

BASCOM LT
Language Reference
V 1.26

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

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

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

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

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

copyright MCS Electronics. All rights reserved.

LANGUAGE FUNDAMENTALS

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

These in turn combine to form statements that make up a program.

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

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

Character Set

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

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

The BASCOM LT numeric characters are the digits 0-9.

The letters can be used as parts of hexadecimal numbers.

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

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

The BASCOM LT program line

BASCOSM LT program lines have the following syntax:

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

Using Line Identifiers

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

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

BASCOM LT keywords are not permitted. The following are valid alphanumeric line labels:

Alpha:
ScreenSUB:
Test3A:

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

alpha:
Alpha:
ALPHA:

Line labels may begin in any column, as long as they are the first characters other than blanks on the line. Blanks are not allowed between an alphabetic label and the colon following it.

A line can have only one label.

BASCOM LT Statements

A BASCOM LT statement is either “executable” or “non-executable”.

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

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

The following BASCOM LT statements are non-executable:

- **REM** or ‘ (starts a comment)
- **DIM**

A “comment” is a non-executable statement used to clarify a program’s operation and purpose.

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

The following lines are equivalent:

```
PRINT “Quantity remaining” : REM Print report label.  
PRINT “Quantity remaining” ‘ Print report label.
```

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

```
FOR I = 1 TO 5 : PRINT “G’day, mate.” : NEXT I
```

BASCOM LT LineLength

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

Data Types

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

Elementary Data Types

- **Bit (1/8 byte)**
- **Integer (two bytes).**
Integers are stored as signed sixteen-bit binary numbers ranging in value from -32,768 to +32,767.
- **Word (two bytes).**
Words are stored as unsigned sixteen-bit binary numbers ranging in value from 0 to 65535.
- **Byte (1 byte).**
Bytes are stored as unsigned 8-bit binary numbers ranging in value from 0 to 255.
- **String (up to 254 bytes).**
Strings are stored as bytes and are terminated with a 0-byte.
A string dimensioned with a length of 10 bytes will occupy 11 bytes of external memory.
- **Long (4 bytes)**
Bytes are stored as signed 32-bit binary numbers ranging in value from -2147483648 to 2147483647

Variables can be stored internal (default) or external.

Variables

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

A numeric variable, can be assigned only a numeric value (either integer, word, byte or bit). The following list shows some examples of variable assignments:

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

Variable Names

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

A variable name cannot be a reserved word, but embedded reserved words are allowed. For example, the following statement is illegal because AND is a reserved word.

```
AND = 8
```

However, the following statement is legal:

```
ToAND = 8
```

Reserved words include all BASCOM LT commands, statements, function names, internal registers and operator names.

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

`a = &HA` , `a = &B1010` and `a = 10` are all the same.

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

```
Dim b1 As Bit, l as Integer, k as Byte , s As String * 10 , L as Long
Dim b2 as Xram Byte
```

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

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

Expressions and Operators

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

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

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

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

Expressions and Operators

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

Operators perform mathematical or logical operations on values.

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

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

Arithmetic

Arithmetic operators are `+`, `-`, `*` and `\`.

- **Integer**
Integer division is denoted by the backslash (`\`).
Example: `PRINT X\Y`
- **Modulo Arithmetic**
Modulo arithmetic is denoted by the modulus operator `MOD`.
Modulo arithmetic provides the remainder, rather than the quotient, of an integer division.
Example: `X = 10 \ 4 : remainder = 10 MOD 4`
- **Overflow and division by zero**
Division by zero, produces an error.
At this moment there is no message, so you have to insure yourself that such won't happen.

Relational Operators

Relational operators are used to compare two values as shown in the table below. The result can be used to make a decision regarding program flow.

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

Logical Operators

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

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

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

Example

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

Output

```
16
11
```

Compiler Limits

There are some limitations to the compiler.
 You can perform only **one** calculation in a formula.

Good

$a = a * b1$

$a = \text{Asc}(s) + \text{Len}(s)$

False

$a = a * b1 + c$

Limit	Number
Maximum allowed labels	500
Maximum allowed variable names	500
Maximum number of INTEGER/WORD variables	10*
Maximum number of BYTE variables	20*
Maximum number of BIT variables	126*
Maximum number of STRING variables	up to available external memory
Maximum number of ALIAS statements	128

*Depending on the used statements and the used variables of the other types.

The AT89C2051 has 128 bytes of internal RAM.

A maximum of 32 bytes are used internally for the registers, the PRINT & INPUT routines and the SETDATA & GETDATA routines.

The rest can be used by your program.

The stack uses some space too. So it depends on the used statements how much variables you can use.

8 used bit variables will use 1 byte;

1 used byte will use 1 byte;

1 used integer/word will use 2 bytes;

1 used long will use 4 bytes;

1 string with a length of 10 bytes will use 11 bytes of memory.

Maximum nesting :

Operation	Max
FOR .. NEXT	50
IF .. THEN	50
DO .. LOOP	50
WHILE .. WEND	50
SELECT CASE	25

1WRESET,1WREAD,1WRITE

Action

These routines can be used to communicate with Dallas Semiconductor's 1Wire-devices.

Syntax

1WRESET
1WRITE var1
var2 = 1WREAD()

Remarks

1WRESET	Reset the 1WIRE bus.The errorvariable ERR will return 1 if a fault occurs.
1WRITE var1	Sends the value of var1 to the bus.
Var2 = 1WREAD()	Reads a byte from the bus and places it into var2.

