BASCOM-8051

LANGUAGE REFERENCE

1WRESET,1WREAD,1WWRITE

Action

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

Syntax

1WRESET
1WWRITE var1
var2 **= 1WREAD()**

Remarks

1WRESET	Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred.
1WWRITE var1	Sends the value of var1 to the bus.
var2 = 1WREAD()	Reads a byte from the bus and places it into var2.

var1 : Byte, Integer, Word, Long, Constant.
var2 : Byte, Integer, Word, Long.

Example

```
'-----------------------------------------------
'                    1WIRE.BAS
' Demonstrates 1wreset, 1wwrite and 1wread()
' pullup of 4K7 required to VCC from P.1
' DS2401 serial button connected to P1.1
'-----------------------------------------------
Config 1wire = P1.1               'use this pin
Dim Ar(8) As Byte , A As Byte , I As Byte

1wreset                           'reset the bus
Print Err                         'print error 1 if error
1wwrite &H33                      'read ROM command
For I = 1 To 8
  Ar(I) = 1wread()                'read byte
Next
For I = 1 To 8
  Printhex Ar(I);                 'print output
Next
Print                             'linefeed
End
```

$ASM - $END ASM

Action
Start of inline assembly code block.

Syntax
$ASM

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

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

$INCLUDE

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

Syntax
$INCLUDE file

Remarks

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

Example
```
'-----------------------------------------------------------
'                   (c) 1997,1998 MCS Electronics
'-----------------------------------------------------------
'  file: INCLUDE.BAS
'  demo: $INCLUDE
'-----------------------------------------------------------
Print "INCLUDE.BAS"
$include c:\bascom\123.bas             'include file that prints Hello
Print "Back in INCLUDE.BAS"
End
```

$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.

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

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

See also
$CRYSTAL

Example
```
$BAUD = 2400
$CRYSTAL = 14000000   ' 14 MHz crystal
PRINT "Hello"
END
```

$CRYSTAL

Action
Instruct the compiler to override the crystal frequency options setting.

Syntax
$CRYSTAL = var

Remarks

var	Frequency of the crystal.

var : Constant.

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

See also
$BAUD

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

$IRAMSTART

Action
Compiler directive to specify starting internal memory location.

Syntax
$IRAMSTART = constant

Remarks

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

See also
$NOINIT $RAMSTART

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

$DEFAULT XRAM

Action
Compiler directive to place each dimensioned variable as XRAM.

Syntax
$DEFAULT XRAM

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

Example
```
$DEFAULT XRAM
Dim X As Integer 'will go to XRAM
Dim Z As IRAM Integer 'will be stored in IRAM
```

$LARGE

Action
Instructs the compiler that LCALL statements must be used.

Syntax
$LARGE

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

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

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

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

Example
```
$LCD = &HA000 'writing to this address will make the E-line of the LCD high.
LCD "Hello world"
```

$NOBREAK

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

Syntax
$NOBREAK

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

See also
BREAK

Example
```
$NOBREAK
BREAK          ' this isn't compiled into code so the simulator will not pause
End
```

$NOINIT

Action
Instruct the compiler that no initialisation must be performed.

Syntax
$NOINIT

Remarks

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

See also
$NOSP

Example
```
$NONIT
'your program goes here
End
```

$NOSP

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

Syntax
$NOSP

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

See also
$NOINIT

Example
```
$NOSP
$NOINIT
End
```

$OBJ

Action
Includes Intel objectcode.

Syntax
$OBJ obj

Remarks
obj is the object code to include.

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

$RAMSTART

Action
Specifies the location of the external RAM memory.

Syntax
$RAMSTART = [&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.

See also
$RAMSIZE

Example
```
$ROMSTART = &H4000
$RAMSTART = 0
$RAMSIZE = &H1000
```

$RAMSIZE

Action
Specifies the size of the external RAM memory.

Syntax
$RAMSIZE = [&H] size

Remarks

size	Size of external RAM memory chip.

size : Constant.

See also
$RAMSTART

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

$ROMSTART

Action
Specifies the location of the ROM memory.

Syntax
$ROMSTART = [&H] *address*

Remarks

address	The (hex)-address where the code must start. Default is 0. This value will be used when $ROMSTART is not specified. You can use this option when you want to test the code in RAM. The code must be uploaded and placed into the specified address and can be called from a monitor program. The monitor program must relocate the interrupts to the correct address! When $ROMSTART = &H4000 is specified the monitor program must perform a LJMP instruction. For address 3 this must be &H4003. Otherwise, interrupts can not be handled correctly. That is up to the monitor program.

See also
$RAMSTART

Example
```
$ROMSTART = &H4000    'ROM enabled at 4000 hex
```

$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 ACC.

With the redirection of the INPUT command, you can use your own routines.
This way you can use other devices as input devices.
Note that the INPUT statement is terminated when a RETURN code (13) is received.

See also
$SERIALOUTPUT

Example
```
$SERIALINPUT = Myinput
 'here goes your program
END

!myinput:
 ;perform the needed actions here
  mov a, sbuf  ;serial input buffer to acc
ret
```

$SERIALINPUT2LCD

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

Syntax
$SERIALINPUT2LCD

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

See also
$SERIALINPUT , $SERIALOUTPUT

Example
```
$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 (SBUF). The character is placed into ACC.

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

Example

```
$SERIALOUTPUT = MyOutput
 'here goes your program
END

!myoutput:
 ;perform the needed actions here
  mov sbuf, a  ;serial output buffer  (default)
ret
```

$SIM

Action
Generates code without waiting loops for the simulator.

Syntax
$SIM

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

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

See also
-

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

ABS()

Action
Returns the absolute value of a numeric variable.

Syntax
var **= ABS(**var2**)**

Remarks

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

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

The absolute value of a number is always positive.

See also
-

Difference with QB
You can not use numeric constants since the absolute value is obvious for numeric constants.
Does also not work with Singles.

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

Output
```
1000
```

ALIAS

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

Syntax
newvar **ALIAS** oldvar

Remarks

oldvar	Name of the variable such as P1.1
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
```
direction ALIAS P1.1  'now you can refer to P1.1 with the variable direction
SET direction         'has the same effect as SET P1.1
END
```

ASC()

Action
Convert a string into its ASCII value.

Syntax
var = ASC(string**)**

Remarks

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

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

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

See also
CHR()

Example
```
Dim a as byte, s as String * 10
s = ABC
a = Asc(s)
Print a
End
```

Output
```
65
```

BCD()

Action
Converts a variable into its BCD value.

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

Remarks

var	Variable to convert.

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

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

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

See also
MAKEBCD, MAKEDEC

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

BITWAIT

Action
Wait until a bit is set or reset.

Syntax
BITWAIT x **SET/RESET**

Remarks

X	Bit variable or internal register like P1.x , where x ranges form 0-7.

When using bit variables be sure that they are set/reset by software.
When you use internal registers that can be set/reset by hardware such as P1.0 this doesn't apply.

See also
-

Example
```
Dim a as bit
BITWAIT a , SET              'wait until bit a is set
BITWAIT P1.7, RESET   'wait until bit 7 of Port 1 is 0.
End
```

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

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

BREAK

Action
Generates a reserved opcode to pause the simulator.

Syntax
BREAK

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

The reserved opcode used is A5.

See also
$NOBREAK

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

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

With the CALL statement, you can call a procedure or subroutine.
As much as, 10 parameters can be passed but you can also call a subroutine without parameters.
For example: **Call Test2**

The call statement enables you to implement your own statements.

You don't have to use the CALL statement:
Test2 will also call subroutine test2

When you don't supply the CALL statement, you must leave out the parenthesis.
So Call Routine(x,y,z) must be written as Routine x,y,x

See also
DECLARE, SUB

Example
```
Dim a as byte, b as byte
Declare Sub Test(b1 as byte)
a = 65
Call test (a)         'call test with parameter A
test a                'alternative call
End

SUB Test(b1 as byte) 'use the same variable as the declared one
  LCD b       'put it on the LCD
  Lowerline
  LCD BCD(b1)
End SUB
```

CHR()

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

Syntax
PRINT CHR(var**)**
s = CHR(var**)**

Remarks

var	Byte, Integer/Word 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.

See also
ASC()

Example
```
Dim a as byte
a = 65
LCD a
Lowerline
LCDHEX a
LCD Chr(a)
End
```

CLS

Action
Clear the LCD display and set the cursor home.

Syntax
CLS

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

See also
$LCD , LCD

Example
```
Cls
LCD " Hello"
End
```

CONST

Action
Declares a symbolic constant.

Syntax
DIM symbol AS CONST value

Remarks

Symbol	The name of the symbol.
Value	The value to assign to the symbol.

Assigned constants consume no program memory.
The compiler will replace all occurrences of the symbol with the assigned value.

See also
DIM

Example
```
'-----------------------------------------------------
'              (c) 1997,1998 MCS Electronics
'              CONST.BAS
'-----------------------------------------------------
Dim A As Const 5                        'declare a as a constant
Dim B1 As Const &B1001
Waitms A                                'wait for 5 milliseconds
Print A
Print B1
End
```

CONFIG

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

CONFIG TIMER0, TIMER1
CONFIG TIMER2 (for 8052 compatible chips)
CONFIG LCD
CONFIG LCDBUS
CONFIG LCDPIN
CONFIG BAUD
CONFIG 1WIRE
CONFIG SDA
CONFIG SCL
CONFIG DEBOUNCE
CONFIG WATCHDOG
CONFIG SPI

CONFIG TIMER0, TIMER1

Action
Configure TIMER0 or TIMER1.

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

Remarks

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

So CONFIG TIMER0 = COUNTER, GATE = INTERNAL, MODE=2 will configure TIMER0 as a COUNTER with not external gatecontrol , in mode 2 (auto reload)

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

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

Example
```
CONFIG TIMER0=COUNTER, MODE=1, GATE=INTERNAL
COUNTER0 = 0          'reset counter 0
START COUNTER0        'enable the counter to run
```

