCs) are 2 bytes (16 bits) long with the most significant bit indicating the sign. Long integers are 4 bytes long. Positive values are straightforward binary numbers. For example:

1 Decimal = 1 Binary

2 Decimal = 10 Binary

22 Decimal = 10110 Binary, etc.

However, negative integers are represented using the two's complement scheme. To get the two's complement representation for a negative number, take the binary representation for the number's absolute value and then flip all the bits and add 1. For example:

4 Decimal = 0000 0000 0000 0100

1111 1111 1111 1011 Flip the Bits

-4 = 1111 1111 1111 1100 Add 1

Note that adding any combination of two's complement numbers together using ordinary binary arithmetic produces the correct result.

Floating-Point Complications

Every decimal integer can be exactly represented by a binary integer; however, this is not true for fractional numbers. In fact, every number that is irrational in base 10 will also be irrational in any system with a base smaller than 10.

For binary, in particular, only fractional numbers that can be represented in the form p/q, where q is an integer power of 2, can be expressed exactly, with a finite number of bits.

Even common decimal fractions, such as decimal 0.0001, cannot be represented exactly in binary. (0.0001 is a repeating binary fraction with a period of 104 bits!)

This explains why a simple example, such as the following

```
SUM = 0
FOR I% = 1 TO 10000
SUM = SUM + 0.0001
NEXT I%
PRINT SUM ' Theoretically = 1.0.
```

will PRINT 1.000054 as output. The small error in representing 0.0001 in binary propagates to the sum.

For the same reason, you should always be very cautious when making comparisons on real numbers. The following example illustrates a common programming error:

```
item1# = 69.82#
item2# = 69.20# + 0.62#
IF item1# = item2# then print "Equality!"
```

This will NOT PRINT "Equality!" because 69.82 cannot be represented exactly in binary, which causes the value that results from the assignment to be SLIGHTLY different (in binary) than the value that is generated from the expression. In practice, you should always code such comparisons in such a way as to allow for some tolerance.

General Floating-Point Concepts

It is very important to realize that any binary floating-point system can represent only a finite number offloating-point values in exact form. All other values must be approximated by the closest representable value. The IEEE standard specifies the method for rounding values to the "closest" representable value. BASCOM supports the standard and rounds according to the IEEE rules.

Also, keep in mind that the numbers that can be represented in IEEE are spread out over a very wide range. You can imagine them on a number line. There is a high density of representable numbers near 1.0 and -1.0 but fewer and fewer as you go towards 0 or infinity.

The goal of the IEEE standard, which is designed for engineering calculations, is to maximize accuracy (to get as close as possible to the actual number). Precision refers to the number of digits that you can represent. The IEEE standard attempts to balance the number of bits dedicated to the exponent with the number of bits used for the fractional part of the number, to keep both accuracy and precision within acceptable limits.

IEEE Details

Floating-point numbers are represented in the following form, where [exponent] is the binary exponent:

X = Fraction * 2^(exponent - bias)

[Fraction] is the normalized fractional part of the number, normalized because the exponent is adjusted so that the leading bit is always a 1. This way, it does not have to be stored, and you get one more bit of precision. This is why there is an implied bit. You can think of this like scientific notation, where you manipulate the exponent to have one digit to the left of the decimal point, except in binary, you can always manipulate the exponent so that the first bit is a 1, since there are only 1s and 0s.

[bias] is the bias value used to avoid having to store negative exponents.

The bias for single-precision numbers is 127 and 1023 (decimal) for double-precision numbers.

The values equal to all 0's and all 1's (binary) are reserved for representing special cases. There are other special cases as well, that indicate various error conditions.

Single-Precision Examples

2 = 1 * 2^1 = 0100 0000 0000 0000 ... 0000 0000 = 4000 0000 hex

Note the sign bit is zero, and the stored exponent is 128, or

100 0000 0 in binary, which is 127 plus 1. The stored mantissa is (1.) 000 0000 ... 0000 0000, which has an implied leading 1 and

binary point, so the actual mantissa is 1.

-2 = -1 * 2^1 = 1100 0000 0000 0000 ... 0000 0000 = C000 0000 hex
Same as +2 except that the sign bit is set. This is true for all
IEEE format floating-point numbers.

4 = 1 * 2^2 = 0100 0000 1000 0000 ... 0000 0000 = 4080 0000 hex
Same mantissa, exponent increases by one (biased value is 129, or
100 0000 1 in binary.

6 = 1.5 * 2^2 = 0100 0000 1100 0000 ... 0000 0000 = 40C0 0000 hex

Same exponent, mantissa is larger by half -- it's
(1.) 100 0000 ... 0000 0000, which, since this is a binary
fraction, is 1-1/2 (the values of the fractional digits are 1/2,
1/4, 1/8, etc.).

1 = 1 * 2^0 = 0011 1111 1000 0000 ... 0000 0000 = 3F80 0000 hex
Same exponent as other powers of 2, mantissa is one less than
2 at 127, or 011 1111 1 in binary.

.75 = 1.5 * 2^1 = 0011 1111 0100 0000 ... 0000 0000 = 3F40 0000 hex
The biased exponent is 126, 011 1111 0 in binary, and the mantissa

is (1.) 100 0000 ... 0000 0000, which is 1-1/2.

2.5 = 1.25 * 2^1 = 0100 0000 0010 0000 ... 0000 0000 = 4020 0000 hex
Exactly the same as 2 except that the bit which represents 1/4 is
set in the mantissa.

0.1 = 1.6 * 2^4 = 0011 1101 1100 1100 ... 1100 1101 = 3DCC CCCD hex
1/10 is a repeating fraction in binary. The mantissa is just shy of 1.6, and the biased exponent says that 1.6 is to be divided by 16 (it is 011 1101 1 in binary, which is 123 n decimal). The true exponent is 123 - 127 = -4, which means that the factor by which to multiply is 2**-4 = 1/16. Note that the stored mantissa is rounded up in the last bit. This is an attempt to represent the unrepresentable number as accurately as possible. (The reason that 1/10 and 1/100 are not exactly representable in binary is similar to the way that 1/3 is not exactly representable in decimal.)

0 = 1.0 * 2^-128 = all zeros -- a special case.

Other Common Floating-Point Errors
The following are common floating-point errors:
1. Round-offerror
This error results when all of the bits in a binary number cannot
be used in a calculation.

Example: Adding 0.0001 to 0.9900 (Single Precision)

Decimal 0.0001 will be represented as:

(1.)10100011011011100010111 * 2^(-14+Bias) (13 Leading 0s in
Binary!)

0.9900 will be represented as:

(1.)11111010111000010100011 * 2^(-1+Bias)

Now to actually add these numbers, the decimal (binary) points must be aligned. For this they must be Unnormalized. Here is the resulting addition:

.00000000000000011010001101 * 2^0 <- Only 11 of 23 Bits retained
+.111111010111000010100011 * 2^0
____
.111111010111011100110000 * 2^0

This is called a round-off error because some computers round when shifting for addition. Others simply truncate. Round-off errors are important to consider whenever you are adding or multiplying two very different values.

2. Subtracting two almost equal values

.1235
-.1234
_____
.0001

This will be normalized. Note that although the original numbers each had four significant digits, the result has only one significant digit.

3. Overflow and underflow
This occurs when the result is too large or too small to be represented by the data type.

4. Quantizing error
This occurs with those numbers that cannot be represented in exact form by the floating-point standard.

Rounding

When a Long is assigned to a single, the number is rounded according to the rules of the IEEE committee.

For explanation: 1.500000 is exact the middle between 1.00000 and 2.000000. If x.500000 is always rounded up, than there is trend for higher values than the average of all numbers. So their rule says, half time to round up and half time to round down, if value behind LSB is exact ..500000000.

The rule is, round this .500000000000 to next even number, that means if LSB is 1 (half time) to round up, so the LSB is going to 0 (=even), if LSB is 0 (other half time) to round down, that means no rounding.

This rounding method is best since the abnsolute error is 0.

You can override the default IEEE rounding method by specifying the $LONG2FLOAT.LBX library which rounds up to the next number. This is the method used up to 1.11.7.4 of the compiler.

The following table shows the difference in rounding methods. While the result seem strange at first, the IEEE rounding gives the best result .

IEEE 754 Rounding			Normal Rounding			IEEE 754 Rounding			Normal Roun	
LONG	Single	0	Long	Single	0,5	LONG	Single	0	Long	Sing
16777217	16777216	-1	16777217	16777218	1	33554433	33554432	-1	33554433	33554
16777218	16777218	0	16777218	16777218	0	33554434	33554432	-2	33554434	33554
16777219	16777220	1	16777219	16777220	1	33554435	33554436	1	33554435	33554
16777220	16777220	0	16777220	16777220	0	33554436	33554436	0	33554436	33554
16777221	16777220	-1	16777221	16777222	1	33554437	33554436	-1	33554437	33554
16777222	16777222	0	16777222	16777222	0	33554438	33554440	2	33554438	33554
16777223	16777224	1	16777223	16777224	1	33554439	33554440	1	33554439	33554
16777224	16777224	0	16777224	16777224	0	33554440	33554440	0	33554440	33554
16777225	16777224	-1	16777225	16777226	1	33554441	33554440	-1	33554441	33554
16777226	16777226	0	16777226	16777226	0	33554442	33554440	-2	33554442	33554
16777227	16777228	1	16777227	16777228	1	33554443	33554444	1	33554443	33554
16777228	16777228	0	16777228	16777228	0	33554444	33554444	0	33554444	33554
16777229	16777228	-1	16777229	16777230	1	33554445	33554444	-1	33554445	33554
16777230	16777230	0	16777230	16777230	0	33554446	33554448	2	33554446	33554
16777231	16777232	1	16777231	16777232	1	33554447	33554448	1	33554447	33554
16777232	16777232	0	16777232	16777232	0	33554448	33554448	0	33554448	33554
16777233	16777232	-1	16777233	16777234	1	33554449	33554448	-1	33554449	33554
16777234	16777234	0	16777234	16777234	0	33554450	33554448	-2	33554450	33554
16777235	16777236	1	16777235	16777236	1	33554451	33554452	1	33554451	33554
16777236	16777236	0	16777236	16777236	0	33554452	33554452	0	33554452	33554
16777237	16777236	-1	16777237	16777238	1	33554453	33554452	-1	33554453	33554
16777238	16777238	0	16777238	16777238	0	33554454	33554456	2	33554454	33554
16777239	16777240	1	16777239	16777240	1	33554455	33554456	1	33554455	33554
16777240	16777240	0	16777240	16777240	0	33554456	33554456	0	33554456	33554
16777241	16777240	-1	16777241	16777242	1	33554457	33554456	-1	33554457	33554
16777242	16777242	0	16777242	16777242	0	33554458	33554456	-2	33554458	33554
16777243	16777244	1	16777243	16777244	1	33554459	33554460	1	33554459	33554

Arrays

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

The index must be a numeric constant, a byte, an integer, word or long.

The maximum number of elements is 65535.

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

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

Example:

```
'create an array named a, with 10 elements (1 to 10)
Dim A(10) As Byte
'create an integer
Dim C As Integer
'now fill the array
For C = 1 To 10
'assign array element
A(c) = C
' print it
Print A(c)
Next
'you can add an offset to the index too
C = 0
A(c + 1) = 100
Print A(c + 1)
End
```

Strings

A string is used to store text. A string must be dimensioned with the length specified.

DIM S as STRING * 5

Will create a string that can store a text with a maximum length of 5 bytes.

The space used is 6 bytes because a string is terminated with a null byte.

To assign the string:

s = "abcd"

To insert special characters into the string :

s= "AB{027}cd"

The {ascii} will insert the ASCII value into the string.

The number of digits must be 3. s = "{27} will assign "{27}" to the string instead of escape character 27!

Casting

In BASCOM-AVR when you perform operations on variables they all must be of the same data type.

long = long1 * long2 ' for example

The assigned variables data type determines what kind of math is performed.

For example when you assign a long, long math will be used.

If you try to store the result of a LONG into a byte, only the LSB of the LONG will be stored into the BYTE.

Byte = LONG

When LONG = 256 , it will not fit into a BYTE. The result will be 256 AND 255 = 0.

Of course you are free to use different data types. The correct result is only guaranteed when you are using data types of the same kind or that that result always can fit into the target data type.

When you use strings, the same rules apply. But there is one exception:

Dim b as Byte

b = 123 ' ok this is normal

b = "A" ' b = 65

When the target is a byte and the source variable is a string constant denoted by "", the ASCII value will be stored in the byte. This works also for tests :

IF b = "A" then ' when b = 65

END IF

This is different compared to QB/VB where you can not assign a string to a byte variable.

SINGLE CONVERSION

When you want to convert a SINGLE into a byte, word, integer or long the compiler will automatic convert the values when the source string is of the SINGLE data type.

integer = single

You can also convert a byte, word, integer or long into a SINGLE by assigning this variable to a SINGLE.

single = long

Mixing ASM and BASIC

BASCOM allows you to mix BASIC with assembly.

This can be very useful in some situations when you need full control of the generated code.

Almost all assembly mnemonics are recognized by the compiler. The exceptions are: SUB, SWAP,CALL and OUT. These are BASIC reserved words and have priority over the ASM mnemonics. To use these mnemonics precede them with the **!** - sign.

For example :

```
Dim a As Byte At &H60 'A is stored at location &H60
Ldi R27 , $00   'Load R27 with MSB of address
Ldi R26 , $60   'Load R26 with LSB of address
Ld R1, X  'load memory location $60 into R1
!SWAP R1   'swap nibbles
```

As you can see the SWAP mnemonic is preceded by a ! sign.

Another option is to use the assembler block directives:

```
$ASM
 Ldi R27 , $00   'Load R27 with MSB of address
 Ldi R26 , $60   'Load R26 with LSB of address
 Ld R1, X    'load memory location $60 into R1
 SWAP R1   'swap nibbles
$END ASM
```

A special assembler helper function is provided to load the address into the register X or Z. Y can may not be used because it is used as the soft stack pointer.

```
Dim A As Byte 'reserve space
LOADADR a, X 'load address of variable named A into register pair X
```

This has the same effect as :

```
Ldi R26 , $60 'for example !
Ldi R27, $00 'for example !
```

Some registers are used by BASCOM

R4 and R5 are used to point to the stack frame or the temp data storage

R6 is used to store some bit variables:

R6 bit 0 = flag for integer/word conversion

R6 bit 1 = temp bit space used for swapping bits

R6 bit 2 = error bit (ERR variable)

R6 bit 3 = show/noshow flag when using INPUT statement

R8 and R9 are used as a data pointer for the READ statement.

All other registers are used depending on the used statements.

To Load the address of a variable you must enclose them in brackets.

Dim B As Bit
Lds R16, {B} 'will replace {B} with the address of variable B

To refer to the bitnumber you must precede the variable name by BIT.
Sbrs R16 , **BIT.** B 'notice the point!
Since this was the first dimensioned bit the bit number is 7. Bits are stored in bytes and the first dimensioned bit goes in the LS bit.

To load an address of a label you must use :
LDI ZL, Low(lbl * 1)
LDI ZH , High(lbl * 1)
Where ZL = R30 and may be R24, R26, R28 or R30
And ZH = R31 and may be R25, R27, R29 or R31.
These are so called register pairs that form a pointer.

When you want to use the LPM instruction to retrieve data you must multiply the address with 2 since the AVR object code consist of words.
LDI ZL, Low(lbl * 2)
LDI ZH , High(lbl * 2)
LPM ; get data into R0
Lbl:

Atmel mnemonics must be used to program in assembly.
You can download the pdf from www.atmel.com that shows how the different mnemonics are used.
Some points of attention :
* All instructions that use a constant as a parameter only work on the upper 16 registers (r16 -r31)
So LDI R15,12 WILL NOT WORK

* The instruction SBR register, K
will work with K from 0-255. So you can set multiple bits!

The instruction SBI port, K will work with K from 0-7 and will set only ONE bit in a IO-port register.

The same applies to the CBR and CBI instructions.

You can use constants too:
.equ myval = (10+2)/4
ldi r24,myval+2 '5
ldi r24,asc("A")+1 ; load with 66

Or in BASIC with CONST :

CONST Myval = (10+2) / 4
Ldi r24,myval

How to make your own libraries and call them from BASIC?
The files for this sample can be found as libdemo.bas in the SAMPLES dir and as mylib.lib in the LIB dir.

First determine the used parameters and their type.
Also consider if they are passed by reference or by value

For example the sub test has two parameters:
x which is passed by value (copy of the variable)
y which is passed by reference(address of the variable)

In both cases the address of the variable is put on the soft stack which is indexed by the Y pointer.

The first parameter (or a copy) is put on the soft stack first
To refer to the address you must use:
ldd r26 , y + 0
ldd r27 , y + 1
This loads the address into pointer X

The second parameter will also be put on the soft stack so :
The reference for the x variable will be changed :

To refer to the address of **x** you must use:
ldd r26 , y + 2
ldd r27 , y + 3

To refer to the last parameter **y** you must use
ldd r26 , y + 0
ldd r27 , y + 1

Write the sub routine as you are used too but include the name within brackets []

[test]
test:
ldd r26,y+2 ; load address of x
ldd r27,y+3
ld r24,x ; get value into r24
inc r24 ; value + 1
st x,r24 ; put back
ldd r26,y+0 ; address of y
ldd r27,y+1
st x,r24 ; store
ret ; ready
[end]

To write a function goes the same way.
A function returns a result so a function has one additional parameter.

It is generated automatic and it has the name of the function.
This way you can assign the result to the function name
For example:

Declare Function Test(byval x as byte , y as byte) as byte
A virtual variable will be created with the name of the function in this case **test**
It will be pushed on the softstack with the Y-pointer.
To reference to the result or name of the function (test) the address will be:
y + 0 and y + 1
The first variable **x** will bring that to y + 2 and y + 3
And the third variable will cause that 3 parameters are saved on the soft stack
To reference to test you must use :
ldd r26 , y + 4
ldd r27 , y + 5

To reference to x
ldd r26 , y + 2
ldd r27 , y + 3
And to reference y
ldd r26 , y + 0
ldd r27 , y + 1

When you use exit sub or exit function you also need to provide an additional label. It starts with **sub_** and must be completed with the function / sub routine name. In our example:
sub_test:

When you use local variables thing become more complicated.
Each local variable address will be put on the soft stack too
When you use 1 local variable its address will become
ldd r26, y+0
ldd r27 , y + 1
All other parameters must be increased with 2 so the reference to y variable changes from
ldd r26 , y + 0 to ldd r26 , y + 2
ldd r27 , y + 1 to ldd r27 , y + 3
And of course also for the other variables.

When you have more local variables just add 2 for each.
Finally you save the file as a **.lib** file
Use the library manager to compile it into the lbx format.
The declare sub / function must be in the program where you use the sub / function.

The following is a copy of the libdemo.bas file :

```
'define the used library
$lib "mylib.lib"
'also define the used routines
$external Test

'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X As Byte , Y As Byte)


'reserve some space
Dim Z As Byte


'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example
End
```

When you use ports in your library you must use **.equ** to specify the address:
.equ EEDR=$1d
In R24, **EEDR**

This way the library manager know the address of the port during compile time.

As an alternative precede the mnemonic with a * so the code will not be compiled into the lib. The address of the register will be resolved at un time in that case.

This chapter is not intended to learn you ASM programming. But when you find a topic is missing to interface BASCOM with ASM send me an email.

Assemblermnemonics

BASCOM supports the mnemonics as defined by Atmel.

The Assembler accepts mnemonic instructions from the instruction set.

A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Mnemonics	Operands	Description	Operation	Flags	Clock
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H	1
SUB	Rd, Rr	Subtract without Carry	Rd = Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	Rd = Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd = Rd - K - C	Z,C,N,V,H	1
AND	Rd, Rr	Logical AND	Rd = Rd · Rr	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	Rd = Rd · K	Z,N,V	1
OR	Rd, Rr	Logical OR	Rd = Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	Rd = Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR	Rd = Rd Å Rr	Z,N,V	1
COM	Rd	Ones Complement	Rd = $FF - Rd	Z,C,N,V	1
NEG	Rd	Twos Complement	Rd = $00 - Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd = Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd = Rd · ($FFh - K)	Z,N,V	1
INC	Rd	Increment	Rd = Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd = Rd - 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd = Rd · Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd = Rd Å Rd	Z,N,V	1
SER	Rd	Set Register	Rd = $FF	None	1
ADIW Adiw r24, K6	Rdl, K6	Add Immediate to Word	Rdh:Rdl = Rdh:Rdl + K	Z,C,N,V,S	2
SBIW Sbiw R24,K6	Rdl, K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1, R0 = Rd * Rr	C	2 *
BRANCH INSTRUCTIONS					
RJMP	K	Relative Jump	PC = PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC = Z	None	2
JMP	K	Jump	PC = k	None	3
RCALL	K	Relative Call Subroutine	PC = PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC = Z	None	3
CALL	K	Call Subroutine	PC = k	None	4
RET		Subroutine Return	PC = STACK	None	4
RETI		Interrupt Return	PC = STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC = PC + 2 or 3	None	1 / 2
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H,	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	If (Rr(b)=0) PC = PC + 2 or 3	None	1 / 2
SBRS	Rr, b	Skip if Bit in Register Set	If (Rr(b)=1) PC = PC + 2 or 3	None	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	If(I/O(P,b)=0) PC = PC + 2 or 3	None	2 / 3
SBIS	P, b	Skip if Bit in I/O Register Set	If(I/O(P,b)=1) PC = PC + 2 or 3	None	2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC=PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC=PC+k + 1	None	1 / 2

BREQ	K	Branch if Equal	if (Z = 1) then PC = PC + k + 1	None	1 / 2
BRNE	K	Branch if Not Equal	if (Z = 0) then PC = PC + k + 1	None	1 / 2
BRCS	K	Branch if Carry Set	if (C = 1) then PC = PC + k + 1	None	1 / 2
BRCC	K	Branch if Carry Cleared	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRSH	K	Branch if Same or Higher	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRLO	K	Branch if Lower	if (C = 1) then PC = PC + k + 1	None	1 / 2
BRMI	K	Branch if Minus	if (N = 1) then PC = PC + k + 1	None	1 / 2
BRPL	K	Branch if Plus	if (N = 0) then PC = PC + k + 1	None	1 / 2
BRGE	K	Branch if Greater or Equal, Signed	if (N V= 0) then PC = PC+ k + 1	None	1 / 2
BRLT	K	Branch if Less Than, Signed	if (N V= 1) then PC = PC + k + 1	None	1 / 2
BRHS	K	Branch if Half Carry Flag Set	if (H = 1) then PC = PC + k + 1	None	1 / 2
BRHC	K	Branch if Half Carry Flag Cleared	if (H = 0) then PC = PC + k + 1	None	1 / 2
BRTS	K	Branch if T Flag Set	if (T = 1) then PC = PC + k + 1	None	1 / 2
BRTC	K	Branch if T Flag Cleared	if (T = 0) then PC = PC + k + 1	None	1 / 2
BRVS	K	Branch if Overflow Flag is Set	if (V = 1) then PC = PC + k + 1	None	1 / 2
BRVC	K	Branch if Overflow Flag is Cleared	if (V = 0) then PC = PC + k + 1	None	1 / 2
BRIE	K	Branch if Interrupt Enabled	if (I = 1) then PC = PC + k + 1	None	1 / 2
BRID	K	Branch if Interrupt Disabled	if (I = 0) then PC = PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Copy Register	Rd = Rr	None	1
LDI	Rd, K	Load Immediate	Rd = K	None	1
LDS	Rd, k	Load Direct	Rd = (k)	None	2

LD	Rd, X	Load Indirect	Rd = (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd = (X), X = X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X = X- 1, Rd =(X)	None	2
LD	Rd, Y	Load Indirect	Rd = (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd = (Y), Y = Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y = Y - 1, Rd = (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd = (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd = (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd = (Z), Z = Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z = Z - 1, Rd = (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd = (Z + q)	None	2
STS	k, Rr	Store Direct	(k) = Rr	None	2
ST	X, Rr	Store Indirect	(X) = Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) = Rr, X = X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X = X- 1, (X) = Rr	None	2
ST	Y, Rr	Store Indirect	(Y) = Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) = Rr, Y = Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y = Y - 1, (Y) = Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) = Rr	None	2
ST	Z, Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) = Rr, Z = Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z = Z - 1, (Z) = Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) = Rr	None	2
LPM		Load Program Memory	R0 =(Z)	None	3

IN	Rd, P	In Port	Rd = P	None	1
OUT	P, Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push Register on Stack	STACK = Rr	None	2
POP	Rd	Pop Register from Stack	Rd = STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) =Rd(n),Rd(0)= 0,C=Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n) = Rd(n+1), Rd(7) =0, C=Rd(0)	Z,C,N,V	1
ROL	Rd	RotateLeft ThroughCarry	Rd(0) =C, Rd(n+1) =Rd(n),C=Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right ThroughCarry	Rd(7) =C,Rd(n) =Rd(n+1),C¬Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) = Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) « Rd(7..4)	None	1
BSET	S	Flag Set	SREG(s) = 1	SREG(s)	1
BCLR	S	Flag Clear	SREG(s) = 0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	I/O(P, b) = 1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) = 0	None	2
BST	Rr, b	Bit Store from Register to T	T = Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) = T	None	1
SEC		Set Carry	C = 1	C	1
CLC		Clear Carry	C = 0	C	1
SEN		Set Negative Flag	N = 1	N	1
CLN		Clear Negative Flag	N = 0	N	1
SEZ		Set Zero Flag	Z = 1	Z	1
CLZ		Clear Zero Flag	Z = 0	Z	1
SEI		Global Interrupt Enable	I = 1	I	1
CLI		Global Interrupt Disable	I = 0	I	1

SES		Set Signed Test Flag	S = 1	S	1
CLS		Clear Signed Test Flag	S = 0	S	1
SEV		Set Twos Complement Overflow	V = 1	V	1
CLV		Clear Twos Complement Overflow	V = 0	V	1
SET		Set T in SREG	T = 1	T	1
CLT		Clear T in SREG	T = 0	T	1
SHE		Set Half Carry Flag in SREG	H = 1	H	1
CLH		Clear Half Carry Flag in SREG	H = 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

*) Not available in base-line microcontrollers

The Assembler is not case sensitive.

The operands have the following forms:

Rd: R0-R31 or R16-R31 (depending on instruction)
Rr: R0-R31
b: Constant (0-7)
s: Constant (0-7)
P: Constant (0-31/63)
K: Constant (0-255)
k: Constant, value range depending on instruction.
q: Constant (0-63)
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions

Reserved Words

The following table shows the reserved BASCOM statements or characters.

^
!
;
$BAUD
$BAUD1
$BOOT
$CRYSTAL
$DATA
$DBG
$DEFAULT
$END
$EEPROM
$EXTERNAL
$INCLUDE
$LCD
$LCDRS
$LCDPUTCTRL
$LCDPUTDATA
$LCDVFO
$LIB
$MAP
$REGFILE
$SERIALINPUT
$SERIALINPUT1
$SERIALINPUT2LCD
$SERIALOUTPUT
$SERIALOUTPUT1
$TINY
$WAITSTATE
$XRAMSIZE
$XRAMSTART

1WRESET
1WREAD
1WWRITE

ACK
ABS()
ALIAS
AND
ACOS
AS
ASC()
ASIN
AT
ATN
ATN2

BAUD
BCD()
BIN
BIN2GREY
BINVAL
BIT
BITWAIT
BLINK
BOOLEAN
BYTE
BYVAL

CALL
CAPTURE1
CASE
CHECKSUM
CHR()
CIRCLE
CLS
CLOSE
COMPARE1A
COMPARE1B
CONFIG
CONST
COS
COSH
COUNTER
COUNTER0
COUNTER1
COUNTER2
CPEEK()
CPEEKH()
CRC8
CRC16
CRYSTAL
CURSOR

DATA
DATE$
DBG
DEBOUNCE
DECR
DECLARE
DEFBIT

DEFBYTE
DEFLNG
DEFWORD
DEG2RAD
DEGSNG
DEFLCDCHAR
DEFINT
DEFWORD
DELAY
DIM
DISABLE
DISKSIZE
DISKFREESIZE
DISPLAY
DO
DOWNTO
DTMFOUT

ELSE
ELSEIF
ENABLE
END
EOF
ERAM
ERASE
ERR
EXIT
EXP
EXTERNAL

FIX
FLUSH
FOR
FOURTH
FOURTHLINE
FREEFILE
FUNCTION

GATE
GET
GETADC()
GETKBD
GETATKBD
GETRC5()
GLCDDATA
GLCDCMD
GOSUB
GOTO
GREY2BIN

HEXVAL()
HIGH()
HOME

I2CINIT
I2CRECEIVE
I2CSEND
I2CSTART
I2CSTOP
I2CRBYTE
I2CWBYTE
IDLE
IF
INCR
INKEY
INP()
INPUT
INPUTBIN
INPUTHEX
INT
INT0
INT1
INTEGER
INTERNAL
INSTR
IS
ISCHARWAITING

LCASE()
LCD
LCDAT
LEFT
LEFT()
LEN()
LINE
LOAD
LOADLABEL
LOC
LOF
LOCAL
LOCATE
LOG
LOG10
LONG
LOOKUP()
LOOKUPSTR()

LOOP
LTRIM()
LOOKDOWN
LOW()
LOWER
LOWERLINE

MAKEBCD()
MAKEDEC()
MAKEINT()
MID()
MIN
MAX
MOD
MODE

NACK
NEXT
NOBLINK
NOSAVE
NOT

OFF
ON
OR
OUT
OUTPUT

PEEK()
POKE
PORTA
PORTB
PORTC
PORTD
PORTE
PORTF
PORTG
POWER
POWERDOWN
PRINT
PRINTBIN
PULSEOUT
PUT
PWM1A
PWM1B

RAD2DEG
RC5SEND
RC6SEND
READ
READEEPROM
REM
RESET
RESTORE
RETURN
RIGHT
RIGHT()
ROTATE
ROUND
RTRIM()

SEEK
SELECT
SERIAL
SET
SERIN
SEROUT
SETFONT
SGN
SHIFT
SHIFTLCD
SHIFTCURSOR
SHIFTIN
SHIFTOUT
SHOWPIC
SHOWPICE
SIN
SINH
SONYSEND
SOUND
SPACE()
SPC
SPIINIT
SPIIN
SPIMOVE
SPIOUT
START
STEP
STR()
STRING()
STOP
STOP TIMER
SUB
SWAP
SQR

TAN
TANH
THEN
TIME$
THIRD
THIRDLINE
TIMER0
TIMER1
TIMER2
TO
TRIM()

UCASE()
UNTIL
UPPER
UPPERLINE

VAL()
VARPTR()

WAIT
WAITKEY()
WAITMS
WAITUS
WATCHDOG
WRITEEEPROM
WEND
WHILE
WORD

XOR
XRAM

Error Codes

The following table lists errors that can occur.

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

32	Variable not dimensioned
33	GLOBAL variable can not be ERASED
34	Invalid number of parameters
35	3 parameters expected
36	THEN expected
37	Invalid comparison operator
38	Operation not possible on BITS
39	FOR expected
40	Variable can not be used with RESET
41	Variable can not be used with SET
42	Numeric parameter expected
43	File not found
44	2 variables expected
45	DO expected
46	Assignment error
47	UNTIL expected
50	Value doesn't fit into INTEGER
51	Value doesn't fit into WORD
52	Value doesn't fit into LONG
60	Duplicate label
61	Label not found
62	SUB or FUNCTION expected first
63	Integer or Long expected for ABS()
64	, expected
65	device was not OPEN
66	device already OPENED
68	channel expected
70	BAUD rate not possible
71	Different parameter type passed then declared
72	Getclass error. This is an internal error.
73	Printing this FUNCTION not yet supported
74	3 parameters expected
80	Code does not fit into target chip
81	Use HEX(var) instead of PRINTHEX

82	Use HEX(var) instead of LCDHEX
85	Unknown interrupt source
86	Invalid parameter for TIMER configuration
87	ALIAS already used
88	0 or 1 expected
89	Out of range : must be 1-4
90	Address out of bounds
91	INPUT, OUTPUT, BINARY, or RANDOM expected
92	LEFT or RIGHT expected
93	Variable not dimensioned
94	Too many bits specified
95	Falling or rising expected for edge
96	Prescale value must be 1,8,64,256 or 1024
97	SUB or FUNCTION must be DECLARED first
98	SET or RESET expected
99	TYPE expected
100	No array support for IRAM variables
101	Can'tfind HW-register
102	Error in internal routine
103	= expected
104	LoadReg error
105	StoreBit error
106	Unknown register
107	LoadnumValue error
108	Unknown directive in device file
109	= expected in include file for .EQU
110	Include file not found
111	SUB or FUNCTION not DECLARED
112	SUB/FUNCTION name expected
113	SUB/FUNCTION already DECLARED
114	LOCAL only allowed in SUB or FUNCTION
115	#channel expected
116	Invalid register file
117	Unknown interrupt

200	.DEF not found
201	Low Pointer register expected
202	.EQU not found, probably using functions that are not supported by the selected chip
203	Error in LD or LDD statement
204	Error in ST or STD statement
205	} expected
206	Library file not found
207	Library file already registered
210	Bit definition not found
211	External routine not found
212	LOW LEVEL, RISING or FALLING expected
213	String expected for assignment
214	Size of XRAM string 0
215	Unknown ASM mnemonic
216	CONST not defined
217	No arrays allowed with BIT/BOOLEAN data type
218	Register must be in range from R16-R31
219	INT0-INT3 are always low level triggered in the MEGA
220	Forward jump out of range
221	Backward jump out of range
222	Illegal character
223	* expected
224	Index out of range
225	() may not be used with constants
226	Numeric of string constant expected
227	SRAM start greater than SRAM end
228	DATA line must be placed after the END statement
229	End Sub or End Function expected
230	You can not write to a PIN register
231	TO expected
232	Not supported for the selected micro
233	READ only works for normal DATA lines, not for EPROM data
234	') block comment expected first
235	'(block comment expected first

236	Value does not fit into byte
238	Variable is not dimensioned as an array
239	Invalid code sequence because of AVR hardware bug
240	END FUNCTION expected
241	END SUB expected
242	Source variable does not match the target variable
243	Bit index out of range for supplied data type
244	Do not use the Y pointer
245	No arrays supported with IRAM variable
246	No more room for .DEF definitions
247	. expected
248	BYVAL should be used in declaration
249	ISR already defined
250	GOSUB expected
251	Label must be named SECTIC
252	Integer or Word expected
253	ERAM variable can not be used
254	Variable expected
255	Z or Z+ expected
256	Single expected
257	"" expected
258	SRAM string expected
259	- not allowed for a byte
260	Value larger than string length
261	Array expected
262	ON or OFF expected
263	Array index out of range
264	Use ECHO OFF and ECHO ON instead
265	offset expected in LDD or STD like Z+1
266	TIMER0, TIMER1 or TIMER2 expected
267	Numeric constant expected
268	Param must be in range from 0-3
269	END SELECT expected
270	Address already occupied

322	Data type not supported with statement
323	Label too long
324	Chip not supported by I2C slave library
325	Pre-scale value must be 1,8,32,128,256 or 1024
326	#ENDIF expected
327	Maximum size is 255
328	Not valid for SW UART
999	DEMO/BETA only supports 2048 bytes of code

Other error codes are internal ones. Please report them when you get them.

Newbie problems

When you are using the AVR without knowledge of the architecture you can experience some problems.

-I can not set a pin high or low

-I can not read the input on a pin

The AVR has 3 registers for each port. A port normally consist of 8 pins. A port is named with a letter from A-F.

All parts have PORT**B**.

When you want to set a single pin high or low you can use the SET and RESET statements. But before you use them the AVR chip must know in which direction you are going to use the pins.

Therefore there is a register named DDRx for each port. In our sample it is named DDR**B**. When you write a 0 to the bit position of the pin you can use the pin as an input. When you write a 1 you can use it as output.

After the direction bit is set you must use either the PORTx register to set a logic level or the PINx register to READ a pin level.

Yes the third register is the PINx register. In our sample PINB.

For example :

```
DDRB = &B1111_0000 ' upper nibble is output, lower nibble is input
SET PORTB.7 'will set the MS bit to +5V
RESET PORTB.7 'will set MS bit to 0 V
```

To read a pin :

```
Print PINB.0 'will read LS bit and send it to the RS-232
```

You may also read from PORTx but it will return the value that was last written to it.

To read or write whole bytes use :

```
PORTB = 0 'write 0 to register making all pins low
PRINT PINB 'print input on pins
```

I want to write a special character but they are not printed correct:

Well this is not a newbie problem but I put it here so you could find it.

Some ASCII characters above 127 are interpreted wrong depending on country settings. To print the right value use : PRINT "Test{123}?"

The {xxx} will be replaced with the correct ascii character.

You must use 3 digits otherwise the compiler will think you want to print {12} for example. This should be {012}

Tips and tricks

This section describes tips and tricks received from users.

Kyle Kronyak : Using all the RAM from an external RAM chip.

I have found a way to use the 607 bytes of external SRAM that are normally not available when using hardware SRAM support with BASCOM-AVR. It's actually quite simple. Basically the user just has to disconnect A15 from /CE on the SRAM module, and tie /CE to ground. This makes the chip enabled all the time. Addresses 1-32768 will then be available! The reason is because normally when going above 32768, the A15 pin would go high, disabling the chip. When A15 is not connected to /CE, the chip is always enabled, and allows the address number to "roll over". Therefore address 32162 is actually 0, 32163 is actually 1, 32164 is actually 2, etc. I have only tested this on a 32k SRAM chip. It definitely won't work on a 64k chip, and I believe it already works on any chip below 32k without modification of the circuit.

Programming problems

- When you have unreliable results, use a shielded LPT cable
- The AVR chips have a bug, if the erase is not complete.. It tend's to hang at some point. sometimes although the system reports erased but blank check report "not empty". As per Atmel Data Errrta You must drop the vcc by 0.5V (a diode 1N4148 in Series) if the erase is not happening. (Such Chip's are unreliable and hence can be used only if you are sure). This can happen after you have programmed the chip many times
-

Links

Here are some links to software or information that might be useful:

A WINZIP clone to ZIP and UNZIP software

http://ipsoft.cjb.net/

$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 precede each line with the **!** sign.

Most ASM mnemonics can be used without the preceding **!** too.

See also the chapter Mixing BASIC and Assembly and assembler mnemonics

Example

```
Dim C As Byte

Loadadr C , X 'load address of variable C into register X


$asm
Ldi R24,1 'load register R24 with the constant 1
St X,R24 ;store 1 into variable c
$end Asm
Print C
End
```

$BAUD

Action

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

Syntax

$BAUD =var

Remarks

Var	The baud rate that you want to use.

var : Constant.

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

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

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a XTAL that is a multiple of the baud rate.

See also

$CRYSTAL , BAUD

Example

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

$BAUD1

Action

Instruct the compiler to set the baud rate for the second hardware UART.

Syntax

$BAUD1 =var

Remarks

Var	The baud rate that you want to use.

var : Constant.

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

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a XTAL that is a multiple of the baud rate.

See also

$CRYSTAL , BAUD , $BAUD

Example

```
$baud1 = 2400
$crystal = 14000000 ' 14 MHz crystal
Open "COM2:" For BINARY As #1
Print#1, "Hello"
'Now change the baud rate in a program
Baud1 = 9600 '
Print#1, "Did you change the terminal emulator baud rate too?"
Close #1
End
```

$BGF

Action

Includes a BASCOM Graphic File.

Syntax

$BGF "file"

Remarks

file	The file name of the BGF file to include.

Use SHOWPIC to display the BGF file.

See also

SHOWPIC , PSET , 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
```

$BOOT

Action

Instruct the compiler to include boot loader support.

Syntax

$BOOT =address

Remarks

address	The boot loader address.

Some new AVR chips have a special boot section in the upper memory of the flash.

By setting some fuse bits you can select the code size of the boot setion.

The code size also determines the address of the boot loader.

With the boot loader you can reprogram the chip when a certain condition occurs.

The sample checks a pin to see if a new program must be loaded.

When the pin is low there is a jump to the boot address.

The boot code must always be located at the end of your program.

It must be written in ASM since the boot loader may not access the application flash rom. This because otherwise you could overwrite your running code!

The example is written for the M163. You can use the Upload file option of the terminal emulator to upload a new hex file. The terminal emulator must have the same baud rate as the chip. Under Options, Monitor, set the right upload speed and set a monitor delay of 20. Writing the flash take time so after every line a delay must be added while uploading a new file.

See also

BOOT128.BAS example and BOOT128X.BAS example from the sample dir.

Partial Example

Look at the BOOT.BAS example for the boot loader section.

```
'--------------------------------------------------------------
' BOOT.BAS
' Bootloader example for the M163
' set fusebit FE to 512 bytes for bootspace for this example
'At start up a message is displayed. When you make PIND.7 low the
'bootloader will be started
'This program serves as an example. It can be changed for other chips.
'Especially the page size and the boot entry location might need a change
'--------------------------------------------------------------

'Our communication settings
$crystal = 4000000
$baud = 19200
Print "Checking bootloader"
Portd.7 = 1
If Pind.7 = 0 Then
Print "Entering bootloader"
jmp $1e00 ' make a jump to the boot code location. See the datasheet for
the entrypoint
End If
Print "Not entering bootloader"

'you code would continue here
End
```

$CRYSTAL

Action

Instruct the compiler to override the crystal frequency options setting.

Syntax

$CRYSTAL =var

Remarks

var	Frequency of the crystal.

var : Constant.

The frequency is selectable from the Compiler Settings. It is stored in a configuration file. The $CRYSTAL directive overrides this setting.

See also

$BAUD , BAUD

Example

```
$baud = 2400
$crystal = 4000000
Print "Hello"
End
```

$DATA

Action

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

Syntax

$DATA

Remarks

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

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

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

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

$EEPROM and $DATA.

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

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

The READ statement that is used to read the DATA info may only be used with normal DATA lines. It does not work with DATA stored in EEPROM.

See also

$EEPROM , READEEPROM , WRITEEEPROM

ASM

NONE

Example

```
'-------------------------------------------------------------
' READDATA.BAS
' Copyright 1999-2002 MCS Electronics
'-------------------------------------------------------------

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

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

```
Next

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


Restore Dta4
Read L : Print L 'long type

End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

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

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

$DBG

Action

Enables debugging output to the hardware UART.

Syntax

$DBG

Remarks

Calculating the hardware, software and frame space can be a difficult task.

With $DBG the compiler will insert characters for the various spaces.

To the Frame space ‘F’ will be written. When you have a frame size of 4, FFFF will be written.

To the Hardware space ‘H’ will be written. If you have a hardware stack space of 8, HHHHHHHH will be written to this space.

To the software space ‘S’ will be written. If you have a software stack space of 6, SSSSSS will be written.

The idea is that when a character is overwritten, it is being used. So by watching the spaces you can determine if the space is used or not.

With the DBG statement a record is written to the HW UART. The record must be logged to a file so it can be analyzed by the stack analyzer.

Make the following steps to determine the proper values:

?? ????Make the frame space 40, the softstack 20 and the HW stack 50

?? ????Add $DBG to the top of your program

?? ????Add a DBG statement to every Subroutine or Function

?? ????Open the terminal emulator and open a new log file. By default it will have the name of your current program with the .log extension

?? ????Run your program and notice that it will dump information to the terminal emulator

?? ????When your program has executed all sub modules or options you have build in, turn off the file logging and turn off the program

?? ????Choose the Tools Stack analyzer option

?? ????A window will be shown with the data from the log file

?? ????Press the Advise button that will determine the needed space. Make sure that there is at least one H, S and F in the data. Otherwise it means that all the data is overwritten and that you need to increase the size.

?? ????Press the Use button to use the advised settings.

As an alternative you can watch the space in the simulator and determine if the characters are overwritten or not.

The DBG statement will assign an internal variable named ___SUBROUTINE

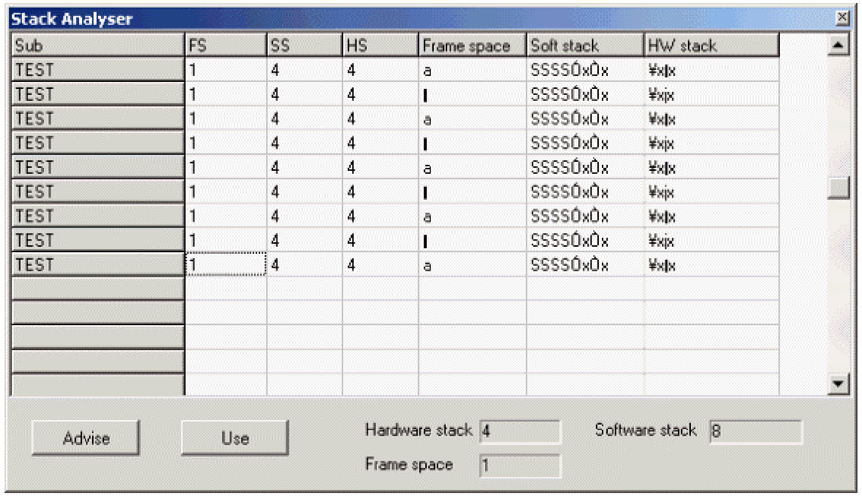
Because the name of a SUB or Function may be 32 long, this variable uses 33 bytes!

___SUBROUTINE will be assigned with the name of the current SUB or FUNCTION.

When you first run a SUB named Test1234 it will be assigned with Test1234

When the next DBG statement is in a SUB named Test, it will be assigned with Test.

The 234 will still be there so it will be shown in the log file.



Every DBG record will be shown as a row.

The columns are:

Column	Description
Sub	Name of the sub or function from where the DBG was used
FS	Used frame space
SS	Used software stack space
HS	Used hardware stack space
Frame space	Frame space
Soft stack	Soft stack space
HW stack	Hardware stack space

The Frame space is used to store temp and local variables.

It also stores the variables that are passed to subs/functions by value.

Because PRINT , INPUT and the FP num<>String conversion routines require a small buffer, the compiler always is using 16 bytes of frame space.

When the advise is to use 2 bytes of frame space, the setting will be 16+2=18.

For example when you use : print var, var need to be converted into a string before it can be printed or shown with LCD.

An alternative for the buffer would be to setup a temp buffer and free it once finished. This gives more code overhead.

In older version of BASCOM the start of the frame was used for the buffer but that gave conflicts when variables were printed from an ISR.

The buffer solution will be changed in a future version of BASCOM when a different approach will be used.

See also

DBG

$DEFAULT

Action

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

Syntax

$DEFAULT =var

Remarks

Var	SRAM, XRAM, ERAM

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

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

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

Each Dim statement will place the variable in XRAM in that case.

To switch back to the default behavior, use $END $DEFAULT

See also

NONE

ASM

NONE

Example

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

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

$EEPLEAVE

Action

Instructs the compiler not to recreate or erase the EEP file.

Syntax

$EEPLEAVE

Remarks

When you want to store data in the EEPROM, and you use an external tool to create the EEP file, you can use the $EEPLEAVE directive.

Normally the EEP file will be created or erased, but the directive will not touch any existing EEP file.

Otherwise you would erase an existing EEP file, created with another tool.

See also

$EEPROMHEX

Example

NONE

$EEPROM

Action

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

Syntax

$EEPROM

Remarks

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

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

A separate file is generated with the EEP extension. This file can be used to program the EEPROM. The build in STK200/300 programmer supports the EEP file.

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

$EEPROM and $DATA.

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

To switch back to the default behavior of the DATA lines, you must use the $DATA directive. In fact you need to switch back with $DATA always to the normal mode after your last DATA line when you are using $EEPROM.

When you use the normal mode there is no need to add $DATA or to switch back with $DATA.

It is important to know that the RESTORE and READ statements do NOT work with DATA lines that are stored in the EPROM.

RESTORE and READ only work with normal DATA lines.

The $EEPROM directive is only added to allow you to create a memory image of the EPROM.

To store and retrieve data from EPROM you should use an ERAM variable :

```
Dim Store As Eram Byte , B As Byte
B = 10 'assign value to b
Store = B 'value is stored in EPROM !
B = Store 'get the value back
```

See also

$DATA , WRITEEEPROM , READEEPROM

ASM

NONE

Example

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

Lbl:
DATA 100

$eeprom 'the following DATA lines data will go to the EEP 'file
Data 200

$data 'switch back to normal
Lbl2:
Data 300
```

$EEPROMHEX

Action

Instruct the compiler to store the data in the EEP file in Intel HEX format instead of binary format.

Syntax

$EEPROMHEX

Remarks

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

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM. $EEPROM must be used to create a EEP file that holds the data.

The EEP file is by default a binary file. When you use the STK500 you need an Intel HEX file. Use $EEPROMHEX to create an Intel Hex EEP file.

$EEPROMHEX must be used together with $EEPROM.

See also

[$EEPROMLEAVE]

Example

```
$eeprom 'the following DATA lines data will go to the EEP file
Data 200 , 100,50

$data

This would create an EEP file of 3 bytes. With the values 200,100 and 50.
Add $eeprommhex in order to create an Intel Hex file.

This is how the EEP filecontent looks when using $eeprommhex

:0A00000001020304050A141E283251
:00000001FF
```

$EXTERNAL

Action

Instruct the compiler to include ASM routines from a library.

Syntax

$EXTERNAL Myroutine [, myroutine2]

Remarks

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

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

See also

[$LIB]

Example

```
'-----------------------------------------------------------------------
-
' LIBDEMO.BAS
' (c) 2002 MCS Electronics
'In order to let this work you must put the mylib.lib file in the LIB dir
'And compile it to a LBX
'-----------------------------------------------------------------------
-
'define the used library
$lib "mylib.lbx"
'you can also use the original ASM :
'$LIB "mylib.LIB"

'also define the used routines
$external Test

'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X As Byte , Y As Byte)


'reserve some space
Dim Z As Byte


'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example
End
```

$INC

Action

Includes a binary file in the program at the current position.

Syntax

$INC label , size | nosize , "file"

Remarks

Label	The name of the label you can use to refer to the data.
Nosize	Specify either nosize or size. When you use size, the size of the data will be included. This way you know how many bytes you can retrieve.
File	Name of the file which must be included.

Use RESTORE to get a pointer to the data. And use READ, to read in the data.

The $INC statement is an alternative for the DATA statement.

While DATA works ok for little data, it is harder to use on large sets of data.

Example

```
'-----------------------------------------------------------------------
' (c) 2004 MCS Electronics
' $INC demo
'-----------------------------------------------------------------------

'do not confuse $inc with INC and $INCLUDE

$regfile = "m162def.dat"
$crystal = 4000000

Dim Size As Word , W As Word , B As Byte

Restore L1 ' set pointer to label
Read Size ' get size of the data

Print Size ; " bytes stored at label L1"
For W = 1 To Size
Read B : Print Chr(b);
Next

End

'include some data here
$inc L1 , Size , "c:\test.bas"
'when you get an error, insert a file you have on your system
```

$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) 1999-2002 MCS Electronics
'-----------------------------------------------------------------------
' file: INCLUDE.BAS
' demo: $INCLUDE
'-----------------------------------------------------------------------
Print "INCLUDE.BAS"
'Note that the file 123.bas contains an error
$include "123.bas" 'include file that prints Hello
Print "Back in INCLUDE.BAS"
End
```

To get the program working rename the file a_rename.bas into a.bas

The file a.bas is located in the samples dir.

$LCD

Action

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

Syntax

$LCD =[&H]*address*

Remarks

Address	The address where must be written to, to enable the LCD display and the RS line of the LCD display. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7) The RS line of the LCD can be configured with the LCDRS statement. On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

See also

$LCDRS

Example

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

Cls
LCD "Hello world"
```

$LCDPUTCTRL

Action

Specifies that LCD control output must be redirected.

