

# MadLab PICLAB

version 2.1

Written by James Hutchby  
Copyright © MadLab Ltd. 2000-2004  
All Rights Reserved

[info@madlab.org](mailto:info@madlab.org)  
[www.madlab.org](http://www.madlab.org)

MadLab® is a registered service mark of MadLab Ltd. in the UK.  
PBASIC and BASIC Stamp are registered trademarks of Parallax Inc.  
PIC is a registered trademark of Microchip Technology Inc.

## Contents

---

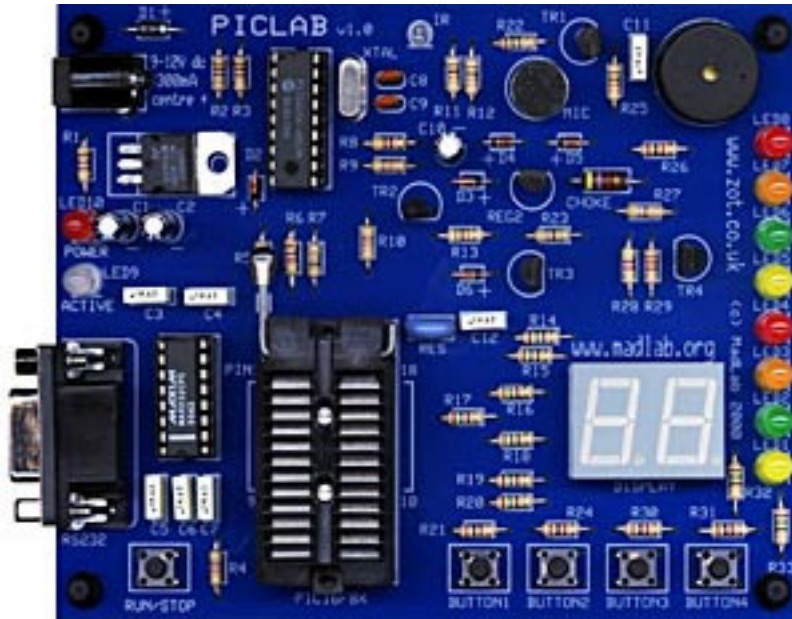
Introduction .....	3
PICLAB Programmer .....	3
System Requirements.....	4
Packing List.....	5
Installing PICLAB .....	6
Hardware Setup .....	7
PICLAB Programmer .....	7
PICBOT .....	7
Quick Start .....	8
Commands.....	9
File Menu .....	9
Edit Menu.....	10
Options Menu .....	11
Program Menu .....	13
Window Menu .....	15
Help Menu .....	16
Toolbar.....	17
BASIC Programs.....	18
Differences to BASIC Stamp PBASIC .....	18
BASIC Instructions.....	19
Operators and Precedence.....	54
System Variables .....	55
PICLAB Programmer - Tester Board .....	55
PICLAB Programmer - BASIC Stamp.....	56
PICLAB Programmer - Generic PIC .....	56
PICBOT .....	57
Peripherals.....	58
Software Updates .....	62

## Introduction

---

The PICLAB software supports two different pieces of hardware, PICLAB Programmer and PICBOT.

### PICLAB Programmer



PICLAB Programmer functions both as a normal PIC device programmer, and as a tester board for running simple PIC programs.

As a device programmer (where a programmed PIC is destined for some other board), it can program PICs with industry-standard hex files produced by assemblers. The Microchip development system MPLab produces hex files in the correct format. You can download the software free from the Microchip website ([www.microchip.com](http://www.microchip.com)).

PICs can also be programmed in BASIC using the integrated editor and optimising compiler in PICLAB.

Additionally PICs can be programmed with flowcharts constructed using Crocodile Technology 1.5 and above from Crocodile Clips ([www.crocodile-clips.com](http://www.crocodile-clips.com)). Support for PICLAB Programmer is built in to Crocodile Technology and it can be controlled from within this application.

As a tester board, BASIC programs and Crocodile Technology flowcharts can be downloaded into a PIC and run in situ. PICLAB features a number of on-board peripherals which can be controlled from BASIC or the flowchart, with good correspondence between the operation of the virtual microcontroller on the Crocodile Technology screen and the physical microcontroller on the bench.

PICLAB Programmer is designed to work with Microchip's 18-pin flash PIC family. These devices can be re-programmed without the need for a UV eraser and so are ideal in an educational environment.

PICLAB Programmer currently supports the following devices: PIC16F83, PIC16F84, PIC16F84A, PIC16F627, PIC16F628. Support for new flash PICs will be added in the future (see Software Updates).

## **System Requirements**

---

IBM PC or compatible

486 processor or better

Windows 95 or later

10MB free disk space

free serial port

## **Packing List**

---

PICLAB Programmer

mains power adaptor

RS232 serial cable

installation disk

## **Installing PICLAB**

---

The software is provided as a self-extracting compressed file. Simply run the executable and select the directory you wish to install the files into.

If you are using PICLAB in conjunction with Crocodile Technology 1.5 and above from Crocodile Clips, you can also select the directory to install support for Crocodile Technology. This would normally be the sub-directory "PlugIns\MadLab" in your Crocodile Technology directory.

## Hardware Setup

---

### PICLAB Programmer

Connect the RS232 cable (9-way male to 9-way female) to PICLAB Programmer and to your PC.

Connect the power supply to the power connector. The supply should be capable of delivering 9V to 12V regulated dc at a maximum current of 300mA, fitted with a 2.1mm power plug with centre positive. The POWER LED (red) on PICLAB Programmer should come on when the power is connected, and the ACTIVE LED should flash green twice.

Insert a PIC (16F83, 16F84, 16F84A, 16F627 or 16F628) into the ZIF socket on PICLAB with the notch in the PIC nearest the ZIF handle. Push the handle down to lock the PIC in place.

The RUN/STOP button starts and stops execution of the program in the PIC in the ZIF socket. The ACTIVE LED is green when the program is running, and is yellow when the PIC is being programmed. A PIC should never be inserted or removed from the ZIF socket when the ACTIVE LED is on.

### PICBOT

Connect the RS232 cable (9-way male to 9-way female) to PICBOT and to your PC.

Insert 4 AA batteries into the battery box and slide the power switch to the on position. PICBOT should beep twice and its LEDs should flash if it is functioning properly.

The green GO button starts the program in PICBOT running, and the red STOP button halts it (but see Compiler Options). It can also be controlled from the PC.

Note that the RS232 connector shouldn't be removed from PICBOT when it is powered up, and PICBOT should never be powered down while downloading and writing to the PIC memory (when both LEDs are lit).

## Quick Start

---

After installing the software and setting up the hardware, open one of the example BASIC programs in the subdirectory "programmer/examples" or "robot/examples" located in the installation directory. Make sure Tester Board is selected in the Options menu if you are using PICLAB Programmer, then select Compile in the Program menu (or click the gearwheels icon). After compilation select Download Memory in the Program menu (or click the down arrow icon) and click Begin. After the program has successfully downloaded press the RUN/STOP button on PICLAB Programmer, or the green GO button on PICBOT.



## Commands

---

### File Menu

#### New

Opens and clears a new BASIC window for program editing. If the current window has been modified and not saved then you are prompted to save the file.

#### Open

Opens a BASIC, assembler or hex file from disk. Hex files must be in Intel INHX8M format.

#### Close

Closes the current window. If its contents have been modified and not saved then you are prompted to save the file.

#### Save

Saves the contents of the current window to disk. BASIC, Assembler and Memory windows can be saved. BASIC and assembler are saved as text files, memory is saved in Intel INHX8M format.

#### Save As

Saves the contents of the current window with a new filename.

#### Print Setup

Allows a printer to be selected.

#### Print

Prints the contents of the current window.

#### Exit

Exits the application.