```
DELAY                          'wait a while
PRINT COUNTER0         'print it
END
```

CONFIG LCD

Action
Configure the LCD display.

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 Default 16 * 2 is assumed.

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 databus.

Syntax
CONFIG LCDBUS = constant

Remarks

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

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

See also
CONFIG LCD

Example
```
$LCD = &H8000          'address of enable signal
Config LCDBUS = 4             '4 bit mode
```

CONFIG BAUD

Action

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

Syntax

CONFIG BAUD = baudrate

Remarks

Baudrate	Baudrate to use : 4800 or 9600

Example

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

CONFIG 1WIRE

Action
Configure the pin to use for 1WIRE statements.

Syntax
CONFIG 1WIRE = pin

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

See also
1WRESET , 1WREAD , 1WWRITE

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

CONFIG SDA

Action
Overrides the SDA pin assignment from the Option Settings.

Syntax
CONFIG SDA = pin

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

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

See also
CONFIG SCL

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

CONFIG SCL

Action
Overrides the SCL pin assignment from the Option Settings.

Syntax
CONFIG SCL = pin

Remarks

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

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SCL pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options.

See also
CONFIG SDA

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

CONFIG DEBOUNCE

Action
Configures the delaytime 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.
Note that the delay time is based on a 12 MHz clock frequency.

See also
DEBOUNCE

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

CONFIG SPI

Action
Configures the SPI related statements.

Syntax
CONFIG SPI = SOFT, DIN = PIN, DOUT = PIN , CS = PIN, CLK = PIN

Remarks

DIN	Data input. Pin is the pin number to use such as p1.0
DOUT	Data output. Pin is the pin number to use such as p1.1
CS	Chip select. Pin is the pin number to use such as p1.2
CLK	Clock. Pin is the pin number to use such as p1.3

See also
SPIIN SPIOUT

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

CONFIG LCDPIN

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

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

Remarks
P1.1 etc. are just an example in the syntax.

See also
CONFIG LCD

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

CONFIG WATCHDOG

Action
Configures the watchdog timer from the AT89C8252

Syntax
CONFIG WATCHDOG = time

Remarks

time	The interval constant in mS the watchdogtimer will count to. 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.

See also
START WATCHDOG , STOP WATCHDOG , RESET WATCHDOG

Example
```
'-----------------------------------------------------
'                (c) 1998 MCS Electronics
' WATCHD.BAS demonstrates the AT89S8252 watchdog timer
' select 89s8252.dat !!!
'-----------------------------------------------------
Config Watchdog = 2048                  'reset after 2048 mSec
Start Watchdog                          'start the watchdog timer
Dim I As Word
For I = 1 To 10000
  Print I                               'print value
  '  Reset Watchdog
  'you will notice that the for next doesnt finish because of the reset
  'when you unmark the RESET WATCHDOG statement it will finish because the
  'wd-timer is reset before it reaches 2048 msec
Next
End
```

COUNTERx

Action

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

Syntax

COUNTERX = var or
var = **COUNTERX**

Remarks

var	A byte, Integer/Word variable or constant that is assigned to the counter.
counterX	COUNTER0 , COUNTER1 or COUNTER2.

Use counterX = 0 to reset the counter.
The counter can count from 0 to 255 in mode 2 (8-bit auto reload).
And to 65535 in mode 1(16-bit).

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

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

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

After you have accessed the COUNTERx variable, you must START the timer with the statement START COUNTERx, because accessing the TIMER/COUNTER via the COUNTERx variable will STOP the TIMER.

Example

```
'-----------------------------------------------------------
'                     (c) 1997,1998 MCS Electronics
'-----------------------------------------------------------
'  file: COUNTER.BAS
'  demo: COUNTER
'-----------------------------------------------------------
'     Connect the timer input P3.4 to a frequency generator
'     *TIMER/COUNTER 1 is used for RS-232 baudrate generator
'-----------------------------------------------------------
Dim A As Byte , C As Integer
Config Timer0 = Counter , Gate = Internal , Mode = 1
'Timer0 = counter : timer0 operates as a counter
'Gate = Internal  : no external gate control
'Mode = 1         : 16-bit counter


Counter0 = 0                              'clear counter
Start Counter0                            'enable the counter to count
```

```
Do                                      'set up a loop
  A = Inkey                             'check for input
  C = Counter0                          'get counter value
  Start Counter0                        'Restart the timer
  Print C                               'print it
Loop Until A = 27                       'until escape is pressed

End
```

For the next example the ASM code is shown:

```
COUNTER0 = 1000
```

Generated code :

```
Clr TCON.4
Mov tl0,#232
Mov th0,#3
```

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.

See also
PEEK , POKE , INP , OUT

Example
```
'-----------------------------------------------------
'              (c) 1998 MCS Electronics
'                   PEEK.BAS
' demonstrates PEEk, POKE, CPEEK, INP and OUT
'
'-----------------------------------------------------
Dim I As Integer , B1 As Byte

'dump internal memory
For I = 0 To 127                          'for a 8052 225 could be used
'  Break
  B1 = Peek(i)                            'get byte from internal memory
  Printhex B1 ; "  ";
  'Poke I , 1                              'write a value into memory
Next
Print                                     'new line
'be careful when writing into internal memory !!
```

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

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 values to be read by subsequent READ statements.

Syntax
DATA var [, varn]

Remarks

var	Numeric or string constant.

Difference with QB
Integer and Word constants must end with the **%** -sign.
Long constants must end with the **&**-sign.
Single constants must end with the **!**-sign.

See also
READ , RESTORE

Example
```
DIM a AS BYTE, I AS BYTE, L AS Long, S As XRAM  STRING * 15
RESTORE DTA          'point to data
FOR a = 1 TO 3
  READ a : PRINT a   'read data and print it
NEXT
RESTORE DTA2         'point to data
READ I : PRINT I
READ I : PRINT I
RESTORE DTA3
READ L : PRINT L
RESTORE DTA4
READ S : PRINT S
END

DTA1:
DATA 5, 10, 100

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

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

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

DEBOUNCE

Action
Debounce a port pin connected to a switch.

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

Remarks

Px.y	A port pin like P1.0 , to examine.
state	0 for jumping when Px.y is low , 1 for jumping when Px.y is high
label	The label to GOTO when the specified state is detected
SUB	The label to GOSUB when the specified state is detected

When you specify the optional parameter SUB, a GOSUB to label is performed instead of a GOTO.
The DEBOUNCE statements wait for a port pin to get high(1) or low(0).
When it does it waits 25 mS and checks again (eliminating bounce of a switch)
When the condition is still true and there was no branch before, it branches to the label.
When DEBOUNCE is executed again, the state of the switch must have gone back in the original position before it can perform another branch.
Each DEBOUNCE statement which use a different port uses 1 BIT of the internal memory to hold its state.

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

See also
CONFIG DEBOUNCE

Example
```
'-----------------------------------------------------
'                  DEBOUN.BAS
'           demonstrates DEBOUNCE
'-----------------------------------------------------
CONFIG DEBOUNCE = 30  'when the config statement is not used a default of 25mS
will be used
Do
  'Debounce P1.1 , 1 , Pr 'try this for branching when high(1)
  Debounce P1.0 , 0 , Pr,SUB
  '                    ^----- label to branch to
  '               ^---------- branch when P1.0 goes low(0)
  '         ^---------------- examine P1.0

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

```
Pr:
  Print "P1.0 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 faster then var = var - 1.

See also
INCR

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

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

DECLARE SUB

Action
Declares a subroutine.

Syntax
DECLARE SUB TEST[(var as type)]

Remarks

test	Name of the procedure.
Var	Name of the variable(s). Maximum 10 allowed.
Type	Type of the variable(s). Bit, Byte,Word/Integer, Long or String.

You must declare each sub before writing the sub procedure.

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
```

Defint, DefBit, DefByte, DefWord

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

Syntax
DEFBIT b
DEFBYTE c
DEFINT I
DEFWORD x

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

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

DEFLCDCHAR

Action
Define a custom LCD character.

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

Remarks

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

char : Byte, Integer, Word, Long, Constant.
r1-r8 : Constant.

You can use the LCD designer to build the characters.

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

The special characters can be printed with the Chr() function.

See also
Edit LCD designer

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

DELAY

Action
Delay program execution for a short time.

Syntax
DELAY

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

See also
WAIT , WAITMS

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

DIM

Action
Dimension a variable.

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

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 in external memory
IRAM	Specify IRAM to store variable in internal memory (default)

A string variable needs an additional length parameter:
Dim s As XRAM **String * 10**
In this case, the string can have a length of 10 characters.

Note that BITS can only be stored in internal memory.

Difference with QB
In QB you don't need to dimension each variable before you use it. In BASCOM you must dimension each variable before you use it.
In addition, the XRAM/IRAM options are not available in QB.

See Also
CONST , ERASE

Example
```
'-----------------------------------------------------------
'                     (c) 1997-1999 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 A As String * 10                      'string with length of 10 characters


Dim ar(10) As Byte                        'dimension array
'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
End
```

DISABLE

Action
Disable specified interrupt.

Syntax
DISABLE interrupt

Remarks
Interrupt : **INT0**, **INT1**, **SERIAL**, **TIMER0**, **TIMER1 or TIMER2**.

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

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

See also
ENABLE

Example
```
ENABLE INTERRUPTS    'enable the setting of interrupts
ENABLE TIMER0        'enable TIMER0
DISABLE SERIAL'disables the serial interrupt.
DISABLE INTERRUPTS   'disable all interrupts
```

DISPLAY

Action
Turn LCD display on or off.

Syntax
DISPLAY ON / OFF

Remarks
The display is turned on at power up.

See also
-

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.

See also
EXIT , WHILE WEND , FOR , NEXT

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

ELSE

Action
Executed if the IF-THEN expression is false.

Syntax
ELSE

Remarks
You don't have to use the ELSE statement in an IF THEN .. END IF structure.
You can use the ELSEIF statement to test for another condition.

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

See also
IF , END IF SELECT CASE

Example
```
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
ENABLE interrupt

Remarks

Interrupt	**INT0**, **INT1**, **SERIAL**, **TIMER0, TIMER1** or **TIMER2**

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

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

See also
DISABLE

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

END

Action
Terminate program execution.