Syntax

$LCDPUTCTRL = label

Remarks

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

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

See also

$LCDPUTDATA

Example

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

'define used crystal
$crystal = 4000000

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

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

End

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

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

MyoutputCtrl:
Pushall 'save all registers
```

```
'your code here
Popall 'restore registers
Return
```

$LCDPUTDATA

Action

Specifies that LCD data output must be redirected.

Syntax

$LCDPUTDATA =label

Remarks

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

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

See also

$LCDPUTCTRL

Example

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

'define used crystal
$crystal = 4000000

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

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

End

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

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

MyoutputCtrl:
```

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

$LCDRS

Action

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

Syntax

$LCDRS =[&H]*address*

Remarks

Address	The address where must be written to, to enable the LCD display. The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7) On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

See also

$LCD

Example

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

Cls
LCD "Hello world"
```

$LCDVFO

Action

Instruct the compiler to generate very short Enable pulse for VFO displays.

Syntax

$LCDVFO

Remarks

VFO based displays need a very short Enable pulse. Normal LCD displays need a longer pulse. To support VFO displays this compiler directive has been added.

The display need to be instruction compatible with normal HD44780 based text displays.

ASM

NONE

See also

NONE

Example

NONE

$LIB

Action

Informs the compiler about the used libraries.

Syntax

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

Remarks

Libname1 is the name of the library that holds ASM routines that are used by your program. More filenames can be specified by separating the names by a comma.

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

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

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

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

Creating your own LIB file

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

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

copyright = Your name

www = optional location where people can find the latest source

email = your email address

comment = AVR compiler library

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

date = date of last modification

statement = A statement with copyright and usage information

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

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

[test]

Test:

ldd r26,y+2 ; load address of X

ldd r27,y+3

ld r24,x ; get value into r24

Inc r24 ; value + 1

St x,r24 ; put back

ldd r26,y+0 ; address pf Y

ldd r27,y+1

st x,r24 ; store
ret ; ready
[end]

After you have saved your library in the LIB subdirectory you must compile it with the LIB Manager. Or you can include it with the LIB extension in which case you don't have to compile it.

About the assembler.

When you reference constants that are declared in your basic program you need to put a star(*) before the line.

'basicprogram

CONST myconst = 7

'asm lib

*** sbi portb, myconst**

By adding the *, the line will be compiled when the basic program is compiled. It will not be changed into object code in the LBX file.

When you use constants you need to use valid BASIC constants:

Ldi r24,12

Ldi r24, 1+1

Ldi r24, &B001

Ldi r24,0b001

Ldi r24,&HFF

Ldi r24,$FF

Ldi r24,0xFF

Other syntax is NOT supported.

See also

$EXTERNAL

Example

```
'-----------------------------------------------------------------------
-
' LIBDEMO.BAS
' (c) 2002 MCS Electronics
'In order to let this work you must put the mylib.lib file in the LIB dir
'And compile it to a LBX
'-----------------------------------------------------------------------
-
'define the used library
$lib "mylib.lib" 'the original asm will be used not the compiled object
code

'also define the used routines
$external Test

'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X As Byte , Y As Byte)


'reserve some space
Dim Z As Byte


'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example
End
```

$MAP

Action

Will generate label info in the report.

Syntax

$MAP

Remarks

The $MAP directive will put an entry for each line number with the address into the report file. This info can be used for debugging purposes with other tools.

See also

NONE

ASM

NONE

Example

```
$MAP
```

$NOINIT

Action

Instruct the compiler to generate code without initialization code.

Syntax

$NOINIT

Remarks

$NOINIT could be used together with $ROMSTART to generate boot loader code.

See also

$ROMSTART

ASM

For a simple project the following code will be generated for a 2313:

```
RJMP _BASICSTART
RETI
RETI
RETI
RETI
RETI
RETI
RETI
RETI
RETI
RETI
_BASICSTART:
; disable the watchdog timer
ldi _temp1,$1F
out WDTCR,_temp1
ldi _temp1,$17
out WDTCR,_temp1
;Init stackpointer
Ldi R24,$DF ; hardware stack pointer
Out SPL,R24
ldi YL,$C8 ; softstack pointer
ldi ZL,$98
Mov _SPL,ZL ; point to start of frame data
Clr YH
Mov_SPH,YH
Ldi ZL,$7E ;number of bytes
Ldi ZH,$00
Ldi XL,$60 ; start of RAM
```

```
Ldi XH,$00
Clr R24
_ClearRAM:
St X+,R24 ; clear
Sbiw ZL,1
Brne _ClearRAM
Clr R6 ; clear internal used flags
;##### Dim X As Byte
;##### X = 1
Ldi _temp1,$01
Sts $0060,R24 ; write value to memory
```

As you can see this program just assigns 1 to a byte named X.

First the interrupt vectors are setup then the watchdog timer is cleared , the stacks are set up, the memory is cleared and an internal register R6 is cleared.

After that the program begins and you can see that 1 is written to variable X.

Now with $NOINIT the code would look like this :

```
_BASICSTART:
; disable the watchdog timer
ldi _temp1,$1F
out WDTCR,_temp1
ldi _temp1,$17
out WDTCR,_temp1
;Init stackpointer
Ldi R24,$DF ; hardware stack pointer
Out SPL,R24
ldi YL,$C8 ; softstack pointer
ldi ZL,$98
Mov _SPL,ZL ; point to start of frame data
Clr YH
Mov _SPH,YH
Ldi ZL,$7E ;number of bytes
Ldi ZH,$00
Ldi XL,$60 ; start of RAM
Ldi XH,$00
Clr R24
_ClearRAM:
St X+,R24 ; clear
Sbiw ZL,1
Brne _ClearRAM
Clr R6 ; clear internal used flags
;##### Dim X As Byte
;##### X = 1
Ldi _temp1,$01
Sts $0060,R24 ; write value to memory
```

As you can see the difference is that the interrupt vectors are not setup.

The intention for the $NOINIT directive is to create support for a boot loader. As the boot loader needs are not studied yet, the $NOINIT will most likely be changed in the near future.

$NORAMCLEAR

Action

Instruct the compiler to not generate initial RAM clear code.

Syntax

$NORAMCLEAR

Remarks

Normally the SRAM is cleared in the initialization code. When you don't want the SRAM to be cleared(set to 0) you can use this directive.

Because all variables are automatically set to 0 or ""(strings) without the $NORAMCLEAR, using $NORAMCLEAR will set the variables to an unknown value. That is, the variables will probably set to FF but you cannot count on it.

See also

$NOINIT

$PROG

Action

Directive to auto program the lock and fuse bits.

Syntax

$PROG LB, FB , FBH , FBX

Remarks

While the lock and fusebits make the AVR customizable, the settings for your project can give some problems.

The $PROG directive will create a file with the projectname and the PRG extension.

Every time you program the chip, it will check the lock and fuse bit settings and will change them if needed.

So in a new chip, the lock and fusebits will be set automaticly. A chip that has been programmed with the desired settings will not be changed.

The programmer has an option to create the PRG file from the current chip settings.

The LB, FH, FBH and FBX values are stored in hexadecimal format in the PRJ file.

You may use any notation as long as it is a numeric constant.

Some chips might not have a setting for FBH or FBX, or you might not want to set all value. In that case, do NOT specify the value. For example:

$PROG &H20 ,,,

This will only write the Lockbit settings.

$PROG ,,&H30,

This will only write the FBH settings.

LB	Lockbit settings
FB	Fusebit settings
FBH	Fusebit High settings
FBX	Extended Fusebit settings

Sometimes the datasheet refers to the Fusebit as the Fusebit Low settings.

The $PROG setting is only supported by the AVRISP, STK200/300, Sample Electronics and Universal MCS Programmer Interface.

See also

Programmers

$REGFILE

Action

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

Syntax

$REGFILE = "name"

Remarks

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

The $REGFILE statement overrides the setting from the Options menu.

The settings are stored in a <project>.CFG file and the directive is added for compatibility with BASCOM-8051

The $REGFILE directive must be the first statement in your program. It may not be put into an included file since only the main source file is checked for the $REGFILE directive.

See also

NONE

ASM

NONE

Example

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

$ROMSTART

Action

Instruct the compiler to generate a hex file that starts at the specified address.

Syntax

$ROMSTART =address

Remarks

Address	The address where the code must start. By default the first address is 0. The bin file will still begin at address 0..

The $ROMFILE could be used to locate code at a different address for example for a boot loader.

See also

NONE

ASM

NONE

Example

```
$ROMSTART = &H4000
```

$SERIALINPUT

Action

Specifies that serial input must be redirected.

Syntax

$SERIALINPUT =label

Remarks

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

With the redirection of the INPUT command, you can use your own input 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.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM port. When you want to use a keyboard or remote control as the input device you can write a custom routine that puts the data into register R24 once it asks for this data.

See also

$SERIALOUTPUT

Example

```
'--------------------------------------------------
' $serialinput.bas
' (c) 1999 MCS Electronics
' demonstrates $SERIALINPUT redirection of serial input
'--------------------------------------------------
'define chip to use
$regfile = "8535def.dat"

'define used crystal
$crystal = 4000000

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

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

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

'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and restore
'all registers so we can use all BASIC statements
'$SERIALINPUT requires that the character is passed back in R24
Myinput:
Pushall 'save all registers
W = 0 'reset counter
Myinput1:
Incr W 'increase counter
Sbis USR, 7 ' Wait for character
Rjmp myinput2 'no charac waiting so check again
Popall 'we got something
Err = 0 'reset error
In _temp1, UDR ' Read character from UART
Return 'end of routine
Myinput2:
If W > 1000000 Then 'with 4 MHz ca 10 sec delay
rjmp Myinput_exit 'waited too long
Else
Goto Myinput1 'try again
End If
Myinput_exit:
Popall 'restore registers
Err = 1 'set error variable
ldi R24, 13 'fake enter so INPUT will end
Return
```

$SERIALINPUT1

Action

Specifies that serial input of the second UART must be redirected.

Syntax

$SERIALINPUT1 =label

Remarks

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

With the redirection of the INPUT command, you can use your own input 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.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM2 port. When you want to use a keyboard or remote control as the input device you can write a custom routine that puts the data into register R24 once it asks for this data.

See also

$SERIALOUTPUT1

$SERIALINPUT2LCD

Action

This compiler directive will redirect all serial input to the LCD display instead of echo-ing 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

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

$SERIALOUTPUT

Action

Specifies that serial output must be redirected.

Syntax

$SERIALOUTPUT =label

Remarks

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

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 , $SERIALINPUT2LCD

Example

```
$serialoutput = Myoutput
'your program goes here
Do
Print "Hello"
Loop
End

myoutput:
'perform the needed actions here
'the data arrives in R24
'just set the output to PORTB
!out portb,r24
ret
```

$SERIALOUTPUT1

Action

Specifies that serial output of the second UART must be redirected.

Syntax

$SERIALOUTPUT1 =label

Remarks

Label	The name of the assembler routine that must be called when a character is send to the serial buffer (UDR1). The character is placed into R24.

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

$SERIALINPUT1

$SIM

Action

Instruct the compiler to generate empty wait loops for the WAIT and WAITMS statements. This to allow faster simulation.

Syntax

$SIM

Remarks

Simulation of a WAIT statement can take a long time especially when memory view windows are opened.

The $SIM compiler directive instructs the compiler to not generate code for WAITMS and WAIT. This will of course allows faster simulation.

When your application is ready you must remark the $SIM directive or otherwise the WAIT and WAITMS statements will not work as expected.

When you forget to remove the $SIM option and you try to program a chip you will receive a warning that $SIM was used.

See also

NONE

ASM

NONE

Example

```
$SIM
Do
Wait 1
Loop
```

$TINY

Action

Instruct the compiler to generate initialize code without setting up the stacks.

Syntax

$TINY

Remarks

The tiny11 for example is a powerful chip. It only does not have SRAM. BASCOM depends on SRAM for the hardware stack and software stack.

When you like to program in ASM you can use BASCOM with the $TINY directive.

Some BASCOM statements will also already work but the biggest part will not work.

BASCOM will support a subset of the BASCOM statements and function to be used with the chips without SRAM. There will be a special tiny.lib that will use little registers and will have at most a 3 level deep call since tiny chips do have a 3 level deep hardware stack that may be used for calls.

Note that the generated code is not yet optimized for the tiny parts. The $tiny directive is just a start of the tiny parts implementation!

No support is available for this feature until the tiny.lib is implemented.

See also

NONE

ASM

NONE

Example

```
$tiny
dim X AS iram BYTE, y AS iram BYTE
X = 1 : Y = 2 : X = x + y
```

$WAITSTATE

Action

Compiler directive to activate external SRAM and to insert a WAIT STATE for a slower ALE signal.

Syntax

$WAITSTATE

Remarks

The $WAITSTATE can be used to override the Compiler Chip Options setting.

See also

NA

Example

```
$WAITSTATE
```

$XRAMSIZE

Action

Specifies the size of the external RAM memory.

Syntax

$XRAMSIZE =[&H] size

Remarks

Size	Size of external RAM memory chip.

size : Constant.

The size of the chip can be selected from the Options Compiler Chip menu.
The $XRAMSIZE overrides this setting. It is important that $XRAMSTART precedes $XRAMSIZE

See also

$XRAMSTART

Example

```
$XRAMSTART = &H300

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

$XRAMSTART

Action

Specifies the location of the external RAM memory.

Syntax

$XRAMSTART =[&H]address

Remarks

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

address : Constant.

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

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

It is important that when you use $XRAMSTART and $XRAMSIZE that $XRAMSTART comes before $XRAMSIZE.

See also

$XRAMSIZE

Example

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

1WIRECOUNT

Action

This statement reads the number of 1wire devices attached to the bus.

Syntax

var2 **= 1WIRECOUNT( )**

var2 **= 1WIRECOUNT(** port , pin **)**

Remarks

var2	A WORD variable that is assigned with the number of devices on the bus.
port	The PIN port name like PINB or PIND.
pin	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

The variable must be of the type word or integer.

You can use the 1wirecount() function to know how many times the 1wsearchNext() function should be called to get all the ID's on the bus.

The 1wirecount function will take 4 bytes of SRAM.

___1w_bitstorage , Byte used for bit storage :

lastdeviceflag bit 0

id_bit bit 1

cmp_id_bit bit 2

search_dir bit 3

___1wid_bit_number, Byte

___1wlast_zero, Byte

___1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib.

_1wire_Count : (calls _1WIRE, _1WIRE_SEARCH_FIRST , _1WIRE_SEARCH_NEXT)

Parameters passed : R24 : pin number, R30 : port , Y+0,Y+1 : 2 bytes of soft stack, X : pointer to the frame space

Returns Y+0 and Y+1 with the value of the count. This is assigned to the target variable.

See also

1WWRITE , 1WRESET , 1WREAD , 1WSEARCHFIRST, 1WSEARCHNEXT

Example

```
'-----------------------------------------------------------------------
-
' '1wireSearch.bas
```

```
' (c) 2000 MCS Electronics
' revision b, 27 dec 2000
'----------------------------------------------------------------------
-
Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
' lastdeviceflag bit 0
' id_bit bit 1
' cmp_id_bit bit 2
' search_dir bit 3
'___1wid_bit_number, Byte
'___1wlast_zero, Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8 'print the number
Print Hex(reg_no(i));
Next
Print

Do
'Now search for other devices
Reg_no(1) = 1wsearchnext()
For I = 1 To 8
Print Hex(reg_no(i));
Next
Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line 1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1) 'for example look at pin PINB.1
End
```

1WRESET

Action

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

Syntax

1WRESET

1WRESET , PORT , PIN

Remarks

1WRESET	Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred
Port	The register name of the input port. Like PINB, PIND.
Pin	The pin number to use. In the range from 07. May be a numeric constant or variable.

The variable ERR is set when an error occurs.

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

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

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

The syntax for additional 1-wire devices is :

1WRESET **port** , **pin**

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

var = 1WREAD(bytes) , for the configured 1 wire pin

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

See also

1WREAD , 1WWRITE

Example

```
'--------------------------------------------------------------
' 1WIRE.BAS (c) 2002 MCS Electronics
' demonstrates 1wreset, 1wwrite and 1wread()
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'--------------------------------------------------------------
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


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

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8) 'read 8 bytes

For I = 1 To 8
Print Hex(ar(i)); 'print output
Next
Print 'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
Ar(i) = 0 'clear array to see that it works
Next

1wreset Pinb , 2 'use this port and pin for the second device
1wwrite &H33 , 1 , Pinb , 2 'note that now the number of bytes must be
specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2) 'read 8 bytes from portB on pin 2

For I = 1 To 8
Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3 'for pin 0-3
1wreset Pinb , I
1wwrite &H33 , 1 , Pinb , I
Ar(1) = 1wread(8 , Pinb , I)
For A = 1 To 8
Print Hex(ar(a));
Next
Print
Next

End
```

1WREAD

Action

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

Syntax

var2 = **1WREAD(** [bytes] **)**

var2 = **1WREAD(** bytes , port , pin **)**

Remarks

var2	Reads a byte from the bus and places it into var2. Optional the number of bytes to read can be specified.
Port	The PIN port name like PINB or PIND.
Pin	The pin number of the port. In the range from 0-7. Maybe a numeric constant or variable.

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

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

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

The syntax for additional 1-wire devices is :

1WRESET port, pin

1WWRITE var/constant , bytes, port, pin

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

See also

1WWRITE , 1WRESET

Example

```
'--------------------------------------------------------------
' 1WIRE.BAS (c) 2002 MCS Electronics
' demonstrates 1wreset, 1wwrite and 1wread()
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'--------------------------------------------------------------
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte


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

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8) 'read 8 bytes

For I = 1 To 8
Print Hex(ar(i)); 'print output
Next
Print 'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
Ar(i) = 0 'clear array to see that it works
Next

1wreset Pinb , 2 'use this port and pin for the second device
1wwrite &H33 , 1 , Pinb , 2 'note that now the number of bytes must be
specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2) 'read 8 bytes from portB on pin 2

For I = 1 To 8
Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3 'for pin 0-3
1wreset Pinb , I
1wwrite &H33 , 1 , Pinb , I
Ar(1) = 1wread(8 , Pinb , I)
For A = 1 To 8
Print Hex(ar(a));
Next
Print
Next

End
```

1WSEARCHFIRST

Action

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

Syntax

var2 = **1WSEARCHFIRST( )**

var2 = **1WSEARCHFIRST(** port , pin **)**

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 first 1wire device on the bus.
port	The PIN port name like PINB or PIND.
pin	The pin number of the port. In the range from 0-7. Maybe a numeric constant or variable.

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 nul may be returned as a value and the nul is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM.

___1w_bitstorage , Byte used for bit storage :

lastdeviceflag bit 0

id_bit bit 1

cmp_id_bit bit 2

search_dir bit 3

___1wid_bit_number, Byte

___1wlast_zero, Byte

___1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib.

_1wire_Search_First : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)

Parameters passed : R24 : pin number, R30 : port , X : address of target array

Returns nothing.

See also

1WWRITE , 1WRESET , 1WREAD , 1WSEARCHNEXT, 1WIRECOUNT

Example

```
'--------------------------------------------------------------------
-
' '1wireSearch.bas
' (c) 2000 MCS Electronics
' revision b, 27 dec 2000
'--------------------------------------------------------------------
-
Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
' lastdeviceflag bit 0
' id_bit bit 1
' cmp_id_bit bit 2
' search_dir bit 3
'___1wid_bit_number, Byte
'___1wlast_zero, Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8 'print the number
Print Hex(reg_no(i));
Next
Print

Do
'Now search for other devices
Reg_no(1) = 1wsearchnext()
For I = 1 To 8
Print Hex(reg_no(i));
Next
Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
```

```
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line 1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1) 'for example look at pin PINB.1
End
```

1WSEARCHNEXT

Action

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

Syntax

var2 = **1WSEARCHNEXT()**

var2 = **1WSEARCHNEXT(** port , pin **)**

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.
Port	The PIN port name like PINB or PIND.
Pin	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

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 nul may be returned as a value and the nul is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM.

___1w_bitstorage , Byte used for bit storage :

lastdeviceflag bit 0

id_bit bit 1

cmp_id_bit bit 2

search_dir bit 3

___1wid_bit_number, Byte

___1wlast_zero, Byte

___1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib.

_1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)

Parameters passed : R24 : pin number, R30 : port , X : address of target array

Returns nothing.

See also

1WWRITE , 1WRESET , 1WREAD , 1WSEARCHFIRST, 1WIRECOUNT

Example

```
'------------------------------------------------------------------------
-
' '1wireSearch.bas
' (c) 2000 MCS Electronics
' revision b, 27 dec 2000
'------------------------------------------------------------------------
-
Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
' lastdeviceflag bit 0
' id_bit bit 1
' cmp_id_bit bit 2
' search_dir bit 3
'___1wid_bit_number, Byte
'___1wlast_zero, Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8 'print the number
Print Hex(reg_no(i));
Next
Print

Do
'Now search for other devices
Reg_no(1) = 1wsearchnext()
For I = 1 To 8
Print Hex(reg_no(i));
Next
Print
Loop Until Err = 1



'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line 1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1) 'for example look at pin PINB.1
End
```

1WVERIFY

Action

This verifies if an ID is available on the 1wire bus.

Syntax

1WVERIFY ar(1)

Remarks

Ar(1)	A byte array that holds the ID to verify.

Returns ERR set to 0 when the ID is found on the bus otherwise it will be 1.

ASM

The following asm routines are called from mcs.lib.

_1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)

See also

1WWRITE , 1WRESET , 1WREAD , 1WSEARCHFIRST, 1WIRECOUNT

Example

```
'--------------------------------------------------------------------------
-
' '1wireSearch.bas
' (c) 2000 MCS Electronics
' revision b, 27 dec 2000
'--------------------------------------------------------------------------
-
Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted

'The following internal bytes are used by the scan routines
'___1w_bitstorage , Byte used for bit storage :
' lastdeviceflag bit 0
' id_bit bit 1
' cmp_id_bit bit 2
' search_dir bit 3
'___1wid_bit_number, Byte
'___1wlast_zero, Byte
'___1wlast_discrepancy , Byte
'___1wire_data , string * 7 (8 bytes)

'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte

'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word

'Now search for the first device on the bus
Reg_no(1) = 1wsearchfirst()

For I = 1 To 8 'print the number
Print Hex(reg_no(i));
Next
Print

Do
'Now search for other devices
Reg_no(1) = 1wsearchnext()
For I = 1 To 8
Print Hex(reg_no(i));
Next
Print
Loop Until Err = 1


'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the 1wirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W


'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line 1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1) 'for example look at pin PINB.1
End
```

1WWRITE

Action

This statement writes a variable to the 1wire bus.

Syntax

1WWRITE var1

1WWRITE var1, bytes

1WWRITE var1 , bytes , port , pin

Remarks

var1	Sends the value of var1 to the bus. The number of bytes can be specified too but this is optional.
bytes	The number of bytes to write. Must be specified when port and pin are used.
port	The name of the PORT PINx register like PINB or PIND.
pin	The pin number in the range from 0-7. May be a numeric constant or variable.

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

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

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

The syntax for additional 1-wire devices is :

1WRESET **port , pin**

1WWRITE **var/constant, bytes, port , pi**n

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

See also

1WREAD , 1WRESET

Example

```
'--------------------------------------------------------------
' 1WIRE.BAS (c) 2000 MCS Electronics
' demonstrates 1wreset, 1wwrite and 1wread()
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
'--------------------------------------------------------------
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"


Config 1wire = Portb.0 'use this pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte

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

'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8) 'read 8 bytes

For I = 1 To 8
Print Hex(ar(i)); 'print output
Next
Print 'linefeed
Loop


'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!

'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
Ar(i) = 0 'clear array to see that it works
Next

1wreset Pinb , 2 'use this port and pin for the second device
1wwrite &H33 , 1 , Pinb , 2 'note that now the number of bytes must be
specified!
'1wwrite Ar(1) , 5,pinb,2

'reading is also different
Ar(1) = 1wread(8 , Pinb , 2) 'read 8 bytes from portB on pin 2

For I = 1 To 8
Print Hex(ar(i));
Next

'you could create a loop with a variable for the bit number !
For I = 0 To 3 'for pin 0-3
1wreset Pinb , I
1wwrite &H33 , 1 , Pinb , I
Ar(1) = 1wread(8 , Pinb , I)
For A = 1 To 8
Print Hex(ar(a));
Next
Print
Next

End
```

ALIAS

Action

Indicates that the variable can be referenced with another name.

Syntax

newvar **ALIAS** oldvar

Remarks

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

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

See also

CONST

Example

```
Config Pinb.1 = Output
Direction Alias Portb.1 'now you can refer to PORTB.1 with the variable
direction
Do
Set Direction  'has the same effect as SET PORTB.1
Waitms 1
Reset Directopn
Loop
End
```

ABS()

Action

Returns the absolute value of a numeric signed variable.

Syntax

var = **ABS(**var2**)**

Remarks

Var	Variable that is assigned the absolut e value of var2.
Var2	The source variable to retrieve the absolute value from.

var : Integer , Long or Single.
var2 : Integer, Long or Single.

The absolute value of a number is always positive.

See also

NONE

Asm

Calls: _abs16 for an Integer and _abs32 for a Long
Input: R16-R17 for an Integer and R16-R19 for a Long
Output:R16-R17 for an Integer and R16-R19 for a Long
Calls _Fltabsmem for a single from the fp_trig library.

Example

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

ACOS

Action

Returns the arccosine of a single in radians.

Syntax

var =**ACOS** (x)

Remarks

Var	A single variable that is assigned with the ACOS of variable x.
X	The single to get the ACOS of. Input is valid from –1 to +1 and returns ? to 0. If Input is < -1 than ? and input is > 1 than 0 will returned.

If Input is cause of rounding effect in single-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-poin t back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within –1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , COS , SIN , TAN , ATN , ASIN , ATN2

Example

```
Dim S As Single , x As Single, y As Single
x= 0.5 : S= Acos(x)
Print S
```

ASC

Action

Assigns a numeric variable with the ASCII value of the first character of a string.

Syntax

var = ASC(string **)**

Remarks

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

var : Byte, Integer, Word, Long.

string : String, Constant.

Note that only the first character of the string will be used.

When the string is empty, a zero will be returned.

See also

CHR

Asm

NONE

Example

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

ASIN

Action

Returns the arcsine of a single in radians.

Syntax

var =**ASIN**(x)

Remarks

Var	A single variable that is assigned with the ASIN of variable x.
X	The single to get the ASIN of. Input is valid from –1 to +1 and returns -?/2 to +?/2. If Input is < -1 than -?/2 and input is > 1 than ?/2 will returned.

If Input is cause of rounding effect in single-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-point back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within –1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , COS , SIN , TAN , ATN , ACOS , ATN2

Example

```
Dim S As Single , x As Single, y As Single
x= 0.5 : S= Asin(x)
Print S
```

ATN

Action

Returns the Arctangent of a single in radians.

Syntax

var =**ATN** (single)

Remarks

Var	A numeric variable that is assigned with the arctangent of variable single.
Single	The single variable to get the arctangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , COS , SIN , TAN

Example

```
Dim S As Single
S = Atn(1) * 4
Print S ' prints 3.141593 PI
```

ATN2

Action

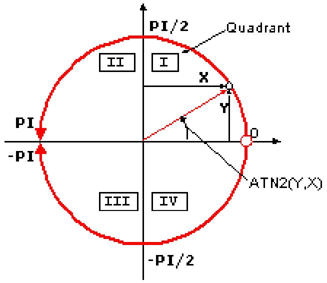
ATN2 is a four-quadrant arc-tangent.
While the ATN-function returns from -?/2 (-90°) to ?/2 (90°), the ATN2 function returns the whole range of a circle from -? (-180°) to +? (180°). The result depends on the ratio of Y/X and the signs of X and Y.

Syntax

var =**ATN2**(y, x)

Remarks

Var	A single variable that is assigned with the ATN2 of variable single.
X	The single variable with the distance in x-direction.
Y	The single variable with the distance in y-direction



Quadrant	Sign Y	Sign X	ATN2
I	+	+	0 to ? /2
II	+	-	?/2 to ?
III	-	-	-? /2 to -?
IV	-	+	0 to –? /2

If you go with the ratio Y/X into ATN you will get same result for X greater zero (right side in coordinate system) as with ATN2. ATN2 uses X and Y and can give information of the angle of the point over 360° in the coordinates system.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , COS , SIN , TAN , ATN

Example

```
Dim S As Single , x As Single, y As Single
S = Atn2(y,x)
Print S
```

BASE64DEC

Action

Converts Base-64 data into the original data.

Syntax

Result = **Base64Dec**(source)

Remarks

Result	A string variable that is assigned with the un-coded string.
Source	The source string that is coded with base-64.

Base-64 is not an encryption protocol. It sends data in 7-bit ASCII data format. MIME, web servers, and other Internet servers and clients use Base-64 coding.

The provided Base64Dec() function is a decoding function. It was written to add authentication to the webserver sample.

When the webserver asks for authentication, the client will send the user and password unencrypted, but base-64 coded to the webserver.

Base-64 coded strings are always in pairs of 4 bytes. These 4 bytes represent 3 bytes.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, CLOSESOCKET , SOCKETLISTEN

Example

Config Tcpip = Int0 , Mac = 00.00.12.34.56.78 , Ip = 192.168.0.10 , Submask =

BAUD

Action

Changes the baud rate for the hardware UART.

Syntax

BAUD = var

BAUD #x , const

Remarks

Var	The baud rate that you want to use.
X	The channel number of the software uart.
Const	A numeric constant for the baud rate that you want to use.

Do not confuse the BAUD statement with the $BAUD compiler directive.

And do not confuse $CRYSTAL and CRYSTAL

$BAUD overrides the compiler setting for the baud rate and BAUD will change the current baud rate.

BAUD = ... will work on the hardware UART.

BAUD #x, yyyy will work on the software UART.

See also

$CRYSTAL , $BAUD

Asm

NONE

Example

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

BCD

Action

Converts a variable stored in BCD format into a string.

Syntax

PRINT BCD(var**)**
LCD BCD(var **)**

Remarks

Var	Variable to convert.

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

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

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

See also

MAKEDEC , MAKEBCD

Asm

Calls: _BcdStr
Input: X hold address of variable
Output: R0 with number of bytes, frame with data.

Example

```
Dim A As Byte
A = 65
Print A ' 65
Print Bcd(a) ' 41
End
```

BIN

Action

Convert a numeric variable into the binary string representation.

Syntax

Var = Bin(source)

Remarks

Var	The target string that will be assigned with the binary representation of the variable source.
Source	The numeric variable that will be converted.

The BIN() function can be used to display the state of a port.
When the variable source has the value &B10100011 the string named var will be assigned with "10100011".
It can be easily printed to the serial port.

See also

HEX, STR , VAL , HEXVAL , BINVAL

ASM

NONE

Example

```
Dim A As Byte
A = &H14
Print Bin(a)
Prints 00010100
```

BINVAL

Action

Converts a string representation of a binary number into a number.

Syntax

var **= Binval(** s **)**

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type. Should contain only 0 and 1 digits.

See also

STR , HEXVAL , HEX , BIN , VAL

Example

```
Dim a as byte, s As String * 10
s = "1100"
a = BinVal(s) 'convert string
Print A ' 12
End
```

BIN2GREY

Action

Returns the Grey-code of a variable.

Syntax

var1 = **bin2grey**(var2)

Remarks

var1	Variable that will be assigned with the Grey code.
var2	A variable that will be converted.

Grey code is used for rotary encoders. Bin2grey() works with byte , integer, word and long variables.

The data type of the variable that will be assigned determines if a byte, word or long conversion will be done.

See also

GREY2BIN

ASM

Depending on the data type of the target variable the following routine will be called from mcs.lbx: _grey2Bin for bytes , _grey2bin2 for integer/word and _grey2bin4 for longs.

Example

```
'-----------------------------------------------------------------------
-------
' (c) 2001-2004 MCS Electronics
' This sample show the Bin2Grey and Grey2Bin functions
' Credits to Josef Franz Vögel for an improved and word/long extended
version
'-----------------------------------------------------------------------
-------

'Bin2Gey() converts a byte,integer,word or long into grey code.
'Grey2Bin() converts a grey code into a binary value

Dim B As Byte ' could be word,integer or long too

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
Print B ; Spc(10) ; Bin2grey(b)
Next

Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
Print B ; Spc(10) ; Grey2bin(b)
Next

End
```

BITWAIT

Action

Wait until a bit is set or reset.

Syntax

BITWAIT x **, SET/RESET**

Remarks

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

When using bit variables make sure that they are set/reset by software otherwise your program will stay in a loop.

When you use internal registers that can be set/reset by hardware such as PORTB.0 this doesn't apply since this state can change as a result from for example a key press.

See also

NONE

Asm

Calls: NONE

Input: NONE

Output: NONE

Code : shown for address 0-31

label1:

Sbic PINB.0,label2

Rjmp label1

Label2:

Example

```
Dim A As Bit
Bitwait A , Set 'wait until bit a is set
Bitwait Portb.7 , Reset 'wait until bit 7 of Port B is 0.
End
```

BLOAD

Action

Writes the Content of a File into SRAM

Syntax

BLoad sFileName, wSRAMPointer

Remarks

sFileName	(String) Name of the File to be read
wSRAMPointer	(Word) Variable, which holds the SRAM Address to which the content of the file should be written

This function writes the content of a file to a desired space in SRAM. A free handle is needed for this function.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , KILL , DISKFREE , DISKSIZE , GET , PUT

FILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	_BLoad	
Input	X: Pointer to string with filename	Z: Pointer to Long-variable, which holds the start position of SRAM
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'now the good old bsave and bload
Dim Ar(100) As Byte , I As Byte
For I = 1 To 100
Ar(i) = I ' fill the array
Next

Wait 2

W = Varptr(ar(1))
Bsave "josef.img" , W , 100
For I = 1 To 100
Ar(i) = 0 ' reset the array
Next

Bload "josef.img" , W ' Josef you are amazing !
```

```
For I = 1 To 10
Print Ar(i) ; "  " ;
Next
Print
```

BSAVE

Action

Save a range in SRAM to a File

Syntax

BSave sFileName, wSRAMPointer, wLength

Remarks

sFileName	(String) Name of the File to be written
wSRAMPointer	(Word) Variable, which holds the SRAM Address, from where SRAM should be written to a File
wLength	(Word) Count of Bytes from SRAM, which should be written to the file

This function writes a range from the SRAM to a file. A free file handle is needed for this function.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_BSave**	
Input	X: Pointer to string with filename	Z: Pointer to Long-variable, which holds the start position of SRAM
	r20/r21: Count of bytes to be written	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'now the good old bsave and bload
Dim Ar(100) As Byte , I As Byte
For I = 1 To 100
Ar(i) = I ' fill the array
Next

Wait 2

W = Varptr(ar(1))
Bsave "josef.img" , W , 100
For I = 1 To 100
Ar(i) = 0 ' reset the array
Next
```

```
Bload "josef.img" , W ' Josef you are amazing !

For I = 1 To 10
Print Ar(i) ; "  " ;
Next
Print
```

BYVAL

Action

Specifies that a variable will be passed by value.

Syntax

Sub Test(BYVAL var)

Remarks

Var	Variable name

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

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

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

By default passing by reference is used.

Note that calling by reference will generate less code.

See also

CALL , DECLARE , SUB , FUNCTION

ASM

NONE

Example

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

CALL

Action

Call and execute a subroutine.

Syntax

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

Remarks

Var1	Any BASCOM variable or constant.
Var-n	Any BASCOM variable or constant.
Test	Name of the subroutine. In this case Test.

You can call sub routines with or without passing parameters.

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

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

With the CALL statement, you can call a procedure or subroutine.

For example: **Call Test2**

The call statement enables you to implement your own statements.

You don't have to use the CALL statement:

Test2 will also call subroutine test2

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

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

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

See also

DECLARE , SUB, EXIT, FUNCTION , LOCAL

Example

```
Dim A As Byte , B As Byte 'dimension some variables
Declare Sub Test(b1 As Byte , Byval B2 As Byte)'declare the SUB program
A = 65 'assign a value to variable A
Call Test(a , 5) 'call test with parameter A and constant
Test A , 5 'alternative call
Print A 'now print the new value
End

Sub Test(b1 As Byte , Byval B2 As Byte) 'use the same variable names as
'the declared one
Print B1 'print it
Print Bcd(b2)
B1 = 10 'reassign the variable
B2 = 15 'reassign the variable
End Sub
```

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

Variable A is changed however.

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

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

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

CHECKSUM

Action

Returns a checksum of a string.

Syntax

PRINT Checksum(var**)**

b = Checksum(var**)**

Remarks

Var	A string variable.
B	A numeric variable that is assigned with the checksum.

The checksum is computed by counting all the bytes of the string variable.

Checksums are often used with serial communication.

The checksum is a byte checksum. The following VB code is equivalent :

Dim Check as Byte

Check = 255

For x = 1 To Len(s$)

Check = check – ASC(mid$(s$,x,1))

Next

See also

CRC8 , CRC16

Example

```
Dim S As String * 10 'dim variable
S = "test" 'assign variable
Print Checksum(s) 'print value (192)
End
```

CHR

Action

Convert a numeric variable or a constant to a string with a length of 1 character. The character represents the ASCII value of the numeric value.

Syntax

PRINT CHR(var**)**

s = CHR(var**)**

Remarks

Var	Numeric variable or numeric constant.
S	A string variable.

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

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

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

The Chr() function is handy in combination with the LCD custom characters where you ca redefine characters 0-7 of the ASCII table.

See also

ASC()

Example

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

CIRCLE

Action

Draws a circle on a graphic display.

Syntax

CIRCLE(x0,y0) , radius, color

Remarks

X0	Starting horizontal location of the line.
Y0	Starting vertical location of the line.
Radius	Radius of the circle
Color	Color of the circle

Example

```
'-----------------------------------------------------------------
' (c) 2001-2004 MCS Electronics
' T6963C graphic display support demo 240 * 128
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin connected to
' 1 GND GND
'2 GND GND
'3 +5V +5V
'4 -9V -9V potmeter
'5 /WR PORTC.0
'6 /RD PORTC.1
'7 /CE PORTC.2
'8 C/D PORTC.3
'9 NC not conneted
'10 RESET PORTC.4
'11-18 D0-D7 PA
'19 FS PORTC.5
'20 NC not connected

$crystal = 8000000
'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2

Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255 ' diagonal line
Line(0 , 127) -(239 , 0) , 255 ' diagonal line
Line(0 , 0) -(240 , 0) , 255 ' horizontal upper line
Line(0 , 127) -(239 , 127) , 255 'horizontal lower line
Line(0 , 0) -(0 , 127) , 255 ' vertical left line
Line(239 , 0) -(239 , 127) , 255 ' vertical right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
Pset X , 20 , 255 ' set the pixel
Next

For X = 0 To 140
Pset X , 127 , 255 ' set the pixel
Next

Wait 2

'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
Circle(20 , 20) , 20 , 255 ' show circle
Wait 1
Circle(20 , 20) , 20 , 0 'remove circle
Wait 1
Next

Wait 2
```

```
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje ' show 2 since we have a big display
Wait 2
Cls Text ' clear the text
End


'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

CLS

Action

Clear the LCD display and set the cursor to home.

Syntax

CLS

Syntax for graphical LCD

CLS
CLS TEXT
CLS GRAPH

Remarks

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

For graphical LCD displays CLS will clear both the text and the graphical display.

See also

$LCD , LCD , SHIFTLCD , SHIFTCURSOR , SHIFTLCD

Example

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

CLOCKDIVISION

Action

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

Syntax

CLOCKDIVISON = var

Remarks

Var	Variable or numeric constant that sets the clock division. Valid values are from 2-129. A value of 0 will disable the division.

On the MEGA 103 and 603 the system clock frequency can be divided so you can save power for instance. A value of 0 will disable the clock divider. The divider can divide from 2 to 127. So the other valid values are from 2- 127.

Some routines that rely on the system clock will not work proper anymore when you use the divider. WAITMS for example will take twice the time when you use a value of 2.

See also

POWERSAVE

Example

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

CLOSE

Action

Closes an opened device.

Syntax

OPEN "device" for MODE As #channel

CLOSE #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits. The format for COM1 is : COM1: Some chips have 2 UARTS. You can use COM2: to open the second HW UART. The format for the software UART is: COMpin:speed,8,N,stop bits[,INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. An optional parameter **,INVERTED** can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.
MODE	You can use BINARY or RANDOM for COM1 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 >0.

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

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

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

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

For the AVR-DOS filesystem, you may place the CLOSE at any place in your program. This because the filesystem supports real file handles.

See also

OPEN , PRINT

Example

```
'-----------------------------------------------------
' (c) 2000 MCS Electronics
' OPEN.BAS
```

```
' demonstrates software UART
'-------------------------------------------------------
$crystal = 10000000 'change to the value of the XTAL you have installed


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related
statements
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
'store in byte
B = Inkey(#2)
'when the value > 0 we got something
If B > 0 Then
Print #1 , Chr(b) 'print the character
End If
Loop Until B = 27


Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B


'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

CLOSESOCKET

Action

Closes a socket connection.

Syntax

CloseSocket socket

Remarks

Socket	The socket number you want to close in the range of 0 - 3. When the socket is already closed, no action will be performed.

You must close a socket when you receive the SOCK_CLOSE_WAIT status.

You may also close a socket if that is needed by your protocol.

You will receive a SOCK_CLOSE_WAIT status when the server closes the connection.

When you use CloseSocket you actively close the connection.

Note that it is not needed to wait for a SOCK_CLOSE_WAIT message in order to close a socket connection.

After you have closed the connection, you need to use GetSocket in order to use the socket number again.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, TCPREAD, SOCKETLISTEN

Example

```
Closesocket I ' close the connection
```

CONFIG

The CONFIG statement is used to configure the hardware devices.

DIRECTIVE	RE-USABLE
CONFIG 1WIRE	NO
CONFIG ACI	YES
CONFIG ADC	NO
CONFIG ATEMU	NO
CONFIG BCCARD	NO
CONFIG CLOCK	NO
CONFIG COM1	YES
CONFIG COM2	YES
CONFIG DATE	NO
CONFIG DEBOUNCE	NO
CONFIG GRAPHLCD	NO
CONFIG I2CDELAY	NO
CONFIG I2CSLAVE	NO
CONFIG INTx	YES
CONFIG KBD	NO
CONFIG KEYBOARD	NO
CONFIG LCD	NO
CONFIG LCDBUS	NO
CONFIG LCDMODE	NO
CONFIG LCDPIN	NO
CONFIG RC5	NO
CONFIG PORT	YES
CONFIG SERIALIN	NO
CONFIG SERIALIN1	NO
CONFIG SERIALOUT	NO
CONFIG SERIALOUT1	NO
CONFIG SERVOS	NO
CONFIG PS2EMU	NO
CONFIG SDA	NO
CONFIG SCL	NO
CONFIG SPI	NO

CONFIG TCPIP	NO
CONFIG TIMER0	YES
CONFIG TIMER1	YES
CONFIG TIMER2 and 3	YES
CONFIG WATCHDOG	YES
CONFIG WAITSUART	NO
CONFIG X10	NO

Some CONFIG directives are intended to be specified once. Others can be used multiple times. For example you can specify that a port must be set to input after you have specified that it is used as an input.

You cannot change the LCD pins during run time. In that case the last specification will be used or an error message will be displayed.

CONFIG 1WIRE

Action

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

Syntax

CONFIG 1WIRE = pin

Remarks

Pin	The port pin to use such as PORTB.0

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

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

You can however use multiple pins and thus multiple busses. Al 1wire commands and functions need the port and pin in that case.

The 1wire commands and function will automatically set the DDR and PORT register bits to the proper state. You do not need to bring the pins into the right state yourself.

It is important that you use a pull up resistor of 4K7 ohm on the 1wire pin. The build in pull up resistor of the AVR is not sufficient.

Also notice that some 1wire chips also need +5V.

See also

1WRESET, 1WREAD, 1WWRITE

Example

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

CONFIG ACI

Action

Configures the Analog Comparator.

Syntax

CONFIG ACI = ON|OFF, **COMPARE** = ON|OFF, **TRIGGER**=TOGGLE|RISING|FALLING

Remarks

ACI	Can be switched on or off
COMPARE	Can be on or off. When switched ON, the TIMER1 in capture mode will trigger on ACI too.
TRIGGER	Specifies which comparator events trigger the analog comparator interrupts.

See also

NONE

Example

NONE

CONFIG ADC

Action

Configures the A/D converter.

Syntax

CONFIG ADC = single, PRESCALER = AUTO, REFERENCE = opt

Remarks

ADC	Running mode. May be SINGLE or FREE.
PRESCALER	A numeric constant for the clock divider. Use AUTO to let the compiler generate the best value depending on the XTAL
REFERENCE	Some chips like the M163 have additional reference options. Value may be OFF , AVCC or INTERNAL. See the data sheets for the different modes.

See also

GETADC

ASM

The following ASM is generated

In _temp1,ADCSR ; get settings of ADC

Ori _temp1, XXX ; or with settings

Out ADCSR,_temp1 ; write back to ADC register

Example

```
Config Adc = Single , Prescaler = Auto, Reference = Internal
```

CONFIG ATEMU

Action

Configures the PS/2 keyboard data and clock pins.

Syntax

CONFIG ATEMU= int , DATA = data, CLOCK=clock

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.
CLOCK	The pin that is connected to the CLOCK line.

Male (Plug)	Female (Socket)	5-pin DIN (AT/XT): 1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

Male (Plug)	Female (Socket)	6-pin Mini-DIN (PS/2): 1 - Data 2 - Not Implemented 3 - Ground 4 - +5v 5 - Clock 6 - Not Implemented

Old PC's are equipped with a 5 -pin DIN female connector. Newer PC's have a 6 -pin mini DIN female connector.

The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.

The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCANKBD statement allows you to send keyboard commands.

Note that unlike the mouse emulator, the keyboard emulator is also recognized after your PC has booted.

See also

SENDSCANKBD

Example

```
'--------------------------------------------------------------------------
---
' PS2_KBDEMUL.BAS
' (c) 2002-2004 MCS Electronics
' PS2 AT Keyboard emulator
'--------------------------------------------------------------------------
---
$regfile = "2313def.dat"
$crystal = 4000000
$baud = 19200

$lib "mcsbyteint.lbx" ' use optional lib since we use only bytes


'configure PS2 AT pins
Enable Interrupts ' you need to turn on interrupts yourself since an INT
is used
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
' ^------------------------ used interrupt
' ^----------- pin connected to DATA
' ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin


Waitms 500 ' optional delay

'rcall _AT_KBD_INIT
Print "Press t for test, and set focus to the editor window"
Dim Key2 As Byte , Key As Byte
Do
Key2 = Waitkey() ' get key from terminal
Select Case Key2
Case "t" :
Waitms 1500
Sendscankbd Mark ' send a scan code
Case Else
End Select
Loop
Print Hex(key)

Mark: ' send mark
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
&H42 , &HF0 , &H42
' ^ send 12 bytes
' m a r k
```

CONFIG CLOCK

Action

Configures the timer to be used for the TIME$ and DATE$ variables.

Syntax

CONFIG CLOCK = soft | USER [, GOSUB = SECTIC]

Remarks

Soft	Use SOFT for using the software based clock routines. Use USER to write/use your own code in combination with an I2C clock chip for example.
Sectic	This option allows to jump to a user routine with the label sectic. Since the interrupt occurs every second you may handle various tasks in the sectic label. It is important that you use the name SECTIC and that you return with a RETURN statement from this label. The usage of the optional SECTIC routine will use 30 bytes of the hardware stack. This option only works with the SOFT clock mode. It does not work in USER mode.

When you use the CONFIG CLOCK directive the compiler will DIM the following variables automatic : _sec , _min , _hour, _day , _month , _year

The variables TIME$ and DATE$ will also be dimensioned. These are special variables since they are treated different. See TIME$ and DATE$.

The _sec, _min and other internal variables can be changed by the user too.

But of course changing their values will change the DATE$/TIME$ variables.

The compiler also creates an ISR that gets updates once a second. This works only for the 8535, M163 and M103 and M603, or other AVR chips that have a timer that can work in asynchrony mode.

For the 8535, timer2 is used. It can not be used my the user anymore! This is also true for the other chips async timer.

Notice that you need to connect a 32768 Hz crystal in order to use the timer in async mode, the mode that is used for the clock timer.

When you choose the USER option, only the internal variables are created. With the USER option you need to write the clock code yourself.

See the datetime.bas example that shows how you can use a DS1307 clock chip for the date and time generation.

Numeric Values to calculate with Date and Time:

- ?????SecOfDay: (Type LONG) Seconds elapsed since Midnight. 00:00:00 start with 0 to 85399 at 23:59:59.
- ?????SysSec: (Type LONG) Seconds elapsed since begin of century (at 2000-01-01!). 00:00:00at2000-01-01 start with 0 to 2147483647 (overflow of LONG-Type) at 2068-01-19 03:14:07
- ?????DayOfYear: (Type WORD) Days elapsed since first January of the current year.
  First January start with 0 to 364 (365 in a leap year)
- ?????SysDay: (Type WORD) Days elapsed since begin of century (at 2000-01-01!). 2000-01-01 starts with 0 to 36524 at 2099-12-31
- ?????DayOfWeek: (Type Byte) Days elapsed since Monday of current week. Monday start with 0 to Sunday = 6

With the numeric type calculations with Time and date are possible. Type 1 (discrete Bytes) and 2 (Strings) can be converted to an according numeric value. Than Seconds (at SecOfDay and SysSec) or Days (at DayOfYear, SysDay), can be added or subtracted. The Result can be converted back.

See also

TIME$, DATE$, CONFIG DATE

ASM

The following ASM routines are called from datetime.lib

_soft_clock. This is the ISR that gets called once per second.

Example

```
'-----------------------------------------------------------
' MEGACLOCK.BAS
' (c) 2000-2004 MCS Electronics
'-----------------------------------------------------------
'This example shows the new TIME$ and DATE$ reserved variables
'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000 'address for E and RS
$lcdrs = &H8000 'address for only E
Config Lcd = 20 * 4 'nice display from bg micro
Config Lcdbus = 4 'we run it in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus 'tell about the bus mode

'[now init the clock]
Config Date = Mdy , Separator = / ' ANSI-Format

Config Clock = Soft 'this is how simple it is
'The above statement will bind in an ISR so you can not use the TIMER anymore!
'For the M103 in this case it means that TIMER0 can not be used by the user anymore

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'---------------------------------------------------

'clear the LCD display
Cls

Do
Home 'cursor home
Lcd Date$ ; " " ; Time$ 'show the date and time
Loop

'The clock routine does use the following internal variables:
```

```
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

CONFIG COM1

Action

Configures the UART of AVR chips that have an extended UART like the M8.

Syntax

CONFIG COM1 =dummy , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|2,databits=4|6|7|8|9,clockpol=0|1

Remarks

synchrone	0 for synchrone operation (default) and 1 for asynchrone operation.
Parity	None, disabled, even or odd
Stopbits	The number of stopbits : 1 or 2
Databits	The number of databits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

Note that not all AVR chips have the extended UART.

CONFIG COM2

Action

Configures the UART of AVR chips that have a second extended UART like the M128.

Syntax

CONFIG COM2 =dummy , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|2,databits=4|6|7|8|9,clockpol=0|1

Remarks

synchrone	0 for synchrone operation (default) and 1 for asynchrone operation.
Parity	None, disabled, even or odd
Stopbits	The number of stopbits : 1 or 2
Databits	The number of databits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

Note that not all AVR chips have two extended UARTS.

CONFIG DATE

Action

Configure the Format of the Date String for Input to and Output from BASCOM – Date functions

Syntax

CONFIG DATE = DMY , Separator = char

Remarks

DMY	The Day, month and year order. Use DMY, MDY or YMD.
Char	A character used to separate the day, month and year. Use / , - or . (dot)

The next table shows the common formats of date and the associated statements.

Country	**Format**	**Statement**
American	mm/dd/yy	Config Date = MDY, Separator = /
ANSI	yy.mm.dd	Config Date = YMD, Separator = .
Britisch/French	dd/mm/yy	Config Date = DMY, Separator = /
German	dd.mm.yy	Config Date = DMY, Separator = .
Italian	dd-mm-yy	Config Date = DMY, Separator = -
Japan/Taiwan	yy/mm/dd	Config Date = YMD, Separator = /
USA	mm-dd-yy	Config Date = MDY, Separator = -

When you live in Holland you would use :

CONFIG DATE = DMY, separator =-

This would print 24-04-02 for 24 November 2002.

When you line in the US, you would use :

CONFIG DATE = MDY , separator = /

This would print 04/24/02 for 24 November 2002.

See also

CONFIG CLOCK , DATE TIME functions , DayOfWeek , DayOfYear , SecOfDay , SecElapsed , SysDay

SysSec , SysSecElapsed , Time , Date

Example

```
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format
```

CONFIG DEBOUNCE

Action

Configures the delay time for the DEBOUNCE statement.

Syntax

CONFIG DEBOUNCE = time

Remarks

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

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

See also

DEBOUNCE

Example

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


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

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

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

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

CONFIG I2CDELAY

Action

Compiler directive that overrides the internal I2C delay routine.

Syntax

CONFIG I2CDELAY =value

Remarks

value	A numeric value in the range from 1 to 255. A higher value means a slower I2C clock.

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

By default a value of 5 is used. This will give a 200 kHZ clock.

When you specify 10, 10 uS will be used resulting in a 100 KHz clock.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

CONFIG SCL , CONFIG SDA

Example

```
CONFIG SDA = PORTB.7  'PORTB.7 is the SDA line
CONFIG I2CDELAY = 5
See I2C example for more details.
'--------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'--------------------------------------------------------------
' file: I2C.BAS
' demo: I2CSEND and I2CRECEIVE
'--------------------------------------------------------------
Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)

Const Addressw = 174 'slave write address
Const Addressr = 175 'slave read address

Dim B1 As Byte , Adres As Byte , Value As Byte 'dim byte

Call Write_eeprom(1 , 3) 'write value of three to address 1 of EEPROM


Call Read_eeprom(1 , Value) : Print Value 'read it back
Call Read_eeprom(5 , Value) : Print Value 'again for address 5


'-------- now write to a PCF8474 I/O expander -------
I2csend &H40 , 255 'all outputs high
I2creceive &H40 , B1 'retrieve input
Print "Received data " ; B1 'print it
End

Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.