Example

```
-----  
'                                     1WIRE.BAS  
' demonstrates lwire, lwrite and lread()  
' pullup of 4K7 required to VCC from P.1  
' DS2401 serial button connected to P1.1  
-----  
Config lwire = P1.1           'use this pin  
Dim Ar(8) As Byte , A As Byte , I As Byte  
  
lwreset                       'reset the bus  
Print Err                      'print error 1 if error  
lwrite &H33                   'read ROM command  
For I = 1 To 8  
    A = lread()                'read byte  
    Ar(i) = A                  'place into array  
Next  
For I = 1 To 8  
    A = Ar(i) : Printhex A;     'print output  
Next  
Print                          'linefeed  
End
```

\$INCLUDE

Action

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

Syntax

\$INCLUDE file

Remarks

file	ASCII file which must contain valid BASCOM LT 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 LT programs. You can only include ASCII files!
------	---

Example

```

$INCLUDE test.bas           'include this file at
                             'this position ( ASCII )
$INCLUDE myasm.ASM         'include assembler file
    
```

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

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
```

\$BAUD

Action

Instruct the compiler that you want to use a different baud rate.

Syntax

\$BAUD = var

Remarks

var	The baud rate that you want to use.
-----	-------------------------------------

When you want to use an unsupported crystal/ baud rate you can use this meta command. You must also use the **\$CRYSTAL** meta command. These statements always work together.

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

See also

[\\$CRYSTAL](#)

Example

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

\$CRYSTAL

Action

Instruct the compiler which crystal frequency to use.

Syntax

\$CRYSTAL = var

Remarks

var	Frequency of the crystal.
-----	---------------------------

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

See also

[\\$BAUD](#)

Example

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

\$LARGE

Action

Instructs the compiler that LCALL statements must be used.

Syntax

\$LARGE

Remarks

Internally when a subroutine is called the ACALL statement is used.

The ACALL instruction needs only 2 bytes(the LCALL needs 3 bytes)

The ACALL statement however can only address routines with a maximal offset of 2048. AT89C2051 chips will have no problems with that.

When code is generated for an other 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 databus.

Syntax

\$LCD = [&H]address

Remarks

address	<p>The address where must be written to, to enable the LCD-display. The db0-db7 lines of the LCD must be connected to the datalines D0-D7. The RS line of the LCD must be connected to the address line A0.</p> <p>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.</p>
---------	--

See also

CONFIG LCDBUS

Example

```
CONFIG LCDBUS = 8 'use 8-bit parallel databus mode(faster)
CONFIG LCDBUS = 4 'or use only d7-d4 (less wires)
$LCD = &HA000    'writing to this address will make the E-line of the
                  'LCD high.
```

\$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 initialization must be performed.

Syntax

\$NOINIT

Remarks

BASCOM LT initializes the processor depending on the used statements.

When you want to handle this by yourself you can specify this with the meta statement **\$NOINIT**.

The only initialization that is always done is the setting of the stack pointer and the initialization of the LCD-display (if statements are used).

See also

Example

```
$NONIT
```

```
.....  
.....
```

```
End
```

\$NOSP

Action

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

Syntax

\$NOSP

Remarks

BASCOM LT initializes the processor depending on the used statements.

When you want to handle this by yourself you can specify this with the meta command **\$NOINIT**.

The only initialization that is always done is the setting of the stack pointer and the initialization of the LCD-display (if LCD statements are used).

With the **\$NOSP** meta command the stack will not be initialized either.

See also

\$NOINIT

Example

```
$NOSP  
$NOINIT  
End
```

\$RAMSTART

Action

Specifies the location of the external RAM-memory.

Syntax

\$RAMSTART = [&H]address

Remarks

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

Example

```
$ROMSTART = 4000  
$RAMSTART = 0  
$RAMSIZE = &H1000
```

\$RANGECHECK

Action

Instruct the compiler to perform a rangecheck when variables are assigned.

Syntax

\$RANGECHECK

Remarks

When a variable is assigned to a variable of another type, there is a possibility that the variable doesn't fit into the new variable.

You can for example assign an integer to a byte but the integer must be in the byte range (0-255).

When you don't specify \$rangecheck, no check will be performed to see if the integer fits in the byte.

When you do specify \$rangecheck, an additional check will be performed. When the variable doesn't fit into the new variable, the ERR variable will be set to 1.

Using \$rangecheck will produce more code.

See also

-

Example

```
$rangecheck
CDim a As Byte, I as Integer
I = 1      'assign var
a = I      'ok for bytes, ERR will be 0
I = -1     'invalid for bytes
a = I      'will set ERR to 1
I = 256    'invalid for bytes
a = I      'will set ERR to 1
End
```

\$RAMSIZE

Action

Specifies the size of the external RAM-memory.

Syntax

\$RAMSIZE = [&H] size

Remarks

size	Size of external RAM memory chip.
------	-----------------------------------

Example

```
$ROMSTART = 4000
$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	<p>The (hex)-address where the code must start. Default is 0. This value will be used when \$ROMSTART is not specified.</p> <p>You can use this option when you want to test the code in RAM. The code must be downloaded at 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 = 4000 is specified the monitor program must perform a LJMP instruction. For address 3 this must be 4003. Otherwise interrupts can not be handled correctly. But that is up to the monitor program.</p> <p>Note that a file named xxx.INF will be generated for the simulator to work correctly. This file has the relocated address in it.</p>
---------	---

Example

```
$ROMSTART = 4000
```

\$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 an 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.

Example

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

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

\$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

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 loops  
WAIT 2        'the simulator
```

ABS()

Action

Returns the absolute value of a numeric variable.

Syntax

var = ABS(var2)

Remarks

var	Integer/Word or Long variable that is assigned the absolute value of var2.
Var2	Integer or Long variable.

The absolute value of a number is always positive.

See also

-

Example

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

ASC ()

Action

Convert a string into its ASCII value.

Syntax

var = ASC(stringr)

Remarks

var	Byte,Integer/Word variable.
Var	String variable or 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 XRAM String * 10
s = "ABC"
a = Asc(s)
Print a
End
```

BCD ()

Action

Convert a byte, integer/word variable or a constant to its BCD value.

Syntax

PRINT BCD(var)

LCD BCD(var)

Remarks

var	Byte, Integer/Word variable or numeric 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.

