manual

iL_BAS16

BASIC compiler for PIC microcontroller

compiler release 5.5 - xx

compiled: November, 18. 2004

printed:26.11.04

Ing.Büro Stefan Lehmann
Fürstenbergstraße 8a
D-77756 Hausach

Tel. ++49 (0)7831 452
Fax ++49 (0)7831 96 428
eMail: SL@iL-online.de
www.iL-online.de

Inhaltsverzeichnis

Introduction

This manual is for all four **iL_BAS16** compiler versions which are available (iL_BAS16SES,iL_BAS16STD, iL_BAS16SEP and iL_BAS16PRO). The professional version **iL_BAS16PRO** supports almost every device of Microchip's PIC12Xxx and PIC16Xxx family. **iL_BAS16PRO** has been developed for professional use. **iL_BAS16STD** has a perfect price performance ratio and is excellent for hobbyists. **iL_BAS16SES** and **iL_BAS16SEP** are low price starter versions for those who hesitate and don't want to spend too much money. Our fair upgrade policy minimizes your risk. For upgrading to the higher version the price of your old version reduces the price of the new version (upgrade STD to SEP is not available). All supported PIC devices are listed in appendix III "supported devices"

Microchip's microcontroller family PIC12C5xx, PIC12C6xx, PIC16C5x, 16C7x, 16C8x and 16F8x replaces a lot of older circuits with standard logic ic, but also older systems whoes microprocessor runs out of production or isn't up to date. The instruction set of these microcontrollers is small, powerful and easy to learn. This garantees the phenomenal success. We, the Ingenieurbuero Lehmann, started in designing developping tools for these microcontrollers. The goal was to create a reasonably priced tool, and we got it. All our daily experience in PIC applications since 1992 results in iL_BAS16. High performance and an excellent code generator turns this compiler into a usefull and professional tool. New PICs will be added to the long list of supported devices. In case of trouble we will help you with a qualified support.

The BASIC compiler iL_BAS16xxx has been developed under the attention of the small, and sometimes tiny program memory. Questions like: "does it makes sens to use a compiler for such small program memories" can be answered with "yes". Sure there is some code overhead when using compilers. But this overhead could be minimized by code optimization. iL_BAS16xxx uses three different ways of code optimization. First, redundant code is eliminated. Second, only used parts of the runtime library are linked to the final program. Third, the compiler iL_BAS16xx generates various codes. For example: IF VAR=0 THEN generates totally different codes as IF VAR < 1 THEN. The compiler's capability to produce such different codes is not simply done, but needs a lot of knowledge of the PICs and compilers internas .

iL_BAS16xxx is developped for Microchips PIC family exclusivly. Therefore it is optimized to the PICs hard- and software internas.

iL_EDy is an integrated development enviroment for the compiler iL_BAS16 and its features. These are the PIC assembler iL_ASS16, the PIC simulator iL_SIM16 (still DOS) and the PIC programer iL_PRG16. Along with the compiler versions iL_BAS16SEP and iL_BAS16PRO the program iL_PAGE0.EXE is delivered. This module is necessary to calculate the long calls and long jumps in PIC devices that have more than 2k words of program memory. iL_EDy has 8 pages. The first page is for the main program. Its name is used by iL_EDy for calling the compiler, assembler etc. Page 8 is for the error list file. All other pages are for include files or the documentation during developing.

The PIC BASIC compiler iL_BAS16 is optimized for Microchips PIC12Xxx and PIC16Xxx family. Its characteristics are many powerful instructions. The generated code is compact and fast. So in most cases you don't need assembler inline to accelerate execution time.

The compiler generates waiting loops for e.g. timing of serial i /o. You don't need to calculate the exact values for these loops, this is the compiler's job. To do that, you tell the compiler the frequency of the xtal build in your application. All calculations the compiler does are optimized to 4 MHz. Using other xtals, especially with lower frequency difference of 10% or more, may occur.

!!! IMPORTANT !!!
Because of truncation calculation rounding errors will occur. These errors have an effect to measuring times or output frequencies.
!!!!

Installation for Windows

Installation for Windows:

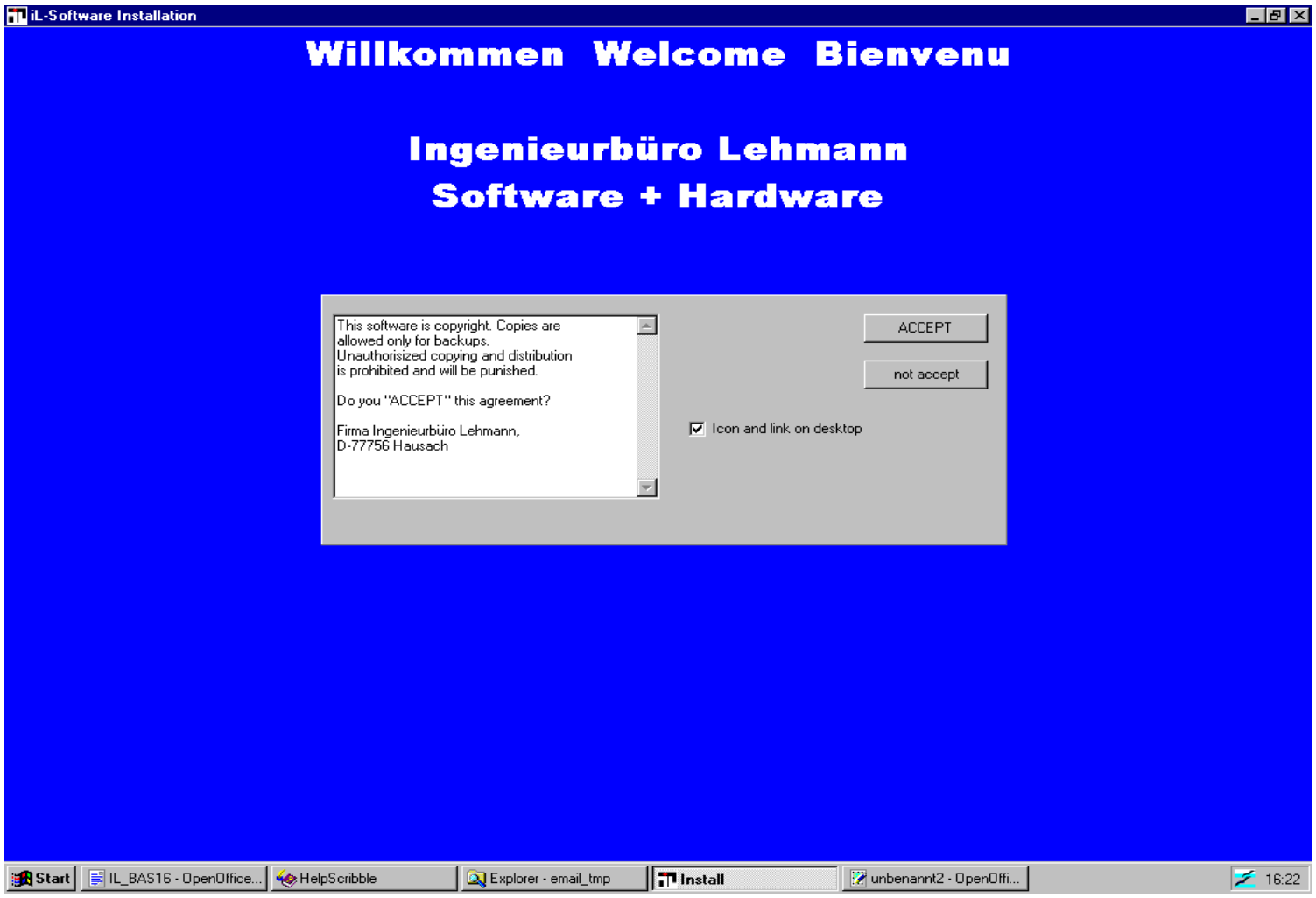
iL_BAS16 will be delivered on CD or email. The files are unpacked on the CDs except the Micochip data-sheets in the sub-directory DOC. In case of email-delivering all files are packed and the data sheets are missing. They can be downloaded easily from the internet. Because the names of the data-sheets are only composed of numbers there is no logical connection between the data-name and the described PIC. Which file you have to load down from internet you can find at the file PIC_LIT.LST.

If the software is available in a packed form, it has to be unpacked at frist. Some unzip programs support the start of packed files. In case you shoulnd't have this opportunity unzip a new temporary subdirectory. The CDs don't have any auto boot. To start the installation open the explorer by clicking on INSTALL.EXE.

At the end all files and subdirectories will be copied in a selected directory. In case of having activated "automatical link on desktop" it will be created now. There will be no other modifications or entries exept the subdirectory and the link. Also the modification file will stay in this local directory. So the compiler is easiy been removed out of the computer by deleting the directory and the link. Then check if all files are not "read-only".

The following windows appear.

Installation for Windows (cont.)



You will start the program-IDE by clicking at your link on your desktop or the file iL_EDy.EXE out of the explorer. You can

Installation for DOS

To install the Compiler iL_BAS16 for the operating system DOS the files can be easily copied into a directory which you have created already. Call editor ED16x.EXE. This Editor permits to start the compiler iL_BAS16, the assembler ASS16 and , if present, the programing device iL_PRG16 and the simulator iL_SIM16 with the help of key ALT and a function key. Of course you can use your own editor. In case it permits a start of external programs, you can start this program by a batch file. If necessary you have to adapt these batch files to their enviroment. Please pay attention that all programs, even your basic source program, have to be in the same directory (at the moment only DOS-version). If you use the supplied editor ED16X, you can start the compiler out of its surface. To do so please press key combination ALT-F4. Now a batch program will be started, which at first compiles the compiler with the actual file in the editor. If necessary iL_PAGE0 searches for GOTOs and CALLs beyond page bounderies. Then the assembler will be invoked immediately, which assembles the new created SRC-file. If both compilation runs are done without mistakes you can activate the simulator (if present) with the help of the key combination ALF-F2, to test the program. Beside testing the compiled programs in machine level you also can carry out a high level language debugging. In doing this you need the simulator iL_SIM16 version 5.2 following.

Necessary files among others by DOS-installation:

absolutely	***necessary***	***dokumentation***	***example programe***	
Ass16c.bat	Baudcalc.exe	Ass16_gb.pdf	Adinp.bas	Asm.bas
Bcom16.bat	Default.equ	Bas16_gb.pdf	Data.bas	Doze.bas
Dpmi16bi.ovl	Ed16x.exe	Ed16x.doc	Dtmfout.bas	Err.bas
Edix16x.exe	Edx16x.exe	Il_ass16.pdf	Fornext.bas	Freqin.bas
Il_ass16.exe	Il_ass16.txt	Il_bas16.doc	Gosub.bas	High.bas
Il_bacom.exe	Il_bas16.equ	Il_bas16.pdf	I2cslave.bas	Ifthen.bas
Il_bas16.exe	Il_bas16.idt	Il_prg16.pdf	Inkey.bas	Input.bas
Il_bas16.pic	Il_bas16.txt	Il_sim16.pdf	Interrpt.bas	Ltc1286.bas
Il_prg16.cfg	Il_prg16.exe	Sim16_gb.pdf	Pwm.bas	Serin.bas
Il_prg16.hlp	Il_sim16.txt	Serout.bas	Wuerfel.bas	
Prg16c.bat	Rtm.exe			
Sim16c.bat				

You also can activate the compiler out of the DOS-surface. Enter the following command:

iL_BAS16 filename

Do not enter any file extension. The compiler converts the basic source text into an assembler program. This new created file contains the extension .SRC and can be translated to the main machine code with the help of the assembler.

When using the professional version iL_BAS16PRO and iL_BAS16SEP you now have to start **iL_PAGE0 filename**

Invoking the assembler by calling **iL_ASS16 filename**. You don't have to indicat any extension by giving the file name.

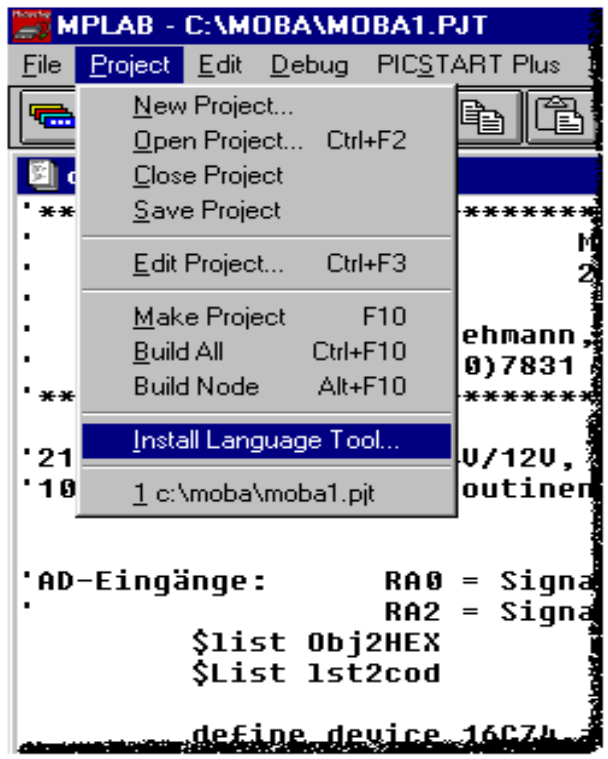
Hint:

ED16X can't handle any file out of the directory limits that means the editor has to be in the same subdirectory as the file to be edited. The full version EDX doesn't have that restriction and also makes full use of the complete heap storage. That makes it possible to use files with up to 6500 lines.

Integration in MPLAB (5.2 only)

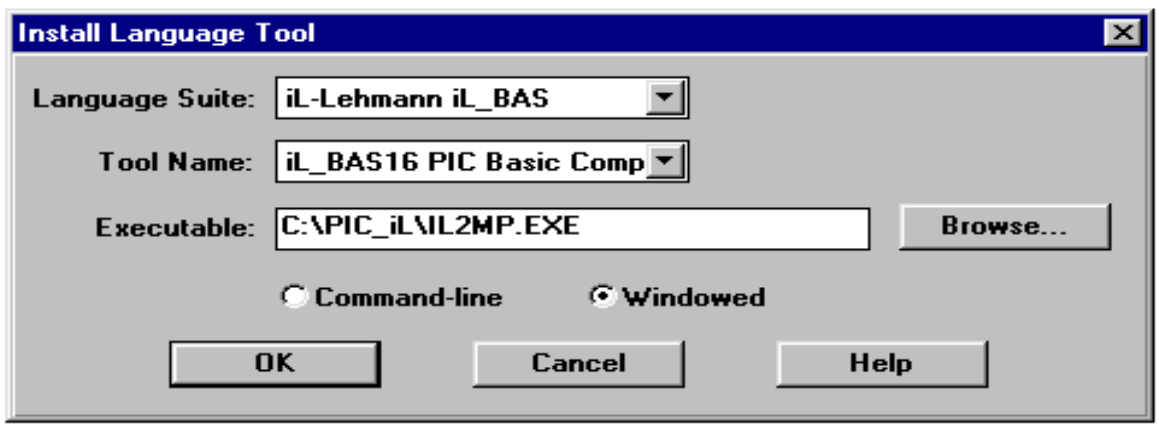
Now iL_BAS16 can be easily annexed to the developement enviroment MPLAB (from version 5.2 following) of Microchip.
Carry out following steps:

- Install MPLAB starting from version 5.2, if you haven't already done.
- Install Windows-version of iL_BAS16.
- Copy the files iL_BAS.MTC and TLiLPic.INI manually out of the directory of the compiler into the directory of MPLAB.
- Start MPLAB.
- In case you haven't already installed a project, do it now.
- Select **Project** and topic **Install Language Tool...**



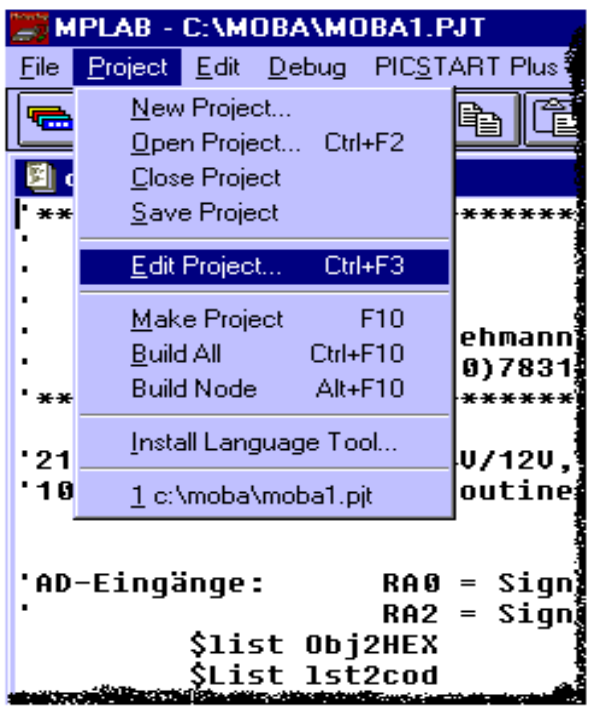
- Chose **Language Suite** and select **iL-Lehmann iL_BAS.**
- The right **tool name** appears self-employed.
- At **Executable** enter file **iL2MP.EXE** incl. path name, in which the compiler is installed.
- Activate **Windowed** (very important!).
- Confirm the input with **OK.**

Integration in MPLAB (5.2 only) (cont.)



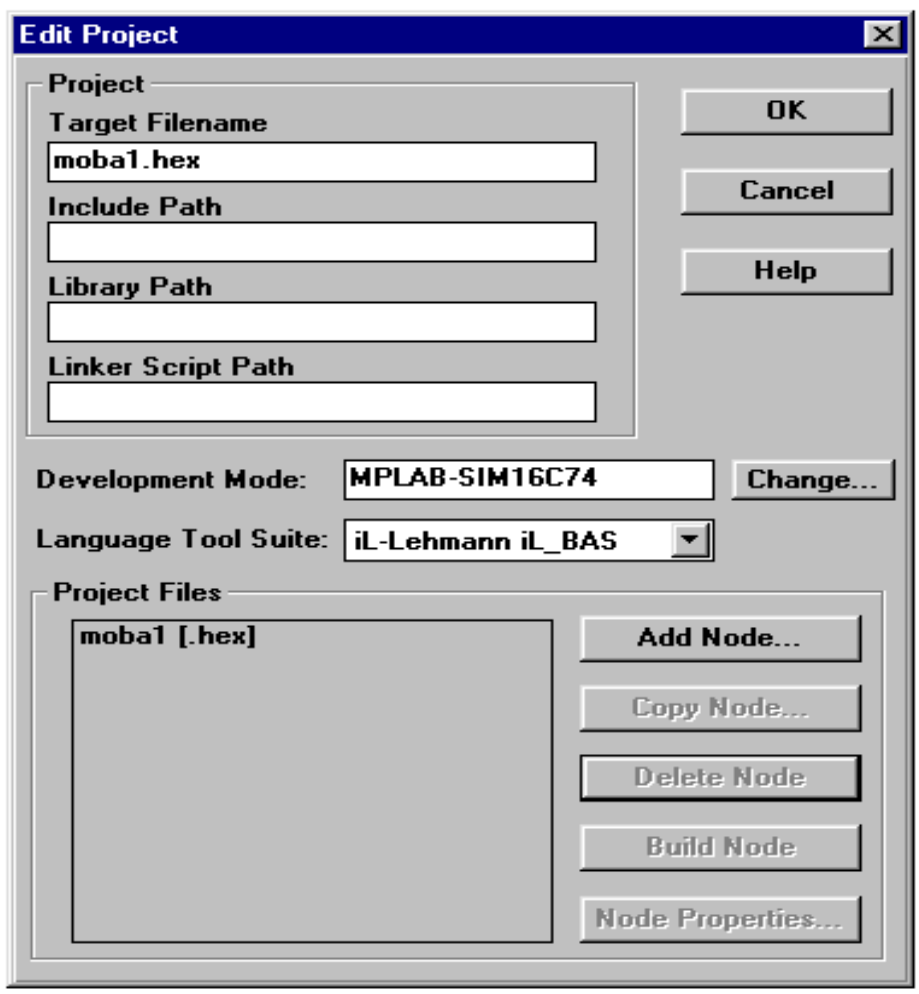
- Choose under **project** the point **Edit Project...** .

Integration in MPLAB (5.2 only) (cont.)



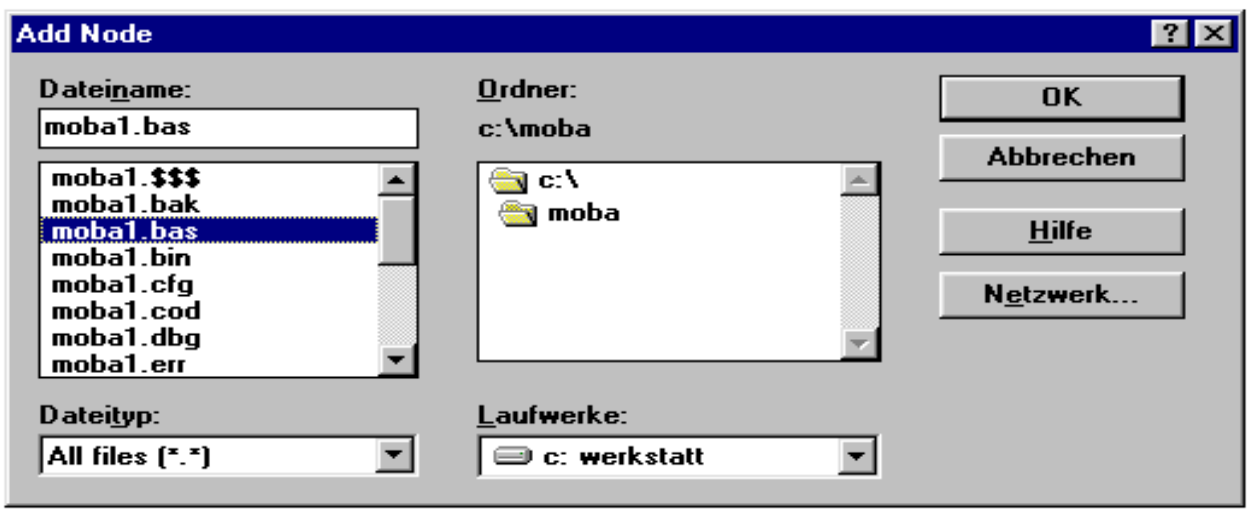
- In case you already have installed a project in the window, **Target Filename** appears the corresponding Hex-file.
- The boxes **Include Path, Library Path and Left Script Path** remain empty.
- When ???? choose, wether you just want to work with the simulator or also with the emulator.
- In the **Language Tool Suite** again you have to choose **iL-Lehmann iL_BAS**
- In the window **Project Files** you can find the file which is noted under the target filename. Press **Add Node...** .