'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
I2cstart 'start condition
I2cwbyte Addressw 'slave address
I2cwbyte Adres 'asdress of EEPROM
I2cwbyte Value 'value to write
I2cstop 'stop condition
Waitms 10 'wait for 10 milliseconds
End Sub


'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
I2cstart 'generate start
I2cwbyte Addressw 'slave adsress
I2cwbyte Adres 'address of EEPROM
I2cstart 'repeated start
I2cwbyte Addressr 'slave address (read)
I2crbyte Value , Nack 'read byte
I2cstop 'generate stop
End Sub
```

CONFIG I2CSLAVE

Action

Configures the I2C slave mode.

Syntax

CONFIG I2CSLAVE = address , **INT =** interrupt , **TIMER =** tmr

\

Remarks

Address	The slave address you want to assign to the I2C slave chip. This is an address that must be even like 60. So 61 cannot be used.
Interrupt	The interrupt that must be used. This is INT0 by default.
Tmr	The timer that must be used. This is TIMER0 by default.

While the interrupt can be specified, you need to change the library code when you use a non-default interrupt. For example when you like to use INT1 instead of the default INT0.

The same applies to the TIMER. You need to change the library when you like to use another timer.

Example

```
'---------------------------------------------------------------
'-------------------------------------------------------------------------
--
' I2C SLAVE LIBRARY DEMO
' PCF8574 emulator
' (c) 2002 MCS Electronics
'-------------------------------------------------------------------------
--
'This program shows how you could use the I2C slave library to create a
PCF8574
'The PCF8574 is an IO extender chip that has 8 pins.
'The pins can be set to a logic level by writing the address followed by
a value
'In order to read from the pins you need to make them '1' first

'This program uses a AT90S2313, PORTB is used as the PCF8574 PORT
'The slave library needs INT0 and TIMER0 in order to work.
'SCL is PORTD.4 (T0)
'SDA is PORTD.2 (INT0)
'Use 10K pull up resistors for both SCL and SDA

'The Slave library will only work for chips that have T0 and INT0
connected to the same PORT.
'These chips are : 2313,2323, 2333,2343,4433,tiny22, tiny12,tiny15, M8
'The other chips have build in hardware I2C(slave) support.



'specify the XTAL connected to the chip
$crystal = 3684000

'specify the used chip
$regfile = "2313def.dat"

'specify the slave address. This is &H40 for the PCF8574
'You always need to specify the address used for write. In this case &H40
'

'The config i2cslave command will enable the global interrupt enable flag
!
Config I2cslave = &B01000000 ' same as &H40

'A byte named _i2c_slave_address_received is generated by the compiler.
'This byte will hold the received address.

'A byte named _i2c_slave_address is generated by the compiler.
'This byte must be assigned with the slave address of your choice

'the following constants will be created that are used by the slave
library:

' _i2c_pinmask = &H14
' _i2c_slave_port = Portd
' _i2c_slave_pin = Pind
' _i2c_slave_ddr = Ddrd
' _i2c_slave_scl = 4
' _i2c_slave_sda = 2

'These values are adjusted automatic depending on the selected chip.
'You do not need to worry about it, only provided as additional info

'by default the PCF8574 port is set to input
Config Portb = Input
Portb = 255 'all pins high by default

'DIM a byte that is not needed but shows how you can store/write the I2C
DATA
Dim Bfake As Byte


'empty loop
Do
' you could put your other program code here
'In any case, do not use END since it will disable interrupts

Loop


'here you can write your other program code
'But do not forget, do not use END. Use STOP when needed

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!
' The following labels are called from the slave library
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!

'When the master wants to read a byte, the following label is always
called
'You must put the data you want to send to the master in variable _a1
```

```
which is register R16
I2c_master_needs_data:
'when your code is short, you need to put in a waitms statement
'Take in mind that during this routine, a wait state is active and the
master will wait
'After the return, the waitstate is ended
Config Portb = Input ' make it an input
_a1 = Pinb ' Get input from portB and assign it
Return


'When the master writes a byte, the following label is allways called
'It is your task to retrieve variable _A1 and do something with it
'_A1 is register R16 that could be destroyed/altered by BASIC statements
'For that reason it is important that you first save this variable

I2c_master_has_data:
'when your code is short, you need to put in a waitms statement
'Take in mind that during this routine, a wait state is active and the
master will wait
'After the return, the waitstate is ended

Bfake = _a1 ' this is not needed but it shows how you can store _A1 in a
byte
'after you have stored the received data into bFake, you can alter R16
Config Portb = Output ' make it an output since it could be an input
Portb = _a1 'assign _A1 (R16)
Return


'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

'You could simply extend this sample so it will use 3 pins of PORT D for
the address selection
'For example portD.1 , portd.2 and portD.3 could be used for the address
selection
'Then after the CONFIG I2CSLAVE = &H40 statement, you can put code like:
'Dim switches as Byte ' dim byte
'switches = PIND ' get dip switch value
'switches = switches and &H1110 ' we only need the lower nibble without
'_i2c_slave_address = &H40 + switches ' set the proper address
```

CONFIG INTx

Action

Configures the way the interrupts 0,1 and 4-7 will be triggered.

Syntax

CONFIG INTx = state

Where X can be 0,1 and 4 to 7 in the MEGA chips.

Remarks

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

The MEGA has also INT0-INT3. These are always low level triggered so there is no need /possibility for configuration.

The number of interrupt pins depend on the used chip. Most chips only have int0 and int1.

Example

```
'--------------------------------------------------------------
'Sample for the MEGA103
Config INT4 = LOW LEVEL


End
```

CONFIG GRAPHLCD

Action

Configures the Graphical LCD display.

Syntax

Config GRAPHLCD = type , **DATAPORT** = port, **CONTROLPORT=**port , **CE** = pin , **CD** = pin , WR = pin, RD=pin, RESET= pin, FS=pin, **MODE** = mode

Remarks

Type	This must be 240 * 64, 128* 128, 128 * 64 , 160 * 48 or 240 * 128. For SED displays use 128 * 64sed or 120* 64SED
Dataport	This is the name of the port that is used to put the data on the LCD data pins db0-db7. PORTA for example.
Controlport	This is the name of the port that is used to control the LCD control pins. PORTC for example
Ce	The pin number that is used to enable the chip on the LCD.
Cd	The pin number that is used to control the CD pin of the display.
WR	The pin number that is used to control the /WR pin of the display.
RD	The pin number that is used to control the /RD pin of the display.
FS	The pin number that is used to control the FS pin of the display. Not needed for SED based displays.
RESET	The pin number that is used to control the RESET pin of the display.
MODE	The number of columns for use as text display. Use 8 for X-pixels / 8 = 30 columns for a 240 pixel screen. When you specify 6, 240 / 6 = 40 columns can be used.

This is a first implementation for Graphic support. It is based on the T6963C chip that is used in many displays. At the moment there is only support for pin mode. That is, the LCD is controlled by changing logic levels on the pins.

Memory mapped or bus mode will be added later. But pin mode can be used with any micro so that is why this is first implemented.

The following connections were used:

PORTA.0 to PORTA.7 to DB0-DB7 of the LCD

PORTC.5 to FS, font select of LCD

PORTC.2 to CE, chip enable of LCD

PORTC.3 to CD, code/data select of LCD

PORTC.0 to WR of LCD, write

PORTC.1 to RD of LCD, read

PORTC.4 to RESET of LCD, reset LCD

The LCD used from www.conrad.de needs a negative voltage for the contrast.

Two 9V batteries were used with a pot meter.

Some displays have a Vout that can be used for the contrast(Vo)

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

LOCATE row,column Will place the cursor at the specified row and column

The row may vary from 1 to 16 and the column from 1 to 40. This depends on the size and mode of the display.

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.

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

LINE(x0,y0) – (x1,y1) , color Will draw a line from the coordinate x0,y0 to x1,y1.

Color must be 0 to clear the line and 255 for a black line.

The Graphic routines are located in the glib.lib or glib.lbx files.

You can hard wire the FS and RESET and change the code from the glib.lib file so these pins can be used for other tasks.

See also

SHOWPIC , PSET , $BGF , LINE , LCD

Example

```
'-----------------------------------------------------------------
' (c) 2001 MCS Electronics
' T6963C graphic display support demo
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin connected to
```

```
' 1 GND GND
'2 GND GND
'3 +5V +5V
'4 -9V -9V potmeter
'5 /WR PORTC.0
'6 /RD PORTC.1
'7 /CE PORTC.2
'8 C/D PORTC.3
'9 NC not conneted
'10 RESET PORTC.4
'11-18 D0-D7 PA
'19 FS PORTC.5
'20 NC not connected


'First we define that we use a graphic LCD
Config Graphlcd = 240 * 64 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2


'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part

'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"

'wait 1 sec
Wait 1


' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
Pset X , 20 , 255 ' set the pixel
Next

Wait 1


'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje

Wait 1
Cls Text ' clear the text
End



'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

CONFIG KBD

Action

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

Syntax

CONFIG KBD = PORTx , DEBOUNCE = value [, DELAY = value]

Remarks

PORTx	The name of the PORT to use such as PORTB or PORTD.
DEBOUNCE	By default the debounce value is 20. A higher value might be needed. The maximum is 255.
Delay	An optional parameter that will cause Getkbd() to wait the specified amount of time after the key is detected. This parameter might be added when you call GetKbd() repeatedly in a loop. Because of noise and static electricity, wrong values can be returned. A delau of say 100 mS, can eliminate this problem.

The GETKBD() function can be used to read the pressed key from a matrix keypad attached to a port of the uP.

You can define the port with the CONFIG KBD statement.

In addition to the default behavior you can configure the keyboard to have 6 rows instead of 4 rows.

CONFIG KBD = PORTx , DEBOUNCE = value , **rows=6, row5=pinD.6, row6=pind.7**

This would specify that row5 is connected to pind.6 and row7 to pind.7

Note that you can only use rows=6. Other values will not work.

See also

GETKBD

CONFIG KEYBOARD

Action

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

Syntax

CONFIG KEYBOARD = PINX.y , DATA = PINX.y , KEYDATA = table

Remarks

KEYBOARD	The PIN that serves as the CLOCK input.
DATA	The PIN that serves as the DATA input.
KEYDATA	The label where the key translation can be found. The AT keyboard returns scan codes instead of normal ASCII codes. So a translation table s needed to convert the keys. BASCOM allows the use of shifted keys too. Special keys like function keys are not supported.

The AT keyboard can be connected with only 4 wires: clock,data, gnd and vcc.

Some info is displayed below. This is copied from an Atmel datasheet.

The INT0 or INT1 shown can be in fact any pin that can serve as an INPUT pin.

The application note from Atmel works in interrupt mode. For BASCOM I rewrote the code so that no interrupt is needed/used.

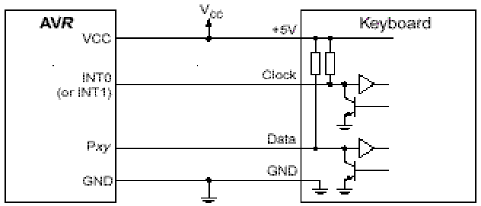


Table 1. AT Keyboard Connector Pin Assignments

AT Computer		
Signals	**DIN41524, Female at Computer, 5-pin DIN 180°**	**6-pin Mini DIN PS2 Style Female at Computer**
Clock	1	5
Data	2	1
nc	3	2,6
GND	4	3
+5V	5	4
Shield	Shell	Shell

See also

GETATKBD

CONFIG LCD

Action

Configure the LCD display and override the compiler setting.

Syntax

CONFIG LCD = LCDtype

Remarks

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

When you have a 16 * 2 display, you don't have to use this statement.

The 16 * 1a is special. It is used for 2 * 8 displays that have the address of line 2, starting at location &H8.

The 20*4A is also special. It uses the addresses &H00, &H20, &H40 and &H60 for the 4 lines.

Example

```
Config Lcd = 40 * 4
Lcd "Hello" 'display on LCD
Fourthline 'select line 4
Lcd "4" 'display 4
End
```

CONFIG LCDBUS

Action

Configures the LCD data bus and overrides the compiler setting.

Syntax

CONFIG LCDBUS = constant

Remarks

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

Use this statement together with the $LCD = address statement.

When you use the LCD display in the bus mode the default is to connect all the data lines. With the 4-bit mode, you only have to connect data lines d7-d4.

See also

CONFIG LCD

Example

```
$lcd = &HC000 'address of enable and RS signal
$lcdrs = &H800 'address of enable signal
Config Lcdbus = 4 '4 bit mode
Lcd "hello"
```

CONFIG LCDMODE

Action

Configures the LCD operation mode and overrides the compiler setting.

Syntax

CONFIG LCDMODE = type

Remarks

Type	**PORT** will drive the LCD in 4-bit port mode and is the default. In PORT mode you can choose different PIN's from different PORT's to connect to the upper 4 data lines of the LCD display. The RS and E can also be connected to a user selectable pin. This is very flexible since you can use pins that are not used by your design and makes the board layout simple. On the other hand, more software is necessary to drive the pins. **BUS** will drive the LCD in bus mode and in this mode is meant when you have external RAM and so have an address and data bus on your system. The RS and E line of the LCD display can be connected to an address decoder. Simply writing to an external memory location select the LCD and the data is sent to the LCD display. This means the data-lines of the LCD display are fixed to the data-bus lines. Use $LCD = address and $LCDRS = address, to specify the addresses that will enable the E and RS lines.

See also

CONFIG LCD , $LCD , $LCDRS

Example

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

CONFIG LCDPIN

Action

Override the LCD-PIN select options.

Syntax

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

CONFIG LCDPIN = PIN , PORT=PORTx, E=PN, RS=PN

Remarks

PN	The name of the PORT pin such as PORTB.2 for example.
PORTX	When you want to use the LCD in 8 bit data, pin mode, you must specify the PORT to use.

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

The config command is preferred over the menu settings since the code makes clear which pins are used. The CONFIG statement overrides the Options setting.

See also

CONFIGLCD

Example

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

The above example must be typed on one line.

CONFIG PS2EMU

Action

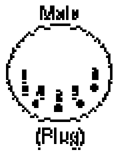
Configures the PS2 mouse data and clock pins.

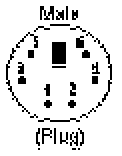
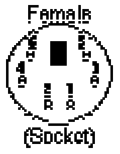
Syntax

CONFIG PS2EMU= int , DATA = data, CLOCK=clock

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.
CLOCK	The pin that is connected to the CLOCK line.

		5-pin DIN (AT/XT): 1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

		6-pin Mini-DIN (PS/2): 1 - Data 2 - Not Implemented 3 - Ground 4 - +5v 5 - Clock 6 - Not Implemented

Old PC's are equipped with a 5 -pin DIN female connector. Newer PC's have a 6 -pin mini DIN female connector.

The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.

The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCAN and PS2MOUSEXY statements allow you to send mouse commands.

Note that the mouse emulator is only recognized after you have booted your PC. Mouse devices can not be plugged into your PC once it has booted. Inserting a mouse or mouse device when the PC is already booted, may damage your PC.

See also

SENDSCAN, PS2MOUSEXY

Example

```
'--------------------------------------------------------------------------
-
' PS2_EMUL.BAS
' (c) 2003-2004 MCS Electronics
' PS2 Mouse emulator
'--------------------------------------------------------------------------
-
$regfile = "2313def.dat"
$crystal = 4000000
$baud = 19200

$lib "mcsbyteint.lbx" ' use optional lib since we use only bytes


'configure PS2 pins
Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0
'  ^------------------------ used interrupt
'  ^----------- pin connected to DATA
'  ^- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin


Waitms 500 ' optional delay

Enable Interrupts ' you need to turn on interrupts yourself since an INT
is used

Print "Press u,d,l,r,b, or t"
Dim Key As Byte
Do
Key = Waitkey() ' get key from terminal
Select Case Key
Case "u" : Ps2mousexy 0 , 10 , 0 ' up
Case "d" : Ps2mousexy 0 , -10 , 0 ' down
Case "l" : Ps2mousexy -10 , 0 , 0 ' left
Case "r" : Ps2mousexy 10 , 0 , 0 ' right
Case "b" : Ps2mousexy 0 , 0 , 1 ' left button pressed
Ps2mousexy 0 , 0 , 0 ' left button released
Case "t" : Sendscan Mouseup ' send a scan code
Case Else
End Select
Loop
```

```
Mouseup:
Data 3 , &H08 , &H00 , &H01 ' mouse up by 1 unit
```

CONFIG RC5

Action

Overrides the RC5 pin assignment from the Option Compiler Settings.

Syntax

CONFIG RC5 = pin [,TIMER=2]

Remarks

Pin	The port pin to which the RC5 receiver is connected.
TIMER	Must be 2. The micro must have a timer2 when you want to use this option. This additional parameter will cause that TIMER2 will be used instead of the default TIMER0.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the RC5 pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project.CFG file.

See also

GETRC5

Example

```
CONFIG RC5 = PIND.5  'PORTD.5 is the RC5 input line
```

CONFIG SDA

Action

Overrides the SDA pin assignment from the Option Compiler 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. In BASCOM-AVR the settings are also stored in the project.CFG file.

See also

CONFIG SCL , CONFIG I2CDELAY

Example

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

CONFIG SCL

Action

Overrides the SCL pin assignment from the Option Compiler 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. Of course BASCOM-AVR also stores the settings in a project.CFG file.

See also

CONFIG SDA, CONFIG I2CDELAY

Example

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

CONFIG SERIALIN

Action

Configures the hardware UART to use a buffer for input

Syntax

CONFIG SERIALIN = BUFFERED , SIZE = size

Remarks

size	A numeric constant that specifies how large the input buffer should be. The space is taken from the SRAM.

The following internal variables will be generated :

_RS_HEAD_PTR0 , a byte counter that stores the head of the buffer

_RS_TAIL_PTR0 , a byte counter that stores the tail of the buffer.

_RS232INBUF0 , an array of bytes that serves as a ring buffer for the received characters.

ASM

Routines called from MCS.LIB :

_GotChar. This is an ISR that gets called when ever a character is received.

When there is no room for the data it will not be stored.

So the buffer must be emptied periodic by reading from the serial port using the normal statements like INKEY() and INPUT.

Since URXC interrupt is used by _GotChar, you can not use this interrupt anymore. Unless you modify the _gotchar routine of course.

See also

CONFIG SERIALOUT

Example

```
'--------------------------------------------------------------
' RS232BUFFER.BAS
' (c) 2000-2004, MCS Electronics
' This example shows the difference between normal and buffered
' serial INPUT
'--------------------------------------------------------------
$crystal = 4000000
$baud = 9600

'first compile and run this program with the line below remarked
Config Serialin = Buffered , Size = 20


'dim a variable
Dim Name As String * 10

'the enabling of interrupts is not needed for the normal serial mode
'So the line below must be remarked to for the first test
Enable Interrupts
```

```
Print "Start"
Do
'get a char from the UART
Name = Inkey()

If Err = 0 Then 'was there a char?
Print Name 'print it
End If

Wait 1 'wait 1 second
Loop

'You will see that when you slowly enter characters in the terminal
emulator
'they will be received/displayed.
'When you enter them fast you will see that you loose some chars

'NOW remove the remarks from line 11 and 18
'and compile and program and run again
'This time the chars are received by an interrupt routine and are
'stored in a buffer. This way you will not loose characters providing
that
'you empty the buffer
'So when you fast type abcdefg, they will be printed after each other
with the
'1 second delay

'Using the CONFIG SERIAL=BUFFERED, SIZE = 10 for example will
'use some SRAM memory
'The following internal variables will be generated :
'_Rs_head_ptr0 BYTE , a pointer to the location of the start of the
buffer
'_Rs_tail_ptr0 BYTE , a pointer to the location of tail of the buffer
'_RS232INBUF0 BYTE ARRAY , the actual buffer with the size of SIZE
```

CONFIG SERIALIN1

Action

Configures the second hardware UART to use a buffer for input

Syntax

CONFIG SERIALIN1 = BUFFERED , SIZE = size

Remarks

Size	A numeric constant that specifies how large the input buffer should be. The space is taken from the SRAM.

The following internal variables will be generated :

_RS_HEAD_PTR1 , a byte counter that stores the head of the buffer

_RS_TAIL_PTR1 , a byte counter that stores the tail of the buffer.

_RS232INBUF1 , an array of bytes that serves as a ring buffer for the received characters.

ASM

Routines called from MCS.LIB :

_GotChar1. This is an ISR that gets called when ever a character is received.

When there is no room for the data it will not be stored.

So the buffer must be emptied periodic by reading from the serial port using the normal statements like INKEY() and INPUT.

Since URXC1 interrupt is used by _GotChar1, you can not use this interrupt anymore. Unless you modify the _gotchar1 routine of course.

See also

CONFIG SERIALOUT1

Example

```
'-----------------------------------------------------------------
' RS232BUFFER1.BAS
' (c) 2000-2004, MCS Electronics
' This example shows the difference between normal and buffered
' serial INPUT
' Works only for chips with 2 UARTS
'-----------------------------------------------------------------
$regfile = "m161def.dat"
$crystal = 4000000
$baud = 9600

'first compile and run this program with the line below remarked
Config Serialin1 = Buffered , Size = 20


'dim a variable
Dim Name As String * 10

Open "com2:" For Binary As #1
```

```
'the enabling of interrupts is not needed for the normal serial mode
'So the line below must be remarked to for the first test
Enable Interrupts

Print "Start"
Do
'get a char from the UART
Name = Inkey(#1)

If Err = 0 Then 'was there a char?
Print #1 , Name 'print it
End If

Wait 1 'wait 1 second
Loop
Close #1

'You will see that when you slowly enter characters in the terminal
emulator
'they will be received/displayed.
'When you enter them fast you will see that you loose some chars

'NOW remove the remarks from line 11 and 18
'and compile and program and run again
'This time the chars are received by an interrupt routine and are
'stored in a buffer. This way you will not loose characters providing
that
'you empty the buffer
'So when you fast type abcdefg, they will be printed after each other
with the
'1 second delay

'Using the CONFIG SERIAL1=BUFFERED, SIZE = 10 for example will
'use some SRAM memory
'The following internal variables will be generated :
'_Rs_head_ptr1 BYTE , a pointer to the location of the start of the
buffer
'_Rs_tail_ptr1 BYTE , a pointer to the location of tail of the buffer
'_RS232INBUF1 BYTE ARRAY , the actual buffer with the size of SIZE
```

CONFIG SERIALOUT

Action

Configures the hardware UART to use a buffer for output

Syntax

CONFIG SERIALOUT = BUFFERED , SIZE = size

Remarks

size	A numeric constant that specifies how large the output buffer should be. The space is taken from the SRAM.

The following internal variables will be used when you use CONFIG SERIALOUT

_RS_HEAD_PTRW0 , byte that stores the head of the buffer

_RS_TAIL_PTRW0 , byte that stores the tail of the buffer

_RS232OUTBUF0, array of bytes for the ring buffer that stores the printed data.

ASM

Routines called from MCS.LIB :

_CHECKSENDCHAR. This is an ISR that gets called when ever the transmission buffer is empty.

Since UDRE interrupt is used , you can not use this interrupt anymore. Unless you modify the _CheckSendChar routine of course.

When you use the PRINT statement to send data to the serial port, the UDRE interrupt will be enabled. And so the _CheckSendChar routine will send the data from the buffer.

See also

CONFIG SERIALIN

Example

```
'-----------------------------------------------------------------------
----
' RS232BUFFEROUT.BAS
' (c) 2000-2004 MCS Electronics
'Sample demonstrates how to use a serial output buffer
'-----------------------------------------------------------------------
----
$baud = 9600
$crystal = 4000000

'setup to use a serial output buffer
'and reserve 20 bytes for the buffer
Config Serialout = Buffered , Size = 20

'It is important since UDRE interrupt is used that you enable the
interrupts
Enable Interrupts
Print "Hello world"
Do
```

```
Wait 1
'notice that using the UDRE interrupt will slown down execution of
waiting loops like waitms
Print "test"
Loop
End
```

CONFIG SERIALOUT1

Action

Configures the second hardware UART to use a buffer for output

Syntax

CONFIG SERIALOUT1 = BUFFERED , SIZE = size

Remarks

Size	A numeric constant that specifies how large the output buffer should be. The space is taken from the SRAM.

The following internal variables will be used when you use CONFIG SERIALOUT

_RS_HEAD_PTRW1 , byte that stores the head of the buffer

_RS_TAIL_PTRW1 , byte that stores the tail of the buffer

_RS232OUTBUF1, array of bytes for the ring buffer that stores the printed data.

ASM

Routines called from MCS.LIB :

_CHECKSENDCHAR1. This is an ISR that gets called when ever the transmission buffer is empty.

Since UDRE1 interrupt is used , you can not use this interrupt anymore. Unless you modify the _CheckSendChar1 routine of course.

When you use the PRINT statement to send data to the serial port, the UDRE1 interrupt will be enabled. And so the _CheckSendChar1 routine will send the data from the buffer.

See also

CONFIG SERIALIN1

Example

```
'------------------------------------------------------------------------
----
' RS232BUFFEROUT1.BAS
' (c) 2000-2004 MCS Electronics
'Sample demonstrates how to use a serial output buffer on the second UART
'this sample will only work for chips with a seond UART like the M161 and
M128
'------------------------------------------------------------------------
----
$regfile = "m161def.dat"
$baud1 = 9600
$crystal = 4000000

'setup to use a serial output buffer
'and reserve 20 bytes for the buffer
Config Serialout1 = Buffered , Size = 20
Open "Com2:" For Binary As #1

'It is important since UDRE interrupt is used that you enable the
```

```
interrupts
Enable Interrupts
Print #1 , "Hello world"
Do
Wait 1
'notice that using the UDRE interrupt will slown down execution of
waiting loops like waitms
Print #1 , "test"
Loop
End


Close #1
```

CONFIG SPI

Action

Configures the SPI related statements.

Syntax for software SPI

CONFIG SPI = SOFT, DIN = PIN, DOUT = PIN , SS = PIN|**NONE**, CLOCK = PIN

Syntax for hardware SPI

CONFIG SPI = HARD, DINTERRUPT=ON|OFF, ATA ORDER = LSB|MSB , MASTER = YES|NO , POLARITY = HIGH|LOW , PHASE = 0|1, CLOCKRATE = 4|16|64|128 , NOSS=1|0

Remarks

SPI	**SOFT** for software emulation of SPI, this allows you to choose the PINS to use. Only works in master mode. **HARD** for the internal SPI hardware, that will use fixed pins of the microprocessor.
DIN	Data input or MISO. Pin is the pin number to use such as PINB.0
DOUT	Data output or MOSI. Pin is the pin number to use such as PORTB.1
SS	Slave Select. Pin is the pin number to use such as PORTB.2 Use NONE when you do not want the SS signal to be generated. See remarks
CLOCK	Clock. Pin is the pin number to use such as PORTB.3
DATA ORDER	Selects if MSB or LSB is transferred first.
MASTER	Selects if the SPI is run in master or slave mode.
POLARITY	Select HIGH to make the CLOCK line high while the SPI is idle. LOW will make clock LOW while idle.
PHASE	Refer to a data sheet to learn about the different settings in combination with polarity.
CLOCKRATE	The clock rate selects the division of the of the oscillator frequency that serves as the SPI clock. So with 4 you will have a clockrate of 4.000000 / 4 = 1 MHz , when a 4 MHZ XTAL is used.
NOSS	1 or 0. Use 1 when you do not want the SS signal to be generated in master mode.
INTERRUPT	Specify ON or OFF. ON will enable the SPI interrupts to occur. While OFF disables SPI interrupts. ENABLE SPI and DISABLE SPI will accomplish the same.

The default setting for hardware SPI when set from the Compiler, Options, SPI menu is MSB first, POLARITY = HIGH, MASTER = YES, PHASE = 0, CLOCKRATE = 4

When you use CONFIG SPI = HARD alone without the other parameters, the SPI will only be enabled. It will work in slave mode then with CPOL =0 and CPH=0.

In hardware mode the SPIINIT statement will set the SPI pins to :

sbi DDRB,7 ; SCK output

cbi DDRB,6 ; MISO input
sbi DDRB,5 ; MOSI output

In softmode the SPIINIT statement will set the SPI pins for example to :

sbi PORTB,5 ;set latch bit hi (inactive)SS
sbi DDRB,5 ;make it an output SS
cbi PORTB,4 ;set clk line lo
sbi DDRB,4 ;make it an output
cbi PORTB,6 ;set data-out lo MOSI
sbi DDRB,6 ;make it an output MOSI
cbi DDRB,7 ;MISO input
Ret

When you want to address multiple slaves with the software SPI you need multiple pins to select/activate the slave chip. Specify NONE for SS in that case. This also means that before every SPI command you need to set the logic level to 0 to address the chip and after the SPI command you need to set it back to a logic high level.

The hardware SPI also has this option. The NOSS parameter with a value of 1, will not set the SS line to logic 0 when the SPI operation begins. You need to set SS or any other pin of your choice to a logic 0 yourself. After the SPI command(s) are used you need to set it back to a logic 1 to deselect the slave chip.

All SPI routines are SPI-master routines. Example 2 below demonstrates how to create a soft SPI slave. In the samples directory you will also find a SPI hardware master and SPI hardware slave sample.

See also

SPIIN , SPIOUT , SPIINIT , SPI

Example

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

Example2

```
'-----------------------------------------------------------------------
-------
' SPI-SOFTSLAVE.BAS
' (c) 2004 MCS Electronics
' sample that shows how to implement a SPI SLAVE with software
'-----------------------------------------------------------------------
-------
'Some atmel chips like the 2313 do not have a SPI port.
'The BASCOM SPI routines are all master mode routines
'This example show how to create a slave using the 2313
'ISP slave code

'we use the 2313
$regfile = "2313def.dat"

'XTAL used
$crystal = 4000000

'baud rate
$baud = 19200

'define the constants used by the SPI slave
Const _softslavespi_port = Portd ' we used portD
Const _softslavespi_pin = Pind 'we use the PIND register for reading
Const _softslavespi_ddr = Ddrd ' data direction of port D

Const _softslavespi_clock = 5 'pD.5 is used for the CLOCK
Const _softslavespi_miso = 3 'pD.3 is MISO
Const _softslavespi_mosi = 4 'pd.4 is MOSI
Const _softslavespi_ss = 2 ' pd.2 is SS
'while you may choose all pins you must use the INT0 pin for the SS
'for the 2313 this is pin 2

'PD.3(7), MISO must be output
'PD.4(8), MOSI
'Pd.5(9) , Clock
'PD.2(6), SS /INT0

'define the spi slave lib
$lib "spislave.lbx"
'sepcify wich routine to use
$external _spisoftslave

'we use the int0 interrupt to detect that our slave is addressed
On Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 interrupt to trigger when a falling edge is
detected
Config Int0 = Falling
'finally we enabled interrupts
Enable Interrupts

'
Dim _ssspdr As Byte ' this is out SPI SLAVE SPDR register
Dim _ssspif As Bit ' SPI interrupt revceive bit
Dim Bsend As Byte , I As Byte , B As Byte ' some other demo variables

_ssspdr = 0 ' we send a 0 the first time the master sends data
Do
If _ssspif = 1 Then
Print "received: " ; _ssspdr
Reset _ssspif
_ssspdr = _ssspdr + 1 ' we send this the next time
End If
Loop
```

CONFIG SERVOS

Action

Configures how much servo's will be controlled.

Syntax

CONFIG SERVOS = X , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = rl

Remarks

Servo's need a variable pulse in order to operate. The CONFIG SERVOS directive will se up a byte array with the servo pulse width values and will initialize an ISR that uses TIMER0.

X	The number of servo's you want to control. Each used servo will use one byte of SRAM.
PORT	The port pin the servo is attached too.
RL	The reload value for the ISR in uS.

When you use for example :

Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10

The internal ISR will execute every 10 uS.

An arrays named SERVO() will be created and it can hold 2 bytes : servo(1) and servo(2).

By setting the value of the servo() array you control how long the positive pulse will last. After it has reached this value it will be reset to 0.

The reload value should be set to 10. After 20 mS, a new pulse will be generated.

You can use other reload values but it will also mean that the repeat value will change.

The PORT pins specified must be set to work as an output pin by the user.

CONFIG PINB.0 = OUTPUT

Will set a pin to output mode.

Resources used

TIMER0 is used to create the ISR.

ASM

NONE

Example

```
'--------------------------------------------------
' (c) 2001 MCS Electronics
' servo.bas demonstrates the SERVO option
'---------------------------------------------------

'Servo's need a pulse in order to operate
'with the config statement CONFIG SERVOS we can specify how many servo's
we
'will use and which port pins are used
'A maximum of 16 servos might be used
'The SERVO statements use one byte for an interrupt counter and the
TIMER0
'This means that you can not use TIMER0 anymore
'The reload value specifies the interval of the timer in uS
Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10
'we use 2 servos with 10 uS resolution

'we must configure the port pins used to act as output
Config Portb = Output

'finally we must turn on the global interrupt
Enable Interrupts

'the servo() array is created automatic. You can used it to set the
'time the servo must be on
Servo(1) = 100 '1000 uS on
Servo(2) = 200 ' 2000 uS on


Dim I As Byte
Do
For I = 0 To 100
Servo(1) = I
Waitms 1000
Next

For I = 200 To 0 Step -1
Servo(1) = I
Waitms 1000
Next
Loop
End
```

CONFIG TCPIP

Action

Configures the TCP/IP W3100A chip.

Syntax

CONFIG TCPIP = int , **MAC** = mac , **IP** = ip, **SUBMASK** = mask, **GATEWAY** = gateway, **LOCALPORT** = port, **TX** = tx, **RX** = rx , **NOINIT** = 0|1

Remarks

Int	The interrupt to use such as INT0 or INT1. For the Easy TCP/IP PCB, use INT0.
MAC	The MAC address you want to assign to the W3100A. The MAC address is a unique number that identifies your chip. You must use a different address for every W3100A chip in your network. Example : 123.00.12.34.56.78 You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation.
IP	The IP address you want to assign to the W3100A. The IP address must be unique for every W3100A in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address.
SUBMASK	The submask you want to assign to the W3100A. The submask is in most cases 255.255.255.0
GATEWAY	This is the gateway address of the W3100A. The gateway address you can determine with the IPCONFIG command at the command prompt : C:\>ipconfig Windows 2000 IP Configuration Ethernet adapter Local Area Connection 2: Connection-specific DNS Suffix . : IP Address. . . . . : 192.168.0.3 Subnet Mask . . . . : 255.255.255.0 Default Gateway . . . . : **192.168.0.1** Use 192.168.0.1 in this case.
LOCALPORT	A word value that is assigned to the LOCAL_PORT internal variable. See also Getsocket. As a default you can assign a value of 5000.
TX	A byte which specifies the transmit buffer size of the W3100A. The W3100A has 4 sockets. A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes. This is binary notation. And the Msbits specify the size of socket 3. For example, you want to assign 2048 bytes to each socket for transmission : TX = &B01010101 Since the transmission buffer size may be 8KB in total, you can split them up in 4 parts of 2048 bytes : 01. When you want to use 1 socket with 8KB size, you would use : TX = &B11. You can use only 1 socket in that case : socket 0.
RX	A byte which specifies the receive buffer size of the W3100A. The W3100A has 4 sockets. A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes. This is binary notation. And the Msbits specify the size of socket 3. For example, you want to assign 2048 bytes to each socket for reception : RX = &B01010101 Since the receive buffer size may be 8KB in total, you can split them up in 4 parts of 2048 bytes : 01. When you want to use 1 socket with 8KB size, you would use : RX = &B11. You can use only 1 socket in that case : socket 0. Consult the W3100A pdf for more info.
Noinit	Make this 1 when you want to configure the TCP, MAC, Subnetmask and GateWay dymanic. Noinit will only make some important settings and you need to use SETTCP in order to finish the setup.

The CONFIG TCPIP statement may be used only once.

Interrupts must be enabled before you use CONFIG TCPIP.

Configuring the W3100A will init the chip.

After the CONFIG TCPIP, you can already PING the chip!

See also

GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN

Example

```
Config Tcpip = Int0 , Mac = 00.00.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 ,
Gateway = 192.168.0.1 , Localport = 1000 , Tx = $55 , Rx = $55
'Now use PING at the command line to send a ping:
PING 192.168.0.8
Or use the easytcp application to ping the chip.
```

CONFIG TIMER0

Action

Configure TIMER0.

Syntax

CONFIG TIMER0 = COUNTER , PRESCALE= 1|8|64|256|1024 , EDGE=RISING/FALLING , CLEAR TIMER = 1|0

CONFIG TIMER0 = TIMER , PRESCALE= 1|8|64|256|1024

Remarks

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

When configured as a COUNTER:

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

When configured as a TIMER:

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

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

When you use the STOP TIMER0 statement, the TIMER is stopped.

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

So before using the START and STOP TIMER0 statements, use the CONFIG statement first.

Example

```
'-----------------------------------------------------------------------
' TIMER0.BAS
' example that shows how to use TIMER0 related statements
'-----------------------------------------------------------------------

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

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

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

Do
Print Tcnt0
Loop Until Tcnt0 >= 10
'when 10 pulses are count the loop is exited
'or use the special variable TIMER0
Timer0 = 0


'Now configire it as a TIMER
'The TIMER can have the systemclock as an input or the systemclock
divided
'by 8,64,256 or 1024
'The prescale parameter excepts 1,8,64,256 or 1024
Config Timer0 = Timer , Prescale = 1

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

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

'Again you can access the value with the tcnt0 register
Print Tcnt0
'or
Print Timer0
'when the timer overflows, a flag named TOV0 in register TIFR is set
'You can use this to execute an ISR
'To reset the flag manual in non ISR mode you must write a 1 to the bit
position
'in TIFR:
Set Tifr.1



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

'(

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

'Define the ISR handler
On Ovf0 Tim0_isr
'you may also use TIMER0 for OVF0, it is the same

Enable Timer0 ' enable the timer interrupt
Enable Interrupts 'allow interrupts to occur
Do
'your program goes here
Loop

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

')

End
```

CONFIG TIMER1

Action

Configure TIMER1.

Syntax

CONFIG TIMER1 = COUNTER | TIMER | PWM ,
EDGE=RISING | FALLING , PRESCALE= 1|8|64|256|1024 ,
NOISE CANCEL=0 |1, CAPTURE EDGE =RISING | FALLING ,
CLEAR TIMER = 1|0,
COMPARE A= CLEAR | SET | TOGGLE I DISCONNECT ,
COMPARE B= CLEAR | SET | TOGGLE I DISCONNECT ,
PWM = 8 | 9 10 ,
COMPARE A PWM= CLEAR UP| CLEAR DOWN | DISCONNECT
COMPARE B PWM= CLEAR UP| CLEAR DOWN | DISCONNECT

Remarks

The TIMER1 is a 16 bit counter. See the hardware description of TIMER1.
It depends on the chip if COMPARE B is available or not.

The syntax shown above must be on one line. Not all the options need to be selected.

Here is the effect of the various options.

EDGE	You can select whether the TIMER will count on the falling or rising edge. Only for COUNTER mode.
CAPTURE EDGE	You can choose to capture the TIMER registers to the INPUT CAPTURE registers With the CAPTURE EDGE = FALLING/RISING, you can specify to capture on the falling or rising edge of pin ICP
NOISE CANCELING	To allow noise canceling you can provide a value of 1.
PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter. Valid values are 1 , 8, 64, 256 or 1024

The TIMER1 also has two compare registers A and B
When the timer value matches a compare register, an action can be performed

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

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

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

Using COMPARE A, COMPARE B, COMPARE A PWM or COMPARE B PWM will set the corresponding pin for output. When this is not wanted you can use the alternative NO_OUTPUT version that will not alter the output pin.
For example : COMPARE A NO_OUTPUT , COMPARE A PWM NO_OUTPUT

Example

```
'-----------------------------------------------------------------
' TIMER1.BAS for the 8515
'-----------------------------------------------------------------

Dim W As Word

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

'First like TIMER0 , it can be set to act as a TIMER or COUNTER
'Lets configure it as a TIMER that means that it will count and that
'the input is provided by the internal clock.
'The internal clock can be divided by 1,8,64,256 or 1024
Config Timer1 = Timer , Prescale = 1024


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


'To use it as a COUNTER, you can choose on which edge it is triggered
Config Timer1 = Counter , Edge = Falling, , Prescale = 1024
'Config Timer1 = Counter , Edge = Rising

'Also you can choose to capture the TIMER registers to the INPUT CAPTURE
registers
'With the CAPTURE EDGE = , you can specify to capture on the falling or
rising edge of pin ICP
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling , ,
Prescale = 1024
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'To allow noise canceling you can also provide :
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling , Noise
Cancel = 1, , Prescale = 1024

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

```
'The TIMER also has two compare registers A and B
'When the timer value matches a compare register, an action can be
performed
Config Timer1 = Counter , Edge = Falling , Compare A = Set , Compare B =
Toggle, , Prescale = 1
'SET , will set the OC1X pin
'CLEAR, will clear the OC1X pin
'TOGGLE, will toggle the OC1X pin
'DISCONNECT, will disconnect the TIMER from output pin OC1X


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


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

'to set the PWM registers, just assign a value to the compare A and B
registers
Compare1a = 100
Compare1b = 200

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

End
```

CONFIG TIMER2

Action

Configure TIMER2.

Syntax for the 8535

CONFIG TIMER2 = TIMER | PWM , ASYNC=ON |OFF,
PRESCALE = 1 | 8 | 32 | 64 | 128 | 256 | 1024 ,
COMPARE= CLEAR | SET | TOGGLE I DISCONNECT ,
PWM = ON | OFF ,
COMPARE PWM = CLEAR UP| CLEAR DOWN | DISCONNECT ,
CLEAR TIMER = 1|0

Syntax for the M103

CONFIG TIMER2 = COUNTER| TIMER | PWM ,
EDGE= FALLING |RISING,
PRESCALE = 1 | 8 | 64 | 256 | 1024 ,
COMPARE= CLEAR | SET | TOGGLE I DISCONNECT ,
PWM = ON | OFF ,
COMPARE PWM = CLEAR UP| CLEAR DOWN | DISCONNECT ,
CLEAR TIMER = 1|0

Remarks

The TIMER2 is an 8 bit counter.

It depends on the chip if it can work as a counter or not.

The syntax shown above must be on one line. Not all the options need to be selected.

Here is the effect of the various options.

EDGE	You can select whether the TIMER will count on the falling or rising edge. Only for COUNTER mode.

PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter. Valid values are 1 , 8, 64, 256 or 1024 or 1 , 8, 32 , 64 , 256 or 1024 for the M103

The TIMER2 also has a compare registers

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

COMPARE	The action can be: SET will set the OC2 pin CLEAR will clear the OC2 pin TOGGLE will toggle the OC2 pin DISCONNECT will disconnect the TIMER from output pin OC2

And the TIMER can be used in 8 bit PWM mode
You can specify if the counter must count UP or down after a match
to the compare registers

COMPARE PWM	PWM compare mode. Can be CLEAR UP or CLEAR DOWN

Example

```
'-----------------------------------------------------------------

Dim W As Byte

Config Timer2 = Timer , ASYNC = 1 , Prescale = 128
On TIMER2 Myisr
ENABLE INTERRUPTS
ENABLE TIMER2
DO

LOOP

MYISR:
'get here every second with a 32768 KHz xtal
RETURN

'You can read or write to the timer with the COUNTER2 or TIMER2 variable
W = Timer2
Timer2 = W
```

CONFIG WAITSUART

Action

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

Syntax

CONFIG WAITSUART =value

Remarks

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

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

See also

OPEN

Example

```
See OPEN example for more details.
```

CONFIG WATCHDOG

Action

Configures the watchdog timer.

Syntax

CONFIG WATCHDOG = time

Remarks

Time	The interval constant in mS the watchdog timer will count to before it will reset your program. Possible settings : 16 , 32, 64 , 128 , 256 , 512 , 1024 and 2048.

When the WD is started, a reset will occur after the specified number of mS.

With 2048, a reset will occur after 2 seconds, so you need to reset the WD in your programs periodically with the **RESET WATCHDOG** statement.

See also

START WATCHDOG , STOP WATCHDOG , RESET WATCHDOG

Example

```
'-----------------------------------------------------
' (c) 1999 MCS Electronics
' WATCHD.BAS demonstrates the watchdog timer
'-----------------------------------------------------
Config Watchdog = 2048 'reset after 2048 mSec
Start Watchdog 'start the watchdog timer
Dim I As Word
For I = 1 To 1000

Print I 'print value
'Reset Watchdog
'you will notice that the for next doesnt finish because of the reset
'when you unmark the RESET WATCHDOG statement it will finish because the
'wd-timer is reset before it reaches 2048 msec
Next
End
```

CONFIG PORT

Action

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

Syntax

CONFIG PORTx =state

CONFIG PINx.y =state

Remarks

state	A constant that can be INPUT or OUTPUT. INPUT will set the data direction register to input for port X. OUTPUT will set the data direction to output for port X. You can also use a number for state. &B0001111, will set the upper nibble to input and the lower nibble to output. You can also set one port pin with the CONFIG PIN = state, statement. Again, you can use INPUT, OUTPUT or a number. In this case the number can be only zero or one.

state : Constant.

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

Example

```
'-------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'-------------------------------------------------------------
' file: PORT.BAS
' demo: PortB and PortD
'-------------------------------------------------------------
Dim A As Byte , Count As Byte

'configure PORT D for input mode
Config Portd = Input

'reading the PORT, will read the latch, that is the value
'you have written to the PORT.
'This is not the same as reading the logical values on the pins!
'When you want to know the logical state of the attached hardware,
'you MUST use the PIN register.
A = Pind

'a port or SFR can be treated as a byte
A = A And Portd

Print A 'print it

Bitwait Pind.7 , Reset 'wait until bit is low


'We will use port B for output
Config Portb = Output
```

```
'assign value
Portb = 10 'set port B to 10
Portb = Portb And 2

Set Portb.0 'set bit 0 of port B to 1

Incr Portb

'Now a light show on the STK200
Count = 0
Do
Incr Count
Portb = 1
For A = 1 To 8
Rotate Portb , Left 'rotate bits left
Wait 1
Next
'the following 2 lines do the same as the previous loop
'but there is no delay
' Portb = 1
' Rotate Portb , Left , 8
Loop Until Count = 10
Print "Ready"

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

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

End
```

CONFIG X10

Action

Configures the pins used for X10.

Syntax

CONFIG X10 = pinZC , TX = portpin

Remarks

PinZC	The pin that is connected to the zero cross output of the TW-523. This is a pin that will be used as INPUT.
Portpin	The pin that is connected to the TX pin of the Tw-523. TX is used to send X10 data to the TW-523. This pin will be used in output mode.

The TW-523RJ-11 connector has the following pinout:

Pin	Description	Connect to micro
1	ZeroCross	Input pin. Add 5.1K pull up.
2	GND	GND
3	RX	Not used.
4	TX	Output pin. Add 1K pull up.

See also

X10DETECT , X10SEND

Example

```
'--------------------------------------------------------------------------
' X10.BAS
' (c) 2002-2004 MCS Electronics
' This example needs a TW-523 X10 interface
'--------------------------------------------------------------------------
$crystal = 8000000
$baud = 19200

'define the house code
Const House = "M" ' use code A-P

Waitms 500 ' optional delay not really needed

'dim the used variables
```

```
Dim X As Byte

'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
'  ^--zero cross
'  ^--- transmission pin

'detect the TW-523
X = X10detect()
Print X ' 0 means error, 1 means 50 Hz, 2 means 60 Hz

Do
Input "Send (1-32) ", X
'enter a key code from 1-31
'1-16 to address a unit
'17 all units off
'18 all lights on
'19 ON
'20 OFF
'21 DIM
'22 BRIGHT
'23 All lights off
'24 extended code
'25 hail request
'26 hail acknowledge
'27 preset dim
'28 preset dim
'29 extended data analog
'30 status on
'31 status off
'32 status request

X10send House , X ' send the code
Loop
End
```

CONST

Action

Declares a symbolic constant.

Syntax

CONST symbol = numconst

CONST symbol = stringconst

CONST symbol = expression

Remarks

Symbol	The name of the symbol.
Numconst	The numeric value to assign to the symbol.
Stringconst	The string to assign to the symbol
Expression	An expression that returns a value to assign the constant

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

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

See also

ALIAS

Difference with BASCOM-8051

In BASCOM-8051 only numeric constants can be used.

Example

```
'dimension some variables
Dim Z As String * 10
Dim B As Byte

'assign some constants
'constants dont use program memory
Const S = "test"
Const A = 5 'declare a as a constant
Const B1 = &B1001

'or use an expression to assign a constant
Const X = (b1 * 3) + 2
Const Ssingle = Sin(1)
```

COS

Action

Returns the cosine of a single

Syntax

var =**COS**(single)

Remarks

Var	A numeric variable that is assigned with cosine of variable single.
Single	The single variable to get the cosine of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , SIN

Example

Show sample

COSH

Action

Returns the cosine hyperbole of a single

Syntax

var =**COSH** (single)

Remarks

Var	A numeric variable that is assigned with cosine hyperbole of variable single.
Single	The single variable to get the cosine hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , COS , SIN , TANH , SINH

Example

Show sample

COUNTER0 and COUNTER1

Action

Set or retrieve the internal 16 bit hardware register.

Syntax

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

Remarks

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

Because the above 16 bit register pairs must be accessed somewhat differently than you may expect, they are implemented as variables.

The exception is TIMER0/COUNTER0, this is a normal 8 bit register and is supplied for compatibility with the syntax.

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

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

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

Note that the available registers may vary from chip to chip.

The BASCOM documentation used the 8515 to describe the different hardware registers.

CPEEK

Action

Returns a byte stored in code memory.

Syntax

var = **CPEEK(** address **)**

Remarks

Var	Numeric variable that is assigned with the content of the program memory at address
Address	Numeric variable or constant with the address location

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

Cpeek(0) will return the first byte of the file. Cpeek(1) will return the second byte of the binary file.

See also

PEEK , POKE , INP , OUT

Example

```
'-----------------------------------------------------
' (c) 1998-2000 MCS Electronics
' PEEK.BAS
' demonstrates PEEk, POKE, CPEEK, INP and OUT
'
'-----------------------------------------------------
Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31 'only 32 registers in AVR
B1 = Peek(i) 'get byte from internal memory
Print Hex(b1) ; " ";
'Poke I , 1 'write a value into memory
Next
Print 'new line
'be careful when writing into internal memory !!

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

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

End
```

CPEEKH

Action

Returns a byte stored in upper page of code memory of M103.

Syntax

var **= CPEEKH(** address **)**

Remarks

Var	Numeric variable that is assigned with the content of the program memory at address
Address	Numeric variable or constant with the address location

CpeekH(0) will return the first byte of the upper 64KB.

Since the M103 has 64K words of code space the LPM instruction can not access the 64 upper Kbytes.

The CpeekH() function peeks in the upper 64 KB.

This function should be used with the M103 only.

See also

PEEK , POKE , INP , OUT

Example

```
'-----------------------------------------------------
' (c) 1998-2000 MCS Electronics
' PEEK.BAS
' demonstrates PEEk, POKE, CPEEK, INP and OUT
'
'-----------------------------------------------------
Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31 'only 32 registers in AVR
B1 = Peek(i) 'get byte from internal memory
Print Hex(b1) ; " ";
'Poke I , 1 'write a value into memory
Next
Print 'new line
'be careful when writing into internal memory !!

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

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

CRC8

Action

Returns the CRC8 value of a variable or array.

Syntax

Var = **CRC8**(source , L)

Remarks

Var	The variable that is assigned with the CRC8 of variable source.
Source	The source variable or first element of the array to get the CRC8 of.
L	The number of bytes to check.

CRC8 is used in communication protocols to check if there are no transmission errors.
The 1wire for example returns a crc byte as the last byte from it's ID.

The code below shows a VB function of crc8

```
Function Docrc8(s As String) As Byte
Dim j As Byte
Dim k As Byte
Dim crc8 As Byte
crc8 = 0
For m = 1 To Len(s)
x = Asc(Mid(s, m, 1))
For k = 0 To 7
j = 1 And (x Xor crc8)
crc8 = Fix(crc8 / 2) And &HFF
x = Fix(x / 2) And &HFF
If j <> 0 Then
crc8 = crc8 Xor &H8C
End If
Next k
Next
Docrc8 = crc8
End Function
```

See also

CHECKSUM , CRC16

ASM

The following routine is called from mcs.lib : _CRC8

The routine must be called with Z pointing to the data and R24 must contain the number of bytes to check.

On return, R16 contains the CRC8 value.

The used registers are : R16-R19, R25.

```
;##### X = Crc8(ar(1) , 7)
Ldi R24,$07     ; number of bytes
Ldi R30,$64     ; address of ar(1)
Ldi R31,$00     ; load constant in register
Rcall _Crc8     ; call routine
Ldi R26,$60     ; address of X
St X,R16     ; store crc8
```

Example

```
Dim Ar(8) As Byte , X As Byte

'init array
Ar(1) = &H2
Ar(2) = &H1C
Ar(3) = &HB8
Ar(4) = 1
Ar(5) = 0
Ar(6) = 0
Ar(7) = 0

'get crc8 of array. Scan 7 bytes
X = Crc8(ar(1) , 7)
```

CRC16

Action

Returns the CRC16 value of a variable or array.

Syntax

Var = **CRC16**(source , L)

Remarks

Var	The variable that is assigned with the CRC16 of variable source. Should be a word or integer variable.
Source	The source variable or first element of the array to get the CRC16 value from.
L	The number of bytes to check.

CRC16 is used in communication protocols to check if there are no transmission errors.

The 1wire for example returns a crc byte as the last byte from it's ID.

Use CRC8 for the 1wire routines.

There are a lot of different CRC16 routines. There is no real standard since the polynomial will vary from manufacture to manufacture.