Syntax
END

Remarks
STOP can also be used to terminate a program.

When an END or STOP statement is encountered, a never-ending loop is generated.

See also
STOP

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

END IF

Action
End an IF .. THEN structure.

Syntax
END IF or **ENDIF**

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

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

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

Example
```
Dim nmb As Byte
AGAIN:                          'label
INPUT " Number "  , nmb        'ask for number
IF a = 10 THEN                  'compare
   PRINT " Number is 10"        'yes
ELSE                            'no
    IF nmb > 10 THEN            'is it greater
         PRINT " Number > 10"   'yes
    ELSE                        'no
         PRINT " Number < 10"   'print this
    END IF                      'end structure
END IF                          'end structure
END                             'end program
```

ERASE

Action
Erases a variable so memory will be released.

Syntax
ERASE var

Remarks
var The name of the variable to erase.
The variable must be dimensioned before you can erase it.

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

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

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

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

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

EXIT

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

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

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

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

FOR

Action
Execute a block of statements a number of times.

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

Remarks

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

var : Byte, Integer, Word, Long, Single.
start: Byte, Integer, Word, Long, Single, Constant.
end : Byte, Integer, Word, Long, Single, Constant.
step : Byte, Integer, Word, Long, Single, Constant.

For incremental loops, you must use TO.
For decremental loops, you must use DOWNTO.
You must end a FOR structure with the NEXT statement.
The use of STEP is optional. By default, a value of 1 is used.

See also
NEXT , EXIT FOR

Example
```
y = 10                          'make y 10
FOR a = 1 TO 10                 'do this 10 times
       FOR x = y TO 1           'this one also
              PRINT x ; a       'print the values
       NEXT                     'next x (count down)
NEXT                            'next a (count up)


Dim S as Single
For S = 1 To 2 Step 0.1
  Print S
Next
END
```

FOURTHLINE

Action
Reset LCD cursor to the fourth line.

Syntax
FOURTHLINE

Remarks
Only valid for LCD displays with 4 lines.

See also
HOME , UPPERLINE , LOWERLINE , THIRDLINE , LOCATE

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

FUSING

Action
Formats a floating-point value.

Syntax
var = Fusing(source, mask)

Remarks

Var	The string that is assigned with the result.
source	A variable of the type single that must be formatted.
mask	The formatting mask . ###.## The # sign is used to indicate the number of digits before and after the decimal point. Normal rounding is used.

See also
STR

Example
```
$large
Dim X As Single , Y As Single , Result As Single
Dim I As Integer
Dim Buf As String * 16
Input "Enter x " , X                                    'ask for 2 values
Input "Enter y " , Y
Print "X+Y=" ; : Result = X + Y : Print Result          'calculate
Print "X-Y=" ; : Result = X - Y : Print Result
Print "X/Y=" ; : Result = X / Y : Print Result
Print "X*Y=" ; : Result = X * Y : Print Result

Buf = Fusing(result , #.##)                             'format a string
Print Buf                                               'print it
```

GETRC

Action
Retrieves the value of a resistor or a capacitor.

Syntax
var = GETRC(pin)

Remarks

Var	The variable that receives the value.
pin	The port pin for the R/C is connection.

See also

{bmc Getrc.bmp}

Example

```
'-----------------------------------------------------------------
'                             GETRC.BAS
'  retrieve resistor value
'  Connect 10KOhm variable resistor from +5V to P1.7 for this example
'  Connect 10nF capacitor from P1.7 to ground
'  The GETRC(pin) function measures the time needed to charge the capacitor
'-----------------------------------------------------------------
Config Timer0 = Timer , Gate = Internal , Mode = 1   'the GETRC() functions needs
timer 0
$baud = 9600                                  'just my settings
$crystal = 11059200
Dim W As Word                                 'allocate space for variable

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


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

'As you can see there is a reasonable linearity
```

```
'So you can do some math to get the resistor/capacitor value
'But the function is intended to serve as a rough indication for resistor values
'You can also change the capacitor to get larger values.
'With 10nF, the return value fits into a byte
'Of course the R or the C value must be known in order to calculate the other
value.
```

GETRC5

Action
Retrieves a RC5 infrared code and subaddress.

Syntax
GETRC5(address , command)

Remarks

address	The RC5 sub address received.
command	The RC5 command received.

Use a sharp infrared receiver SFH506-36 and connect it to port pin 3.2 to use this command. This statement works together with the INT0 interrupt. See the example below on how to use it.

{bmc sfh506.bmp}

Example
```
'-----------------------------------------------------------------------------
'                       RC5.BAS  (c) 1999 MCS Electronics
'  connect SFH506-36 IR-receiver to PORT 3.2 (INT0)
'-----------------------------------------------------------------------------
Dim New As Bit
Dim Command As Byte , Subaddress As Byte

clr tcon.0
On Int0 Receiverc5 Nosave
Enable Int0
Enable Interrupts
Do
  If New = 1 Then                                           'received new code
     Print Command ; "  " ; Subaddress
     New = 0                                                'reset new bit
  End If
Loop



Receiverc5:                                                 'interrupt routine
  Getrc5(Subaddress, command)
  New = 1
Return
```

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

Example
```
GOSUB Routine         'branch to routine
Print "Hello"         'after being at 'routine' print this
END                   'terminate program

Routine:              'this is a subroutine
       x = x + 2      'perform some math
       PRINT X'print result
RETURN                'return
```

GOTO

Action
Jump to the specified label.

Syntax
GOTO label

Remarks
Labels can be up to 32 characters long.
When you use duplicate labels, the compiler will give you a warning.

See also
GOSUB

Example
```
Start:                'a label must end with a colon
A = A + 1             'increment a
IF A < 10 THEN'is it less than 10?
       GOTO Start     'do it again
END IF                'close IF
PRINT " Ready"        'that is it
```

HEX()

Action
Returns a string representation of a hexadecimal number.

Syntax
var = Hex(x)

Remarks

var	A string variable.
X	A numeric variable such as Byte, Integer or Word.

See also
HEXVAL

Example
```
Dim a as Byte, S as String * 10
a = 123
s  = Hex(a)
Print s
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.

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

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

Difference with QB
In QB you can use the VAL() function to convert hexadecimal strings.
But since that would require an extra test for the leading &H signs, that are required in QB, a separate function was designed.

See also
HEX , VAL , STR

Example
```
Dim a as Integer, s as string * 15
s = "000A"
a = Hexval(s) : Print a
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

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

HOME

Action
Place the cursor at the specified line at location 1.

Syntax
HOME UPPER / LOWER /THIRD / FOURTH

Remarks
If only HOME is used than the cursor will be set to the upperline.
You can also specify the first letter of the line like: HOME U

See also
CLS , LOCATE , LCD

Example
```
Lowerline
LCD " Hello"
Home Upper
LCD " Upper"
```

I2CRECEIVE

Action
Receives data from an I2C serial device.

Syntax
I2CRECEIVE slave, var
I2CRECEIVE slave, var ,b2W, b2R

Remarks

slave	A byte, Word/Integer variable or constant with the slave address from the I2C-device.
Var	A byte or integer/word variable that will receive the information from the I2C-device.
b2W	The number of bytes to write. Be cautious not to specify too many bytes!
b2R	The number of bytes to receive. Be cautious not to specify too many bytes!

In BASCOM LT you could specify DATA for var, but since arrays are supported now you can specify and array instead of DATA.

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

See also
I2CSEND

Example
```
x = 0                          'reset variable
slave = &H40                   'slave address of a PCF 8574 I/O IC
I2CRECEIVE  slave, x           'get the value
PRINT x                        'print it


Dim buf(10) as String
buf(1) = 1 : buf(2) = 2
I2CRECEIVE slave, buf(), 2, 1'send two bytes and receive one byte
Print buf(1)                   'print the received byte
```

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.

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

See also
I2CRECEIVE

Example
```
x = 5                                   'assign variable to 5
Dim ax(10) As Byte
slave = &H40                            'slave address of a PCF 8574 I/O IC
bytes = 1                               'send 1 byte
I2CSEND slave, x                        'send the value or


For a = 1 to 10
    ax(a) = a                           'Fill dataspace
Next
bytes = 10
I2CSEND slave,ax(),bytes
END
```

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, 8/9
I2CWBYTE val

Remarks

var	A variable that receives the value from the I2C-device.
8/9	Specify 8 or ACK if there are more bytes to read. (**ACK**) Specify 9 or NACK if it is the last byte to read. (**NACK**)
val	A variable or constant to write to the I2C-device.

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

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

See also

I2CRECEIVE I2CSEND

Example

```
-------- Writing and reading a byte to an EEPROM 2404 -----------------
DIM a As Byte
DIM adresW AS CONST 174      'write of 2404
DIM adresR AS CONST 175      'read adres of 2404
I2CSTART                     'generate start
I2CWBYTE  adresW             'send slaveadres
I2CWBYTE  1                  'send adres 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 slaveadres
I2CWBYTE 1            'write adres of EEPROM to read
I2CSTART              'generate repeated start
I2CWBYTE adresR       'write slaveadres of EEPROM
I2CRBYTE a,9          'receive value into a. 9 means last byte to receive
I2CSTOP               'generate stop
PRINT a               'print received value
END
```

IDLE

Action
Put the processor into the idle mode.

Syntax
IDLE

Remarks
In the idle mode, the system clock is removed from the CPU but not from the interrupt logic, the serial port or the timers/counters.
The idle mode is terminated either when an interrupt is received or upon system reset through the RESET pin.

See also
POWERDOWN

Example
```
IDLE
```

IF

Action
Allows conditional execution or branching, based on the evaluation of a Boolean expression.