See also

-

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, internal register or P1.x or P3.x, where x ranges from 0-7.
---	---

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

When you use internal registers that can be set/reset by hardware such as P1.0 this doesn't apply.

See also

-

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
```

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 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.

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 LT variable or constant..
var-n	Any BASCOM LT variable or constant.
Test	Name of the subroutine. In this case Test

With the CALL statement you can call a procedure or sub routine.
As much as 10 parameters can be passed but you can also call a subroutine without parameters.
for examples : **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 without call statement
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

-

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

-

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

```
DIM b1 as byte
DIM a AS CONST 5      'assign 5 to symbolic name a
Print a
a = a + 1              'THIS WILL RESULT In AN ERROR
b1 = a + 1            'this is ok
```

CONFIG TIMER0, TIMER1

Action

Configure TIMER0 or TIMER1.

Syntax

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

CONFIG LCD = LCD type

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.
LCD type	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

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 microprocessor support for other statements 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

Configure the LCD-databus width.

Syntax

CONFIG LCDBUS = constant

Remarks

constant	8 for 8-bit mode(default) Or use 4 for 4-bit databus mode. 4-bit mode only uses d7-d4.
----------	---

This statement works together with the \$LCD statement.

Example

```

$LCD = &H6000      'parallel bus mode
CONFIG LCDBUS = 4 '4-bit mode
CONFIG LCD = 40 * 4
LCD "Hello"       'display on LCD
FOURTHLINE       'select line 4
LCD "4"           'display 4
END
    
```

CONFIG BAUD

Action

Configure the uP to select the intern baudrate generator.
This baudrate 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 delaytime in mS.
------	---

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

See also

DEBOUNCE

Example

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

CONFIG SPI

Action

Configure the pins to use for SPI statements.

Syntax

CONFIG SPI = soft , DIN = Pin, DOUT = Pin, CS = Pin, CLK = Pin

Remarks

pin	The port pin to use such as P1.0
soft	Some uP's support hardware SPI. At the moment there is only support for software SPI so you must specify soft.

See also

SPIINIT , SPIOUT

Example

```
Dim Ar(4) As Byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SpiInit      'bring pins to good logic state
SPIOUT ar(1) , 4 'send 4 bytes
End
```

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) 1999 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

```


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) 1999 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  
  Prinhex 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

-

Example

```
Dim a as byte
a = 255
LCD a
CURSOR OFF          'hide cursor
Wait 1
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.
-----	-----------------------------

Integer and Word values must end with the % sign.

A Long constant must end with the &-sign.

See also

Example

```
DIM a AS BYTE, I AS BYTE, S As String * 15, L as Long
RESTORE Dta1
FOR a = 1 TO 3
  READ a : PRINT a
NEXT
RESTORE Dta2
READ I : PRINT I
READ I : PRINT I
RESTORE Dta4
READ L : PRINT L
Restore Dta3: Read S: Print s; : Read S: Print S
END
```

```
DTA1:
DATA 5, 10, 100
```

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

```
DTA3:
DATA "Hello" , "Word"
```

```
DTA4:
DATA 12345678& 'A Long must end with the &-sign.
```

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.

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 an other branch.

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

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
  '                ^----- 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	An integer/Word or Byte variable.
-----	-----------------------------------

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

See also

[INCR](#)

Example

```
DO                                'start loop
    DECR a                        'decrement a by 1
    PRINT a                       'print a
LOOP UNTIL a > 10                 'repeat until a is greater than 10
```

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 or Integer.

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
```

Example

```
Defbyte 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	Byte, Integer/Word variable or constant representing the character (0-7).
r1-r8	The row values for the character.

You can use the LCD designer to build the characters.

See also

Edit LCD designer

Example

```
DefLCDchar 0,1,2,3,4,5,6,7,8 'define special character
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

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**] type

Remarks

var	Any valid variable name such as b1, i or longname.
Type	Bit, Byte, Word/Integer or String
XRAM	Specify XRAM to store variable in external memory

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.

See Also

[CONST](#)

Example

```

DIM b1 As Bit, c As Integer      'dimension a bit and an integer
DIM s As XRAM String * 10      'length of string is 10
s = "Hello"
Set b1                          'set bit to 1
c = 10                          'let c = 10

DIM b2 As XRAM Byte            'use external memory too
    
```

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.

See also

[ENABLE](#)

Example

ENABLE INTERRUPTS	'enable the setting of interrupts
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           'assign variable
LCD a            'show on LCD
DISPLAY OFF      'turn display off
Wait 1           'wait 1 second
DISPLAY ON       'turn in back 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

```
DO
  A = A + 1
  PRINT A
LOOP UNTIL A = 10
End
```

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

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 .. 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

Example

```
A = 10
IF A > 10 THEN
    PRINT " A >10"
ELSE
    PRINT "A not greater than 10"
END IF
```

'let a = 10
'make a decision
'this will not be printed
'alternative
'this will be printed

ENABLE

Action

Enable specified interrupt.

Syntax

ENABLE interrupt

Remarks

Interrupt	INT0, INT1, SERIAL, TIMER0, TIMER1 or TIMER2.
-----------	--

By default all interrupts are disabled.

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

See also

[DISABLE](#)

Example

```
ENABLE INTERRUPTS
```

```
ENABLE TIMER1
```

'enables the TIMER1 interrupt

END

Action

Terminate program execution.

Syntax

END

Remarks

STOP can also be used to terminate a program.

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

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.

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. (yet)

Example

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

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 must be increased with each time NEXT is encountered.