See also

CHECKSUM , CRC8

ASM

The following routine is called from mcs.lib : _CRC16

The routine must be called with X pointing to the data. The soft stack –Y must contain the number of bytes to scan.

On return, R16 and R17 contain the CRC16 value.

The used registers are : R16-R19, R25.

```
;##### X = Crc16(ar(1) , 7)
Ldi R24,$07     ; number of bytes
St –y, R24
Ldi R26,$64     ; address of ar(1)
Ldi R27,$00     ; load constant in register
Rcall _Crc16     ; call routine
Ldi R26,$60     ; address of X
St X+,R16     ; store crc16 LSB
St X , R17     ; store CRC16 MSB
```

Example

```
Dim Ar(8) As Byte , X As Word

'init array
Ar(1) = &H2
Ar(2) = &H1C
Ar(3) = &HB8
Ar(4) = 1
Ar(5) = 0
Ar(6) = 0
Ar(7) = 0

'get crc16 of array. Scan 7 bytes
X = Crc16(ar(1) , 7)
```

CRYSTAL

Action

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

Syntax

CRYSTAL = var (old option do not use !!)

___CRYSTAL1 = var

BAUD #1, 2400

Remarks

With the software UART you can generate good baud rates. But chips such as the ATtiny22 have an internal 1 MHz clock. The clock frequency can change during runtime by influence of temperature or voltage.

The crystal variable can be changed during runtime to change the baud rate.

The above has been changed in version 1.11

Now you still can change the baud rate with the crystal variable.

But you dont need to dimension it. And the name has been changed:

___CRYSTALx where x is the channel number.

When you opened the channel with #1, the variable will be named ___CRYSTAL1

But a better way is provided now to change the baud rate of the software uart at run time. You can use the BAUD option now:

Baud #1 , 2400 'change baud rate to 2400 for channel 1

When you use the baud # option, you must specify the baud rate before you print or use input on the channel. This will dimension the ___CRYSTALx variable and load it with the right value.

When you don't use the BAUD # option the value will be loaded from code and it will not use 2 bytes of your SRAM.

The ___CRYSTALx variable is hidden in the report file because it is a system variable. But you may assign a value to it after BAUD #x, zzzz has dimensioned it.

The old CRYSTAL variable does not exist anymore.

Some values for 1 MHz internal clock :

66 for 2400 baud

31 for 4800 baud

14 for 9600 baud

See also

OPEN , CLOSE

Example

```
Dim B as byte
Open "comd.1:9600,8,n,1,inverted" For Output As #1
Print #1 , B
Print #1 , "serial output"
baud #1, 4800 'use 4800 baud now
Print #1, "serial output"
___CRYSTAL1 = 255
Close #1
End
```

CURSOR

Action

Set the LCD Cursor State.

Syntax

CURSOR ON / OFF BLINK / NOBLINK

Remarks

You can use both the ON or OFF and BLINK or NOBLINK parameters.

At power up the cursor state is ON and NOBLINK.

See also

DISPLAY , LCD

Example

```
Dim a As Byte
a = 255
Lcd A
Cursor Off 'hide cursor
Wait 1 'wait 1 second
Cursor Blink 'blink cursor
End
```

DATA

Action

Specifies constant values to be read by subsequent READ statements.

Syntax

DATA var [, varn]

Remarks

Var	Numeric or string constant.

The DATA related statements use the internal registers pair R8 and R9 to store the data pointer.

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

DATA $34

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

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

Another way to include special ASCII characters in your string constant is to use {XXX}. You need to include exactly 3 digits representing the ASCII character. For example 65 is the ASCII number for the character A.

DATA "TEST{065}"

Will be read as TESTA.

While :

DATA "TEST{65}" will be read as :

TEST{65}. This because only 2 digits were included instead of 3.

{xxx} works only for string constants. It will also work in a normal string assignment :

s = "{065}" . This will assign A to the string s.

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

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

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

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

For example this is fine:

Print "Hello"

Goto jump

DATA "test"

Jump:

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

The following example will case some problems:

Dim S As String * 10

```
Print "Hello"
Restore lbl
Read S
DATA "test"
Print S
```

When the END statement is used it must be placed BEFORE the DATA lines.

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 , $DATA , $EEPROM

Example

```
'-------------------------------------------------------------
' READDATA.BAS
' Copyright 1999-2000 MCS Electronics
'-------------------------------------------------------------

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

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

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


Restore Dta4
Read L : Print L 'long type

End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

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

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

DAYOFWEEK

Action

Returns the Day of the Week of a Date.

Syntax

Target = **DayOfWeek**()

Target = **DayOfWeek**(bDayMonthYear)

Target = **DayOfWeek**(strDate)

Target = **DayOfWeek**(wSysDay)

Target = **DayOfWeek**(lSysSec)

Remarks

Target	A Byte – variable, that is assigned with the day of the week
BDayMonthYear	A Byte – variable, which holds the Day-value followed by Month(Byte) and Year (Byte)
StrDate	A String, which holds a Date-String in the format specified in the **CONFIG DATE** statement
WSysDay	A Word – variable, which holds the System Day (SysDay)
LSysSec	A Long – variable, which holds the System Second (SysSec)

The Function can be used with five different kind of Input:

1. 1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. 2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Week can be calculated of every date.
3. 3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement
4. 4. With a System Day – Number.
5. 5. With a System Second - Number

The Return-Value is in the range of 0 to 6, Monday starts with 0.

The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See Also

Date and Time routines , CONFIG DATE , CONFIG CLOCK, SYSDAY, SYSSEC

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format

Dim bWeekDay as Byte , strWeekDay as String * 10
Dim strDate as String * 8
Dim bDay as Byte , bMonth as Byte , bYear as Byte
Dim wSysDay as Word
Dim lSysSec as Long

' Example 1 with internal RTC-Clock
_Day = 24 : _Month = 10 : _Year = 2 ' Load RTC-Clock for example -
testing
bWeekDay = DayOfWeek()
strWeekDay = Lookupstr(bWeekDay , WeekDays)
print "Weekday-Number of " ; Date$ ; " is " ; bWeekday ; " = " ;
strWeekday
' Weekday-Number of 02.10.24 is 3 = Thursday


' Example 2 with defined Clock - Bytes (Day / Month / Year)
bDay = 26 : bMonth = 11 : bYear = 2
bWeekDay = DayOfWeek(bDay)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
print "Weekday-Number of Day="; bDay ; " Month="; bMonth ; " Year=" ;
bYear ; " is " ; bWeekday ; " = " ; strWeekday
' Weekday-Number of Day=26 Month=11 Year=2 is 1 = Tuesday


' Example 3 with System Day
wSysDay = 2000 ' that is 2005-06-23
bWeekDay = DayOfWeek(wSysDay)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
print "Weekday-Number of System Day " ; wSysDay ; " is " ; bWeekday ; " =
" ; strWeekday
' Weekday-Number of System Day 2000 is 3 = Thursday


' Example 4 with System Second
lSysSec = 123456789 ' that is 2005-06-23
bWeekDay = DayOfWeek(lSysSec)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
print "Weekday-Number of System Second " ; lSysSec ; " is " ; bWeekday ;
" = " ; strWeekday
' Weekday-Number of System Second 123456789 is 5 = Saturday


' Example 5 with Date-String
strDate = "02.11.26" ' we have configured Date in ANSI
bWeekDay = DayOfWeek(strDate)
strWeekDay = Lookupstr(bWeekDay , WeekDays)
print "Weekday-Number of " ; strDate ; " is " ; bWeekday ; " = " ;
strWeekday
' Weekday-Number of 02.11.26 is 1 = Tuesday

End

WeekDays:
Data "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"
```

DAYOFYEAR

Action

Returns the Day of the Year of a Date

Syntax

Target = **DayOfYear**()

Target = **DayOfYear**(bDayMonthYear)

Target = **DayOfYear**(strDate)

Target = **DayOfYear**(wSysDay)

Target = **DayOfYear**(lSysSec)

Remarks

Target	A Integer, that is assigned with the Day of the Year
BDayMonthYear	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
StrDate	A String, which holds a Date-String in the format specified in the **CONFIG DATE** statement
WSysDay	A Variable (Word) which holds a System Day (SysDay)
LsysSec	A Variable (Long) which holds a System Second (SysSec)

The Function can be used with five different kind of Input:

1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
4. With a System Day Number (WORD)
5. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 364 (365 in a leap year). January the first starts with 0.

The function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also

[Date and Time Routines](), [SysSec](), [SysDay]()

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format
Dim strDate as String * 8
Dim bDay as Byte , bMonth as Byte , bYear as Byte
Dim wSysDay as Word
Dim lSysSec as Long
Dim wDayOfYear as Word


' Example 1 with internal RTC-Clock
_day = 20 : _Month = 11 : _Year = 2 ' Load RTC-Clock for example -
testing
wDayOfYear = DayOfYear()
print "Day Of Year of " ; Date$ ; " is " ; wDayOfYear
' Day Of Year of 02.11.20 is 323


' Example 2 with defined Clock - Bytes (Day / Month / Year)
bDay = 24 : bMonth = 5 : bYear = 8
wDayOfYear = DayOfYear(bDay)
print "Day Of Year of Day=" ; bDay ; " Month=" ; bMonth ; " Year=" ; bYear
; " is " ; wDayOfYear
' Day Of Year of Day=24 Month=5 Year=8 is 144


' Example 3 with Date - String
strDate = "04.10.29"
wDayOfYear = DayOfYear(strDate)
print "Day Of Year of " ; strDate ; " is " ; wDayOfYear
' Day Of Year of 04.10.29 is 302

' Example 4 with System Day
wSysDay = 3000
wDayOfYear = DayOfYear(wSysDay)
print "Day Of Year of System Day " ; wSysDay ; " is " ; wDayOfYear
' Day Of Year of System Day 3000 is 78


' Example 5 with System Second
lSysSec = 123456789
wDayOfYear = DayOfYear(lSysSec)
print "Day Of Year of System Second " ; lSysSec ; " is " ; wDayOfYear
' Day Of Year of System Second 123456789 is 332
```

DATE$

Action

Internal variable that holds the date.

Syntax

DATE$ = "mm/dd/yy"

var = DATE$

Remarks

The DATE$ variable is used in combination with the CONFIG CLOCK directive.

The CONFIG CLOCK statement will use the TIMER0 or TIMER2 in async mode to create a 1 second interrupt. In this interrupt routine the _Sec, _Min and _Hour variables are updated. The _dat, month and _year variables are also updated. The date format is in the same format as for QB/VB.

When you assign DATE$ to a string variable these variables are assigned to the DATE$ variable.

When you assign the DATE$ variable with a constant or other variable, the _day, _month and _year variables will be changed to the new date.

The only difference with QB/VB is that all data must be provided when assigning the date. This is done for minimal code. You can change this behavior of course.

The async timer is only available in the M103, 90S8535, M163 and M32(3), Mega128, Mega8. For other chips it will not work.

ASM

The following asm routines are called.

When assiging DATE$: _set_date (calls _str2byte)

When reading DATE$: _make_dt (calls _byte2str)

See also

TIME$, CONFIG CLOCK

Example

```
'-----------------------------------------------------------
' MEGACLOCK.BAS
' (c) 2000-2004 MCS Electronics
'-----------------------------------------------------------
'This example shows the new TIME$ and DATE$ reserved variables
'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000 'address for E and RS
$lcdrs = &H8000 'address for only E
Config Lcd = 20 * 4 'nice display from bg micro
Config Lcdbus = 4 'we run it in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus 'tell about the bus mode

'[now init the clock]
Config Date = Mdy , Separator = / ' ANSI-Format

Config Clock = Soft 'this is how simple it is
'The above statement will bind in an ISR so you can not use the TIMER anymore!
'For the M103 in this case it means that TIMER0 can not be used by the user anymore

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'---------------------------------------------------

'clear the LCD display
Cls

Do
Home 'cursor home
Lcd Date$ ; " " ; Time$ 'show the date and time
Loop

'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

DATE

Action

Returns a date-value (String or 3 Byte for Day, Month and Year) depending of the Type of the Target

Syntax

bDayMonthYear = **Date**(lSysSec)

bDayMonthYear = **Date**(lSysDay)

bDayMonthYear = **Date**(strDate)

strDate = **Date**(lSysSec)

strDate = **Date**(lSysDay)

strDate = **Date**(bDayMonthYear)

Remarks

StrDate	A Date-String in the format specified in the **CONFIGDATE** statement
LsysSec	ALONG – variable which holds the System Second (SysSec = TimeStamp)
LsysDay	A WORD – variable, which holds then System Day (SysDay)
BDayMonthYear	ABYTE – variable, which holds Days, followed by Month (Byte) and Year (Byte)

Converting to String:

The target string must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Softclock date format (3 Bytes for Day, Month and Year):

Three Bytes for Day, Month and Year must follow each other in SRAM. The variable-name of the first Byte, that one for Day must be passed to the function.

See also

Date and Time Routines , DAYOFYEAR, SYSDAY

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format
Dim strDate as String * 8
Dim bDay as Byte , bMonth as Byte , bYear as Byte
Dim wSysDay as Word
Dim lSysSec as Long
' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to
Date - String
bDay = 29 : bMonth = 4 : bYear = 12
strDate = Date(bDay)
print "Dat values: Day="; bDay ; " Month="; bMonth ; " Year=" ; bYear ; "
converted to string " ; strDate
' Dat values: Day=29 Month=4 Year=12 converted to string 12.04.29

' Example 2: Converting from System Day to Date - String
wSysDay = 1234
strDate = Date(wSysDay)
Print "System Day " ; wSysDay ; " is " ; strDate
' System Day 1234 is 03.05.19

' Example 3: Converting from System Second to Date String
lSysSec = 123456789
strDate = Date(lSysSec)
Print "System Second " ; lSysSec ; " is " ; strDate
' System Second 123456789 is 03.11.29

' Example 4: Converting SystemDay to defined Clock - Bytes (Day / Month /
Year)
wSysDay = 2000
bDay = Date(wSysDay)
print "System Day " ; wSysDay ; " converted to Day="; bDay ; " Month=";
bMonth ; " Year=" ; bYear
' System Day 2000 converted to Day=23 Month=6 Year=5

' Example 5: Converting Date - String to defined Clock - Bytes (Day /
Month / Year)
strDate = "04.08.31"
bDay = Date(strDate)
print "Date " ; strDate ; " converted to Day="; bDay ; " Month="; bMonth
; " Year=" ; bYear
' Date 04.08.31 converted to Day=31 Month=8 Year=4

' Example 6: Converting System Second to defined Clock - Bytes (Day /
Month / Year)
lSysSec = 123456789
bDay = Date(lSysSec)
print "System Second " ; lSysSec ; " converted to Day="; bDay ; "
Month="; bMonth ; " Year=" ; bYear
' System Second 123456789 converted to Day=29 Month=11 Year=3
```

DBG

Action

Prints debug info to the hardware UART

Syntax

DBG

Remarks

See $DBG for more information

DEBOUNCE

Action

Debounce a port pin connected to a switch.

Syntax

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

Remarks

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

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

The DEBOUNCE statement tests the condition of the specified pin and it true there will be a delay for 25 mS and the condition will be checked qagain.(eliminating bounce of a switch)

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

When the condition is not true, or the logic level on the pin is not of the specified level, the code on the next line will be executed.

When DEBOUNCE is executed again, the state of the switch must have gone back in the original position before it can perform another branch. So if you are waiting for a pin to go low, and the pin goes low, the pin must change to high, before a new low level will result in another branch.

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

DEBOUNCE will not wait for the input value to met the specified condition. You need to use BITWAIT if you want to wait until a bit will have a certain value.

So DEBOUNCE will not halt your program while a BITWAIT can halt your program if the bit will never have the specified value. You can combine BITWAIT and DEBOUNCE statements by preceding a DEBOUNCE with a BITWAIT statement.

See also

CONFIG DEBOUNCE , BITWAIT

Example

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


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

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

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

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

DECR

Action

Decrements a variable by one.

Syntax

DECR var

Remarks

Var	Variable to decrement.

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

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

See also

INCR

Example

```
'--------------------------------------------------------------
' (c) 2000 MCS Electronics
'--------------------------------------------------------------
' file: DECR.BAS
' Demo: DECR
'--------------------------------------------------------------
Dim A As Byte , I As Integer

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


I = 1000
Decr I
Print I
End
```

DECLARE FUNCTION

Action

Declares a user function.

Syntax

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

Remarks

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

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

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

Use BYVAL to pass a copy of the variable.

See the CALL statement for more details.

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

Bits are global and can not be passed with functions or subs.

See also

CALL, SUB

Example

```
'-----------------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
' Demonstration of user function
'-----------------------------------------------------------------------

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

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

T = Myfunction(k , Z)
Print T
End

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

P = I

'because I is passed by value, altering will not change the original
'variable named k
I = 10

P = Val(s) + I

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

DECLARE SUB

Action

Declares a subroutine.

Syntax

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

Remarks

test	Name of the procedure.
Var	Name of the variable(s).
Type	Type of the variable(s). Byte, Word, Integer, Long, Single or String.

When BYREF or BYVAL is not provided, the parameter will be passed by reference.
Use BYREF to pass a variable by reference with its address.
Use BYVAL to pass a copy of the variable.
See the CALL statement for more details.

You must declare each sub before writing or calling the sub procedure.
Bits are global and can not be passed to a sub or function.

See also

CALL, SUB

Example

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

Print a ; b1 ; c

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


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

DEFxxx

Action

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

Syntax

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

Difference with QB

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

Example

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

DEFLCDCHAR

Action

Define a custom LCD character.

Syntax

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

Remarks

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

You can use the LCD designer to build the characters.

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

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

See also

Tools LCD designer

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
```

DEG2RAD

Action

Converts an angle in to radians.

Syntax

var =**DEG2RAD**(single)

Remarks

Var	A numeric variable that is assigned with the degrees of variable single.
Single	The single variable to get the degrees of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG

Example

```
Dim S As Single
S = 90
S = Deg2Rad(s)
Print S
```

DELAY

Action

Delay program execution for a short time.

Syntax

DELAY

Remarks

Use DELAY to wait for a short time.

The delay time is ca. 1000 microseconds.

See also

WAIT , WAITMS

Example

```
Portb = 5
Delay
```

DIM

Action

Dimension a variable.

Syntax

DIM var **AS** [**XRAM/SRAM/ERAM** **]** type [AT location] [OVERLAY]

Remarks

Var	Any valid variable name such as b1, i or longname. var can also be an array : ar(10) for example.
Type	Bit, Byte, Word, Integer, Long, Single or String
XRAM	Specify XRAM to store variable into external memory
SRAM	Specify SRAM to store variable into internal memory (default)
ERAM	Specify ERAM to store the variable into EEPROM
OVERLAY	Specify that the variable is overlaid in memory.

A string variable needs an additional length parameter:

Dim s As XRAM **String * 10**

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

Note that BITS can only be stored in internal memory.

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

The OVERLAY option will not use any variable space. It will create a pointer:

Dim x as Long at $60 ‘long uses 60,61,62 and 63 hex of SRAM

Dim b1 as Byte at $60 OVERLAY

Dim b2 as Byte at $61 OVERLAY

B1 and B2 are not real variables! They point to a place in memory. In this case to &H60 and &H61. By assigning the pointer B1, you will write to memory location &H60 that is used by variable X.

You can also read the content of B1: Print B1

This will print the content of memory location &H60.

By using a pointer you can manipulate the individual bytes of real variables.

Another example

Dim L as Long at &H60

Dim W as Word at &H62 OVERLAY

W will now point to the upper two bytes of the long.

For XRAM variables, you need additional hardware : an external RAM chip.

For ERAM variables, it is important to understand that these are not normal variables. ERAM variables serve as a way to simple read and write the EEPROM memory. You can use READEEPROM and WRITEEEPROM for that purpose too.

ERAM variables only can be assigned to SRAM variables, and ERAM variables can be assigned to SRAM variables. You can not use an ERAM variable as you would use a normal variable.

Dim b as byte, bx as ERAM byte

B= 1

Bx=b ' write to EEPROM

B=bx ' read from EEPROM

Difference with QB

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

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

See Also

CONST , LOCAL

Example

```
'--------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'--------------------------------------------------------------
' file: DIM.BAS
' demo: DIM
'--------------------------------------------------------------
Dim B1 As Bit 'bit can be 0 or 1
Dim A As Byte 'byte range from 0-255
Dim C As Integer 'integer range from -32767 - +32768
Dim L As Long
Dim W As Word
Dim S As String * 10 'length can be up to 10 characters

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


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

'Assign bytes
A = 12
A = A + 1

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

W = 50000
Print W

'Assign long
L = 12345678
Print L

'Assign string
S = "Hello world"
Print S

End
```

DIR

Action

Returns the filename that matches the specified filemask.

Syntax

sFile = **Dir**(mask)

sFile = **Dir**()

Remarks

SFile	A string variable that is assigned with the filename.
Mask	A file mask with a valid DOS filemask like *.TXT Use *.* to select all files.

The first function call needs a file mask. All other calls do not need the filemask. In fact when you want to get the next filename from the directory, you must not provide a mask after the first call.

Dir() returns an empty string when there are no more file or when no file name is found that matches the mask.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILELEN , FILEDATE , FILETIME , FILEDATETIME

WRITE , INPUT

ASM

Calls	_Dir ; with filemask	_Dir0 ; without filemask
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Example

```
'Lets have a look at the file we created
Print "Dir function demo"
S = Dir( "*.*")
'The first call to the DIR() function must contain a file mask
' The * means everything.
'
While Len(s) > 0 ' if there was a file found
Print S ; " " ; Filedate() ; " " ; Filetime() ; " " ; Filelen()
' print file , the date the fime was created/changed , the time and the
size of the file
S = Dir() ' get next
Wend
```

DISABLE

Action

Disable specified interrupt.

Syntax

DISABLE interrupt

Remarks

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

By default all interrupts are disabled.

To disable all interrupts specify INTERRUPTS.

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

The interrupts that are available will depend on the used microprocessor.

See also

ENABLE

Example

```
'--------------------------------------------------------------------
' SERINT.BAS
' (c) 1999-2001 MCS Electronics
' serial interrupt example for AVR
'--------------------------------------------------------------------
'$regfile = "8535def.dat"
```

```
Const Cmaxchar = 20 'number of characters

Dim B As Bit 'a flag for signalling a received character
Dim Bc As Byte 'byte counter
Dim Buf As String * Cmaxchar 'serial buffer
Dim D As Byte

'Buf = Space(Cmaxchar)
'unremark line above for the MID() function in the ISR
'we need to fill the buffer with spaces otherwise it will contain garbage

Print "Start"

On Urxc Rec_isr 'define serial receive ISR
Enable Urxc 'enable receive isr



Enable Interrupts 'enable interrupts to occur

Do
If B = 1 Then 'we received something
Disable Serial
Print Buf 'print buffer
Print Bc 'print character counter


'now check for buffer full
If Bc = Cmaxchar Then 'buffer full
Buf = "" 'clear
Bc = 0 'rest character counter
End If

Reset B 'reset receive flag
Enable Serial
End If
Loop

Rec_isr:
Print "*"
If Bc < Cmaxchar Then 'does it fit into the buffer?
Incr Bc 'increase buffer counter


If Udr = 13 Then 'return?
Buf = Buf + Chr(0)
Bc = Cmaxchar
Else
Buf = Buf + Chr(udr) 'add to buffer
End If


' Mid(buf , Bc , 1) = Udr
'unremark line above and remark the line with Chr() to place
'the character into a certain position
'B = 1 'set flag
End If
B = 1 'set flag
Return
```

DISKFREE

Action

Returns the free size of the Disk

Syntax

lFreeSize =**DiskFree** ()

Remarks

lFreeSize	A Long Variable, which is assigned with the available Bytes on the Disk in Bytes

This functions returns the free size of the disk in Bytes.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKSIZE , GET , PUT
FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN
WRITE , INPUT

ASM

Calls	**_GetDiskFreeSize**	
Input	none	
Output	r16-r19: Long-Value of free Bytes	

Example

```
Dim Gbtemp1 As Byte ' scratch byte
Gbtemp1 = Initfilesystem(1) ' we must init the filesystem once
If Gbtemp1 > 0 Then
Print #1 , "Error " ; Gbtemp1
Else
Print #1 , " OK"
Print "Disksize : " ; Disksize() ' show disk size in bytes
Print "Disk free: " ; Diskfree() ' show free space too
End If
```

DISKSIZE

Action

Returns the size of the Disk

Syntax

lSize =**DiskSize** ()

Remarks

lSize	A Long Variable, which is assigned with the capacity of the disk in Bytes

This functions returns the capacity of the disk.

Same Function in QB:

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT

FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN

WRITE , INPUT

ASM

Calls	**_GetDiskSize**	
Input	none	
Output	16-r19: Long-Value of capacity in Bytes	

Example

```
Dim Gbtemp1 As Byte ' scratch byte
Gbtemp1 = Initfilesystem(1) ' we must init the filesystem once
If Gbtemp1 > 0 Then
Print #1 , "Error " ; Gbtemp1
Else
Print #1 , " OK"
Print "Disksize : " ; Disksize() ' show disk size in bytes
Print "Disk free: " ; Diskfree() ' show free space too
End If
```

DISPLAY

Action

Turn LCD display on or off.

Syntax

DISPLAY ON / OFF

Remarks

The display is turned on at power up.

See also

LCD

Example

```
Dim A As Byte
a = 255
Lcd A
Display Off
Wait 1
Display On
End
```

DO-LOOP

Action

Repeat a block of statements until condition is true.

Syntax

DO

statements

LOOP [UNTIL expression **]**

Remarks

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

The DO-LOOP is always performed at least once.

See also

EXIT , WHILE-WEND , FOR-NEXT

Example

```
'--------------------------------------------------------------
' (c) 1999 MCS Electronics
'--------------------------------------------------------------
' file: DO_LOOP.BAS
' demo: DO, LOOP
'--------------------------------------------------------------
Dim a As Byte

A = 1 'assign a var
Do 'begin a do..loop
Print A 'print var
Incr A 'increase by one
Loop Until A = 10 'do until a=10
End

'You can write a never-ending loop with the following code
Do
'Your code goes here
Loop
```

DriveCheck

Action

Checks the Drive, if it is ready for use

Syntax

bErrorCode = **DriveCheck**()

Remarks

BerrorCode	A Byte Variable, which is assigned with the return value of the function

This function checks the drive, if it is ready for use (for example, whether a compact flash card is inserted). The functions returns 0 if the drive can be used, otherwise an errorcode is returned. For Errorcode see section Errorcodes.

See also

DriveReset , DriveInit , DriveGetIdentity , DriveWriteSector , DriveReadSector

ASM

Calls	**_DriveCheck**	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
Dim bError as Byte
bError = DriveCheck()
```

DriveGetIdentity

Action

Gets the Parameter information from the Card/Drive

Syntax

bErrorCode = **DriveGetIdentity** (wSRAMPointer)

Remarks

BErrorCode	A Byte Variable, which is assigned with the errorcode of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which the information of the Drive should be written

TheIdentifyDrive Function gets the parameter information (512 Bytes) from the CompactFlash Memory Card/Drive and writes it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. This information are for examples number of sectors of the card, serial number and so on. Refer to the Card/Drive manual for further information. The functions returns 0 if no error occured. For Errorcode see section Errorcodes.

Note: For meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck, DriveReset , DriveInit , DriveWriteSector , DriveReadSector

ASM

Calls	**_DriveGetIdentity**	
Input		Z: SRAM-Address of buffer *)
Output	r25: Errorcode	C-Flag: Set on Error

**) Please note: This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.*

Example

```
Dim bError as Byte
Dim aBuffer(512) as Byte ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word ' Address-Pointer for write

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(aBuffer(1))

' Now read the parameter Information from CF-Card
bError = DriveGetIdentity( wSRAMPointer)
```

DriveInit

Action

Sets the AVR-Hardware (PORTs, PINs) attached to the Drive and resets the Drive.

Syntax

bErrorCode = **DriveInit** ())

Remarks

BErrorCode	A Byte Variable, which is assigned with the errorcode of the function

Set the Ports and Pins attaching the Drive for Input/Output and give initial values to the output-pins. After that the Drive is reset. Which action is done in this function depends of the drive and its kind of connection to the AVR. The functions returns 0 if no error occured. For Errorcode see section Errorcodes.

See also

DriveCheck, DriveReset , DriveGetIdentity , DriveWriteSector , DriveReadSector

ASM

Calls	**_DriveInit**	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
Dim bError as Byte
bError = DriveInit()
```

DriveReset

Action

Resets the Drive.

Syntax

bErrorCode = **DriveReset** ()

Remarks

BErrorCode	A Byte Variable, which is assigned with the errorcode of the function

This function resets the drive and brings it to an initial state. The functions returns 0 if no error occured. For Errorcode see section Errorcodes.

See also

DriveCheck, DriveInit , DriveGetIdentity , DriveWriteSector , DriveReadSector

ASM

Calls	**_DriveReset**	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
Dim bError as Byte
bError = DriveReset()
```

DriveReadSector

Action

Read a Sector (512 Bytes) from the (Compact Flashcard-) Drive

Syntax

bErrorCode = **DriveReadSector** (wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the errorcode of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which the Sector from the Drive should be written
lSectorNumber	A Long Variable, which give the sectornumber on the drive be transfer.

Reads a Sector (512 Bytes) from the Drive and write it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. The functions returns 0 if no error occured. For Errorcode see section Errorcodes.

Note: wSRAMPointer is not the variable, to which the content of the desired drive-sector should be written, it is the Word-Variable/Value which contains the SRAM address of the range, to which 512 Bytes should be written from the Drive. This gives you the flexibility to read and write every SRAM-Range to and from the drive, even it is not declared as variable. If you know the SRAM-Address (from the compiler report) of a buffer you can pass this value directly, otherwise you can get the adress with the BASCOM-function VARPTR (see example).

See also

DriveCheck, DriveReset , DriveInit , DriveGetIdentity , DriveWriteSector

ASM

Calls	_DriveReadSector	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

*) *Please note: This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.*

Example

```
Dim bError as Byte
Dim aBuffer(512) as Byte ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word ' Address-Pointer for write
Dim lSectorNumber as Long ' Sector Number

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(aBuffer(1))

' Set Sectornumber, sector 32 normally holds the Boot record sector of first partition
lSectorNumber = 32

' Now read in sector 32 from CF-Card
bError = DriveReadSector( wSRAMPointer , lSectorNumber)
' Now Sector number 32 is in Byte-Array bBuffer
```

DriveWriteSector

Action

Write a Sector (512 Bytes) to the (Compact Flashcard-) Drive

Syntax

bErrorCode = **DriveWriteSector** (wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the errorcode of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer), from which the Sector to the Drive should be written
lSectorNumber	A Long Variable, which give the sectornumber on the drive to transfer.

Writes a Sector (512 Bytes) from SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing to the Drive to sectornumber lSectornumber. The functions returns 0 if no error occured. For Errorcode see section Errorcodes.

Note: For meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck, DriveReset , DriveInit , DriveGetIdentity , DriveReadSector

ASM

Calls	**_DriveWriteSector**	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

*) *Please note: This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.*

Example

```
Dim bError as Byte
Dim aBuffer(512) as Byte ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word ' Address-Pointer for read
Dim lSectorNumber as Long ' Sector Number

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(aBuffer(1))

' Set Sectornumber
lSectorNumber = 3

' Now Write in sector 3 from CF-Card
bError = DriveWriteSector( wSRAMPointer , lSectorNumber)
```

DTMFOUT

Action

Sends a DTMF tone to the compare1 output pin of timer 1.

Syntax

DTMFOUT number, duration

DTMFOUT string , duration

Remarks

Number	A variable or numeric constant that is equivalent with the number of your phone keypad.
Duration	Time in mS the tone will be generated.
string	A string variable that holds the digits to be dialed.

The DTMFOUT statement is based on an Atmel application note (314).

It uses TIMER1 to generate the dual tones. As a consequence, timer1 can not be used in interrupt mode by your application. You may use it for other tasks.

Since the TIMER1 is used in interrupt mode you must enable global interrupts with the statement ENABLE INTERRUPTS. The compiler could do this automatic but when you use other interrupts as well it makes more sense that you enable them.

The working range is from 4 MHz to 10 MHz system clock(xtal).

The DTMF output is available on the TIMER1 OCA1 pin. For a 2313 this is PORTB.3.

Take precautions when connecting the output to your telephone line.

Ring voltage can be dangerous!

System Resources used

TIMER1 in interrupt mode

See also

NONE

ASM

The following routine is called from mcs.lib : _DTMFOUT

R16 holds the number of the tone to generate, R24-R25 hold the duration time in mS.

Uses R9,R10,R16-R23

The DTMF table is remarked in the source and shown for completeness, it is generated by the compiler however with taking the used crystal in consideration.

Example

```
'--------------------------------------------------------------------
' DTMFOUT.BAS
' demonstrates DTMFOUT statement based on AN 314 from Atmel
' min osc.freq is 4 MHz, max freq is 10 MHz
'--------------------------------------------------------------------

'since the DTMFOUT statement uses the TIMER1 interrupt you must enable
'global interrupts
'This is not done by the compiler in case you have more ISRs
Enable Interrupts


'the first sample does dtmfout in a loop
Dim Btmp As Byte

Do
' there are 16 dtmf tones
For Btmp = 0 To 15
Dtmfout Btmp , 500 ' dtmf out on PORTB.3 for the 2313 for 500 mS
'output is on the OC1A output pin
Waitms 1000 ' wait 1 sec
Next
Loop
End

'the keypad of most phones looks like this :
'1 2 3 optional are A
'4 5 6 B
'7 8 9 C
'* 0 # D

'the DTMFOUT translates a numeric value from 0-15 into :
' numeric value phone key
' 0 0
' 1 1
' 2 2
' 3 3
' etc.
' 9 9
' 10 *
' 11 #
' 12 A
' 13 B
' 14 C
' 15 D
```

ECHO

Action

Turns the ECHO on or off while asking for serial INPUT.

Syntax

ECHO value

Remarks

Value	ON to enable ECHO and OFF to disable ECHO.

When you use INPUT to retrieve values for variables, all info you type can be echoed back. In this case you will see each character you enter. When ECHO is OFF, you wil not see the characters you enter.

In versions 1.11.6.2 and earlier the ECHO options were controlled by an additional paramter on the INPUT statement line like : INPUT "Hello " , var NOECHO

This would suppress the ECHO of the typed data. The new syntax works by setting ECHO ON and OFF. For backwards compatibility, using NOECHO on the INPUT statement line will also work. In effect it will turn echo off and on automatic.

By default, ECHO is always ON.

See also

INPUT

ASM

The called routines from mcs.lib are _ECHO_ON and _ECHO_OFF

The following ASM is generated when you turn ECHO OFF.

Rcall Echo_Off

This will set bit 3 in R6 that holds the ECHO state.

When you turn the echo ON the following code will be generated

Rcall Echo_On

Example

```
Dim Var As Byte
'turn off echo
Echo Off
'when you enter the info you will not see it
Input Var
'turn it on again
Echo On
'now you will see what you enter !
Input Var
```

ELSE

Action

Executed if the IF-THEN expression is false.

Syntax

ELSE

Remarks

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

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

IF a = 1 THEN

…

ELSEIF a = 2 THEN

..

ELSEIF b1 > a THEN

…

ELSE

…

END IF

See also

IF , END IF , SELECT

Example

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

ENABLE

Action
Enable specified interrupt.

Syntax
ENABLEinterrupt

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt
COMPARE1A,OC1A or COMPARE1, OC1	TIMER1 OUTPUT COMPARE A interrupt In case of only one compare interrupt
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt
SPI	SPI interrupt
URXC	Serial RX complete interrupt
UDRE	Serial data register empty interrupt
UTXC	Serial TX complete interrupt
SERIAL	Disables URXC, UDRE and UTXC
ACI	Analog comparator interrupt
ADC	A/D converter interrupt

By default all interrupts are disabled.
To enable the enabling and disabling of interrupts use ENABLE INTERRUPTS.
Other chips might have additional interrupt sources such as INT2, INT3 etc.

See also
DISABLE

Example
```
Enable Interrupts 'allow interrupts to be set
Enable Timer1 'enables the TIMER1 interrupt
```

END

Action
Terminate program execution.

Syntax
END

Remarks
STOP can also be used to terminate a program.

When an END statement is encountered, all interrupts are disabled and a never-ending loop is generated. When a STOP is encountered the interrupts will not be disabled. Only a never ending loop will be created.

See also
STOP

Example
```
Print "Hello" 'print this
End 'end program execution and disable all interrupts
```

EOF

Action

Returns the End of File Status.

Syntax

bFileEOFStatus =**EOF**(#bFileNumber)

Remarks

bFileEOFStatus	(Byte) A Byte Variable, which issigned with the EOF Status
bFileNumber	(Byte) Number of the opened file

This functions returns information about the End of File Status

Return value	Status
0	NOT EOF
255	EOF

In case of error (invalid filenumber) 255 (EOF) is returned too.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileEOF**	
Input	r24: Filenumber	
Output	r24: EOF Status	r25: Errorcode
	C-Flag: Set on Error	

Example

```
Ff = Freefile() ' get file handle
Open "test.txt" For Input As #ff ' we can use a constant for the file too
Print Lof(#ff) ; " length of file"
Print Fileattr(#ff) ; " file mode" ' should be 1 for input
Do
Line Input #ff , S ' read a line
' line input is used to read a line of text from a file
Print S ' print on terminal emulator
Loop Until Eof(#ff) <> 0
'The EOF() function returns a non-zero number when the end of the file is
reached
'This way we know that there is no more data we can read
Close #ff
```

EXIT

Action

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

Syntax

EXIT FOR
EXIT DO
EXIT WHILE
EXIT SUB
EXIT FUNCTION

Remarks

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

Example

```
'--------------------------------------------------------------
' (c) 2000 MCS Electronics
'--------------------------------------------------------------
' file: EXIT.BAS
' demo: EXIT
'--------------------------------------------------------------
Dim B1 As Byte , A As Byte

B1 = 50 'assign var
For A = 1 To 100 'for next loop
If A = B1 Then 'decision
Exit For 'exit loop
End If
Next
Print "Exit the FOR..NEXT when A was "; A

A = 1
Do
Incr A
If A = 10 Then
Exit Do
End If
Loop
Print "Loop terminated"
End
```

EXP

Action

Returns e(the base of the natural logarithm) to the power of a single variable.

Syntax

Target = **Exp**(source)

Remarks

Target	The single that is assigned with the Exp() of single target.
Source	The source single to get the Exp of.

See also

LOG , LOG10

Example

```
Dim X As Single

X = Exp(1.1 )
Print X
'prints 3.004166124
X = 1.1
X = Exp(x)
Print X
'prints 3.004164931
```

FILEATTR

Action

Returns the file open mode.

Syntax

bFileAttribut =**FileAttr** (bFileNumber)

Remarks

bFileAttribut	(Byte) File open mode, See table
bFileNumber	(Byte) Number of the opened file

This functions returns information about the File open mode

Return value	Open mode
1	INPUT
2	OUTPUT
8	APPEND
32	BINARY

See also

INITF ILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileAttr**	
Input	r24: Filenumber	
Output	24: File open mode	r25: Errorcode
	C-Flag: Set on Error	

Example

```
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string
Close #2
```

FILEDATE

Action

Returns the date of a file

Syntax

sDate =**FileDate** ()

sDate =**FileDate** (file)

Remarks

Sdate	A string variable that is assigned with the date.
File	The name of the file to get the date of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE, GET, PUT , FILELEN , FILETIME , FILEDATETIME , DIR WRITE , INPUT

ASM

Calls	**_FileDateS ; with filename**	**_FileDateS0 ; for current file from DIR()**
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Example

```
Print "File demo"
Print Filelen( "josef.img") ; " length" ' length of file
Print Filetime( "josef.img") ; " time" ' time file was changed
Print Filedate( "josef.img") ; " date" ' file date
```

FILEDATETIME

Action

Returns the file date and time of a file

Syntax

Var = **FileDateTime** ()

Var = **FileDateTime** (file)

Remarks

Var	A string variable or byte array that is assigned with the file datge and time of the specified file
File	The name of the file to get the date time of.

When the target variable is a string, it must be dimensioned with a length of at least 17 bytes.
When the target variable is a byte array, the array size must be at least 6 bytes.

When you use a numeric variable, the internal file date and time format will be used.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT
, FILELEN , FILEDATE , FILETIME , DIR
WRITE , INPUT

ASM

Calls	**_FileDateTimeS**	**_FileDateTimeS0**
Input		
Output		

Calls	**_FileDateTimeB**	**_FileDateTimeB0**
Input		
Output		

Example(partial)

```
' Read and print Directory and show Filename, Date, Time, Size
' for all files matching pStr1 and create/update younger than pDays
Sub Directorylist(pstr1 As String , Byval Pdays As Word)
Local lFileName as String * 12 ' hold file name for print
Local lwCounter as Word , lFileSizeSum as Long ' for summary
Local lwNow as Word , lwDays as Word
Local lSec as Byte , lMin as Byte , lHour as byte , lDay as byte , lMonth as byte , lYear
as byte
print "Listing of all Files matching " ; pStr1 ; " and create/last update date within " ;
pdays ; " days"
lwNow = SysDay()
lwCounter = 0 : lFileSizeSum = 0
lFileName = Dir(pStr1)
While lFileName <> ""
lsec = FileDateTime()
lwDays = lwNow - SysDay (lDay) ' Days between Now and last File Update; uses lDay, lMonth,
lYear
if lwDays <= pDays then ' days smaller than desired with parameter
print lFileName ; FileDate() ; " " ; FileTime() ; " " ; filelen()
incr lwCounter : lFileSizeSum = FileLen() + lFileSizeSum
end if
lFileName = Dir()
WEnd
print lwCounter ; " File(s) found with " ; lFileSizeSum ; " Byte(s)"
End Sub
```

FILELEN

Action

Returns the size of a file

Syntax

lSize =**FileLen** ()

lSize =**FileLen** (file)

Remarks

lSize	A Long Variable, which is assigned with the filesize in bytes of the fle.
File	A string or string constant to get the filelength of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT

, FILEDATE , FILETIME , FILEDATETIME , DIR

WRITE , INPUT

ASM

Calls	**_FileLen**	
Input		
Output		

Example

```
Print "File demo"
Print Filelen( "josef.img") ; " length" ' length of file
Print Filetime( "josef.img") ; " time" ' time file was changed
Print Filedate( "josef.img") ; " date" ' file date
```

FILETIME

Action

Returns the time of a file

Syntax

sTime =**FileTime** ()

sTime =**FileTime** (file)

Remarks

Stime	A string variable that is assigned with the file time.
File	The name of the file to get the time of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT

, FILELEN , FILEDATE , FILEDATETIME , DIR

WRITE , INPUT

ASM

Calls	**_FileTimeS ; with file param**	_FileTimeS0 ; current file
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Example

```
Print "File demo"
Print Filelen( "josef.img") ; " length" ' length of file
Print Filetime( "josef.img") ; " time" ' time file was changed
Print Filedate( "josef.img") ; " date" ' file date
```

FIX

Action

Returns for values greater then zero the next lower value, for values less then zero the next upper value.

Syntax

var =**FIX**(x)

Remarks

Var	A single variable that is assigned with the FIX of variable x.
X	The single to get the FIX of.

See Also

INT , ROUND , SGN

Example

```
'--------------------------------------------------------------------------
-------
' ROUND_FIX_INT.BAS
'--------------------------------------------------------------------------
-------
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

FLUSH

Action

Write current buffer of File to Card and updates Directory

Syntax

Flush #bFileNumber
Flush

Remarks

BFileNumber	Filenumber, which identifies an opened file such as #1 or #ff

This function writes all information of an open file, which is not saved yet to the Disk. Normally the Card is updated, if a file will be closed or changed to another sector.

When no filenumber is specified, all open files will be flushed.

See also

INITFILESYSTEM , OPEN , CLOSE, PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileFlush**	**_FilesAllFlush**
Input	r24: filenumber	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
$include "startup.inc"
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string

Flush #2 ' flush to disk
Close #2
```

FORMAT

Action

Formats a numeric string.

Syntax

target = Format(source, "mask")

Remarks

target	The string that is assigned with the formatted string.
source	The source string that holds the number.
mask	The mask for formatting the string. When spaces are in the mask, leading spaces will be added when the length of the mask is longer than the source string. " " '8 spaces when source is "123" it will be " 123". When a + is in the mask (after the spaces) a leading + will be assigned when the number does not start with the - sign. "+" with number "123" will be "+123". When zero's are provided in the mask, the string will be filled with leading zero;s. " +00000" with 123 will be " +00123" An optional decimal point can be inserted too: "000.00" will format the number 123 to "001.23" Combinations can be made but the order must be : spaces, + , 0 an optional point and zero's.

See also

FUSING

Example

```
'-------------------------------------------------------------------
' (c) 2000 MCS Electronics
' Format.bas
'-------------------------------------------------------------------
Dim S As String * 10
Dim I As Integer

S = "12345"
S = Format(s , "+")
Print S

S = "123"
S = Format(s , "00000")
Print S

S = "12345"
S = Format(s , "000.00")
Print S

S = "12345"
S = Format(s , " +000.00")
Print S

End
```

FOR-NEXT

Action

Execute a block of statements a number of times.

Syntax

FOR var= start **TO** end [**STEP** value]

Note that in 1.11a downto was supported. This has been rewritten for better compatibility.

Remarks

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

For incremental loops, you must use TO.

For decremental loops, you must use a negative step size.

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

EXIT FOR

Example

```
'--------------------------------------------------------------
' (c) 2000 MCS Electronics
'--------------------------------------------------------------
' file: FOR_NEXT.BAS
' demo: FOR, NEXT
'--------------------------------------------------------------
Dim A As Byte , B1 As Byte , C As Integer


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

Print "Now lets count down"
For C = 10 To -5 Step -1
Print "This is C " ; C
Next



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


End
```

FOURTHLINE

Action

Set LCD cursor to the start of the fourth line.

Syntax

FOURTHLINE

Remarks

Only valid for LCD displays with 4 lines.

See also

HOME , UPPERLINE , LOWERLINE , THIRDLINE LOCATE

Example

```
Dim a as byte
a = 255
Lcd A
Fourthline
Lcd A
Upperline
End
```

FRAC

Action

Returns the fraction of a single.

Syntax

var =**FRAC** (single)

Remarks

var	A numeric single variable that is assigned with the fraction of variable single.
single	The single variable to get the fraction of.

The fraction is the right side after the decimal point of a single.

See Also

INT

Example

```
Dim A As Single
A = 9.123
A = Frac(A)
Print A ' prints 0.123
```

FREEFILE

Action

Returns a free Filenumber.

Syntax

bFileNumber = **FreeFile**()

Remarks

bFileNumber	(Byte) Free Filenumber, which can be used for opening next file

This function gives you a free filenumber, which can be used for file – opening statements. In contrast to QB this file numbers start with 128 and goes up to 255. Use range 1 to 127 for user defined filenumbers to avoid filenumber conflicts with the systemnumbers from FreeFile()

This function is implemented for compatility with QB.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_GetFreeFileNumber**	
Input	none	
Output	r24: Filenumber	r25: Errorcode
	C-Flag: Set on Error	

Example

```
Ff = Freefile() ' get file handle
Open "test.txt" For Input As #ff ' we can use a constant for the file too
Print Lof(#ff) ; " length of file"
Print Fileattr(#ff) ; " file mode" ' should be 1 for input
Do
Line Input #ff , S ' read a line
' line input is used to read a line of text from a file
Print S ' print on terminal emulator
Loop Until Eof(ff) <> 0
'The EOF() function returns a non-zero number when the end of the file is
reached
'This way we know that there is no more data we can read
Close #ff
```

FUSING

Action

FUSING returns a formatted string of a single value.

Syntax

target = Fusing(source, "mask")

Remarks

target	The string that is assigned with the formatted string.
source	The source variable of the type SINGLE that will be converted
mask	The mask for formatting the string. The mask is a string constant that always must start with #. After the decimal point you can provide the number of digits you want the string to have: #.### will give a result like 123.456. Rounding is used when you use the # sign. So 123.4567 will be converted into 123.457 When no rounding must be performed, you can use the & sign instead of the # sign. But only after the DP. #.&&& will result in 123.456 when the single has the value 123.4567

When the single is zero, **0.0** will be returned, no matter how the mask is set up.

See also

FORMAT , STR

Example

```
'--------------------------------------------------------------------
' FUSING.BAS
' (c) 2001 MCS ELectronics
'--------------------------------------------------------------------

Dim S As Single , Z As String * 10

'now assign a value to the single
S = 123.45678
'when using str() you can convert a numeric value into a string
Z = Str(s)
Print Z 'prints 123.456779477

Z = Fusing(s , "#.##")

'now use some formatting with 2 digits behind the decimal point with
rounding
Print Fusing(s , "#.##") 'prints 123.46

'now use some formatting with 2 digits behind the decimal point without
```

```
rounding
Print Fusing(s , "#.&&") 'prints 123.45

'The mask must start with #.
'It must have at least one # or & after the point.
'You may not mix & and # after the point.
End
```

GET

Action

Reads a byte from the hardware or software UART.

Reads data from a file opened in BINARY mode.

Syntax

GET #channel, var

GET #channel, var , [pos] [, length]

Remarks

GET in combination with the software/hardware UART is provided for compatibility with BASCOM-8051. It reads one byte

GET in combination with the AVR-DOS filesystem is very flexible and versatile. It works on files opened in BINARY mode and you can reads all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be assigned with the data from the file
Pos	This is an optional parameter that may be used to specify the postion where the reading must start from. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be read from the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte wil be read. When the variable is a word or integer, 2 bytes will be read. When the variable is a long or single, 4 bytes will be read. When the variable is a string, the number of bytes that will be read is equal to the dimensioned size of the string. DIM S as string * 10 , would read 10 bytes.

Note that when you specify the length for a string, the maximum length is 255. The maximum length for a non-string array is 65535.

Example :

GET #1 , var ,,2 ‘ read 2 bytes, start at current position

GET #1, var , PS ‘ start at position stored in long PS

GET #1, var , PS, 2 ‘ start at position stored in long PS and read 2 bytes

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , PUT

FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN

WRITE , INPUT

ASM

current position

Byte:

_FileGetRange_1
Input:
r24: File number
X: Pointer to variable
T-Flag cleared

Word/Integer:

_FileGetRange_2
Input:
r24: File number
X: Pointer to variable
T-Flag cleared

Long/Single:

_FileGetRange_4
Input:
r24: File number
X: Pointer to variable
T-Flag cleared

String (<= 255 Bytes) with fixed length

_FileGetRange_Bytes
Input:
r24: File number
r20: Count of Bytes
X: Pointer to variable
T-Flag cleared

Array (> 255 Bytes) with fixed length

_FileGetRange
Input:
r24: File number
r20/21: Count of Bytes
X: Pointer to variable
T-Flag cleared

Output from all kind of usage:

goto new position first

_FileGetRange_1
Input:
r24: File number
X: Pointer to variable
r16-19 (A): New position (1-based)
T-Flag Set

_FileGetRange_2
Input:
r24: File number
X: Pointer to variable
r16-19 (A): New position (1-based)
T-Flag Set

_FileGetRange_4
Input:
r24: File number
X: Pointer to variable
r16-19 (A): New position (1-based)
T-Flag Set

_FileGetRange_Bytes
Input:
r24: File number
r20: Count of bytes
X: Pointer to variable
r16-19 (A): New position (1-based)
T-Flag Set

_FileGetRange
Input:
r24: File number
r20/21: Count of bytes
X: Pointer to variable
r16-19 (A): New position (1-based)
T-Flag Set

r25: Error Code
C-Flag on Error
X: requested info

Example

```
'for the binary file demo we need some variables of different types
Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp As Long
Dim Stxt As String * 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Seek(#2) ; " = LOC+1"
Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string

Flush #2 ' flush to disk
Close #2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
B = Seek(#2 , L) ' reset is the same as using SEEK #2,L
Get #2 , B ' get the byte
Get #2 , W ' get the word
Get #2 , L ' get the long
Get #2 , Sn ' get the single
Get #2 , Stxt ' get the string
Close #2
```

GETADC

Action

Retrieves the analog value from channel 0-7.

Syntax

var = GETADC(channel)

Remarks

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

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

The pins of the A/D converter input can be used for digital I/O too.

But it is important that no I/O switching is done while using the A/D converter.

Make sure you turn on the AD converter with the START ADC statement or by setting the proper bit in the ADC configuration register.

See also

CONFIG ADC

Example

```
'--------------------------------------------------------------------
' ADC.BAS
' demonstration of GETADC() function for 8535 micro
'--------------------------------------------------------------------
$regfile = "m163def.dat"

'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc

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

Dim W As Word , Channel As Byte

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

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF : AREF, internal reference turned off
'AVCC : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin

'Using the additional param on chip that do not have the internal
reference will have no effect.
```

GETATKBD

Action

Reads a key from a PC AT keyboard.

Syntax

var = GETATKBD()

Remarks

var	The variable that is assigned with the key read from the keyboard. It may be a byte or a string variable. When no key is pressed a 0 will be returned.

The GETAKBD() function needs 2 input pins and a translation table for the keys. You can read more about this at the CONFIG KEYBOARD compiler directive.