Syntax
IF expression **THEN**

[**ELSEIF** expression **THEN**]
[**ELSE**]

END IF

Remarks

expression	Any expression that evaluates to true or false.

New is te ability to use the one line version of IF :
IF expression THEN statement [ELSE statement]
The use of [ELSE] is optional.

Also new is the ability to test on bits :
IF var.bit = 1 THEN

See also
ELSE , END IF

Example
```
DIM A AS INTEGER
A = 10
IF A = 10 THEN                                  'test expression
       PRINT " This part is executed."          'this will be printed
ELSE
       PRINT " This will never  be executed."   'this not
END IF
IF A = 10 THEN PRINT "New in BASCOM"
IF A = 10 THEN GOTO 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.

There are often situations where you want a number to be increased by 1.
The **INCR** statement is faster then var = var + 1.

See also
DECR

Example
```
DO                  'start loop
       INCR a       'increment a by 1
       PRINT a'print a
LOOP UNTIL a > 10   'repeat until a is greater than 10
```

INKEY

Action
Returns the ASCII value of the first character in the serial input buffer.

Syntax
var = **INKEY**

Remarks

var	Byte, Integer, Word, Long or String variable.

If there is no character waiting, a zero will be returned.

The INKEY routine can be used when you have a RS-232 interface on your uP.
See the manual for a design of a RS-232 interface.
The RS-232 interface can be connected to a comport of your computer.

See also
WAITKEY

Example
```
DO                          'start loop
       A = INKEY            'look for character
       IF A > 0 THEN        'is variable > 0?
          PRINT A           'yes , so print it
       END IF
LOOP                        'loop forever
```

INP()

Action
Returns a byte read from a hardware port or external memory location.

Syntax
var = **INP**(address)

Remarks

var	Numeric variable that receives the value.
address	The address where to read the value from.

The INP statement only works on systems with an uP that can address external Memory.

See also
OUT

Example
```
Dim a As Byte
a = INP(&H8000)       'read value that is placed on databus(d0-d7) at
                      'hex address 8000
PRINT a
END
```

INPUTBIN

Action
Read binary values from the serialport.

Syntax
INPUTBIN var1 [,var2]

Remarks

var1	The variable that is assigned with the characters from the serial port.
var2	An optional second (or more) variable that is assigned with the characters from the serial.

The number of bytes to read is depending from the variable you use.
When you use a byte variable, 1 character is read from the serial port.
An integer will wait for 2 characters and an array will wait until the whole array is filled.

Note that the INPUTBIN statement doesn't wait for a <RETURN> but just for the number of bytes.

See also
PRINTBIN

Example
```
Dim a as Byte, C as Integer
INPUTBIN a, c         'wait for 3 characters
End
```

INPUTHEX

Action
Allows input from the keyboard during program execution.

Syntax
INPUTHEX [" prompt"] , var [, varn] [NOECHO]

Remarks
prompt	An optional string constant printed before the prompt character.
Var,varn	A numeric variable to accept the input value.
NOECHO	Disables input echoed back to the Comport.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP.
See the manual for a design of an RS-232 interface.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as input device.
You can also use the build in terminal emulator.

If var is a byte then the input must be 2 characters long.
If var is an integer/word then the input must be 4 characters long.
If var is a long then the input must be 8 characters long.

Difference with QB
In QB you can specify &H with INPUT so QB will recognise that a hexadecimal string is used.
BASCOM implement a new statement: INPUTHEX.

See also
INPUT

Example
```
Dim  x As Byte
INPUTHEX " Enter a number " , x     'ask for input
```

INPUT

Action
Allows input from the keyboard during program execution.

Syntax
INPUT [" prompt"] , var [, varn] [NOECHO]

Remarks

prompt	An optional string constant printed before the prompt character.
Var,varn	A variable to accept the input value or a string.
NOECHO	Disables input echoed back to the Comport.

The INPUT routine can be used when you have an RS-232 interface on your uP.
See the manual for a design of an RS-232 interface.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as an input device.
You can also use the build in terminal emulator.

Difference with QB
In QB you can specify &H with INPUT so QB will recognise that a hexadecimal string is used.
BASCOM implements a new statement : INPUTHEX.

See also
INPUTHEX PRINT

Example
```
'-----------------------------------------------------------
'                 (c) 1997,1998 MCS Electronics
'-----------------------------------------------------------
'  file: INPUT.BAS
'  demo: INPUT, INPUTHEX
'-----------------------------------------------------------
'To use another baudrate and crystalfrequency use the
'metastatements $BAUD =  and $CRYSTAL =
$baud = 1200                              'try 1200 baud for example
$crystal = 12000000                       '12 MHz

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15                      'only for uP with XRAM support

Input "Use this to ask a question " , V
Input B1                                  'leave out for no question

Input "Enter integer " , C
Print C
```

```
Inputhex "Enter hex number (4 bytes) " , C
Print C
Inputhex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho                          'suppress echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho                          'without echo
Print S
End
```

LCD

Action
Send constant or variable to LCD display.

Syntax
LCD x

Remarks

x	Variable or constant to display.

More variables can be displayed separated by the **;** -sign
LCD a ; b1 ; " constant"
The LCD statement behaves just like the PRINT statement.

See also
LCDHEX , $LCD CONFIG LCD

Example
```
'-----------------------------------------------------------
'                 (c) 1997,1998 MCS Electronics
'-----------------------------------------------------------
'  file: LCD.BAS
'  demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
'        CURSOR, DISPLAY
'-----------------------------------------------------------
Dim A As Byte
Config Lcd = 16 * 2                     'configure lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2
'When you don't include this option 16 * 2 is assumed

'$LCD = address will turn LCD into 8-bit databus mode
'       use this with uP with external RAM and/or ROM
'       because it Ain't 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 letterer 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 0 , 31 , 17 , 17 , 17 , 17 , 17 , 31 , 0' replace ? with number (0-7)
Deflcdchar 1 , 16 , 16 , 16 , 16 , 16 , 16 , 16 , 31' replace ? with number (0-7)
Cls                                      'cls is needed after deflcdchar
Lcd Chr(0) ; Chr(1)                     'print the special character

'----------------- Now use an internal routine ------------
Acc = 1                                 'value into ACC
Call Write_lcd                          'put it on LCD
End
```

LCDHEX

Action
Send variable in hexadecimal format to the LCD display.

Syntax
LCDHEX var

Remarks

var	Variable to display.

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

The same rules apply as for PRINTHEX.

See also
LCD

Example
```
Dim a as byte
a = 255
LCD a
Lowerline
LCDHEX a
End
```

LEFT()

Action

Return the specified number of leftmost characters in a string.

Syntax

var = **Left(**var1 , n **)**

Remarks

var	The string that is assigned.
Var1	The source string.
n	The number of characters to get from the source string.

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

For string operations, all the strings must be of the same type : internal or external.

Example

```
Dim s As XRAM String * 15, z As XRAM 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.

Example
```
Dim S As String * 12
Dim A As Byte
S = "test"
A = Len(s)
Print A  ' prints 4
```

LOAD

Action
Load specified TIMER with a value for autoreload mode.

Syntax
LOAD TIMER , value

Remarks

TIMER	TIMER0, TIMER1 or TIMER2.
Value	The variable or value to load.

When you use the ON TIMERx statement with the TIMER/COUNTER in mode 2,
you can specify on which interval the interrupt must occur.
The value can range from 1 to 255 for TIMER0 and TIMER1.
For TIMER2 the range is 1-65535.

The LOAD statement calculates the correct reload value out of the parameter.
The formula : TLx = THx = (256-value)
For TIMER2 : RCAP2L = RCAP2H = (65536 - value)

The load statement is not intended to assign/read a value to/from the timers/counters. Use COUNTERx instead.

See additional hardware for more details

Example
```
LOAD TIMER0, 100              'load TIMER0 with 100

Will generate :
Mov tl0,#h'9C
Mov th0,#h'9C

LOAD TIMER2, 1000
Will generate:
Mov RCAP2L,#24
Mov RCAP2H,#252
```

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
CONFIG LCD , LCD , HOME , CLS

Example
```
LCD "Hello"
Locate 1,10
LCD "*"
```

LOOKUP

Action
Returns a value from a table.

Syntax
var **=LOOKUP(** value**,** label **)**

Remarks

Var	The returned value
Value	A value with the index of the table
Label	The label where the data starts

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

Difference with BASCOM LT
In BASCOM LT, the lookup function only works with byte tables.
In BASCOM-8051, you can use it with integer, word, long and single types too.

See also
LOOKUPSTR

Example
```
DIM b1 As Byte ,  I as Integer
b1 = Lookup(1, dta)
Print b1              ' Prints 2 (zero based)

I = Lookup(0,DTA2)
End


DTA:
DATA 1,2,3,4,5

DTA2:          'integer data
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

Value : Byte, Integer, Word, Long, Constant. Range(0-255)

See also
LOOKUP

Example
```
Dim s as string, idx as Byte
idx = 0 : s = LookupStr(idx,Sdata)
Print s  'will print 'This'
End

Sdata:
Data "This" , "is" ,"a test"
```

LOW

Action
Retrieves the least significant byte of a variable.

Syntax
var = LOW (s)

Remarks

Var	The variable that is assigned with the LSB of var S.
S	The source variable to get the LSB from.

See also
HIGH

Example
```
Dim I As Integer , Z As Byte
I = &H1001
Z = Low(I)  ' is 1
```

LOWERLINE

Action
Reset the LCD cursor to the lowerline.

Syntax
LOWERLINE

Remarks
-

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.

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
MAKEDEC BCD()

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
LCD a
Lowerline
LCD BCD(a)
a = MakeDEC(a)
LCD "   "  ; a
End
```

MID()

Action

The MID function returns part of a string (a sub string).
The MID statement replaces part of a string variable with another string.