For incremental loops you must use TO.
 For decremental loops you must use DOWNTO.
 You must end a FOR structure with the NEXT statement.
 The use of STEP is optional. By default 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 x           'next x (count down)
NEXT a              'next a (count up)
END
    
```

FOURTHLINE

Action

Reset LCD-cursor to the fourth line.

Syntax

FOURTHLINE

Remarks

Only valid for LCD-displays with 4 lines.

See also

-

Example

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

GETAD()

Action

Retrieves the analog value from channel 0-7.

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 8 = 0-2.5 Volt 128 = 2.5-5 Volt 4 = 0-1.25 Volt 72 = 1.25-2.5 Volt 140 = 2.5-3.75 Volt 12 = 3.75-5 Volt

Works only for the 80515 and compatibles.

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
```

GETDATA()

Action

Retrieve value from buffer.

Syntax

`var = GETDATA(x)`

Remarks

<code>var</code>	Receives the value from the buffer.
<code>x</code>	Place in the buffer where to retrieve the value from (1- 8).

The compiler has an internal buffer of 8 bytes which you can use for general purpose. For example you can fill the buffer and send the contents to an I2C device.

See also

[SETDATA](#)

Example

```
Dim a as byte
Setdata 1,12
a = Getdata(1)
Print a
End
```

GOSUB

Action

Branch to and execute subroutine.

Syntax

GOSUB label

Remarks

label	The name of the label where to branch to.
-------	---

The routine must end with the RETURN statement.

See also

[GOTO](#)

Example

<pre>GOSUB Routine END Routine: x = x + 2 PRINT X RETURN</pre>	<pre>'branch to routine 'terminate program 'this is a subroutine 'perform some math 'print result 'return</pre>
--	---

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 semicolon
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's it
```

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.

See also

VAL

Example

```
Dim a as Integer, s as XRAM String * 15  
s = "000A"  
a = Hexval(s) : Print a  
End
```

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

-

Example

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

I2CRECEIVE

Action

Receives data from an I2C serial device.

Syntax

I2CRECEIVE slave, var

I2CRECEIVE slave, DATA ,b2W, b2R

Remarks

slave	A byte, Word/Integer variable or number with the slave address from the I2C-device.
Var	A byte or integer/word variable that will receive the information from the I2C-device Use DATA to specify the DATA-buffer. When you use DATA then b2W and b2R must be specified.
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!

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

SETDATA 1,64                          'fill data space
SETDATA 1,65
I2CRECEIVE slave, DATA, 2, 1          'send two bytes and receive one byte
x = GETDATA(1)                         'print the received byte
    
```

I2CSEND

Action

Send data to an I2C-device.

Syntax

I2CSEND slave, var

I2CSEND slave, DATA , bytes

Remarks

slave	The slave address off the I2C-device.
Var	A byte, integer/word or number that holds the value which will be send to the I2C-device. Specify DATA to specify that the value(s) will come from the DATA-space. If you specify DATA then bytes must be specified.
Bytes	The number of bytes to send. Be cautious not to specify too many bytes! - Byte variable : bytes = 1 - Integer/word variable : bytes = 1 - DATA : 1-8

This command works only with additional hardware. See [appendix D](#).

See also

[I2CRECEIVE](#)

Example

```
x = 5                                'assign variable to 5
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 8
    x = Getdata(a)                     'get variable from DATA-space
    I2CSEND slave,x
Next

or

For a = 1 to 8
    Setdata(a,a )                     'Fill dataspace
Next
bytes = 8
I2CSEND slave,DATA,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 byte or integer/word 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 byte, integer/word 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](#)

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 a 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**

Remarks

expression	Any expression that evaluates to true or false.
------------	---

See also

[ELSE](#) , [END IF](#)

Example

```
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
```

INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

var	An integer/word or byte variable.
-----	-----------------------------------

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

See also

[DECR](#)

Example

```
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 or integer/word 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.

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.

Syntax

var = **INP**(address)

Remarks

var	Byte or integer/word 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

Reads a binary stream of data from the serial port.

Syntax

INPUTBIN , var [, varn]

Remarks

Var,varn	A variable that will be assigned the received data. The number of characters to receive depends on the variable type. A byte can store 1 character so when you specify a byte variable, program execution will be continued when 1 byte is received. You can also use a string or an array.
----------	---

See also

PRINTBIN

Example

```

'-----
'                               (c) 1997-1999 MCS Electronics
'-----
'To use another baudrate and crystal frequency 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 Ar(4) as Byte

Inputbin V , B1 , Ar(1)                      '2 + 4 characters to receive
Printbin Ar(1)                                'print it
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 COM-port.

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

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

See also

INPUT

Example

```
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 COM-port.

The INPUT routine can be used when you have a RS-232 interface on your uP.

See the manual for a design of a RS-232 interface.

The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as an input device.

You can also use the build in terminal emulator.

See also

[INPUTHEX](#)

Example

```
INPUT "Enter a number" , v      'ask for input
PRINT v                        'print it
Dim s as XRAM string * 20
Input "Your name " , s
Print "Hello " ; 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"

See also

[LCDHEX](#)

Example

```
Dim a as byte
a = 255
LCD "A = " ; a
End
```

LCDHEX

Action

Send variable in hexadecimal format to the LCD-display.

Syntax

LCDHEX var

Remarks

var	Byte or Integer/Word variable.
-----	--------------------------------

The same rules apply as PRINTHEX.

See also

[LCD](#)

Example

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

LEFT()

Action

Return a specified number of leftmost characters in a string.

Syntax

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

Remarks

var	The string that is assigned.
Var1	The sourcestring.
l	The number of characters to get from the sourcestring.

Example