Integration in MPLAB (5.2 only) (cont.)



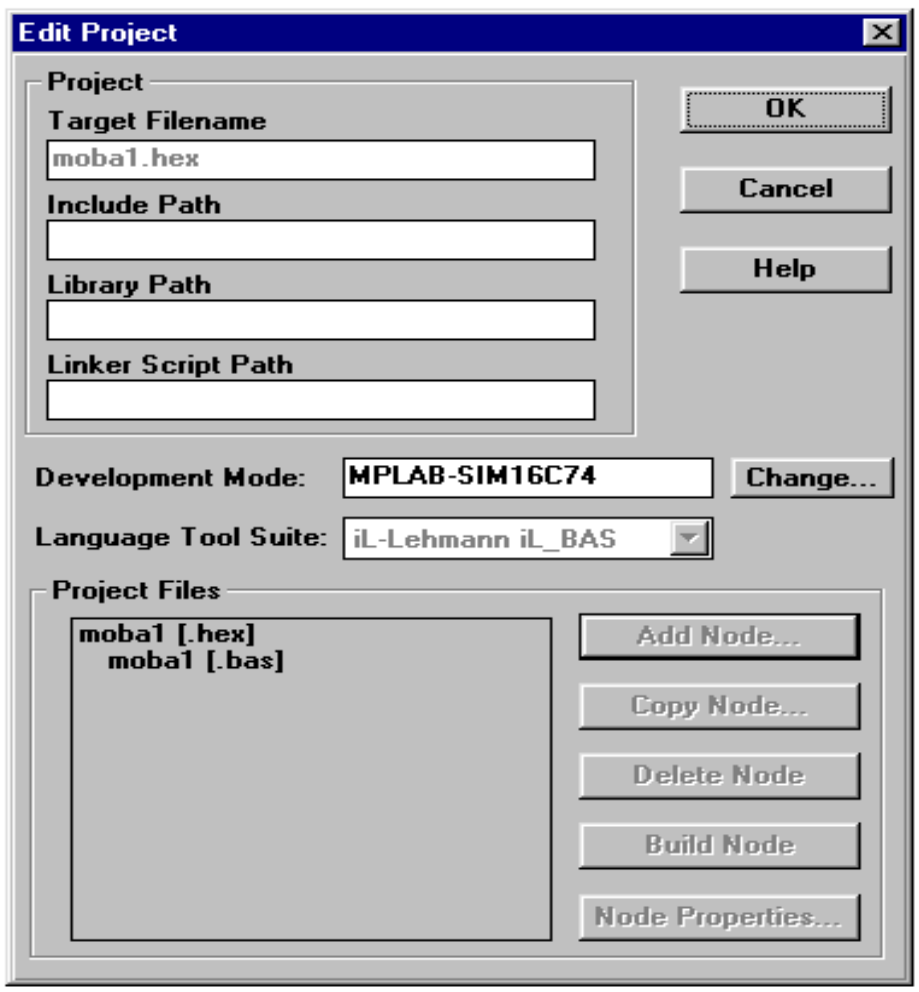
- In the box **Filename** insert the file which it noted under the target filename with the extension **BAS**

Integration in MPLAB (5.2 only) (cont.)



- This input apperars in the window ProjectFiles.

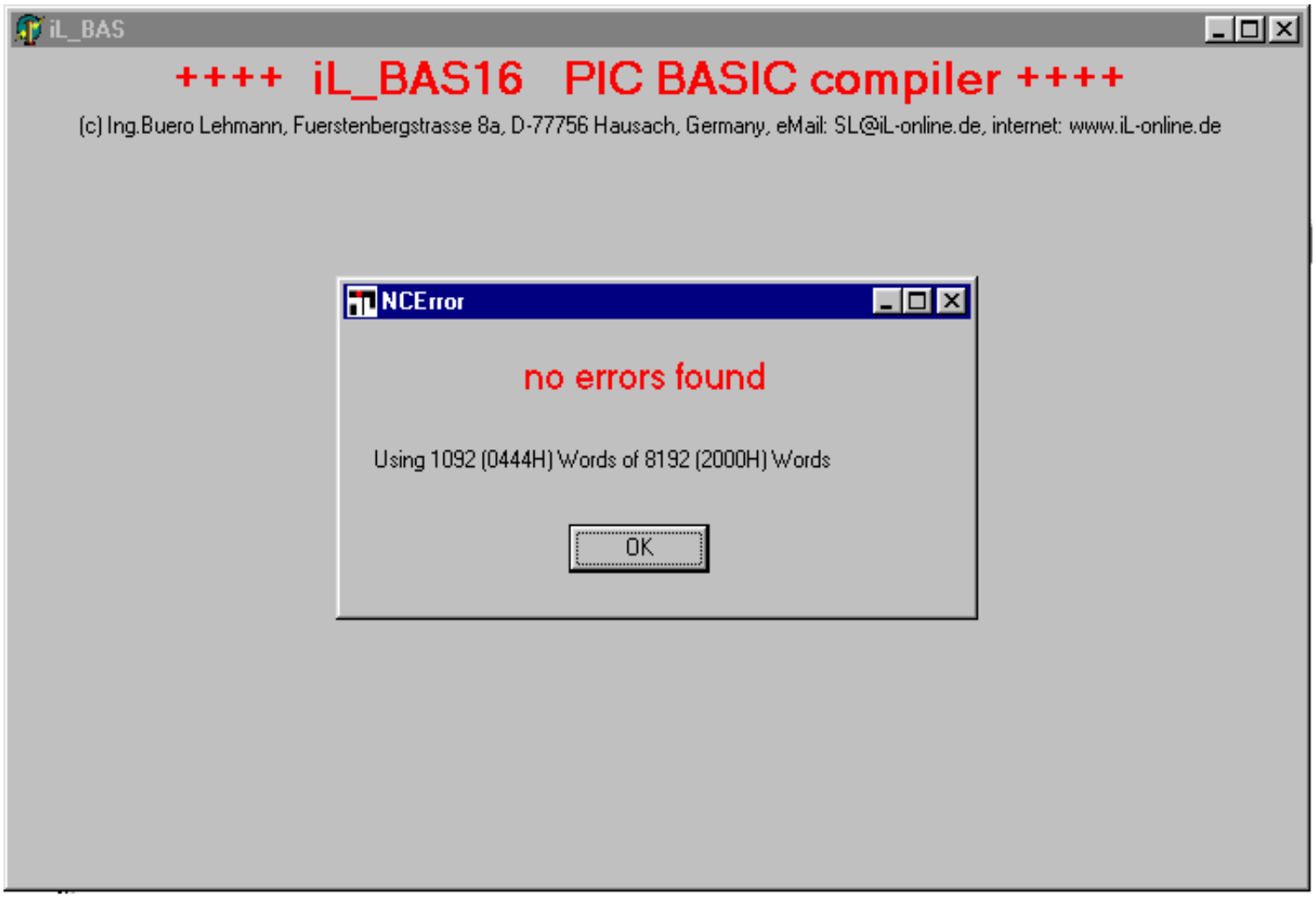
Integration in MPLAB (5.2 only) (cont.)



- Confirm with **OK**. The integration of iL_BAS16 in MPLAB is finished.

- Indicate the switch **$LST2COD and $OBJ2HEX** in your **BASIC-source-text.** With this you create a COD-file, which MPLAB needs for symbolic debugging.
- If you activate **Make Project** , **Build All** or **Build Node** under **Project** the compiler will be started automatically.

Integration in MPLAB (5.2 only) (cont.)



When compiling is successful a MPLAB system window will appear.

Integration in MPLAB (5.2 only) (cont.)

If not, it means searching mistakes !!!

brief description of iL_BAS16

Legal questions:
This shareware version corresponds to a full version with a limitation of the module to the PIC 16C83. The shareware version is allowed to be copied completely and to be distributed to other customers. The use in commercial business is limitated to two projekts. So this software can be tested extensively.

The complete version of this programm is liable to the copyright and is therfore not allowed to be copied. Only licencees are allowed to make backup copies. Contraventions are criminally punishable.

The Compiler:
This compiler translates a BASIC program into machine language. At the same time it tries to optimate the result. Redundante commands will be removed as far as possible. However it could be possible to optimize the result manually because the check up will take part just within one function. Is there a redundant code between two functions it won't be removed.The runtime routine will only be included to the program when a corresponding call is really done.

The actually availabe standard version iL_BAS16STD only supplies the PIC12C508, PIC16C54 and 16C55, PIC16C64, PIC16C71 and PIC16C84 / 16F84. The calculation of program jumps over the page boundaries is not supported. You also can implement a time base at these interrupt capable modules. To insert own interrupt service routines is possible as with the professional version.

The provessional versrion supports the PIC 12C508, 12C509,16C53 - 16C58, 16C61 - 16C66, 16C71 - 16C74, 16C83 - 16C84, 16CF84 and 16F87x. Otherones will follow. Because implemantation of devices with different hardware resources is quite complicate it will be realised successively.Here I'd like to encourage you to let me know your kind of solutions. The additional RAM-memory of the 16C57 can be used as data-array (see variables).

Updates and Upgrades:

Like other software products the functions of the BASIC-Compiler iL_BAS16 will be modified and completed permanently. Our cost-effective UPDATE / UPGRADE-service enables you continually to work with the recent program version.

You can find our latest news about command extension, improvments, etc. on our homepage. The Update service will be transacted usually by email. All clients can update their products within 6 month after bying after previous consultation. After that period the fees for an Update are between 5,- and 30,- Euro a piece. To Upgrade from standard to professionel version will just cost the differential amount.

Fundamentals:

Important !!!

Following restrictions are valid for the iL_BAS16:

For one line only one order or instruction
Exception REM respectively (').
Use of double point is not possible
Exception LABEL (branch mark).
****
No calcualtions within a statement or a function
(e.g. INC a * 2)
only variables and constants.

brief description of iL_BAS16 (cont.)

****

Arithmethic and logical links are not allowed in comparing-operations of IF- inquiries.
They have to be made before an IF-instruction.

****

When using symbols or labeles (also variable namens, branch marks, constant names) then:

****

they must begin with letters and be at least 2 character long

****

the first 16 characters are significant

****

except the first characters you also can use numbers and "_"

****

mutated vowels are allowed

****

There's no distinction between upper- and lowercase letters.

****

Never change the OPTION register!

First time user problems

Here you can find most of the questions of beginners and answers. Other questions and answers you can find at chapter

Problem 1

How can I use together the BASIC-Compiler iL_BAS16 and the programing device of Microchip?

In the Basic-program you have to set the switch $LIST /M_OBJ. The created HEX-file will be loaded into the programing software of Microchip. Then the configurations (WDT, OSC, e.g.) must be set manually.

Problem 2

Some pins of the PIC 12C508 don't work correctly.
It's important to know that special pins of some modules are only allowed to work in one special direction (=dedicated pins). There's also the case that changes also have to be done in another registers beside the TRIS-register. This doesn't happen automatically by the compiler. You have to program it. Some things can be explained by reading the data sheets of the corresponding processor.
Dedicated pins are also found at other pics!

Project - file

A project consists of one or more (up to 8) files. By opening a new project a new project file will be set up in the directory of the editor iL_EDy. The new file includes beside the 8 file names also their attributs.

Hint !!!

At the moment the project name shouldn't have more than 8 characters (DOS-convention), because the simulator is a DOS-program which can not handle files and directoray names with more than 8 characters.

At the same time the new project name will be proposed as the new file name for the first map. The extension which is proposed is BAS. The name of the first map will be handed over to the compiler and assembler without the extension. Thess programs complet the name with BAS (compiler iL_BAS16), SRC (Assembler iL_ASS16, Pagemodul iL_PAGE0), LST (simulator iL_SIM16) and OBJ (programing device iL_PRG16).

First time user problems

Here you can find most of the questions of beginners and answers. Other questions and answers you can find at chapter FAQs.

Problem 1

How can I use the BASIC-Compiler iL_BAS16 together with the programing device of Microchip?

In the Basic-program you have to set the switch $LIST /M_OBJ. The created HEX-file will be loaded into the programing software of Microchip. Then the configurations (WDT, OSC, e.g.) must be set manually.

Problem 2

Some pins of the PIC 12C508 don't work correctly.
It's important to know that special pins of some modules are only allowed to work in one special direction. There's also the case that changes also have to be done in another registers beside the TRIS-register. This doesn't happen automatically by the compiler. You have to program it. Some things can be explained by reading the data sheets of the corresponding processor.

Editor

When starting a new project it is useful to define the project's name first. After writing the program you can save them in different directories under the main topic FILE. The main file must be in the first page because its name is used as a parameter when the compiler, simulator etc is invoked. Page 2 to 7 are for include files or for documentation. The error file is listed in page 8.

Administration of all projects is done by project files which are in the same directory as iL_EDY. When starting a new project iL_EDy generates the name of the first page automatically. This name consists of the project's name plus the extension BAS. It can be overwritten when it is saved for the first time.

Important and often used topics can be reached by separate buttons. If you use the topic "PassCompiler" you must invoke each module in the right order. 1.) iL_BAS16; 2.) iL_Page0 if SEP or PRO version and 3.) iL_ASS16. Topic "Compiler" does all these actions automatically in the right order. It is better to use this function.

ATTENTION!!!
Invoking the compiler, simulator, programer etc means that all files will be saved automatically. Also the project files will be updated.

Since January 2004 a subset of WORDSTAR compatible key functions are available.

CTRL-Q and CTRL-K

CTRL-K 1..9 set/clear bookmark
CTRL-K B mark start of block
CTRL-K C copy block
CTRL-K K mark end of block
CTRL-K S save open window
CTRL-K T mark word
CTRL-K V move block
CTRL-K Y delete block

CTRL-Q 1..9 jump to bookmark
CTRL-Q A search and replace
CTRL-Q F search
CTRL-Q L undo

CTRL-L search next
CTRL-N insert line
CTRL-U stop seach or replace function
CTRL-Y delete line

Product info

Four versions are available:

Standard version:	iL_BAS16STD
Professional version:	iL_BAS16PRO
Spezial Standard (1)	iL_BAS16SES
Spezial Professional (2)	iL_BAS16SEP

(1)
only for 16F84 and 16F628

(2)
only for 12F629, 16F84, 16F627, 16F628 and 16F877):

BASIC instructions overview

ADDELAY(only 16C7x and 16F87x)
ADINP(only 16C7x and 16F87x)
ASM
BINTOASC (also CONASC)
BINTOBCD
BINTODEC (also CONDEC)
BITPOS
CALVAL (only PICs with internal and activeded RC oscillator)
CLOCK and CLOCK1(not for 12C5xx, 12E5xx and 16C5x)
CONASC
CONDEZ
CURSOFF
CURSON
DATA(not for 12C5xx, 12E5xx and 16C5x)
DEC
DELAY
DOZE
DTMFOUT
EEDATA
END
ENDASM
ERR
FOR-TO-NEXT
FREQIN
GOSUB
GOTO
HIGH
I2CDELAY
I2CHARDS (only for PIC with internal I2C module)
I2CINIT
I2CRD
I2CREAD
I2CSLAVE
I2CSP
I2CST
I2CWR
I2CWRITE
IF-THEN-ELSE
INC
INKEY
INP
INPUT
INTERRUPT
INTEND
INTPROC
LCDCLEAR
LCDDELAY
LCDINIT
LCDTYPE
LCDWRITE
LET

BASIC instructions overview (cont.)

LOCATE
LOFREQ
LOOKDN
LOOKUP
LOW
ON-GOSUB
ON-GOTO
OUTP
OUTPUT
PEEK
POKE
PRINT(only 16F87x)
PULSIN
PULS_IN
PULSOUT
PWM
RANDOM
RCTIME
READDATA
READ(only 12Exx, 16X8x and 16F87x)
REM
RES
RESTORE
RETURN
REVERS
SERIN
SEROUT
SET
SETBAUD
SLEEP
SOUND
SWAP
TOGGLE
TRIS
TXDDELAY
VARPTR
WAIT
WRITE(only 12Exx, 16X8x and 16F87x)

Compiler switches

Compiler switches or compiler directives are useful to controll the work of the compiler. These are instructions which tell the compiler e.g. the oscillator frequency (XTAL) of the target hardware. Knowing this, the compiler can calculate the values for e.g. the delays in the BASIC instruction WAIT or SERIN etc. for exact timing.

$CCON und $CCOFF turns on and off the monitoring of an overflow in a 8 bit addition or subtraction

$IF $ELSE $ENDIF enables a conditional compiling

$INCLUDE links an include file to the main file

$LIST controls the layout of the OBJ, BIN and LST files

$LRANGE defines the range around the page bounderies for iL_PAGE0

$LST2COD creates a COD-file, which is necessary for the symbolic debugging of (among other things) MPLAB.

$NCALDEF suppresses the insertation of a calibration default value

$OBJ2HEX creates beside the OBJ-file the HEX-file too

$OLDVAR compiler works with the old variable declaration (do not use anymore)

$WDTUSR disables the automatic insertation of a CLRWDT after each BASIC keyword. The user has to do it himself.

DEFINE variable, constant defines symbols, constants, etc
DEFINE device defines the cpu type
DEFINE other defines stack, key table etc.

DATE inserts the actual date into SRC and LST file

TIME inserts the actual time into SRC and LST file

XTAL crystal frequency of the target hardware

$CCON und $CCOFF

8 bit addition/subtraction is not realized with the help of routines in the runtime library, but it is code directly. In case of an overflow error the overflow bit in the ERR variable is set. Often, this is not necessary. $CCOFF suppresses that overflow check. To give you the choice to switch on/off this overflow check you can set these compiler switches anywhere in your program to optimize code length and execution speed.

You can use $CCON and $CCOFF in your program as often as you like to. So a deliberated activation at a special position in the program is possible while the rest of the program will be generated compactly. The code created after $CCOFF will not execute the carryover. When $CCON is arrived the carry-bit will be copied into the ERR-variable.

8 bit multiplication/division and all 16 and 32 bit arithmetics are not effected by $CCON and $CCOFF. In these cases the overflow bit is checked anyway.

$IF $ELSE $ENDIF

The compiler switches$IF *cond*, $ELSE and $ENDIF allow a conditional comiling. So you have the possibility to let translate the programcode between $IF and $ELSE respectively $ENDIF by setting the condition *cond* . Not having set this condition the programcode just will be translated between $ELSE and $ENDIF. *cond* is a constant defined by DEFINE. When the value is 1 or another value <>0 the condition is true and the $IF bloc will be carried out. The condition will be not true if the value is 0. In this case the $ELSE bloc will, if available, be worked out.

Example 1:
The line between $IF and $ENDIF shall be executed.

```
$IF switch
        LET ...
$ENDIF
```

Example 2:
The line between $ELSE and $ENDIF shall be executed.

```
$IF switch
        LET value = 10
$ELSE
        LET value = 20
$ENDIF
```

Attention !
Nesting is not possible.

$INCLUDE

$INCLUDE allows including a file during compilation. So you can divide good-sized projects into managable file sizes. A lot of those good-sized projects become complex if function are far-scattered all over the whole project. Also a lot of functions are already been tested or shall be taken over out of another program. Will all these function groups e.g. declaration of variables, in- output- routine, and so on be put together in one module, whereas each module is availabe as a seperate file, even the most good-sized projects will be easy to follow and easy to handle.

If the filename has no extension INC is assumed. If path and drive is missing these of the main file are assumed.

The DEVICE-instruction including the specification of the type of processor **must** stand before the first $INCLUDE otherwise you will get the error message "unknown CPU".

$LIST

$LIST is used to route the assembler which translates the by the comiler transtated program into an OBJ-file. Sometimes it is necessary, to exercise an influence on the assembler for example to configure the OBJ-file the way that the PICSTART-plus programming device of Micochip is able to read the file correctly (iL_PROG16 is using an extented format).

$LIST /M_OBJ creates a conveniant file for the PICSTART.
$LIST INHX8 creates an OBJ-file with Intel-Hex8-format (default).
$LIST INH16 creates an OBJ-file with Intel-Hex16-format.
$LIST C=xxx defines the number of characters per line in the LST-file.
$LIST BIN additionally creates a binary file (is needed by some programing devices).
$LIST BINX creates a binary file where high- and low-byte are swapped.
$LIST OBJ2HEX instead of OBJ the hexfile will get the file-ending HEX.

$LRANGE

$LRANGE is used to induce the compiler module iL_PAGE0 at pics with program memory which are bigger than one page (see program-pages) that relevant arears around the page boundary is individually adjustable. So it determins how huge the area around the page boundary shall be where a GOTO will be transformed into a LJMP respectively a CALL into a LCALL. This individuel adaption became necessary because now and then there were constellations in the compilation which sometimes caused a crash of the program because of the narrow boundaries (16 before and 16 after the page boundary) iL_PAGE0 couldn't carry out the transcription into LJMP or LCALL. If there is a strange attitude of the program when it gets bigger than one page it is helpfull to use this switch and set the value to 32 or 48.

$LRANGE 48

Important !!!