Syntax

var = **MID(**var1 ,st [, l] **)**
MID(var ,st [, l] **)** = var1

Remarks

var	The string that is assigned.
Var1	The source string.
st	The starting position.
l	The number of characters to get/set.

Operations on strings require that all strings are of the same type(internal or external)

See also

LEFT , RIGHT

Example

```
Dim s As XRAM String * 15, z As XRAM String * 15
s = "ABCDEFG"
z = Mid(s,2,3)
Print z       'BCD
z="12345"
Mid(s,2,2) = z
Print s       'A12DEFG
End
```

MOD

Action
Returns the remainder of a division.

Syntax
ret = var1 **MOD** var2

Remarks

ret	The variable that receives the remainder.
var1	The variable to divide.
var2	The divisor.

Example
```
a = 10 MOD 3          'divide 10 through 3
PRINT a               'print remainder (1)
```

NEXT

Action
Ends a FOR..NEXT structure.

Syntax
NEXT [var]

Remarks

Var	The index variable that is used as a counter when you form the structure with FOR var. Var is optional and not needed.

You must end each FOR statement with a NEXT statement.

See also
FOR

Example
```
y = 10                        'make y 10
FOR a = 1 TO 10               'do this 10 times
       FOR x = y TO 1'this one also
              PRINT x ; a     'print the values
       NEXT                   'next x (count down)
NEXT a                        'next a (count up) END
```

ON Interrupt

Action
Execute subroutine when specified interrupt occurs.

Syntax
ON interrupt label [NOSAVE]

Remarks

interrupt	INT0, INT1, SERIAL, TIMER0 ,TIMER1 or TIMER2. Chip specific interrupts can be found under microprocessor support.
Label	The label to jump to if the interrupt occurs.
NOSAVE	When you specify NOSAVE, no registers are saved and restored in the interrupt routine. So when you use this option be sure to save and restore used registers.

You must return from the interrupt routine with the RETURN statement.
You may have only one RETURN statement in your interrupt routine because the compiler restores the registers and generates a RETI instruction when it encounters a RETURN statement in the ISR.

You can't use TIMER1 when you are using SERIAL routines such as PRINT,
Because TIMER1 is used as a BAUD RATE generator.

When you use the INT0 or INT1 interrupt you can specify on which condition the interrupt must be triggered.
You can use the Set/Reset statement in combination with the TCON-register for this purpose.

SET TCON.0 : trigger INT0 by falling edge.
RESET TCON.0 : trigger INT0 by low level.
SET TCON.2 : trigger INT1 by falling edge.
RESET TCON.2 : trigger INT1 by low level.

See Hardware for more details

Example
```
ENABLE INTERRUPTS
ENABLE INT0                  'enable the interrupt
ON INT0 Label2 nosave        'jump to label2 on INT0
DO                           'endless loop
LOOP
END

Label2:
    PRINT " An hardware interrupt occurred!"            'print message
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 **]**

Remarks

var	The numeric variable to test. This can also be a SFR such as P1.
label1, label2	The labels to jump to depending on the value of *var*.

Note that the value is zero based. So when var = 0, the first specified label is jumped/branched.

Example

```
x = 2                                   'assign a variable interrupt
ON x GOSUB lbl1, lbl2,lbl3              'jump to label lbl3
x=0
ON x GOTO lbl1, lbl2 , lbl3
END


lbl3:
  PRINT " lbl3"
RETURN


Lbl1:

                                        Lbl2:
```

OPEN - CLOSE

Action
Opens and closes a device.

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

Remarks

device	There are 2 hardware devices supported: COM1 and COM2. With the software UART, you must specify the portpin and the baudrate. COM3.0:9600 will use PORT 3.0 at 9600 baud.
MODE	You can use BINARY, INPUT or OUTPUT for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT.
channel	The number of the channel to open. Must be a positive constant.

Since there are uP's such as the 80537 with 2 serial channels on board, the compiler must know which serial port you want to use. That is why the OPEN statement is implemented. With only 1 serial port on board, you don't need this statement.
The statements that support the device are PRINT , PRINTHEX, INPUT and INPUTHEX.

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

The software UART, only supports the GET and PUT statements to retrieve and send data. COM1: and COM2: are hardware ports, and can be used with PRINT etc.

See also

Example 1
```
'only works with a 80517 or 80537
CONFIG BAUD1 = 9600                 'serial 1 baudrate
OPEN "COM2:" FOR BINARY AS #1       'open the port
PRINT #1, "Hello"                   'print to serial 1
PRINT "Hello"                       'print to serial 0
CLOSE #1                            'close the channel
```

Example 2
```
'works with every port pin
Dim A As Byte , S As String * 16 , I As Byte , Dum As Byte

'a software comport is named after the pin you use
'for example P3.0  will be "COM3.0:"  (so there is no P)
'for software comports, you must provide the baudrate
'So for 9600 baud, the devicename is "COM3.0:9600"
'When you want to use the pin for sending, you must open the device for OUTPUT
'When you want to use the pin for receiving, you must open the device for INPUT

'At this time only variables can be send and received with the PUT and GET
statements.
```

```
'In the feature PRINT etc. will support these software comports.

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


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

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


Close #1                                                    ' finally close
device
Close #2
End
```

OUT

Action
Sends a byte to a hardware port or external memory address.

Syntax
OUT address, value

Remarks

address	The address where to send the byte to.
value	The variable or value to send.

The OUT statement only works on systems with an uP that can address external Memory.

See also
INP

Example
```
Dim a as byte
OUT &H8000,1          'send  1 to the databus(d0-d7) at hex address 8000
END

Will generate :
Mov A,#1
Mov dptr,#h'8000
Movx @dptr,a
```

P1,P3

Action
P1 and P3 are special function registers that are treated as variables.

Syntax
Px = var
var = **Px**

Remarks

x	The number of the port. (1 or 3). **P3.6 can't be used with an AT89C2051!**
Var	The variable to retrieve or to set.

Note that other processors can have additional ports such as P0,P2,P4 etc.
When you select the proper **.DAT** file, you can also use these ports as variables.
In fact, you can use any SFR as a variable in BASCOM.

ACC = 0 'will reset the accumulator for example

See hardware for a more detailed description of the ports.

Example
```
Dim a as BYTE, b1 as BIT
a = P1                'get value from port 1
a = a OR 2            'manipulate it
P1 = a                'set port 1 with new value
P1 = &B10010101       'use binary notation
P1 = &HAF             'use hex notation
b1 = P1.1             'read pin 1.1
P1.1 = 0              'set it to 0
```

PEEK()

Action
Returns a byte stored in internal memory.

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.(0-255)

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 memory location.

Syntax
POKE address , value

Remarks

address	Numeric variable with the address of the memory location to set. (0-255)
value	Value to assign. (0-255)

Be careful with the POKE statement because you can change variables with it that can cause your program to function incorrect.

See also
PEEK , CPEEK , INP , OUT

Example
```
POKE 127, 1          'write 1 to address 127
End
```

POWERDOWN

Action
Put processor into powerdown mode.

Syntax
POWERDOWN

Remarks
The powerdown mode stops the system clock completely.
The only way to reactivate the micro controller is by system reset.

See also
IDLE

Example
```
POWERDOWN
```

PRINT

Action
Send output to the RS-232 port.

Syntax
PRINT var ; " constant"

Remarks

var	The variable or constant to print.

You can use a semicolon (;) to print more than one variable at one line.
When you end a line with a semicolon, no linefeed will be added.

The PRINT routine can be used when you have a RS-232 interface on your uP.
See the manual for a design of a RS-232 interface.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator as an output device.
You can also use the build in terminal emulator.

See also
PRINTHEX , INPUT , OPEN , CLOSE

Example
```
'-----------------------------------------------------------
'                (c) 1997,1998 MCS Electronics
'-----------------------------------------------------------
'  file: PRINT.BAS
'  demo: PRINT, PRINTHEX
'-----------------------------------------------------------
Dim A As Byte , B1 As Byte , C As Integer
A = 1
Print "print variable a " ; A
Print                                   'new line
Print "Text to print."                  'constant to print


B1 = 10
Printhex B1                             'print in hexa notation
C = &HA000                              'assign value to c%
Printhex C                              'print in hex notation
Print C                                 'print in decimal notation

C = -32000
Print C
Printhex C
Rem Note That Integers Range From -32767 To 32768
End
```

PRINTBIN

Action
Print binary content of a variable to the serial port.

Syntax
PRINTBIN var [,varn]

Remarks
var	The variable which value is send to the serial port.
varn	Optional variables to send.

PRINTBIN is equivalent to PRINT CHR(var); but whole arrays can be printed this way.

When you use a Long for example, 4 bytes are printed.

See also
INPUTBIN

Example
```
Dim a(10) as Byte, c as Byte
For c = 1 To 10
  a(c) = a            'fill array
Next
PRINTBIN a(1) 'print content
```

PRINTHEX

Action
Sends a variable in hexadecimal format to the serial port.

Syntax
PRINTHEX var

Remarks
var The variable to print.

The same rules apply to PRINTHEX as PRINT.

The PRINTHEX routine can be used when you have a RS-232 interface on your uP.
See the manual for a design of a RS-232 interface.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator as an output device.
You can also use the build in terminal emulator.

See also
PRINT , INPUTHEX

Example
```
Dim x  As Byte
INPUT x              'ask for var
PRINT x              'print it in decimal format
PRINTHEX "Hex " ; x         'print it in hex format
```

PRIORITY

Action
Sets the priority level of the interrupts.

Syntax
PRIORITY SET / RESET interrupt

Remarks

SET	Bring the priority level of the interrupt to a higher level.
RESET	Bring the priority level of the interrupt to a lower level.
Interrupt	The interrupt to set or reset.

The interrupts are: **INT0, INT1, SERIAL, TIMER0, TIMER1 and TIMER2.**

Interrupt INT0 always has the highest priority.
When more interrupts occur at the same time the following order is used to handle the interrupts.

Note that other microprocessors can have additional/other interrupt setting.
Read microprocessor support to check the additions.

Interrupt	**Priority**
INT0	1 (highest)
TIMER0	2
INT1	3
TIMER1	4
SERIAL	5 (lowest)

Example