The Getatkbd function will wait for a pressed key. When you want to escape from the waiting loop you can set the ERR bit from an interrupt routine for example.

Getatkbd is using 2 bits from register R6 : bit 4 and 5 are used to hold the shift and control key status.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK		KEY	MAKE	BREAK		KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	F0,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	F0,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	F0,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

These are the usable scan codes from the keyboard. If you want to implement F1 , you look at the generated scan code : 05 hex. So in the table, at position 5+1=6, you write the value for F1.

In the sample program below, you can find the value 200. When you now press F1, the value form the table will be used so 200 will be returned.

See also

CONFIG KEYBOARD

Example

```
'-----------------------------------------------------------------------
----
' PC AT-KEYBOARD Sample
' (c) 2000 MCS Electronics
'-----------------------------------------------------------------------
```

```
----
'For this example :
'connect PC AT keyboard clock to PIND.2 on the 8535
'connect PC AT keyboard data to PIND.4 on the 8535

$regfile = "8535def.dat"

'The GetATKBD() function does not use an interrupt.
'But it waits until a key was pressed!

'configure the pins to use for the clock and data
'can be any pin that can serve as an input
'Keydata is the label of the key translation table
Config Keyboard = Pind.2 , Data = Pind.4 , Keydata = Keydata

'Dim some used variables
Dim S As String * 12
Dim B As Byte

'In this example we use SERIAL(COM) INPUT redirection
$serialinput = Kbdinput

'Show the program is running
Print "hello"

Do
'The following code is remarked but show how to use the GetATKBD()
function
' B = Getatkbd() 'get a byte and store it into byte variable
'When no real key is pressed the result is 0
'So test if the result was > 0
' If B > 0 Then
' Print B ; Chr(b)
' End If

'The purpose of this sample was how to use a PC AT keyboard
'The input that normally comes from the serial port is redirected to the
'external keyboard so you use it to type
Input "Name " , S
'and show the result
Print S
Loop
End

'Since we do a redirection we call the routine from the redirection
routine
'
Kbdinput:
'we come here when input is required from the COM port
'So we pass the key into R24 with the GetATkbd function
' We need some ASM code to save the registers used by the function
$asm
push r16 ; save used register
push r25
push r26
push r27

Kbdinput1:
rCall _getatkbd ; call the function
tst r24 ; check for zero
breq Kbdinput1 ; yes so try again
pop r27 ; we got a valid key so restore registers
pop r26
pop r25
pop r16
$end Asm
'just return
Return

'The tricky part is that you MUST include a normal call to the routine
'otherwise you get an error
'This is no clean solution and will be changed
B = Getatkbd()

'This is the key translation table

Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 , 50
, 0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 ,
114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117 ,
55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 ,
112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 , 0
, 0

'shifted keys UPPER case
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 , 0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 , 37
, 0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 ,
40 , 0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 ,
63 , 0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 , 0
, 0
```

GETDSTIP

Action

Returns the IP address of the peer.

Syntax

Result = **GetDSTIP**(socket)

Remarks

Result	A LONG variable that will be assigned with the IP address of the peer or destination IP address.
Socket	The socket number (0-3)

When you are in server mode, it might be desirable to detect the IP address of the connectingclient.

You can use this for logging, security, etc.

The IP number MSB, is stored in the LS byte of the variable.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, CLOSESOCKET , SOCKETLISTEN , GETDSTPORT

Example

```
Dim L as Long
L = GetdstIP(i) ' store current IP number of socket i
```

GETDSTPORT

Action

Returns the port number of the peer.

Syntax

Result = **GetDSTPort**(socket)

Remarks

Result	A WORD variable that is assigned with t he port number of the peer or destination port number.
Socket	The socket number.

When you are in server mode, it might be desirable to detect the port number of the connecting client.

You can use this for logging, security, etc.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, CLOSESOCKET , SOCKETLISTEN , GETDSTIP

Example

```
Dim P as Word
P = GetdstPORT(i) ' store current port number of socket i
```

GETKBD

Action

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

Syntax

var = GETKBD()

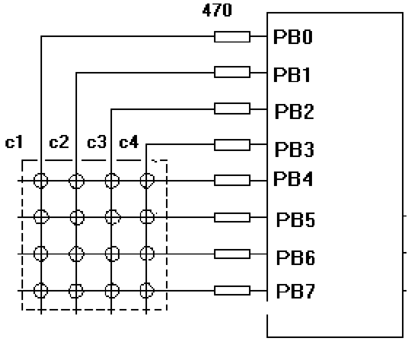
Remarks

Var	The numeric variable that is assigned with the value read from the keyboard

The GETKBD() function can be attached to a port of the uP.

You can define the port with the CONFIG KBD statement.

A schematic for PORTB is shown below



Note that the port pins can be used for other tasks as well. But you might need to set the port direction of those pins after you have used getkbd(). For example the LCD pins are set to output at the start of your program. A call to getkbd() would set the pins to input.

By setting DDR.x register you can set the pins to the proper state again.

As an alternative you can use CONFIG PIN or CONFIG PORT.

When no key is pressed 16 will be returned.

When using the 2 additional rows, 24 will be returned when no key is pressed.

On the STK200 this might not work since other hardware is connected too that interferes.

You can use the Lookup() function to convert the byte into another value. This because the GetKBD() function does not return the same value as the key pressed. It will depend on which keyboard you use.

Sometimes it can happen that it looks like a key is pressed while you do not press a key. This is caused by the scanning of the pins which happens at a very high frequency.

It will depend on the used keyboard. You can add series resistors with a value of 470-1K

The routine will wait for 100 mS by default after the code is retrieved. With CONFIG KBD you can set this delay.

See also

CONFIG KBD

Example

```
'-------------------------------------------------
' GETKBD.BAS
' (c) 2000 MCS Electronics
'-------------------------------------------------
'specify which port must be used
'all 8 pins of the port are used
Config Kbd = Portb

'dimension a variable that receives the value of the pressed key
Dim B As Byte

'loop for ever
Do
B = Getkbd()
'look in the help file on how to connect the matrix keyboard
Print B
'when no key is pressed 16 will be returned
'use the Lookup() function to translate the value to another one
' this because the returned value does not match the number on the
keyboard
Loop
Lcd B
End
```

GETRC

Action

Retrieves the value of a resistor or a capacitor.

Syntax

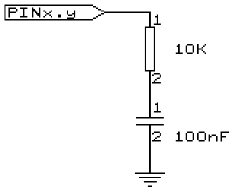
var =**GETRC**(pin , number)

Remarks

Var	The word variable that is assigned with the value.
Pin	The PIN name for the R/C is connection.
Number	The port pin for the R/C is connection.

The name of the input port (PIND for example) must be passed even when all the other pins are configured for output. The pin number must also be passed. This may be a constant or a variable.

A circuit is shown below:



The capacitor is charged and the time it takes to discharge it is measured and stored in the variable. So when you vary either the resistor or the capacitor, different values will be returned. This function is intended to return a relative position of a resistor wiper, not to return the value of the resistor. But with some calculations it can be retrieved.

See also

NONE

Example

```
'-------------------------------------------------------------------------
--
' GETRC.BAS
' demonstrates how to get the value of a resistor
' The library also shows how to pass a variable for use with individual
port
' pins. This is only possible in the AVR architecture and not in the 8051
'-------------------------------------------------------------------------
--
'The function works by charging a capacitor and uncharge it little by
little
'A word counter counts until the capacitor is uncharged.
'So the result is an indication of the position of a pot meter not the
actual
'resistor value

'This example used the 8535 and a 10K ohm variable resistor connected to
PIND.4
'The other side of the resistor is connected to a capacitor of 100nF.
'The other side of the capacitor is connected to ground.
'This is different than BASCOM-8051 GETRC! This because the architecture
is different.

'The result of getrc() is a word so DIM one
Dim W As Word
Do
'the first parameter is the PIN register.
'the second parameter is the pin number the resistor/capacitor is
connected to
'it could also be a variable!
W = Getrc(pind , 4)
Print W
Wait 1
Loop
```

GETRC5

Action

Retrieves the RC5 remote code from a IR transmitter.

Syntax

GETRC5(address, command)

Uses

TIMER0

Remarks

address	The RC5 address
command	The RC5 command.

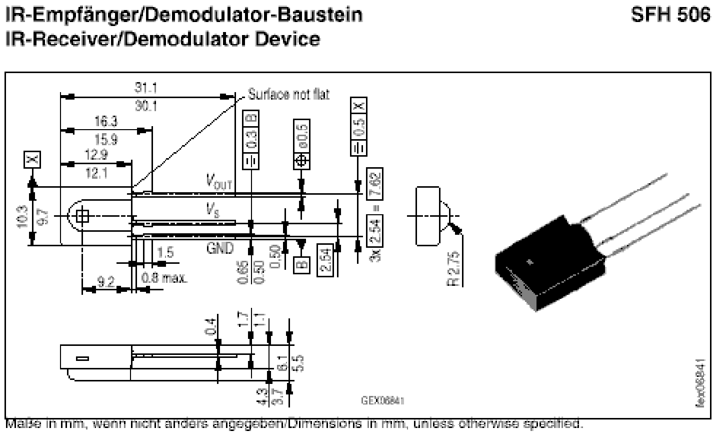
This statement used the AVR 410 application note. Since a timer is needed for accurate delays and background processing TIMER0 is used by this statement.

Also the interrupt of TIMER0 is used by this statement.

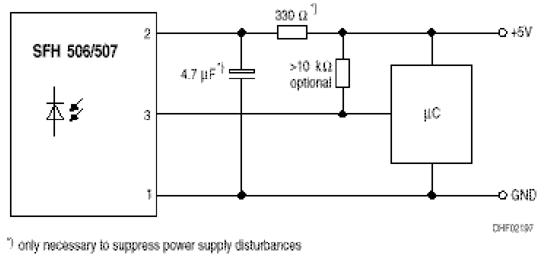
TIMER0 can be used by your application since the values are preserved by the statement but a delay can occur. The interrupt can not be reused.

GETRC5 supports extended RC5 code form version 1.11.6.9 thanks to Gert Boer who implemented the extended RC5 reception.

The SFH506-36 is used from Siemens. Other types can be used as well.



For a good operation use the following values for the filter.



Most audio and video systems are equipped with an infra-red remote control.

The RC5 code is a 14-bit word bi-phase coded signal.

The two first bits are start bits, always having the value 1.

The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remotecontrol transmitter.

Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

For extended RC5 code, the extended bit is bit 6 of the command.

The toggle bit is stored in bit 7 of the command.

See also

CONFIG RC5

Example

```
'-----------------------------------------------------------------------------
'  RC5.BAS
'  (c) 1999-2000 MCS Electronics
'  based on Atmel AVR410 application note
'-----------------------------------------------------------------------------
'use byte library for smaller code
$lib "mcsbyte.lbx"

'This example shows how to decode RC5 remote control signals
'with a SFH506-35 IR receiver.

'Connect to input to PIND.2 for this example
'The GETRC5 function uses TIMER0 and the TIMER0 interrupt.
'The TIMER0 settings are restored however so only the interrupt can not
'be used anymore for other tasks


'tell the compiler which pin we want to use for the receiver input

Config Rc5 = Pind.2

'the interrupt routine is inserted automatic but we need to make it occur
'so enable the interrupts
```

```
Enable Interrupts

'reserve space for variables
Dim Address As Byte , Command As Byte
Print "Waiting for RC5..."

Do
'now check if a key on the remote is pressed
'Note that at startup all pins are set for INPUT
'so we dont set the direction here
'If the pins is used for other input just unremark the next line
'Config Pind.2 = Input
Getrc5(address , Command)

'we check for the TV address and that is 0
If Address = 0 Then
'clear the toggle bit
'the toggle bit toggles on each new received command
Command = Command And &B01111111
Print Address ; " " ; Command
End If
Loop
End
```

GETSOCKET

Action

Creates a socket for TCP/IP communication.

Syntax

Result = **GetSocket** (socket, mode, port, param)

Remarks

Result	A byte that is assigned with the socket number you requested. When the operation fails, it will return 255.
Mode	The socket mode. Use sock_stream(1), sock_dgrm(2), sock_ipl_raw(3), sock) or macl_raw(4). The modes are defined with constants. For TCP/IP communication you need to specify sock_stream or the equivalent value 1. For UDP communication you need to specify sock_dgrm or the equivalent value 2.
Port	This is the local port that will be used for the communication. You may specify any value you like but each socket must have it's own local port number. When you use 0, the value of LOCAL_PORT will be used. LOCAL_PORT is assigned with CONFIG TCPIP. After the assignment, LOCAL_PORT will be increased by 1. So the simplest way is to setup a local port with CONFIG TCPIP, and then use 0 for port.
Param	Optional parameter. Use 0 for default. 128 : send/receive broadcast message in UDP 64 : use register value with designated timeout value 32 : when not using no delayed ack 16: when not using silly window syndrome Consult the W3100A documentation for more information.

After the socket has been initialized you can use SocketConnect to connect to a client, or SocketListen to act as a server.

See also

CONFIG TCPIP, SOCKETCONNECT, SOCKETSTAT ,
TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN

Example

```
I = Getsocket(0 , Sock_stream , 5000 , 0) ' get a new socket
```

GLCDCMD

Action

Sends a command byte to the SED graphical LCD display.

Syntax

GLCDCMD byte

Remarks

byte	A variable or numeric constant to send to the display.

With GLCDCMD you can write command bytes to the display. This is convenient to control the display when there is no specific statement available.

You need to include the glibSED library with :

$LIB "glibsed.lbx"

See also

CONFIG GRAPHLCD , LCDAT, GLCDDATA

Example

```
'-------------------------------------------------------------
```

GLCDDATA

Action

Sends a data byte to the SED graphical LCD display.

Syntax

GLCDDATA byte

Remarks

byte	A variable or numeric constant to send to the display.

With GLCDDATA you can write data bytes to the display. This is convenient to control the display when there is no specific statement available.

You need to include the glibSED library with :

$LIB "glibsed.lbx"

See also

CONFIG GRAPHLCD , LCDAT, GLCDCMD

Example

```
'-------------------------------------------------------------
```

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

```
'-------------------------------------------------------------
' (c) 1999 MCS Electronics
'-------------------------------------------------------------
' file: GOSUB.BAS
' demo: GOTO, GOSUB and RETURN
'-------------------------------------------------------------

Goto Continue
Print "This code will not be executed"

Continue: 'end a label with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End




Routine: 'start a subroutine
Print "This will be executed"
Return 'return from subroutine
```

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

GREY2BIN

Action

Returns the numeric value of a Grey code.

Syntax

var1 = **grey2bin**(var2)

Remarks

var1	Variable that will be assigned with the binary value of the Grey code.
var2	A variable in Grey format that will be converted.

Grey code is used for rotary encoders. Grey2bin() works for byte, integer, word and long variables.

See also

BIN2GREY

ASM

Depending on the data type of the target variable the following routine will be called from mcs.lbx: _Bin2grey for bytes , _Bin2Grey2 for integer/word and _Bin2grey4 for longs.

Example

```
'-----------------------------------------------------------------------
-------
' (c) 2001-2004 MCS Electronics
' This sample show the Bin2Grey and Grey2Bin functions
' Credits to Josef Franz Vögel for an improved and word/long extended
version
'-----------------------------------------------------------------------
-------

'Bin2Gey() converts a byte,integer,word or long into grey code.
'Grey2Bin() converts a grey code into a binary value

Dim B As Byte ' could be word,integer or long too

Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
Print B ; Spc(10) ; Bin2grey(b)
Next

Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
Print B ; Spc(10) ; Grey2bin(b)
Next

End
```

HEX

Action

Returns a string representation of a hexadecimal number.

Syntax

var = Hex(x)

Remarks

var	A string variable.
X	A numeric variable of data type Byte, Integer, Word, Long or Single.

See also

HEXVAL , VAL , STR , BIN

Example

```
Dim A As Byte , S As String * 2 , Sn As Single
a = 123
s = Hex(a)
Print s
Print Hex(a)
Sn = 1.2
Print Hex(sn)
End
```

HEXVAL

Action

Convert string representing a hexadecimal number into a numeric variable.

Syntax

var = **HEXVAL(** x **)**

Remarks

Var	The numeric variable that must be assigned.
X	The hexadecimal string that must be converted.

Difference with QB

In QB you can use the VAL() function to convert hexadecimal strings.

But since that would require an extra test for the leading &H signs that are required in QB, a separate function was designed.

See also

HEX , VAL , STR , BIN

Example

```
Dim A As Byte , S As String * 2 , Sn As Single
S = "A"
A = Hexval(s)
Print A ; Spc(10) ; Hex(a)
End
```

HIGH

Action

Retrieves the most significant byte of a variable.

Syntax

var = HIGH(s)

Remarks

Var	The variable that is assigned with the MSB of var S.
S	The source variable to get the MSB from.

See also

LOW , HIGHW

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = High(i) ' is 10 hex or 16 dec
End
```

HIGHW

Action

Retrieves the most significant word of a long variable.

Syntax

var =**HIGHW**(s)

Remarks

Var	The variable that is assigned with the MS word of var S.
S	The source variable to get the MSB from.

There is no LowW() function. This because when you assign a Long to a word or integer, only the lower part is assigned. For this reason you do not need a Loww() function.

See also

LOW , HIGH

Example

```
Dim X As Word , L As Long
L = &H12345678
X = Highw(l)
Print Hex(x)
```

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

Example

```
Cls
Lowerline
Lcd "Hello"
Home Upper
Lcd "Upper"
End
```

I2CINIT

Action

Initializes the SCL and SDA pins.

Syntax

I2CINIT

Remarks

By default the SCL and SDA pins are in the right state when you reset the chip. Both the PORT and the DDR bits are set to 0 in that case.

When you need to change the DDR and/or PORT bits you can use I2CINIT to bring the pins in the proper state again.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CSEND , I2CSTART , I2CSTOP , I2CRBYTE , I2CWBYTE

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
I2CINIT
Dim X As Byte , Slave As Byte
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
```

I2CRECEIVE

Action

Receives data from an I2C serial device.

Syntax

I2CRECEIVE slave, var

I2CRECEIVE slave, var ,b2W, b2R

Remarks

Slave	A byte, Word/Integer variable or constant with the slave address from the I2C-device.
Var	A byte or integer/word variable that will receive the information from the I2C-device.
b2W	The number of bytes to write. Be cautious not to specify too many bytes!
b2R	The number of bytes to receive. Be cautious not to specify too many bytes!

You can specify the base address of the slave chip because the read/write bit is set/reset by the software.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CSEND , I2CSTART , I2CSTOP , I2CRBYTE , I2CWBYTE

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , Slave As Byte
X = 0 'reset variable
Slave = &H40 'slave address of a PCF 8574 I/O IC
I2creceive Slave , X 'get the value
Print X 'print it


Dim Buf(10) As Byte
Buf(1) = 1 : Buf(2) = 2
I2creceive Slave , Buf(1) , 2 , 1 'send two bytes and receive one byte
Print Buf(1) 'print the received byte

End
```

I2CSEND

Action

Send data to an I2C-device.

Syntax

I2CSEND slave, var

I2CSEND slave, var , bytes

Remarks

Slave	The slave address off the I2C -device.
Var	A byte, integer/word or numbers that holds the value, which will be, send to the I2C-device.
Bytes	The number of bytes to send.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CRECEIVE , I2CSTART , I2CSTOP , I2CRBYTE , I2CWBYTE

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , A As Byte , Bytes As Byte
x = 5 'assign variable to 5
Dim Ax(10) As Byte
Const Slave = &H40 'slave address of a PCF 8574 I/O IC
I2csend Slave , X 'send the value or


For a = 1 to 10
ax(a) = a 'Fill dataspace
Next
Bytes = 10
I2csend Slave , Ax(1) , Bytes
END
```

I2START,I2CSTOP, I2CRBYTE, I2CWBYTE

Action

I2CSTART generates an I2C start condition.

I2CSTOP generates an I2C stop condition.

I2CRBYTE receives one byte from an I2C-device.

I2CWBYTE sends one byte to an I2C-device.

Syntax

I2CSTART

I2CSTOP

I2CRBYTE var, ack/nack

I2CWBYTE val

Remarks

Var	A variable that receives the value from the I2C-device.
ack/nack	Specify ACK if there are more bytes to read. Specify NACK if it is the last byte to read.
Val	A variable or constant to write to the I2C -device.

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

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CSEND , I2CRECEIVE , I2CSTART , I2CSTOP , I2CRBYTE , I2CWBYTE

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
'-------- Writing and reading a byte to an EEPROM 2404 -----------------
Dim A As Byte
Const Adresw = 174 'write of 2404
Const Adresr = 175 'read address of 2404
I2cstart 'generate start
I2cwbyte Adresw 'send slave address
I2cwbyte 1 'send address of EEPROM
I2cwbyte 3 'send a value
I2cstop 'generate stop
Waitms 10 'wait 10 mS because that is the time that the chip needs to
```

```
write the data

'----------------now read the value back into the var a-----------------
--
I2cstart 'generate start
I2cwbyte Adresw 'write slave address
I2cwbyte 1 'write address of EEPROM to read
I2cstart 'generate repeated start
I2cwbyte Adresr 'write slave address of EEPROM
I2crbyte A , Nack 'receive value into a. nack means last byte to receive
I2cstop 'generate stop
Print A 'print received value
End
```

IDLE

Action

Put the processor into the idle mode.

Syntax

IDLE

Remarks

In the idle mode, the system clock is removed from the CPU but not from the interrupt logic, the serial port or the timers/counters.

The idle mode is terminated either when an interrupt is received(from the watchdog, timers, external level triggered or ADC) or upon system reset through the RESET pin.

See also

POWERDOWN

Example

```
IDLE
```

IF-THEN-ELSE-END 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.

The one line version of IF can be used :
IF expression THEN statement [ELSE statement]
The use of [ELSE] is optional.

Tests like IF THEN can also be used with bits and bit indexes.
IF var.bit = 1 THEN
^--- bit is a variable or numeric constant in the range from 0-255

```
Dim Var As Byte , Idx As Byte
Var = 255
Idx = 1
If Var .idx = 1 Then
Print "Bit 1 is 1"
End If
```

See also

ELSE

Example

```
Dim A As Integer
A = 10
If A = 10 Then 'test expression
Print "This part is executed." 'this will be printed
Else
Print "This will never be executed." 'this not
End If
If A = 10 Then Print "New in BASCOM"
If A = 10 Then Goto Label1 Else Print "A<>10"
Label1:

Rem The following example shows enhanced use of IF THEN
If A.15 = 1 Then 'test for bit
Print "BIT 15 IS SET"
End If
Rem the following example shows the 1 line use of IF THEN [ELSE]
If A.15 = 0 Then Print "BIT 15 is cleared" Else Print "BIT 15 is set"
```

INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

Var	Any numeric variable.

See also

DECR

Example

```
Dim A As Byte
Do 'start loop
Incr A 'increment a by 1
Print A 'print a
Loop Until A > 10 'repeat until a is greater than 10
Print A
```

InitFileSystem

Action

Initialize the file system

Syntax

bErrorCode = **InitFileSystem** (bPartitionNumber)

Remarks

bErrorCode	(Byte) Error Result from Routine, Returns 0 if no Error
bPartitionNumber	(Byte) Partitionnumber on the Flashcard Drive (normally 1)

Reads the Master boot record and the partition boot record (Sector) from the flashcard and initializes the filesystem.

This function must be called before any other file-system function is used.

See also

OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	**_GetFileSystem**	
Input	r24: partitionnumber (1-based)	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
Dim bErrorCode as Byte
bErrorCode = InitFileSystem(1)
If bErrorCode > 0 then
Print "Error: "; bErrorCode
Else
Print "Filesystem successfully initialized"
End If
```

INITLCD

Action

Initializes the LCD display.

Syntax

INITLCD

Remarks

The LCD display is initialized automatic at start up when LCD statements are used by your code.
If fore some reason you would like to initialize it again you can use the INITLCD statement.

ASM

The generated ASM code :

Rcall _Init_LCD

See also

LCD

Example

```
' ----------------------------------------------------------
```

INKEY

Action

Returns the ASCII value of the first character in the serial input buffer.

Syntax

var = **INKEY()**

var = **INKEY(#channel)**

Remarks

Var	Byte, Integer, Word, Long or String variable.
Channel	A constant number that identifies the opened channel if software UART mode

If there is no character waiting, a zero will be returned.

Use the IsCharWaiting() function to check if there is a byte waiting.

The INKEY routine can be used when you have a RS-232 interface on your uP.

The RS-232 interface can be connected to a comport of your computer.

See also

WAITKEY , ISCHARWAITING

Example

```
Dim A As Byte
Do 'start loop
A = Inkey() 'look for character
If A > 0 Then 'is variable > 0?
Print A 'yes , so print it
End If
Loop 'loop forever
'The example above is for the HARDWARE UART

'The OPEN.BAS sample contains a sample for use with the software UART.
```

INP

Action

Returns a byte read from a hardware port or any internal 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. (0- **&H**FFFF)

The PEEK() function will read only the lowest 32 memory locations (registers).

The INP() function can read from any memory location since the AVR has a linear memory model.

When you want to read from XRAM memory you must enable external memory access in the Compiler Chip Options.

See also

OUT PEEK

Example

```
Dim A As Byte
A = Inp(&H8000) 'read value that is placed on databus(d0-d7) at hex
address 8000
Print A
End
```

INPUTBIN

Action

Read binary data from the serial port.

Syntax

INPUTBIN var1 [,var2]

INPUTBIN #channel , var1 [,var2]

Remarks

var1	The variable that is assigned with the characters from the serial port.
var2	An optional second (or more) variable that is assigned with the data from the serial input stream.

The channel is for use with the software UART routine and must be used with OPEN and CLOSE.

The number of bytes to read depends on the variable you use.

When you use a byte variable, 1 character is read from the serial port.

An integer will wait for 2 characters and an array will wait until the whole array is filled.

Note that the INPUTBIN statement doesn't wait for a <RETURN> but just for the number of bytes.

You may also specify an additional numeric parameter that specifies how many bytes will be read. This is convenient when you are filling an array.

Inputbin ar(1) , 4 ' will fill 4 bytes starting at index 1.

See also

PRINTBIN

Example

```
Dim A As Byte , C As Integer
Inputbin A , C 'wait for 3 characters
End
```

INPUTHEX

Action

Allows hexadecimal input from the keyboard during program execution.

Syntax

INPUTHEX [" prompt"] , var [, varn]

Remarks

prompt	An optional string constant printed before the prompt character.
Var,varn	A numeric variable to accept the input value.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as input device.
You can also use the build in terminal emulator.
The input entered may be in lower or upper case (0-9 and A-F)

If var is a byte then the input can be maximum 2 characters long.
If var is an integer/word then the input can be maximum 4 characters long.
If var is a long then the input can be maximum 8 characters long.

Difference with QB

In QB you can specify &H with INPUT so QB will recognize that a hexadecimal string is being used.
BASCOM implements a new statement: INPUTHEX.

See also

INPUT , ECHO

Example

```
Dim X As Byte
Echo On
Inputhex "Enter a number " , X 'ask for input like AF
Echo Off
Inputhex "Enter a number " , X 'ask for input like ab
Echo On
End
```

INPUT

Action

Allows input from the keyboard during program execution.
Reads data from a file

Syntax

INPUT [" prompt"] , var[, varn]
INPUT #ch, var[, varn]

Remarks

Prompt	An optional string constant printed before the prompt character.
Var,varn	A variable to accept the input value or a string.
Ch	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.

The INPUT routine can be used when you have an RS-232 interface on your uP.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as an input device.
You can also use the built-in terminal emulator.

For usage with AVR-DOS file system, you can read variables from an opened file. Since these variables are stored in ASCII format, the data is converted to the proper format automaticly.
When you use INPUT with a file, the prompt is not supported.

Difference with QB

In QB you can specify &H with INPUT so QB will recognize that a hexadecimal string is being used.
BASCOM implements a new statement : INPUTHEX.

See also

INPUTHEX , PRINT , ECHO , WRITE

Example

```
'--------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'--------------------------------------------------------------
' file: INPUT.BAS
' demo: INPUT, INPUTHEX
'--------------------------------------------------------------
'To use another baudrate and crystalfrequency use the
'metastatements $BAUD = and $CRYSTAL =
$baud = 9600 'try 1200 baud for example
$crystal = 4000000 '4 MHz
```

```
Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15

Input "Use this to ask a question " , V
Input B1 'leave out for no question

Input "Enter integer " , C
Print C


Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho 'supress echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho 'without echo
Print S
End

Dim X As Byte
Echo On
Inputhex "Enter a number " , X 'ask for input
Echo Off
Inputhex "Enter a number " , X 'ask for input
Echo On
End
```

INSTR

Action

Returns the position of a sub string in a string.

Syntax

var =**INSTR(** start , string , substr **)**

var =**INSTR(** string , substr **)**

Remarks

Var	Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found.
Start	An optional numeric parameter that can be assigned with the first position where must be searched in the string. By default (when not used) the whole string is searched starting from position 1.
String	The string to search.
Substr	The search string.

No constant can be used for string it must be a string.

Only substr can be either a string or a constant.

See also

NONE

Example

```
Dim S As String * 20 , 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
```

INT

Action

Returns the integer part of a single.

Syntax

var =**INT**(single)

Remarks

Var	A numeric variable that is assigned with the integer of variable single.
Single	The single variable to get the integer of.

The fraction is the right side after the decimal point of a single.
The integer is the left side before the decimal point.
1234.567 1234 is the integer part, .567 is the fraction

See Also

FRAC , FIX , ROUND

Example

```
'--------------------------------------------------------------------------
-------
' ROUND_FIX_INT.BAS
'--------------------------------------------------------------------------
-------
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

IP2STR

Action

Convert an IP number into it's string representation.

Syntax

Var = IP2STR(num)

Remarks

An IP number is represented with dots like 192.168.0.1.
The IP2STR function converts an IP number into a string.
This function is intended to be used in combination with the TCP/IP routines.

Var	The string variable that is assigned with the IP number
Num	A variable that contains the ip number is numeric format.

See also

CONFIG TCPIP

ISCHARWAITING

Action

Returns 1 when a character is waiting in the hardware UART buffer.

Syntax

var = **ISCHARWAITING()**

var = **ISCHARWAITING(#channel)**

Remarks

Var	Byte, Integer, Word or Long variable.
Channel	A constant number that identifies the opened channel.

If there is no character waiting, a zero will be returned.

If there is a character waiting, a one (1) will be returned.

The character is not retrieved or altered by the function.

While the Inkey() will get the character from the HW UART when there is a character in the buffer, it will return a zero when the character is zero. This makes it unusable to work with binary data that might contain the value 0.

With IsCharWaiting() you can first check for the presence of a character and when the function returns 1, you can retrieve the character with Inkey or Waitkey.

See also

WAITKEY , INKEY

Example

```
'--------------------------------------------------------------
' (c) 1997-2004 MCS Electronics
'--------------------------------------------------------------
' file: INKEY.BAS
' demo: INKEY , WAITKEY
'--------------------------------------------------------------
Dim A As Byte , S As String * 2
Do
A = Inkey() 'get ascii value from serial port
's = Inkey()
If A > 0 Then 'we got something
Print "ASCII code " ; A ; " from serial"
End If
Loop Until A = 27 'until ESC is pressed

A = Waitkey() 'wait for a key
's = waitkey()
Print Chr(a)

'wait until ESC is pressed
Do
Loop Until Inkey() = 27

'When you need to receive binary data and the bibary value 0 ,
'you can use the IScharwaiting() function.
'This will return 1 when there is a char waiting and 0 if there is no
char waiting.
'You can get the char with inkey or waitkey then.
End
```

KILL

Action

Delete a file from the Disk

Syntax

Kill sFileName

Remarks

sFileName	A String variable or string expression, which denotes the file to delete

This function deletes a file from the disk. A file in use can't be deleted. WildCards in Filename are not supported. Check the DOS-Error in variable gDOSError.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , DISKFREE , DISKSIZE, GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_DeleteFile**	
Input	X: Pointer to string with filename	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'We can use the KILL statement to delete a file.
'A file mask is not supported
Print "Kill (delete) file demo"
Kill "test.txt"
```

LCASE

Action

Converts a string in to all lower case characters.

Syntax

Target =**Lcase**(source)

Remarks

Target	The string that is assigned with the lower case string of string target.
Source	The source string.

See also

UCASE

ASM

The following ASM routines are called from MCS.LIB : _LCASE

The generated ASM code : (can be different depending on the micro used)

```
;##### Z = Lcase(s)
Ldi R30,$60
Ldi R31,$00 ; load constant in register
Ldi R26,$6D
Rcall _Lcase
```

Example

```
Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

LCD

Action

Send constant or variable to LCD display.

Syntax

LCD x

Remarks

X	Variable or constant to display.

More variables can be displayed separated by the **;** -sign

LCD a ; b1 ; "constant"

The LCD statement behaves just like the PRINT statement. So SPC() can be used too.

See also

$LCD , $LCDRS , CONFIG LCD

Example

```
'--------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'--------------------------------------------------------------
' file: LCD.BAS
' demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
' CURSOR, DISPLAY
'--------------------------------------------------------------

$sim
'REMOVE the above command for the real program !!
'$sim is used fr faster simulation


'note : tested in PIN mode with 4-bit

'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7
= Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

Rem with the config lcdpin statement you can override the compiler
settings


Dim A As Byte
Config Lcd = 16 * 2 'configure lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines

'$LCD = address will turn LCD into 8-bit databus mode
' use this with uP with external RAM and/or ROM
' because it aint need the port pins !

Cls 'clear the LCD display
Lcd "Hello world." 'display this at the top line
Wait 1
Lowerline 'select the lower line
Wait 1
Lcd "Shift this." 'display this at the lower line
Wait 1
For A = 1 To 10
Shiftlcd Right 'shift the text to the right
Wait 1 'wait a moment
Next

For A = 1 To 10
Shiftlcd Left 'shift the text to the left
Wait 1 'wait a moment
Next

Locate 2 , 1 'set cursor position
Lcd "*" 'display this
Wait 1 'wait a moment

Shiftcursor Right 'shift the cursor
Lcd "@" 'display this
Wait 1 'wait a moment

Home Upper 'select line 1 and return home
Lcd "Replaced." 'replace the text
Wait 1 'wait a moment

Cursor Off Noblink 'hide cursor
Wait 1 'wait a moment
Cursor On Blink 'show cursor
Wait 1 'wait a moment
Display Off 'turn display off
Wait 1 'wait a moment
Display On 'turn display on
'-----------------NEW support for 4-line LCD------
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third 'goto home on line three
Home Fourth
Home F 'first letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1

'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line

Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 ' replace ?
with number (0-7)
```

```
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 ' replace ?
with number (0-7)
Cls 'select data RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1) 'print the special character

'----------------- Now use an internal routine ------------
_temp1 = 1 'value into ACC
!rCall _write_lcd 'put it on LCD
End
```

LCDCONTRAST

Action

Set the contrast of a TEXT LCD.

Syntax

LCDCONTRAST x

Remarks

X	A variable or constant in the range from 0-3.

Some displays support changing the contrast. Noritake displays have this option for example.

See also

NONE

Example

NONE

LCDAT

Action

Send constant or variable to a SED graphical display.

Syntax

LCDAT x **,** y , var [, inv]

Remarks

X	X location. In the range from 0-63. The SED displays columns are 1 pixel width.
Y	Y location.
Var	The constant or variable to display
inv	Optional number. Value 0 will show the data normal. Any other value will invert the data.

You need to include the glibSED library with :

$LIB "glibsed.lbx"

See also

CONFIG GRAPHLCD , SETFONT, GLCDCMD, GLCDDATA

Example

```
'-------------------------------------------------------------
```

LEFT

Action

Return the specified number of leftmost characters in a string.

Syntax

var = **Left(**var1 , n **)**

Remarks

Var	The string that is assigned.
Var1	The source string.
n	The number of characters to get from the source string.

See also

RIGHT , MID

Example

```
Dim S As Xram String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Left(s , 5)
Print Z 'ABCDE
End
```

LEN

Action

Returns the length of a string.

Syntax

var =**LEN** (string)

Remarks

var	A numeric variable that is assigned with the length of string.
string	The string to calculate the length of.

Strings can be maximum 254 bytes long.

Example

```
Dim S As String * 12
Dim A As Byte
S = "test"
A = Len(s)
Print A ' prints 4
Print Len(s)
```

LINE

Action

Draws a line on a graphic display.

Syntax

LINE(x0,y0) – (x1,y1), color

Remarks

X0	Starting horizontal location of the line.
Y0	Starting vertical location of the line.
X1	Horizontal end location of the line
Y1	Vertical end location of the line.

See Also

LINE , CONFIG GRAPHLCD

Example

```
'-----------------------------------------------------------------
' (c) 2001-2004 MCS Electronics
' T6963C graphic display support demo 240 * 128
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin connected to
' 1 GND GND
'2 GND GND
'3 +5V +5V
'4 -9V -9V potmeter
'5 /WR PORTC.0
'6 /RD PORTC.1
'7 /CE PORTC.2
'8 C/D PORTC.3
'9 NC not conneted
'10 RESET PORTC.4
'11-18 D0-D7 PA
'19 FS PORTC.5
'20 NC not connected

$crystal = 8000000

'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns
```

```
'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part

Cursor Off

Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30


Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"

Wait 2

Cls Text


'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255 ' diagonal line
Line(0 , 127) -(239 , 0) , 255 ' diagonal line
Line(0 , 0) -(240 , 0) , 255 ' horizontal upper line
Line(0 , 127) -(239 , 127) , 255 'horizontal lower line
Line(0 , 0) -(0 , 127) , 255 ' vertical left line
Line(239 , 0) -(239 , 127) , 255 ' vertical right line


Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
Pset X , 20 , 255 ' set the pixel
Next

For X = 0 To 140
Pset X , 127 , 255 ' set the pixel
Next

Wait 2


'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje ' show 2 since we have a big display
Wait 2
Cls Text ' clear the text
End



'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

LINE INPUT

Action

Read a Line from an opened File.

Syntax

LineInput #bFileNumber, sLineText

Remarks

BfileNumber	(Byte) Filenumber, which identifies an opened file
SlineText	(String) A string, which is assigned with the next line from the file.

Only valid for files opened in mode INPUT. Line INPUT works only with strings. It is great for working on text files.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT
FILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileLineInput**	
Input	r24: filenumber	X: Pointer to String to be written fromfile
	r25: Stringlength	
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'Ok we want to check if the file contains the written lines
Ff = Freefile() ' get file handle
Open "test.txt" For Input As #ff ' we can use a constant for the file too
Print Lof(#ff) ; " length of file"
Print Fileattr(#ff) ; " file mode" ' should be 1 for input
Do
Line Input #ff , S ' read a line
' line input is used to read a line of text from a file
Print S ' print on terminal emulator
Loop Until Eof(ff) <> 0
'The EOF() function returns a non-zero number when the end of the file is
reached
'This way we know that there is no more data we can read
Close #ff
```

LTRIM

Action

Returns a copy of a string with leading blanks removed

Syntax

var =**LTRIM**(org)

Remarks

Var	String that receives the result.
Org	The string to remove the leading spaces from

See also

RTRIM , TRIM

ASM

NONE

Example

```
Dim S As String * 6
S = " AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

LOAD

Action

Load specified TIMER with a reload value.

Syntax

LOAD TIMER , value

Remarks

TIMER	TIMER0 , TIMER1 or TIMER2
Value	The variable or value to load.

The TIMER0 does not have a reload mode. But when you want the timer to generate an interrupt after 10 ticks for example, you can use the RELOAD statement.

It will do the calculation. (256-value)

So LOAD TIMER0, 10 will load the TIMER0 with a value of 246 so that it will overflow after 10 ticks.

TIMER1 is a 16 bit counter so it will be loaded with the value of 65536-value.

LOADADR

Action

Loads the address of a variable into a register pair.

Syntax

LOADADR var , reg

Remarks

var	A variable which address must be loaded into the register pair X, Y or Z.
reg	The register X, Y or Z.

The LOADADR statement serves as an assembly helper routine.

Example

```
Dim S As String * 12
Dim A As Byte
$ASM
loadadr S , X 'load address into R26 and R27
ld _temp1, X 'load value of location R26/R27 into R24(_temp1)
$END ASM
```

LOADLABEL

Action

Assigns a word variable with the address of a label.

Syntax

Var = LOADLABEL(label **)**

Remarks

var	The variable that is assigned with the address of the label.
lbl	The name of the label

In some cases you might need to know the address of a point in your program. To perform a Cpeek() for example.

You can place a label at that point and use LoadLabel to assign the address of the label to a variable.

LOC

Action

Returns the position of last read or written Byte of the file

Syntax

lLastReadWritten =**Loc** (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
lLastReadWritten	(Long) Variable, whichsigned with the Position of last read or written Byte (1-based)

This function returns the position of the last read or written Byte. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError. If the file position pointer is changed with the command SEEK, this function can not be used till the next read/write operation.

Difference with QB

This function differs from QB. In QB the byte position is divided by 128.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT

FILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileLoc**	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string
```

```
Flush #2 ' flush to disk
Close #2
```

LOF

Action

Returns the length of the File in Bytes

Syntax

lFileLength =**LOF** (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
LFileLength	(Long) Variable, which issigned with the Length of the file (1-based)

This function returns the length of an opened file. If an error occures, 0 is returned. Check DOS-Error in variable gbDOSError.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Calls	**_FileLOF**	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string

Flush #2 ' flush to disk
Close #2
```

LOCAL

Action

Dimensions a variable LOCAL to the function or sub program.

Syntax

LOCAL var **As** Type

Remarks

Var	The name of the variable
Type	The data type of the variable.

There can be only LOCAL variables of the type BYTE, INTEGER, WORD, LONG, SINGLE or STRING.

A LOCAL variable is a temporary variable that is stored on the frame.
When the SUB or FUNCTION is terminated, the memory will be released back to the frame.
BIT variables are not possible because they are GLOBAL to the system.

The AT , ERAM, SRAM, XRAM directives can not be used with a local DIM statement. Also local arrays are not possible.

See also

DIM

ASM

NONE

Example

```
'-----------------------------------------------------
' (c) 2000 MCS Electronics
' DECLARE.BAS
' Note that the usage of SUBS works different in BASCOM-8051
'-----------------------------------------------------
' First the SUB programs must be declared

'Try a SUB without parameters
Declare Sub Test2

'SUB with variable that can not be changed(A) and
'a variable that can be changed(B1), by the sub program
'When BYVAL is specified, the value is passed to the subprogram
'When BYREF is specified or nothing is specified, the address is passed
to
'the subprogram

Declare Sub Test(byval A As Byte , B1 As Byte)
Declare Sub Testarray(byval A As Byte , B1 As Byte)
'All variable types that can be passed
'Notice that BIT variables can not be passed.
'BIT variables are GLOBAL to the application
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S
As String)

'passing string arrays needs a different syntax because the length of the
strings must be passed by the compiler
'the empty () indicated that an array will be passed
Declare Sub Teststr(b As Byte , Dl() As String)

Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String * 10
'dim used variables
Dim Ar(10) As Byte
Dim Sar(10) As String * 8 'strng array

For Bb = 1 To 10
Sar(bb) = Str(bb) 'fill the array
Next
Bb = 1
'now call the sub and notice that we always must pass the first address
with index 1
Call Teststr(bb , Sar(1))


Call Test2 'call sub
Test2 'or use without CALL
'Note that when calling a sub without the statement CALL, the enclosing
parentheses must be left out
Bb = 1
Call Test(1 , Bb) 'call sub with parameters
Print Bb 'print value that is changed

'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S

'now pass an array
'note that it must be passed by reference
Testarray 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
Print "ar(3) = " ; Ar(3)
End

'End your code with the subprograms
'Note that the same variables and names must be used as the declared ones

Sub Test(byval A As Byte , B1 As Byte) 'start sub
Print A ; " " ; B1 'print passed variables
B1 = 3 'change value
'You can change A, but since a copy is passed to the SUB,
'the change will not reflect to the calling variable
End Sub

Sub Test2 'sub without parameters
Print "No parameters"
End Sub


Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
Local X As Byte
```

```
X = 5 'assign local

B = X
I = -1
W = 40000
L = 20000
S = "test"
End Sub

Sub Testarray(byval A As Byte , B1 As Byte) 'start sub
Print A ; " " ; B1 'print passed variables
B1 = 3 'change value of element with index 1
B1(1) = 3 'specify the index which does the same as the line above
B1(3) = 3 'modify other element of array
'You can change A, but since a copy is passed to the SUB,
'the change will not reflect to the calling variable
End Sub

'notice the empty() to indicate that a string array is passed
Sub Teststr(b As Byte , Dl() As String)
Dl(b) = Dl(b) + "add"
End Sub
```

LOCATE

Action

Moves the LCD cursor to the specified position.

Syntax

LOCATE y , x

Remarks

X	Constant or variable with the position. (1-64*)
Y	Constant or variable with the line (1 - 4*)

* Depending on the used display

See also

CONFIGLCD , LCD , HOME , CLS

Example

```
LCD "Hello"
Locate 1,10
LCD "*"
```

LOG

Action

Returns the natural logarithm of a single variable.

Syntax

Target = **Log**(source)

Remarks

Target	The single that is assigned with the LOG() of single target.
Source	The source single to get the LOG of.

See also

EXP , LOG10

Example

Show sample

LOG10

Action

Returns the base 10 logarithm of a single variable.

Syntax

Target = **Log10**(source)

Remarks

Target	The single that is assigned with the base 10 logarithm of single target.
Source	The source single to get the base 10 LOG of.

See also

EXP , LOG

Example

Show sample

LOOKDOWN

Action

Returns the index of a series of data.

Syntax

var =**LOOKDOWN(** value, label, entries **)**

Remarks

Var	The returned index value
Value	The value to search for
Label	The label where the data starts
entries	The number of entries that must be searched

When you want to look in BYTE series the VALUE variable must be dimensioned as a BYTE. When you want to look in INTEGER or WORD series the VALUE variable must be dimensioned as an INTEGER.

The LookDown function is the counterpart of the LookUp function.

Lookdown will search the data for a value and will return the index when the value is found. It will return –1 when the data is not found.

See also

LOOKUPSTR , LOOKUP

Example

```
'-----------------------------------------------------------
' LOOKDOWN.BAS
' (c) 2001 MCS Electronics
'-----------------------------------------------------------

Dim Idx as integer, search as byte,entries as byte

'we want to search for the value 3
Search = 3
'there are 5 entries in the table
Entries = 5

'lookup and return the index
Idx = Lookdown(search , Label , Entries)
Print Idx

Search = 1
Idx = Lookdown(search , Label , Entries)
Print Idx


Search = 100
Idx = Lookdown(search , Label , Entries)
Print Idx ' return -1 if not found

'looking for integer or word data requires that the search variable is
'of the type integer !
Dim Isearch As Integer
Isearch = 400
Idx = Lookdown(isearch , Label2 , Entries)
Print Idx ' return 3

End


Label:
Data 1 , 2 , 3 , 4 , 5

Label2:
Data 1000% , 200% , 400% , 300%
```

LOOKUP

Action

Returns a value from a table.

Syntax

var **=LOOKUP(** value**,** label **)**

Remarks

Var	The returned value
Value	A value with the index of the table
Label	The label where the data starts

The value can be up to 65535. 0 will return the first entry.

See also

LOOKUPSTR

Example

```
Dim B1 As Byte , I As Integer
B1 = Lookup(2 , Dta)
Print B1 ' Prints 3 (zero based)

I = Lookup(0 , Dta2) ' print 1000
Print I
End


Dta:
Data 1 , 2 , 3 , 4 , 5
Dta2:
Data 1000% , 2000%
```

LOOKUPSTR

Action

Returns a string from a table.

Syntax

var =LOOKUPSTR(value, label)

Remarks

Var	The string returned
Value	A value with the index of the table. The index is zero-based. That is, 0 will return the first element of the table.
Label	The label where the data starts

The index value can have a maximum value of 255.

See also

LOOKUP

Example

```
Dim S As String * 4 , Idx As Byte
Idx = 0 : S = Lookupstr(idx , Sdata)
Print S 'will print 'This'
End

Sdata:
Data "This" , "is" ,"a test"
```

LOW

Action

Retrieves the least significant byte of a variable.

Syntax

var =**LOW**(s)

Remarks

Var	The variable that is assigned with the LSB of var S.
S	The source variable to get the LSB from.

See also

HIGH , HIGHW

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = Low(I) ' is 1
End
```

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
```

MAKEBCD

Action
Convert a variable into its BCD value.

Syntax
var1 = **MAKEBCD**(var2)

Remarks

var1	Variable that will be assigned with the converted value.
Var2	Variable that holds the decimal value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from decimal to BCD.

For printing the bcd value of a variable, you can use the BCD() function which converts a BCD number into a BCD string.

See also
MAKEDEC , BCD

Example
```
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcd Bcd(a)
A = Makebcd(a)
LCD "  " ; a
End
```

MAKEINT

Action
Compact two bytes into a word or integer.

Syntax
varn = MAKEINT(LSB , MSB)

Remarks

Varn	Variable that will be assigned with the converted value.
LSB	Variable or constant with the LS Byte.
MSB	Variable or constant with the MS Byte.

The equivalent code is:

varn = (256 * MSB) + LSB

See also
LOW , HIGH

Example
```
Dim a As Integer, I As Integer
A = 2
I = Makeint(a , 1) 'I = (1 * 256) + 2 = 258
End
```

MAKEDEC

Action

Convert a BCD byte or Integer/Word variable to its DECIMAL value.

Syntax

var1 = **MAKEDEC**(var2)

Remarks

var1	Variable that will be assigned with the converted value.
var2	Variable that holds the BCD value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from BCD to decimal.

See also

MAKEBCD

Example

```
Dim A As Byte
a = 65
Print A
Print Bcd(a)
A = Makedec(a)
Print Spc(3) ; A
End
```

MAX

Action

Returns the maximum value of a word array.

Syntax

var1 = **MAX** (var2)

MAX(ar(1), m ,idx)

Remarks

var1	Variable that will be assigned with the maximum value.
var2	The first address of the array.
	The MAX statement can return the index too
Ar(1)	Starting element to get the maximum value and index of.
M	Returns the maximum value of the array.
Idx	Return the index of the array that contains the maximum value. Returns 0 if there is no maximum value.

See also

MIN

Example

```
'-----------------------------------------------------------------------
---
' (c) 2001-2004 MCS
' minmax.bas
' This example show the MIN and MAX functions
' These functions only work on WORD arrays at the moment !!!!!
'-----------------------------------------------------------------------
---
'Dim some variables
Dim Wb As Word , B As Byte
Dim W(10) As Word

'fill the word array with values from 1 to 10
For B = 1 To 10
W(b) = B
Next

Print "Max number " ; Max(w(1))
Print "Min number " ; Min(w(1))


End
```

MIN

Action

Returns the minimum value of a word array.

Syntax

var1 = **MIN**(var2)

MIN(ar(1), m , idx)

Remarks

var1	Variable that will be assigned with the minimum value.
var2	The first address of the array.
	The MIN statement can return the index too
Ar(1)	Starting element to get the minimum value and index of
M	Returns the minimum value of the array
Idx	Return the index of the array that contains the minimum value. Returns 0 if there is no minimum value.

See also

MAX

Example

```
'--------------------------------------------------------------------------
---
' (c) 2001-2004 MCS
' minmax.bas
' This example show the MIN and MAX functions
' These functions only work on WORD arrays at the moment !!!!!
'--------------------------------------------------------------------------
---
'Dim some variables
Dim Wb As Word , B As Byte
Dim W(10) As Word

'fill the word array with values from 1 to 10
For B = 1 To 10
W(b) = B
Next

Print "Max number " ; Max(w(1))
Print "Min number " ; Min(w(1))


End
```

MID

Action

The MID function returns part of a string (a sub string).

The MID statement replaces part of a string variable with another string.

Syntax

var = **MID(**var1 ,st [, l] **)**

MID(var ,st [, l] **)** = var1

Remarks

var	The string that is assigned.
Var1	The source string.
st	The starting position.
l	The number of characters to get/set.

See also

LEFT, RIGHT

Example

```
Dim S As String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Mid(s , 2 , 3)
Print Z 'BCD
Z = "12345"
Mid(s , 2 , 2) = Z
Print S 'A12DEFG
End
```

ON INTERRUPT

Action

Execute subroutine when the specified interrupt occurs.

Syntax

ON interrupt label [NOSAVE]

Remarks

Interrupt	INT0, INT1, INT2, INT3, INT4,INT5, TIMER0 ,TIMER1, TIMER2, ADC , EEPROM , CAPTURE1, COMPARE1A, COMPARE1B,COMPARE1. Or you can use the AVR name convention : OC2 , OVF2, ICP1, OC1A, OC1B, OVF1, OVF0, SPI, URXC, UDRE, UTXC, ADCC, ERDY and ACI.
Label	The label to jump to if the interrupt occurs.
NOSAVE	When you specify NOSAVE, no registers are saved and restored in the interrupt routine. So when you use this option make sure to save and restore all used registers. When you omit NOSAVE all used registers will be saved. These are SREG , R31 to R16 and R11 to R0. R12 – R15 are not saved. When you use floating point math in the ISR(not recommended) you must save and restore R12-R15 yourself in the ISR. My_Isr: Push R12 ' save registers Push R13 Push R14 Push R15 Single = single + 1 ' we use FP Pop R15 ' restore registers Pop R14 Pop R13 Pop R12 RETURN

You must return from the interrupt routine with the RETURN statement.