Of course the possibilities of $LRANGE are limited. Depending on the sequence of the program, it may be, you have to choose a very huge value. For example when the entry point of a huge subroutine is on the same page the return however on the following one. $LRANGE has to be choosen very huge because of an unhandily positioned subroutine in the memory. That leades to a waste of resources (here program memory space). A way out of this dilemma offers a jumptable at the beginning of the program. The trick is letting have the subroutine only two lines and so prevent an overflow of the memory limit within the subroutine. An example will show this:

```
'As near as possible at the beginning of the program:
'After the DEFINE-instruction but before
'the first BASIC-commands
                GOTO  MAIN    'overjump the following UPs
 UP1            GOTO  UP1X
 UP1Z           RETURN
 UP2            GOTO  UP2X
 UP2Z           RETURN
 UP3            GOTO  UP3X
 UP3Z           RETRUN
                ...
 UP1X           do something
                GOTO  UP1Z
                ...
 UP2X           do something
                GOTO  UP2Z
                ...
 UP3X           do something
                GOTO  UP3Z
                ...
                ...
                GOSUB  UP1
                ...
                GOSUB  UP3
                ...
                GOBUB  UP2
```

Now it doesn't matter from where the subroutine will be started and where are the jump-in and jump-out because a GOTO respectively LJMP sets the corresponding page bits correctly unlike RETURN. Now it's sufficient to set $LRANGE16. This is a default-value which can be omited.

Also see Program Pages

$LST2COD

the compiler switch $LST2COD creates a COD-file. Beside the program code this file also contains the symbol table and the information for the debugger respectively emulator. If you want to test the BASIC-source-text with the help of the MPLAB you need this file.

Furthermore the compiler must create a HEX-file. You need the compiler switch $OBJ2HEX.

Have the compiler working correctly under MPLAB you must integrate it carefully into the IDE of the MPLAB. Please read corresponding chapter Integration to MPLAB

$NCALDEF

Normally the compiler creates a default-value (80H) for modules with active INTERNEN RC-OSZILLATOR which will be programed into the module if it's a JW-type which highest memory position is deleted. Because some programing devices won't interpret this information correctly and therefore will get a wrong value in the OSCCAL-register you can suppress this automatic working by $NCALDEF (No CALibration DEFault).

see also CALVAL

$OBJ2HEX

Some programing devices or emulator (PICSTART respectively MPLAB) need a HEX-file. The compiler iL_:BAS16 only creates an OBJ-file. The compiler-switch $OBJ2HEX enforces the generation of such a HEC-file.

If you like to work with MPLAB you have to pay attention to following points which are described in chaper Integration to MPLAB

$OLDVAR

iL_BAS16xxx version 4 and below uses other kind of variable declaration than version 5 and higher. To compile old programs with version 5 $OLDVAR must switch back. But take care, it only works with PIC devices which are supported by version 1 to 4. $OLDVAR must be the first command in the program.

It is better to change the old kind of variable declaration to the new one. This is more efficent.

new declaration

Important !!!
Please use $OLDVAR only in projects which were created with compiler version 2, 3, and 4. $OLDVAR only works with those processortypes which generally were defined at that time. Processors which will be added new from version 5 therefore don't work.

$WDTUSR

If watch-dog-timer is enabled by WDT_ON in the DEFINE DEVICE line the compiler generates a CLRWDT after each BASIC command, automatically. There is only one exeption: an bit IF-clauses with a GOTO on itself (e.g. WAIT: IF RA,0=0 THEN GOTO WAIT). In this case the watchdog restarts the PIC if the program waits to long in this loop. To avoid this circumstance you should add a dummy line.

There a also some critical applications in which the author wants to place the CLRWDT commands himself. Do disable the automatic use simply compiler switch $WDTUSR. Keep WDT_ON in DEFINE DEVICE line. Now place the BASIC keyword CLRWDT at the right position in your program to avoid a reset by watchdog.

DEFINE variable, constant

DEFINE defines name of a variable or constant, defines the type of cpu and its configurations.

Variables are defined by a name consists of at least two characters. The first must be a letter, and the address where the value should be stored.

for example:
DEFINE *counter* = $30 AS BYTE
the memory address 30H gets the symbolic name *counter.*

DEFINE *sum* = 49 AS WORD
the contents of the 16 bit variable *sum* is stored in the memory location 31H (low byte) and 32H (high byte).

DEFINE *value* = $40 AS DBLWORD
memory capacities 40H upto 43H will be defined as 32-bit variables with the name *value* (also see DEFINE others).

Constants are defined by a name, consists of at least two characters (the first must be a letter) and the desired value. To give the compiler the ability to see the difference between the symbol of a variable and a constant an "AS CONST" is added.

e.g.
DEFINE limit = 100 AS CONST
the compiler replaces the symbol *limit* by the value 100. *limit* is treated as a byte.

DEFINE *maximum* = $4000 AS CONST
maximum is replaced by 4000H. *maximum* it treated as a word.

DEFINE device

To define the target cpu the DEFINE DEVICE must be used. Additional parameters are put into the configuration word or used for the right initialization.

DEFINE DEVICE *cpu_type, watchdog, code_protection, oscillator-type, ad_converter, misc*

cpu_type (depends on the compiler version and will be update continously)
12C508, 12C509, 12E518, 12E519, 12F629, 12C671, 12C672, 12E673, 12E674, 12F675, 16C505, 16C53, 16C54, 16C55, 16C554, 16C556, 16C558, 16C56, 16C57, 16C58, 16C61, 16C62, 16C62A, 16C620, 16C621, 16C622, 16E623, 16E624, 16E625, 16F627, 16F628, 16C63, 16C64, 16C64A, 16C65, 16C65A, 16C66, 16C66A, 16C67, 16C67A, 16C71, 16C710, 16C711, 16C715, 16C72, 16C73, 16C73A, 16C74, 16C74A, 16C76, 16C77, 16F818, 16F819, 16C83, 16F83, 16C84, 16F84, 16F870, 16F871, 16F872, 16F873, 16F873A, 16F874, 16F874A, 16F876, 16F876A, 16F877, 16F877A, 16F88

watchdog:
WDT_ON = watchdog on
WDT_OFF = watchdog out (default)

code_protection:
PROTECT_ON = code is read protected
PROTECT_OFF = code can be read with the help of the programing device (default)

JW (overwites a PROTECT_ON; JW devices should never be code protected!)

oscillator_type:
RC_OSC = external RC circuit
LP_OSC = xtal oscillator (low frequency)
XT_OSC = xtal oscillator or external oscillator circuit
HS_OSC = xtal oscillator (high frequency)
IRC_OSC = internal RC oscillator
ERC_OSC = extrenal oscillator (for those devices with IRC_OSC possible)

ad_converter (only 12X67x, 16C7x and 16F87x):
ADCFG0, ADCFG1, ADCFG2,..
setting which Pin shall work as an analog- pin and which one as a digital-pin.
(Important !!! Using the 12F629 and 12F657 the ADCFG has different allocation commandment.)
see ADINP

Comparator (only 12F62x, 16C62x and 16E62x):
CMCFG0, CMCFG1, CMCFG2,...
Defines how the pins shall work together with the comparators respectively which one worka as a digital-I/O.
VRCGF = defines the value which is loaded into the VRCON-register. Among other things the reference voltage will be set. This setting Can be changed during the program flow by writing into the VRCON-register.
see COMPARATOR

misc (if supported):
PWRTE_OFF = power up timer enable off
PWRTE_ON = power up timer enable on
MCLR_EXT = master clear is at pin MCLR
MCLR_INT = master clear is internal generated (pin MCLR is used as i/o pin)
RBUP_ON = weak pull up resistors on port RB are active
RBUP_OFF = weak pull up resistors on port RB are inactive

DEFINE device (cont.)

GPWU_ON = changes on port GP generates a weak up (resets the cpu)
GPWU_OFF = reset is only generated by power on and MCLR pin
GPPU_ON = weak pull up resistors on port GP are active
GPPU_OFF = weak pull up resistors on port GP are inactive
BODEN_ON = brown out detect enable on
BODEN_OFF = brown out detect enable off
OSC2_IO = pin OSC2 is i/o pin (not clockout)
T0CS_INT = Timer 0 clock source is derived by f/4
T0SE0 = Timer 0 active edge is low to high transition
T0SE1 = Timer 0 active edge is high to low transition

DEFINE other

DEFINE STACK = *addr*
16 bytes data memory in page 0 are used to store the contents of ERG, ARG1 etc, and other important registers if an interrupt occurs.

DEFINE SERIN = SOFT
DEFINE SEROUT= SOFT
forces the compiler to link the software routines for the SERIN and SEROUT instruction even there is an usart beyond these pins.

DEFINE KEYS *port, table*
initializes a 4 by 4 key matrix on *port*. A translation table can be added where the scancode is replaced by another code. Scancode is from 1 to 16. Should scancode 1 be the ascii character "0", scancode 2 should be ascii "1" etc then write:
DEFINE KEYS RB, 0,"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"
see INKEY

DEFINE ARITH32 = address
settles the storage area where the 32-bit operations calculate their provisional result and their final result. The area contains 16 byte and also can be used otherwise. Attention !!!

DATE

The actual date of compiling is insert in the new SRC file. After assembling this date is also in the LST file.

TIME

The compiler replaces this keyword in the new created SRC-file by the actual system-time. So it also appears after assembling in the LST-file.

XTAL

XTAL defines the crystal frequency of the target application in MHz. The compiler uses this value to calculate the numbers of loops in delay routines for e.g. WAIT, SERIN, SEOUT etc.

e.g
XTAL 4.0
target application runs with 4 MHz.

XTAL 0.032
target application runs with 32 kHz.

CLK1XTAL

Using TMR1 instead TMR0 as the time base for CLOCK command (CLOCK1) it may be driven by either the internal Fosc/4 cycle or an external xtal. To define the xtal frequency for TMR1 module use CLK1XTAL.

Assembler Code

Generally it is possible to write and insert control sections in assembler code (e.g. with the BASIC-commands ASM and ENDASM). Though usually not necessary it may be particularly useful to operate with fine tuning. If you access to hardware resources (timer, their prescaler, interrupts, interfaces, etc.) conflicts can't be excluded because the compiler itself accesses to some of those hardware resources. For example when using corresponding commands (CLOCK, I2CHARDS, SERIN, etc.) or by the always enclosed default option of the compiler to the specal function register of the PIC.
To avoid such conflicts you have several alternatives for choice:

1.) Limit the assembler written code to those hardware resources which do not access to the iL_BAS16. Depending on the PIC type you have different numbers of special functions (e.g. up to four timers). So depending on the PIC-type there may be free resources which are not used by Basic and therefore are distributed freely.

2.) To limit the access of the compiler so that it can't access to those resources which are programed in assembler by yourself. First you have to delete all lines which contain the corresponding presettings (special function register) out of the default list. Then the PICs own default-settings are valued. Second: non of the concerned Basic-commands are allowed to be used. The list "overview of basic-commands" will show you the corresponding commands (column: used resources).

The second step is not to be recommend because you need necessarily detailed knowledge of how the compiler works.

3.) Set all corresponding registers as needed in parts of the assembler-program yourself before ending this part of the program
but at least before the use of the corresponing basic-commands. Reset all changed presettings corresponding to the list.
You should be aware that some commands act continuesly (e.g. timer-interrupts).

Read chapter ASSEMBLER

INTERRUPTS (general)

Only the following parts have interrupts: 12C67x, 16C6x, 16C7x, 16X8x and 16F87x. For an easy use of interrupts some special keywords are implemented. These keywords are: INTERRUPT, INTPROC and INTEND. The INTERRUPT keyword tells the compiler, that somewhere in the BASIC program is a subroutine which is only entered in case of an interrupt event. That is the way it does the preparation to enter this routine after it has checked the internal interrupt events (CLOCK and I2C hardware slave). The user interrupt service routine is put into the interrupt chain. Serveral registers controll the interrupt handling. INTCON0, ADCON0, EECON0, INTCON1 etc.. In these registers you have to set and reset bits to enable and disable the desired interrupt. The interrupt routine defined by the user is opened with INTPROC and closed with INTEND, similar to ASM and ENDASM. This interrupt service routine (ISR) can be written in BASIC or machine language or both mixed. Within the ISR you must check if this is the right interrupt source which released the interrupt. If yes, you must reset the corresponding bit (interrupt flag).

There are several interrupt sources. But there is no priority order. This means that an ISR cannot be interruped. The following
interrupt sources are available.

INT edge selectable at pin RB0
RTCC overflow of the rtcc register (255 to 0)
RB changing the input level at pin RB4 to RB7 (logical OR)
EEPROM if a write cycle is finished
ADC at the end of a conversation cycle.

All these interrupts can be enabled and disabled by setting or resetting the corresponding bit in the INTCON register. Additionally the global interrupt enable bit (GIE) must be set to "1" that an interrupt may occur. If the controller is in sleep mode the GIE bit defines what happens if an interrupt occurs. In case of GIE=0 a reset will be generated, if GIE is "1" the controller continues with entering the ISR.

INT interrupt
The external interrupt input RB0/INT is sensitive for either a falling or rising edge. Which slope will release the interrupt is defined in the option register. When the right slope is found and the INTE bit and GIE bit of the INTCON register are set an interrupt occurs. The program branches to the ISR where the INTF bit should be checked. If not set, the interrupt has been released by another source and the ISR should be left. But if set, first service the interrupt and then reset this bit. This interrupt also can terminate the sleep-mode of the processor when the INTE-bit is set to 1.

RTCC interrupt
An overflow from FFH to 0 of the rtcc register will set the RTIF flag to "1". If RTIE and GIE are set, an interrupt will be released. Then the status of the GIE-bits decides wether the process will be restarted (GIE=0, RESET) or branched to ISR.

RB interrupt
A slope at any of the inputs RB4 to RB7 will set the RBIF bit in INTCON0. Only those pins are evaluated which are defined as inputs. An interrupt will be released if GEI and RBIE are set to "1".

ADC interrupt (only 12X67x, 16C7x and 16F87x)
At the end of a conversion cycle the ADIF bit will be set. Only if GIE and ADIE are set too an interrupt will be released.

EEPROM interrupt (only 16X8x and 16F87x)
If a write cycle of the internal eeprom is finished, the EEIF bit is set. An interrupt will be released if GEI a EEIE are set to "1".

Saving and restoring the internal registers like W, STATUS, PC, PCLATH, FSR, ERG, ARG1 etc is handled by the compiler. In case you have not got enough experience you should use the STACK define instruction to save all registers. In some cases, where no compiler register (ERG, ARG1 etc) is used, only W, STATUS, PC etc must be saved.

INTERRUPTS (general) (cont.)

What happens if an interrupt occurs?
If an interrupt occurs the PIC finishes the actual instruction and branch with a self generated call instruction to address 0004. But this only happens, when GIE and the corresponding interrupt enable flag is set. The compiler puts a goto instruction to the label $CLK to this address. If the clock instruction is part of the program, the timer 0 interrupt is served here. After finishing this isr the program enters the users isr. These isr's are the lines between INTPROC and ENDPROC. This can be BASIC instruction or assembler instruction when ASM and ENDASM is used.
To save the W- and Status-register you can use following programseqency:

```
push    movwf   temp_w      ;win a ram-cell
        swapf   status,w    ;statusregister in W
        movwf   temp_s      ;and in a ram-cell

pop     swapf   temp_s,w    ;rebuilt status register
        movwf   status
        swapf   temp_w      ;W-register
        swapf   temp_w,w
```

These routines will be automatically included by the compiler as soon as the CLOCK-, I2CHARDS- or the interrupt-controlled SERIN-command occurs in the program.

The INTCON0 register
In this register the several interrupt sources can be enabled or disabled individually by setting or resetting the corresponding bit. All interrupts can be masked individually. The GIE bit is the global interrupt enable bit.
(The location of these bits may be various).

The contents of the INTCON-register can be different from module to module. In some cases you have to take into account the PIE register! In case of doubt please see the corresponding data sheet of the producer.

bit 7	GIE	Global Interrupt Enable, 0=disable, 1=enable
16C7x:		
bit 6	ADIE	AD conversation ready, 0=disable adc interrupt, 1=enables adc interrupt
16C8x:		
bit 6	EEIE	EEPROM write cycle finished, 0=disable, 1=enable
bit 5	RTIE	0=no timer 0 interrupts, 1=timer 0 overflow (FF > 0) generates an interrupt
bit 4	INTE	0= no interrupt on RB0 HL or LH transition, 1=enables this interrupt (slope selected by option register)
bit 3	RBIE	0=no interrupt on RB4 to RB7 change, 1= any change on RB4 to RB7 generates an interrupt
bit 2	RTIF	1 if timer 0 generates an interrupt, must be reset by software
bit 1	INT	interrupt from RB0
bit 0	RB	interrupt from change on RB4 to RB7

The interrupt flags are set regardless of the interrupt enable bits. So these bits are also useful for polling.

The INTCON1-register

16C7x	ADCON0 (08H)
bit 7	ADCS1
bit 6	ADCS0
bit 5	res.

INTERRUPTS (general) (cont.)

bit 4	CHS1	
bit 3	CHS0	
bit 2	GO/DONE	
bit 1	ADIF	set, if the AD-modifier is ready the software will be reset
bit 0	ADON	

16X8x EECON1 (88H)

bit 7	res	
bit 6	res	
bit 5	res	
bit 4	EEIF	
bit 3	WRERR	set, when write cycle is finished successing reset by software
bit 2	WREN	
bit 1	WR	
bit 0	RD	

IMPORTANT!!!

The main problem of the PIC interrupts are the missing push and pop instructions. In case of an interrupt, a complex BASIC instruction, which uses the compiler registers ERG, ARG1 etc, should be interrupted in such a way, that none of these registers are disturbed. E.g. the BASIC instruction LET Z1 = T1 * V1 is in process. The registers ERG, ARG1 etc are used. Also parts of the runtime library are used, too. Many small steps are necessary to do this job. Now, during the process an interrupt occurs. The processor finishes the actual assembler instruction which is a part of the BASIC instruction and branches to address 0004H. This is done by a call instruction to save the actual program counter value on top of stack. The isr is entered. If now a BASIC instruction uses the ERG or any other of these compiler registers their old contents is lost and the results in the main routine are corrupted.

But how can we solve this? If this interrupt would be ignored (but not lost) until the BASIC instruction is finished, there wouldn't be any problem because the registers ERG, ARG1 etc are not used for parameters from one BASIC instruction to the next. This behavior is reached by adding GEID to the keyword INTERRUPT. Now the compiler disables all interrupts (GIE) if a complex BASIC instruction must be executed. This prevents the interruption of the BASIC instruction. The disadvantage is the unpredictable time between the occurence of the interrupt and the entrence of the isr. If interrupts occur with a fixed period, the isr is not called with the same periode. An output signal derived from these interrupts gets a jitter.

Is there a way to avoid this disadvantage? Yes, there is. If using PIC with more data memory we can define a 16 byte area as stack e.g. DEFINE STACK = addr . addr is the start address of a memory location of free 16 bytes in page 0. In case of interrupts all important registers are stored in this location. At the end of isr the old values are restored. The main program keeps its values and calculates the final result correctly. GIED is not necessary if using STACK.

Labels

Labels must begin with a letter and have a length of 2 characters minimum and 8 characters maximum. Any combination of letters, numbers and underscores is allowed. Do not use reserved words as labels like command- and varialbe-names. No problem are reserved words as a part of a label. Only the first 16 characters are significant!

Right:

START
loop
loop_1

Wrong:

99 (1. character is no letter)
_loop (")
HIGH (reserved word)

Address labels are targets of a GOTO or GOSUB instruction and must be followed by a colon.

Example:

```
START:
        FOR a1 = 0 TO 100
        . . .
        NEXT a1
        GOTO start
```

Symbols for variables or constants must be declared by the DEFINE instruction.

Example:

```
        DEFINE start=1
        DEFINE stop=100
        DEFINE counter=a
TEST:
        FOR counter=start to stop
        . . .
        NEXT counter
```

Labels and symbols cannot be redefined.

Example:

```
        DEFINE start=1
        . . .
        . . .
        DEFINE start=100      error message occurs!!!!
```

Constants

Constants must start with a letter and have a length of 2 charaters minimum and 16 characters maximum. Within the constants any combination of letters, numbers and underscores are allowed. Do not use reserved words as constants. Reserved words as a part of a constant are allowed. C onstants have 16 significant characters, maximum!

You can enter constant values in four different ways:

decimal , hex, binary, ASCII

Hex numbers must have a leading dollar sign, binary numbers a leading percentage sign. Ascii characters or strings are enclosed by quotes. Decimal numbers are not marked.
Symbols as constants must be created with the help of DEFINE-Direktive.

e.g.

```
DEFINE start = 50   AS CONST      REM  decimal
DEFINE start = $FF  AS CONST      REM  hex  ( decimal 255 )
DEFINE start = %00001111  AS CONST                REM  binary  ( Decimal 15 )
DEFINE start = "A"  AS CONST      REM  ASCII  ("A" = decimal 65)
```

The range of values are 0 ... 255, 0 ... 65535 or 0 ... 4294967296. To give a constant a negative value will be interpreted by the compiler as an error.

right:

```
DEFINE start = 1 AS CONST
```

wrong:

```
DEFINE start = -1 AS CONST
```

Within a definition you can not use arithmetic or logical equations to calculate the constant's value.

But Attention !
Using a FOR-NEXT-loop the following expressions are allowed:

```
FOR a = start TO start+99
```

Variables

Since compiler-version 5.5 the professional version iL_BAS15PRO has 32-bit-arithmetic implemented.Therefore we had to introduce the variable type DBLWORD. These variables need 4 memory locations. In Addition the compiler needs RAM memory for calculations. Because this RAM memory is exclusively needed when using 32-bit operations, a special method was implemented to allow the use of this RAM memory in another way during the rest of the time. But big risks are affected by this. The 32-bit-operation overwrites the dates without warning. In case you need those dates later on, they don't have to be placed in this area.The additional RAM memory is called ARITH32 and will be settled with DEFINE. It contains 16 memory locations and is only generating code when you really carry out 32 bit arithmetic operations.