## **Edit Menu**

### **Undo**

Undoes the last editor operation. One level of undo is available.

### **Cut**

Cuts selected text to the clipboard. Only applicable to the BASIC window.

### **Copy**

Copies selected text to the clipboard.

### **Paste**

Pastes text from the clipboard. Only applicable to the BASIC window.

### **Delete**

Deletes selected text. Only applicable to the BASIC window.

### **Select All**

Selects all the text in the current window.

### **Find**

Finds a text string in the current window.

### **Find Next**

Finds the next occurrence of the text string in the current window.

### **Replace**

Replaces a text string in the current window with another text string.

### **Set Font**

Allows the font face, style and size to be changed. The font is used in any new windows subsequently opened, but doesn't change the font in existing windows.

### **Tabs**

Sets the tab stop position.

## Options Menu

### Set up Programmer

Allows the PC serial port (COM1 to COM4) to be selected.

### PIC Options

PICLAB Programmer only. Allows selection of various PIC options.

Device: selects the PIC in use (16F83, 16F84, 16F84A, 16F627 or 16F628).

Clock: the target-board clock frequency when programming PICs for boards other than PICLAB. Take care that you don't select a faster clock than the PIC you are using can work at (the maximum speed of the 'F84 is 10MHz, and the 'F84A is 20MHz).

ID: a 4 hexadecimal-digit code used for identification purposes. Typically contains a software version number.

The remaining checkboxes control the configuration options of the PIC. Refer to the Microchip ([www.microchip.com](http://www.microchip.com)) documentation for full details. When running programs on PICLAB Programmer these options are pre-configured, as is the clock frequency.

### Compiler Options

PICLAB Programmer only. The Tester board option is the same as that found in the main Options Menu. Check this box if you are running programs on PICLAB Programmer.

PICLAB Programmer only. The compiler can generate code for a PIC which emulates a BASIC Stamp (PORTB used as a bi-directional 8-bit port). Note that this is not an actual BASIC Stamp but rather a PIC with similar I/O.

The compiler will generate 8-bit or 16-bit variable object code. You wouldn't normally need to use 16-bit, the exception being if counters or other variables in your program or flowchart could exceed 255. Programs and flowcharts compiled using 16-bit variables are roughly twice the size of 8-bit compilations, allow a smaller number of program variables in memory, and run more slowly.

Signed variables can also be enabled. Signed variables are allowed to be negative, as opposed to unsigned variables which are always positive. There is a program memory overhead associated with their use. See **DIM** for more information.

PICLAB will optionally generate assembler language that corresponds to a compiled BASIC program or Crocodile Technology flowchart. This is useful for learning more about PIC assembly language and also gives the location of any compilation errors. Select this checkbox to enable assembler output.

Very large BASIC programs can cause the assembler buffer to overflow. If this happens then disable assembler output to allow a compilation to succeed.

PICLAB Programmer only. Weak pull-ups can be enabled or disabled when compiling for PICs for other boards - refer to the Microchip documentation for details.

PICBOT only. The STOP button can be made available for use by your program. Normally the STOP button halts execution of a program but this can be overridden by selecting the "STOP button" option. If checked then PICBOT cannot be halted by pressing the STOP button, but can still be halted from the PC (unless "RS232 comms" is enabled).

PICBOT only. Bytes received on the serial port can be made available for use by your program. Normally received bytes control the downloading and running of programs etc, but this can be overridden by selecting the "RS232 comms" option. If checked then PICBOT will not respond to stop commands from the PC, and can only be halted by pressing the STOP button (if not disabled) or turning the power switch off. See **RXBYTE** for more information.

## **I/O Options**

PICLAB Programmer only. The peripherals connected to the PICLAB 18-pin PIC can be selectively enabled and disabled. The I/O Options dialog indicates which peripherals are enabled, and how the inputs and outputs of the PIC are logically connected. Note that the buzzer and speaker are mutually exclusive, and that the low and high 7-segments are connected to outputs 0 to 3 and outputs 4 to 7 respectively.

If you access the peripherals in your BASIC programs through the use of system variables then there is never any need to disable pins because there is no sharing.

## **Monitor Sensors**

PICBOT only. Continuously monitors the digital and analogue sensors on board and displays their values. Also displays the status of the expansion connector inputs. This is a useful tool for testing expansion boards.

## **Tester Board**

PICLAB Programmer only. If ticked then downloaded programs can be run on PICLAB Programmer (as opposed to using it to program PICs for other boards). Deselect this option if you are developing programs for other boards.

## **Auto-run**

If ticked then downloaded programs are run automatically (equivalent to pressing the RUN/STOP or GO button after downloading).

## **CT Expert Mode**

When selected enables Crocodile Technology expert mode. This allows access to the PICLAB toolbar when downloading flowcharts. Clicking on the download button in a Crocodile Technology flowchart brings up PICLAB's main window. The PIC Options, Compiler Options and I/O Options can be changed prior to downloading the flowchart by clicking the compile button followed by the download button.

## **Program Menu**

### **Reset Programmer**

Resets PICLAB Programmer or PICBOT and displays the firmware version number. Used to check that communications with the PC are working.

### **Device Usage**

PICBOT only. Displays usage statistics for the PIC, specifically the number of times the FLASH program memory has been completely written.

### **Compile**

Compiles the current BASIC program into object code, ready for downloading, and optionally generates assembler code (if the Assembler option is checked in Compiler Options). An Errors window displays the results of the compilation and any errors found.

A BASIC program must be successfully compiled without errors before it can be downloaded.

### **Download Memory**

Downloads memory to a PIC. The different memory areas of a PIC can be individually downloaded by checking or unchecking the boxes in the Download dialog (PICLAB Programmer only).

A PIC contains four memory areas:

Program memory - the program itself

Data memory - non-volatile memory (EEPROM) for permanent variable storage

Configuration - the configuration bits such as the oscillator type, code-protection flag etc.

ID - an identification code

To cancel the operation before starting click on the close box in the top right hand corner of the dialog.

### **Upload Memory**

Uploads memory from a PIC. The different memory areas of a PIC can be individually uploaded by checking or unchecking the boxes in the Upload dialog (PICLAB Programmer only).

If a PIC has been code-protected then memory will read as all zeroes.

To cancel the operation before starting click on the close box in the top right hand corner of the dialog.

## **Verify Memory**

Verifies memory from a PIC (i.e. compares the PIC memory with the Memory window). The different memory areas of a PIC can be individually verified by checking or unchecking the boxes in the Verify dialog (PICLAB Programmer only). Note that memory is automatically verified when downloaded, but this option provides an additional check. A PIC that has been code-protected however will fail to verify.

To cancel the operation before starting click on the close box in the top right hand corner of the dialog.

## **Erase Device**

PICLAB Programmer only. Erases all the memory in a PIC.

## **Run Program**

Runs the program downloaded to the PIC. Equivalent to pressing the RUN/STOP button on PICLAB Programmer, or the green GO button on PICBOT. The ACTIVE LED on PICLAB Programmer is green when a program is running.

PICLAB Programmer - this option is not available if Tester Board has not been selected in the Options Menu.

## **Stop Program**

Stops the program in the PIC. Equivalent to pressing the RUN/STOP button on PICLAB Programmer, or the red STOP button on PICBOT.

PICLAB Programmer - this option is not available if Tester Board has not been selected in the Options Menu.

## **Window Menu**

### **BASIC**

Opens the BASIC window for program editing.

### **Assembler**

Opens the Assembler window to display compiled assembler code.

### **Memory**

Opens the Memory window to display the object code of a compiled program, a loaded hex file, or uploaded memory.

### **Monitor**

PICBOT only. Opens the Monitor window which is used for bi-directional serial port communications with PICBOT. See "RS232 comms" option in Compiler Options, and **PRINT**.

### **Cascade**

Cascades all open windows.