The first RETURN statement that is encountered that is outside a condition will generate a RETI instruction. You may have only one such 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. All other RETURN statements are converted to a RET instruction.

The possible interrupt names can be looked up in the selected microprocessor register file. 2313def.dat for example shows that for the compare interrupt the name is COMPARE1. (look at the bottom of the file)

What are interrupts good for?

An interrupt will halt your program and will jump to a specific part of your program. You can make a DO .. LOOP and poll the status of a pin for example to execute some code when the input on a pin changes.

But with an interrupt you can perform other tasks and when then pin input changes a special part of your program will be executed. When you use INPUT "Name ", v for example to get a user name via the RS-232 interface it will wait until a RETURN is received. When you have an interrupt routine and the int occurs it will branch to the interrupt code and will execute the interrupt code. When it is finished it will return to the Input statement, waiting until a RETURN is entered.

Maybe a better example is writing a clock program. You could update a variable in your program that updates a second counter. But a better way is to use a TIMER interrupt and update a seconds variable in the TIMER interrupt handler.

There are multiple interrupt sources and it depends on the used chip which are available.

To allow the use of interrupts you must set the global interrupt switch with a ENABLE INTERRUPTS statement. This only allows that interrupts can be used. You must also set the individual interrupt switches on!

ENABLE TIMER0 for example allows the TIMER0 interrupt to occur.

With the DISABLE statement you turn off the switches.

When the processor must handle an interrupt it will branch to an address at the start of flash memory. These addresses can be found in the DAT files.

The compiler normally generates a RETI instruction on these addresses so that in the event that an interrupt occurs, it will return immediately.

When you use the ON ... LABEL statement, the compiler will generate code that jumps to the specified label. The SREG and other registers are saved at the LABEL location and when the RETURN is found the compiler restores the registers and generates the RETI so that the program will continue where it was at the time the interrupt occurred.

When an interrupt is services no other interrupts can occur because the processor(not the compiler) will disable all interrupts by clearing the master interrupt enable bit. When the interrupt is services the interrupt is also cleared so that it can occur again when the conditions are met that sets the interrupt.

It is not possible to give interrupts a priority. The interrupt with the lowest address has the highest interrupt!

Finally some tips :

* when you use a timer interrupt that occurs each 10 uS for example, be sure that the interrupt code can execute in 10 uS. Otherwise you would loose time.

* it is best to set just a simple flag in the interrupt routine and to determine it's status in the main program. This allows you to use the NOSAVE option that saves stack space and program space. You only have to Save and Restore R24 and SREG in that case.

* Since you can not PUSH a hardware register, you need to load it first:

PUSH R24 ; since we are going to use R24 we better save it

IN r24, SREG ; get content of SREG into R24
PUSH R24 ; we can save a register

;here goes your asm code
POP R24 ;get content of SREG

OUT SREG, R24 ; save into SREG
POP R24 ; get r24 back

Example

```
Enable Interrupts
Enable Int0 'enable the interrupt
On Int0 Label2 Nosave 'jump to label2 on INT0
Do 'endless loop
nop
```

```
Loop
End

Label2:
Dim A As Byte
If A > 1 Then
Return 'generates a RET because it is inside a condition
End If
Return 'generates a RETI because it is the first RETURN
Return 'generates a RET because it is the second RETURN
```

ON VALUE

Action

Branch to one of several specified labels, depending on the value of a variable.

Syntax

ON var [**GOTO**] [**GOSUB**] label1 **[**, label2 **] [,CHECK]**

Remarks

Var	The numeric variable to test. This can also be a SFR such as PORTB.
label1, label2	The labels to jump to depending on the value of *var*.
CHECK	An optional check for the number of provided labels.

Note that the value is zero based. So when var is 0, the first specified label is jumped/branched.

It is important that each possible value has an associated label.

When there are not enough labels, the stack will get corrupted. For example :

On value label1, label2

And value = 2, there is no associated label.

You can use the optional CHECK so the compiler will check the value against the number of provided labels. When there are not enough labels for the value, there will be no GOTO or GOSUB and the next line will be executed.

ASM

The following code will be generated for a non-MEGA micro with ON value GOTO.

```
Ldi R26,$60       ; load address of variable
Ldi R27,$00 ; load constant in register
Ld R24,X
Clr R25
Ldi R30, Low(ON_1_ * 1)     ; load Z with address of the label
Ldi R31, High(ON_1_ * 1)
Add zl,r24       ; add value to Z
Adc zh,r25
Ijmp       ; jump to address stored in Z
ON_1_:
Rjmp lbl1       ; jump table
Rjmp lbl2
Rjmp lbl3
```

The following code will be generated for a non-MEGA micro with ON value GOSUB.

```
;##### On X Gosub L1 , L2
Ldi R30,Low(ON_1_EXIT * 1)
Ldi R31,High(ON_1_EXIT * 1)
```

```
Push R30 ;push return address
Push R31
Ldi R30,Low(ON_1_ * 1)      ;load table address
Ldi R31,High(ON_1_ * 1)
Ldi R26,$60
Ld R24,X
Clr R25
Add zl,r24 ; add to address of jump table
Adc zh,r25
Ijmp        ; jump !!!
ON_1_:
Rjmp L1
Rjmp L2
ON_1_EXIT:
```

As you can see a jump is used to call the routine. Therefore the return address is first saved on the stack.

Example

```
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:
Print "lbl1"

Lbl2:
Print "lbl2"
```

OPEN

Action

Opens a device.

Syntax

OPEN "device" for MODE As #channel

OPEN file FOR MODE as #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stopbits. The format for COM1 and COM2 is : COM1: or COM2: There is no speed/baud rate parameter since the default baud rate will be used that is specified with $BAUD or $BAUD1 The format for the software UART is: COMpin:speed,8,N,stopbits[,INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. 7 bit data or 8 bit data may be used. For parity N, O or E can be used. An optional parameter ,INVERTED can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232. For the AVR-DOS filesystem, Device can also be a string or filename constant like "readme.txt" or sFileName
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT. For the AVR-DOS filesystem, MODE may be INPUT, OUTPUT, APPEND or BINARY.
Channel	The number of the channel to open. Must be a positive constant >0. For the AVR-DOS filesystem, the channel may be a positive constant or a numeric variable. Note that the AVD-DOS filesystem uses real filehandles. The software UART does not use real file handles.

UART

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

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

In DOS the #number is a DOS file number that is passed to low level routines. In BASCOM the channel number is only used to identify the channel but there are no file handles. So opening a channel, will not use a channel. And closing the channel is only needed to make the syntax compatible with QB.

What is the difference?

In QB/VB you can close the channel in a subroutine like this:

OPEN "com1:" for binary as #1

Call test

Close #1

End

Sub test

Print #1, "test"

End Sub

This will work since the filenumber is a real variable in the OS.
In BASCOM it will not work : the CLOSE must come after the last I/O statement:

OPEN "com1:" for binary as #1

Call test

End

Sub test

Print #1, "test"

End Sub

Close #1

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

AVR-DOS

The AVR-DOS file system uses real file handles. This means that the CLOSE statement can be used at any place in your program just as with QB/VB.

See also

CLOSE , CRYSTAL

PRINT, LINE INPUT , LOC , LOF , EOF

Example 1

```
'--------------------------------------------------
' (c) 2000 MCS Electronics
' OPEN.BAS
' demonstrates software UART
'--------------------------------------------------
$crystal = 10000000 'change to the value of the XTAL you have installed


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related
statements
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
'store in byte
B = Inkey(#2)
'when the value > 0 we got something
If B > 0 Then
Print #1 , Chr(b) 'print the character
End If
Loop Until B = 27


Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B


'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

OUT

Action

Sends a byte to a hardware port or internal or external memory address.

Syntax

OUT address, value

Remarks

Address	The address where to send the byte to in the range of 0 -FFFF hex.
Value	The variable or value to send.

The OUT statement can write a value to any AVR memory location.

It is advised to use Words for the address. An integer might have a negative value and will write of course to a word address. So it will be 32767 higher as supposed. This because an integer has it's most significant bit set when it is negative.

To write to XRAM locations you must enable the External RAM access in the Compiler Chip Options.

You do not need to use OUT when setting a port variable. Port variables and other registers of the micro can be set like this : PORTB = value , where PORTB is the name of the register.

See also

INP

Example

```
Out &H8000 , 1 'send 1 to the databus(d0-d7) at hex address 8000
End
```

PEEK

Action

Returns the content of a register.

Syntax

var = **PEEK(** address **)**

Remarks

Var	Numeric variable that is assigned with the content of the memory location address
Address	Numeric variable or constant with the address location.(0-31)

Peek() will read the content of a register.

Inp() can read any memory location

See also

POKE , CPEEK , INP , OUT

Example

```
Dim A As Byte
A = Peek(0) 'return the first byte of the internal memory (r0)
End
```

POKE

Action

Write a byte to an internal register.

Syntax

POKEaddress , value

Remarks

Address	Numeric variable with the address of the memory location to set. (0-31)
Value	Value to assign. (0-255)

See also

PEEK , CPEEK , INP , OUT

Example

```
Poke 1 , 1 'write 1 to R1
End
```

POPALL

Action

Restores all registers that might be used by BASCOM.

Syntax

POPALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all registers and POPALL restores all registers.

See also

PUSHALL

POWER

Action

Returns the power of a single variable and its argument

Syntax

var =**POWER** (single, raise)

Remarks

Var	A numeric variable that is assigned with the power of variable single ^ raise.
Single	The single variable to get the power of.

The POWER function works for positive singles only.

When you use a ^ b , the sign will be preserved.

While Excel does not allow raising a negative single, QB does allow it.

The Power functions uses less code compared with the code that is generated when you use ^ for floating point values.

It is important that you use single variables for both single and raise. Constants are not accepted.

See Also

EXP ,LOG, LOG10

Example

Show sample

POWERDOWN

Action

Put processor into power down mode.

Syntax

POWERDOWN

Remarks

In the power down mode, the external oscillator is stopped. The user can use the WATCHDOG to power up the processor when the watchdog timeout expires. Other possibilities to wake up the processor is to give an external reset or to generate an external level triggered interrupt.

See also

IDLE , POWERSAVE

Example

```
Powerdown
```

POWERSAVE

Action

Put processor into power save mode.

Syntax

POWERSAVE

Remarks

The POWERSAVE mode is only available in the 8535, Mega8, Mega163.

See also

IDLE, POWERDOWN

Example

```
Powersave
```

PRINT

Action

Send output to the RS-232 port.

Writes a string to a file.

Syntax

PRINT [#channel ,] var ; " constant"

Remarks

Var	The variable or constant to print.

You can use a semicolon (;) to print more than one variable at one line.

When you end a line with a semicolon, no linefeed and carriage return will be added.

The PRINT routine can be used when you have a RS-232 interface on your uP.

The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator as an output device.

You can also use the build in terminal emulator.

The AVR-DOS filesystem also supports PRINT. But in that case, only strings can be written to disk.

When you need to print to the second hardware UART, or to a software UART, you need to specify a channel : PRINT #1, "test"

The channel must be openeded first before you can print to it. Look at OPEN and CLOSE for more details about the optional channel. For the first hardware UART, there is no need to use channels.

PRINT " test" will always use the first hardware UART.

See also

INPUT ,OPEN , CLOSE , SPC

Example

```
'--------------------------------------------------------------
' (c) 1999-2000 MCS Electronics
'--------------------------------------------------------------
' file: PRINT.BAS
' demo: PRINT, HEX
'--------------------------------------------------------------
Dim A As Byte , B1 As Byte , C As Integer , S As String * 4
A = 1
Print "print variable a " ; A
Print 'new line
Print "Text to print." 'constant to print


B1 = 10
Print Hex(b1) 'print in hexa notation
C = &HA000 'assign value to c%
```

```
Print Hex(c) 'print in hex notation
Print C 'print in decimal notation

C = -32000
Print C
Print Hex(c)
Rem Note That Integers Range From -32767 To 32768
End
```

PRINTBIN

Action

Print binary content of a variable to the serial port.

Syntax

PRINTBIN var [**;** varn]

PRINTBIN #channel**,** var [**;** varn]

Remarks

Var	The variable which value is send to the serial port.
varn	Optional variables to send.

The channel is optional and for use with OPEN and CLOSE statements.

PRINTBIN is equivalent to PRINT CHR(var);

When you use a Long for example, 4 bytes are printed.

Multiple variables may be sent. They must be separated by the **;** sign.

The number of bytes to send can be specified by an additional numeric parameter. This is convenient when sending the content of an array.

Printbin ar(1) ; 3 ‘ will send 3 bytes from array ar().

Printbin ar(1) ; 2 ; ar(2) ; 4 ‘ will send 2 bytes from array ar() starting at index 1, then 4 bytes from array ar() starting at index 4.

See also

INPUTBIN

Example

```
Dim A(10) As Byte , C As Byte
For C = 1 To 10
A(c) = c 'fill array
Next
Printbin A(1) 'print content of a(1). Not the whole array will be sent!

End
```

PSET

Action

Sets or resets a single pixel.

Syntax

PSET X , Y, value

Remarks

X	The X location of the pixel. In range from 0-239.
Y	The Y location of the pixel. In range from 0-63.
value	The value for the pixel. 0 will clear the pixel. 1 Will set the pixel.

The PSET is handy to create a simple data logger or oscilloscope.

See also

SHOWPIC , CONFIG GRAPHLCD , LINE

Example

```
'-----------------------------------------------------------------
' (c) 2001 MCS Electronics
' T6963C graphic display support demo
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin connected to
' 1 GND GND
'2 GND GND
'3 +5V +5V
'4 -9V -9V potmeter
'5 /WR PORTC.0
'6 /RD PORTC.1
'7 /CE PORTC.2
'8 C/D PORTC.3
'9 NC not conneted
'10 RESET PORTC.4
'11-18 D0-D7 PA
'19 FS PORTC.5
'20 NC not connected


'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 64 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2


'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part

'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"

'wait 1 sec
Wait 1


' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
Pset X , 20 , 255 ' set the pixel
Next

Wait 1


'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje

Wait 1
Cls Text ' clear the text
End



'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

PS2MOUSEXY

Action

Sends mouse movement and button information to the PC.

Syntax

PS2MOUSEXY X , Y, button

Remarks

X	The X-movement relative to the current position. The range is –255 to 255.
Y	The Y-movement relative to the current position. The range is –255 to 255.
Button	A variable or constant that represents the button state. 0 – no buttons pressed 1. 1- left button pressed 2. 2- right button pressed 4- middle button pressed You can combine these values by adding them. For example, 6 would emulate that the right and middle buttons are pressed. To send a mouse click, you need to send two ps2mouseXY statements. The first must indicate that the button is pressed, and the second must release the button. Ps2mouseXY 0,0,1 ' left mouse pressed PsmouseXY 0,0,0 ' left mouse released

The SENDSCAN statement could also be used.

See also

SENDSCAN, CONFIG PS2EMU

PULSEIN

Action

Returns the number of units between two occurrences of an edge of a pulse.

Syntax

PULSEIN var , PINX , PIN , STATE

Remarks

var	A word variable that is assigned with the result.
PINX	A PIN register like PIND
PIN	The pin number(0-7) to get the pulse time of.
STATE	May be 0 or 1. 0 means sample 0 to 1 transition. 1 means sample 1 to 0 transition.

ERR variable will be set to 1 in case of a time out. A time out will occur after 65535 unit counts. With 10 uS units this will be after 655.35 mS.

You can add a bitwait statement to be sure that the PULSEIN statement will wait for the start condition. But when using the BITWAIT statement and the start condition will never occur, your program will stay in a loop.

The PULSIN statement will wait for the specified edge.

When state 0 is used, the routine will wait until the level on the specified input pin is 0. Then a counter is started and stopped until the input level gets 1.

No hardware timer is used. A 16 bit counter is used. It will increase in 10 uS units. But this depends on the XTAL. You can change the library routine to adjust the units.

See also

PULSEOUT

ASM

The following ASM routine is called from mcs.lib

_pulse_in (calls _adjust_pin)

On entry ZL points to the PINx register , R16 holds the state, R24 holds the pin number to sample.

On return XL + XH hold the 16 bit value.

Example

```
Dim w As Byte
pulsein w , PIND , 1 , 0 'detect time from 0 to 1
print w
end
```

PULSEOUT

Action

Generates a pulse on a pin of a PORT of specified period in 1uS units for 4 MHz.

Syntax

PULSEOUT PORT , PIN , PERIOD

Remarks

PORT	Name of the PORT. PORTB for example
PIN	Variable or constant with the pin number (0-7).
PERIOD	Number of periods the pulse will last. The periods are in uS when an XTAL of 4 MHz is used.

The pulse is generated by toggling the pin twice, thus the initial state of the pin determines the polarity.

The PIN must be configured as an output pin before this statement can be used.

See also

PULSEIN

Example

```
Dim A As Byte
Config Portb = Output 'PORTB all output pins
Portb = 0 'all pins 0
Do
For A = 0 To 7
Pulseout Portb , A , 60000 'generate pulse
Waitms 250 'wait a bit
Next
Loop 'loop for ever
```

PUSHALL

Action

Saves all registers that might be used by BASCOM.

Syntax

PUSHALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all registers. Use POPALL to restore the registers.

See also

POPALL

PUT

Action

Writes a byte to the hardware or software UART.

Writes data to a file opened in BINARY mode.

Syntax

PUT #channel, var
PUT #channel, var ,[pos] [,length]

Remarks

PUT in combination with the software/hardware UART is provided for compatibility with BASCOM-8051. It writes one byte

PUT in combination with the AVR-DOS filesystem is very flexible and versatile. It works on files opened in BINARY mode and you can write all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be written to the file
Pos	This is an optional parameter that may be used to specify the postion where the data must be written. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be written to the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte wil be written. When the variable is a word or integer, 2 bytes will be written. When the variable is a long or single, 4 bytes will be written. When the variable is a string, the number of bytes that will be written is equal to the dimensioned size of the string. DIM S as string * 10 , would write 10 bytes.

Note that when you specify the length for a string, the maximum length is 255. The maximum length for a non-string array is 65535.

Example:

PUT #1, var

PUT #1, var , , 2‘ write 2 bytes at default position

PUT #1, var ,PS, 2 ‘ write 2 bytes at location storied in variable PS

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET, FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

current position	**Goto new position first**
Byte:	
_FilePutRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FilePutRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FilePutRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FilePutRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Array (> 255 Bytes) with fixed length	
_FilePutRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange Input: r24: File number r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage:

r25: Error Code

C-Flag on Error

Example

```
'for the binary file demo we need some variables of different types
Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp As Long
Dim Stxt As String * 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Seek(#2) ; " = LOC+1"
Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode" ' should be 32 for binary
Put #2 , Sn ' write a single
Put #2 , Stxt ' write a string

Flush #2 ' flush to disk
Close #2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
B = Seek(#2 , L) ' reset is the same as using SEEK #2,L
Get #2 , B ' get the byte
Get #2 , W ' get the word
Get #2 , L ' get the long
Get #2 , Sn ' get the single
Get #2 , Stxt ' get the string
Close #2
```

RAD2DEG

Action

Converts a value in radians to degrees.

Syntax

var =**RAD2DEG**(single)

Remarks

Var	A numeric variable that is assigned with the angle of variable single.
Single	The single variable to get the angle of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

DEG2RAD

Example

```
Dim S As Single
S = 90
S = Deg2Rad(s)
Print S
```

RC5SEND

Action

Sends RC5 remote code.

Syntax

RC5SEND togglebit, address, command

Uses

TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 32 to set the toggle bit
Address	The RC5 address
Command	The RC5 command.

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a datasheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infra-red remote control.

The RC5 code is a 14-bit word bi-phase coded signal.

The two first bits are start bits, always having the value 1.

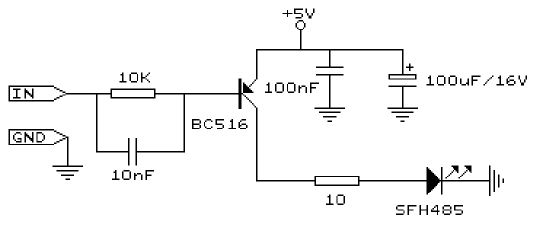
The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.

Fivesystem bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

An IR booster circuit is shown below:



See also

[CONFIG RC5](), [GETRC5]()

Example

```
'-----------------------------------------------------------------
' RC5SEND.BAS
' (c) 2004 MCS Electronics
' code based on application note from Ger Langezaal
' +5V <---[A Led K]---[220 Ohm]---> Pb.3
' RC5SEND is using TIMER1, no interrupts are used
'-----------------------------------------------------------------
$regfile = "2313def.dat"
$crystal = 4000000

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12 ' power on off
Togbit = 0 ' make it 0 or 32 to set the toggle bit
Address = 0
Do
Waitms 500
Rc5send Togbit , Address , Command
Loop

End
```

RC6SEND

Action

Sends RC6 remote code.

Syntax

RC6SEND togglebit, address, command

Uses

TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 1 to set the toggle bit
Address	The RC6 address
Command	The RC6 command.

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a datasheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infrared remote control.

The RC6 code is a 16-bit word bi-phase coded signal.

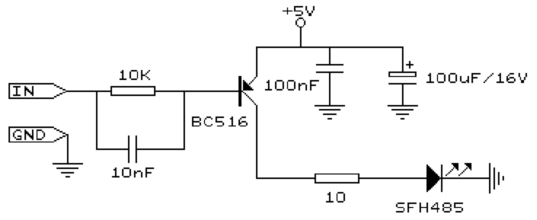
The header is 20 bits long including the toggle bits.

Eight system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is eight bits long, allowing up to 256 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

An IR booster circuit is shown below:



Device	Address
TV	0
VCR	5
SAT	8
DVD	4

This is not a complete list.

Command	Value	Command	Value
Key 0	0	Balance right	26
Key 1	1	Balance left	27
Key 2-9	2-9	Channel search+	30
Previous program	10	Channel search -	31
Standby	12	Next	32
Mute/demute	13	Previous	33
Personal preference	14	External 1	56
Display	15	External 2	57
Volume up	16	TXT submode	60
Volume down	17	Standby	61
Brightness up	18	Menu on	84
Brightness down	19	Menu off	85
Saturation up	20	Help	129
Saturation down	21	Zoom -	246
Bass up	22	Zoom +	247
Bass down	23		
Treble up	24		
Treble down	25		

This list is by far not complete.

Since there is little info about RC6 on the net available, use code at your own risk!

See also

CONFIG RC5 , GETRC5 , RC5SEND

Example

```
'-----------------------------------------------------------------
' RC6SEND.BAS
' (c) 2004 MCS Electronics
' code based on application note from Ger Langezaal
' +5V <---[A Led K]---[220 Ohm]---> Pb.3
' RC5SEND is using TIMER1, no interrupts are used
'-----------------------------------------------------------------
$regfile = "2313def.dat"
$crystal = 4000000

Dim Togbit As Byte , Command As Byte , Address As Byte

Command = 12 ' power on off
Togbit = 0 ' make it 0 or 1 to set the toggle bit
```

```
Address = 0
Do
Waitms 500
Rc6send Togbit , Address , Command
Loop

End
```

READ

Action

Reads those values and assigns them to variables.

Syntax

READ var

Remarks

Var	Variable that is assigned data value.

It is best to place the DATA lines at the end of your program.

Difference with QB

It is important that the variable is of the same type as the stored data.

See also

DATA, RESTORE

Example

```
'--------------------------------------------------------------
' READDATA.BAS
' Copyright 1999-2000 MCS Electronics
'--------------------------------------------------------------

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

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

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


Restore Dta4
Read L : Print L 'long type

End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%
```

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

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

READEEPROM

Action

Reads the content from the DATA EEPROM and stores it into a variable.

Syntax

READEEPROM var , address

Remarks

Var	The name of the variable that must be stored
Address	The address in the EEPROM where the data must be read from.

This statement is provided for compatibility with BASCOM-8051.

You can also use :

```
Dim V as Eram Byte 'store in EEPROM
Dim B As Byte 'normal variable
B = 10
V = B 'store variable in EEPROM
B = V 'read from EEPROM
```

When you use the assignment version, the datatypes must be equal!

According to a datasheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset so don't use it.

You may also use ERAM variables as indexes. Like :

```
Dim ar(10) as Eram Byte
```

When you omit the address label in consecutive reads, you must use a new READEEPROM statement. It will not work in a loop:

```
Readeeprom B , Label1
Print B
Do
Readeeprom B
Print B Loop
Until B = 5
```

This will not work since there is no pointer maintained. The way it will work :

```
ReadEEprom B , Label1 ' specify label
ReadEEPROM B ' read next address in EEPROM
ReadEEPROM B ' read next address in EEPROM
```

See also

WRITEEEPROM , $EEPROM

ASM

NONE

Example

```
Dim B As Byte
Writeeeprom B , 0 'store at first position
Readeeprom B , 0 'read byte back
```

Example 2

```
'-----------------------------------------------------------------
' EEPROM2.BAS
' This example shows how to use labels with READEEPROM
'-----------------------------------------------------------------
'first dimension a variable
Dim B As Byte
Dim Yes As String * 1

'Usage for readeeprom and writeeprom :
'readeeprom var, address

'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!

'first tell the compiler that we are using EEPROM to store the DATA
$eeprom
'specify a label
label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50

'Switch back to normal data lines in case they are used
$data

'All the code above does not generate real object code
'It only creates a file with the EEP extension

'Use the new label option
Readeeprom B , Label1
Print B 'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B 'prints 2

Readeeprom B , Label2
Print B 'prints 10
Readeeprom B
Print B 'prints 20

'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Label1
B = 101
Writeeeprom B

'read it back
Readeeprom B , Label1
Print B 'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B 'prints 2


End
```

READMAGCARD

Action

Read data from a magnetic card.

Syntax

Readmagcard var , count , 5|7

Remarks

Var	A byte array the receives the data.
Count	A byte variable that returns the number of bytes read.
5|7	A numeric constant that specifies if 5 or 7 bit coding is used.

There can be 3 tracks on a magnetic card.

Track 1 strores 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 tored with 5 bit coding.

The ReadMagCard routine works with ISO7811-2 5 and 7 bit decoding.

The returned numbers for 5 bit coding are:

Returned number	ISO characterT
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	hardware control
11	start byte
12	hardware control
13	separator
14	hardware control
15	stop byte

Example

```
'-----------------------------------------------------------------------
--
' (c) 2000 MCS Electronics
' MAGCARD.BAS
' This example show you how to read data from a magnetic card
'It was tested on the DT006 SimmStick.
'-----------------------------------------------------------------------
--
'[reserve some space]
Dim Ar(100) As Byte , B As Byte , A As Byte

'the magnetic card reader has 5 wires
'red - connect to +5V
'black - connect to GND
'yellow - Card inserted signal CS
'green - clock
'blue - data

'You can find out for your reader which wires you have to use by
connecting +5V
'And moving the card through the reader. CS gets low, the clock gives a
clock pulse of equal pulses
'and the data varies
'I have little knowledge about these cards and please dont contact me
about magnectic readers
'It is important however that you pull the card from the right direction
as I was doing it wrong for
'some time :-)
'On the DT006 remove all the jumpers that are connected to the LEDs

'[We use ALIAS to specify the pins and PIN register]
_mport Alias Pinb 'all pins are connected to PINB
_mdata Alias 0 'data line (blue) PORTB.0
_mcs Alias 1 'CS line (yellow) PORTB.1
_mclock Alias 2 'clock line (green) PORTB.2

Config Portb = Input 'we only need bit 0,1 and 2 for input
Portb = 255 'make them high

Do
Print "Insert magnetic card" 'print a message
Readmagcard Ar(1) , B , 5 'read the data
Print B ; " bytes received"
For A = 1 To B
Print Ar(a); 'print the bytes
Next
Print
Loop

'By specifying 7 instead of 5 you can read 7 bit data
```

REM

Action

Instruct the compiler that comment will follow.

Syntax

REM or **'**

Remarks

You can and should comment your program for clarity and your later sanity.
You can use REM or **'** followed by your comment.
All statements after REM or **'** are treated as comments so you cannot use statements on the same line after a REM statement.

Block comments can be used too:

'(start block comment
print "This will not be compiled
') end block comment

Example

```
REM TEST.BAS version 1.00
PRINT a ' " this is comment : PRINT " hello"
    ^--- this will not be executed!
```

RESET

Action

Reset a bit to zero.

Syntax

RESET bit
RESET var.x

Remarks

bit	Can be a SFR such as PORTB.x, or any bit variable where x=0-7.
var	Can be a byte, integer word or long variable.
x	Constant of variable to reset.(0-7) for bytes and (0-15) for Integer/Word. For longs(0-31)

See also

SET

Example

```
Dim b1 as bit, b2 as byte, I as Integer
Reset Portb.3 'reset bit 3 of port B
Reset B1 'bitvariable
Reset B2.0 'reset bit 0 of bytevariable b2
Reset I.15 'reset MS bit from I
End
```

RESTORE

Action

Allows READ to reread values in specified DATA statements by setting data pointer to beginning of data statement.

Syntax

RESTORE label

Remarks

label	The label of a DATA statement.

See also

DATA , READ , LOOKUP

Example

```
'--------------------------------------------------------------
' READDATA.BAS
' Copyright 1999-2000 MCS Electronics
'--------------------------------------------------------------

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

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

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


Restore Dta4
Read L : Print L 'long type

End


Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%

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

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

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
```

RIGHT

Action

Return a specified number of rightmost characters in a string.

Syntax

var = **RIGHT(**var1 ,n **)**

Remarks

var	The string that is assigned.
Var1	The source string.
st	The number of bytes to copy from the right of the string.

See also

LEFT , MID

Example

```
Dim S As String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Right(s , 2)
Print Z 'FG
End
```

RND

Action

Returns a random number.

Syntax

var =**RND(** limit **)**

Remarks

Limit	Word that limits the returned random number.
Var	The variable that is assigned with the random number.

The RND() function returns an Integer/Word and needs an internal storage of 2 bytes. (___RSEED). Each new call to Rnd() will give a new positive random number.

Notice that it is a software based generated number. And each time you will restart your program the same sequence will be created.

You can use a different SEED value by dimensioning and assigning ___RSEED yourself:

Dim ___rseed as word : ___rseed = 10234

Dim I as word : I = rnd(10)

When your application uses a timer you can assign ___RSEED with the timer value. This will give a better random number.

See also

NONE

Example

```
Dim I As Integer
Do
I = Rnd(100) 'get random number from 0-99
Print I
Waitms 100
Loop
End
```

ROTATE

Action

Rotate all bits one place to the left or right.

Syntax

ROTATE var**, LEFT/RIGHT** [**,** shifts]

Remarks

Var	Byte, Integer/Word or Long variable.
Shifts	The number of shifts to perform.

The ROTATE statement rotates all the bits in the variable to the left or right. All bits are preserved so no bits will be shifted out of the variable.

This means that after rotating a byte variable with a value of 1, eight times the variable will be unchanged.

When you want to shift out the MS bit or LS bit, use the SHIFT statement.

See also

SHIFT, SHIFTIN , SHIFTOUT

Example

```
Dim a as Byte
a = 128
Rotate A , Left , 2
Print a '2
End
```

ROUND

Action

Returns a values rounded to the nearest value.

Syntax

var =**ROUND**(x)

Remarks

Var	A single variable that is assigned with the ROUND of variable x.
X	The single to get the ROUND of.

Round(2.3) = 2 , Round(2.8) = 3

Round(-2.3) =-2 , Round(-2.8) = -3

See Also

INT , FIX , SGN

Example

```
'--------------------------------------------------------------------------
-------
' ROUND_FIX_INT.BAS
'--------------------------------------------------------------------------
-------
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

RTRIM

Action

Returns a copy of a string with trailing blanks removed

Syntax

var =**RTRIM**(org)

Remarks

var	String that is assigned with the result.
org	The string to remove the trailing spaces from

See also

TRIM , LTRIM

ASM

NONE

Example

```
Dim S As String * 6
S = " AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

SECELAPSED

Action

Returns the elapsed Seconds to a former assigned time-stamp.

Syntax

Target = **SecElapsed**(TimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
TimeStamp	A variable (LONG), which holds a timestamp like the output of an earlier called SecOfDay()

The Function works with the SOFTCLOCK variables _sec, _min and _hour and considers a jump over midnight and gives a correct result within 24 hour between two events.

The Return-Value is in the range of 0 to 86399.

See also

Date and Time Routines , SecOfDay , SysSecElapsed

Example

```
Enable Interrupts
Config Clock = Soft

Dim lTimeStamp as Long
Dim lSecondsElapsed as Long

lTimeStamp = SecOfDay()
Print "Now it's " ; lTimeStamp ; " seconds past midnight"

' do other stuff
' some time later

lSecondsElapsed = SecElapsed(lTimeStamp)
Print "Now it's " ; lSecondsElapsed ; " seconds later"
```

SECOFDAY

Action

Returns the Seconds of a Day.

Syntax

Target = **SecOfDay**()

Target = **SecOfDay**(bSecMinHour)

Target = **SecOfDay**(strTime)

Target = **SecOfDay**(lSysSec)

Remarks

Target	A variable (LONG), that is assigned with the Seconds of the Day
bSecMinHour	A Byte, which holds the Second-value followed by Minute(Byte) and Hour(Byte)
strTime	A String, which holds the time in the format „hh:mm:ss"
LSysSec	A Variable (Long) which holds the System Second

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Time of SOFTCLOCK (_sec, _min, _hour) is used.
2. With a user defined time array. It must be arranged in same way (Second, Minute, Hour) as the internal SOFTCLOCK time. The first Byte (Second) is the input by this kind of usage. So the Second of Day can be calculated of every time.
3. With a time-String. The time-string must be in the Format „hh:mm:ss".
4. With a System Second Number (LONG)

The Return-Value is in the range of 0 to 86399 from 00:00:00to23:59:59.

Novalidity-check of input is made.

See also

Date and Time Routines , SysSec

Example

```
Enable Interrupts
Config Clock = Soft
Dim strtime as String * 8
Dim bSec as Byte, bMin as Byte, bHour as Byte
Dim lSecOfDay as Long
Dim lSysSec as Long
```

```
' Example 1 with internal RTC-Clock
_Sec = 12: _Min = 30: _Hour = 18 ' Load RTC-Clock for example - testing
lSecOfDay = SecOfDay()
print "Second of Day of " ; time$ ; " is " ; lSecOfDay
' Second of Day of 18:30:12 is 66612

' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
bSec = 20: bMin = 1: bHour = 7

lSecOfDay = SecOfDay(bSec)
print "Second of Day of Sec="; bsec ; " Min="; bmin ; " Hour=" ; bHour ;
" is " ; lSecOfDay
' Second of Day of Sec=20 Min=1 Hour=7 is 25280

' Example 3 with Time - String
strTime = "04:58:37"
lSecOfDay = SecOfDay(strTime)
print "Second of Day of " ; strTime ; " is " ; lSecOfDay
' Second of Day of 04:58:37 is 17917

' Example 4 with System Second
lSysSec = 1234456789
lSecOfDay = SecOfDay(lSysSec)
print "Second of Day of System Second " ; lSysSec ; " is " ; lSecOfDay
' Second of Day of System Second 1234456789 is 59989
```

SEEK

Action

Function: Returns the position of the next Byte to be read or written

Statement: Sets the position of the next Byte to be read or written

Syntax

Function: NextReadWrite =**Seek** (#bFileNumber)

Statement: **Seek** #bFileNumber, NewPos

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
NextReadWrite	A Long Variable, which is assigned with the Position of the next Byte to be read or written (1-based)
NewPos	A Long variable that holds the new position the filepointer must be set too.

This function returns the position of the next Byte to be read or written. If an error occures, 0 is returned. Check DOS-Error in variable gbDOSError.

The statetement also returns an error in the gbDOSerror variable in the event that an error occurs. You can for example not set the fileposition behinds the filesize.

In QB/VB the file is filled with 0 bytes when you set the filepointer behind the size of the file. For embedded systems this does not seem a good idea.

Seek and Loc seems to do the same function, but take care : the seek function will return the position of the next read/write, while the Loc function returns the position of the last read/write. You may say that Seek = Loc+1.

Difference with QB

In QB/VB you can use seek to make the file bigger. When a file is 100 bytes long, setting the filepointer to 200 will increase the file with 0 bytes. By design this is not the case in AVR-DOS.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUTFILEDATE , FILETIME , FILEDATETIME , DIR , FILELENWRITE , INPUT

ASM

Function Calls	_FileSeek	
Input	r24: filenumber	X: Pointer to Long-variable, which gets the result

Output	r25: Errorcode	C-Flag: Set on Error

Statement Calls	_FileSeekSet	
Input	r24: filenumber	X: Pointer to Long-variable with the position
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
Open "test.biN" For Binary As #2
Put #2 , B ' write a byte
Put #2 , W ' write a word
Put #2 , L ' write a long
Ltemp = Loc(#2) + 1 ' get the position of the next byte
Print Ltemp ; " LOC" ' store the location of the file pointer
Print Seek(#2) ; " = LOC+1"
Close #2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
Seek #2 , Ltemp ' set the filepointer
Sn = 1.23 ' change the single value so we can check it better
Put #2 , Sn = 1 'specify the file position
Close #2
```

SELECT-CASE-END SELECT

Action

Executes one of several statement blocks depending on the value of an expression.

Syntax

SELECT CASE var

CASE test1 **:** *statements*
[**CASE** test2 **:** *statements*]
CASE ELSE : *statements*
END SELECT

Remarks

Var	Variable. to test
Test1	Value to test for.
Test2	Value to test for.

You can test for conditions to like:

CASE IS > 2 :

Another option is to test for a range :

CASE 2 TO 5 :

See also

IF THEN

Example

```
Dim X As Byte

Do
Input "X ? ", X
Select Case X
Case 1 To 3 : Print "1 , 2 or 3 will be ok"
Case 4 : Print "4"
Case Is > 10 : Print ">10"
Case Else : Print "no"
End Select
Loop

End
```

SET

Action

Set a bit to the value one.

Syntax

SET bit

SET var.x

Remarks

Bit	Bitvariable.
Var	A byte, integer, word or long variable.
X	Bit of variable (0-7) to set. (0-15 for Integer/Word) and (0-31) for Long

See also

RESET

Example

```
Dim B1 As Bit , B2 As Byte , C As Word , L As Long
Set Portb.1 'set bit 1 of port B
Set B1 'bit variable
Set B2.1 'set bit 1 of var b2
Set C.15 'set highest bit of Word
Set L.31 'set MS bit of LONG

End
```

SETFONT

Action

Sets the font used for LCDAT on SED based Graphical LCD displays.

Syntax

SETFONT font

Remarks

font	The name of the font that need to be used with LCDAT statements.

Since SED- based displays do not have their own font generator, you need to define your own fonts. You can create and modify your own fonts with the FontEditor Plugin.

SETFONT will set an internal used data pointer to the location in memory where you font is stored. The name you specify is the same name you use to define the font.

You need to include the used fonts with the $include directive:

$INCLUDE "font8x8.font"

The order is not important.

You need to include the glibSED library with :

$LIB "glibsed.lbx"

See also

CONFIG GRAPHLCD , LCDAT, GLCDCMD, GLCDDATA

Example

```
'-------------------------------------------------------------
```

SETTCP

Action

(Re) Configures the TCP/IP W3100A chip.

Syntax

SETTCP MAC , IP , SUBMASK , GATEWAY

Remarks

MAC	The MAC address you want to assign to the W3100A. The MAC address is a unique number that identifies your chip. You must use a different address for every W3100A chip in your network. Example : 123.00.12.34.56.78 You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation.
IP	The IP address you want to assign to the W3100A. The IP address must be unique for every W3100A in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address.
SUBMASK	The submask you want to assign to the W3100A. The submask is in most cases 255.255.255.0
GATEWAY	This is the gateway address of the W3100A. The gateway address you can determine with the IPCONFIG command at the command prompt : C:\>ipconfig Windows 2000 IP Configuration Ethernet adapter Local Area Connection 2: Connection-specific DNS Suffix . : IP Address. . . . . : 192.168.0.3 Subnet Mask . . . . : 255.255.255.0 Default Gateway . . . . : **192.168.0.1** Use 192.168.0.1 in this case.

The CONFIG TCPIP statement may be used only once.

When you want to set the TCP/IP settings dynamicly for instance when the settings are stored in EEPROM, you can not use constants. For this purpose, SETTCP must be used

SETTCP can take a variable or a constant for each parameter.

When you set the TCP/IP settings dynamicly, you do not need to set them with CONFIG TCPIP. In the CONFIG TCPIP you can use the NOINIT parameter so that the MAC and IP are not initialized which saves code.

See also

GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN , CONFIG TCPIP

Example

See the DHCP2.BAS example from the Sample dir.

SENDSCAN

Action

Sendsscan codes to the PC.

Syntax

SENDSCAN label

Remarks

Label	The name of the label that contains the scan codes.

The SENDSCAN statement can send multiple scan codes to the PC.

The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

The following table lists all mouse scan codes.

Emulated Action	Data sent to host
Move up one	08,00,01
Move down one	28,00,FF
Move right one	08,01,00
Move left one	18,FF,00
Press left button	09,00,00
Release left button	08,00,00
Press middle button	0C,00,00
Release middle button	08,00,00
Press right button	0A,00,00
Release right button	08,00,00

To emulate a left mouse click, the data line would look like this:

DATA 6 , &H09, &H00, &H00, &H08 , &H00, &H00

^ send 6 bytes

^ left click

^ release

See also

PS2MOUSEXY , CONFIG PS2EMU

SENDSCANKBD

Action

Sends keyboard scan codes to the PC.

Syntax

SENDSCANKBD label

Remarks

Label	The name of the label that contains the scan codes.

The SENDSCANKBD statement can send multiple scan codes to the PC.

The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

The following tables lists all scan codes.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK		KEY	MAKE	BREAK		KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	F0,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	F0,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	F0,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C

O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

ACPI Scan Codes

Key	Make Code	Break Code
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

Windows Multimedia Scan Codes

Key	Make Code	Break Code
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

To emulate volume up, the data line would look like this:

```
DATA 5 , &HE0, &H32, &HE0, &HF0 , &H32
^ send 5 bytes
^ volume up
```

See also

CONFIG ATEMU

SERIN

Action

Reads serial data from a dynamic software UART.

Syntax

SERIN var , bts , port , pin, baud , parity , dbits , sbits

Remarks

While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opened the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different parameter values, the code size of these routines is larger.

Parameter	Description
Var	A variable that will be assigned with the received data.
Bts	The number of bytes to receive. String variables will wait for a return (ASCII 13). There is no check if the variable you assign is big enough to hold the result.
Port	The name of the port to use. This must be a letter like A for portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. 0= NONE, 1 = EVEN, 2 = ODD
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.

The use of SERIN will create an internal variable named ___SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with $CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ___SER_BAUD will not hold the passed baud rate but will hold the bit delay used internal.

Since the SW UART is dynamic you can change all the params at run time. For example you can store the baud rate in a variable and pass this variable to the SERIN routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

ASM

The routine called is named _serin and is stored in mcs.lib

For the baud ate calculation, _calc_baud is called.

See also

SEROUT

Example

```
'-----------------------------------------------------------------------
' serin_out.bas
' (c) 2004 MCS Electronics
' demonstration of DYNAMIC software UART
'-----------------------------------------------------------------------

'tip : Also look at OPEN and CLOSE

'tell the compiler which XTAL was used
$crystal = 4000000

'tell the compiler which chip we use
$regfile = "2313def.dat"

'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG

Mybaud = 19200
Do
'first get some data
Serin S , 0 , D , 0 , Mybaud , 0 , 8 , 1
'now send it
Serout S , 0 , D , 1 , Mybaud , 0 , 8 , 1
' ^ 1 stop bit
' ^--- 8 data bits
' ^----- even parity (0=N, 1 = E, 2=O)
' ^------------- baud rate
' ^------------------- pin number
' ^---------------------- port so PORTA.0 and PORTA.1 are used
' ^-------------------------- for strings pass 0
' ^------------------------------- variable
Wait 1
Loop
End

'because the baud rate is passed with a variable in theis example, you
could change it under user control
'for example check some DIP switches and change the variable mybaud
```

SEROUT

Action

Sends serial data through a dynamic software UART.

Syntax

SEROUT var , bts , port , pin, baud , parity , dbits , sbits

Remarks

While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opened the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different parameter values, the code size of these routines is larger.

Parameter	Description
Var	A variable which content is send through the UART. A constant can NOT be used.
Bts	The number of bytes to receive. String variables will wait for a return (ASCII 13). There is no check if the variable you assign is big enough to hold the result.
Port	The name of the port to use. This must be a letter like A for portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. 0= NONE, 1 = EVEN, 2 = ODD
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.

The use of SEROUT will create an internal variable named ___SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with $CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ___SER_BAUD will not hold the passed baud rate but will hold the bit delay used internal.

Since the SW UART is dynamic you can change all the params at run time. For example you can store the baud rate in a variable and pass this variable to the SEROUT routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

The SEROUT will use the pin in Open Collector mode. This means that you can connect several AVR chips and poll the ' bus' with the SERIN statement.

ASM

The routine called is named _serout and is stored in mcs.lib

For the baud ate calculation, _calc_baud is called.

See also

SERIN

Example

```
'--------------------------------------------------------------------
' serin_out.bas
' (c) 2004 MCS Electronics
' demonstration of DYNAMIC software UART
'--------------------------------------------------------------------

'tip : Also look at OPEN and CLOSE

'tell the compiler which XTAL was used
$crystal = 4000000

'tell the compiler which chip we use
$regfile = "2313def.dat"

'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG

Mybaud = 19200
Do
'first get some data
Serin S , 0 , D , 0 , Mybaud , 0 , 8 , 1
'now send it
Serout S , 0 , D , 1 , Mybaud , 0 , 8 , 1
' ^ 1 stop bit
' ^--- 8 data bits
' ^----- even parity (0=N, 1 = E, 2=O)
' ^------------- baud rate
' ^------------------- pin number
' ^---------------------- port so PORTA.0 and PORTA.1 are used
' ^------------------------- for strings pass 0
' ^------------------------------ variable
Wait 1
Loop
End

'because the baud rate is passed with a variable in theis example, you
could change it under user control
'for example check some DIP switches and change the variable mybaud
```

SGN

Action

Returns the sign of a single value.

Syntax

var =**SGN**(x)

Remarks

Var	A single variable that is assigned with the SGNS of variable x.
X	The single to get the sign of.

For values <0, -1 will be returned

For 0, 0 will be returned

For values >0, 1 will be returned

See Also

INT , FIX , ROUND

Example

```
Dim S As Single , x As Single, y As Single
x= 2.3 : S= ROUND(x) '2
x= -2.3 : S = ROUND(x) '-2
Print S
```

SHIFT

Action

Shift all bits one place to the left or right.

Syntax

SHIFT var **, LEFT/RIGHT** [**,** shifts]

Remarks

Var	Byte, Integer/Word, Long or Singlevariable.
Shifts	The number of shifts to perform.

The SHIFT statement rotates all the bits in the variable to the left or right.

When shifting LEFT the most significant bit, will be shifted out of the variable. The LS bit becomes zero. Shifting a variable to the left, multiplies the variable with a value of two.

When shifting to the RIGHT, the least significant bit will be shifted out of the variable. The MS bit becomes zero. Shifting a variable to the right, divides the variable by two.

A Shift performs faster than a multiplication or division.

See also

ROTATE , SHIFTIN , SHIFTOUT

Example

```
Dim a as Byte
a = 128
Shift A , Left , 2
Print a '0
End
```

SHIFTCURSOR

Action

Shift the cursor of the LCD display left or right by one position.

Syntax

SHIFTCURSOR LEFT / RIGHT

See also

SHIFTLCD

Example

```
LCD "Hello"
SHIFTCURSOR LEFT
End
```

SHIFTIN

Action

Shifts a bit stream into a variable.

Syntax

SHIFTIN pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as an input.PINB.2 for example
Pclock	The port pin which generates the clock.
Var	The variable that is assigned.
Option	Option can be : 0 – MSB shifted in first when clock goes low 1 – MSB shifted in first when clock goes high 2 – LSB shifted in first when clock goes low 3 – LSB shifted in first when clock goes high Adding 4 to the parameter indicates that an external clock signal is used for the clock. In this case the clock will not be generated. So using 4 will be the same a 0 (MSB shifted in first when clock goes low) but the clock must be generated by an external signal. 4 – MSB shifted in first when clock goes low with ext. clock 5 – MSB shifted in first when clock goes high with ext. clock 6 – LSB shifted in first when clock goes low with ext. clock 7 – LSB shifted in first when clock goes high with ext. clock
Bits	Optional number of bits to shift in. Maximum 255.
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified. When the number of bits is default you can use NULL for the BITS parameter.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.

When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips.

The PIN is normally connected with the output of chip that will send information.

The PCLOCK pin can be used to clock the bits as a master, that is the clock pulses will be generated. Or it can sample a pin that generates these pulses.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data read from the chip is stored in this variable.

The OPTIONS is a constant that specifies the direction of the bits. The chip that outputs the data may send the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data must be stored.

When you add 4 to the constant you tell the compiler that the clock signal is not generated but that there is an external clock signal.

The number of bits may be specified. You may omit this info. In that case the number of bits of the element data type will be used.

The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

See also

SHIFTOUT , SHIFT

Example

```
Dim A As Byte
Config Pinb.0 = Input ' set pin to input
Config Pinb.1 = Output
Portb.0 = 1
Shiftin Pinb.0 , Portb.1 , A , 4 , 4 , 10 'shiftin 4 bits and use
external clock
Shift A , Right , 4 'adjust
Shiftin Pinb.0 , Portb.1 , A 'read 8 bits

End
```

SHIFTOUT

Action

Shifts a bit stream out of a variable into a port pin .

Syntax

SHIFTOUT pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as a data output.
Pclock	The port pin which generates the clock.
Var	The variable that is shifted out.
Option	Option can be : 0 – MSB shifted out first when clock goes low 1 – MSB shifted out first when clock goes high 2 – LSB shifted out first when clock goes low 3 – LSB shifted out first when clock goes high
Bits	Optional number of bits to shift out.
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified. When the default must be used you can also use NULL for the number of bits.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.

When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips.

The PIN is normally connected with the input of a chip that will receive information.

The PCLOCK pin is used to clock the bits out of the chip.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data that is stored in the variableis sent with PIN.

The OPTIONS is a constant that specifies the direction of the bits. The chip that reads the data may want the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data is sent to PIN.

The number of bits may be specified. You may omit this info. In that case the number of bits of the element data type will be used.

The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

See also

SHIFTIN , SHIFT

Example

```
Dim a as byte
Config Pinb.0 = Output
Config Pinb.1 = Input
Shiftout Portb.0 , Portb.1 , A , 3 , 4 , 10 'shiftout 4 bits
```

```
Shiftin Pinb.0 , Portb.1 , A , 3 'shiftout 8 bits
End
```

SHIFTLCD

Action

Shift the LCD display left or right by one position.

Syntax

SHIFTLCD LEFT / RIGHT

Remarks

NONE

See also

SHIFTCURSOR

Example

```
Cls
Lcd "Very long text"
Shiftlcd Left
Wait 1
Shiftlcd Right
End
```

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 a 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.

You can store multiple pictures when you use multiple labels and $BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also

PSET , $BGF , CONFIG GRAPHLCD , LINE , CIRCLE , SHOWPICE

Example

```
'-----------------------------------------------------------------
' (c) 2001 MCS Electronics
' T6963C graphic display support demo
'-----------------------------------------------------------------

'The connections of the LCD used in this demo
'LCD pin connected to
' 1 GND GND
'2 GND GND
'3 +5V +5V
'4 -9V -9V potmeter
'5 /WR PORTC.0
'6 /RD PORTC.1
'7 /CE PORTC.2
'8 C/D PORTC.3
'9 NC not conneted
'10 RESET PORTC.4
'11-18 D0-D7 PA
'19 FS PORTC.5
'20 NC not connected



'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 64 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5
'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2


'Dim variables (y not used)
Dim X As Byte , Y As Byte


'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part

'locate works like the normal LCD locate statement
' LOCATE LINE,COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1

'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"

'wait 1 sec
Wait 1


' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
Pset X , 20 , 255 ' set the pixel
Next

Wait 1


'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje

Wait 1
Cls Text ' clear the text
End



'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"

'You could insert other picture data here
```

Label:

$BGF "mcs.bgf" 'data will be inserted here

SHOWPICE

Action

Shows a BGF file stored in EEPROM on the graphic display

Syntax

SHOWPICE x, y , label

Remarks

Showpice can display a converted BMP file that is stored in the EEPROM of the micro processor. 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 a 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.

You can store multiple pictures when you use multiple labels and $BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also

PSET , $BGF , CONFIG GRAPHLCD , LINE , SHOWPIC , CIRCLE

Example

```
'------------------------------------------------------------------------
-------
' showpicE.bas
' demonstrates showing a picture from EEPROM
'------------------------------------------------------------------------
-------
$crystal = 8000000
$regfile = "8535def.dat"
'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce
= 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8
'The dataport is th e portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