If you change from a former version to a new one:

iL_BAS16 version 5 introduces a new kind of variable declaration. This became necessary because the new generation of PICs (16F87x) has large data memory spreading over 4 banks. This large amount of data registers cannot be handled by the old way. It would result in confusion. If you worked with version 4 or lower you can compile old programs with version 5 by using compiler switch $OLDVAR. The only restrictions are: no other PICs except those defined in version 4 and no access to bank 2 and bank 3. The new declaration needs file DEFAULT.EQU.

The new variable declaration gives you more flexibility and an easier access to those file registers which represents hardware functions (e.g. rtcc, tris). New variables exist out of at least 2 characters. The first must be a letter (A to Z). Letters, numbers and underscore are allowed within the variable. Don't use '$', '%' and '#'. All reserved words should not be used as variables but allowed as a part of it. Only the first sixteen characters are significant.

All predefined symbols used by iL_BAS16 (compiler) or iL_ASS16 (assembler) are listed in DEFAULT.EQU. You also may use this predefined symbols in BASIC statements and assembler statements. Your own and new variables have to be defined by the DEFINE assignment at the top of the program or within a separate file which is imported by the compiler (include files). The register address which is accessed by the defined symbol is not longer assigned by the old predefined variables A, B, C, .. , AA, AB, .., but with the real absolute address.

e.g.

```
   DEFINE limit=10 AS CONST     'symbol  limit gets the value of 10

   DEFINE counter=$40
or DEFINE counter=$40 AS BYTE                    ' counter is 8 bit wide
```

This is the way to define byte variables. AS BYTE is not needed, but useful for better reading. A 16 bit variable (word) is defined as follows:

```
   DEFINE result=$50 AS WORD
or DEFINE result=80 AS WORD
```

32-bit variables (only professional version) will be initiate with:

```
DEFINE  variable = $50  AS  DBLWORD
```

The 32-bit variables in professional version 5.5 will be defined according the above proceeding:

```
DEFINE      ZW_ERG = $60 AS DBLWORD
DEFINE      ZW_ERG = 96  AS DBLWORD
```

Additionally you need the following instruction:

```
DEFINE      ARITH32 = address
```

Variables (cont.)

This fixes the memory area, where the 32-bit operation calcuates its intermediate and its final result. The area contains 16 byte und can possibly be used in other ways. ATTENTION !!!

A 32-bit-variable occupies 4 memory locations.

The specification of AS BYTE is optional and can be left off. But it is to be recommended to do this little extra work because of clearness.

The BASIC compiler iL_BAS16 only supports unsigned byte, word and doubleword values. All values stored in variables must fit into 8 bits for single variables and into 16 bits for double variables. 8 bit variables will fit in a range from zero to 255. 16 bit variables ranges from zero to 65535 and 32 bit ranges from zero to 4294967296

"Table of usable variable addresses" tables show the memory map of each pic. Usable register addresses are marked with > and < . Some PICs have very less memory. Be careful when using registers twice.

IMPORTANT !!!
Take care not to define a word variable to the last memory address of a bank because only the low byte will keep in this register while the higher byte is set on register 00 (indirect) of the same! page.

ARRAY variables:
Only simple variable assignments are possible, a calculation is not allowed with an array variable as an argument . The additional data RAM of the PIC12C509 and PIC16C57 is only used for single dimensional arrays. You have no access to this data memory by using simple variables.

Example:

PIC16C57:

```
DEFINE field_1 = $30
DEFINE field_2 = $50
DEFINE filed_3 = $70
DEFINE pointer = $1F
....
LET field_1(5)=55 'access address $35
LET field_1(16)=10 'access field_2 first item.
```

PIC12C509: only addresses 30H to 3FH.

Arrays in other PICs than those above are used like:

```
DEFINE field = $60 AS BYTE        'Arrays are only byte variables
DEFINE pointer=$51 AS BYTE        'index variable must be 8 bit
....
LET field(1)=55                   'access to address $61
LET pointer=5
LET field(pointer)=10             'access to address $65
```

IMPORTANT !!!

Variables (cont.)

Take care that the last memory location of a page is not used by a word variable because the compiler cannot handle a wrap around to the next page and the first memory location of a page is not usable.

Devices with plenty of data RAM helps you to get a good solution for the problem of re-entrance while an interrupt occures. You can save all registers by using the runtime library because there are no PUSH and POP commands for the PIC. For saving these registers you need 16 bytes in the first RAM page (address < 80H). For more details see chapter "INTERRUPTS".

INTERNAL COMPILER VARIABLES

Internally the compiler uses the following 16-bit-variable on which you have of course access. For example you have accesse to the rest of a division (ARG1):

ARG0 o. ERG	low-order 16-Bit result
ARG1	higher-order 16-Bit result
ARG2	16-Bit argument
ARG3	16-Bit argument
ARG4	16-Bit argument
ARG5	8-Bit argument
ERR	8-Bit error term

The predefined variables ERG, ARG1, ARG2, ARG3, ARG4 and ARG5 are used by the compiler as a loop counter and for calculations. It depends on the respevtive module where all internal variables will be layed down inside the memory of the PIC. It is defined under Scale of variable addresses in column "compiler internal".

The ERR-variable

An ERR-variable informs you about internal operations.It is bit-sensed and has the following functions:

bit 7	overflow at arithmetic routines (will be deleted before each operation)
bit 6	will be used by the compiler as a flag of the return adress out of the RUN
bit 5	TIME-library at the 16C54 and 16C55, because it just has a
bit 4	two-stage stack, which here is not sufficient
bit 3	timeout in SERIN or I2C occur
bit 2	counts the nesting of the GOSUB-routine, because this check-up
bit 1	can't be done at the moment of compilation
bit 0	000 = no UP, 001 = 1 UP, 101 = 5 UP = are not allowed.

32-bit-arithmetic

Using 32-bit-arithmetic (only PRO-version) you need additional "internal" compiler-variables, which will be settled by DEFINE ARITH32.

PORT- and TRIS-register

When using the 12C5xx und 16C5x the TRIS-informations must be hold ready in the working memory because here in contrary to the 16C6x, 16C7x and 16B8x the TRIS-register can't be read out.

RATRIS
RBTRIS
RCTRIS

Variables (cont.)

This shows why the modules 12C5xx and 16C5x only have eight 8-bit respectively four 16-bit-variable.

The variables RA, RB, RC, RD, RE indicate the corresponding ports. At a lot of commands the corresponding pin has to be specified beside the port. This will be done either by a comma or a point. The specification by a komma has the advantage, that you also can use a variable.

for example

```
LET     a1 = 7
HIGH    RA,a1     REM  set the most significant bit a1 to 1 ( Bit 7 = 1 )
LOW     RB.3
```

Making a value assignment to a variable you have to pay attention to the permitted value range. The compiler doesn't make any check-up. It shortens the result corresponding to the type of variable.

For example

```
LET     a1 = 266          REM  assignment to a 8-bit variable
PRINT "Variable a1 = " ,a1    REM  output on display
```

The value of a variable a1 -> 10 (a1 = 266 - 256 = 10)

The variable definition of the DEFAULT.EQU file you will find at appendix Anhang I

Modules with more memory capacity offer an elegant solution to the reentrance-problem when an interrupt appears. Whereas with "small modules" only the W-and status-register can be saved, here all variables which were used by the runtime-library can be buffered (compensation for PUSH and POP). A 16-bit storage area will be settled with the help of the DEFINE-instruction at the first RAM page (adress <80H).

On the following page you will find the memory areas which can be used for user-available-variables. Some PICs only have a few variables. In those cases you can use further memory capacity if you don't use the commands DATA,READDATA and RESTORE because these commands need a 16-bit-pointer. To find out the real address of these pointers please look at DEFAULT.EQU.

Compatibility of the variable-size

Assingments are allowed when variable_c and variable_a are different types:

LET variable_C = variable_A

To avoid mistakes the type of variable A must have at least the same size as variable C, or it must be made certain that the numeric value of variable_A is small enough to fit into variable_C. Otherwise the higher bits will be deflacated and simultaniously signalized in the error-variable (in case this option wasn't deselected by compilerswitch $CCOFF).

The same applies to all functions concerning their results like
LET variable_C = summand_a + summand_b.
At multiplication note that normally the result requires the next higher variable-size against the size of the factors.

Variables (cont.)

Different variable-types can be written facultatively mixed into an argument (input value) of a function. You must be aware of that then automatically the longer computing time variant of the function will be carried out which corresponds to the bigger type of the two input-variables.

The variables will be placed in the mermory after the following scheme:
DEFINE var8 = $40 as byte => the contents of variable var8 is in 40H.
DEFINE var16=?40 as word => the Low-Byte is in 40H and the High-Byte of the 16-bit variable in 41H.
DEFINE var32=$40 as dblword => the LOW-Byte is in 40H, the following one in 41H, the next but one in 42H and the highest byte of the doubleword variable in 43H.

To program computing-time-optimized

1.)
PICs with several RAM-banks apply to: variables which are used often will be managed fastest if they lie in bank 0. Bank 1 and 2 are normally quicker than bank 3.

2.)
Program sections, e.g. often used routines, will be managed fastest when they are in page 0 (at that place you also find the compiler-internal funktionroutines). Therefore the subroutins should be be there too.

3.)
Work if anyhow possible exclusively with Byte-variables. A very compact code will be produced for the logical and arithmetic commands. 16-bit-operations can be avoid by using handy computing steps.

4.)
To manipulate single bits use also SET, RES and TOGGLE. The PIC knows such compact bit-commands and then the compiler generates them too.

5.)
Please examine when using IF-clauses at 0 (= 0) respectively not equal 0. When using byte-variables a more compact code will be generated compared to if-clauses <, >, <= or >=. 16-bit and 32-bit comparisions also need the subtract-routine in the runtime-library which needs a lot of time.

6.)
If you don't need a range check at the byte-addition respectively -subtraction switch on the corresponding code generation by using $CCOFF.

7.)
Avoid text in LCDWRITE. Instead remove the text into DATA-fields and read out character after character. It is possible to write continuously in the LCD by setting the cursor position to 0,0.

Variable internal

Internal variables used by the compiler iL_BAS16

iL_BAS16 needs several registers for internal use e.g. parameters for library routines or loop counters. These variables are also predefined and are named as follows:

ARG0 or ERG low byte of 16 bit result
ARG1 high byte of 16 bit result
ARG2 16 bit auxillary register
ARG3 16 bit auxillary register
ARG4 16 bit auxillary register
ARG5 8 bit auxillary register
ERR 8-bit error variable

Contrary to other PIC types the PIC12C5xx and PIC16C5x have no readable TRIS register. So this information must be kept in a shadow register. This is necessary because of the bit manipulation of the TRIS register.

Shadow register
RATRIS
RBTRIS
RCTRIS

Now you can imagine how rare data memory in PIC12C5xx and PIC16C5x is. But anyway you have eight 8 bit- respectively four 16 bit- variables and you can write BASIC programs for such tiny devices. No doubt!

The ERR variable
In this internal variable the compiler stores flags and values of results of its calculation.

Bit 7 overflow in arithmetic functions (will be cleared before operation begins)

Bit 6 used by the compiler to find the return address in runtime library (only
Bit 5 PIC12C5x and PIC16C5x, because they only have a two level hardware
Bit 4 stack, which is not sufficient).

Bit 3 timeout occurs in SERIN or I2C

Bit 2 counts how deep the GOSUB routines are nested because
Bit 1 this cannot be found out during compilation.
Bit 0 000 = no GOSUB, 001 = 1 GOSUB, 101 = 5 GOSUB = that is too much.

Ports are named as RA, RB, RC, RD, RE. If you want to access to one bit in a variable or port you simply add the bit number or pin number to the variable separated by a point or comma. Using a point, the specification must be a number in the range of 0 to 7. If the separation is a comma you can write a number or a variable.

e.g.

```
LET a1 = 7
HIGH RC,a1        REM the upper bit of port RC is set ( bit 7 = 1 )
LOW RB.3
```

If you try to assign a value to a variable that is beyond the limits (255 or 65535) the compiler terminates compilation with an error message.

Variable internal (cont.)

e.g.

LET a1 = 266 REM error will occur

(see DEFAULT.EQU in appendix)

Table of usable variable addresses

This file showshow the compiler uses the internal data memory of each PIC. The addresses listed in coloumn "USER" are usable for your program.
Normally you have the same memory map for the A- or B-type. In case of doubts please compare the Microchip-data sheets for both types.

PIC	ROM	PIC internal	Compiler internal	USER (hex)	USER (decimal)	amount	remark
12C508	512	000-006	008-016	> **017-01F**	**23-31**<=	8	
12E508	512	000-006	008-016	> **017-01F**	**23-31**<=	8	
12C509	1024	000-006	008-016	> **017-01F**	**23-31**<=	8	030-03F for arrays
12E509	1024	000-006	008-016	> **017-01F**	**23-31**<=	8	030-03F for arrays
12F629	1024	000-01F 080-09F	04E-05F	> **020-04D**	**32-77**<=	45	
12C671	1024	000-01F 080-09F 0A0-0A1	020-022 070-07F	> **023-06F** > **0A2-0BF**	**35-111**<= 76 **162-191**<= 29		
12C672	2048	000-01F 080-09F 0A0-0A1	020-022 070-07F	> **023-06F** > **0A2-0BF**	**35-111**<= 76 **162-191**<= 29		
12E673	1024	000-01F 080-09F 0A0-0A1	020-022 070-07F	> **023-06F** > **0A2-0BF**	**35-111**<= 76 **162-191**<= 29		
12E674	2048	000-01F 080-09F 0A0-0A1	020-022 070-07F	> **023-06F** > **0A2-0BF**	**35-111**<= 76 **162-191**<= 29		
12F675	1024	000-01F 080-09D	04E-05F	> **020-04D**	**32- 77**<= 45		
16C505	1024	000-007	008-017	> **018-01F**	**24-31**<=	8	030-03F for arrays 050-05F for arrays 070-07F for arrays
16C53	384	000-007	008-017	> **018-01F**	**24-31**<=	8	
16C54	512	000-007	008-017	> **018-01F**	**24-31**<=	8	
16C55	512	000-007	008-017	> **018-01F**	**24-31**<=	8	
16C56	1024	000-007	008-017	> **018-01F**	**24-31**<=	8	
16C57	2048	000-007	008-017	> **018-01F**	**24-31**<=	8	030-03F for arrays

Table of usable variable addresses (cont.)

050-05F for arrays
070-07F for arrays

16C58 2048 000-007 008-017 > **018-01F** **24-31**<= 8 030-03F for arrays
050-05F for arrays
070-07F for arrays

16C554 512 000-01F 020-031 > **032-06F** **50-111**<= 61
080-09F

16C556 1024 000-01F 020-031 > **032-06F** **50-111** <= 61
080-09F

16C558 2048 000-01F 020-031 > **032-07F** **50-127**<= 77
080-09F 0A0-0A0 > **0A1-0BF** **161-191** <= 31

16C61 1024 000-00B 00C-01E > **01F-02F** **31-47** <= 17

16C62 2048 000-01F 020-032 > **033-07F** **51-127**<= 76
080-09F 0A0-0A0 > **0A1-0BF** **161-191**<= 31

16C620 512 000-01F 020-032 > **033-06F** **51-111**<= 60
080-09F

16C621 1024 000-01F 020-032 > **033-06F** **51-111**<= 60
080-09F

16C622 2048 000-01F 020-032 > **033-07F** **51-127**<= 76
080-09F 0A0-0A0 > **0A1-0BF** **161-191**<= 31

16E623 512 000-01F 020-022 > **023-06F** **35-111**<= 76
080-09F 070-07F

16E624 1024 000-01F 020-022 > **023-06F** **35-111**<= 76
080-09F 070-07F

16E625 2048 000-01F 020-022 > **023-06F** **35-111**<= 76
080-09F 070-07F > **0A0-0BF** **161-191**<= 32

16F627 1024 000-01F 06D-07F > **020-06C** **32-108**<= 76
080-09F 0ED-0FF > **0A0-0EC** **160-236**<= 76
100-11F 170-17F > **120-14F** **288-335**<= 47
180-1FF

16F628 2048 000-01F 06D-07F > **020-06C** **32-108**<= 76
080-09F 0ED-0FF > **0A0-0EC** **160-236**<= 76
100-11F 170-17F > **120-14F** **288-335**<= 47
180-1FF

16C63 4096 000-01F 020-032 > **033-07F** **51-127**<= 76
080-09F 0A0-0A0 > **0A1-0FF** **161-255**<= 94

Table of usable variable addresses (cont.)

16C64 2048 000-01F 020-032 >**033-07F 51-127**<= 76
080-09F 0A0-0A0 > **0A1-0BF 161-191**<= 31

16C65 4096 000-01F 020-032 > **033-07F 51-127**<= 76
080-09F 0A0-0A0 > **0A1-0FF 161-255**<= 94

16C66 8192 000-01F 020-032 > **033-07F 51-127**<= 76
080-09F 0A0-0A0 > **0A1-0FF 161-255**<= 94
100-11F 120-120
180-19F 1A0-1A0

16C67 8192 000-01F 020-032 > **033-07F 51-127**<= 76
080-09F 0A0-0A0 > **0A1-0FF 161-255**<= 94
100-11F 120-120
180-19F 1A0-1A0

16C71 1024 000-00B 00C-01E > **01F-02F 31-47** <= 17
080-09F

16C72 2048 000-01F 020-031 > **032-07F 50-127**<= 77
080-09F 0A0-0A0 > **0A1-0BF 161-191**<= 31

16C73 4096 000-01F 020-031 > **032-07F 50-127**<= 77
080-09F 0A0-0A0 > **0A1-0FF 161-255**<= 94

16C74 4096 000-01F 020-031 > **032-07F 50-127**<= 77
080-09F 0A0-0A0 > **0A1-0FF 161-255**<= 94

16C710 512 000-00B 00C-01E > **01F-02F 31-47** <= 17
080-08B

16C711 1024 000-00B 00C-01E > **01F-04F 31-79** <= 49
080-08B

16C715 2048 000-01F 06D-07F > **020-06C 32-108**<= 76
080-09F

16C83 512 000-00B 00C-01E > **01F-02F 31-47** <= 17
080-08B

16C84 1024 000-00B 00C-01E > **01F-02F 31-47 <**= 17
080-08B

16F83 512 000-00B 00C-01E > **01F-02F 31-47** <= 17
080-08B

16F84 1024 000-00B 00C-01E > **01F-04F 31-79** <= 49
080-09B

16F818 1024 000-01F 06D-07F > **020-06C 32-108**<= 76
080-09F > **0A0-OBF 160-191**<= 32

Table of usable variable addresses (cont.)

```
          100-10F
            180-18F

16F819  1024  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F  0ED-0FF    > 0A0-0EC   160-236<=  76
              100-10F  16D-17F    > 120-16C   288-364<=  76
              180-18F  1ED-1FF

16F870  2048  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F
              100-11F
              180-19F

16F871  4096  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F
              100-11F
              180-19F

16F872  4096  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F
              100-11F
              180-19F

16F873  4096  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F  0ED-0FF    > 0A0-0EC   160-236<=  76
              100-11F
              180-19F

16F874  4096  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F  0ED-0FF    > 0A0-0EC   160-236<=  76
              100-11F
              180-19F

16F876  8192  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F  0ED-0FF    > 0A0-0EC   160-236<=  76
              100-11F  16D-17F    > 120-16C   288-364<=  76
              180-19F  1ED-1FF    > 1A0-1EC   416-492<=  76

16F877  8192  000-01F  06D-07F    > 020-06C    32-108<=  76
              080-09F  0ED-0FF    > 0A0-0EC   160-236<=  76
              100-11F  16D-17F    > 120-16C   288-364<=  76
              180-19F  1ED-1FF    > 1A0-1EC   416-492<=  76
```

IMPORTANT !!!

Take care that the last memory location of a page is not used by a WORD-variable, or the last three location by a Doubleword, because the compiler cannot handle a wrap around to the next page and the first memory locations of a page is not usable.

Table of usable variable addresses (cont.)

ARRAYs

When using the modules PIC16C6x, PIC16C7x, PIC16C8x and PIC16F8x you can define the data memory as an array or a number of arrays. You simply have to append an index. This index will be used as an offset to calculate the absolute memory address. The offset will be added to the address of the reference-variable. Restrictions by using these arrays correspond to those arrays of the
PIC16C57.

e.g.

```
DEFINE  field = $50 AS BYTE
...
...
 LET field(2) = 10                    'the value 10 is writen in memory address $52 (=$50+2)
```

Modules with more memory capacity offer an elegant solution for the reentrance-problem when an interrupt appears. At smaller modules only the W- and status-register can be saved. Using bigger ones all variables which are used in the runtime-library will be buffered (replace for PUSH and POP). Therefore a 16-Byte memory area at the 1 RAM page will be fixed with the help of the DEFINE-instruction.

IF constructions

The IF-THEN-ELSE-instruction compares variables and/or constants.

```
IF var_a = 1 THEN LET var_b = var_a * 2 ELSE GOTO more
```

IF clauses cannot be nested !!! It is forbidden to:

WRONG! **IF var_a = 1 THEN IF...**

You can't use logical combinations in If-causes. It's forbidden to:

Wrong! IF a1 and 1 = THEN

Because in the example above only one bit is checked you should use the following more efficient formulation,

IF a1,0 = 1 THEN

otherwise you have to write a separate line for the logical combinations;

see also IF-THEN-ELSE

Mathematical operators

The compiler supports following functions:

Addition + a + b
Subtraction - a - b
Multiplication * a * b
Division / a / b
Modulo mod a mod b (modulo, rest of division)

Shift- and rotation instructions:

shl ***var*** shl 11110011 => 11100110 contents of *var* shift to the left (fill up with 0)
shr ***var*** shr 11110011 => 01111001 contents of *var* shift to the right (fill up with 0)
rotl ***var*** rotl 11110011 => 11100111 contents of *var* rotates to the left (bit 7 becomes bit 0)
rotr ***var*** rotr 11110011 => 11111001 contents of *var* rotates to the right (bit 0 becomes bit 7)
(Numbers above are in binary format).