```
Dim s as XRAM String * 15, z as XRAM String * 15
s = "ABCDEFGH"
z = Left(s,5)
Print z           'ABCDE
End
```

LOAD

Action

Load specified TIMER with a value.

Syntax

LOAD TIMER , value

Remarks

TIMER	TIMER0, or TIMER1.
<i>Value</i>	The variable or value to load.

When you use the ON TIMERx statement with the TIMER in mode 2, you can specify on which interval the interrupt must occur. The value can range from 0 to 255 for TIMER0 and TIMER1.

See [Additional hardware](#) for more details

Example

LOAD TIMER0, 100 ' load TIMER0 with 100 (256-100)=156 will be stored in order to reload after 100 counts.

LOCATE

Action

Moves the LCD cursor to the specified position.

Syntax

LOCATE *y* , *x*

Remarks

<i>x</i>	Constant or variable with the position. (1-64*)
<i>y</i>	Constant or variable with the line (1 - 4*)

* depending on the used display

See also

CONFIG

Example

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

LOOKUP()

Action

Returns a byte from a table.

Syntax

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

Remarks

var	The byte returned
value	A byte value with the index of the table
label	The label where the data starts

See also

LOOKUPSTR

Example

```
DIM b1 As Byte
b1 = Lookup(1, dta)
Print b1           ' Prints 2 (zero based)
End
```

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

LOOKUPSTR()

Action

Returns a string from a table.

Syntax

var =LOOKUPSTR(value, label)

Remarks

var	The string returned
value	A byte value with the index of the table
label	The label where the data starts

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"
```

LOWERLINE

Action

Reset the LCD-cursor to the lowerline.

Syntax

LOWERLINE

Remarks

-

See also

-

Example

```
Dim a as byte
LCD "Test"
LOWERLINE
LCD "Hello"
End
```

MAKEBCD()

Action

Convert a decimal byte or Integer/Word variable to it's BCD value.

Syntax

```
var1 = MAKEBCD(var2)
```

Remarks

var1	Byte or Integer/Word variable that will receive the converted value.
var2	Byte or Integer/Word variable or numeric constant that holds the decimal value.

When you want to use an I2C clock device which stores it's values as BCD values you can use this function to convert variables from decimal to BCD.

See also

[MAKEDEC](#)

Example

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

MAKEDEC()

Action

Convert a BCD byte or Integer/Word variable to it's DECIMAL value.

Syntax

```
var1 = MAKEDEC(var2)
```

Remarks

var1	Byte or Integer/Word variable that will receive the converted value.
var2	Byte or Integer/Word variable or numeric constant that holds the BCD value.

When you want to use an I2C clock device which stores it's 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
```

MAKEINT

Action

Convert 2 bytes 2 a Word/Integer.

Syntax

`var1 = MAKEINT(var2, var3)`

Remarks

var1	Integer/Word variable that will receive the converted value.
var2	A byte or numeric constant that holds the LSB.
var3	A byte or numeric constant that holds the MSB

See also

MAKEBCD , MAKEDEC

Example

```
Dim aL As Byte, aH as Byte, I as Integer
aL = 2 : aH = 1
I = MakeInt(aL, aH)
Print I '258
End
```

MID()

Action

The MID function returns part of a string (a substring).

The MID statement replaces part of a string variable with another string.

Syntax

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

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

Remarks

<i>var</i>	The string that is assigned.
<i>Var1</i>	The sourcestring.
<i>st</i>	The starting position.
<i>l</i>	The number of characters to get/set.

Example

```
Dim s as XRAM String * 15, z as XRAM String * 15
s = "ABCDEFGH"
z = Mid(s,2,3)
Print z           `BCD
z="12345"
Mid(s,2,2) = z
Print s           `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

<i>var</i>	The index variable that is used as a counter when you form the structure with FOR <i>var</i> .
------------	--

You must end each FOR statement with a NEXT statement.

See also

[FOR](#)

Example

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

ON INTERRUPT

Action

Execute subroutine when specified interrupt occurs.

Syntax

ON interrupt label

Remarks

interrupt	INT0, INT1, SERIAL, TIMER0 ,TIMER1 or TIMER2.
label	The label to jump to if the interrupt occurs.

You must return from the interrupt routine with the RETURN statement.
 You cannot use TIMER1 when you are using SERIAL routines such as PRINT because TIMER1 is used as a BAUDRATE generator.

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

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       '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 variable to test. (Byte or integer/word). 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 first value starts at 0. So when var = 0 the first specified label is jumped/branched.

Example

```
x = 2                                'assign a variable interrupt
ON x GOSUB lb11, lb12,lb13           'jump to label 'lb13'
END
lb13:
    PRINT "lb13"
RETURN
```

OUT

Action

Sends a byte to a hardware port.

Syntax

OUT address, value

Remarks

address	The address where to send the byte to.
value	Byte, Integer/Word or constant 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
```

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 on other uP's other ports can be available such as P0 and P2.

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 at address
address	Numeric variable or constant with the address location.(0-255)

See also

POKE , CPEEK

Example

```
DIM a As Byte
a = Peek( 0 )      'return the first byte of the internal memory (r0)
End
```

POKE

Action

Sets a byte stored in internal memory.

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 registers with it which can cause your program to function incorrect.

See also

PEEK

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 microcontroller 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](#)

Example

```
PRINT x                                    'print the var
Print Chr(x) ; Bcd(x)                    'use conversion routines
```

PRINTBIN

Action

Send the binary value of a variable to the RS-232 port.

Syntax

PRINTBIN var [; varn]

Remarks

var,varn	The variable to print.
----------	------------------------

You can use a semicolon (;) to print more variables.

See also

INPUTBIN

Example

```

'-----
'                               (c) 1997-1999 MCS Electronics
'-----
Dim A As Byte , Ar(4) As Byte
A = 1 : Ar(1) = 1
Printbin a ; ar(1)           'send 1 + 4 characters
End

```

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](#)

Example

```
INPUT x           'ask for var
PRINT x           'print it in decimal format
PRINTHEX 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.

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.
-----	---------------------------------------

See also

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 BASCOM LT statements after a REM statement.

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 a bit/byte variable.

Syntax

RESET bit

RESET byte.x

Remarks

bit	Can be a P1.x, P3.x or any bitvariable where x=0-7.
byte	Can be any byte variable.
x	Bit of variable (0-7) to reset.