'we will load the picture data into EEPROM so we specify $EEPROM
'the data must be specified before the showpicE statement.
$eeprom
Plaatje:
'the $BGF directive will load the data into the EEPROM or FLASH depending
on the $EEPROM or $DATA directive
$bgf "mcs.bgf"
'switch back to normal DATA (flash) mode
$data

'Clear the screen will both clear text and graph display
Cls
'showpicE is used to show a picture from EEPROM
'showpic must be used when the data is located in Flash
Showpice 0 , 0 , Plaatje
End
```

SIN

Action

Returns the sine of a single

Syntax

var =**SIN**(single)

Remarks

Var	A numeric variable that is assigned with sinus of variable single.
Single	The single variable to get the sinus of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , COS

Example

Show sample

SINH

Action

Returns the sinus hyperbole of a single

Syntax

var =**SINH** (single)

Remarks

Var	A numeric variable that is assigned with sinus hyperbole of variable single.
Single	The single variable to get the sinus hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , COS , SIN , TANH , COSH

Example

Show sample

SOCKETCONNECT

Action

Establishes a connection to a TCP/IP server.

Syntax

Result = **SocketConnect**(socket, IP, port)

Remarks

Result	A byte that is assigned with 0 when the connection succeeded. It will return 1 when an error occurred.
IP	The IP number of the server you want to connect to. This may be a number like 192.168.0.2 or a LONG variable that is assigned with an IP number. Note that the LSB of the LONG, must contain the MSB of the IP number.
Port	The port number of the server you are connecting to.

You can only connect to a server. Standardized servers have dedicated port numbers. For example, the HTTP protocol (web server) uses port 80.

After you have established a connection the server might send data. This depends entirely on the used protocol. Most servers will send some welcome text, this is called a banner.

You can send or receive data once the connection is established.

The server might close the connection after this or you can close the connection yourself. This also depends on the protocol.

See also

CONFIG TCPIP, GETSOCKET , SOCKETSTAT , TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN

Example

```
J = Socketconnect(i , 194.109.6.52 , 25) ' smtp server
```

SOCKETLISTEN

Action

Opens a socket in server(listen) mode.

Syntax

SocketListen socket

Remarks

Socket	The socket number you want to close in the range of 0 -3.

The socket will listen to the port you specified with the GetSocket function.

You can listen to a maximum of 4 sockets at the same time.

After the connection is closed by either the client or the server, a new connection need to be created and the SocketListen statement must be used again.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET

Example

```
I = Getsocket(0 , Sock_stream , 5000 , 0) ' get a new socket
Socketlisten I ' listen
#if Debug
Print "Listening on socket : " ; I
#endif
```

SOCKETSTAT

Action

Returns information of a socket.

Syntax

Result = **SocketStat**(socket , mode)

Remarks

Result	A word variable that is assigned with the result.
Socket	The socket number you want to get information of
Mode	A parameter that specified what kind of information you want to retrieve. SEL_CONTROL or 0 : returns the status register value SEL_SEND or 1 : returns the number of bytes that might be placed into the transmission buffer. SEL_RECV or 2 : returns the number of bytes that are stored in the reception buffer.

The SocketStat function contains actual 3 functions. One to get the status of the connection, one to determine how many bytes you might write to the socket, and one to determine how many bytes you can read from the buffer.

When you specify mode 0, one of the following byte values will be returned:

Value	State	Description
0	SOCK_CLOSED	Connection closed
1	SOCK_ARP	Standing by for reply after transmitting ARP request
2	SOCK_LISTEN	Standing by for connection setup to the client when acting in passive mode
3	SOCK_SYNSENT	Standing by for SYN,ACK after transmitting SYN for connecting setup when acting in active mode
4	SOCK_SYNSENT_ACK	Connection setup is complete after SYN,ACK is received and ACK is transmitted in active mode
5	SOCK_SYNRECV	SYN,ACK is being transmitted after receiving SYN from the client in listen state, passive mode
6	SOCK_ESTABLISHED	Connection setup is complete in active, passive mode
7	SOCK_CLOSE_WAIT	Connection being terminated
8	SOCK_LAST_ACK	Connection being terminated
9	SOCK_FIN_WAIT1	Connection being terminated
10	SOCK_FIN_WAIT2	Connection being terminated
11	SOCK_CLOSING	Connection being terminated
12	SOCK_TIME_WAIT	Connection being terminated
13	SOCK_RESET	Connection being terminated after receiving reset packet from peer.
14	SOCK_INIT	Socket initializing
15	SOCK_UDP	Applicable channel is initialized in UDP mode.
16	SOCK_RAW	Applicable channel is initialized in IP layer RAW mode
17	SOCK_UDP_ARP	Standing by for reply after transmitting ARP request packet to the destination for UDP transmission
18	SOCK_UDP_DATA	Data transmission in progress in UDP RAW mode
19	SOCK_RAW_INIT	W3100A initialized in MAC layer RAW mode

The SocketStat function is also used internally by the library.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, TCPWRITE, TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN

Example

```
Tempw = Socketstat(i , 0) ' get status
Select Case Tempw
Case Sock_established

End Select
```

SONYSEND

Action

Sends Sony remote IR code.

Syntax

SONYSEND address

Uses

TIMER1

Remarks

Address	The address of the Sony device.

SONY CD Infrared Remote Control codes (RM-DX55)

Function	Hex	Bin
Power	A91	1010 1001 0001
Play	4D1	0100 1101 0001
Stop	1D1	0001 1101 0001
Pause	9D1	1001 1101 0001
Continue	B91	1011 1001 0001
Shuffle	AD1	1010 1101 0001
Program	F91	1111 1001 0001
Disc	531	0101 0011 0001
1	011	0000 0001 0001
2	811	1000 0001 0001
3	411	0100 0001 0001
4	C11	1100 0001 0001
5	211	0010 0001 0001
6	A11	1010 0001 0001
7	611	0110 0001 0001
8	E11	1110 0001 0001
9	111	0001 0001 0001
0	051	0000 0101 0001
>10	E51	1110 0101 0001
enter	D11	1101 0001 0001
clear	F11	1111 0001 0001
repeat	351	0011 0101 0001
disc -	BD1	1011 1101 0001
disc +	H7D1	0111 1101 0001
|<<	0D1	0000 1101 0001
>>|	8D1	1000 1101 0001
<<	CD1	1100 1101 0001
>>	2D1	0010 1101 0001
SONY Cassette	RM-J901)	
Deck A		
stop	1C1	0001 1100 0001
play >	4C1	0100 1100 0001
play <	EC1	1110 1100 0001
>>	2C1	0010 1100 0001
<<	CC1	1100 1100 0001
record	6C1	0110 1100 0001
pause	9C1	1001 1100 0001
Dec B		
stop	18E	0001 1000 1110
play >	58E	0101 1000 1110
play <	04E	0000 0100 1110
>>	38E	0011 1000 1110
<<	D8E	1101 1000 1110
record	78E	0111 1000 1110
pause	98E	1001 1000 1110

--[SONY TV Infrared Remote Control codes (RM-694)]--------------------------

program + = &H090 : 0000 1001 0000
program - = &H890 : 1000 1001 0000
volume + = &H490 : 0100 1001 0000
volume - = &HC90 : 1100 1001 0000
power = &HA90 : 1010 1001 0000
sound on/off = &H290 : 0010 1001 0000
1 = &H010 : 0000 0001 0000
2 = &H810 : 1000 0001 0000
3 = &H410 : 0100 0001 0000
4 = &HC10 : 1100 0001 0000
5 = &H210 : 0010 0001 0000
6 = &HA10 : 1010 0001 0000
7 = &H610 : 0110 0001 0000
8 = &HE10 : 1110 0001 0000
9 = &H110 : 0001 0001 0000
0 = &H910 : 1001 0001 0000
-/-- = &HB90 : 1011 1001 0000

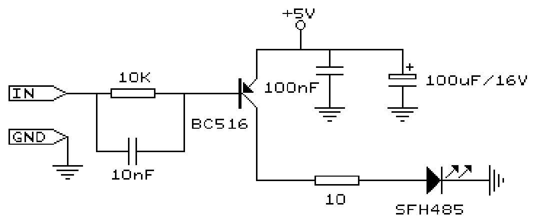
For more SONY Remote Control info:

http://www.fet.uni-hannover.de/purnhage/

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a datasheet for the proper pin when used with a different chip.

An IR booster circuit is shown below:



See also

CONFIG RC5 , GETRC5

Example

```
'-----------------------------------------------------------------
' SONYSEND.BAS
' (c) 2004 MCS Electronics
' code based on application note from Ger Langezaal
' +5V <--[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3
'-----------------------------------------------------------------
$regfile = "2313def.dat"
$crystal = 4000000


Do
Waitms 500
Sonysend &HA90
Loop

End
```

SOUND

Action

Sends pulses to a port pin.

Syntax

SOUND pin, duration, pulses

Remarks

Pin	Any I/O pin such as PORTB.0 etc.
Duration	The number of pulses to send. Byte, integer/word or constant.
Pulses	The time the pin is pulled low and high. This is the value for a loop counter.

When you connect a speaker or a buzzer to a port pin (see hardware) , you can use the SOUND statement to generate some tones.

The port pin is switched high and low for *pulses* times.

This loop is executed *duration* times.

The SOUND statement is not intended to generate accurate frequencies. Use a TIMER to do that.

See also

NONE

Example

```
SOUND PORTB.1 , 10000, 10 'BEEP
End
```

SPACE

Action

Returns a string that consists of spaces.

Syntax

var = **SPACE(**x **)**

Remarks

X	The number of spaces.
Var	The string that is assigned.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

See also

STRING

Example

```
Dim s as String * 15, z as String * 15
s = Space(5)
Print " {" ;s ; " }" '{  }

Dim A as Byte
A = 3
S = Space(a)

End
```

SPC

Action

Prints the number of specified spaces.

Syntax

PRINT **SPC(**x **)**

Remarks

X	The number of spaces to print.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

SPC can be used with LCD too.

The difference with the SPACE function is that SPACE returns a number of spaces while SPC() can only be used with printing. Using SPACE() with printing is also possible but it will use a temporary buffer while SPC does not use a temporary buffer.

See also

SPACE

Example

```
Dim s as String * 15, z as String * 15
Print "{" ; SPC(5) ; "}"    '{  }
LCD "{" ; SPC(5) ; "}"    '{  }
```

SPIIN

Action

Reads a value f rom the SPI -bus.

Syntax

SPIIN var, bytes

Remarks

Var	The variable which receives the value read from the SPI-bus.
Bytes	The number of bytes to read.

See also

SPIOUT, SPIINIT, CONFIG SPI , SPIMOVE

Example

```
Dim A(10) As Byte
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 , Clock
= Portb.3
Spiinit
Spiin A(1) , 4 'read 4 bytes and store in a(1), a(2) , a(3) and a(4)

End
```

SPIINIT

Action

Initiate the SPI pins.

Syntax

SPIINIT

Remarks

After the configuration of the SPI pins, you must initialize the SPI pins to set them for the right data direction. When the pins are not used by other hardware/software, you only need to use SPIINIT once.

When other routines change the state of the SPI pins, use SPIINIT again before using SPIIN and SPIOUT.

See also

SPIIN , SPIOUT

ASM

Calls _init_spi

Example

```
Dim A(10) As Byte
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 , Clock
= Portb.3
Spiinit
Spiin A(1) , 4 'read 4 bytes and store in a(1), a(2) , a(3) and a(4)

End
```

SPIOUT

Action

Sends a value of a variable to the SPI-bus.

Syntax

SPIOUT var , bytes

Remarks

var	The variable whose content must be send to the SPI-bus.
bytes	The number of bytes to send.

See also

SPIIN , SPIINIT , CONFIG SPI , SPIMOVE

Example

```
Dim A(10) As Byte
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 , Clock
= Portb.3
Spiinit
Spiout A(1) , 4 'write 4 bytes a(1), a(2) , a(3) and a(4)

End
```

SQR

Action

Returns the Square root of a variable.

Syntax

var =**SQR**(single)

Remarks

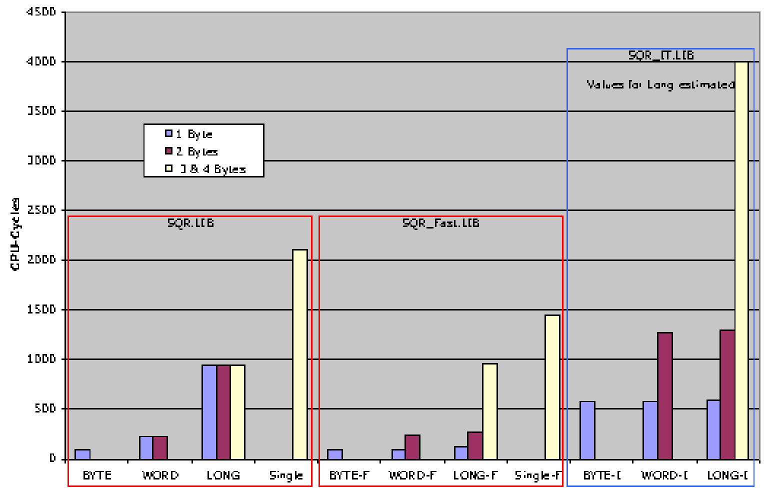
var	A numeric single variable that is assigned with the SQR of variable single.
single	The single variable to get the SQR of.

When SQR is used with a single, the FP_TRIG library will be used.

When SQR is used with bytes, integers, words and longs, the SQR routine from MCS.LBX will be used.

As an alternative you can use the library SQR_IT.LBX or SQR.LBX. By default, the code from FP_TRIG.LIB will be used.

Different algorithm's can be used to calculate the SQR. By default the fast algorithm code is used. The following picture shows the difference for the three methods:



Example

```
Dim A As Single
A = 9.0
A = Sqr(A)
Print A ' prints 3.0
```

START

Action

Start the specified device.

Syntax

START device

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power) or ADC(A/D converter power) .

You must start a timer/counter in order for an interrupt to occur (when the external gate is disabled).

TIMER0 and COUNTER0 are the same device.
The AC and ADC parameters will switch power to the device and thus enabling it to work.

See also

STOP

Example

```
'--------------------------------------------------------------------
' ADC.BAS
' demonstration of GETADC() function for 8535 micro
'--------------------------------------------------------------------
$regfile = "m163def.dat"

'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

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

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

Dim W As Word , Channel As Byte

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

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF : AREF, internal reference turned off
'AVCC : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin

'Using the additional param on chip that do not have the internal
reference will have no effect.
```

STCHECK

Action

Calls a routine to check for various stack overflows. This routine is intended for debug purposes.

Syntax

STCHECK

Remarks

The different stack spaces used by BASCOM-AVR lead to lots of questions about them.

The STCHECK routine can help to determine if the stack size are trashed by your program. The program STACK.BAS is used to explain the different settings.

Note that STCHECK should be removed form your final program. That is once you tested your program and found out is works fine, you can remove the call to STCHECK since it costs time and code space.

The settings used are :

HW stack 8

Soft stack 2

Frame size 14

Below is a part of the memory of the 90S2313 used for the example:

```
C0  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA  CB  CC  CD  CE  CF
D0  D1  D2  D3  D4  D5  D6  D7  D8  D9  DA  DB  DC  DD  DE  DF
FR  FR  FR  FR  FR  FR  FR  FR
FR  FR  FR  FR  FR  FR  YY  YY  SP  SP  SP  SP  SP  SP  SP  SP
```

Since the last memory in SRAM is DF, the hardware stack is occupied by D8-DF(8 bytes)

When a call is made or a push is used the data is saved at the position the hardware stack pointer is pointing to. After this the stack pointer is decreased.

A call uses 2 bytes so SP will be SP-2. (DF-2) =DD

When 8 bytes are stored the SP will point to D7. Another call or push will thus destroy memory position D7 which is occupied by the soft stack.

The soft stack begins directly after the hardware stack and is also growing down.

The Y pointer(r28+r29) is used to point to this data.

Since the Y pointer is decreased first and then the data is saved, the pointer must point at start up to a position higher. That is D8, the end of the hardware space.

St -y,r24 will point to D8-1=D7 and will store R24 at location D7.

Since 2 bytes were allocated in this example we use D7 and D6 to store the data.

When the pointer is at D6 and another St -y,r24 is used, it will write to position D5 which

is the end of the frame space that is used as temporarily memory.

The frame starts at C8 and ends at D5. Writing beyond will overwrite the soft stack.

And when there is no soft stack needed, it will overwrite the hardware stack space.

The map above shows FR(frame), YY(soft stack data) and SP(hardware stack space)

How to determine the right values?

The stack check routine can be used to determine if there is an overflow.

It will check :

-if SP is below it's size. In this case below D8.

-if YY is below it's size in this case when it is D5

-if the frame is above its size in this case D6

When is YY(soft stack) used? When you use a LOCAL variable inside a SUB or function. Each local variable will use 2 bytes.

When you pass variables to user Subroutines or functions it uses 2 bytes for each parameter.

call mysub(x,y) will use 2 * 2 = 4 bytes.

local z as byte ' will use another 2 bytes

This space is freed when the routine ends.

But when you call another sub inside the sub, you need more space.

sub mysub(x as byte,y as byte)

call testsub(r as byte) ' we must add another 2 bytes

When you use empty(no params) call like :

call mytest() , No space is used.

When do you need frame space?

When ever you use a num<>string conversion routine like:

Print b (where b is a byte variable)

Bytes will use 4 bytes max (123+0)

Integer will use 7 bytes max (-12345+0)c

Longs will use 16 bytes max

And the single will use 24 bytes max

When you add strings and use the original the value must be remembered by the compiler.

Consider this :

s$ = "abcd" + s$

Here you give s$ a new value. But you append the original value so the original value must be remembered until the operation has completed. This copy is stored in the frame too.

So when string s$ was dimmed with a length of 20, you need a frame space of 20+1(null byte)

When you pass a variable by VALUE (BYVAL) then you actually pass a copy of the variable.

When you pass a byte, 1 byte of frame space is used, a long will take 4 bytes.

When you use a LOCAL LONG, you also need 4 bytes of frame space to store the local long.

The frame space is reused and so is the soft stack space and hardware stack space.

So the hard part is to determine the right sizes!

The stack check routine must be called inside the deepest nested sub or function.

Gosub test

test:
gosub test1
return

test1:
' this is the deepest level so check the stack here
stcheck
return

Stcheck will use 1 variable named ERROR. You must dimension it yourself.
Dim Error As Byte

Error will be set to :
1: if hardware stack grows down into the soft stack space
2: if the soft stack space grows down into the frame space
3: if the frame space grows up into the soft stack space.

The last 2 errors are not necessarily bad when you consider that when the soft stack is not used for passing data, it may be used by the frame space to store data. Confusing right.?

ASM

Routines called by STCHECK :
_StackCheck : uses R24 and R25 but these are saved and restored.
Because the call uses 2 bytes of hardware stack space and the saving of R24 and R25 also costs 2 bytes, it uses 4 more bytes of hardware stack space than your final program would do that f course does not need to use STCHECK.

Example

Here is the stack.bas sample that can be found in the samples dir.
It uses conditional compilation so you can test the various errors.

```
'this sample shows how to check for the stack sizes

'note that the called routine (_STACKCHECK) will use 4 bytes
'ofhardware stack space
'So when your program works, you may subtract the 4 bytes of the needed
hardware stack size
'in your final program that does not include the STCHECK

'testmode =0 will work
'testmode =1 will use too much hardware stack
'testmode =2 will use too much soft stack space
'testmode =3 will use too much frame space
Const Testmode = 0
'compile and test the program with testmode from 0-3


'you need to dim the ERROR byte !!
Dim Error As Byte

#if Testmode = 2
Declare Sub Pass(z As Long , Byval K As Long)
#else
Declare Sub Pass()
#endif


Dim I As Long
I = 2
Print I
'call the sub in your code at the deepest level
'normally within a function or sub


#if Testmode = 2
Call Pass(i , 1)
#else
Call Pass()
#endif
End


#if Testmode = 2
Sub Pass(z As Long , Byval K As Long)
#else
Sub Pass()
#endif
#if Testmode = 3
Local S As String * 13
#else
Local S As String * 8
#endif

Print I
Gosub Test
End Sub


Test:
#if Testmode = 1
push r0 ; eat some hardware stack space
push r1
push r2
#endif

' *** here we call the routine ***
Stcheck
' *** when error <>0 then there is a problem ***
#if Testmode = 1
pop r2
pop r1
pop r0
#endif

Return
```

STOP

Action

Stop the specified device. Or stop the program

Syntax

STOP device

STOP

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power) or ADC(A/D converter power) .

The single STOP statement will end your program by generating a never ending loop. When END is used it will have the same effect but in addition it will disable all interrupts.

The STOP statement with one of the above parameters will stop the specified device.

TIMER0 and COUNTER0 are the same device.

The AC and ADC parameters will switch power off the device to disable it and thus save power.

See also

START , END

Example

```
'--------------------------------------------------------------------
' ADC.BAS
' demonstration of GETADC() function for M163 micro
'--------------------------------------------------------------------
$regfile = "m163def.dat"

'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

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

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

Dim W As Word , Channel As Byte

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

'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
'OFF : AREF, internal reference turned off
'AVCC : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin

'Using the additional param on chip that do not have the internal
reference will have no effect.
```

STR

Action

Returns a string representation of a number.

Syntax

var **= Str(** x **)**

Remarks

var	A string variable.
X	A numeric variable.

The string must be big enough to store the result.

See also

VAL , HEX, HEXVAL , MCSBYTE , BIN

Difference with QB

In QB STR() returns a string with a leading space. BASCOM does not return a leading space.

Example

```
Dim A As Byte , S As String * 10
A = 123
S = Str(a)
Print S ' 123
End
```

STRING

Action

Returns a string consisting of m repetitions of the character with ASCII Code n.

Syntax

var = **STRING(**m ,n **)**

Remarks

Var	The string that is assigned.
N	The ASCII-code that is assigned to the string.
M	The number of characters to assign.

Since a string is terminated by a 0 byte, you can't use 0 for n.

Using 0 for m will result in a string of 255 bytes, because there is no check on a length assign of 0.

See also

SPACE

Example

```
Dim S As String * 15
S = String(5 , 65 )
Print S 'AAAAA
End
```

SPIMOVE

Action

Sends and receives a value or a variable to the SPI -bus.

Syntax

var = SPIMOVE(byte)

Remarks

Var	The variable that is assigned with the received byte(s) from the SPI-bus.
Byte	The variable or constant whose content must be send to the SPI-bus.

See also

SPIIN , SPIINIT , CONFIG SPI

Example

```
CONFIG SPI = SOFT, DIN = PINB.0, DOUT = PORTB.1, SS=PORTB.2, CLOCK =
PORTB.3
SPIINIT
Dim a(10) as Byte , X As Byte
SPIOUT a(1) , 5 'send 5 bytes
SPIOUT X , 1 'send 1 byte
A(1) = SpiMove(5) ' move 5 to SPI and store result in a(1)
End
```

SUB

Action

Defines a Sub procedure.

Syntax

SUB Name[(var1 , …)**]**

Remarks

Name	Name of the sub procedure, can be any non-reserved word.
var1	The name of the parameter.

You must end each subroutine with the END SUB statement.

You can copy the DECLARE SUB line and remove the DECLARE statement. This ensures that you have the right parameters.

See the DECLARE SUB topic for more details.

SYSSEC

Action

Returns a Number, which represents the System Second

Syntax

Target = **SysSec**()

Target = **SysSec**(bSecMinHour)

Target = **SysSec**(strTime, strDate)

Target = **SysSec**(wSysDay)

Remarks

Target	A Variable (LONG), that is assigned with the System-Second
BSecMinHour	A Byte, which holds the Sec-value followed by Min(Byte), Hour (Byte), Day(Byte), Month(Byte) and Year(Byte)
StrTime	A time-string in the format „hh:mm:ss"
StrDate	A date-string in the format specified in the Config Date statement
wSysDay	A variable (Word) which holds the System Day (SysDay)

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Time and Date of SOFTCLOCK (_sec, _min, _hour, _day, _month, _year) is used.
2. With a user defined time and Date array. It must be arranged in same way (Second, Minute, Hour, Day, Month, Year) as the internal SOFTCLOCK time/date. The first Byte (Second) is the input by this kind of usage. So the System Second can be calculated of every time/date.
3. With a time-String and a date-string. The time-string must be in the Format „hh:mm:ss". The date-string must be in the format specified in the Config Date statement
4. With a System Day Number (Word). The result ist the System Second of this day at 00:00:00.

The Return-Value is in the Range of 0 to 2147483647. 2000-01-01 at 00:00:00 starts with 0.

The Function is valid from 2000-01-01 to 2068-01-19 03:14:07. In the year 2068 a LONG – overflow will occur.

See also

[Date and Time Routines](), [SYSSECELAPSED](), [SYSDAY]()

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format
Dim strDate as String * 8
Dim strtime as String * 8
Dim bSec as Byte, bMin as Byte, bHour as Byte
Dim bDay as Byte , bMonth as Byte , bYear as Byte
Dim wSysDay as Word
Dim lSysSec as Long


' Example 1 with internal RTC-Clock
' Load RTC-Clock for example - testing
_Sec = 17 : _Min = 35 : _Hour = 8 : _Day = 16 : _Month = 4 : _Year = 3
lSysSec = SysSec()
print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; lSysSec
' System Second of 08:35:17 at 03.04.16 is 103797317


' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
bSec = 20: bMin = 1: bHour = 7 : bDay = 22 : bMonth = 12 : bYear = 1
lSysSec = SysSec(bSec)
strTime = time_sb(bSec): strDate = date_sb(bDay)
print "System Second of " ; strTime ; " at " ; strDate ; " is " ; lSysSec
' System Second of 07:01:20 at 01.12.22 is 62319680


' Example 3 with Time and Date - String
strTime = "04:58:37"
strDate = "02.09.18"
lSysSec = SysSec(strTime, strDate)
print "System Second of " ; strTime ; " at " ; strDate ; " is " ; lSysSec
' System Second of 04:58:37 at 02.09.18 is 85640317



' Example 4 with System Day
wSysDay = 2000
lSysSec = SysSec(wSysDay)
print "System Second of System Day " ; wSysDay ; " (00:00:00) is " ;
lSysSec
' System Second of System Day 2000 (00:00:00) is 172800000
```

SYSSECELAPSED

Action

Returns the elapsed Seconds to a earlier assigned system-time-stamp.

Syntax

Target = **SysSecElapsed**(SystemTimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
SystemTimeStamp	A variable (LONG), which holds a Systemtimestamp like the output of an earlier called SysSec()

The Return-Value is in the Range of 0 to 2147483647. The Function is valid from 2000-01-01 to 2068-01-19 at 03:14:07. In the year 2068 a LONG – overflow will occur.

The difference to the pair DayOfSec and SecElapsed is, that SysSec and SysSecElapsed can be used for event distances larger than 24 hours.

See also

Date and Time Routines , SECELAPSED, SYSSEC

Example

```
Enable Interrupts
Config Clock = Soft

Dim lSystemTimeStamp as Long
Dim lSystemSecondsElapsed as Long

lSystemTimeStamp = SysSec()
Print "Now it's " ; lSystemTimeStamp ; " seconds past 2000-01-01
00:00:00"

' do other stuff
' some time later

lSystemSecondsElapsed = SysSecElapsed(lSystemTimeStamp)
Print "Now it's " ; lSystemSecondsElapsed ; " seconds later"
```

SYSDAY

Action

Returns a number, which represents the System Day

Syntax

Target = **SysDay**()

Target = **SysDay**(bDayMonthYear)

Target = **SysDay**(strDate)

Target = **SysDay**(lSysSec)

Remarks

Target	A Variable (LONG), that is assigned with the System-Day
bDayMonthDay	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
strDate	A String, which holds a Date-String in the format specified in the **CONFIGDATE** statement
lSysSec	A variable, which holds a System Second (SysSec)

The Function can be used with 4 different kind of inputs:

1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
4. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 36524. 2000-01-01 starts with 0.

The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also

Date and Time Routines , ConfigDate , Config Clock , SysSec

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator = . ' ANSI-Format
Dim strDate as String * 8
Dim bDay as Byte , bMonth as Byte , bYear as Byte
```

```
Dim wSysDay as Word
Dim lSysSec as Long

' Example 1 with internal RTC-Clock
_day = 20 : _Month = 11 : _Year = 2 ' Load RTC-Clock for example -
testing
wSysDay = SysDay()
print "System Day of " ; Date$ ; " is " ; wSysDay
' System Day of 02.11.20 is 1054


' Example 2 with defined Clock - Bytes (Day / Month / Year)
bDay = 24 : bMonth = 5 : bYear = 8
wSysDay = SysDay(bDay)
print "System Day of Day="; bDay ; " Month="; bMonth ; " Year=" ; bYear ;
" is "; wSysDay
' System Day of Day=24 Month=5 Year=8 is 3066

' Example 3 with Date - String
strDate = "04.10.29"
wSysDay = SysDay(strDate)
print "System Day of " ; strDate ; " is "; wSysDay
' System Day of 04.10.29 is 1763


' Example 4 with System Second
lSysSec = 123456789
wSysDay = SysDay(lSysSec)
print "System Day of System Second " ; lSysSec ; " is " ; wSysDay
' System Day of System Second 123456789 is 1428
```

SWAP

Action

Exchange two variables of the same type.

Syntax

SWAP var1, var2

Remarks

var1	A variable of type bit, byte, integer, word, long or string.
var2	A variable of the same type as var1.

After the swap, var1 will hold the value of var2 and var2 will hold the value of var1.

Example

```
Dim A As Integer , B1 As Integer
A = 1 : B1 = 2 'assign two integers
Swap A , B1 'swap them
Print A ; B1 'prints 21

End
```

TAN

Action

Returns the tangent of a single

Syntax

var =**TAN** (single)

Remarks

Var	A numeric variable that is assigned with tangent of variable single.
Single	The single variable to get the tangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , COS , SIN

Example

Show sample

TCPREAD

Action

Reads data from an open socket connection.

Syntax

Result = **TCPread**(socket , var, bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually received from the socket. When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3).
Var	The name of the variable that will be assigned with the data from the socket.
Bytes	The number of bytes to read. Only valid for non-string variables.

When you use TCPread with a string variable, the routine will wait for CR + LF and it will return the data without the CR + LF.

For strings, the function will also not overwrite the string.

For example, your string is 10 bytes long and the line you receive is 80 bytes long, you will receive only the first 10 bytes after CR + LF is encountered.

Also, for string variables, you do not need to specify the number of bytes to read since the routine will wait for CR + LF.

For other data types you need to specify the number of bytes.

There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, CLOSESOCKET , SOCKETLISTEN

Example

NONE

TCPWRITE

Action

Write data to a socket.

Syntax

Result = **TCPwrite**(socket , var , bytes)

Result = **TCPwrite**(socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
Socket	The socket number you want to send data to(0-3).
Var	A constant string like "test" or a variable. When you send a constant string, the number of bytes to send does not need to be specified.
Bytes	A word variable or numeric constant that specifies how many bytes must be send.
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.
EPROM	An indication for the compiler so it knows that you will send data fromEPROM.

The TCPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.

When you want to send data from an array, you need to specify the element : var(idx) for example.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITESTR , TCPREAD , CLOSESOCKET , SOCKETLISTEN

Example

```
Tempw = Tcpwrite(i , "HTTP/1.0 200 OK{013}{010}")
```

TCPWRITESTR

Action

Sends a string to an open socket connection.

Syntax

Result = **TCPwriteStr**(socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
Socket	The socket number you want to send data to (0-3).
Var	The name of a string variable.
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF. This option was added because many protocols expect CR + LF after the string.

The TCPwriteStr function is a special variant of the TCPwrite function.

It will use TCPWrite to send the data.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPREAD, CLOSESOCKET , SOCKETLISTEN

Example

```
'-----------------------------------------------------------------
' SMTP.BAS
' (c) 2004 MCS Electronics
' sample that show how to send an email with SMTP protocol
'-----------------------------------------------------------------

$regfile = "m161def.dat" ' used processor
$crystal = 4000000 ' used crystal
$baud = 19200 ' baud rate
$lib "tcpip.lbx" ' specify the name of the tcp ip lib

'W3100A constants
Const Sock_stream = $01 ' Tcp
Const Sock_dgram = $02 ' Udp
Const Sock_ipl_raw = $03 ' Ip Layer Raw Sock
```

```
Const Sock_macl_raw = $04 ' Mac Layer Raw Sock
Const Sel_control = 0 ' Confirm Socket Status
Const Sel_send = 1 ' Confirm Tx Free Buffer Size
Const Sel_recv = 2 ' Confirm Rx Data Size

'socket status
Const Sock_closed = $00 ' Status Of Connection Closed
Const Sock_arp = $01 ' Status Of Arp
Const Sock_listen = $02 ' Status Of Waiting For Tcp Connection Setup
Const Sock_synsent = $03 ' Status Of Setting Up Tcp Connection
Const Sock_synsent_ack = $04 ' Status Of Setting Up Tcp Connection
Const Sock_synrecv = $05 ' Status Of Setting Up Tcp Connection
Const Sock_established = $06 ' Status Of Tcp Connection Established
Const Sock_close_wait = $07 ' Status Of Closing Tcp Connection
Const Sock_last_ack = $08 ' Status Of Closing Tcp Connection
Const Sock_fin_wait1 = $09 ' Status Of Closing Tcp Connection
Const Sock_fin_wait2 = $0a ' Status Of Closing Tcp Connection
Const Sock_closing = $0b ' Status Of Closing Tcp Connection
Const Sock_time_wait = $0c ' Status Of Closing Tcp Connection
Const Sock_reset = $0d ' Status Of Closing Tcp Connection
Const Sock_init = $0e ' Status Of Socket Initialization
Const Sock_udp = $0f ' Status Of Udp
Const Sock_raw = $10 ' Status of IP RAW
Const Debug = -1 ' for sending feedback to the terminal


#if Debug
Print "Start of SMTP demo"
#endif

Enable Interrupts ' enable interrupts
'specify MAC, IP, submask and gateway
'local port value will be used when you do not specify a port value while
creating a connection
'TX and RX are setup to use 4 connections each with a 2KB buffer
Config Tcpip = Int0 , Mac = 00.44.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx =
$55 , Rx = $55

'dim the used variables
Dim S As String * 50 , I As Byte , J As Byte , Tempw As Word
#if Debug
Print "setup of W3100A complete"
#endif

'First we need a socket
I = Getsocket(0 , Sock_stream , 5000 , 0)
' ^ socket numer ^ port
#if Debug
Print "Socket : " ; I
'the socket must return the asked socket number. It returns 255 if there
was an error
#endif

If I = 0 Then ' all ok
'connect to smtp server
J = Socketconnect(i , 194.000.000.0002 , 25) ' smtp server and SMTP port
25
' ^socket
' ^ ip address of the smtp server
' ^ port 25 for smtp
#if Debug
Print "Connection : " ; J
Print S_status(1)
#endif
If J = 0 Then ' all ok
#if Debug
Print "Connected"
#endif
Do
Tempw = Socketstat(i , 0) ' get status
Select Case Tempw
Case Sock_established ' connection established
Tempw = Tcpread(i , S) ' read line
#if Debug
Print S ' show info from smtp server
#endif
If Left(s , 3) = "220" Then ' ok
Tempw = Tcpwrite(i , "HELO xxxxxx{013}{010}" ) ' send username
#if Debug
Print Tempw ; " bytes written" ' number of bytes actual send
#endif
Tempw = Tcpread(i , S) ' get response
#if Debug
Print S ' show response
#endif
If Left(s , 3) = "250" Then ' ok
Tempw = Tcpwrite(i , "MAIL FROM:<tcpip@mcselec.com>{013}{010}") ' send
from address
Tempw = Tcpread(i , S) ' get response
#if Debug
Print S
#endif
If Left(s , 3) = "250" Then ' ok
Tempw = Tcpwrite(i , "RCPT TO:<tcpip@mcselec.com>{013}{010}") ' send TO
address
Tempw = Tcpread(i , S) ' get response
#if Debug
Print S
#endif
If Left(s , 3) = "250" Then ' ok
Tempw = Tcpwrite(i , "DATA{013}{010}") ' speicfy that we are going to
send data
Tempw = Tcpread(i , S) ' get response
#if Debug
Print S
#endif
If Left(s , 3) = "354" Then ' ok
Tempw = Tcpwrite(i , "From: tcpip@mcselec.com{013}{010}")
Tempw = Tcpwrite(i , "To: tcpip@mcselec.com{013}{010}")
Tempw = Tcpwrite(i , "Subject: BASCOM SMTP test{013}{010}")
Tempw = Tcpwrite(i , "X-Mailer: BASCOM SMTP{013}{010}")
Tempw = Tcpwrite(i , "{013}{010}")
Tempw = Tcpwrite(i , "This is a test email from BASCOM SMTP{013}{010}")
Tempw = Tcpwrite(i , "Add more lines as needed{013}{010}")
Tempw = Tcpwrite(i , ".{013}{010}") ' end with a single dot

Tempw = Tcpread(i , S) ' get response
#if Debug
Print S
#endif
If Left(s , 3) = "250" Then ' ok
```

```
Tempw = Tcpwrite(i , "QUIT{013}{010}") ' quit connection
Tempw = Tcpread(i , S)
#if Debug
Print S
#endif
End If
End If
End If
End If
End If
End If
Case Sock_close_wait
Print "CLOSE_WAIT"
Closesocket I ' close the connection
Case Sock_closed
Print "Socket CLOSED" ' socket is closed
End
End Select
Loop
End If
End If
End 'end program
```

TANH

Action

Returns the hyperbole of a single

Syntax

var =**TANH** (single)

Remarks

Var	A numeric variable that is assigned with hyperbole of variable single.
Single	The single variable to get the hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG , DEG2RAD , ATN , COS , SIN , SINH , COSH

Example

Show sample

THIRDLINE

Action

Reset LCD cursor to the third line.

Syntax

THIRDLINE

Remarks

NONE

See also

UPPERLINE , LOWERLINE , FOURTHLINE

Example

```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

TIME$

Action

Internal variable that holds the time.

Syntax

TIME$ = "hh:mm:ss"

var = TIME$

Remarks

The TIME$ variable is used in combination with the CONFIG CLOCK and CONFIG DATE directive.

The CONFIG CLOCK statement will use the TIMER0 or TIMER2 in async mode to create a 1 second interrupt. In this interrupt routine the _Sec, _Min and _Hour variables are updated. The time format is 24 hours format.

When you assign TIME$ to a string variable these variables are assigned to the TIME$ variable.

When you assign the TIME$ variable with a constant or other variable, the _sec, _Hour and _Min variables will be changed to the new time.

The only difference with QB/VB is that all digits must be provided when assigning the time. This is done for minimal code. You can change this behavior of course.

ASM

The following asm routines are called from mcs.lib.

When assiging TIME$: _set_time (calls _str2byte)

When reading TIME$: _make_dt (calls _byte2str)

See also

DATE$, CONFIG CLOCK

CONFIG DATE

Example

```
'-----------------------------------------------------------
' MEGACLOCK.BAS
' (c) 2000-2001 MCS Electronics
'-----------------------------------------------------------
'This example shows the new TIME$ and DATE$ reserved variables
'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32.768 KHz xtal to the timer
'And of course some BASCOM code

'This example is written for the STK300 with M103
Enable Interrupts

'[configure LCD]
$lcd = &HC000 'address for E and RS
$lcdrs = &H8000 'address for only E
Config Lcd = 20 * 4 'nice display from bg micro
Config Lcdbus = 4 'we run it in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus 'tell about the bus mode

'[now init the clock]
Config Clock = Soft 'this is how simple it is
```

```
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore

'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"

'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used

Time$ = "02:20:00"

'clear the LCD display
Cls

Do
Home 'cursor home
Lcd Date$ ; "  " ; Time$ 'show the date and time
Loop

'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

TIME

Action

Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target

Syntax

bSecMinHour = **Time**(lSecOfDay)

bSecMinHour = **Time**(lSysSec)

bSecMinHour = **Time**(strTime)

strTime = **Time**(lSecOfDay)

strTime = **Time**(lSysSec)

strTime = **Time**(bSecMinHour)

Remarks

bSecMinHour	A BYTE – variable, which holds the Second-value followed by Minute (Byte) and Hour (Byte)
strTime	A Time – String in Format „hh:mm:ss"
lSecOfDay	A LONG – variable which holds Second Of Day (SecOfDay)
lSysSec	A LONG – variable which holds System Second (SysSec)

Converting to a time-string:

The target string strTime must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Softclock format (3 Bytes for Second, Minute and Hour):

Three Bytes for Seconds, Minutes and Hour must follow each other in SRAM. The variable-name of the first Byte, that one for Second must be passed to the function.

See also

Date and Time Routines , SECOFDAY, SYSSEC

Example

```
Enable Interrupts
Config Clock = Soft

Dim strtime as String * 8
Dim bSec as Byte, bMin as Byte, bHour as Byte
Dim lSecOfDay as Long
Dim lSysSec as Long
```

```
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour) to
Time - String
bSec = 20: bMin = 1: bHour = 7
strTime = Time(bSec)
print "Time values: Sec="; bsec ; " Min="; bmin ; " Hour=" ; bHour ; "
converted to string " ; strTime
' Time values: Sec=20 Min=1 Hour=7 converted to string 07:01:20


' Example 2: Converting System Second to Time - String
lSysSec = 123456789
strTime = Time(lSysSec)
print "Time of Systemsecond " ; lSysSec ; " is " ; strTime
' Time of Systemsecond 123456789 is 21:33:09


' Example 3: Converting Second of Day to Time - String
lSecOfDay = 12345
strTime = Time(lSecOfDay)
print "Time of Second of Day " ; lSecOfDay ; " is " ; strTime
' Time of Second of Day 12345 is 03:25:45

' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)
lSysSec = 123456789
bSec = Time(lSysSec)
Print "System Second " ; lSysSec ; " converted to Sec="; bsec ; " Min=";
bmin ; " Hour=" ; bHour
' System Second 123456789 converted to Sec=9 Min=33 Hour=21


' Example 4: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
lSecOfDay = 12345
bSec = Time(lSecOfDay)
Print "Second of Day " ; lSecOfDay ; " converted to Sec="; bsec ; "
Min="; bmin ; " Hour=" ; bHour
' Second of Day 12345 converted to Sec=45 Min=25 Hour=3
```

TOGGLE

Action

Toggles the state of an output pin or bit variable.

Syntax

TOGGLE pin

Remarks

pin	Any port pin like PORTB.0 or bit variable. A port poin must be configured as an output pin before TOGGLE can be used.

With TOGGLE you can simply invert the output state of a port pin.

When the pin is driving a relais for example and the relais is OFF, one TOGGLE statement will turn the relais ON. Another TOGGLE will turn the relais OFF again.

See also

CONFIG PORT

ASM

NONE

Example

```
Dim Var As Byte
CONFIG PINB.0 = OUTPUT ' portB.0 is an output now
TOGGLE PORTB.0 'toggle state
WAITMS 1000cho 'wait for 1 sec
TOGGLE PORTB.0 'toggle state again
```

TRIM

Action

Returns a copy of a string with leading and trailing blanks removed

Syntax

var =**TRIM**(org)

Remarks

Var	String that receives the result.
Org	The string to remove the spaces from

See also

RTRIM , LTRIM

ASM

NONE

Example

```
Dim S As String * 6
S = " AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

UCASE

Action

Converts a string in to all upper case characters.

Syntax

Target =**Ucase**(source)

Remarks

Target	The string that is assigned with the upper case string of string target.
Source	The source string.

See also

LCASE

ASM

The following ASM routines are called from MCS.LIB : _UCASE

X must point to the target string, Z must point to the source string.

The generated ASM code : (can be different depending on the micro used)

```
;##### Z =Ucase(s)
Ldi R30,$60
Ldi R31,$00 ; load constant in register
Ldi R26,$6D
Rcall _Ucase
```

Example

```
Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

UDPREAD

Action

Reads data via UDP protocol.

Syntax

Result = **UDPread**(socket , var, bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually received from the socket. When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3).
Var	The name of the variable that will be assigned with the data from the socket.
Bytes	The number of bytes to read. Only valid for non-string variables.

When you use UDPread with a string variable, the routine will wait for CR + LF and it will return the data without the CR + LF.

For strings, the function will also not overwrite the string.

For example, your string is 10 bytes long and the line you receive is 80 bytes long, you will receive only the first 10 bytes after CR + LF is encountered.

Also, for string variables, you do not need to specify the number of bytes to read since the routine will wait for CR + LF.

For other data types you need to specify the number of bytes.

There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

The socketstat function will return a length of the number of bytes + 8 for UDP. This because UDP sends also a 8 byte header. It contains the length of the data, the IP number of the peer and the port number.

The UDPread function will fill the following variables with this header data:

Peersize, PeerAddress, PeerPort

You need to DIM these variables in your program when you use UDP.

Use the following line :

Dim Peersize As Integer , Peeraddress As Long , Peerport As Word

Make sure you maintain the shown order.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPWRITESTR, CLOSESOCKET , SOCKETLISTEN , UDPWRITE, UDPWRITESTR

Example

```
Result = Socketstat(idx , Sel_recv) ' get number of bytes waiting
If Result > 0 Then
Print "Bytes waiting : " ; Result
Temp2 = Result - 8 'the first 8 bytes are always the UDP header which
consist of the length, IP number and port address
Temp = Udpread(idx , S(1) , Result) ' read the result
For Temp = 1 To Temp2
Print S(temp) ; " " ; ' print result
Next
End If
```

UDPWRITE

Action

Write UDP data to a socket.

Syntax

Result = **UDPwrite**(IP, port, socket , var , bytes)

Result = **UDPwrite**(IP, port, socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
IP	The IP number you want to send data to. Use the format 192.168.0.5 or use a LONG variable that contains the IP number.
Port	The port number you want to send data too.
Socket	The socket number you want to send data to(0-3).
Var	A constant string like "test" or a variable. When you send a constant string, the number of bytes to send does not need to be specified.
Bytes	A word variable or numeric constant that specifies how many bytes must be send.
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.
EPROM	An indication for the compiler so it knows that you will send data from EPROM.

The UDPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.

When you want to send data from an array, you need to specify the element : var(idx) for example.

Note that UDPwrite is almost the same as TCPwrite. Since UDP is a connection-less protocol, you need to specify the IP address and the port number.

UDP only requires an opened socket.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITESTR, TCPREAD, CLOSESOCKET , SOCKETLISTEN , UDPWRITESTR , UDPREAD

Example

See UDPwriteStr

UDPWRITESTR

Action

Sends a string via UDP.

Syntax

Result = **UDPwriteStr**(IP, port, socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket. When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
IP	The IP number y ou want to send data to. Use the format 192.168.0.5 or use a LONG variable that contains the IP number.
Port	The port number you want to send data too.
Socket	The socket number you want to send data to (0-3).
Var	The name of a string variable.
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF This option was added because many protocols expect CR + LF after the string.

The UDPwriteStr function is a special variant of the UDPwrite function.

It will use UDPWrite to send the data.

See also

CONFIG TCPIP, GETSOCKET , SOCKETCONNECT, SOCKETSTAT , TCPWRITE, TCPREAD, CLOSESOCKET , SOCKETLISTEN , UDPWRITE, UDPREAD

Example

```
'-----------------------------------------------------------------------
--
' UDPTEST.BAS
' (c) 2002-2004 MCS Electronics
' start the easytcp.exe program after the chip is programmed and press
UDP button
'-----------------------------------------------------------------------
--
$regfile = "M161def.dat"
$crystal = 4000000
$baud = 19200

Const Sock_stream = $01 ' Tcp
Const Sock_dgram = $02 ' Udp
Const Sock_ipl_raw = $03 ' Ip Layer Raw Sock
Const Sock_macl_raw = $04 ' Mac Layer Raw Sock
Const Sel_control = 0 ' Confirm Socket Status
Const Sel_send = 1 ' Confirm Tx Free Buffer Size
Const Sel_recv = 2 ' Confirm Rx Data Size

'socket status
Const Sock_closed = $00 ' Status Of Connection Closed
Const Sock_arp = $01 ' Status Of Arp
Const Sock_listen = $02 ' Status Of Waiting For Tcp Connection Setup
Const Sock_synsent = $03 ' Status Of Setting Up Tcp Connection
Const Sock_synsent_ack = $04 ' Status Of Setting Up Tcp Connection
Const Sock_synrecv = $05 ' Status Of Setting Up Tcp Connection
Const Sock_established = $06 ' Status Of Tcp Connection Established
Const Sock_close_wait = $07 ' Status Of Closing Tcp Connection
Const Sock_last_ack = $08 ' Status Of Closing Tcp Connection
Const Sock_fin_wait1 = $09 ' Status Of Closing Tcp Connection
Const Sock_fin_wait2 = $0a ' Status Of Closing Tcp Connection
Const Sock_closing = $0b ' Status Of Closing Tcp Connection
Const Sock_time_wait = $0c ' Status Of Closing Tcp Connection
Const Sock_reset = $0d ' Status Of Closing Tcp Connection
Const Sock_init = $0e ' Status Of Socket Initialization
Const Sock_udp = $0f ' Status Of Udp
Const Sock_raw = $10 ' Status of IP RAW


$lib "tcpip.lib" ' specify the tcpip library
Print "Init , set IP to 192.168.0.8" ' display a message
Enable Interrupts ' before we use config tcpip , we need to enable the
interrupts
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx = $55
, Rx = $55

'Use the line below if you have a gate way
'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx =
$55 , Rx = $55

Dim Idx As Byte ' socket number
Dim Result As Word ' result
Dim S(80) As Byte
Dim Sstr As String * 20
Dim Temp As Byte , Temp2 As Byte ' temp bytes
'-----------------------------------------------------------------------
---
'When you use UDP, you need to dimension the following variables in
exactly the same order !
Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
'-----------------------------------------------------------------------
Declare Function Ipnum(ip As Long) As String ' a handy function

'like with TCP, we need to get a socket first
'note that for UDP we specify sock_dgram
```

```
Idx = Getsocket(idx , Sock_dgram , 5000 , 0) ' get socket for UDP mode,
specify port 5000
Print "Socket " ; Idx ; " " ; Idx

'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition of
the IP and PORT
Do
Temp = Inkey() ' wait for terminal input
If Temp = 27 Then ' ESC pressed
Sstr = "Hello"
Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255)
End If
Result = Socketstat(idx , Sel_recv) ' get number of bytes waiting
If Result > 0 Then
Print "Bytes waiting : " ; Result
Temp2 = Result - 8 'the first 8 bytes are always the UDP header which
consist of the length, IP number and port address
Temp = Udpread(idx , S(1) , Result) ' read the result
For Temp = 1 To Temp2
Print S(temp) ; " " ; ' print result
Next
Print
Print Peersize ; " " ; Peeraddress ; " " ; Peerport ' these are assigned
when you use UDPREAD
Print Ipnum(peeraddress) ' print IP in usual format
Result = Udpwrite(192 , 168 , 0 , 3 , Peerport , Idx , S(1) , Temp2) '
write the received data back
End If
Loop
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size

'this function can be used to display an IP number in normal format
Function Ipnum(ip As Long) As String
Local T As Byte , J As Byte
Ipnum = ""
For J = 1 To 4
T = Ip And 255
Ipnum = Ipnum + Str(t)
If J < 4 Then Ipnum = Ipnum + "."
Shift Ip , Right , 8
Next
End Function

End
```

UPPERLINE

Action

Reset LCD cursor to the upperline.

Syntax

UPPERLINE

Remarks

NONE

See also

LOWERLINE , THIRDLINE , FOURTHLINE

Example

```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

VAL

Action

Converts a string representation of a number into a number.

Syntax

var **= Val(** s **)**

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type.

See also

[STR](), [HEXVAL](), [HEX](), [BIN]()

Example

```
Dim a as byte, s As String * 10
s = "123"
a = Val(s) 'convert string
Print A ' 123
End
```

VARPTR

Action

Retrieves the memory-address of a variable.

Syntax

var =**VARPTR**(var2)

Remarks

Var	The variable that receives the address of var2.
Var2	A variable to retrieve the address from.

See also

NONE

Example

```
Dim W As Byte
Print Hex(varptr(w)) ' 0060
```

VER

Action

Returns the AVR -DOS version

Syntax

result = **Ver()**

Remarks

Result	A numeric variable that is assigned with the AVR-DOS version. The version number is a byte and the first release is version 1.

When you have a problem, MCS can ask you for the AVR -DOS version number. The VER() function can be used to return the version number then.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_AVRDOSVer	
Input		
Output	R16 loaded with value	

Example

```
Dim S As string * 10 , W As Word ,L As Long
S = "write"
Open "write.dmo" For Output As #2
Write #2 , S , W , L ' write is also supported
Close #2
Print  Ver()
```

WAIT

Action

Suspends program execution for a given time.

Syntax

WAIT seconds

Remarks

seconds	The number of seconds to wait.

No accurate timing is possible with this command.

When you use interrupts, the delay may be extended.

See also

DELAY , WAITMS

Example

```
WAIT 3  'wait for three seconds
Print "*"
```

WAITKEY

Action

Wait until a character is received in the serial buffer.

Syntax

var = **WAITKEY()**
var = **WAITKEY(#channel)**

Remarks

var	Variable that receives the ASCII value of the serial buffer. Can be a numeric variable or a string variable.
#channel	The channel used for the software UART.