### **Tile Vertically**

Tiles all open windows vertically.

### **Tile Horizontally**

Tiles all open windows horizontally.

### **Arrange Icons**

Arranges any iconic windows along the bottom of the screen.

### **Close All**

Closes all open windows.

## **Help Menu**

### **Contents**

Displays the online help pages.

### **Using help**

How to use help.


















### **About**

Displays the copyright message and the software version number.



## Toolbar

---

<u>Button</u>	<u>Action</u>	<u>Menu equivalent</u>
	Opens BASIC, assembler or hex file	File Open
	Saves current window	File Save
	Cuts text to clipboard	Edit Cut
	Copies text to clipboard	Edit Copy
	Pastes text from clipboard	Edit Paste
	Sets up programmer	Options Set up programmer
	Sets PIC options	Options PIC Options
	Sets compiler options	Options Compiler Options
	Sets I/O options	Options  /O Options
	Resets programmer	Program Reset Programmer
	Compiles BASIC	Program Compile
	Downloads memory to PIC	Program Download Memory
	Uploads memory from PIC	Program Upload Memory
	Runs PIC program	Program Run Program
	Stops PIC program	Program Stop Program
	About the application	Help About
	Help table of contents	Help Contents

## **BASIC Programs**

---

A BASIC program consists of a number of lines (statements) containing labels, instructions, constants, and variables.

Labels identify sections of a program. They are up to 32 characters long, and can contain letters, numbers and the underscore character `_`, but the first character must be a letter. The definition of a label requires a colon immediately following the label, but the colon is not needed when the label is used in a **GOTO** or **GOSUB** instruction. Labels are not case-sensitive. See **GOTO** for an example of the use of a label.

Instructions can be in upper or lower case. Instructions do not have to be indented in a line but it is good practice to do so. The keywords for instructions cannot be used as labels or variable names.

Constants can be decimal (base 10), hexadecimal (base 16), binary (base 2) or character. Hexadecimal constants are preceded with a `$` symbol, binary constants with a `%` symbol, and characters are enclosed in double quotes. Embedded double quotes in characters and strings are preceded by a backslash character. See **LET** for examples of constants.

Variables can be single-bit, 8-bit, 16-bit, signed or unsigned. The names of variables follow the same rules as labels and are not case-sensitive. See **DIM**.

All arithmetic is integer arithmetic. Overflows are not detected.

The hardware peripherals are accessed through pre-defined variables (system variables) or by using pin numbers. Many peripherals have multiple names or aliases.

The installed "examples" directory contains sample BASIC programs.

### **Differences to BASIC Stamp PBASIC**

Negative numbers are allowed. Complex expressions are allowed with operator precedence.

Extra instructions and synonyms have been introduced, some PBASIC instructions are not supported, and there are changes to the syntax:

**BEEP**, **EVERY** are new instructions

**DIM** is not required in PBASIC

**EEPROM** and **SOUND** have different syntax

**HIGH** and **LOW** function differently if a variable is used

**IF** statements allow instructions as well as labels and include an **ELSE** part

**ON** is similar to **BRANCH**

**PRINT** is similar to **DEBUG**

**RXBYTE** and **TXBYTE** are equivalent to **SERIN** and **SEROUT**

## BASIC Instructions

---

**BEEP**  
**CLEAR** <pin>|<variable>  
**CLS**  
**DATA** <address>,<constant>|<string>,<constant>|<string>,...  
**DIM** <variable>{:1|1u|8|8u|8s|16|16u|16s|u|s},<variable>,...  
**EEPROM** <address>,<constant>|<string>,<constant>|<string>,...  
**END**  
**EVERY** <period> **GOSUB** <label>  
**FOR** <variable> = <start> **TO** <end> {**STEP** <step>}  
**GOSUB** <label>  
**GOTO** <label>  
**HIGH** <pin>|<variable>  
**IF** <expression> **THEN** <label>|{**THEN**} <instruction> {**ELSE** <label>|<instruction>}  
{**LET**} <variable> = <expression>  
**LOW** <pin>|<variable>  
**NEXT** {<variable>}  
**ON** <expression> **GOTO**|**GOSUB** <label>,<label>,...  
**PAUSE** <duration>  
**PRINT** <expression>|<string>,<expression>|<string>,...  
**RANDOM** <variable>  
**READ** <address>,<variable>,<variable>,...  
**REM**  
**RESET** <pin>|<variable>  
**RETURN**  
**RXBYTE** <variable>,<variable>,...  
**RXCHAR** <variable>,<variable>,...  
**SET** <pin>|<variable>  
**SOUND** <note>,<duration>  
**SYMBOL** <symbol> = <constant>|<variable>  
**TOGGLE** <pin>|<variable>  
**TXBYTE** <expression>|<string>,<expression>|<string>,...  
**TXCHAR** <expression>|<string>,<expression>|<string>,...  
**WRITE** <address>,<expression>|<string>,<expression>|<string>,...

| indicates alternatives, {} indicates optional

<symbol>, <variable>, <label> = string of letters, numbers and underscores with a letter as the first character

<expression>, <start>, <end> = expression involving constants, symbols, variables and operators

<constant>, <step> = simple constant or expression that evaluates to a constant

<string> = string of characters delimited by double quotes

<pin> = 0 to 7, or symbolic name

<address> = 0 to 63 for PICLAB Programmer (0 to 127 for '62X), or 0 to 95 for PICBOT

<period> = 0 to 255 hundredths of a second

<duration> = 0 to 65535 milliseconds

<note> = 0 to 59 (5 octaves A to G) for PICLAB Programmer, or 0 to 71 (6 octaves A to G) for PICBOT

Indicates a comment (remark). Comments are aids to the understanding of a program and are ignored by the compiler. ' can be used by itself on a line, or after other instructions.

See also **REM**.

```
' this is a comment
    IF x > 10 GOTO error      ' test x against upper limit
```

## **BEEP**

PICLAB Programmer - Tester Board option must be enabled. Sounds a short beep. *Note* is reset to its default value by this instruction (only applicable to PICBOT).

See also **SOUND**.

```
BEEP                                ' double beep
PAUSE 250
BEEP
```

**CLEAR** <pin>|<variable>

See **LOW**.

## **CLS**

PICBOT only. Clears the monitor window if open.

```
CLS                                ' clear window
PRINT "The result is ",result
```

**DATA** <address>,<constant>|<string>,<constant>|<string>,...

Initialises EEPROM data memory with 8-bit data. EEPROM memory can be used to set up data tables, or as additional memory for message strings etc. **DATA** statements can appear anywhere in a program.

<address> is a constant or constant expression that evaluates to an address in the range 0 to 63 for PICLAB Programmer (0 to 127 for '62X), or 0 to 95 for PICBOT. An address outside this range causes an error. The data to be stored can be a constant expression or a string. Data is stored in consecutive memory locations, and strings are stored one byte per character without a terminator.

**EEPROM** is a synonym for this instruction. See also **READ**, **WRITE** (this instruction differs from **WRITE** in that the data is written when the program is downloaded, not when it is executed).

```
DATA 10,40,45,50,55,60      ' data table with 5 entries
READ 10+n,x                ' reading the n'th entry

DATA 20,"an error message"

DATA 30,1234>>8,1234&$ff    ' storing a 16-bit constant
```



**DIM** <variable>{:1|1u|8|8u|8s|16|16u|16s|u|s},<variable>,...

Defines a variable and optionally specifies its size and signed properties. Variables do not have to be defined to be used but it is good practice to do so. Variables can then be made no larger than they need to be. For example, a flag variable which only holds the values 0 and 1 can be efficiently stored in a single bit rather than in a complete byte. Memory for variables is in short supply so it is advisable not to waste it. 16-bit variables should be used sparingly because the compiler needs to generate extra code to handle them which means that program memory fills more quickly. Additionally 16-bit code runs more slowly. Signed variables also introduce an overhead compared to unsigned variables.