```
PRIORITY SET SERIAL          'serial int highest level
ENABLE SERIAL                'enable serial int
ENABLE TIMER0                'enable timer0 int
ENABLE INTERRUPTS            'activate interrupt handler
ON SERIAL label              'branch to label if serial int occur
DO                           'loop for ever

LOOP

Label:                       'start label
   PRINT " Serial int occurred."          'print message
RETURN                       'return from interrupt
```

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
```
Dim A As Byte, I As Byte, C As Integer, S As XRAM String * 10
RESTORE dta
FOR a = 1 TO 3
  READ i : PRINT i
NEXT
RESTORE DTA2
READ C : PRINT C
READ C : PRINT C
Restore dta3 : Read s : Print s
END

dta:
Data 5,10,15
dta2:
Data 1000%, -2000%
dta3:
Data " hello"
```

REM

Action
Instruct the compiler that comment will follow.

Syntax
REM or **'**

Remarks
You can comment your program for clarity.
You can use REM or **'** followed by your comment.
All statements after REM or **'** are treated as comment so you cannot use statements after a REM statement.

New is the possibility to use block comments:
'(start block comment
print "This will not be compiled
') end block comment

Note that the starting ' sign will ensure compatibility with QB

Example
```
REM  TEST.BAS version 1.00
PRINT a'    " this is comment   :  PRINT " hello"
                        ^--- this will not be executed!
```

RESET

Action
Reset a bit of a PORT (P1.x, P3.x) or an internal bit/byte/integer/word variable.

Syntax
RESET bit
RESET var.x

Remarks

bit	Can be a P1.x, P3.x or any bit variable where x=0-7.
var	Can be a byte, integer or word variable.
x	Constant of variable to reset.(0-7) for bytes and (0-15) for Integer/Word

See also
SET

Example
```
Dim b1 as bit, b2 as byte, I as Integer
RESET P1.3                  'reset bit 3 of port 1
RESET b1                    'bitvariable
RESET b2.0                  'reset bit 0 of bytevariable b2
RESET I.15                  'reset MS bit from I
```

RESTORE

Action
Allows READ to reread values in specified DATA statements.

Syntax
RESTORE label

Remarks

label	The label of a DATA statement.

See also
DATA , READ

Example
```
DIM a AS BYTE, I AS BYTE
RESTORE dta
FOR a = 1 TO 3
  READ a : PRINT a
NEXT
RESTORE DTA2
READ I : PRINT I
READ I : PRINT I
END

DTA1:
Data 5, 10, 100

DTA2:
Data -1%, 1000%
Integers must end with the %-sign. (Integer : <0 or >255)
```

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
```
GOSUB  Pr                          'jump to subroutine
PRINT result                       'print result
END                                'program ends

Pr:                                'start subroutine with label
       result = 5 * y              'do something stupid
       result = result + 100       'add something to it
RETURN                             'return
```

RIGHT()

Action
Return a specified number of rightmost characters in a string.

Syntax
var = **RIGHT(**var1 ,st **)**

Remarks

var	The string that is assigned.
Var1	The sourcestring.
st	The starting position.

All strings must be of the same datatype, internal or external.

See also
LEFT , MID

Example
```
Dim s As XRAM String * 15, z As XRAM String * 15
s = "ABCDEFG"
z = Right(s,2)
Print z                     'FG
End
```

ROTATE

Action
Shifts 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.

Note that the carryflag goes into the LSB or MSB depending on the shift direction. This works just like the ASM statements RLC and RRC. When this behaviour is not wanted, clear the carry bit before a shift with the CLR C statement.

See also
SHIFTIN , SHIFTOUT

Example
```
Dim a as Byte
a = 128
ROTATE a, LEFT , 2
Print a       '1

Generated code :

Mov R7,#2
Mov R0,#h'21
Mov a,@r0
Rlc a
Djnz r7,*-1
Mov @r0,a
```

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.

See also

-

Example

```
Dim b2 as byte
SELECT CASE b2       'set bit 1 of  port 1
  CASE 2 : PRINT "2"
  CASE 4 : PRINT "4"
  CASE IS >5 : PRINT ">5"    'a test requires the IS keyword
  CASE ELSE
END SELECT
END
```

SET

Action
Set a bit of a PORT(P1.x,P3.x) or a bit/byte/integer/word variable.

Syntax
SET bit
SET var.x

Remarks

Bit	P1.x, P3.x or a Bitvariable.
Var	A byte, integer, word or long variable.
X	Bit of variable (0-7) to set. (0-15 for Integer/Word)

See also
RESET

Example
```
Dim b1 as Bit, b2 as byte, c as Word, L as Long
SET P1.1      'set bit 1 of  port 1
SET b1        'bitvariable
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
```

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 and SHIFTOUT

Action
Shifts a bitstream in or out a variable.

Syntax
SHIFTIN pin , pclock , var , option
SHIFTOUT pin , pclock , var , option

Remarks

pin	The portpin which serves as as input/output.
pclock	The portpin which generates the clock.
Var	The variable that is assigned.
Option	Option can be : 0 - MSB shifted in/out first when clock goes low 1 - MSB shifted in/out first when clock goes high 2 - LSB shifted in/out first when clock goes low 3 - LSB shifted in/out first when clock goes high For the SHIFTIN statement, you can add 4 to the parameter to use the external clock signal for shifting.

It depends on the type of the variable, how many shifts will occur.
When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur.

See also

Example
```
Dim a as byte
SHIFTIN P1.0 , P1.1 , a , 0
SHIFTOUT P1.2 , P1.1 , a , 0

For the SHIFTIN example the following code is generated:
Setb P1.1
Mov R0,#h'21
Mov r2,#h'01
__UNQLBL1:
Mov r3,#8
__UNQLBL2:
Clr P1.1
Nop
Nop
Mov c,P1.0
Rlc a
Setb P1.1
Nop
Nop
Djnz r3,__UNQLBL2
Mov @r0,a
Dec r0
Djnz r2,__UNQLBL1

Of course it depends on the parameter, which code will be generated.
To shift with an external clock signal:
SHIFTIN P1.0, P1.1 , a , 4  'add 4 for external clock
```

Generated code:

```
Mov R0,#h'21
Mov r2,#h'01
__UNQLBL1:
Mov r3,#8
__UNQLBL2:
Jnb P1.1,*+0
Mov c,P1.0
Rlc a
Jb P1.1,*+0
Djnz r3,__UNQLBL2
Mov @r0,a
Dec r0
Djnz r2,__UNQLBL1
```

SHIFTLCD

Action
Shift the LCD display left or right by one position.

Syntax
SHIFTLCD LEFT / RIGHT

Remarks
-

See also
SHIFTCURSOR

Example
```
LCD "Very long text"
SHIFTLCD LEFT
Wait 1
SHIFTLCD RIGHT
End
```

SOUND

Action
Sends pulses to a port pin.

Syntax
SOUND pin, duration, frequency

Remarks

pin	Any I/O pin such as P1.0 etc.
duration	The number of pulses to send. Byte, integer/word or constant. (1- 32768).
Frequency	The time the pin is pulled low and high.

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 *frequency* uS.
This loop is executed *duration* times.

See also
-

Example
```
SOUND P1.1 , 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.

Example
```
Dim s as XRAM String * 15, z as XRAM String * 15
s = Space(5)
Print " {"  ;s ; " }"                          '{     }

Dim A as Byte
A = 3
S = Space(a)

Genereated code for last 2 lines :
; ---------- library routine -----------
_sStr_String:
Mov @r1,a
Inc r1
Djnz r2,_sStr_String
Clr a
Mov @r1,a
Ret
;----------------------------------------
Mov R1,#h'22 ; location of string
Mov R2,h'21  ; number of spaces
Mov a,#32
Acall _sStr_String
```

SPIIN

Action
Reads a value from the SPI-bus.

Syntax
SPIIN var, bytes

Remarks

var	The variable that is assigned with the value read from the SPI-bus.
bytes	The number of bytes to read.

See also
SPIOUT , CONFIG SPI

Example
```
Dim a(10) as byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SPIIN a(1) , 4        'read 4 bytes
```

SPIOUT

Action
Sends a value of a variable to the SPI-bus.

Syntax
SPIOUT var , bytes

Remarks

var	The variable whose content must be send to the SPI-bus.
bytes	The number of bytes to send.

See also
SPIIN , CONFIG SPI

Example
```
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
Dim a(10) as Byte , X As Byte
SPIOUT a(1) , 5       'send 5 bytes
SPIOUT X , 1          'send 1 byte
```

START

Action
Start the specified timer/counter.

Syntax
START timer

Remarks

timer	TIMER0, TIMER1, TIMER2, COUNTER0 or COUNTER1.

You must start a timer/counter in order for an interrupt to occur (when the external gate is disabled).

TIMER0 and COUNTER0 are the same device.

See also
STOP TIMERx

Example
```
ON TIMER0 label2
LOAD TIMER0, 100
START TIMER0
DO                    'start loop
LOOP                  'loop forever

label2:               'perform an action here

RETURN
```

STOP

Action
Stop program execution.

Syntax
STOP

Remarks
END can also be used to terminate a program.

When an END or STOP statement is encountered, a never-ending loop is generated.

Example
```
PRINT var             'print something
STOP                  'thats it
```

STOP TIMERx

Action
Stop the specified timer/counter.

Syntax
STOP timer

Remarks
timer	TIMER0, TIMER1, TIMER2, COUNTER0 or COUNTER1.

You can stop a timer when you don't want an interrupt to occur.

TIMER0 and COUNTER0 are the same.