See also

INKEY , ISCHARWAITING

Example

```
Dim A As Byte
A = Waitkey()  'wait for character
Print A
```

WAITMS

Action

Suspends program execution for a given time in mS.

Syntax

WAITMS mS

Remarks

Ms	The number of milliseconds to wait. (1-65535)

No accurate timing is possible with this command.
In addition, the use of interrupts can slow this routine.
When you write to an EEPROM you must wait for 10 mS after the write instruction.

See also

DELAY , WAIT , WAITUS

ASM

WaitMS will call the routine _WAITMS. R24 and R25 are loaded with the number of milliseconds to wait.

Uses and saves R30 and R31.
Depending on the used XTAL the asm code can look like :

```
_WaitMS:

_WaitMS1F:

Push R30 ; save Z

Push R31

_WaitMS_1:

Ldi R30,$E8 ;delay for 1 mS

Ldi R31,$03

_WaitMS_2:

Sbiw R30,1 ; -1

Brne _WaitMS_2 ; until 1 mS is ticked away

Sbiw R24,1

Brne _WaitMS_1 ; for number of mS

Pop R31

Pop R30

Ret
```

Example

```
WAITMS 10  'wait for 10 mS
Print "*"
```

WAITUS

Action

Suspends program execution for a given time in uS.

Syntax

WAITUS uS

Remarks

US	The number of microseconds to wait. (1-65535) This must be a constant. Not a variable!

No accurate timing is possible with this command.

In addition, the use of interrupts can slow this routine.

The minimum delay possible is determined by the used frequency.

The number of cycles that are needed to set and save registers is 17.

When the loop is set to 1, the minimum delay is 21 uS. In this case you can better use a NOP that generates 1 clock cycle delay.

At 4 MHz the minimum delay is 5 uS. So a waitus 3 will also generate 5 uS delay.

Above these values the delay will become accurate.

When you really need an accurate delay you can use a timer for this purpose.

Set the timer to a value and poll until the overflow flag is set. The disadvantage is that you can not use the timer for other tasks during this hardware delay.

The philosophy behind BASCOM is that it should not use hardware resources unless there is no other way to accomplish a task.

See also

DELAY , WAIT , WAITMS

Example

```
WAITUS 10  'wait for 10 uS
Print "*"
```

WHILE-WEND

Action

Executes a series of statements in a loop, as long as a given condition is true.

Syntax

WHILE condition

statements

WEND

Remarks

If the condition is true then any intervening statements are executed until the WEND statement is encountered.

BASCOM then returns to the WHILE statement and checks the condition.

If it is still true, the process is repeated.

If it is not true, execution resumes with the statement following the WEND statement.

So in contrast with the DO-LOOP structure, a WHILE-WEND condition is tested first so that if the condition fails, the statements in the WHILE-WE ND structure are never executed.

See also

DO-LOOP

Example

```
Dim A As Byte
While A <= 10 'if a is smaller or equal to 10
Print A 'print variable a
Incr A
Wend
```

WRITE

Action

Writes data to a sequential file

Syntax

Write #ch , data [,data1]

Remarks

Ch	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Data , data1	A variable who's content are written to the file.

When you write a variables value, you do not write the binary representatrion but the ASCII representation. When you look in a file it contains readable text.

When you use PUT, to write binary info, the files are not readable or contain unreadable characters.

Strings written are surrounded by string delimeters "". Multiple variables written are separated by a comma. Consider this example :

Dim S as String * 10 , W as Word

S="hello" : W = 100

OPEN "test.txt" For OUTPUT as #1

WRITE #1, S , W

CLOSE #1

The file content will look like this : "hello",100

Use INPU T to read the values from value.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_FileWriteQuotationMark	_FileWriteDecInt
	_FileWriteDecByte	_FileWriteDecWord
	_FileWriteDecLong	_FileWriteDecSingle
Input	Z points to variable	
Output		

Example

```
Dim S As string * 10 , W As Word ,L As Long

S = "write"
Open "write.dmo" For Output As #2
Write #2 , S , W , L ' write is also supported
Close #2

Open "write.dmo" For Input As #2
Input #2 , S , W , L ' write is also supported
Close #2
Print S ; " " ; W ; " " ; L
```

WRITEEEPROM

Action

Write a variables content to the DATA EEPROM.

Syntax

WRITEEEPROM var , address

Remarks

var	The name of the variable that must be stored
address	The address in the EEPROM where the variable must be stored. A new option is that you can provide a label name for the address. See example 2.

This statement is provided for compatibility with BASCOM-8051.

You can also use :

Dim V as Eram Byte 'store in EEPROM

Dim B As Byte 'normal variable

B = 10

V = B 'store variable in EEPROM

When you use the assignment version, the data types must be the same!

According to a datasheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset.

For security, register R23 is set to a magic value before the data is written to the EEPROM.

All interrupts are disabled while the EEPROM data is written. Interrupts are enabled automatic when the data is written.

See also

READEEPROM

ASM

NONE

Example

```
Dim B As Byte
WriteEEPROM B ,0  'store at first position
ReadEEPROM B, 0 'read byte back
```

Example 2

```
'-----------------------------------------------------------------
' EEPROM2.BAS
' This example shows how to use labels with READEEPROM
```

```
'-----------------------------------------------------------------
'first dimension a variable
Dim B As Byte
Dim Yes As String * 1

'Usage for readeeprom and writeeprom :
'readeeprom var, address

'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!

'first tell the compiler that we are using EEPROM to store the DATA
$eeprom
'specify a label
label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50

'Switch back to normal data lines in case they are used
$data

'All the code above does not generate real object code
'It only creates a file with the EEP extension

'Use the new label option
Readeeprom B , Label1
Print B 'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B 'prints 2

Readeeprom B , Label2
Print B 'prints 10
Readeeprom B
Print B 'prints 20

'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Label1
B = 101
Writeeeprom B

'read it back
Readeeprom B , Label1
Print B 'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B 'prints 2


End
```

X10DETECT

Action

Returns a byte that indicates if a X10 Power line interface is found.

Syntax

Result = **X10DETECT**( )

Remarks

Result	A variable that will be assigned with 0 if there is no Power Line Interface found. 1 will be returned if the interface is found, and the detected mains frequency is 50 Hz. 2 will be returned if the interface is found and tre detected mains frequency is 60 Hz.

When no TW-523 or other suitable interface is found, the other X10 routines will not work.

See also

CONFIG X10 , X10SEND

Example

```
'-------------------------------------------------------------------------
' X10.BAS
' (c) 2002-2004 MCS Electronics
' This example needs a TW-523 X10 interface
'-------------------------------------------------------------------------
$crystal = 8000000
$baud = 19200

'define the house code
Const House = "M" ' use code A-P

Waitms 500 ' optional delay not really needed

'dim the used variables
Dim X As Byte

'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
' ^--zero cross
' ^--- transmission pin

'detect the TW-523
X = X10detect()
Print X ' 0 means error, 1 means 50 Hz, 2 means 60 Hz
```

```
Do
Input "Send (1-32) " , X
'enter a key code from 1-31
'1-16 to address a unit
'17 all units off
'18 all lights on
'19 ON
'20 OFF
'21 DIM
'22 BRIGHT
'23 All lights off
'24 extended code
'25 hail request
'26 hail acknowledge
'27 preset dim
'28 preset dim
'29 extended data analog
'30 status on
'31 status off
'32 status request

X10send House , X ' send the code
Loop
End
```

X10SEND

Action

Sends a house and key code with the X10 protocol.

Syntax

X10SEND house , code

Remarks

House	The house code in the form of a letter A-P. You can use a constant, or you can use a variable
Code	The code or function to send. This is a number between 1-32.

The X10SEND command needs a TW-523interface.

Only ground, TX and Zero Cross, needs to be connected for transmission.

Use CONFIG X10 to specify the pins.

X10 is a popular protocol used to control equipment via the mains. A 110 Khz signal is added to the normal 50/60 Hz , 220/110 V power.

Notice that experimenting with 110V-240V can be very dangerous when you do not know exactly what you are doing !!!

In the US, X10 is very popular and wide spread. In Europe it is hard to get a TW-523 for 220/230/240 V.

I modified an 110V version so it worked for 220V. On the Internet you can find modification information. But as noticed before, MODIFY ONLY WHEN YOU UNDERSTAND WHAT YOU ARE DOING.

A bad modified device could result in a fire, and your insurance will most likely not pay. A modified device will not pass any CE, or other test.

When the TW-523 is connected to the mains and you use the X10SEND command, you will notice that the LED on the TW-523 will blink.

The following table lists all X10 codes.

Code value	Description
1-16	Used to address a unit. X10 can use a maximum of 16 units per house code.
17	All units off
18	All lights on
19	ON
20	OFF
21	DIM

22	BRIGHT
23	All lights off
24	Extendedode
25	Hailrequest
26	Hail acknowledge
27	Preset dim
28	Presetdim
29	Extended data analog
30	Status on
31	Status off
32	Status request

At www.x10.com you can find all X10 information. The intension of BASCOM is not to learn you everything about X10, but to show you how you can use it with BASCOM.

See also

CONFIG X10 , X10DETECT , X10SEND

Example

```
'-----------------------------------------------------------------------
' X10.BAS
' (c) 2002-2004 MCS Electronics
' This example needs a TW-523 X10 interface
'-----------------------------------------------------------------------
$crystal = 8000000
$baud = 19200

'define the house code
Const House = "M" ' use code A-P

Waitms 500 ' optional delay not really needed

'dim the used variables
Dim X As Byte

'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
' ^--zero cross
' ^--- transmission pin

'detect the TW-523
X = X10detect()
Print X ' 0 means error, 1 means 50 Hz, 2 means 60 Hz

Do
Input "Send (1-32) " , X
'enter a key code from 1-31
'1-16 to address a unit
'17 all units off
'18 all lights on
'19 ON
'20 OFF
'21 DIM
'22 BRIGHT
'23 All lights off
'24 extended code
'25 hail request
'26 hail acknowledge
'27 preset dim
'28 preset dim
'29 extended data analog
'30 status on
'31 status off
'32 status request

X10send House , X ' send the code
Loop
End
```

#IF ELSE ENDIF

Action

Conditional compilation directives intended for conditional compilation.

Syntax

#IF condition

#ELSE

#ENDIF

Remarks

Conditional compilation is supported by the compiler.

What is conditional compilation?

Conditional compilation will only compile parts of your code that meet the criteria of the condition.

By default all your code is compiled.

Conditional compilation needs a constant to test.

So before a condition can be set up you need to define a constant.

```
CONST test = 1
#IF TEST
Print "This will be compiled"
#ELSE
Print "And this not"
#ENDIF
```

Note that there is no THEN and that #ENDIF is not #END IF (no space)

You can nest the conditions and the use of #ELSE is optional.

There are a few internal constants that you can use. These are generated by the compiler:

```
_CHIP = 0
_RAMSIZE = 128
_ERAMSIZE = 128
_SIM = 0
_XTAL = 4000000
_BUILD = 11162
```

_CHIP is an integer that specifies the chip, in this case the 2313

_RAMSIZE is the size of the SRAM

_ERAMSIZE is the size of the EEPROM

_SIM is set to 1 when the $SIM directive is used

_XTAL contains the value of the specified crystal

_BUILD is the build number of the compiler.

The build number can be used to write support for statements that are not available in a certain version :

```
#IF _BUILD >= 11162
s = Log(1.1)
#ELSE
Print "Sorry, implemented in 1.11.6.2"
#ENDIF
```

International Resellers

See http://www.mcselec.com/reseller.htm

LCD4.LIB

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code.

When you want to have less code you need fixed pins for the LCD display.

With the statement $LIB "LCD4.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code:

Rs = PortB.0

RW = PortB.1 we dont use the R/W option of the LCD in this version so connect to ground

E = PortB.2

E2 = PortB.3 optional for lcd with 2 chips

Db4 = PortB.4 the data bits must be in a nibble to save code

Db5 = PortB.5

Db6 = PortB.6

Db7 = PortB.7

You can change the lines from the lcd4.lib file to use another port.

Just change the address used :

.EQU LCDDDR=$17 ; change to another address for DDRD ($11)

.EQU LCDPORT=$18 ; change to another address for PORTD ($12)

See the demo **lcdcustom4bit.bas** in the SAMPLES dir.

Note that you still must select the display that you use with the CONFIG LCD statement.

See also the lcd42.lib for driving displays with 2 E lines.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

GLCD

GLCD.LIB (LBX) is a library for Graphic LCD's based on the T6963C chip.

The library contains code for LOCATE, CLS, PSET, LINE, CIRCLE, SHOWPIC and SHOWPICE.

GLCDSED

GLCDSED.LIB (LBX) is a library for Graphic LCD's based on the SEDXXXX chip.

The library contains modified code for this type of display.

New special statements for this display are:

LCDAT
SETFONT
GLCDCMD
GLCDDATA

See the SED.BAS sample from the sample directory

LCD4E2

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code.

When you want to have less code you need fixed pins for the LCD display.

With the statement $LIB "LCD4E2.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code:

Rs = PortB.0

RW = PortB.1 we don't use the R/W option of the LCD in this version so connect to ground

E = PortB.2

E2 = PortB.3 the second E pin of the LCD

Db4 = PortB.4 the data bits must be in a nibble to save code

Db5 = PortB.5

Db6 = PortB.6

Db7 = PortB.7

You can change the lines from the lcd4e2.lib file to use another port.

Just change the address used :

.EQU LCDDDR=$17 ; change to another address for DDRD ($11)

.EQU LCDPORT=$18 ; change to another address for PORTD ($12)

See the demo **lcdcustom4bit2e.bas** in the SAMPLES dir.

Note that you still must select the display that you use with the CONFIG LCD statement.

See also the lcd4.lib for driving a display with 1 E line.

A display with 2 E lines actually is a display with 2 control chips. They must both be controlled. This library allows you to select the active E line from your code.

In your basic code you must first select the E line before you use a LCD statement.

The initialization of the display will handle both chips.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

MCSBYTE

The numeric<>string conversion routines are optimized when used for byte, integer,word and longs.

When do you use a conversion routine ?

-When you use STR() , VAL() or HEX().

-When you print a numeric variable

-When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size.

But when you use only conversion routines on bytes there is a overhead.

The mcsbyte.lib library is an optimized version that only support bytes.

Use it by including : $LIB "mcsbyte.lbx" in your code.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

See also the library mcsbyteint.lib

MCSBYTEINT

The numeric<>string conversion routines are optimized when used for byte, integer,word and longs.

When do you use a conversion routine ?

-When you use STR() , VAL() or HEX().

-When you print a numeric variable

-When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size.

But when you use only conversion routines on bytes there is a overhead.

The mcsbyteint.lib library is an optimized version that only support bytes, integers and words.

Use it by including : $LIB "mcsbyteint.lbx" in your code.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

See also the library mcsbyte.lib

FP_TRIG

The FP_TRIG library is written by Josef Franz Vögel.

MCS would like to thank him for his great contribution!

All trig functions are stored in fp_trig.lib library.

The fp_trig.lbx contains the compiled object code.

This sample demonstrates all the functions from the library:

```
'-----------------------------------------------------------------------
------
' TEST_FPTRIG2.BAS
' Demonstates FP trig library from Josef Franz Vögel
' The entire FP_TRIG.LIB is written by Josef Franz Vögel
'-----------------------------------------------------------------------
------

$regfile = "8515def.dat"
$lib "FP_Trig.lbx"

Dim S1 As Single , S2 As Single , S3 As Single , S4 As Single , S5 As
Single
Dim Vcos As Single , Vsin As Single , Vtan As Single , Vatan As Single
Dim Wi As Single , B1 As Byte
Dim Ms1 As Single



Const Pi = 3.14159265358979

'calculate PI
Ms1 = Atn(1) * 4


Testing_Power:
Print "Testing Power X ^ Y"
Print "X Y x^Y"
For S1 = 0.25 To 14 Step 0.25
S2 = S1 \ 2
S3 = Power(s1 , S2)
Print S1 ; " ^ " ; S2 ; " = " ; S3
Next
Print : Print : Print

Testing_EXP_log:

Print "Testing EXP and LOG"
Print "x exp(x) log([exp(x)]) Error-abs Error-rel"
Print "Error is for calculating exp and back with log together"
For S1 = -88 To 88
S2 = Exp(s1)
S3 = Log(s2)
S4 = S3 - S1
S5 = S4 \ S1
Print S1 ; " " ; S2 ; " " ; S3 ; " " ; S4 ; " " ; S5 ; " " ;
Print
Next
```

```
Print : Print : Print


Testing_Trig:
Print "Testing COS, SIN and TAN"
Print "Angle Degree Angle Radiant Cos Sin Tan"
For Wi = -48 To 48
S1 = Wi * 15
S2 = Deg2rad(s1)
Vcos = Cos(s2)
Vsin = Sin(s2)
Vtan = Tan(s2)
Print S1 ; "  " ; S2 ; "  " ; Vcos ; "  " ; Vsin ; "  " ; Vtan
Next
Print : Print : Print


Testing_ATAN:
Print "Testing Arctan"
Print "X atan in Radiant, Degree"
S1 = 1 / 1024
Do
S2 = Atn(s1)
S3 = Rad2deg(s2)
Print S1 ; "  " ; S2 ; "  " ; S3
S1 = S1 * 2
If S1 > 1000000 Then
Exit Do
End If
Loop

Print : Print : Print

Testing_Int_Fract:
Print "Testing Int und Fract of Single"
Print "Value Int Frac"
s2 = pi \ 10
For S1 = 1 To 8
S3 = Int(s2)
S4 = Frac(s2)
Print S2 ; "  " ; S3 ; "  " ; S4
S2 = S2 * 10
Next

Print : Print : Print

print "Testing degree - radiant - degree converting"
print "Degree Radiant Degree Diff-abs rel"

For S1 = 0 To 90
S2 = Deg2rad(s1)
S3 = Rad2deg(s2)
S4 = S3 - S1
S5 = S4 \ S1
Print S1 ; "  " ; S2 ; "  " ; S3 ; "  " ; S4 ; "  " ; S5
Next

Testing_Hyperbolicus:
Print : Print : Print
Print "Testing SINH, COSH and TANH"
Print "X sinh(x) cosh(x) tanh(x)"
For S1 = -20 To 20
S3 = Sinh(s1)
S2 = Cosh(s1)
S4 = Tanh(s1)
Print S1 ; "  " ; S3 ; "  " ; S2 ; "  " ; S4
Next
Print : Print : Print

TEsting_LOG10:
Print "Testing LOG10"
Print "X log10(x)"
S1 = 0.01
S2 = Log10(s1)
Print S1 ; "  " ; S2
S1 = 0.1
S2 = Log10(s1)
Print S1 ; "  " ; S2
For S1 = 1 To 100
S2 = Log10(s1)
Print S1 ; "  " ; S2
next

Print : Print : Print
Print "End of testing"

End
```

Back

LCD4BUSY

The LCD4BUSY.LIB can be used when timing is critical.

The default LCD library uses delays to wait until the LCD is ready. The lcd4busy.lib is using an additional pin (WR) to read the status flag of the LCD.

The db4-db7 pins of the LCD must be connected to the higher nibble of the port.

The other pins can be defined.

```
'-----------------------------------------------------------------------
' (c) 2004 MCS Electronics
' lcd4busy.bas shows how to use LCD with busy check
'-----------------------------------------------------------------------
'code tested on a 8515
$regfile = "8515def.dat"

'stk200 has 4 MHz
$crystal = 4000000

'define the custom library
'uses 184 hex bytes total

$lib "lcd4busy.lib"

'define the used constants
'I used portA for testing
Const _lcdport = Porta
Const _lcdddr = Ddra
Const _lcdin = Pina
Const _lcd_e = 1
Const _lcd_rw = 2
Const _lcd_rs = 3


'this is like always, define the kind of LCD
Config Lcd = 16 * 2

'and here some simple lcd code
Cls
Lcd "test"
Lowerline
Lcd "this"
End
```

SPISLAVE

SPISLAVE.LIB (LBX) is a library that can be used to create a SPI slave chip when the chip does not have a hardware SPI interface.

Although most AVR chips have an ISP interface to program the chip, the 2313 for example does not have a SPI interface.

When you want to control various micro's with the SPI protocol you can use the SPISLAVE library.

The spi-softslave.bas sample from the samples directory shows how you can use the SPISLAVE library.

Also look at the spi-slave.bas sample that is intended to be used with hardware SPI.

The sendspi.bas sample from the samples directory shows how you can use the SPI hardware interface for the master controller chip.

```
'-----------------------------------------------------------------------
-------
' SPI-SOFTSLAVE.BAS
' (c) 2004 MCS Electronics
' sample that shows how to implement a SPI SLAVE with software
'-----------------------------------------------------------------------
-------
'Some atmel chips like the 2313 do not have a SPI port.
'The BASCOM SPI routines are all master mode routines
'This example show how to create a slave using the 2313
'ISP slave code

'we use the 2313
$regfile = "2313def.dat"

'XTAL used
$crystal = 4000000

'baud rate
$baud = 19200

'define the constants used by the SPI slave
Const _softslavespi_port = Portd ' we used portD
Const _softslavespi_pin = Pind 'we use the PIND register for reading
Const _softslavespi_ddr = Ddrd ' data direction of port D

Const _softslavespi_clock = 5 'pD.5 is used for the CLOCK
Const _softslavespi_miso = 3 'pD.3 is MISO
Const _softslavespi_mosi = 4 'pd.4 is MOSI
Const _softslavespi_ss = 2 ' pd.2 is SS
'while you may choose all pins you must use the INT0 pin for the SS
'for the 2313 this is pin 2

'PD.3(7), MISO must be output
'PD.4(8), MOSI
'Pd.5(9) , Clock
'PD.2(6), SS /INT0

'define the spi slave lib
$lib "spislave.lbx"
'sepcify wich routine to use
$external _spisoftslave
```

```
'we use the int0 interrupt to detect that our slave is addressed
On Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 interrupt to trigger when a falling edge is
detected
Config Int0 = Falling
'finally we enabled interrupts
Enable Interrupts

'
Dim _ssspdr As Byte ' this is out SPI SLAVE SPDR register
Dim _ssspif As Bit ' SPI interrupt revceive bit
Dim Bsend As Byte , I As Byte , B As Byte ' some other demo variables

_ssspdr = 0 ' we send a 0 the first time the master sends data
Do
If _ssspif = 1 Then
Print "received: " ; _ssspdr
Reset _ssspif
_ssspdr = _ssspdr + 1 ' we send this the next time
End If
Loop
```

EUROTIMEDATE

The CONFIG CLOCK statement for using the asynchrony timer of the 8535, M163, M103 or M128 allows you to use a software based clock. See TIME$ and DATE$.

By default the date format is in MM/DD/YY.

By specifying:

$LIB "EURODATETIME.LBX"

The DATE$ will work in European format : DD-MM-YY

Note that the eurotimedate library should not be used anymore. It is replaced by the DATETIME library which offers more features.

DATETIME

The DatTime library was written by Josef Franz Vögel. It extends the clock routines with date and time calculation.

The following functions are available:

DayOfWeek

DayOfYear

SecOfDay

SecElapsed

SysDay

SysSec

SysSecElapsed

Time

Date

Date and time not to be confused with Date$ and Time$!

TCPIP

The TCPIP library allows you to use the W3100A internet chip from www.i2chip.com

MCS has developed a special development board that can get you started quickly with TCP/IP communication. Look at http://www.mcselec.com/easy_tcp_ip.htm for more info.

The tcpip.lib is bundled with the MCS Easy TCP/IP PCB and/or the IIM7000 module.
By default the library is not available.

The following functions are provided:

CONFIG TCPIP

GETSOCKET

SOCKETCONNECT

SOCKETSTAT

TCPWRITE

TCPWRITESTR

TCPREAD

CLOSESOCKET

SOCKETLISTEN

GETDSTIP

GETDSTPORT

BASE64DEC

UDPWRITE

UDPWRITESTR

UDPREAD

PS2MOUSE_EMULATOR

The PS2 Mouse emulator library is an optional library you can purchase.

The library allows you to emulate an AT PS/2 mouse.

The following statements become available:

CONFIG PS2EMU

PS2MOUSEXY

SENDSCAN

AT_EMULATOR

The PS2 AT Keyboard emulator library is an optional library you can purchase.

The library allows you to emulate an AT PS/2 keyboard.

The following statements become available:

CONFIG ATEMU

SENDSCANKBD

I2CSLAVE

The I2C-Slave library is intended to create I2C slave chips. This is an add-on library that is not included by default.

All BASCOM I2C routines are master I2C routines. The AVR is a fast chip and allows to implement the I2C slave protocol.

You can control the chips with the BASCOM I2C statements like I2CINIT, I2CSEND, I2CRECEIVE, I2CWBYTE, etc. Please consult the BASCOM Help file for using I2C in master mode.

Before you begin

Copy the i2cslave.lib and i2cslave.lbx files into the BASCOM-AVR\LIB directory.

The i2cslave.lib file contains the ASM source. The i2cslave.lbx file contains the compiled ASM source.

Slave address

Every I2C device must have an address so it can be addressed by the master I2C routines.

When you write to an I2C-slave chip the least significant bit (bit0) is used to specify if we want to read from the chip or that we want to write to the chip.

When you specify the slave address, do not use bit 0 in the address!

For example a PCF8574 has address &H40. To write to the chip use &H40, to read from the chip, use &H41. When emulating a PCF8574 we would specify address &H40.

Use the CONFIG statement to specify the slave address:

Config I2cslave = &B01000000 ' same as &H40

Optional use : CONFIG I2CSLAVE = address, INT= int , TIMER = tmr

Where INT is INT0, INT1 etc. and TIMER is TIMER0, TIMER1 etc.

When using other interrupts or timers, you need to change the library source. The library was written for TIMER0 and INT0.

The I2C slave routines use the TIMER0 and INT0. You can not use these interrupts yourself. It also means that the SCL and SDA pins are fixed.

Note that new AVR chips have a TWI or hardware I2C implementation. It is better to use hardware I2C, then the software I2C. The slave library is intended for AVR chips that do not have hardware I2C.

CONFIG I2CSLAVE will enable the global interrupts.

After you have configured the slave address, you can insert your code.

A do-loop would be best:

```
Do
‘ your code here
Loop
```

This is a simple never-ending loop. You can use a GOTO with a label or a While Wend loop too but ensure that the program will never end.

After your main program you need to insert two labels with a return:

When the master needs to read a byte, the following label is always called

You must put the data you want to send to the master in variable _a1 which is register R16

```
I2c_master_needs_data:
'when your code is short, you need to put in a waitms statement
'Take in mind that during this routine, a wait state is active and the master will wait
'After the return, the waitstate is ended
Config Portb = Input ' make it an input
_a1 = Pinb ' Get input from portB and assign it
Return
```

BCCARD

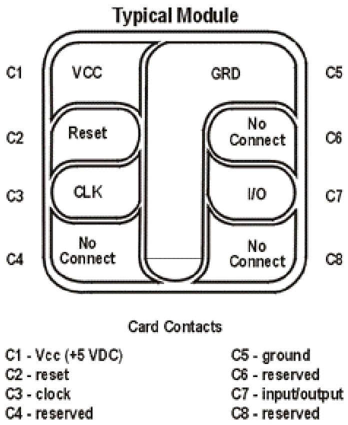
BCCARD.LIB is a library that is available separately from MCS Electronics.

With the BCCARD library you can interface with the BasicCards from www.basiccard.com

BasicCards are also available from MCS Electronics

A BasicCard is a smart card that can be programmed in BASIC.

The chip on the card looks like this :



To interface it you need a smart card connector.

In the sample provided the connections are made as following:

Smart Card PIN	Connect to
C1	+5 Volt
C2	PORTD.4 , RESET
C3	PIN 4 of 2313 , CLOCK
C5	GND
C7	PORTD.5 , I/O

The microprocessor must be clocked with a 3579545 crystal since that is the frequency the Smart Card is working on. The output clock of the microprocessor is connected to the clock pin of the Smart card.

Some global variables are needed by the library. They are dimensioned automatic by the compiler when you use the CONFIG BCCARD statement.

These variables are:

_Bc_pcb : a byte needed by the communication protocol.

Sw1 and SW2, both bytes that correspondent to the BasicCard variables SW1 and SW2

The following statements are especially for the BasicCard:

CONFIG BCCARD , to init the library

BCRESET, to reset the card

BCDEF , to define your function in the card

BCCALL , to call the function in the card

Encryption is not supported by the library.

CONFIG BCCARD

Action

Initializes the pins that are connected to the BasicCard.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

Syntax

CONFIG BCCARD = port , IO=pin, RESET=pin

Remarks

Port	The PORT of the micro that is connected to the BasicCard. This can be B or D for most micro's. (PORTB and PORTD)
IO	The pin number that is connected to the IO of the BasicCard. Must be in the range from 0-7
RESET	The pin number that is connected to the RESET of the BasicCard. Must be in the range from 0-7

The variables SW1, SW2 and _BC_PCB are automatically dimensioned by the CONFIG BCCARD statement.

See Also

BCRESET , BCDEF , BCCALL

Example

```
'----------- configure the pins we use ------------
Config Bccard = D , Io = 5 , Reset = 4
'  ^ PORTD.4
'  ^----------- PORTD.5
'  ^-------------------- PORT D
```

BCRESET

Action

Resets the BasicCard by performing an ATR.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

Syntax

BCRESET

Array(1) **= BCRESET()**

Remarks

Array(1)	When BCRESET is used as a function it returns the result of the ATR to the array named array(1). The array must be big enough to hold the result. Dim it as a byte array of 25.

An example of the returned output when used as a function:

'TS = 3B

'T0 = EF

'TB1 = 00

'TC1 = FF

'TD1 = 81 T=1 indication

'TD2 = 31 TA3,TB3 follow T=1 indicator

'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card

'TB3 = 45 BWT block waiting time

'T1-Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00

' B a s i c C a r d Z C 1 2 3

See the BasicCard manual for more information

When you do not need the result you can also use the BCRESET statement.

See Also

CONFIG BCCARD , BCDEF , BCCALL

Example (no init code shown)

```
'----and now perform an ATR as a function
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
Print I ; "  " ; Hex(buf(i))
Next
'typical returns :
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
```

```
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
' B a s i c C a r d Z C 1 2 3
```

BCDEF

Action

Defines a subroutine name and it's parameters in BASCOM so it can be called in the BasicCard.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

Syntax

BCDEF name([param1 , paramn])

Remarks

name	The name of the procedure. It may be different than the name of the procedure in the BasicCard but it is advised to use the same names.
Param1	Optional you might want to pass parameters. For each parameter you pass, you must specify the data type. Supported data types are byte, Integer, Word, Long, Single and String

For example :

BCDEF Calc(string)

Would define a name 'Calc' with one string parameter.

When you use strings, it must be the last parameter passed.

BCDEF name(byte,string)

BCDEF does not generate any code. It only informs the compiler about the data types of the passed parameters.

See Also

CONFIG BCCARD , BCCALL , BCRESET

Example (no init code shown)

```
'define the procedure in the BasicCard program
Bcdef Paramtest(byte , Word , Long )
```

BCCALL

Action

Calls a subroutine or procedure in the BasicCard.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

Syntax

BCCALL name(nad , cla, ins, p1, p2 [param1 , paramn])

Remarks

name	The name of the procedure to all in the BasicCard. It must be defined first with BCDEF. The name used with BCDEF and BCCALL do not need to be the same as the procedure in the BasicCard but it is advised to use the same names.
NAD	Node address by te. The BasicCard responds to ao all node address values. Use 0 for default.
CLA	Class byte. First byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
INS	Instruction byte. Second byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
P1	Parameter 1 of CLA–INS header.
P2	Parameter 2 of CLA-INS header

When in your BasicCard basic program you use:

'test of passing parameters

Command &hf6 &h01 ParamTest(b as byte, w as integer,l as long)

b=b+1

w=w+1

l=l+1

end command

You need to use &HF6 for CLA and 1 for INS when you call the program:

```
Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
^  NAD
^CLA
^INS
^P1
^P2
```

When you use BCCALL, the NAD, CLA, INS, P1 and P2 are sent to the BasicCard. The parameter values are also sent to the BasicCard. The BasicCard will execute the command defined with CLA and INS and will return the result in SW1 and SW2.

The parameter values altered by the BasicCard are also sent by the BasicCard.

You can not sent constant values. Only variables may be sent. This because a constant can not be changed.

See Also

CONFIG BCCARD , BCDEF , BCRESET

Example

```
'-----------------------------------------------------------------------
------
' BCCARD.BAS
' This AN shows how to use the BasicCard from Zeitcontrol
' www.basiccard.com
' *** The library source is available from MCS for 19 USD ***
'-----------------------------------------------------------------------
------
'connections:
' C1 = +5V
' C2 = PORTD.4 - RESET
' C3 = PIN 4 - CLOCK
' C5 = GND
' C7 = PORTD.5 - I/O

'  /------------------------------\
' | |
' | C1 C5 |
' | C2 C6 |
' | C3 C7 |
' | C4 C8 |
' | |
' \------------------------------/
'
'

'----------- configure the pins we use ------------
Config Bccard = D , Io = 5 , Reset = 4
' ^ PORTD.4
' ^----------- PORTD.5
' ^------------------- PORT D

'Load the sample calc.bas into the basiccard


' Now define the procedure in BASCOM
' We pass a string and also receive a string
Bcdef Calc(string)

'We need to dim the following variables
'SW1 and SW2 are returned by the BasicCard
'BC_PCB must be set to 0 before you start a session

'Our program uses a string to pass the data so DIM it
Dim S As String * 15

'Baudrate might be changed
$baud = 9600
' Crystal used must be 3579545 since it is connected to the Card too
$crystal = 3579545


'Perform an ATR
```

```
Bcreset

'Now we call the procedure in the BasicCard
'bccall funcname(nad,cla,ins,p1,p2,PRM as TYPE,PRM as TYPE)
S = "1+1+3" ' we want to calculate the result of this expression

Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
'  ^-- variable to pass that holds the expression
'  ^------ P2
'  ^---------- P1
'  ^-------------- INS
'  ^------------------- CLA
'  ^------------------------- NAD
'For info about NAD, CLA, INS, P1 and P2 see your BasicCard manual
'if an error occurs ERR is set
' The BCCALL returns also the variables SW1 and SW2
Print "Result of calc : "; S
Print "SW1 = "; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb) ' for test you can see that it toggles between 0 and
40
Print "Error : " ; Err

'You can call this or another function again in this session

S = "2+2"
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Result of calc : "; S
Print "SW1 = "; Hex(sw1)
Print "SW2 = "; Hex(sw2)
'Print Hex(_bc_pcb) ' for test you can see that it toggles between 0 and
40
Print "Error : " ; Err


'perform another ATR
Bcreset
Input "expression " , S
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Answer : " ; S


'----and now perform an ATR as a function
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
Print I ; "  " ; Hex(buf(i))
Next
'typical returns :
'TS = 3B
'T0 = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
' B a s i c C a r d Z C 1 2 3
```

```
'and another test
'define the procedure in the BasicCard program
Bcdef Paramtest(byte , Word , Long )


'dim some variables
Dim B As Byte , W As Word , L As Long

'assign the variables
B = 1 : W = &H1234 : L = &H12345678

Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
Print Hex(sw1) ; Spc(3) ; Hex(sw2)
'and see that the variables are changed by the BasicCard !
Print B ; Spc(3) ; Hex(w) ; "  " ; Hex(l)


'try the echotest command
Bcdef Echotest(byte)
Bccall Echotest(0 , &HC0 , &H14 , 1 , 0 , B)
Print B
End 'end program

' The source code of the used program in the BasicCard :
Rem BasicCard Sample Source Code
Rem ------------------------------------------------------------------
Rem Copyright (C) 1997-2001 ZeitControl GmbH
Rem You have a royalty-free right to use, modify, reproduce and
Rem distribute the Sample Application Files (and/or any modified
Rem version) in any way you find useful, provided that you agree
Rem that ZeitControl GmbH has no warranty, obligations or liability
Rem for any Sample Application Files.
Rem ------------------------------------------------------------------

#Include  CALCKEYS.BAS

Declare ApplicationID = "BasicCard Mini-Calculator"

Rem This BasicCard program contains recursive procedure calls, so the
Rem compiler will allocate all available RAM to the P-Code stack unless
Rem otherwise advised. This slows execution, because all strings have to
Rem be allocated from EEPROM. So we specify a stack size here:

#Stack 120

' Calculator Command (CLA = &H20, INS = &H01)
'
' Input: an ASCII expression involving integers, and these operators:
'
' * / % + - & ^ |
'
```

```
' (Parentheses are also allowed.)
'
' Output: the value of the expression, in ASCII.
'
' P1 = 0: all numbers are decimal
' P1 <> 0: all numbers are hex

' Constants
Const SyntaxError = &H81
Const ParenthesisMismatch = &H82
Const InvalidNumber = &H83
Const BadOperator = &H84

' Forward references
Declare Function EvaluateExpression (S$, Precedence) As Long
Declare Function EvaluateTerm (S$) As Long
Declare Sub Error (Code@)


'test for passing a string
Command &H20 &H01 Calculator (S$)

Private X As Long
S$ = Trim$ (S$)
X = EvaluateExpression (S$, 0)
If Len (Trim$ (S$)) <> 0 Then Call Error (SyntaxError)
If P1 = 0 Then S$ = Str$ (X) : Else S$ = Hex$ (X)

End Command


'test of passing parameters
Command &hf6 &h01 ParamTest( b as byte, w as integer,l as long)
b=b+1
w=w+1
l=l+1
end command



Function EvaluateExpression (S$, Precedence) As Long

EvaluateExpression = EvaluateTerm (S$)

Do
S$ = LTrim$ (S$)
If Len (S$) = 0 Then Exit Function

Select Case S$(1)

Case "*"
If Precedence > 5 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression * _
EvaluateExpression (S$, 6)
Case "/"
If Precedence > 5 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression / _
EvaluateExpression (S$, 6)
Case "%"
If Precedence > 5 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression Mod _
EvaluateExpression (S$, 6)
Case "+"
If Precedence > 4 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression + _
EvaluateExpression (S$, 5)
Case "-"
If Precedence > 4 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression - _
EvaluateExpression (S$, 5)
Case "&"
If Precedence > 3 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression And _
EvaluateExpression (S$, 4)
Case "^"
If Precedence > 2 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression Xor _
EvaluateExpression (S$, 3)
Case "|"
If Precedence > 1 Then Exit Function
S$ = Mid$ (S$, 2)
EvaluateExpression = EvaluateExpression Or _
EvaluateExpression (S$, 2)
Case Else
Exit Function
End Select

Loop

End Function

Function EvaluateTerm (S$) As Long

Do ' Ignore unary plus
S$ = LTrim$ (S$)
If Len (S$) = 0 Then Call Error (SyntaxError)
If S$(1) <> "+" Then Exit Do
S$ = Mid$ (S$, 2)
Loop

If S$(1) = "(" Then ' Expression in parentheses
S$ = Mid$ (S$, 2)
EvaluateTerm = EvaluateExpression (S$, 0)
S$ = LTrim$ (S$)
If S$(1) <> ")" Then Call Error  (ParenthesisMismatch)
```

```
S$ = Mid$ (S$, 2)
Exit Function

ElseIf S$(1) = "-" Then ' Unary minus
S$ = Mid$ (S$, 2)
EvaluateTerm = -EvaluateTerm (S$)
Exit Function

Else ' Must be a number
If P1 = 0 Then ' If decimal
EvaluateTerm = Val& (S$, L@)
Else
EvaluateTerm = ValH (S$, L@)
End If
If L@ = 0 Then Call Error (InvalidNumber)
S$ = Mid$ (S$, L@ + 1)
End If

End Function

Sub Error (Code@)
SW1 = &H64
SW2 = Code@
Exit
End Sub
```

Compact FlashCard Driver

The compact flash card driver library is written by Josef Franz Vögel. He can be contacted via the BASCOM user list.

Note that Josef has put a lot of effort in writing and especially testing the routines. Josef nor MCS Electronics can be held responsible for any damage or data loss of your CF-cards.

Compact flash cards are very small cards that are compatible with IDE drives. They work at 3.3V or 5V and have a huge storage capacity.

The FlashCard Driver provides the functions to access a Compact Flash Card.

At the moment there are six functions:

DriveCheck, DriveReset , DriveInit , DriveGetIdentity , DriveWriteSector , DriveReadSector

The Driver can be used to access the Card directly and to read and write each sector of the card or the driver can be used in combination with a file-system with basic drive access functions.

Because the file system is separated from the driver you can write your own driver.

This way you could use the file system with a serial eprom for example.

For a filesystem at least the functions for reading (DriveReadSector / _DriveReadSector) and writing (DriveWriteSector / _DriveWriteSector) must be provided. The preceeding underslash _ is the label of the according asm-routine. The other functions can, if possible implemented as a NOP – Function, which only returns a No-Error (0) or a Not Supported (224) Code, depending, what makes more sense.

For writing your own Driver to the AVR-DOS FileSystem, check the ASM-part of the functions-description.

ErrorCodes:

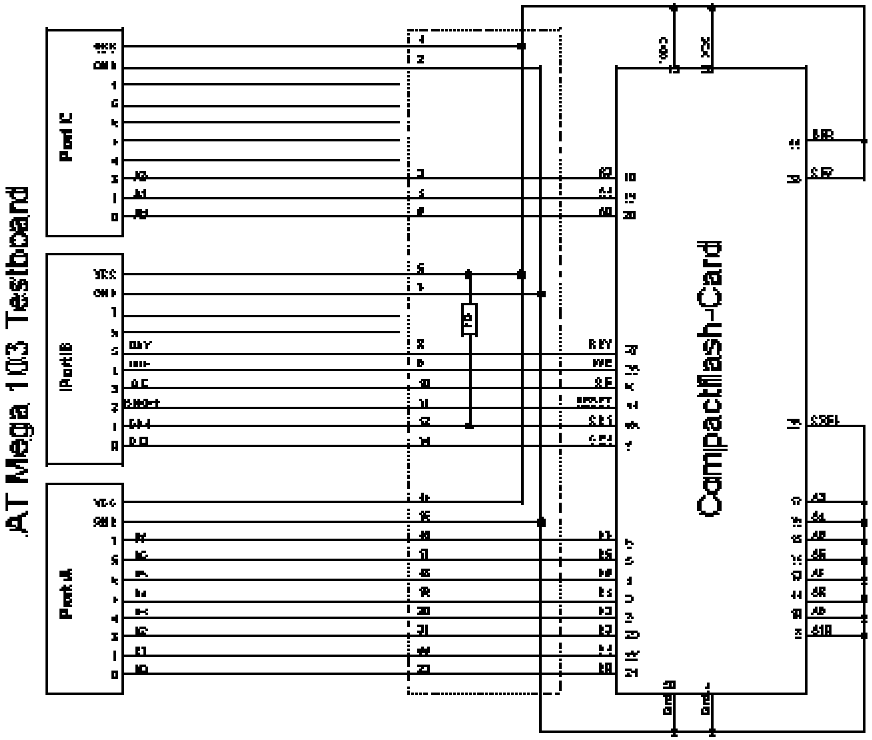
Code	Compiler – Alias	Remark
0	CpErrDriveNoError	No Error
224	cpErrDriveFunctionNotSupported	This driver does not supports this function
225	cpErrDriveNotPresent	No Drive is attached
226	cpErrDriveTimeOut	During Reading or writing a time out occured
227	cpErrDriveWriteError	Error during writing
228	cpErrDriveReadError	Error during reading

At http://www.mcselec.com/an_123.htm you can find the application.

More info about Compact Flash you can find at :

http://www.sandisk.com/download/Product%20Manuals/cf_r7.pdf

A typical connection to the micro is shown below.



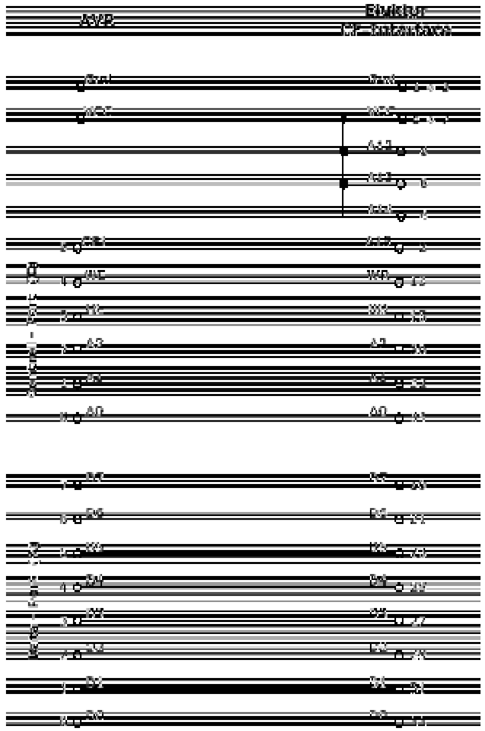
Elektor CF-Interface

The popular Electronics magazine Elektor, published an article about a CF-card interface. This interface was connected to an 89S8252. This interface can be used and will use little pins of the micro.

Note that because of the FAT buffer requirement, it is not possible to use a 8051 micro.,

At this moment, only the Mega128 and the Mega103 AVR micro's are good chips to use with AVR-DOS.

You can use external memory with other chips like the Mega162.



Changes of the hardware pins is possible in the file Config_FlashCardDrive_EL_PIN.bas.

The default library is FlashCardDrive.lib but this interface uses the library FlashCardDrive_EL_PIN.lib.

XRAM CF-Interface for simulation

The XRAM CF-Card interface is created for the purpose of testing the File System routines without hardware.

You can use an external RAM chip (XRAM) for the CF-interface but of course it is not practical in a real world application unless you backup the power with a battery.

For tests with the simulator it is ideal.

Just specify the Config_XRAMDrive.bas file and select a micro that can address external memory such as the M128. Then specify that the system is equipped with 64KB of external RAM.

You can now simulate the flashdisk.bas sample program !

In order to simulate Flashdisk.bas, set the constant XRAMDRIVE to 1. Then select 64KB of esternal RAM and compile.

New CF-Card Drivers

New CF-Card drivers can be made relatively simple.

Have a look at the supplied drivers.

There are always a few files needed :

?? A config file in the format : CONFIG_XXX.bas

?? FlashCardDrive_XXX.LIB

?? FlashCardDrive_XXX.lbx is derived from the LIB file

XXX stands for the name of your driver.

AVR-DOS File System

The AVR-DOS file system is written by Josef Franz Vögel. He can be contacted via the BASCOM user list. Note that it is not permitted to use the AVR-DOS file system for commercial applications without the purchase of a license. A license comes with the ASM source.

Note that Josef has put a lot of effort in writing and especially testing the routines. Josef nor MCS Electronics can be held responsible for any damage or data loss of your CF-cards.

The File-System works with Compact – Flash Cards (see AN 123 Accessing a Compact Flash Card from BASCOM and Compact Flash) and is written for the needs for embedded systems for logging data. There are further functions for binary read and write.

The intention in developing the DOS – filesystem was to keep close to the equivalent QB/VB functions.

The Filesystem works with:

- FAT16, this means you need to use >= 32MB CF cards
- Short file name (8.3)
  (Files with a long file name can be accessed by their short file name alias)
- Files in Root Directory. Subdirs are allowed but the files from the subdir cannot be accessed with the AVR-DOS routines. The root dir can store 512 files. Take in mind that when you use long file names, less filenames can be stored.

Requirements:

- Hardware: see AN 123 on http://www.mcselec.com/an_123.htm
- Software: appr. 2K-Word Code-Space (4000 Bytes)
- SRAM: 561 Bytes for Filesystem Info and DIR-Handle buffer
  517 Bytes if FAT is handled in own buffer (for higher speed), otherwise it is handled with the DIR Buffer
- 534 Bytes for each Filehandle
- This means that a Mega103 or Mega128 is the perfect chip. Other chips have too little internal memory. You could use XRAM memory too with a Mega8515 for example.

File System Configuration in CONFIG_AVR-DOS.BAS

cFileHandles:	***Count of Filehandles: for each file opened at same time, a filehandle buffer of 534 Bytes is needed***
cSepFATHandle:	For higher speed in handling file operations the FAT info can be stored in a own buffer, which needs additional 517 Bytes. Assign Constant cSepFATHandle with 1, if wanted, otherwise with 0.

Memory Usage of DOS – File System:

1. General File System information

Variable Name	Type	Usage
gbDOSError	Byte	holds DOS Error of last file handling routine
gbFileSystem	Byte	File System Code from Master Boot Record
glFATFirstSector	Long	Number of first Sector of FAT Area on the Card
gbNumberOfFATs	Byte	Count of FAT copies
gwSectorsPerFat	Word	Count of Sectors per FAT
glRootFirstSector	Long	Number of first Sector of Root Area on the Card
gwRootEntries	Word	Count of Root Entries
glDataFirstSector	Long	Number of first Sector of Data Area on the Card
gbSectorsPerCluster	Byte	Count of Sectors per Cluster
gwMaxClusterNumber	Word	Highest usable Cluster number
gwLastSearchedCluster	Word	Last cluster number found as free
gwFreeDirEntry	Word	Last directory entry number found as free
glFS_Temp1	Long	temorary Long variable for file system
gsTempFileName	String * 11	temporary String for converting file names

2. Directory

Variable Name	Type	Usage
gwDirRootEntry	Word	number of last handled root entry
glDirSectorNumber	Long	Number of current loaded Sector
gbDirBufferStatus	Byte	Buffer Status
gbDirBuffer	Byte (512)	Buffer for directory Sector

3. FAT

Variable Name	Type	Usage
glFATSectorNumber	Long	Number of current loaded FAT sector
gbFATBufferStatus	Byte	Buffer status
gbFATBuffer	Byte(512)	buffer for FAT sector

4. File handling

Each file handle has a block of 534 Bytes in the variable abFileHandle which is a byte-array of size (534 * cFileHandles)

Variable Name	Type	Usage
FileNumber	Byte	File number for identification of the file in I/O operations to the opened file
FileMode	Byte	File open mode
FileRootEntry	Word	Number of root entry
FileFirstCluster	Word	First cluster
FATCluster	Word	cluster of current loaded sector
FileSize	Long	file size in bytes
FilePosition	Long	file pointer (next read/write) 0-based
FileSectorNumber	Long	number of current loaded sector
FileBufferStatus	Byte	buffer Status
FileBuffer	Byte(512)	buffer for the file sector
SectorTerminator	Byte	additional 00 Byte (string terminator) for direct reading ASCII files from the buffer

ErrorCodes:

Code	Compiler – Alias	Remark

0	cpNoError	No Error
1	cpEndOfFile	Attempt behind End of File
17	cpNoMBR	Sector 0 on Card is not a Master Boot Record
18	cpNoPBR	No Partition Sector
19	cpFileSystemNotSupported	Only FAT16 File system is supported
20	cpSectorSizeNotSupported	Only sector size of 512 Bytes is supported
21	cpSectorsPerClusterNotSupported	Only 1, 2, 4, 8, 16, 32, 64 Sectors per Cluster is supported. This are values of normal formatted partitions. Exotic sizes, which are not power of 2 are not supported
33	cpNoNextCluster	Error in file cluster chain
34	cpNoFreeCluster	No free cluster to allocate (Disk full)
35	cpClusterError	Error in f ile cluster chain
49	cpNoFreeDirEntry	Directoryfull
50	cpFileExist	
65	cpNoFreeFileNumber	No free file number available, only theoretical error, if 255 file handles in use
66	cpFileNotFound	File not found
67	cpFileNumberNotFound	No file handle with such file number
68	cpFileOpenNoHandle	All file handles occupied
69	cpFileOpenHandleInUse	File handle number in use, can't create a new file handle with same file number
70	cpFileOpenShareConflict	Tried to open a file in read and write modus in two file handles
71	cpFileInUse	Can't delete file, which is in use
72	cpFileReadOnly	Can't open a read only file for writing
73	cpFileNoWildCardAllowed	No wildcard allowed in this function
97	cpFilePositionError	
98	cpFileAccessError	function not allowed in this file open mode
99	cpInvalidFilePosition	new file position pointe is invalid (minus or 0)
100	cpFileSizeToGreat	File size to great for function BLoad

Buffer Status: Bit definitions of Buffer Status Byte (Directory, FAT and File)

Bit	DIR	FAT	File	Compiler Alias	Remark
0 (LSB)			•	dBOF	Bottom of File (not yet supported)
1			•	dEOF	End of File
2			•	dEOFinSector	End of File in this sector (last sector)
3	•	•	•	dWritePending	Something was written to sector, it must be saved to Card, before loading next sector
4		•		dFATSector	This is an FAT Sector, at writing to Card, Number of FAT copies must be checked and copy updated if necessary
5			•	dFileEmpty	File is empty, no sector (Cluster) is allocated in FAT to this file

Validity of the file I/O operations regarding the opening modes

	Open mode			
Action	**Input**	**Output**	**Append**	**Binary**
Attr	•	•	•	•
Close	•	•	•	•
Put				•
Get				•
LOF	•	•	•	•
LOC	•	•	•	•
EOF	•	1)	1)	•
SEEK	•	•	•	•
SEEK -Set				•
Line Input	•			•
Print		•	•	•
Input	•			•
Write		•	•	•

1) Position pointer is always at End of File

Supported statements and functions:

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT ,FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT , FILELEN