**DIM** if used must precede any reference to the variable. It is usual to group all the **DIM** statements at the beginning of a program. If a variable is used without (or before) being defined then its properties default to those specified in the Compiler Options (but see **HIGH, LOW, TOGGLE**).

The optional modifiers specify the size and signed properties of the variable:

:1        single-bit variable, 0 or 1  
:8u      8-bit unsigned variable, in the range 0 to 255  
:8s      8-bit signed variable, in the range -128 to 127  
:16u     16-bit unsigned variable, in the range 0 to 65535  
:16s     16-bit signed variable, in the range -32768 to 32767

Single-bit variables are always unsigned. BASIC Stamp variables are 8-bit unsigned. Note that **DIM** does not zero variables.

```
DIM x:16u                    ' 0 to 65535
DIM y:8s                    ' -128 to 127
DIM flag:1                  ' single-bit variable (0 or 1)
DIM x                        ' properties defined by Compiler Options
DIM x:s                     ' signed variable, size defined by
                             ' Compiler Options
```

**EEPROM** <address>,<constant>|<string>,<constant>|<string>,...

See **DATA**.

## **END**

Terminates a BASIC program. Turns all peripherals off and stops execution. This instruction is optional in a program. If execution reaches the last statement it will stop anyway.

```
IF Button1 THEN END      ' terminate program if button pressed
```

## **EVERY** <period> **GOSUB** <label>

PICBOT only. Background processing. The subroutine at <label> is called repeatedly every <period>/100 seconds.

Background processing is asynchronous with the main program. That means that the background subroutine can be called at any time, even in the middle of an instruction or calculation in the main program. It is not guaranteed to be called between statements. This is a potential problem if the background subroutine uses variables altered by the main program. Use a semaphore to guarantee exclusive access in this case. Note that the background subroutine shouldn't call any subroutines itself.

<period> is specified in hundredths of a second, in the range 1 to 255. **EVERY 0 GOSUB** disables background processing, and **EVERY** <period> **GOSUB** leaves the subroutine unchanged but changes the period.

See also **GOSUB**.

```
        DIM semaphore:1
        EVERY 10 GOSUB flash
        SET semaphore
        ....
        CLEAR semaphore
        ....
flash:   IF semaphore RETURN
        TOGGLE Led1
        RETURN
        EVERY 0 GOSUB
```

' critical code  
' this subroutine is called  
' 10 times every second  
' background processing off

**FOR** <variable> = <start> **TO** <end> {**STEP** <step>}

Repeats a sequence of instructions. Instructions between the **FOR** statement and the matching **NEXT** statement are executed one or more times.

<variable> is initialised to the value <start>, instructions in the loop are executed, <step> is added to <variable>, which is then compared to <end>. The loop is repeated until <variable> is greater than <end> if <step> is positive, or until <variable> is less than <end> if <step> is negative. The step is optional and defaults to 1 but can be any positive or negative constant or constant expression. <start> and <end> can be arbitrary expressions. <start> is evaluated once at the beginning of the loop, and <end> is evaluated each time through the loop. Note that the loop always executes at least once irrespective of the values of <start> and <end>.

See also **NEXT**.

```
FOR n = 1 TO 0           ' this loop executes once
....
NEXT

FOR n = 10 TO 1 STEP -1 ' this loop executes 10 times
....
NEXT

FOR n = 1 TO 10         ' this loop executes 5 times
....
n = n + 1
NEXT n

FOR x = 1 TO 10         ' 10 times through outer loop
FOR y = 1 TO 10 STEP 2  ' 10 * 5 times through inner loop
....
NEXT y
NEXT x

LET <variable> = <start> ' FOR loop equivalent
loop:
....
LET <variable> = <variable> + <step>
IF <variable> <= <end> THEN loop
```

## **GOSUB** <label>

Executes a subroutine then continues with the next instruction. Subroutines can be nested (a subroutine calling another subroutine), but they should not be nested more than 4 levels. If you exceed this limit your program will behave unpredictably.

See also **RETURN**.

```

        GOSUB flash                ' simple subroutine call
flash:   ....
        HIGH Led1
        PAUSE 100
        LOW Led1
        RETURN

        GOSUB sub1                 ' nested subroutines
sub1:    ....
        GOSUB sub2                 ' max depth = 4
        ....
        RETURN
sub2:    ....
        RETURN
```

**GOTO** <label>

Unconditional branch. Execution continues with the instruction at <label>.

```
        GOTO error
        . . . .
error:  PRINT "*** Timeout! ***"
        END
```

## **HIGH** <pin>|<variable>

Makes a pin high, or sets a variable to 1.

<pin> can either be a symbolic name, or a number from 0 to 7. Alternatively a variable can be specified in which case this instruction is equivalent to **LET** <variable> = 1. Note that this functions differently to BASIC Stamp PBASIC. If a variable is used in this way before it is defined (see **DIM**) then it is created as a single-bit variable.

**SET** is a synonym for this instruction. See also **LOW**, **TOGGLE**.

```
HIGH Led1           ' turn Led 1 on
HIGH 3              ' make pin 3 high
LOW RightMotorBackward ' right wheel
HIGH RightMotorForward
HIGH x              ' x = 1
LET n = 5
HIGH n              ' this is not the same as HIGH 5
```



**IF** <expression> **THEN** <label>{**THEN**} <instruction> {**ELSE** <label>|<instruction>}

Conditional branching or execution. <expression> is evaluated and depending on whether it is true or false (non-zero or zero) the program branches to <label> or executes <instruction>. The **ELSE** part optionally specifies an alternative branch or instruction to be executed if the **THEN** part is false.

Take care with operator precedence in expressions. For example, **IF** mask & 3 = 0 **THEN** ... The = operator has a higher precedence than the & operator. To make the instruction execute as expected bracket it thus **IF** (mask & 3) = 0 **THEN** ...

```
IF n > 10 LET n = 0           ' wrap around if > 10
IF n > 100 PRINT "too big!"
IF n THEN label              ' branches if n <> 0
IF a & b THEN label          ' might not work (1 & 2 = 0 e.g.)
IF a <> 0 & b <> 0 THEN label ' correct form
IF x > 123 THEN exit         ' these two statements are
IF x > 123 THEN GOTO exit    ' identical
IF flag THEN LET x = x + 1   ' LET is required
IF flag LET x = x + 1       ' shorter format
IF flag x = x + 1           ' causes a syntax error
IF x < 0 THEN error ELSE ok
IF x >= 0 PRINT "positive" ELSE PRINT "negative"
IF mask & 3 = 0 THEN ...    ' actually IF mask & (3 = 0)
                           ' or IF mask & 0
```

**{LET}** <variable> = <expression>

Sets a variable to a value. The keyword is optional. <expression> can be an arbitrary expression involving constants, operators and brackets. If a variable is used before being defined (see **DIM**) then its size and signed properties default to those specified in the Compiler Options.

```
LET x = 0           ' clear variable
x = x + 1          ' increment variable
LET n1 = $ab       ' hexadecimal constant
LET n2 = %10101010 ' binary constant
LET n3 = "*"       ' character constant
LET n4 = "\"       ' embedded double quote
LET n5 = "\\\"     ' embedded backslash
LET z = 10*x + y/2 ' complex expression
```

## **LOW** <pin>|<variable>

Makes a pin low, or zeroes a variable.

<pin> can either be a symbolic name, or a number from 0 to 7. Alternatively a variable can be specified in which case this instruction is equivalent to **LET** <variable> = 0. Note that this functions differently to BASIC Stamp PBASIC. If a variable is used in this way before it is defined (see **DIM**) then it is created as a single-bit variable.