See also
START TIMERx

Example
```
'-------------------------------------------------------------
'                 (c) 1997,1998 MCS Electronics
'-------------------------------------------------------------
'  file: TIMER0.BAS
```

```
'  demo: ON TIMER0
'  *TIMER1 is used for RS-232 baudrate generator
'-----------------------------------------------------------
Dim Count As Byte , Gt As Byte

Config Timer0 = Timer , Gate = Internal , Mode = 2
'Timer0 = counter : timer0 operates as a counter
'Gate = Internal  : no external gate control
'Mode = 2         : 8-bit auto reload (default)

On Timer0 Timer_0_int
Load Timer0 , 100                       'when the timer reaches 100 an interrupt
                                        'will occur
Enable Interrupts                       'enable the use of interrupts
Enable Timer0                           'enable the timer


Rem Setting Of Priority
Priority Set Timer0                     'highest priority
Start Timer0                            'start the timer

Count = 0                               'reset counter
Do
  Input "Number " , Gt
  Print "You entered : " ; Gt
Loop Until Gt = 1                       'loop until users presses ESC key
Stop Timer0
End


Rem The Interrupt Handler For The Timer0 Interrupt
Timer_0_int:
   Inc Count
   If Count = 2 Then
      Print "Timer0 Interrupt occured"
      Count = 0
   End If
Return
```

STR()

Action
Returns a string representation of a number.

Syntax
var **= Str(** x **)**

Remarks

var	A string variable.
X	A numeric variable.

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

The string must be big enough to store the string.

See also
VAL

Difference with QB
In QB STR() returns a string with a leading space. This behaviour is not in BASCOM.

Example
```
Dim a as Byte, S as XRAM String * 10
a = 123
s  = Str(a)
Print s
End
```

STRING()

Action

Returns a string consisting of m repetitions of the character with ASCII Code n.

Syntax

var = **STRING(**m ,n **)**

Remarks

var	The string that is assigned.
n	The ASCII-code that is assigned to the string.
m	The number of characters to assign.

Since a string is terminated by a 0 byte, you can't use 0 for n.
Using 0 for m will result in a string of 255 bytes, because there is no check on a length assign of 0. When you need this let us know.

See also

SPACE

Example

```
Dim s as XRAM String * 15
s = String(5,65)
Print s                     'AAAAA
End
```

SUB

Action
Defines a Sub procedure.

Syntax
SUB Name[(var1)**]**

Remarks

name	Name of the sub procedure, can be any non-reserved word.
var1	The name of the parameter.

You must end each subroutine with the END SUB statement.

You must Declare Sub procedures before the SUB statement.
The parameter names and types must be the same in both the declaration and the Sub procedure.

Parameters are global to the application.
That is the used parameters must be dimensioned with the DIM statement.
Therefore, the variables can be used by the program and sub procedures.
The following examples will illustrate this:

```
Dim a as byte, b1 as byte, c as byte       'dim used variables
Declare Sub Test(a as byte)                'declare subroutine
a = 1 : b1 = 2: c = 3              'assign variables

Print a ; b1 ; c                           'print them

Call Test(b1)                              'call subroutine
Print a ;b1 ; c                            'print variables again
End

Sub Test(a as byte)                        'begin procedure/subroutine
  print a ; b1 ; c                         'print variables
End Sub
```

See also
CALL, DECLARE

Example
-

SWAP

Action
Exchange two variables of the same type.

Syntax
SWAP var1, var2

Remarks

var1	A variable of type bit, byte, integer or word.
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
```

THIRDLINE

Action
Reset LCD cursor to the third line.

Syntax
THIRDLINE

Remarks
-

See also
UPPERLINE , LOWERLINE , FOURTHLINE

Example
```
Dim a as byte
a = 255
LCD a
Thirdline
LCD a
Upperline
End
```

UPPERLINE

Action
Reset LCD cursor to the upperline.

Syntax
UPPERLINE

Remarks
-

See also
LOWERLINE THIRDLINE FOURTHLINE

Example
```
Dim a as byte
a = 255
LCD a
Lowerline
LCD a
Upperline
End
```

VAL()

Action
Converts a string representation of a number into a number.

Syntax
var **= Val(** s **)**

Remarks

var	A numeric variable that is assigned with the value of s.
s	Variable of the string type.

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

See also
STR

Example
```
Dim a as byte, s As XRAM string * 10
s = "123"
a = Val(s)           'convert string
Print a
End
```

VARPTR()

Action
Retrieves the memory-address of a variable.

Syntax
var = VARPTR(var2)

Remarks

var	The variable that is assigned with the address of var2.
var2	A variable to retrieve the address from.

See also
PEEK POKE

Example
```
Dim I As Integer , B1 As Byte
B1 = Varptr(I)

Generated code:
Mov h'23,#h'21
```

WAIT

Action
Suspends program execution for a given time.

Syntax
WAIT seconds

Remarks

seconds	The number of seconds to wait.

The delay time is based on a clockfrequency of 12 Mhz.
No accurate timing is possible with this command.
When you use interrupts, the delay can be extended.

See also
DELAY

Example
```
WAIT 3          'wait for three seconds
Print "*"
```

WAITKEY

Action
Wait until a character is received in the serial buffer.

Syntax
var = **WAITKEY**

Remarks

var	Variable that is assigned with the ASCII value of the serial buffer.

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

See also
INKEY

Example
```
Dim A As Byte
A = Waitkey           'wait for character
Print A
```

WAITMS

Action
Suspends program execution for a given time in mS.

Syntax
WAITMS mS

Remarks

mS	The number of milliseconds to wait. (1-255)

The delay time is based on a clock frequency of 12 Mhz.
No accurate timing is possible with this command.
In addition, the use of interrupts can slow this routine.
This statement is provided for the I2C statements.
When you write to an EEPROM you must wait for 10 mS after the write instruction.

See also
DELAY WAIT

Example
```
WAITMS 10            'wait for 10 mS
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 condition.
If it is still true, the process is repeated.
If it is not true, execution resumes with the statement following the WEND statement.

See also
DO .. LOOP

Example
```
WHILE a <= 10
       PRINT a
      INC a
WEND
```

Hardware - LCD display

The LCD display can be connected as follows:

LCD-DISPLAY	PORT	PIN
DB7	P1.7	14
DB6	P1.6	13
DB5	P1.5	12
DB4	P1.4	11
E	P1.3	6
RS	P1.2	4
RW	Ground	5
Vss	Ground	1
Vdd	+5 Volt	2
Vo	0-5 Volt	3

This leaves P1.1 and P1.0 and P3 for other purposes.

You can change the LCD pin layout from the Options LCD menu.
You can select the display used with the CONFIG LCD statement.

The LCD display operates in 4-bit mode.
See the $LCD statement for operation in 8-bit mode.

BASCOM supports many statements to control the LCD display.
For those who want to have more control, the example below shows how to do so.

```
Acc = 5                 'load register A with value
Call Lcd_control        'it is a control value to control the display
Acc = 65                'load with new value (letter A)
Call Write_lcd          ‘writes it to the LCD display
```

Note that lcd_control and write_lcd are assembler subroutines which can be called from BASCOM.

See manufacture details from your LCD display for the correct assignment.

Microprocessor support

Some microprocessors have additional features compared to the AT89C2051/8051.

8032/8052/AT89S8252
TIMER2

AT89S8252
WATCHDOG
DATA EEPROM
Alternative port-pin functions

80515,80535,80517,80535
GETAD
WATCHDOG
BAUDRATE GENERATOR
INTERRUPTS and PRIORITY

80517,80537
GETAD
WATCHDOG
BAUDRATE GENERATOR
BAUDRATE GENERATOR1
INTERRUPTS and PRIORITY

AT898252 WATCHDOG

The AT89S8252 has a build in watchdog timer.
A watchdog timer is a timer that will reset the uP when it reaches a certain value.
So during program execution this WD-timer must be reset before it exceeds its maximum value.
This is used to be sure a program is running correct.
When a program crashes or sits in an endless loop it will not reset the WD-timer so an automatic reset will occur resulting in a restart.

START WATCHDOG 'will start the watchdog timer.
STOP WATCHDOG 'will stop the watchdog timer.
RESET WATCHDOG 'will reset the watchdog timer.

See also
CONFIG WATCHDOG

Example
```
'-----------------------------------------------------
'                (c) 1998 MCS Electronics
' WATCHD.BAS demonstrates the AT89S8252 watchdog timer
' select 89s8252.dat !!!
'-----------------------------------------------------
Config Watchdog = 2048                  'reset after 2048 mSec
Start Watchdog                          'start the watchdog timer
Dim I As Word
```

```
For I = 1 To 10000
  Print I                                 'print value
  '  Reset Watchdog
  'you will notice that the for next doesnt finish because of the reset
  'when you unmark the RESET WATCHDOG statement it will finish because the
  'wd-timer is reset before it reaches 2048 msec
Next
End
```

WATCHDOG

The AT89S8252 has a build in watchdog timer.
A watchdog timer is a timer that will reset the uP when it reaches a certain value.
So during program execution this WD-timer must be reset before it exceeds its maximum value. This is used to be sure a program is running correct.
When a program crashes or sits in an endless loop it will not reset the WD-timer so an automatic reset will occur resulting in a restart.

CONFIG WATCHDOG = value

value	The time in mS it takes the WD will overflow, causing a reset. Possible values are : 16,32,64,128,256,512,1024 or 2048

START WATCHDOG will start the watchdog timer.
STOP WATCHDOG will stop the watchdog timer.
RESET WATCHDOG will reset the watchdog timer.

Example

```
DIM A AS INTEGER
CONFIG WATCHDOG = 2048              'after 2 seconds a reset will occur
START WATCHDOG              'start the WD
DO
  PRINT a
  a = a + 1                         'notice the reset
  REM RESET WATCHDOG        'delete the REM to run properly
LOOP
END
```

DATA EEPROM

The AT89S8252 has a build in 2Kbytes flash EEPROM.
You can use this to store data.
Two statements are provided: WRITEEEPROM and READEEPROM.

WRITEEEPROM var [, address]

var	Any BASCOM variable name.
Address	The address of the EEPROM where to write the data to. Ranges from 0 to 2047.