Compare:

equal = a = 5
not equal <> a <> b
less than < a < b
greater than > a > b
less or equal <= a <= b
greater or equal >= a >= b

For 8 and 16 bit operations:
The results of multiplication, division and modulo always have 16 bits. It is truncated and adapted to the type of variable that finaly stores the result. For the function of addition and subtraction an optimized code is generated.

e.g.

```
var_a = 200                      REM  8 bit variable
var_b = 200                      REM  8 bit variable
var_c = 0                        REM  8 bit variable
var_u = 0                        REM 16 bit variable
var_c = var_a * var_b      REM result is 8 bit, too ( 64 ), upper byte lost
var_u = var_a * var_b      REM result is 16 bit ( value_u = 40000 )
```

The compiler expands all internal intermediate results to 16 bits (at 8bit-multiplication and -division).

e.g.

```
var_a = 200                                  REM 8 bit variable
var_b = 200                                  REM 8 bit variable
var_u = 0                                    REM 16 bit variable
var_s = 0                                    REM 16 bit variable
var_u = var_a * var_b / var_b                REM result is ok ( var_u = 200 )
var_s = var_a * var_b * var_a / var_b / var_b   REM result is wrong ( var_u = 0 ), because var_a
                                                intermediate result exceeds the range of 16 bits.
```

Mathematical operators (cont.)

32-bit-operations will be done independently from 8-bit- and 16-bit-arithmetic. Independant routines in the runtime-library are competent for this.

If you write mathematical expressions for this BASIC compiler, please take care of the following circumstances: The compiler calculates the equation form left to right without any respect of brackets or order of priority.

e.g.
var_a = 3 + 4 * 5
The result is 3 + 4 * 5 = 35 instead of 3 + (4 * 5) = 23 which would be right, usually

The compiler always calculates with 16 bit resolution. The carryover could be lost, according the range of the variable where the result should be stored in.

Logical operators

The compiler iL_BAS16 supports following logical operands:

logical	AND	and	a and b
logical	OR	or	a or b
logical	EXOR	xor	a xor b

e.g.

```
 LET var_a = 5 and 3         REM 101 and 11 = 001 = 1
 LET var_a = 5 or %1001    REM 101 or 1001 = 1101 = 13
LET var_a = 5 xor $A       REM 101 xor 1111 = 1010 = 10
LET var_aXOR var_b
```

Truth table:

			AND	OR	XOR
0	0	0	0	0	
0	1	0	1	1	
1	0	0	1	1	
1	1	1	1	0	

Remember that there is another priority order in mathematical and logical expressions.

32-bit arithmetic introduction

(only professional version)

The 32-bit arithmetic offers the possibility to work with numbers from 0 to 429496296. These are numbers without sign and use 4 bytes in the memory. To define them use the keyword DBLWORD. Because the 32-bit-arithmetic needs a lot of memory we've chosen a way to treat memory resouces with care. The 8-bit- and 16-bit-arithmetic use a memory area which is at diposal only for the compiler. Those variables are predefined as ERG, ARG1, ARG2, ARG3, ARG4, and ARG5 and are not allowed to be shifted to another memory location. The 32-bit-arithmetic additionally needs memory capacity for arithmetic-routines in the runtime-library. This memory must be in BANK0. The start address of the memory area will be fixed with a DEFINE ARITH32 instruction. The area contains 16 bytes which must be available in a coherent set. So you must be very careful when you are near the bank boundary. Only the 32-bit-arithmetic routines will access to this memory area. So you have the possibility to use this area in another way. But this double-using is quite dangerous what means that only professionals should use it.

The following example shows a mixture of byte-, word- and Dblword-variables:

```
define device 16f877
define a1=$20 as word
define a2=$22 as dblword
 define a3=?120  variables in BANK 2
 define a4=$1A0                  variables in BAKN 3
define xx=$30
 define arith32=$40              'defines the rule of computing
start:
        let a2=1234567
        let a2=a2 * 3
        let a2=a2 and $00ffff00
        set a1,7                 'SET only with BYTE-variables
        set a2,7
        let xx=a2                '32-bit to 8-bit, just LSB
        set a3,7
        let xx=a1                '16-bit to 8-bit, just LSB
        set a4,7
        let xx=a4                'variables of bank 4 to bank 1
        high rb,1
        wait 2000
        low rb,1
        wait 2000
        goto start
```

Comparators

The PIC 16C62x has comparator inputs at port RA. These inputs can be configured in several ways. The procedure is quite equal to the ADCFG for analogue converters. In this case the keywords are CMCFGx and VRCFGx.

With these keywords you can set comparators respectively voltage references. The following connections result:

CMCFG0	comparator reset	
CMCFG1	3 inputs mux	C1OUT = RA2-RA0 (or RA2-RA3) C2OUT = RA2-RA1
CMCFG2	4 inputs mux	C1OUT = Vref-RA0 (Vref-RA3) C2OUT = Vref-RA1 (Vref-RA2)
CMCFG3	2 common Vref	C1OUT = RA2-RA0 C2OUT = RA2-RA1 RA3 = digital io
CMCFG4	2 comparators	C1OUT = RA3-RA0 C2OUT = RA2-RA1
CMCFG5	1 comparator	C1OUT = RA2-RA1 RA0 and RA3 are digital io's
CMCFG6	2 common Vref	C1OUT (RA3) = RA2-RA0 C2OUT (RA4)=RA2-RA1
CMCFG7	off	RA0 to RA3 are digital io's

In CMCFG1 and CMCFG2 mode the CIS bit in CMCOM register defines if RA0 or RA3 (RA1 or RA2) is connected to the comparator. This bit must be set and reset by software using SET and RES commands.

VRCFG defines in which way the reference voltage is used. Because each bit has a different meaning, you have to calculate the value for x.

value add to x
bit 7 value = 0 -> Vref off
value = 128 -> Vref on
bit 6 value = 0 -> Vref no connection to RA2
value = 64 -> Vref is conneted with RA2
bit 5 value = 0 -> Vref large range
value = 32 -> Vref smal range
bit 4 -> not used
bit 3 -> defines the voltage range with formular:
if bit 5 = 1 then: Vref = (x/24)*Vdd; with x = 0 to 15 (bit 3 to bit 0)
if bit 5 = 0 then: Vref = 0.25*Vdd + (x/32) * Vdd; with x = 0 to 15

Hint:
(please read the chapter in MICROCHIPs Databook)

Program memory pages (iL_PAGE0)

The compiler module iL_PAGE0 helps to handle the program memory paging. It is a part of iL_BAS16SEP and iL_BAS16PRO. Other compiler versions do not use this module. If the program memory of the desired PIC is larger than one page, iL_PAGE0 searches and calculates the branches over a page boundery. To reach such a target address it is necessary to modify the PCLATH register or the page preselect bits in the status register before the goto or call instruction. This boring and error intensive job is done by iL_PAGE0 automatically.

iL_PAGE0 works like an assembler. Its iterative algorithem looks for these far branches and replace them by special instructions. The assembler translates them into a series of instructions.

But the possibilities of iL_PAGE0 are limited. Of course you can instruct iL_PAGE by using $LRANGE 2048 to convert all CALLs and GOTOs which occur in a program in their longcall- and longjump- equivalent. But this would be an enormous waste of memory-resources. To 'accomodate' the module iL_PAGE0 with a little program-disciplin will be shown at $LRANGE.

ADDELAY

(only for PICs with built-in analog digital converter)

syntax: ADDELAY const

function: Inserts an additional delay when selecting an new channel or turning on the adc before starting the conversation.

const (1...255) delay t=(16/fq)**const*

description: The value of the hold capacitor is increased from 51pF (16C7x) to 120pF at PIC16F87x. So if you select another channel more time is needed to charge or discharge this capacitor to the new voltage level. If you don't care you may get wrong values when reading the analogue input. Reading twice the same channel the second result is ok. This effect can be eliminated by inserting an additional delay.

ADINP

(only for PICs with built-in analog digital converter)

syntax: ADINP *chanal, var*

function: converts an analogue voltage into its digital equivalent. The maximum value depends on Vref.

chanal (0 ... 7)	selects the I/O input chanal, may be variable or constant
var (8 bit or 16 bit)	where the result of the conversation will be stored

description:

The ADINP allows you to measure a voltage between 0V and Vref. So 0V at the input results in a value of 0 while a voltage of Vref results in a value of 255 (0FFH). The converted value will be stored in a 8 bit variable. To increase the PIC's flexibility, some inputs for ad conversation may also be used as standard i/o. The available configuration is listed in the table below and must be defined in the DEVICE line. If the ADINP command is executed, the program first selects the desired chanal. If the ad converter is turned off, it will be switched on and then the conversation is started. If the RC oscillator is used as timebase, the conversation time is independent of the xtal frequency. Its duration ranges between 20us up to 60us (typical 40us). This time depends on the power supply voltage, temperature and tolerances during the factory process. At the end of conversation (the controller waits until ready) the result is stored into the 8 or 16 bit variable *var*. The number of analogue inputs depends on the numbers of pins the PIC has.

example:

```
define device 16C71,wdt_off,adcfg0
xtal 4.19
adinp 0,a
```

remarks:

Only for 12C67x, 16C7x and 16F87x. Don't forget ADCFGx in the DEVICE line. The analogue inputs are read into a multiplexer which selects the desired input and leads the signal to the ad converter:

16C71, 16C710, 16C711: ADCFGx defines the usage of port RA.

	RA0	RA1	RA2	RA3	Ref.
ADCFG0	analogue	analogue	analogue	analogue	Vref=Vdd
ADCFG1	analogue	analogue	analogue	ref.inp	RA3
ADCFG2	analogue	analogue	digital	digital	Vref=Vdd
ADCFG3	digital	digital	digital	digital	nc

16C72.to 16C77: ADCFGx defines the usage of port A and port E

	RA0	RA1	RA2	RA3	RA5	RE0	RE1	RE2	Ref.
ADCFG0	A	A	A	A	A	A	A	A	Vdd
ADCFG1	A	A	A	Vref	A	A	A	A	RA3
ADCFG2	A	A	A	A	A	D	Dl	D	Vdd
ADCFG3	A	A	A	Vref	A	D	D	D	RA3
ADCFG4	A	A	D	A	D	D	D	D	Vdd
ADCFG5	A	A	D	Vref	D	D	D	D	RA3
ADCFG6	D	D	D	D	D	D	D	D	nc
ADCFG7	D	D	D	D	D	D	D	D	nc

RE0,RE1,RE2 only exist with 40pin modules.

ADINP (cont.)

The PIC16F873, 16F874, 16F876 abd 16F877 have a 10 bit ad converter which can be read either as a 8 bit result into a 8 bit variable or 10 bit result into a 16 bit variable. It is up to you to select the result format as left or right fit by defining ADCFGx,L or ADCFGx,R. To put the result into a byte variable it makes sens to select "left". For 10 bit results in a word variable "right" will be the right choice (see corresponding data sheet of Microchip).

16F87x

	RA0	RA1	RA2	RA3	RA5	RE0	RE1	RE2	Ref.
ADCFG0	A	A	A	A	A	A	A	A	Vdd
ADCFG1	A	A	A	Vref+	A	A	A	A	RA3
ADCFG2	A	A	A	A	A	D	D	D	Vdd
ADCFG3	A	A	A	Vref+	A	D	D	D	RA3
ADCFG4	A	A	D	A	D	D	D	D	Vdd
ADCFG5	A	A	D	Vref+	D	D	D	D	RA3
ADCFG6	D	D	D	D	D	D	D	D	nc
ADCFG7	D	D	D	D	D	D	D	D	nc
ADCFG8	A	A	Vref-	Vref+	A	A	A	A	RA3,RA2
ADCFG9	A	A	A	A	A	A	D	D	Vdd
ADCFG10	A	A	A	Vref+	A	A	D	D	RA3
ADCFG11	A	A	Vref-	Vref+	A	A	D	D	RA3, RA2
ADCFG12	A	A	Vref-	Vref+	A	D	D	D	RA3, RA2
ADCFG13	A	A	Vref-	Vref+	D	D	D	D	RA3, RA2
ADCFG14	A	D	D	D	D	D	D	D	Vdd
ADCFG15	A	D	Vref-	Vref+	D	D	D	D	RA3, RA2

RE0, RE1, RE2 only at 40 pin devices

12F675

Here a bit mask decides between analog- and digital-input.

ATTENTION !!! The corresponding bit in the TRIS-register must be set at 1 (INPUT) so that the pin can work as an analog-input.

	AND	AN1	AN2	AN3
	RA0	RA1	RA2	RA4
ADCFG0	D	D	D	D
ADCFG1	A	D	D	D
ADCFG2	D	A	D	D
ADCFG3	A	A	D	D
ADCFG4	D	D	A	D
ADCFG5	A	D	A	D
ADCFG6	D	A	A	D
ADCFG7	A	A	A	D
ADCFG8	D	D	D	A
ADCFG9	A	D	D	A
ADCFG10	D	A	D	A
ADCFG11	A	A	D	A
ADCFG12	D	D	A	A
ADCFG13	A	D	A	A
ADCFG14	D	A	A	A
ADCFG15	A	A	A	A

ADINP (cont.)

Here only AN1 can be defined as a reference-input by using compiler switch $VCFG. Because it is a 10-bit-converter you have to pay attention to the representational form of the result (see 16F87x).

Important!
Usually the ad converter is kept turned on all the time. But if you wish to turn it off if not used you should add a '*' character to ADCFGx. This effects that the ad converter is turned on and off automatically.

example:

```
ADCFG3*
 ADCFG3*,R    ' only 16F87x
```

ASM

syntax: ASM

function: between the statement ASM and ENDASM you can write machine language instructions

description: the compiler copies all lines between ASM and ENDASM without any changes into the source file. There they will be assembled by iL_ASS16. The syntax of these mnemonics is quite the same as the assembler (created by Microchip) needs. The first column is reserved for labels.

example:

```
            ASM
START       MOVLW 00h       ;clears port B
            MOVWF RB
            ENDASM
```

remarks: ASM and ENDASM are commands and so they need at least one leading space. If your assembler porgram crosses pages, you must take care yourself to set the page "preselect bits" or the PCLATH register accordingly. But you can olso use the special assembler commands LCALL, LJMP, LJC, LJNZ etc.

BITPOS

syntax: BITPOS *var1, var2*

function: Converts a number from 0 to 7 into their bit position value.

var1 result variable
var2 source variable

description: You often get the problem to mask single bits within a variable depending on a counter value. For example: in the first pass bit 0 should be testet, in the second pass bit 1, in the third pass bit 2 etc. BITPOS converts as follows:

0	%00000001	4	%00010000
1	%00000010	5	%00100000
2	%00000100	6	%01000000
3	%00001000	7	%10000000

Example:

```
        LET  var=5
    BITPOS var_B, var_A          REM in var_B is written exclusively
          'REM %00100000  =  32
```

remarks: BITPOS is only usable with 8-bit-variables, it is also suited for 2^n with n=0...7. Only the three lowest bits are taken into account. So value 9 in variable var_A (see example above) has the result %00000010.

CALVAL

(only for PICs with activate internal RC oscillator)

syntax: CALVAL *value*

value is a byte constant

function: Defines a value for the OSCCAL register

description: There are several PICs with an internal RC oscillator. Because these types of oscillators are not very exact in frequency (depends on voltage of power supply and temperature) Microchip measures a calibartion value to trim the 4 MHz frequency as close as possible. This value has to be written into the OSCCAL register. BUT this value will be erased in JW-types because it is programmed at the highest program memory address which is available in this kind of PICs. If you use a JW-PIC read it out first and note the value at the highest memory location. Programmer normally do not program this memory address. But if you use JW-types it is comfortable to define this value (read out of the PIC before erasing) in the BASIC program. This is much easier than changing this value during programming procedure. If no value is defined 80H will be assumed as default.

remarks: If you use such JW-types for the first time put it into the programmer, read the memory contents and note the value of the last memory address on the device. Doing this you do not lose the right value, which is different to each device, to trim the oscillator to 4 MHz.

CALVAL $C0

ensures that C0H will be written into the OSCCAL-register.

* only if internal RC oscillator is selected.

CLOCK and CLOCK1

(not for 12C50x and 16C5x)

syntax: CLOCK *var*

var is a word variable

function: With the PIC12C67x, PIC16C6x, PIC16C7x, PIC16X8x and PIC 16F87x an exact timebase can be implemented by using the internal timer (TMR0) and its interrupt facilities. The internal variable TIMERX will be incremented each 1/100s. If this variable reaches 100 the user defined variable *var* is incremented. If you are using a xtal frequency below 100kHz the internal interrupt occurs each 1/128s and TIMERX counts up to 128. The variable *var* must be 16 bits.

description: The command CLOCK *var* initializes the internal timebase and invokes the automatic counting, a memory-cell *var* will be increased each second by 1. It doesn't matter how you use the variable. You may read or write the variable *var*

Example:

```
        CLOCK time              REM starts the timebase
START:
        IF time = 60 then GOTO minute       REM 1 minute ist over
        GOTO start              REM
```

This command was expanded to use other periodes than the described 10ms. So you get the opportunity making your clock slower or faster. The range is 1.0 to 999.9ms. But take care about the precision because the prescaler value is fixed.

example : CLOCK time,100 'clock runs 10 times faster

CLOCK uses the timer TMR0 and the scaler.

CLOCK1

If watchdog is enabled and CLOCK is also used then the prescaler is assigned to TMR0. That results in a watchdog periode of about 18 ms which is quite short. It can happen a reset by watchdog because internal and user defined functions may exceed 18 ms. If using the compiler switch $WDTUSR a watchdog reset is very possible. You might solve this problem by inserting CLRWDT commands very carefully to clear watchdog within 18 ms.
But this is not easy. To keep the prescaler assigned to watchdog it's necessary to use another timer module. If TMR1 is implemented into the desired device you can use CLOCK1 to activate this timer instead of TMR0. Because TMR1 has its own prescaler the watchdog periode exceeds to 2.3 seconds.

Some devices gives you the oppertunity to drive TMR1 with a second xtal, connect to T1OSI and T1OSO. These pins are usually I/O pins. But this timer runs also during sleep mode. To define its frequency use keyword CLK1XTAL *value (value* in MHz). If defined iL_BAS16 will select it automatically.

example : CLOCK1 time,100 'clock1 runs 10 times faster

CURSOFF

syntax: CURSOFF

function: turns off cursor.

description: This command turns off the cursor on the lcd.

example:

```
START:
        LCDINIT  rb,2,40
        ...
        CURON           REM cursor on, entry mode
LOOP:
        INKEY var_a
        IF var_a = 0 THEN GOTO LOOP
        LCDWRITE 0,0,var_a   REM write input on lcd
        CURSOFF          REM turn cursor off
```

CURSON

syntax: CURSON

function: turns on cursor.

description: This command turns on the cursor on the lcd.

example:

```
START:
        LCDINIT  rb,2,40
        ...
        CURON           REM cursor on, entry mode
LOOP:
        INKEY var_a
        IF var_a = 0 THEN GOTO LOOP
        LCDWRITE 0,0,var_a   REM write input on lcd
        CURSOFF          REM turn cursor off
```

BINTOASC

(also CONASC)

syntax: BINTOASC *var,buffer*

function: CONASC converts a byte or word variable into an ascii string. Converting a byte variable needs a 3 byte buffer for the result, a word variable generates a string of 5 characters.

description: This instruction is useful for sending the contents of a variable to a monitor for debugging.

Number 123 will be converted. In the buffer you will find the following string 49,50,51,(31H,32H,33H).
var is a 8 or 16 bit variable
buffer is a 3 or 5 byte wide memory location for the result

example:

```
DEFINE a1 = $30 as byte
DEFINE buffer = $31 as byte
...
LET  a1 = 87
BINTOASC a1,buffer                'buffer contains 48  56  55  ("087")
```

BINTOBCD

(also CONBCD)

syntax: BINTOBCD *var,buffer*

function: BINTOBCD changes a 8-, 16- or 32-bit-number into a BCD-packed number. You need a buffer of 5 byte, even only one byte-variable chall be changed.

Number 123 will be converted. In the buffer you will find the following string 00H 00H 00H 01H 23H.

BINTODEC

(also CONDEC or CONDEZ)

syntax: BINTODEC *var,buffer*

var is a 8 or 16 bit variable
buffer is a 3 or 5 byte wide memory location for the result

function: CONDEZ converts a 8- or 16-bit number into a 3- or 5-digit decimal number. For converting you need a 3- respectively 5-byte buffer.

description: This instruction is useful for converting a binary number into a decimal number. The contents of the buffer could then be displayed easily

example:

```
DEFINE a1 = $30 as byte
DEFINE buffer = $31 as byte
...
LET  a1 = 87
CONASC a1,buffer                'buffer contains 0 8 7
```

The number 12345 will be stored in the 5-byte-buffer as 1,2,3,4,5.

DATA

(not for PIC12C5xx and PIC16C5x)

syntax: DATA *const1, const2, const3,....*

(const1, const2,... are byte constants)

function: Defines an array of up to 2048 constants (depends on size of program memory)

description: Very often a large amount of constant-values are necessary. In former compiler releases you had to use LOOKUP and LOOKDWN statements. But they only can handle up to about 100 items. DATA breaks through this barrier and puts the constant to the end of the program in contraty to LOOKUP and LOOKDWN. Read an item with READDATA increments an internal pointer which controls the access. To reset or set this pointer to a specified line is the only pointer modification you need. All items are stored in the program memory.

example:

```
START:
        RESTORE                         REM read pointer, points to the first item
        READDATA   var_A                REM read first item (here 65)
        RESTORE    label_1              REM sets read-pointer
        READDATA   var_A                 REM read 5. item here 12
        READDATA var_B,var_C,var_D       REM read next three items
        GOTO START
        DATA 65,66,67,"F"
lab1l_1:    DATA 12,45,32,17,25
```

remarks: see also RESTORE, READDATA

DEC

syntax: DEC *var*

var is a byte or word variable

function: *var* becomes *var* - 1

description: This command DEC *var* is the code optimized equivalent to LET var = var -1. This command is stored code-optimized in contrary to LET *var* = *var*-1. In contrary to this command DEC doesn't check the validity range of the result.