**RESET** and **CLEAR** are synonyms for this instruction. See also **HIGH**, **TOGGLE**.

```
LOW Led2           ' turn Led 2 off
LOW 4              ' make pin 4 low
LOW x              ' x = 0
SYMBOL n = 5
LOW n              ' this is not the same as LOW 5
```

## **NEXT** {<variable>}

Marks the end of a **FOR** loop. The variable can be omitted only for non-nested loops, in which case **NEXT** matches the last **FOR** statement.

See also **FOR**.

```
FOR n = 1 TO 10           ' 10 times through the loop
PRINT n
NEXT n                   ' n could be omitted

FOR x = 1 TO 10
FOR y = 1 TO 10
....
IF error NEXT x         ' exit inner loop prematurely
....
NEXT y
NEXT x
```

**ON** <expression> **GOTO|GOSUB** <label>,<label>,...

Branches to one of a number of addresses, or calls one of a number of subroutines. If the expression evaluates to greater than the number of labels, then execution continues at the next instruction. The number of labels is limited to 32, and the index starts at 0.

See also **GOTO**, **GOSUB**.

```
        ON button GOSUB state1,state2,state3
state1:  ....
        HIGH Led1           ' if button = 0
        RETURN
state2:  HIGH Led2           ' if button = 1
        RETURN
state3:  HIGH Led3           ' if button = 2
        RETURN
```

## **PAUSE** <duration>

Pauses for a number of milliseconds (thousandths of a second). The duration can be a constant or an expression. For PICBOT the accuracy of the pause is 10ms - **PAUSE** 1 and **PAUSE** 10 both wait for the same amount of time (1/100 second). The maximum pause is 65535 ms (a little over a minute).

```
PAUSE 500
```

```
' wait half a second
```

**PRINT** <expression>|<string>,<expression>|<string>,...

PICBOT only. Sends output to the RS232 serial port. This is a useful aid to debugging as it allows variables to be examined etc. **PRINT** works in conjunction with the monitor window.

The following print modifiers are available: %x to print binary, \$x to print hex, and @x or #x to print a character.

**PRINT** outputs a newline after printing its arguments. You can use **PRINT** "\n" to force additional newlines. Note that numbers greater than 32767 will print as negative numbers, and numbers less than -32768 will print as positive numbers.

```
PRINT x," squared is ",x*x
PRINT %255                ' prints "%11111111"
PRINT %11                 ' prints "%00001011" (decimal 11
                          ' in binary)
PRINT %(%11)              ' prints "%00000011"
PRINT $255                 ' prints "$FF"
PRINT #65                  ' prints "A"
PRINT "column1\tcolumn2"  ' prints in two columns (\t = tab)
PRINT "line1\nline2"     ' prints on two lines (\n = newline)
```

## **RANDOM** {<variable>}

PICBOT only. Gets the next number in a pseudo random sequence. The random number is also stored in the 16-bit variable *Rand*.

```
DIM n:16u
RANDOM n
PRINT "The next random number is ",n
PRINT "The last random number was ",Rand
```



**READ** <address>,<variable>,<variable>,...

Reads 8-bit data from EEPROM data memory. EEPROM memory is non-volatile which means that its contents are retained after power is removed. It can be used to store data that is needed permanently, or as additional memory for message strings etc.

<address> is an expression that evaluates to an address in the range 0 to 63 for PICLAB Programmer (0 to 127 for '62X), or 0 to 95 for PICBOT. An address outside this range reads as zero. Data is read from consecutive memory locations into the specified variable(s).

See also **WRITE, DATA**.

```
DIM x:16,hi:8,lo:8      ' read a 16-bit variable as two bytes
READ 10,hi,lo
LET x = (hi<<8) | lo    ' brackets necessary because of precedence
```

## REM

Indicates a comment (remark). Comments are aids to the understanding of a program and are ignored by the compiler. **REM** and ' can be used on lines by themselves, and ' can be used after other instructions.

```
REM this is a comment
```

```
IF x > 10 GOTO error
```

```
LET x = 1
```

```
' test x against upper limit
```

```
REM this will cause a syntax error
```

**RESET** <pin>|<variable>

See **LOW**.

## RETURN

Returns from a subroutine. Execution continues with the instruction after the last **GOSUB**.

See also **GOSUB**.

```
        GOSUB flash
flash:  . . . .
        TOGGLE Led1
        RETURN
```

**RXBYTE** <variable>,<variable>,...

PICBOT only. If RS232 comms are enabled (see Compiler Options) this instruction receives 8-bit data from the RS232 serial port (9600 baud, 1 stop bit, no parity). **RXBYTE** does not timeout but waits indefinitely until a character has been received. Use *RxIn* and *RxError* single-bit variables to check the serial port status. The received data is stored in the specified variable(s).

**RXCHAR** is a synonym for this instruction. See also **TXBYTE**.

```
loop:    IF !RxIn GOTO loop
        RXBYTE x
        If RxError GOTO error      ' x contains error code if error
        DIM result:16,hi:8,lo:8    ' receiving word data -
        RXBYTE hi,lo              ' assemble from two bytes
        LET result = (hi<<8) | lo  ' brackets needed because of precedence
```

**RXCHAR** <variable>,<variable>,...

PICBOT only. See **RXBYTE**.

**SET** <pin>|<variable>

See **HIGH**.

**SOUND** <note>,<duration>

PICLAB Programmer - Tester Board option must be enabled. Sounds a musical tone.

<note> is in the range 0 to 59 (PICLAB Programmer) or 0 to 71 (PICBOT), which corresponds to 5 or 6 octaves A to G. <duration> is in milliseconds but for PICBOT is only accurate to the nearest 10ms. Note that the PICLAB Programmer peripherals are disabled while a sound is playing, and the duration is limited to 4095 ms.

See also **BEEP**, **PAUSE**.

```
SOUND 39,500      ' C
SOUND 41,500      ' D
SOUND 43,500      ' E
SOUND 44,500      ' F
SOUND 46,500      ' G
SOUND 48,500      ' A
SOUND 50,500      ' B
SOUND 51,500      ' C

LET Note = <note>      ' equivalent of SOUND <note>,<duration>
HIGH Speaker
PAUSE <duration>
LOW Speaker
```



**SYMBOL** <symbol> = <constant>|<variable>

Defines a constant or an alias for a variable or pin. **SYMBOL** definitions should be placed prior to any references to them.

```
SYMBOL LIMIT = 100           ' upper limit
IF x > LIMIT GOTO error

LET x = N                    ' backward reference will cause
SYMBOL N = 10                ' an error

SYMBOL SIZE = 10*20         ' constant expression

SYMBOL switch = pin2        ' define an alias
IF switch THEN label

SYMBOL alert = Led1         ' define an alias
HIGH alert
```

## **TOGGLE** <pin>|<variable>

Makes a pin high if it is currently low or low if it is currently high, or toggles a variable between 0 and 1.

<pin> can either be a symbolic name, or a number from 0 to 7. Alternatively a variable can be specified in which case this instruction is equivalent to **LET** <variable> = <variable> ^ 1 when <variable> is a single bit. If a variable is used in this way before it is defined (see **DIM**) then it is created as a single-bit variable.

See also **HIGH**, **LOW**.

```
TOGGLE Led3           ' turn Led3 off if on and vice versa
TOGGLE 5              ' toggle pin 5
TOGGLE x              ' x = 0 if non-zero or x = 1 if zero
DIM flag:1           ' equivalent to flag = flag ^ 1 but
TOGGLE flag          ' more efficient
```

**TXBYTE** <expression>|<string>,<expression>|<string>,...

PICBOT only. Transmits 8-bit data to the RS232 serial port (9600 baud, 1 stop bit, no parity). The data to be transmitted can be an expression or a string.