You can also use the alternative syntax **CLR**.

See also

[SET](#)

Example

```
Dim b1 as bit, b2 as byte
RESET P1.3           'reset bit 3 of port 1
RESET b1             'bitvariable
RESET b2.0           'rest bit 0 of bytevariable b2
CLR b2.0             'use alternative syntax
```

RESTORE

Action

Allows READ to reread values in specified DATA statements.

Syntax

RESTORE label

Remarks

label	The label of a DATA statement.
-------	--------------------------------

See also

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

You must end your subroutines with RETURN.

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

RIGHT()

Action

Return a specified number of rightmost characters in a string.

Syntax

`var = RIGHT(var1 ,st)`

Remarks

<code>var</code>	The string that is assigned.
<code>Var1</code>	The sourcestring.
<code>st</code>	The number of characters to get.

Example

```
Dim s as XRAM String * 15, z as XRAM String * 15
s = "ABCDEFGH"
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**

Remarks

<i>var</i>	Byte , Integer/Word or Long variable.
------------	---------------------------------------

See also

-

Example

```
Dim a as byte
a = 255
ROTATE a, LEFT
Print a
End
```

SELECT

Action

Executes one of several statement blocks depending on the value of an expression.

Syntax

```
SELECT CASE var  
  CASE test1 : statements  
  CASE test2 : statements  
  CASE ELSE : statements  
END SELECT
```

Remarks

var	Numeric or string variable.
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 variable.

Syntax

SET bit

SET byte.x

Remarks

bit	P1.x, P3.x or a Bitvariable.
byte	Can be any byte variable.
x	Bit of variable (0-7) to set.

You can also use the alternative syntax **SETB**.

See also

RESET

Example

```
Dim b1 as Bit, b2 as byte
SET P1.1      'set bit 1 of port 1
SET b1        'bitvariable
SET b2.1      'set bit 1 of var b2
SETB b2.1     'use alternative syntax
```

SETDATA

Action

Place byte in databuffer.

Syntax

SETDATA *x* , *val*

Remarks

<i>x</i>	Position in buffer (1-8).
<i>val</i>	Value to place in buffer. Can be constant a or variable.

See also

-

Example

```
Dim a as byte
a = 255
Setdata 1,a
End
```

SHIFTCURSOR

Action

Shift the cursor of the LCD-display left or right by one position.

Syntax

SHIFTCURSOR LEFT / RIGHT

Remarks

-

See also

-

Example

```
LCD "Hello"  
SHIFTCURSOR LEFT  
End
```

SHIFTLCD

Action

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

Syntax

SHIFTLCD LEFT / RIGHT

Remarks

-

See also

-

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 lower for *frequency* μ S.
This loop is executed *duration* times.

See also

-

Example

```
SOUND P1.1 , 10000 , 10      'BEEP  
End
```

SPACE ()

Action

Returns a string of spaces.

Syntax

var = **SPACE**(x)

Remarks

x	The number of spaces.
Var	The string that is assigned.

Example

```
Dim s as XRAM String * 15, z as XRAM String * 15  
s = Space(5)
```

```
Print "{ " ; s ; " }" ' { }
```

SPIIN

Action

Receives data from the SPI bus.

Syntax

SPIIN var , bytes

Remarks

var	The variable which will receive the data.
Bytes	Number of bytes to receive.

See also

SPIINIT , SPIOOUT

Example

```
Dim Ar(4) As Byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SpiInit          'bring pins to good logic state
SPIOOUT ar(1) , 4 'send 4 bytes
SPINI ar(1) , 2  'receive 2 bytes
End
```

SPIINIT

Action

Initializes the SPI pins.

Syntax

SPIINIT

Remarks

The SPI related pins must be in the correct logic state before SPIOUT can be used. Normally you only have to use this statement once, but when you use the pins between two sessions, you must initialize them again.

See also

SPIOUT

Example

```
Dim Ar(4) As Byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SpiInit          'bring pins to good logic state
SPIOUT ar(1) , 4 'send 4 bytes
```

SPIOUT

Action

Sends data to the SPI bus.

Syntax

SPIOUT data , bytes

Remarks

data	Variable which holds the data to send.
bytes	Number of bytes to send.

See also

SPIINIT

Example

```
Dim Ar(4) As Byte
CONFIG SPI = SOFT, DIN = P1.0, DOUT = P1.1, CS=P1.2, CLK = P1.3
SpiInit          'bring pins to good logic state
SPIOUT ar(1) , 4 'send 4 bytes
End
```

START

Action

Start the specified timer/counter.

Syntax

START *timer*

Remarks

<i>timer</i>	TIMER0, TIMER1, COUNTER0 or COUNTER1.
--------------	---------------------------------------

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

TIMER0 and COUNTER0 are the same.

See also

STOP *TIMERx*

Example

```
ON TIMER0 label2
ENABLE INTERRUPTS
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 all interrupts are disabled and a never ending loop is generated.

Example

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

STOP TIMER

Action

Stop the specified timer/counter.

Syntax

STOP *timer*

Remarks

<i>timer</i>	TIMER0, TIMER1, 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

```
ON TIMER0 label2
LOAD TIMER0, 100
START TIMER0
DO                                'start loop
  IF INKEY = 27 then              'escape
    STOP TIMER0
  END IF
  IF INKEY = 32 then              'space
    START TIMER0
  END IF
LOOP                              'loop forever

label2:  'perform an action here

RETURN
```

STR()

Action

Returns a string representation of a number.

Syntax

var = **Str**(x)

Remarks

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

See also

VAL

Example

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

STRING()

Action

Returns a string of a specified length made up of a repeating character.

Syntax

var = **STRING**(char ,x)

Remarks

var	The string that is assigned.
char	The character that is assigned to the string.
x	The number of characters to assign.

Example

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

SUB

Action

Defines a Sub procedure.