Example:

```
START:
                LET  var_a = 10          REM variable var_a is set to 10
                DEC var_a                REM in var_a now you have  9
```

remarks: see also INC

DELAY

syntax: DELAY *var*

var is a 8- or 16- bit variable or constant

function: delays program execution with software loops. The resolution is 100us.

decription: The command DELAY holds up program execution for a specified time without entering the controller's sleep mode. The power consumption will not be reduced.

example:

```
START:
        DELAY 5          REM holds up for 500us
        GOTO start        REM
```

remarks: Hold up times between 100us and 6,5s. Because this instruction is done by software delay routines, the maximum hold up time depends on the xtal frequency. With high xtal frequency a 16 bit value is not sufficient (warning).

DOZE

syntax: DOZE *duration*

function: The controller enters the sleep mode for a short time

duration (0 ... 7) duration of sleep phase (2 *duration* x 18 ms), can be a variable or constant.

description: The command DOZE enters the sleep mode for a specified time. In sleep mode the power consumption is reduced down to 20 uA (if no output is driven). The variable or constant *duration* accept values of 0 to 7. Greater values will be truncated.

Following times are avaiable:

DOZE *0*	18 ms	
DOZE *1*	36 ms	
DOZE *2*	72 ms	
DOZE *3*	144 ms	
DOZE *4*	288 ms	
DOZE *5*	576 ms	
DOZE *6*	1.152 ms	
DOZE *7*	2.304 ms	about 2,3s

example:

```
                LET  time  = 7          REM  initialization of  the variable  time (2,3 s)
START:
                DOZE  time               REM sleeps for 2.3 s
                GOTO start               REM and sleep for 2,3 s and sleep for 2,3 s and sleep...
```

DOZE uses the watchdog-timer (must be active) and the prescaler. In case CLOCK is active at the same time, the prescaler will be switched over to the watchdog-timer.

DTMFOUT

syntax: DTMFOUT *port,pin_hf, pin_lf,duration,pause, key1, key2, ..., keyn*

function: generates DTMF tones (e.g. telephone)

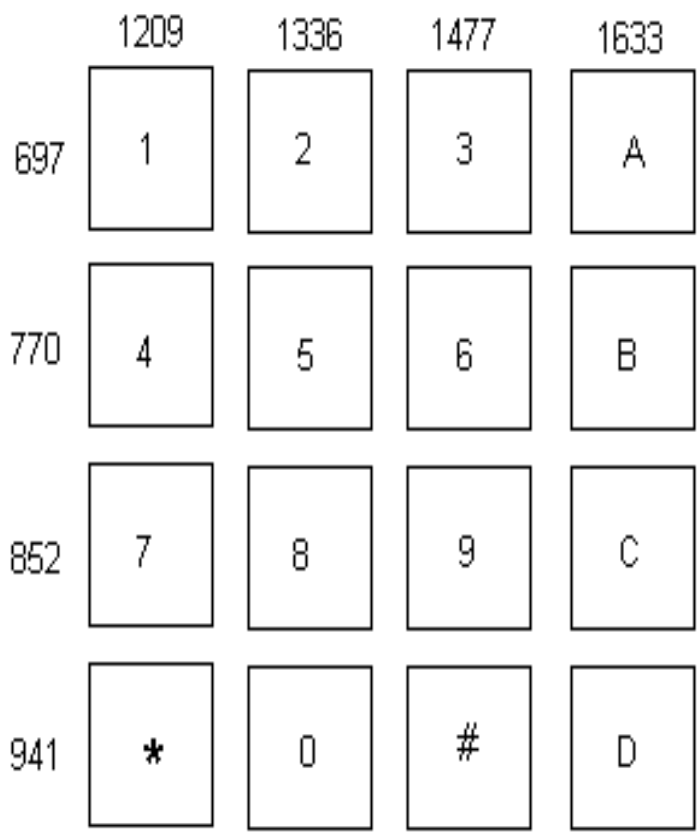
*port*i/o port, variable or constant
*pin_hf*i/o pin where the column frequency is outputted, variable or constant
*pin_lf*i/o pin where the row frequency is outputted, variable or constant
*duration*ca. duration of the tone in msec (must be a constant)
pause ca. duration of the pause between the tones in msec (only constant)
key, keyn(0 ... 15) key(s), variable or contant

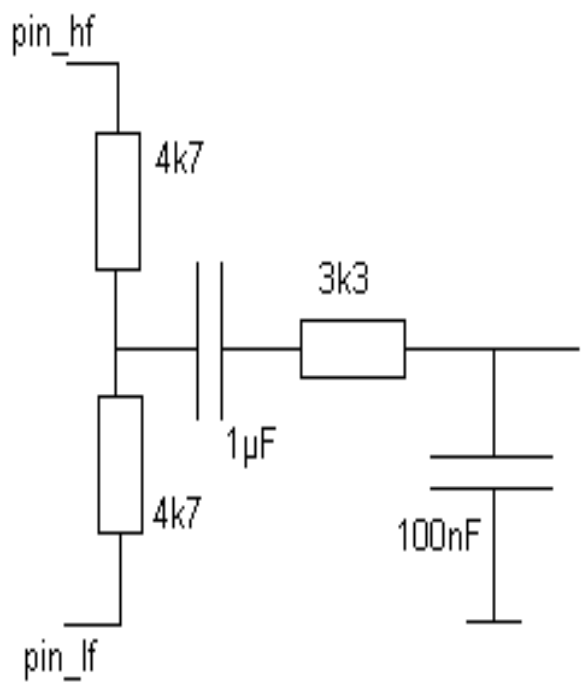
description: The DTMFOUT command output DTMF tones at two port pins. The pins have to belong to the same port and must be wired as shown below. The DTMF tone which is generated is a combination of row and column frequencies. The pins are defined as output, and two different frequencies are generated as long as specified in *duration* . Then a pause follows with the duration of *pause*

example:

```
START:
        DTMFOUT ra.0,1,200,50,10,7,8,3,1,4,5,2    REM generates a sequence
                                                  REM of tones representing
                                                  REM the phone number
                                                  REM 07831 452 at
                                                  REM port ra.0 and ra.1
        GOTO START
```

DTMFOUT (cont.)





row and column frequency in Hz assembly of the output pins

remarks: This command is optimized for a xtal frequency of 12 Mhz. If lower xtal frequencies are used only in some cases it would work fine (e.g. at 4.19MHz, please check others).

decimal value for *key :* 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
equates to keyboard *:* 1 2 3 4 5 6 7 8 9 * 0 # A B C D

EEDATA

(only for PICs with internal eeprom data memory)

syntax: EEDATA(*addr) value1, value2, value3 [,...]* (max. 8-byte-values)

addr is the address where the first byte is stored into
value1, value2 are byte constants

funktion: The constants are programed into the internal eeprom data memory during device programing.

description: Data are written into the internal eeprom data memory by either the WRITE instruction during program execution or during device programing. This allows to put e.g. parameters or passwords into the eeprom data memory without setting them by program operations. Some PIC programers can program the eeprom data memory after the program memory, others need two steps: one to program the program memory another to program the eeprom data memory.

example:

remarks: Not every PIC type has an internal eeprom data memory !

END

syntax: END

function: stops program execution and enters sleep mode to reduce power consumption.

description: Program exectution is terminated to keep the latest output information at the port pins and the sleep mode is entered. This condition can be released only by reset or interrupts if enabled.

example:

```
        LET var_a=0             REM initialization of variable var_a
START:
        . . .
        GOTO START
CONTINUE:

        END                     REM stops execution and enters sleep mode
```

remarks: In sleep mode power consumption is reduced to about 20 uA plus sink and source current of the outputs. If the ports are initialized as output every 2.3s the sleep mode is stoped by the watchdog.

The devices PIC12C50x and PIC16C5x switches the i/o pins to input each 2.3 s when a reset is generated by the watchdog timer.

ENDASM

syntax: ENDASM

function: Assembler code is placed between the keyword ASM and ENDASM.

description: the compiler copies all lines between ASM and ENDASM into the source file without any changes. They will be assembled by iL_ASS16. The syntax of these mnemonics are quite the same as the assembler needs (see iL_ASS16). The first column is reserved for labels. The ENDASM-command must stand alone in a line, even a REM is not allowed. Commands in the same line will not be taken into account. After that ENDASM it will be translated into basic.

example:

```
            ASM
START       MOVLW 00h ;clears port RB
            MOVWF RB
            ENDASM
```

remarks: ASM and ENDASM are commands, so they need one leading space at least. If your assembler porgram crosses pages, you must take care to set the page preselect bits or the PCLATH register accordingly or use LJMP and LCALL.

ERR

syntax: ERR *var*

function: transfers the contents of the internal error variable into the user defined variable *var*

var (8 bit) variable contents the latest error conditions

description: if a arithmetic function is executed, the internal error flag (overflow) is cleared. If the result exceeds the variable's value range this overflow flag is set, in this case var has the value of 1. To check this flag, you should copy the contents of that internal variable into your own by using this command. The internal error variable contents more information than it is described for the overflow flag. Their contents is bit orientated with the following jobs.

Bit 7 overflow in arithmetic functions (will be cleared before operation begins)
Bit 6 used by the compiler to find the return address in runtime library (only
Bit 5 PIC12C5x and PIC16C5x, because they only have a two level hardware
Bit 4 stack, which is not sufficient.
Bit 3 timeout occurs in SERIN, PULSIN or I2C
Bit 2 counts how deep the GOSUB routines are nested because
Bit 1 this cannot be found out during compilation.
Bit 0 000 = no UP, 001 = 1 UP, 101 = 5 UP =that is too much.

remark: ERR *value* puts the error code to the value *value.*

FOR TO NEXT

syntax: FOR *var* = *start* TO *end* ..... NEXT *var*

function: executes program loops; nesting depth is 16 levels maximum.

var (8 bit)	is the counting variable
start (8 bit)	start value of *var* when entering the loop, can be variable or constant
end (8 bit)	termination value, if reached, program exits loop (variable or constant)

description: FOR-NEXT loops are program loops which capsules a series of program statements which must be repeated serveral times. At the beginning the counting variable *var* is set to the initialization value *start* . The following statements are executed until the keyword NEXT is found. NEXT increments the counter variable *var* . Then program execution is continued at the statement following FOR-TO until the counter variable reachs the value of *var* . The increment step is +1 and fixed. Only sixteen FOR-NEXT loops can be nested. The initialization and termination value can also be expressions, not only variables and constants. The same restrictions such as the keyword LET are vaild.

example:

```
START:    FOR var_a = 0 TO 5
          GOSUB read              REM wait for external signal
          NEXT var_a
          END

READ:     IF ra.0 = 1 THEN GOTO store
          PULSOUT ra.1,5          REM no signal, puls out at RA,1
          RETURN
STORE:    LET var_b = 1           REM signal ok, memorize
          RETURN
```

remarks: *var, start* and *end* are 8 bit variables (0-255). The loop is executed at least one time regardless of the values of *start* and *end*

From version 4.20 onwards leaving FOR-NEXT loops with the help of GOTO is allowed and do not cause any program confusion as in former releases, independent of the PIC you use (also for PIC 12C5xx respectively 16C5x).

FREQIN

syntax: FREQIN *port,pin,time,var*

function: measures the frequency of a signal connected to a certain *pin* . The measuring *time* is selectable.

port (8 bit)	port name (RA, RB...)
pin (8 bit)	number of the entrance-pin (0 .. 7)
time (16 bit)	constant (max. 10000)
var (16 bit)	variable to assimilate the result

description: Measuring time must be a constant value from 1000 to 10. While 1000 results in a solution of 1 Hz, a 10 results in a solution of 100Hz. The real measuring time is only the half (e.g. 500ms if 1000) because both edges (falling and rising) are counted.

example:

```
START:
        FREQIN RB,1,1000,var_s      REM counts the rising and falling edges at RB,1
          REM during 500ms. Because both
          REM edges will be counted
          REM this corresponds to the frequency.
```

Attention!

This command can be used for xtal frequencies up to 9.9 MHz. If using a higher frequency an error message will be shown. Neverthenless you could use FREQIN at higher frequencies if *time* is reduced. In this case the result must be corrected. The result of the multiplication of *time* and *xtal* may not exceed 9900.

GOSUB

syntax: GOSUB *addr*

function: calls subroutines, nested four times maximum at PICs with 14 bit core. PICs with a 12 bit core have no enough stack space, so nesting of GOSUBs are not allowed.

addr label used as target

description: Jump to a subroutine. The statement GOSUB stores the address of the next following statement on the top of the stack. Then the program branches to the subroutine. At the end of the subroutine the keyword RETURN forces the program to branch to that statement whose address is stored on the top of the stack.

example:

```
START:      FOR var_a = 0 TO 5
            GOSUB read                  REM branches to subroutine
            NEXT var_a
            END
             REM here is the subroutine
             REM with the inquiry of an external
             REM signal and a PULS-output

READ_X:     IF ra.0 = 1 THEN GOTO memory
            PULSOUT ra.1,5
            RETURN
              REM signal recognized, remember

MEMORY:     LET var_b = 1
            RETURN
```

GOTO

syntax: GOTO *addr*

function: branches to a target label, unconditionally

addr Label used as target

description: This compiler iL_BAS16 does't need any line numbers. Targets of branches are marked with labels (*addr*). A colon terminates this label. The statement GOTO continues program execution at the statement following the corresponding label *adr*. If GOTO is used in IF-CLAUSES it is obligatory and must not be left out.

example:

```
START:
            INPUT ra,var_a
            IF var_a = 1 THEN GOTO mark_1
            ...
            GOTO start
MARK_1:     ...
```

HIGH

syntax: HIGH *port,pin*

function: Changes the tris register to configure that *pin* at *port* as output (changes the TRIS-register) and sets the output signal to high (1).

*port*i/o port, can be a variable (8 bit) or constant
*pin*i/o pin , can be a variable (8 bit) or constant

description: The statement HIGH sets the corresponding bit within the tris register to zero so that the pin becomes an output and then sets the output level to high (1). This pin remains output until changed by a TRIS or INP statement.

example:

```
                LET var_a = 1     REM variable a = pin 1
START:           INPUT ra,var_a                  REM pin 1 is still input
                IF var_a = 0 THEN GOTO mark_1
                HIGH ra,var_a     REM pin 1 becomes output and 5V
MARK_1: ...
```

remarks: Do not use the statement HIGH for other variables than *port* variables. To set a bit within a regular variable use the statement SET.

I2CDELAY

syntax: I2CDELAY *value*

function: slows down the clock frequency of SCL at the I2C bus

value only constants, number range of 1 to 99 (default = 2)

description: If the I2C bus is connected to long wires it will be an advantage to slow down the transfer rate. Because the I2C bus is a synchronous bus slow down the clock on SCL, that is enough. Another reason to slow down transfer rate is a slow slave device, e.g. if you define a PIC as your own slave device with a high flexibility but slow rate.

example 1:

```
                DEFINE sda=2
                DEFINE scl=3
                I2CINIT rb,sda,scl          REM Controller works as master
                I2CDELAY 4
START:
                ....
.               ...
MARK_1:         ...
```

example 2:

```
                DEFINE sda=2
                DEFINE scl=3
                I2CINIT rb,sda,scl,SLAVE,ADR=170,BUFFER=$1c     REM PIC is slave

START:
                ....
                ....
 MARK_3:         ...
```

I2CHARDS

syntax: I2CHARDS ADR=*addr,* RXP=*rxp,* RXB=*rxb,* TXP=*txp,* TXB=*txb*

function: Activates the i2c hardware modul (if available) to function as a i2c slave.

addr defines the device address by which the i2c master can access this slave, must be a constant.
rxp pointer for the receive buffer, must be a variable
rxb receive buffer address (first location), can be either a variable or constant
txp pointer for the transmit buffer, must be a variable
txb transmit buffer address (first location), can be either a variable or constant

describtion: This comand uses the i2c hardware modul of some pics. If this hardware is not implemented in the selected device an error message will occur. The communication between master and this slave runs interrupt controlled in the background. *addr* is the device address used by the master to start communication. The last significant bit of *addr* must be 0. *rxp* and *txp* are pointer variables used by the receiving routine (*rxp*) and transmitting routine (*txp*). If there is an address match (the slave is that one the master calls) both pointers (*rxp* and *txp*) are reset to the first buffer location and then all bytes the slave receives are stored in the receiving buffer (*rxb*). In case the master reads data out of the slave all bytes come out of the transmit buffer (*txb*). At the end of communication program control is back to the user. Now you can check the transmit or receive status, modifiy data, trigger ad converter etc.. Checking the pointer during normal program operation gives also information about the i2c status. Turn on and off the i2c hardware-slave modul is done by setting and resetting the interrupt enable bits GIE and PEIE. Before enableing the i2c slave you should reset the SSPIF flag.

example:

```
                xtal 4.194304
                DEFINE device 16C74, wdt_off, xt_osc, adcfg0
                DEFINE rxpointer = $50          'must be in bank 0
                DEFINE txpointer = $51          'must be in bank 0
                DEFINE rxbuffer = $A0           'must be in bank 0 or 1
                DEFINE txbuffer = $B0           'must be in bank 0 or 1
                DEFINE stack = $60              'necessary cause interrupts
                I2CHARDS adr=100,rxp=rxpointer,rxb=rxbuffer,txp=txpointer,txb=txbuffer
cold:
                set intcon,6                    'release PEIE (same as SET PEIE)
                set intcon,7                    'release GIE (same as SET GIE)
                res sspif
loop:
                goto loop
                ....

or
cold:
        set intcon,6        'PEIE freigeben (auch SET PEIE)
        set intcon,7        'GIE freigeben
        asm
        bcf 0ch,3           'SSPIF-Flag löschen
        endasm
        res sspif           'alternativ zu bcf 0ch,3
loop:
        goto loop           'wartet endlos, da Interruptbetrieb
        let st_fadl=5
        let syn_asy=6
```

I2CHARDS (cont.)

end

remark:

The used pointer in I2CHARDS library is truncated to FFH. If more bytes are read or written the library accesses always to address FFH (bank1).

I2CINIT

syntax: (MASTER) I2CINIT *port,sdapin,sclpin*
syntax: (SLAVE) I2CINIT *port,sdapin,sclpin,SLAVE,ADDR=addr,BUFFER=addr (,WAIT=port,pin*

function: configures the *port* and *pins* as a bus for I2C

port i/o port, can be a variable or constant (8 bit)
sdapin i/o pin it becomes SDA (data line) of the I2C bus
sclpin i/o pin it becomes SCL (clock line) of the I2C bus
to implement the slave function more arguments are necessary
SLAVE links the slave routine to the program.
ADDR=addr device address, can be variable (8 bit) or constant
BUFFER=addr defines the start address of the transmit and receiving buffer

description: This statement changes the tris register. If in master mode these pins become output, in slave mode they become input. In master mode the output levels are set to default levels.

example 1:

```
                DEFINE sda=2
            DEFINE scl=3
            I2CINIT rb,sda,scl          REM PIC is master
START:
            ....
            ....
MARK_1:
             ...
```

example 2:

```
            DEFINE sda=2
            DEFINE scl=3
            I2CINIT rb,sda,scl,SLAVE,ADDR=170,BUFFER=$1c
rem PIC is slave master calls this slave at adr. 170 all received data are stored in 1CH to 7FH.
START:
            ....
            ....
MARK_1:
            ...
```

remark to example 2
The device address is 170 for writing and 171 for reading. You have to take care that there will be no conflict with other devices. The communication buffer has 4 bytes if you use a 16C84 (1CH to 1FH). In this case, the master is allowed to write 4 bytes into this slave (see I2CSLAVE).

remark: During program execution you may not change the according bits in the tris register. The command I2CINIT is a bit-orientated command. Using a 4 MHz crystal you get a SCL frequency of about 30kHz. A 12 MHz crystal is sufficient to get the standard clock rate of SCL (100kHz). Having 20 Mhz you reach the maximum of the bit rate.

The switch WAIT isn't implemented at the present.

Corresponding the I2C-specification you have to connect an pull-up-resistor to both bus SCL and SDA wire because the software gives the slave-module the possibility to insert a WAIT-cycle. therefor SCL will be set up as an input.

I2CINIT (cont.)

remember:

High level on the I2C bus is not active driven but generates by pullup resistors.