**TXCHAR** is a synonym for this instruction. See also **RXBYTE**.

TXBYTE "hello world\n"	' transmits the string followed ' by a newline
DIM x:16	
TXBYTE x	' only transmits low byte
TXBYTE x>>8,x&\$ff	' transmitting a word (16 bits)

**TXCHAR** <expression>|<string>,<expression>|<string>,...

PICBOT only. See **TXBYTE**.

**WRITE** <address>,<expression>|<string>,<expression>|<string>,...

Writes 8-bit data to EEPROM data memory. EEPROM memory is non-volatile which means that its contents are retained after power is removed. It can be used to store data that is needed permanently, or as additional memory for message strings etc.

<address> is an expression that evaluates to an address in the range 0 to 63 for PICLAB Programmer (0 to 127 for '62X), or 0 to 95 for PICBOT. An address outside this range is ignored. The data to be written can be an expression or a string. Data is stored in consecutive memory locations, and strings are stored without a terminator.

See also **READ, DATA** (this instruction differs from **DATA** in that the data is written each time the instruction is executed).

```
WRITE 0,high_score
WRITE 1,"a character string"    ' no string terminator
WRITE 1,"another string",0      ' zero terminator

DIM x:16
WRITE 10,x                      ' only stores low byte
WRITE 10,x>>8,x&$ff            ' stores a complete 16-bit variable
```

## Operators and Precedence

---

BASIC expressions consist of constants, variables and the following operators. Operators with higher precedence are evaluated before those with lower precedence. Unary operators of equal precedence are evaluated right to left, and binary operators of equal precedence are evaluated left to right. Brackets can be used to override the precedence order.

()	brackets	10	highest precedence
+	unary plus	9	no operation
-	unary minus	9	negation (two's complement)
!	logical NOT (unary)	9	$!x = 1$ if $x = 0$ otherwise $!x = 0$
~	bitwise complement (unary)	9	$\sim 0 = 1, \sim 1 = 0$ (one's complement)
*	multiply	8	signed or unsigned multiplication
/	divide	8	signed or unsigned integer division
%	modulus	8	signed or unsigned remainder
+	add	7	signed or unsigned addition
-	subtract	7	signed or unsigned subtraction
>>	shift right	6	$>>n$ equivalent to dividing by 2 to the power of $n$
<<	shift left	6	$<<n$ equivalent to multiplying by 2 to the power of $n$
>=	greater than or equal	5	1 if true, 0 if false
>	greater than	5	1 if true, 0 if false
<=	less than or equal	5	1 if true, 0 if false
<	less than	5	1 if true, 0 if false
=	equal	4	1 if true, 0 if false
<>	not equal	4	1 if true, 0 if false
&	bitwise AND	3	$0 \& 0 = 0, 0 \& 1 = 0, 1 \& 0 = 0, 1 \& 1 = 1$
^	bitwise XOR	2	$0 \wedge 0 = 0, 0 \wedge 1 = 1, 1 \wedge 0 = 1, 1 \wedge 1 = 0$
	bitwise OR	1	$0   0 = 0, 0   1 = 1, 1   0 = 1, 1   1 = 1$

Shift right and shift left are limited to shifts from 0 to 16 positions, and the shift must be a constant. Shift right is an arithmetic shift for signed variables, or a logical shift for unsigned variables.

## System Variables

---

The peripherals connected to PICLAB Programmer and PICBOT can be accessed using the following predefined variables. The use of these variables in instructions is to be preferred to the use of literal pin numbers.

Many of the peripherals have additional names or aliases. They can be accessed equally using the aliases, but there is a degree of doubling-up with some of them. For example the PICLAB Programmer variable "pin2" accesses both pushbutton 3 and the microphone. If the individual variables "Button3" and "SoundSensor" are used instead then the two peripherals can be independently tested.

### PICLAB Programmer - Tester Board

Variable	Bits	Initial value	Range	Aliases	Notes
Button1	1	-	0 - 1	pin0	
Button2	1	-	0 - 1	pin1	
Button3	1	-	0 - 1	pin2	
Button4	1	-	0 - 1	pin3	
SoundSensor	1	-	0 - 1	pin2	
LightSensor	1	-	0 - 1	pin3	
Led1	1	0	0 - 1	YellowLed, YellowLed1	output 0
Led2	1	0	0 - 1	GreenLed, GreenLed1	output 1
Led3	1	0	0 - 1	OrangeLed, OrangeLed1	output 2
Led4	1	0	0 - 1	RedLed, RedLed1	output 3
Led5	1	0	0 - 1	YellowLed2	output 4
Led6	1	0	0 - 1	GreenLed2	output 5
Led7	1	0	0 - 1	OrangeLed2	output 6
Led8	1	0	0 - 1	RedLed2	output 7
Speaker	1	0	0 - 1		output 6
Buzzer	1	0	0 - 1		output 7
Digits	8	0	0 - 255	pins	7-segments

Note: pin2 is either Button3 or SoundSensor, and pin3 is either Button4 or LightSensor. See I/O Options.

## PICLAB Programmer - BASIC Stamp

Variable	Bits	Initial value	Range	Aliases	Notes
pin0	1	-	0 - 1		PORTB,0 (input/output)
pin1	1	-	0 - 1		PORTB,1 (input/output)
pin2	1	-	0 - 1		PORTB,2 (input/output)
pin3	1	-	0 - 1		PORTB,3 (input/output)
pin4	1	-	0 - 1		PORTB,4 (input/output)
pin5	1	-	0 - 1		PORTB,5 (input/output)
pin6	1	-	0 - 1		PORTB,6 (input/output)
pin7	1	-	0 - 1		PORTB,7 (input/output)
pins	8	-	0 - 255		PORTB (input/output)
b0	8	0	0 - 255	w0	w0 >> 8
b1	8	0	0 - 255	w0	w0 & \$ff
b2	8	0	0 - 255	w1	w1 >> 8
b3	8	0	0 - 255	w1	w1 & \$ff
b4	8	0	0 - 255	w2	w2 >> 8
b5	8	0	0 - 255	w2	w2 & \$ff
b6	8	0	0 - 255	w3	w3 >> 8
b7	8	0	0 - 255	w3	w3 & \$ff
b8	8	0	0 - 255	w4	w4 >> 8
b9	8	0	0 - 255	w4	w4 & \$ff
b10	8	0	0 - 255	w5	w5 >> 8
b11	8	0	0 - 255	w5	w5 & \$ff
b12	8	0	0 - 255	w6	w6 >> 8
b13	8	0	0 - 255	w6	w6 & \$ff
w0	16	0	0 - 65535	b0, b1	(b0 << 8)   b1
w1	16	0	0 - 65535	b2, b3	(b2 << 8)   b3
w2	16	0	0 - 65535	b4, b5	(b4 << 8)   b5
w3	16	0	0 - 65535	b6, b7	(b6 << 8)   b7
w4	16	0	0 - 65535	b8, b9	(b8 << 8)   b9
w5	16	0	0 - 65535	b10, b11	(b10 << 8)   b11
w6	16	0	0 - 65535	b12, b13	(b12 << 8)   b13

Notes: LET dirs = %... sets PORTB pins as inputs or outputs (0 = input, 1 = output).

When reading the entire 8-bits of PORTB use the form LET <variable> = pins

## PICLAB Programmer - Generic PIC

Variable	Bits	Initial value	Range	Aliases	Notes
pin0	1	-	0 - 1		PORTA,0 (input)
pin1	1	-	0 - 1		PORTA,1 (input)
pin2	1	-	0 - 1		PORTA,2 (input)
pin3	1	-	0 - 1		PORTA,3 (input)
pins	8	-	0 - 255		PORTB (output)

Note: PORTA is a 4-bit input port, and PORTB is an 8-bit output port.