Syntax

SUB Name[(var1)]

Remarks

name	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.

As a result 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

-

THIRDLINE

Action

Reset LCD-cursor to the third line.

Syntax

THIRDLINE

Remarks

-

See also

-

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

-

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	Any numeric variable such as byte, integer or word.
s	Variable of the string type.

See also

[STR](#)

Example

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

SWAP

Action

Exchange two variables of the same type.

Syntax

SWAP var1, var2

Remarks

var1	Any BASCOM LT variable.
var2	Any BASCOM LT variable.

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
```

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

WAITKEY variable

Remarks

variable	The variable that receives the ASCII value of the character.
----------	--

See also

[INKEY](#)

Example

```
DIM a as Byte
WAITKEY a      'wait for a key press
Print a
End
```

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.

Also the use of interrupts can slow down this routine.

This statement is provided for the I2C statements.

When you write to an EEPROM you must wait for 10 mS after the write instruction.

See also

DELAY WAIT

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 LT 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
```

ASSEMBLER COMMANDS

Assembler statements are recognized by the compiler.

The only exceptions are CLR, SETB, SWAP, INC and DEC because these are valid BASIC statements. You must precede these ASM-statements with the !-sign in BASCOM LT so the compiler knows that they are ASM-statements.

You can also include an assembler file with the **\$INCLUDE FILE.ASM** statements.

The build in assembler is based on the standard Intel mnemonics.

The following codes are used to describe the mnemonics:

Rn	working register R0-R7
direct	128 internal RAM locations, any IO port, control or status register. For example : P1, P3, ACC
@Ri	indirect internal RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data16	16-bit constant included in instruction
bit	128 software flags, any IO pin, control or status bit For example : ACC.0, P1.0, P1.1

Boolean variable manipulation

CLR C	clear carry flag
CLR bit	clear direct bit
SETB C	set carry flag
SETB bit	set direct bit
CPL C	complement carry flag
CPL bit	complement direct bit
ANL C, bit	AND direct bit to carry flag
ORL C, bit	OR direct bit to carry flag
MOV C, bit	Move direct bit to carry flag

Program and machine control

LCALL addr16	long subroutine call
RET	return from subroutine
RETI	return from interrupt
LJMP addr16	long jump
SJMP rel	short jump (relative address)
JMP @A+DPTR	jump indirect relative to the DPTR
JZ rel	jump if accu is zero
JNZ rel	jump if accu is not zero
JC rel	jump if carry flag is set
JNC rel	jump if carry flag is not set
JB bit, rel	jump if direct bit is set
JNB bit, rel	jump if direct bit is not set
JBC bit, rel	jump if direct bit is set & clear bit
CJNE A, direct, rel	compare direct to A & jump if not equal
CJNE A, #data, rel	comp. Immed. to A & jump if not equal
CJNE Rn, #data, rel	comp. Immed. to reg. & jump if not equal
CJNE @Ri, #data, rel	comp. Immed. to ind. & jump if not equal
DJNZ Rn, rel	decrement register & jump if not zero
DJNZ direct, rel	decrement direct & jump if not zero
NOP	no operation

Arithmetic operations	
ADD A,Rn	add register to accu
ADD A,direct	add register byte to accu
ADD A,@Ri	add indirect RAM to accu
ADD A,#data	add immediate data to accu
ADDC A,Rn	add register to accu with carry
ADDC A,direct	add direct byte to accu with carry flag
ADDC A,@Ri	add indirect RAM to accu with carry flag
ADDC A,#data	add immediate data to accu with carry flag
SUBB A,Rn	subtract register from A with borrow
SUBB A,direct	subtract direct byte from A with borrow
SUBB A,@Ri	subtract indirect RAM from A with borrow
SUBB A,#data	subtract immediate data from A with borrow
INC A	increment accumulator
INC Rn	increment register
INC direct	increment direct byte
INC@Ri	increment indirect RAM
DEC A	decrement accumulator
DEC Rn	decrement register
DEC direct	decrement direct byte
DEC@Ri	decrement indirect RAM
INC DPTR	increment datapointer
MUL AB	multiply A & B
DIV AB	divide A by B
DA A	decimal adjust accu

Logical operations	
ANL A,Rn	AND register to accu
ANL A,direct	AND direct byte to accu
ANL A,@Ri	AND indirect RAM to accu
ANL A,#data	AND immediate data to accu
ANL direct,A	AND accu to direct byte
ANL direct,#data	AND immediate data to direct byte
ORL A,Rn	OR register to accu
ORL A,direct	OR direct byte to accu
ORL A,@Ri	OR indirect RAM to accu
ORL A,#data	OR immediate data to accu
ORL direct,A	ORL accu to direct byte
ORL direct,#data	ORL immediate data to direct byte
XRL A,Rn	exclusive OR register to accu
XRL A,direct	exclusive OR direct byte to accu
XRL A,@Ri	exclusive OR indirect RAM to accu
XRL A,#data	exclusive OR immediate data to accu
XRL direct,A	exclusive OR accu to direct byte
XRL direct,#data	exclusive OR immediate data to direct byte
CLR A	clear accu
CPL A	complement accu
RL A	rotate accu left
RLC A	rotate A left through the carry flag
RR A	rotate accu right
RRC A	rotate accu right through the carry flag
SWAP A	swap nibbles within the accu