I2CRD

syntax: I2CRD *addr1, var*

function: reads the value out of the module with the adress I2C *addr1* and hands it over to the variable *var.* At that read command no start- or stop-condition is generated.

adr1 I2C-module-adress
var 8-bit-variable, read value will be stored there.

description: Reads a value out of the I2C-module and hands it over to the variable *var* . Usually memory-modules have a auto-increment-device. When you want to read value out of another than the following adress you have to reset the adress counter of the module with the help of I2CWR

example:

```
                I2CINIT ra,2,3          REM  I2C init.
Start:          I2CSP                   'STOP condition
                I2CST                   'START condition
                I2CWR 160,2             REM Seconde at the RTC PCF 8583P
                I2CRD 160,var_a         REM read secondes
                I2CRD 160,var_b         REM read minutes
                I2CRD 160,var_c         REM read hours
```

I2CREAD

syntax: I2CREAD *addr1, var*

function: reads a byte out of a I2C device with the device address *addr1* and stores it in the variable *var*

addr *I2C*-device address
var 8 bit variable, to store the byte which is read.

description: First the device address is transmitted to all I2C slaves. The device which gets an address match acknowledges the reception. Now the slave device transmits the data byte to the master. If you are using standard memory devices, please remember that an autoincrement sets the internal address pointer to the next memory cell. For sequential read or write no special attention is necessary. But if you want random read you should set the internal address pointer to the desired memory address by using the I2CWRITE statement.

example:

```
            I2CINIT ra,2,3          REM I2C init.

START:      I2CWRITE 160,2          REM addr. With the sec at PCF 8583P
                                    REM set the internal address pointer to the desired address (here 2)
                                    REM this is sequential read
            I2CREAD 160,var_a       REM read the seconds, store it in var_A
            I2CREAD 160,var_b       REM read the minutes
            I2CREAD 160,var_c       REM read the hours
                  ....
            ....
Marke_1     ....
```

I2CSLAVE

syntax: I2CSLAVE

function: Communicate with master via a buffer. The direction of that communication is defined by the R/W bit in the device address.

description: The PIC controller becomes an intelligent slave device (I2CSLAVE). This statement branches to the runtime library. Here it stays in a loop until the address is received. If this address isn't the right one, the program leaves the runtime library with bit 3 set in the internal error variable. In case of address match the further processing depends on the read and write bit in the received address. In one case the slave transmits the contents of the communication buffer, in the other case it receives bytes form the master and stores them into the communication buffer (everytime it's the same buffer). At the end of communication (teminated by a missing ACK bit) the program returns from runtime library. The concept of the communication buffer was choosen because of its flexibility. After return from the runtime library the user can examine bit 3 of the error variable. If this bit is zero (address match), the user can examine the communication buffer if the master has sent datas. If the master has read the buffer, now it can be modified for the next read cycle. Because this function is only implemented in software the transfer rate is relativly low. At 4 MHz it is about 35kHz, at 12MHz it is enough for standard clock rate (100kHz).

example:

```
REM reads 4 bytes from I2C master and writes them into a lcd
            DEFINE ERR=ERR
            DEFINE buffer = $20 as byte
            DEFINE var_a = $20 as byte
            DEFINE var_b = $21 as byte
            DEFINE var_c = $22 as byte
            DEFINE var_d = $23 as byte
            I2CINIT ra,2,3,SLAVE,ADR=170,BUFFER=buffer      REM I2C init.
            LCDINIT rb,1,2
            LCDCLEAR
START:
            IC2SLAVE
            IF ERR,3=1 THEN GOTO START          REM if wrong address
            LCDWRITE 1,1,var_a,var_b,var_c,var_d   REM these variables are within the buffer
```

remarks: see I2CDELAY

I2CSP

syntax: I2CSP

function: creates a STOP-condition

description: Neither a START- or a STOP-condition will be generated with the commands I2CRD and I2CWT. They must be created at the corresponding position with I2CST and I2CSP.
Why?
There are I2C-modules with which you can't use the command I2CREAD respectively I2CWRITE because they have a "modified" protocol. You also have to use these single commands to implement the function REPEATED START.

example:

```
                I2CINIT ra,2,3              REM  I2C init.

START:     I2CSP                    'STOP  condition
           I2CST                    'START condition
           I2CWR 160,2              REM  second at RTC PCF 8583P
           I2CRD  160,var_a          REM  read second
           I2CRD  160,var_b          REM  read minute
           I2CRD  160,var_c          REM read hour
           I2CSP                    ' STOP condition
```

I2CST

syntax: I2CSP

function: creates a STOP-condition.

description: Neither a START- or a STOP-condition will be generated with the commands I2CRD and I2CWT. They must be created at the corresponding position with I2CST and I2CSP.
Why?
There are I2C-modules with which you can't use the command I2CREAD respectively I2CWRITE because they have a "modified" protocol. You also have to use these single commands to implement the function REPEATED START.

example:

```
          I2CINIT ra,2,3      REM  I2C init.
START:    I2CSP               'STOP condition
          I2CST               'START condition
          I2CWR 160,2         REM seconde at RTC PCF 8583P
          I2CRD 160,var_a     REM read secondes
          I2CRD 160,var_b     REM read minutes
          I2CRD 160,var_c     REM read hours
          I2CSP               'STOP condition
```

I2CWR

syntax: I2CWR *addr1, var*

function: writes to a I2C device with the address *addr1* to the memory address *addr2* the contents of the variable or the constant.

Write without creating a START- or STOP-condition.

addr1	I2C device address
addr2/data	value for the internal address pointer
var/const	data bytes to store sequentially in the device at address adr2

description: The constant const or the contents of the variable var is written into the I2C device at address addr2. The selection of the modules takes palce at the I2C-bus itself by its module address *addr1* . Although memory devices most of the modules have an autoincrement build inwhich means the internal addresscounter will be increased automatically during a read- or write-access by one, you must set addr2 (the desired address) at each I2CWR statement.

example:

```
          I2CINIT ra,2,3      REM  I2C init.
START:    I2CSP               'STOP condition
          I2CST               'START condition
          I2CWR 160,2         REM second at RTC PCF 8583P
          I2CRD 160,var_a     REM read second
          I2CRD 160,var_b     REM read minutes
          I2CRD 160,var_c     REM read hours
          I2CSP               'STOP condition
```

I2CWRITE

syntax: I2CWRITE *addr1, addr2, var/const*

function: writes to a I2C device with the address *addr1* to the memory address *addr2* the contents of the variable or the constant.

addr1	I2C device address
addr2/data	value for the internal address pointer (if RAM otherwise any data byte)
var/const	data bytes to store sequentially in the device at address adr2 (if RAM otherwise it is the second data byte).

description: At first the master writes the device address to the I2C bus. If an address match occurs the next data byte is written into this device. If this is a memory device that byte is interpreted as an address and written into the internal address pointer. The constant const or the contents of the variable var is written into the I2C device at address adr2, if the selected device is a memory device.
Otherwise the byte following the device address is a regular data byte. Although the memory devices have an autoincrement build in, you must set addr2 (the desired address) at each I2CWRITE statement.

example:

```
            I2CINIT ra,2,3              REM I2C init.

START:      I2CWRITE 160,2              REM write second at RTC PCF 8583P
            I2CREAD 160,var_a           REM read second
            I2CREAD 160.var_b           REM read minutes
            I2CREAD 160,var_c           REM read hours
                ....
                ....
MARK_1
```

IF-THEN-ELSE

syntax:

IF *var* = *condition* THEN *addr*
IF *var* = *condition* THEN *addr1* ELSE *addr2*
IF *port,bit* = 0 THEN *addr1* ELSE *addr2*
IF *port,bit* = 1 THEN *addr1* ELSE *addr2*

function: conditional branch to label *addr* , if condition match.

port,bit	(8-Bit) port and bit number 0 - 7
var	(8/16-Bit) variable to compare
condition	(8/16-Bit) reference value, can be a variable or constant
adr	target address (Label), to branch

description: IF compares the variable *var* with another value *condition* and branches to the target address *addr* (label), if true

Following comparision are available:

=	equal
<>	not equal
<	less than
>	greater than
<=	less or equal
>=	greater or equal

Attention: having a bit-comparison only the operator '=' is valid !!!

example:

```
IF var_a <= 5 THEN LET var_b = 0 ELSE GOTO loop
IF RA,1 = 0 THEN GOTO loop
```

wrong:

```
IF var_a <= 5 THEN LET var_b = 0 ELSE var_b = 1: GOTO loop
```

remark: If THEN or ELSE are followed by a equation, it must begin with LET . If branches are following you have to use the keywords GOTO or GOSUB . It is understood that only one equation is allowed behind THEN and ELSE.

For multilines the command ON *var* GOTO respectively ON *var* GOSUB is at your disposal.

INC

syntax: INC *var*

function: *var* becoms *var* + 1

description: The command DEC *var* increments the value of the variable *var* by 1. This command is the code optimized equivalent to LET var = var +1. In contrary to command INC which doesn't have regard for it checks the validity range ot the result.

example:

```
START:
        LET var_a = 10      REM variable var_a = 10
        INC var_a           REM var_a = 11
```

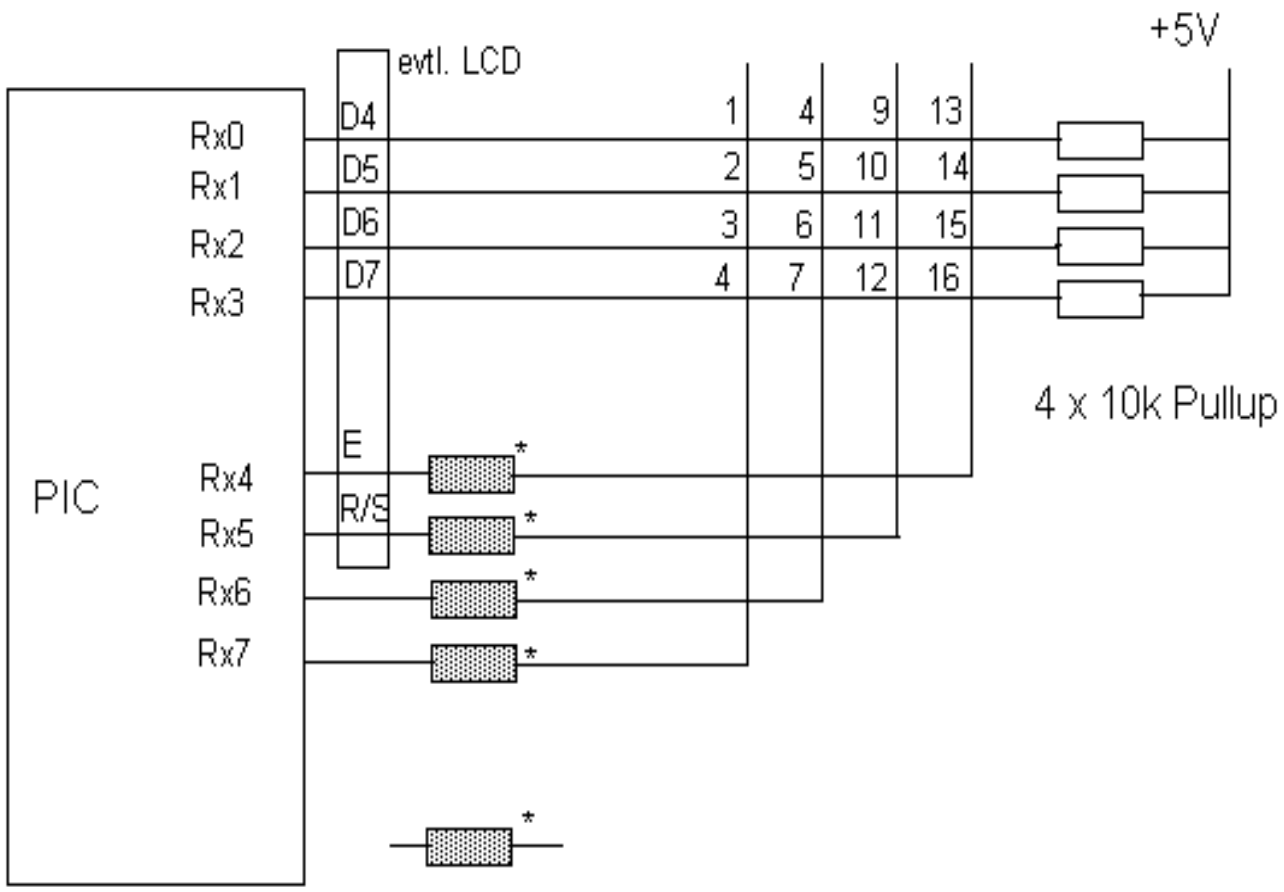
remarks: see also DEC

INKEY

syntax: INKEY *var*

function: reads a 4 x 4 (4 x 3) key matrix and returns a 8 bit value

description: A 4 x 4 key matrix is scanned and the corresponding code is returned into the variable *var* . The key matrix has to be connected in a fixed fashion at port RB or if available at RC or RD. This fixed wiring allows to connect a LCD at the same port, too. In this case the kex matrix is reduced down to 4x3 because the signal E of the LCD may not be used for scanning the matrix. Additional resistores are necessary for decoupling (see schematic). The command INKEY initializes the tris registers to input and output as required. At the end of this command the tris registers is restored by the original value. This is important for the lcd functions. The original output levels are not restored.



* decoupling resistors necessary if a lcd is connected to the same port

To define the desired port and the conversation table use the statement DEFINE KEYS.
Rx4 is not used if LCD is connected at the same port

example:

```
        DEFINE KEYS rb,0,"0","1","2","3","4","5","6","7","8","9", {+
                        "A","B","C","D","E","F"

START:
        INKEY var_a                 REM read key matrix
```

INKEY (cont.)

IF var_a=0 THEN GOTO start REM no key pressed?
MORE:
...

remarks: see also DEFINE KEYS

Between two INKEY commands it must be a delay of 10ms. If this delay is not reached by the other BASIC commands insert a WAIT 10

INP

syntax: INP *port,pin*

function: configures *pin* at *port* as input

port i/o port, can be a variable or constant (8 bits)
pin i/o pin, can be a variable or constant (8 bits)

description: The INP command changes the tris register to get the desired *pin* at *port* as an input. Now this pin can be read by INPUT or IF.

example:

```
        INP ra.0                    REM pin RA.0 is input
START:
        IF ra,0 = 1 THEN GOTO more          REM if a high is at pin RA.0 then branch to MORE

        GOTO start                  REM else jump to START
MORE:
        ...
```

remarks: INP is a bit command

INPUT

syntax: INPUT *port,var*

function: reads the whole *port* into *var* (8 bits)

port i/o port, can be a variable or constant (8 bits)
var (8-Bit) read value transfered into the variable *var* or constant

description: The logic levels at a port are read (all 8 bits at once) and stored in var. The tris register will not be changed.

example:

```
            TRIS ra=15
            INP ra.0                        REM port RA as input
START:
            INPUT ra,var_a
            LET var_a = var_a and 1         REM only bit 0 is relevant
            IF var_a = 1 THEN GOTO more     REM if RA.0 is high, branch to MORE
            GOTO start                      REM jump to START
MORE:
            ...
```

remark: INPUT is a byte command

INTERRUPT

syntax: INTERRUPT *label,level (,gied)*

label	branch address at which starts the interrupt program
level	relevant edge in case of a RB0-interrupt (manipulated OPTION-register), HL or LH
gied	switches off the interrupts before each BASIC-command, after that they are switched on again.

function: tells the compiler that somewhere in the program an user defined interrupt service routine will be found and must be put in the internal interrupt chain.

description: The user defined interrupt service routine (ISR) starting at *label* is queing in the internal interrupt chain. The *level* argument defines the slope at which an interrupt should occur (pin RB0). This information is stored in the option register. The *gied* argument, which is optional, solves the reentrance problem. This problem appears every time when an interrupt occurs while the controller executes a BASIC command that uses the internal register ARG0, ARG1 etc. If the interrupt occurs at that time the actuall program is stopped and the ISR will be started. If this ISR also uses the internal register ARG0, ARG1 etc their contents will be corrupted for the main routine because the original contents is not saved. To store these registers, a stack is necessary. The stack cannot be implemented to PICs with less data memory. So *gied* is the only way to go. If active, the GIE bit (global interrupt enable) at INTCON register is set to zero at the beginning of a BASIC statement and set to one at the end. So all interrupts are disabled during the execution of a BASIC statement. This prevents data corruption in the registers ARG0, ARG1 etc but on the other hand it stretches the interrupt response time with a non constant factor.

See also chapter INTERRUPTS

example:

```
        define device 16c84
        xtal 4.19
        INTERRUPT intserv,hl,gied       REM intserv = name of ISR, hl = falling slope at RB0

START:

        GOTO start                      REM keep in main loop
        ...
        INTPROC
INTSERV:
        ...
        ...
        INTEND
```

remarks: The INTERRUPT command is not available with PIC12C50x and PIC16C5x . (see also chapters INTERRUPTS and DEFINE STACK)

INTEND

syntax: INTEND

function: marks the end of the interrupt service routine (ISR).

description: The ISR will be left by the RETFIE command. This is necessary because the ISR is constructed like a subroutine. The regular return from a subroutine with RET or RETLW cannot be used here because these commands do not restore the GIE bit.

example:

```
            define device 16c84
            xtal 4.19
            INTERRUPT intserv,hl,gied        REM intserv = name of ISR, hl = falling slope at RB0

START:          ....

            GOTO start                       REM keep in main loop

            INTPROC
INTSERV
                ...
                ...
            INTEND                           REM teminates ISR with RETFIE
```

remarks: The INTEND command is not available with PIC12C50x and PIC16C5x . (see also chapters INTERRUPTS and DEFINE STACK)

INTPROC

syntax: INTPROC

function: The lines between INTPROC and INTEND belongs to the user defined interrupt service routine.

description: identifies the following lines as the ISR. The starting address is put into the internal interrupt chain.

example:

```
            define device 16c84
            xtal 4.19
            INTERRUPT intserv,hl,gied        REM intserv = name of ISR, hl = falling slope at RB0

START:
               ....


            GOTO start                       REM jumps to START

            INTPROC                          REM ISR follows
INTSERV:

            ....
            ....
            ....
            INTEND
```

remarks: The INTPROC command is not available with PIC12C50x and PIC16C5x . (see also chapters INTERRUPTS and DEFINE STACK)

LCDCLEAR

syntax: LCDCLEAR

function: clears the contents of the LCD and returns cursor home (position 1,1)

description: The special clear command is sent to the lcd.

example:

```
START:
        LCDCLEAR
        GOTO start      REM jump to START
MORE:
            ....
```

remarks: The LCDCLEAR command does not check wether a lcd is connected or not.

LCDDELAY

syntax: LCDDELAY *const*

function: increase the delay times within the lcd commands.

description: The commands LCDINIT, LCDCLEAR and LCDWRITE do not check the BUSY bit of the lcd. Instead a delay loop prevents data collision in the lcd device. The data sheet shows a wide range of delays especially with the commands LCDCLEAR and HOME. The compiler generates a delay of about 1ms which could be too less for some displays. The command LCDDELAY *const* tells the compiler how many times the delay has to be repeated. The value of *const* is between 1 to 99, the default value is 2 (*).

example:

```
START:
        LCDDELAY 3      REM delay is now 3ms

        GOTO start      REM jump to START
MORE:
          ...
```

* depends on xtal frequency. Sometimes only up to 6 or 7 possible (compiler stops with error message)

LCDINIT

syntax: LCDINIT *port,lines,columns (,NBLK) (,CURSON) (,UPLINES)*

function: sends the initialization string to the LC-display

port	i/o port, must be a constant (e.g. RB)
lines	numbers of lines of the lcd (1,2 or 4)
columns	numbers of columns of the lcd (8, 16, 20, 24, 32, 40)
NBLK	no blank = leading zeros within decimal numbers are visible
CURSOR	turns cursor on
UPLINES	alternative connection of the lcd (see below)

description:
A LCD gets the suitable initialization string. The controller of the lcd device must be a HD44780. To minimize the numbers of used i/o- pins, the lcd is used in a 4 bit mode. Also the R/W pin of the lcd is tied to ground so that no reading of the lcds internal RAM is possible.

Following lcd are supported:

numbers of lines	numbers of columns
1	8, 16, 20, 24, 32, 40
2	16, 20, 32, 40
4	16, 20

The lcd may be connected at port RB, RC or RD. The following assignment of signals and pins is fixed.

Rx0	D4	(11)	(in 4 bit mode, only the upper nibble is used)
Rx1	D5	(12)	
Rx2	D6	(13)	
Rx3	D7	(14)	
Rx4	E	(6)	
Rx5	R/S	(4)	
	GND	(1 u.5)	
	Vcc	(2)	
	Vo	(3)	(contrast)

example:

```
            LCDINIT   RB,4,20       REM LCD initialization
START:
            LCDWRITE 1,1,"Ing.Buero LEHMANN"
            LCDWRITE 2,1,"Fuerstenbergstr. 8a"
            LCDWRITE 3,1,"Phone 0049 7831 452"
            FOR var_a=0 to 255
               LET var_t=var_a*var_a          REM calculate square of A1
               LCDWRITE 4,1,var_a             REM show A on lcd
               LCDWRITE 4,15,var_t            REM show square of A
            NEXT var_a
            GOTO start                        REM loops to START

MORE:
                     ....
```

LCDINIT (cont.)

An alternative connection for the lcd!

Since release 5.5-10 you can connect the lcd to a port in two alternative ways. In this case pins Rx0 and Rx1 are not connected. This is very interesting because the RB0 interrupt is available for other functions. In this case no matrix key pad can be connected parallel to the lcd. Connection is done as followed.

UPLINES	UPLINESX
Rx2 = E (Pin 6)	Rx2 = R/S (Pin 4)
Rx3 = R/S (Pin 4)	Rx3 = E (Pin 6)

the rest of the pins are tied in the same way either for UPLINES or UPLINESX

Rx4 = D4 (Pin 11)
Rx5 = D5 (Pin 12)
Rx6 = D6 (Pin 13)
Rx7 = D7 (Pin 14)
(Pin 5 = RW must tied to GND)
(Pin 1 = GND)
(Pin 2 = Vcc = +5V)
(Pin 3 = Vo = contrast)

To select this connection scheme use keyword UPLINES (upper lines) or UPLINEX (upper lines crossover) with LCDINIT. E.g.

```
LCDINIT rb,4,20,uplines
```

remarks: The contents of the tris register is set automatically. They may not be changed during program execution.

LCDTYPE

n (,var)

syntax: LCDTYPE *n (,var)*

function: defines how the lcd is connected to the PIC

n	constant from 0 to 4
var	additional variable (must be located at page 0) necessary for *n* =3 and *n*=4

description: Many customers asked for implementation of an lcd via a serial-parallel converter e.g. 74LS164. To select the different kind of lcd connection is done by the new BASIC keyword LCDTYPE n (,var).