	When you omit the address the address will be assigned automatic. You can view the assigned address in the report file.

READEEPROM var [, address]

var	Any BASCOM variable name.
Address	The address of the EEPROM where to read the data from. Ranges from 0 to 2047. You can omit the address when you have written a value before with the **WRITEEEPROM** var statement. Because in that case the compiler knows about the address because it is assigned by the compiler.

Example

```
Dim S As String * 15 , S2 As String * 10
S = "Hello" : S2 = "test"

Dim L As Long
L = 12345678
Writeeeprom S
Writeeeprom S2                          'write strings
Writeeeprom L                           'write long

S = "" : S2 = "" : L = 0                'clear variables
Readeeprom L : Print L
Readeeprom S : Print S
Readeeprom S2 : Print S2
End
```

Alternative port-pin functions

The AT89S8252 ports have alternative functions.
The followng table shows the alternative functions.

Port pin	Alternate function
P1.0	T2 external count input to timer.counter 2, clock out
P1.1	T2EX timer/counter 2 capture/reload trigger and direction flag
P1.4	/SS Slave port select input
P1.5	MOSI Master data output, slave data input pin for SPI channel
P1.6	MISO Master data input, slave data output pin for SPI channel
P1.7	SCK Master clock output, slave clock input pin for SPI channel
P3.0	RxD serial input port
P3.1	TxD serial output port
P3.2	/INT0 external interrupt 0
P3.3	/INT1 external interrupt 1
P3.4	T0 timer 0 external input
P3.5	T1 timer 1 external input
P3.6	/WR external data memory write strobe
P3.7	/RD external data memory read strobe

/ Means active low

TIMER2 in 8032 and compatibles

Some microprocessors have an additional timer on board : TIMER2.
This section describes the 8032 compatible TIMER2 and is not compatible with th TIMER2 found in the 80C535 and others.
TIMER2 is a 16-bit timer/counter which can operate as either an event timer or an event counter. TIMER2 has three main operating modes : capture, auto-reload(up or down counting) , and baud rate generator.

Capture mode

In the capture mode there are two options :

- 16-bit timer/counter, which upon overflowing sets, bit TF2, the TIMER2 overflow bit. This bit can be used to generate an interrupt.

  Counter mode :
  CONFIG TIMER2 = COUNTER, GATE = INTERNAL, MODE = 1

  Timer mode:
  CONFIG TIMER2=TIMER, GATE= INTERNAL,MODE =1

- As above but with the added future that a 1 to 0 transition on at external input T2EX causes the current values in the TIMER2 registers TL2 and TH2 to be captured into the capture registers RCAP2L and RCAP2H.

  Counter mode:
  CONFIG TIMER2 = COUNTER, GATE = EXTERNAL, MODE = 1

  Timer mode:
  CONFIG TIMER2=TIMER,GATE=EXTERNAL,MODE=1

In addition the transition at T2EX causes bit EXF2 in T2CON to be set and EXF2 like TF2 can generate an interrupt.

The TIMER2 interrupt routine can interrogate TF2 and EXF2 to determine which event caused the interrupt.
(There is no reload value in this mode. Even when a capture event occurs from T2EX the counter keeps on counting T2EX pin transitions or osc/12 pulses)

Auto reload mode

In the 16-bit auto reload mode, TIMER2 can be configured as a timer or counter, which can be programmed to count, up or down. The counting direction is determined by bit DCEN.
TIMER2 will default to counting up to **&H**FFFF and sets the TF2 overflow flag bit upon overflow. This causes the TIMER2 registers to be reloaded with the 16-bit value in RCAP2L and RCAP2H.
The values in RCAP2L and RCAP2H are pre-set by software means.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

Timer mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

If EXEN2=1 then a 16-bit reload can be triggered either by an overflow or by a 1 to 0 transition at input T2EX. This transition also sets the EXF2 bit. The TIMER2 interrupt, if enabled, can be generated when either TF2 or EXF2 are 1.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=EXTERNAL,MODE=0

Timer mode:
CONFIG TIMER2=TIMER,GATE=EXTERNAL,MODE=0

TIMER2 can also count up or down. This mode allows pin T2EX to control the direction of count. When logic 1 is applied at pin T2EX TIMER2 will count up. TIMER2 will overflow at **&H**FFFF and sets the TF2 flag, which can then generate an interrupt, if the interrupt is enabled. This timer overflow also causes the 16-bit value in RCAP2L en RCAP2H to be reloaded in to the timer registers TL2 and TH2.

Counter mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP

Timer mode:
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP

Logic 0 applied at pin T2EX causes TIMER2 to count down. The timer will under flow when TL2 and TH2 become equal to the value stored in RCAP2L and RCAP2H. TIMER2 under flows sets the TF2 flag and causes **&H**FFFF to be reloaded into the timer registers TL2 and TH2.

Counter mode:
CONFIG
TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=DOWN

Timer mode:
CONFIG
TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=DOWN

The external flag TF2 toggles when TIMER2 under flows or overflows.
The EXF2 flag does not generate an interrupt in counter UP/DOWN mode.

Baud rate generator

This mode can be used to generate a baud rate for the serial port. TIMER1 can be used for an other task this way.
CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=2

Receive only

This mode can be used to generate the baudrate for the receiver only.
TIMER1 can be used for the transmission with an other baudrate.
CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=3

Note that TIMER1 must be setup from assembler this way.

Transmit only

This mode can be used to generate the baud rate for transmitter only.
TIMER1 can be used for the reception with an other baudrate.
CONFIG TIMER2=TIMER,GATE=INTERNAL,MODE=4

Note that TIMER1 must be set-up from assembler this way.
Clock output

Some 8052 deviants have the ability to generate a 50% duty cycle clock on P1.0.
CONFIG TIMER2=TIMER,MODE=5

The output frequency = (fOSC **/** 4) **/** (65536-CAPTURE)

Use CAPTURE = value to set the capture register.

How to determine what caused the interrupt

You can test the bit T2CON.7 to see if a overflow caused the interrupt.
You can test bit T2CON.6 whether either a reload or capture is caused by a negative transition on T2EX.

```
Timer2_ISR:
If T2CON.7 = 1 Then
  Print “Timer overflowed”
Else
  If T2CON.6 = 1 Then
     Print “External transition”
  End if
End If
Return
```

INTERRUPTS and PRIORITY 80515

The 80515 and 80535 have more interrupts and priority is handled different compared to the 8051.

Enable interrupts:
ENABLE AD 'AD converter
ENABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
ENABLE TIMER2EX 'timer2 external reload

Disable interrupts:
DISABLE AD 'AD converter
DISABLE INT2|INT3|INT4|INT5|INT6 'external interrupt 2-6
DISABLE TIMER2EX 'timer2 external reload

Selecting of priority:
PRIORITY SET|RESET source , level
level can be 0,1,2 or 3.(0=lowest,3=highest)

The source can be :
INT0/ADC
TIMER0/INT2
INT0/INT3
TIMER1/INT4
SERIAL/INT5
TIMER2/INT6

Note that only one of the pairs must be selected.
PRIORITY SET INT4,3 'will set INT4 to the highest priority.
When two ints occur with the same priority the first source in the list
Will be handled first. So when both TIMER1 and INT4 have the same priority, TIMER1 will be serviced first.
Look at a datasheet for more details.

GETAD

Action

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

Syntax

var = GETAD(channel, range)

Remarks

var	The variable that is assigned with the A/D value
channel	The channel to measure
range	The internal range selection. 0 = 0-5 Volt 192 = 0 - 3.75 Volt 128 = 0 - 2.5 Volt 64 = 0 - 1.25 Volt 12 = 3.75 - 5 Volt 200 = 2.5 - 3.75 Volt 132 = 1.25 - 2.5 Volt

The GETAD() function is only intended for the 80515, 80535,80517 and 80535.
It is a microprocessor dependend support feature.

See also

Example

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

WATCHDOG 80515

The 80515 and 80535 both have a WD-timer.
This is a 16 bit timer that can't be stopped!
It will reset the system after 65535 uS at 12MHz.

```
START WATCHDOG  'start the WD-timer.
RESET WATCHDOG  'will reset the WD-timer.
```

INTERRUPTS and PRIORITY 80537

The 80517 and 80537 have more interrupts and priority is handled different compared to the 8051.

Enable interrupts:

```
ENABLE AD 'AD converter
ENABLE INT2|INT3|INT4|INT5|INT6    'external interrupt 2-6
ENABLE TIMER2EX                    'timer2 external reload
ENABLE CTF                         'compare timer interrupt
ENABLE SERIAL1                     'serial1 interrupt
```

Disable interrupts:

```
DISABLE AD 'AD converter
DISABLE INT2|INT3|INT4|INT5|INT6   'external interrupt 2-6
DISABLE TIMER2EX                   'timer2 external reload
DISABLE CTF                        'compare timer interrupt
DISABLE SERIAL1                    'serial1 interrupt
```

Selecting of priority:

PRIORITY SET|RESET source , level
level can be 0,1,2 or 3.(0=lowest,3=highest)

source can be :

INT0/ADC/SERIAL1
TIMER0/INT2
INT0/INT3
TIMER1/CTF/INT4
SERIAL/INT5
TIMER2/INT6
Note that only one of the TRIPLE-pairs must be selected.

```
PRIORITY SET INT4,3  'will set INT4 to the highest priority.
```

When two ints occur with the same priority the first source in the list will be handled first. So when both TIMER1 and INT4 have the same priority, TIMER1 will be serviced first.
Look at a datasheet for more details.

CONFIG BAUD1

Action

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

Syntax

CONFIG BAUD1 = baudrate

Remarks

baudrate	Baudrate to use : 2048 – 37500

The 80517 and 80537 have 2 serial ports on board.

See also

CONFIG BAUD

Example

```
CONFIG BAUD1 = 9600   'use internal baud generator
Print "Hello"
End
```