## PICBOT

Variable	Bits	Initial	Range	Aliases	Notes
Button1	1	-	0 - 1	GoButton, pin0	green GO button
Button2	1	-	0 - 1	StopButton, pin1	red STOP button (if enabled)
Obstacle1	1	-	0 - 1	LeftObstacle, pin2	left obstacle sensor
Obstacle2	1	-	0 - 1	RightObstacle, pin3	right obstacle sensor
Input1	1	-	0 - 1	pin4	expansion digital input 1
Input2	1	-	0 - 1	pin5	expansion digital input 2
Input3	1	-	0 - 1	pin6	expansion digital input 3
Input4	1	-	0 - 1	pin7	expansion digital input 4
Motor1Forward	1	0	0 - 1	LeftMotorForward	left motor forward
Motor1Backward	1	0	0 - 1	LeftMotorBackward	left motor backward
Motor2Forward	1	0	0 - 1	RightMotorForward	right motor forward
Motor2Backward	1	0	0 - 1	RightMotorBackward	right motor backward
Led1	1	0	0 - 1	LeftLed	left led
Led2	1	0	0 - 1	RightLed	right led
Speaker	1	0	0 - 1		speaker
Output1	1	0	0 - 1		expansion digital output 1
pins	8	0	0 - 255		output port
LightSensor	8	-	0 - 255	Light	analogue light level
SoundSensor	8	-	0 - 255	Sound	analogue sound level
AnalogueSensor1	8	-	0 - 255	Analogue1	expansion analogue sensor 1
AnalogueSensor2	8	-	0 - 255	Analogue2	expansion analogue sensor 2
Speed1	8	128	0 - 255	LeftSpeed	left motor speed
Speed2	8	128	0 - 255	RightSpeed	right motor speed
Ticks	8	0	0 - 99		increments every 1/100s
Seconds	16	0	0 - 65535		increments every 1s
Note	8	47	0 - 71		current note
Rand	16	?	0 - 65535		last random number
RxIn	1	-	0 - 1		RS232 byte received ?
RxError	1	-	0 - 1		RS232 error ?

Note: the stop button is only available if "STOP button" is checked in Compiler Options.

## Peripherals

---

PICLAB Programmer includes the following peripherals:

- \* 8 LEDs (red, orange, green, yellow, red, orange, green, yellow)
- \* dual 7-segment display
- \* piezo speaker/buzzer
- \* 4 pushbuttons
- \* sound sensor (MIC)
- \* light sensor (IR)

The LEDs are connected to the 8 output pins of the 18-pin PIC in PICLAB Programmer, in parallel with the 7-segment displays. The 7-segment displays represent the hexadecimal value of the output pins, with the high hex digit showing the upper 4 pins and the low hex digit showing the lower 4 pins.

The piezo can function either as a speaker or as a buzzer. The buzzer is a gated fixed-frequency device, whereas the speaker must have its pin toggled to generate a sound. Logically the speaker is connected to `out6`, and the buzzer to `out7`.

Writing to the output pins of the PIC effects all the output peripherals. For example setting all 8 output pins high lights all the LEDs, displays 'FF' on the 7-segments, and sounds a click or tone on the speaker.

The pushbuttons are connected to the 4 input pins of the 18-pin PIC, in parallel with the sound sensor on `in2` and the light sensor on `in3`. Pressing Button3 is equivalent to triggering the sound sensor with a sharp clap or whistle, and pressing Button4 is equivalent to shining a light on the light sensor.

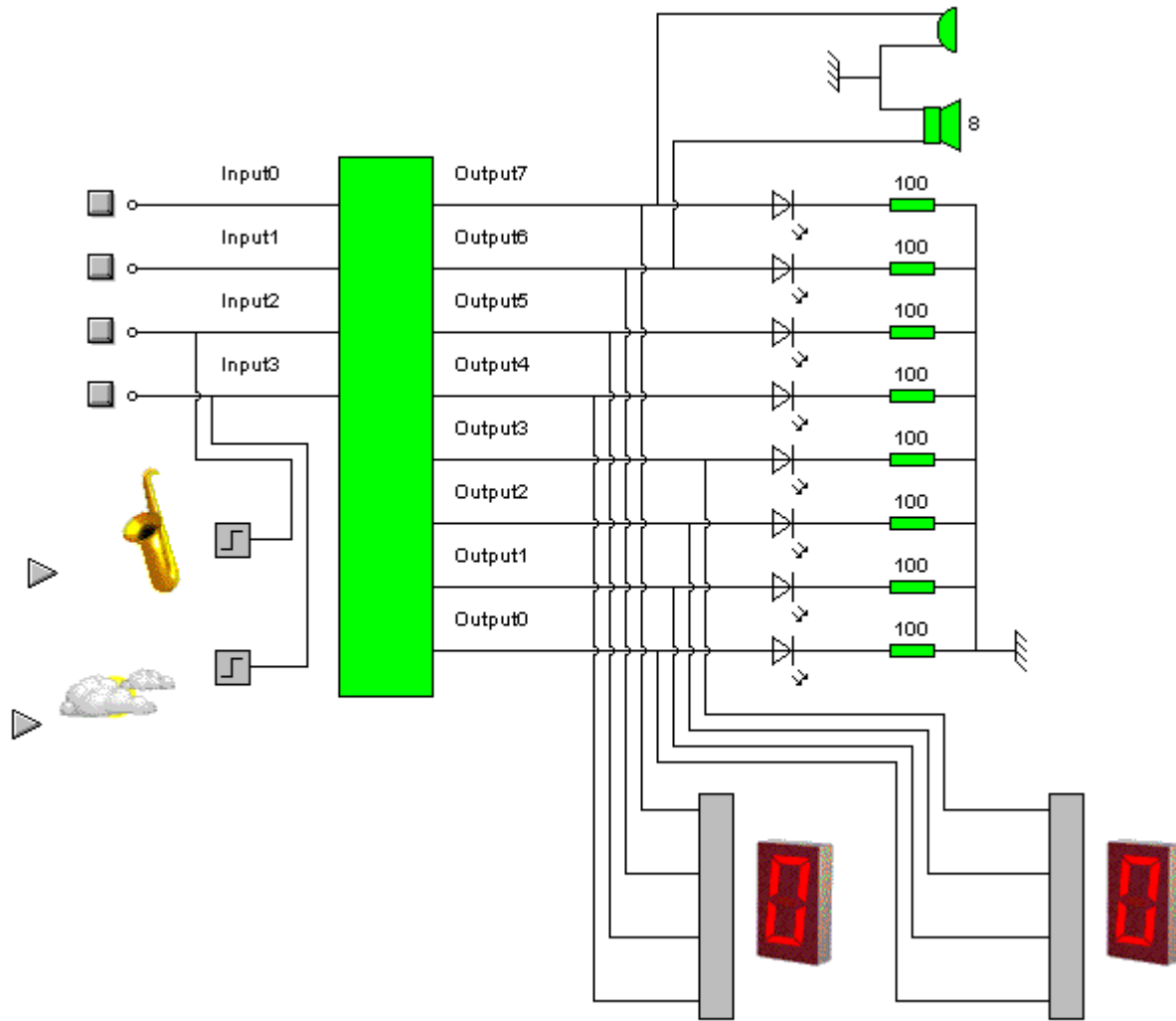
The peripherals can also be accessed in a BASIC program via the system variables. Used in this way there is no 'sharing' or multiplexing of pins and peripherals can be individually controlled. See **HIGH** for some examples.

The peripherals can be individually enabled or disabled via the I/O Options dialog. This is only really relevant to Crocodile Technology flowcharts. If peripherals are controlled via system variables there is no need to disable them because there is no sharing of pins.

The file "piclab.cyt" in the "programmer" directory contains a Crocodile Technology template of the peripherals connected to PICLAB Programmer.

If using PICLAB to program generic PICs destined for other boards, then note that `Port A` is a 4-bit input port and `Port B` is an 8-bit output port.