Data transfer	
MOV A,Rn	move register to accu
MOV A,direct	move direct byte to accu
MOV A,@Ri	move indirect RAM to accu
MOV A,#data	move immediate data to accu
MOV Rn,A	move accu to register
MOV Rn,direct	move direct byte to register
MOV Rn,#data	move immediate data to register
MOV direct,A	move accu to direct byte
MOV direct,Rn	move register to direct byte
MOV direct,direct	move direct byte to direct
MOV direct,@Ri	move indirect RAM to direct byte
MOV direct,#data	move immediate data to direct byte
MOV@Ri,A	move accu to indirect RAM
MOV@Ri,direct	move direct byte to indirect RAM
MOV@Ri,#data	move immediate to indirect RAM
MOV DPTR,#data16	load datapointer with a 16-bit constant
MOVC A,@A+DPTR	move code byte relative to DPTR to A
MOVC A,@A+PC	move code byte relative to PC to A
MOVX A,@Ri	move external RAM (8-bit) to A
MOVX A,@DPTR	move external; RAM (16 bit) to A
MOVX@Ri,A	move A to external RAM (8-bit)
MOVX@DPTR,A	move A to external RAM (16-bit)
PUSH direct	push direct byte onto stack
POP direct	pop direct byte from stack
XCH A,Rn	exchange register with accu
XCH A,direct	exchange direct byte with accu
XCH A,@Ri	exchange indirect RAM with A
XCHD A,@Ri	exchange low-order digit ind. RAM w. A

How to access labels from ASM.

Each label in BASCOM is changed into a period followed by the label name.

Example :

GOTO Test

Test:

generated ASM code:

LJMP .Test

.Test:

How variables are stored.

BIT variables are stored in bytes. These bytes are stored from 20^{hex} - $2F^{\text{hex}}$ thus allowing $16 * 8 = 128$ bit variables. You can access a bit variable as follows:

```
Dim var As Bit      'dim variable
SETB {var}          ; set bit
CLR {var}           ; clear bit
Print var           ; print value
End
```

BYTE variables are stored after the BIT variables.
Starting at address 20^{hex} + (used bytes for bit vars).

INTEGER/WORD variables are stored with the LSB at the lowest memory position.
Long variables are stored with the MSB at the lowest memory position.

You can access BYTE and INTEGER/WORD variables by surrounding the variable with `{}`.

To refer to the MSB of an Integer/Word use **var+1**.

To refer to the MSB of a Long use **var** (for LSB use **var+3**)

The following example shows how to access the variables from ASM

```
Dim t as Byte, c as Integer
CLR a                ; clear register a
MOV {t}, a           ; clear variable t
INC {t}              ; t=t + 1
MOV {c}, {t}         ; c = t
MOV {c+0}, {t}       ; LSB of C = t (you don't have to enter the +0)
MOV {c+1}, {t}       ; MSB of C = t
MOV {c},#10          ; assign value
```

You can also change SFR's from BASIC.

```
P1 = 12              'this is obvious
ACC = 5              'this is ok too
B = 3                'B is a SFR too
!MUL AB              'acc = acc * b
Print acc
```

EXTERNAL variables are stored similar.
 Strings are stored with a terminating zero.

Example :

```

$RAMSTART = 0
Dim s As String * 10          'reserve 10 bytes + 1 for string terminator
s = "abcde"                  'assign string constant to string

ram location 0 = "a"         'first memory location
ram location 1 = "b"
ram location 2 = "c"
ram location 3 = "d"
ram location 4 = "e"
ram location 5 = #0
    
```

External variables must be accessed somewhat different.

```

Dim T as XRAM Byte
mov dptr,#{T}                ; set datapointer
mov a,#65                    ; place A into acc
movx @dptr,a                 ; move to external memory
Print T                      ; print it from basic
    
```

```

Dim T1 as XRAM Integer
mov dptr,#{T1}               ; set datapointer
mov a,#65                    ; place A into acc (LSB)
movx @dptr,a                 ; move to external memory
inc dptr                     ; move datapointer
mov a,#1                     ; 1 to MSB
movx @dptr,a                 ; move to external memory

Print T1                     ; print it from basic
    
```

Memory usage

Some bytes are used by the internal assembler routines.
 You can however use this memory locations when they are not needed by the assembler.

Used bitvariables

bit address	description
00 ^{hex}	bit used for swap statement
01 ^{hex}	bit used for integer/word display
02 ^{hex}	bit used for uppercase conversion in combination with INPUTHEX
03 ^{hex}	bit used for NOECHO in combination with INPUT

Used bytevariables

bit address	description
08 ^{hex} - 23 ^{hex}	bytes used for input/output buffer
24 ^{hex} - 31 ^{hex}	bytes used for the I2C data buffer

Microprocessor support

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

8032/8052/AT89S8252 TIMER2 support

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 feature that a 1 to 0 transition on an 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 &HFFFF and sets the TF2 overflow flag bit upon overflow. This causes the TIMER2 registers to be reloaded with the 16-bit value in RCAP2L and RCAP2H.

The values in RCAP2L and RCAP2H are preset by software means.

Counter mode:

CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

Timer mode:

CONFIG TIMER2=COUNTER,GATE=INTERNAL,MODE=0

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

Counter mode:

```
CONFIG TIMER2=COUNTER,GATE=EXTERNAL,MODE=0
```

Timer mode:

```
CONFIG TIMER2=TIMER,GATE=EXTERNAL,MODE=0
```

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

Counter mode:

```
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP
```

Timer mode:

```
CONFIG TIMER2=COUNTER,GATE=INTERNAL/EXTERNAL,MODE=0,DIRECTION=UP
```

A logic 0 applied at pin T2EX causes TIMER2 to count down. The timer will under flow when TL2 and TH2 become equal to the value stored in RCAP2L and RCAP2H. TIMER2 under flows sets the TF2 flag and causes **&HFFFF** 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 setup from assembler this way.

Clock output

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

The output frequency = $(f_{OSC} / 4) / (65536 - \text{CAPTURE})$

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
```

Note that the flags are cleared automatically when the RETURN statement is encountered.

AT89S8252 support

The AT89S8252 has 8K of flashROM and 2K of data flashROM. It also has build in SPI and watchdog timer support.

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 generate an interrupt. 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
    
```

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 LT 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 LT 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
Writeeprom S
Writeeprom S2          'write strings
Writeeprom L           'write long

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