Schematic of the PICLAB Programmer peripherals



PICBOT includes the following peripherals:

- \* 2 LEDs
- \* 2 infra-red obstacle sensors
- \* 2 motors
- \* 2 pushbuttons
- \* piezo speaker
- \* analogue sound sensor
- \* analogue light sensor
- \* expansion connector with 4 digital inputs, 1 digital output and 2 analogue inputs

The left and right obstacle sensors read high when they detect an object within range on the corresponding side of PICBOT.

The motors are controlled by two pins each. A motor is driven forward by setting the first pin high and the second low, or backward by setting the first pin low and the second high. Setting both pins low stops the motor.

The STOP button is only available to programs if the "STOP button" option is enabled in Compiler Options.

The analogue sound and light sensors return values in the range 0 to 255. The louder a sound or the brighter the ambient light level, the higher the reading.

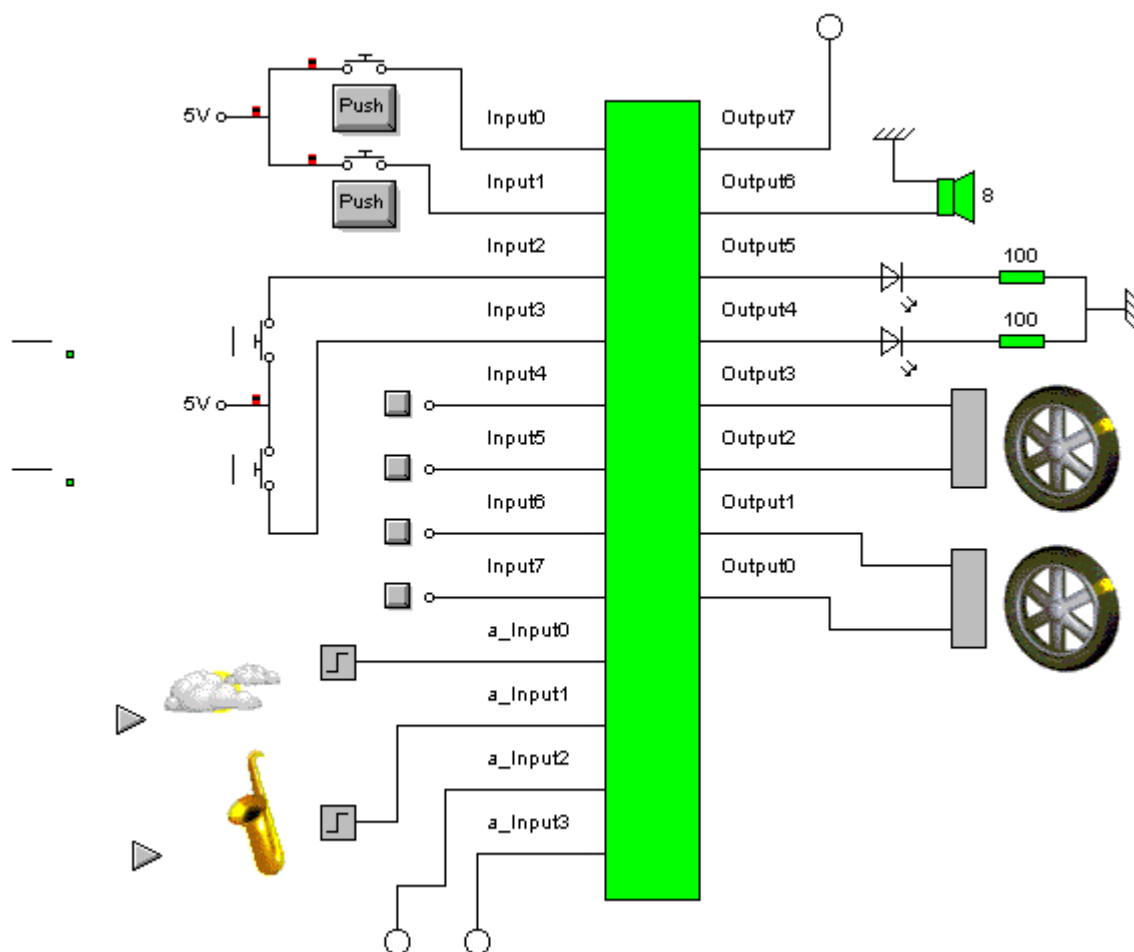
The speaker is a gated device, setting its pin high sounds a tone (the system variable *Note* determines its frequency).

Seven of the pins are connected to the expansion connector for interfacing with add-on boards.

The peripherals can also be accessed in a BASIC program via the system variables. See **HIGH** for some examples. This is the preferred manner of controlling peripherals. It is easier to remember the names of peripherals than which pins they are connected to, and it makes a program self-documenting.

The file "picbot.cyt" in the "robot" directory contains a Crocodile Technology template of the peripherals connected to PICBOT.

Schematic of the PICBOT peripherals



## Compiler Error Messages

---

If you encounter an error then examine the assembler code produced by the compiler (see Compiler Options for how to generate assembler code). This will indicate the context of the error. Errors are also listed in the Errors window along with their line numbers.

You shouldn't normally encounter error messages when downloading a Crocodile Technology flowchart to PICLAB because the flowchart must be syntactically correct. Exceptions are when the ROM memory of the PIC is exceeded (flowchart too big) or the RAM memory is used up (too many program variables).

Here is a complete list of compiler error messages and their causes:

Label not found	Branch or subroutine call to a label that doesn't exist.
Variable not found	Not normally encountered.
Label defined twice	Every label must be unique.
Variable defined twice	A variable has been DIM'ed after its first use. Move DIM statements to the program top.
Too many labels	Maximum of 32 labels in an ON statement.
Unmatched NEXT	No corresponding FOR statement.
Too many FOR-NEXT loops	Maximum of 64 loops in a program.
Invalid pin	Must be a number between 0 and 7.
Invalid address	Must be in the range 0 to 63 for PICLAB Programmer (0 to 127 for '62X), or 0 to 95 for PICBOT.
Invalid size	Variable size must be 1, 8 or 16 bits.
Syntax error	Typically misspelled BASIC keyword.
Invalid operation	Instruction not available (PICBOT only for example).
Number too big	Greater than 255 or less than -256 for example.
Shift too big	Must be between 0 and 16.
Not a constant expression	Expression must not contain any variables.
Divide by 0	Attempted to divide by zero.
Floating point not supported	Attempted to use a number containing a decimal point.
Fatal error	Not normally encountered.
Internal error	Not normally encountered.
End of file	Not normally encountered.
Symbol table overflow	Maximum of 256 symbols, variables or labels.
Too many variables	RAM memory is full. Make variables smaller or double up their use.
Program memory full	Program memory is full. Make 16-bit variables into 8-bit unsigned variables.
Out of memory	Close some applications.
RS232 comms not enabled	Enable the option in Compiler Options.

An additional error you should be aware of (and one that Crocodile Technology will not trap) is a stack overflow. In practice you should limit nested subroutine calls to 4 levels.

## Software Updates

---

Download the latest version of PICLAB from the MadLab website ([www.madlab.org/piclab](http://www.madlab.org/piclab)).

The software is provided as a self-extracting compressed file. Simply run the executable and select the directory you wish to install the files into. See Installing PICLAB.

The file "piclab.hex" in the sub-directory "programmer\firmware" is the latest version of the PICLAB Programmer firmware. You can upgrade your PICLAB Programmer by programming a blank device and then swapping the newly-programmed chip with the existing chip. Make sure the Tester Board option is not ticked in the Options Menu, then load the file "piclab.hex" and download it into a 4MHz PIC16F84 device (XT oscillator, power-up timer on, watchdog timer on). Match the notch in the chip with the notch in the socket when replacing the existing chip.

The firmware in PICBOT is updated automatically by PICLAB. Make sure you have the latest update of PICLAB.