LCDTYPE 0 standard connection as described under LCDINIT. This is a default value
LCDTYPE 1 as LCDTYPE 0 and UPLINES (UPLINES are also selectable with LCDINIT command)
LCDTYPE 2 as LCDTYPE 0 and UPLINESX (UPLINESX are also selectable with LCDINIT command)
LCDTYPE 3,var connections see below
LCDTYPE 4,var connections see below

Connections between lcd, PIC and 74LS164 must be done as followed:

```
LCD pin -->  74LS164 pin    PIC pin    signal name
Vss  1                                 ground
Vdd  2                                 +5V
Uo   3                                 poti (contrast)
RS   4         Q7    13
R/W  5                                 ground
E    6                    enable
D0   7         Q0     3
D1   8         Q1     4
D2   9         Q2     5
D3  10         Q3     6
D4  11         Q4    10
D5  12         Q5    11
D6  13         Q6    12
D7  14                                 ground
               D1     1    data
               D2     2                +5V
               GND    7                ground
               CLK    8    clock
               MR     9                +5V
               U+    14                +5V
```

Along with this kind of lcd controlling there a serveral disadvantages:
- Only 1- or 2-lines lcd are useable. Character per line can be 8, 16, 20, 24, 32 or 40.
- Cursor is not locatable. LOCATE y,x produces a syntax error.
- Only LCDWRITE 1,1,.... or LCDWRITE 0,0,.... is usable.
  To write into the second line you must write more characters as defined in columns (LCDINIT)
- Writing to lcd is much slower.
- An additional variable is necessary.
- No keypad can be connected to the same port.
- Only characters within the range of 0 to 7FH can be used. No special characters are available.

LCDTYPE n (,var) must be placed above LCDINIT !

LCDTYPE (cont.)

The additional variable must be located on page 0.

examples:
LCDTYPE 3,var : LCDINIT *port,clock,lines, columns*
LCDTYPE 4,var : LCDINIT *port,clock,data,enable,lines, columns*
port can be any available port. *clock* defines the connection to pin 8 at 74LS164, data is for pin 1. *enable* is connected directly from PIC to lcd (see LCDINIT).

E.g.
If LCDTYPE 3,var is selected and LCDINIT rb,5,2,16 then *clock* signal is on RB5, *data* on RB6 and *enable* on RB7, the lcd has 2 lines and 16 characters/line.
If you wish *clock* on RB7, *data* on Rb3 and *enable* on RB0 then use LCDTYPE 4,var and LCDINIT rb,7,3,0,2,16.

clock , *data* and *enable* must be 8 bit constants with values of 0 to 7.

Hint:
LCDWRITE 0,0,.... continues at the last cursor position.

LCDWRITE

syntax: LCDWRITE *y,x,"TEXT",varhex,$,varascii,#,vardez,varbin,%,vardez,:2*

function: writes texts, variables and contants onto the lcd.

y,x	sets the cursor to column and row from which on the output will be seen on LCD
"TEXT"	text string to be outputted
varhex,$	variable displayed as hexdecimal number
varascii,#	variable sent as ASCII character to lcd
varbin,%	variable displayed as binary number
vardez	variable displayed as decimal number
vardez,:2	variable displayed as decimal number with 3 digits before and 2 digits behind a point (quasi float appearence)

decription: Normally variables are displayed in decimal notation. A quasi float number appears on the lcd if you add a colon with the numbers of digit for the post decimal position. Adding a formating character, separated by a comma, the variable can be displayed as a hexdecimal or binary number. Behind the colon the number of post decimal positions. This statement must be a constant.

The numbers of digit are fixed:
see:

8 bit hexadecimal	2 digits
16 bit hexadecimal	4 digits
8 bit binary	8 digits
16 bit binary	16 digits
8 bit decimal	3 digits (maybe 4 digits)
16 bit decimal	5 digits (maybe 6 digits)

Doing so you can write numbers flush-right one below the other without problems.

example:

```
        LCDINIT   RC,4,20            REM LCD initialization
START:
        LCDWRITE 1,1,"Ing.Buero LEHMANN"                    REM upper left corner
        LCDWRITE 2,1,"Fuerstenbergstr. 8a"   REM second line
        LCDWRITE 3,1,"Phone ++49 (0)7831  452"
        FOR var_a=0 to 255
            LET var_t=var_a * var_a          REM calculate square
            LCDWRITE 4,1,vaa_a               REM output a as decimal value
            LCDWRITE 4,15,var_t              REM display square
        NEXT var_a
        GOTO start                                  REM jump to START

MORE:
                ....
```

remarks: The contents of the tris register is set automatically. They may not be changed during program execution.

If you want to write to the cursor's next position use the coordinates 0,0.

LCDWRITE (cont.)

Important hints on lcds:

Experience over serveral years shows a wide range of tolerances of lcds specifications. If you get trouble with lcds try following actions:

Problem 1:
lcd cannot be initialized or nothing can be written on.
Solution:
Insert the LCDDELAY instruction. If already done increase the value. Shorten the cable between lcd and microcontroller.

Problem 2:
After running well, suddenly the lcd shows wrong characters. Right and wrong characters can be mixed. Sometime the text is scrolling.
Solution:
First try to solve this problem with the LCDDELAY instruction. If this fails pay attention not to refresh to often. A refresh time of 0.5 to 1 second is enough and makes less trouble.

Problem 3:
There are lcds sold as 1 line by 16 characters. But they are assembled very cheap and a multiplexer ic is left over. Writing to such an lcd the first 8 characters are right but the second 8 characters are lost.
Solution:
They are not really lost but they are invisible because the appear on line 2 which is not assembled. Divide the output into 2 parts and address the second part to line 2. This lcd cannot be used if a number (3 or 5 bytes) should displayd around this border.

LET

syntax: LET *var* = *value* [*operator value*...]

function: assigns a constant or a result of an operation to the variable *var*

var	(8 or 16 bit) Variable with the result of an assignment respectively arithmetic operation.
value	Operand
operator	Operation (+, -, *, ...see below)

description: To the left of the equal sign must be a variable. On the right there may be a constant, another variable or an equation containing variables and constants. The keyword LET is **obligatory** and cannot be left off. The value which shall be assigned to a variable *var* can be the result of a complex mathematical and/or logical calculation.

Following operators are implemented:

Addition	+	a + b
Subtraction	-	a - b
Multiplication	*	a * b
Division	/	a / b
Modulo	mod	a mod b (modulo, division remainder)
logical AND	and	a and b
logical OR	or	a or b
logical EXOR	xor	a xor b

example:

```
LET length = 5                  REM var length = 5
LET width = 3                   REM var width = 3
LET area = length * width       REM var area = 15
LET var_a = 3 + 5 / 2           REM value var_a  = 4
LET field(index) = 3,5,var_a    REM up to 10 items
```

remarks: No parenthesis are allowed so the compiler calculates the expression from the left to the right side without any regards to the mathematical conventions.

LOCATE

syntax: LOCATE *y,x*

function: Moves cursor to position y,x on LCD.

*y,x*defines column and line for the next writing on lcd

decription: Moves cursor to column x in line y. X and y must be equal or less of the values defined in LCDINIT. Turn cursor on or off with the commands CURSON and CURSOFF.

example:

```
            LCDINIT   RC,4,20            REM LCD initialization
START:
            LCDWRITE 1,1,"Ing.Buero LEHMANN"                    REM upper left corner
            LCDWRITE 2,1,"Fuerstenbergstr. 8a"   REM second line
            LOCATE 3,1                                  REM move cursor to line 3
            LCDWRITE 0,0,"Phone ++49 (0)7831  452"   REM write at cursor position
            FOR var_a=0 to 255
                LET var_t=var_a * var_a          REM calculate square
                LCDWRITE 4,1,var_a              REM output a as decimal value
                LCDWRITE 4,15,var_t              REM display square
            NEXT var_a
            GOTO start                                  REM jump to START
MORE:
                ....
```

LOFREQ

syntax: LOFREQ *port,pin,frq,duration*

function: generates a frequency at *port,pin* with the *duration*

port	i/o port, can be a variable or constant
pin	i/o pin, must be a constant
frq	frequency in Hz, must be a constant
duration	time in ms, how long the sound is generated, must be a constant

description: The LOFREQ command generates a sound at an io pin. The frequency range is from 1Hz to 2000Hz. For higher frequencies use the SOUND command.

example:

```
START:
        LOFREQ ra.0,100,20   REM generates a tone for about 20ms
                             REM with the frequency of 100Hz
                             REM at port Ra.0
        GOTO start
```

remarks: The command LOFRQ is optimized to 100 Hz at a xtal frequency of 4Mhz. The preferred range is from 1 Hz - 2000 Hz. At 100Hz the error is less then 1%. At 1000Hz the error is about 4,5% (957Hz) and at 10000Hz the error is about 49% (6680Hz).

LOOKDN

syntax: LOOKDN *var, target value, value 1, value 2,..., value n*

function: looks for a value within a list and if found, it returns the value's position within the list

var (8 bit) variable where the return value (position) is stored, if found must be a variable

target value (8 bit) value to look for, can be a variable or constant
value n... (8 bit) liste of values to compare with target value can be variables or constants.

description: The command LOOKDN compares the *target value* with the values *value 1* , *value 2* etc and if a match occurs it returns the position within the list. The contents of *var* isn't changed if there is no match.

example:

```
        LET var_b = 0
START:
        SERIN ra.0,9600,var_a          REM reads a byte via SERIN
        LOOKDN var_b,var_a,65,88,93    REM if var_a =65 then var_b = 0
                                       REM if var_a =88 then var_b = 1
                                       REM if var_a =93 then var_b = 2
                                       REM if var_a <> 65 and 88 and 93 then var_b = unchanged
```

remarks: If the list consists only of constants the compiler gererates a memory saving code using the instruction **RETLW xx** . But if there is only one variable within that list another code is generated. To use the RETLW instructions, it is necessary that this code is at the beginning of a memory page because of it's 8 bit offset calculation. This is done automatically by the compiler but all items of LOOKDN and LOOKUP may not exceed 100 . If the list exceeds the first 255 program memory words, the compiler doesn't generates a warning. In doubt, do force the compiler to produce no code with retlw instruction by adding a dummy variable at the end of the list. Only 8 bit variables are allowed.

Automatically the compiler sets the generated code close to the beginning. But anyway the total number of items (LOOKUP and LOOKDOWN) isn't allowed to be more than about 100.

LOOKUP

syntax: LOOKUP *var,position, value 1, value 2,.., value n*

function: returns the value out of a list at a certain position

var	(8 bit) variable to store the value picked up at *position*
position	(8 bit) position of the value to read , can be a variable or constant
value n	(8 bit) a list of values, can be variables or constants

description: The command LOOKUP picks up the value at a certain *position* out of the list. If there are enough items within the list, the found value is stored in *var* . If the list is shorter and the value of *position* tries to access beyond the bounds the contents of *var* is not changed.

example:

```
START:
        FOR var_a = 0 to 25
        LOOKUP var_b,var_a,65,66,67, ...    REM converts the offset ( 0-25 )
                                            REM to a1 ascii characters ( A-Z )
        NEXT var_a
```

remarks: If the list only consists of constants the compiler gererates a memory saving code using the instruction **RETLW xx**. But if there is only one variable within that list another code is generated. To use the RETLW instructions, it is necessary that this code is at the beginning of a memory page because of it's 8 bit offset calculation. This is done automatically by the compiler but all items of LOOKDN and LOOKUP may not exceed 100 . If the list exceeds the first 255 program memory words the compiler doesn't generates a warning. In doubt, force the compiler not to produce the code with retlw instruction by adding a dummy variable at the end of the list. Only 8 bit variables are allowed.

LOW

syntax: LOW *port,pin*

function: the *port,pin* becomes an output. The output signal is low (0).

port i/o port, can be a constant or variable (8 bit)
pin i/o pin, can be a constant or variable (8 bit)

description: The command LOW changes the according entry in the tris register to set the pin to output. Afterwards a low signal is outputted. The output mode keeps on. That means the TRIS-register will be changed.

example:

```
            LET var_a=1                     REM variable var_a = pin 1
START:      INP ra,var_a                    REM pin 1 is input
            IF var_a.0=1 THEN GOTO MORE
            LOW ra,var_a                    REM pin 1 is now output and LOW
MORE:
             ...
```

remarks: Do not use the LOW command to change variables, in this case use RES. LOW is a bit command.

ON GOSUB

syntax: ON *var* GOSUB *addr0,addr1, addr2, ...,addr n*

function: branches to a subroutine whose address is determine by the offset in *var*

var (8 bit) variable
addr0, addr1 addresse of subroutines (labels), can be variables or constants.

description: The program branches to the subroutine whose target address (label) is at the position stored in *var*. Is var = 2, subroutine at adr2 is called. Points var to a no existing target, ON-GOSUB is ignored and program execution is continued at the command that follows.

example:

```
START:
        SERIN ra.0,9600,var_a                REM reads seriell port, a value
        ON var_a GOSUB label6,label7,label8  REM if var_a = 0, go to subr. label6
                                             REM if var_a = 1, go to subr. label7
                                             REM if var_a = 2, go to subr. label8

        GOTO start

LABEL6:  . . .                 REM subroutine 1
         RETURN

LABEL7:  . . .                 REM subroutine 2
         RETURN

LABEL8:  . . .                 REM subroutine 3
         RETURN
```

remarks: Nested subroutines are not available for PIC with 12 bit cores. All other types allow up to 4 nested subroutines.

ON GOTO

syntax: ON *var* GOTO *addr0,addr1,...,addr n*

function: branches to address label determine by the contents of *var*

var	(8 bit) variable
addr0, addr1	address label, can be variable or constant

description: The program branches to the routine whose target address (label) is at the position stored in *var* . Is var = 2, routine at label addr2 is called. Points *var* to a no existing target, ON-GOTO is ignored and program execution is continued at the command that follows.

example:

```
START:
            SERIN ra.0,4800,var_a              REM reads variable a via rs232
            ON var_a GOTO label6,label7,label8  REM if var_a = 0, go to label6
                                               REM if var_a = 1, go to label7
                                               REM if var_a = 2, go to label8
            GOTO start
LABEL6:     . . .                              REM entry point for var = 0
LABEL7:     . . .                              REM entry point for var = 1
 LABEL8:     . . .                              REM entry point for var = 2
```

OUTP

syntax: OUTP *port,pin*

function: configures pin as output

port i/o port (variable (8 bits) or constant)
pin i/o pin (variable (8 bits) or constant)

description: The OUTP command changes the tris register so that the corresponding pin becomes output.

remarks: Since the last write command to a port is latched this information appears at the pin immediately after changing the tris register. This is importand if you want to access to a port used as a bidirectional bus, when switching from reading to writing.
OUTP is a bit command.

OUTPUT

syntax: OUTPUT *port,value*

function: puts out 8 bits (parallel) onto a port

port i/o port (8 bit), variable or constant
value (8 bit) variable or constant

description: The contents of a variable or a absolute term are put out onto a port. The port access is 8 bits parallel. The tris register is not changed.

example:

```
START:
        TRIS RB,0
        LET b1=$AA          REM all even bits are set
        OUTPUT rb,b1
        LET b1=$55
        OUTPUT rb,b1        REM all odd bits are set
        GOTO start
```

remarks: OUTPUT is a byte command.

PEEK

syntax: PEEK *var,address*

function: transfers the contents of a file register into a variable

var (8 bit) variable
address (8 bit) absolut address of the file register

description: The contents of any file register (not only those which are used for user variables), addressed by *address* , is transfered to the variable *var*

example:

```
START:
        LET var_b=1
        LET var_a=$1F
        PEEK var_b,var_a        REM the contents of rtcc is loaded in $1F
        GOTO start
```

remarks: PEEK is a byte command.
It was important in compiler versions 4 and older because the variables were predefined. Now just define a symbol to a register.

Attention:
Better use LET
e.g.

```
LET TMR0=5
LET var_a=rtcc
```

POKE

syntax: POKE *address, value*

function: writes value to any file register

address	address of the file register, variable (8 bit) or constant
value	variable (8 bit) or constant

description: The contents of the variable or the constant is transfered to any fileregister (not only those which are used for user variables).

example:

```
START:
        POKE $1F, "A"
        LET var_a=$1F
        PEEK var_b,var_a        REM "A" is transfered into var_b via file register $1F
       GOTO start
```

remarks: POKE is a byte command.
It was important in compiler versions 4 and older because the variables were predefined. Now just define a symbol to a register.

Attention:
see PEEK

PRINT

(only 16F62x, 16F81x, 16F87x and 16F88)

syntax: PRINT *port,pin,baud,"text",var*

function: sends text or variables seriel to the pc for debugging

port	output port
pin	output pin
baud	baud rate
"text"	sends the text enclosed by quotes
var	(8/16 bit) sends the contents of the variable *var*

decsription: The PRINT command is very useful for debugging. Text, varaible etc can be sent to an ascii monitor on the PC. Like the LCDWRITE command, you can output a hexadecimal value while appending a '$', '%' for binary format and '#' for the ascii equivilant.

example:

```
PRINT rb,0,4800,var              REM sends variable
PRINT rb,0,4800,"text",var,$     REM (%=binär, $=hex, .....)
```

remarks: For conversation the PRINT command needs the memory area form 68h to 6Ch on the first page (for conversion into dezimal you need memory area from 68H to 6CH, binary from 69H to 6CH and hex 6CH). The output is always 32-bits, therefore 32-bit library will be linked. Leeding 0 will be replaced by blanks.

PULSIN

ATTENTION !!! Command was changed. To get the old functionality of PULSIN use the keyword PULS_IN.

syntax: PULSIN *port,pin,slope,var*

function: measures the length of a puls at a certain pin

port	(8 bits) i/o port, variable or constant
pin	(8 bits) i/o pin, variable or constant
slope	(1, 0) trigger slope, which starts the measuring (1 = LO-HI; 0 = HI-LO)
var	(8/16 bit) variable to store the result

description: The command PULSIN measures the length of a puls with a solution depending on the xtal frequency. There is no conversation to a timebase e.g. us. The result is a relative value. The measuring is triggered with a selecteable slope and finished by the reverse slope. If *slope* is zero, a falling edge triggers the measuring, one (1) triggers with a rising edge. If slope is a variable only the LSB (bit 0) is relevant.

example:

```
REM             messures the frequency at pin RA0

                TRIS rb = 255               REM TRIS-register at input
START:          PULSIN ra,0,1,var_z         REM read an impulse at ra.0.
                .....
                GOTO start
```

remarks: The corresponding *port,pin* first must be set as input. The result may be stored as a 8 or 16 bit variable.

PULS_IN

ATTENTION!!! Is equal to the old version of PULSIN (before VERSION 4.1-00)

syntax: PULS_IN *port,pin,slope,var*

function: measures the length of a puls at a certain pin

port (8 bits) i/o port, variable or constant
pin (8 bits) i/o pin, variable or contant
slope (1, 0) trigger slope, which starts the measuring (1 = LO-HI; 0 = HI-LO)
var (8/16 bit) variable to store the result

description: The command PULS_IN measures the length of a puls with a solution depending on the xtal frequency. The result is calculated to a timebase in us. The result is an absolute time value. The measuring is triggered with a selecteable slope and finished by the complement slope. If *slope* is zero, a falling edge triggers the measuring, one (1) triggers with a rising edge. If slope is a variable only the LSB (bit 0) is relevant.

example:

```
REM     messures the frequency at pin RA0 (set 50 Hz digital signal source) and puts out
REM     the errors at RB. 50 Hz are a low-impulse of 10 ms.

            TRIS rb = 0                                 REM TRIS-register at input
START:
            PULS_IN ra,0,1,var_z                        REM reads an impulse at ra.0 (16 bit)
            LET var_z = var_z+5                         REM round up
            LET var_z = var_z/1000
            LET var_a = 0                               REM result weighted
            IF var_z <  93 THEN LET var_a = 8
            IF var_z > 108 THEN LET var_a = 8
            IF var_a >    0 THEN GOTO more
            IF var_z <  96 THEN LET var_a = 4
            IF var_z > 104 THEN LET var_a = 4
            IF var_a >    0 THEN GOTO more
            IF var_z = 100 THEN LET var_a = 1 ELSE var_a = 2
MORE:           OUTPUT rb,var_a                         REM put out result at rb
            GOTO start
```

remarks: The corresponding *port,pin* must be set as input first. The result may be a 8 or 16 bit variable. The max. width of the puls that must be measured, depends on the xtal frequency. A higher xtal means smaller pulses may be measured. Using a 8-bit-variable you just have a pulse duration upto 255 us, with a 16-bit-variable you have 0,65535 s. Using a 8-bit-variable and an overflow of the measured value this variable will get the LOW-byte of the internal 16-bit masured value. Outrunning the max value of 0,65535 s the reuslt will be 0. The command is optimized at a xtal of 4 Mhz.

The input port with the command PULSIN is shortend by factor 10 as the corresponding output impulse with the command PULSOUT.

PULSOUT

syntax:

PULSOUT *port,pin,duration,H*
PULSOUT *port,pin,duration,L*
PULSOUT *port,pin,duration*

function: generates a square puls for a certain time

port	i/o port, variable (8 bit) or constant
pin	i/o pin, variable (8 bit) or constant
duration	(16 bit) duration of the puls in units of 10 us, variable or constant
H	generates a high puls (1)
L	generates a low puls (0)

(Absence of H or L will generate a puls which is inverted of the acutal output level)

description: PULSOUT generates a puls with a resolution of 10us. The duration is from 10us up to 0,65535s. The contents of the tris register is not changed. The pin has to be defined for output otherwise no puls is put out.

example:

```
REM  no retriggerable mono stable flip flop with a puls of 100ms.
REM  the duratoin of the puls is specified with a resolution of 10 us.
REM  at times shorter than 210 us (at 4 Mhz) truncation errors are strongly noticeable
START:
        HIGH rb.0
LABEL1:
        IF ra.0 = 0 THEN GOTO label1      REM wait until ra.0 = 1
        PULSOUT rb,0,10000,H              REM generate a puls of 100ms
                                          REM not retriggerable
WAIT:
        IF ra.0 = 1 THEN GOTO wait
        GOTO label1
```

remarks: The puls generated by PULSOUT is 10 times longer than the result of PULS_IN . The max. width of the output puls depends on the xtal frequency.

PWM

syntax: PWM *port,pin,level,duration*

function: generates a puls-width-modulated signal for a certain time.

port	i/o port, variable (8 bit) or constant
pin	i/o pin, variable (8 bit) or constant
level	duty cycle that defines the output voltage, variable (8 bit) or constant (1-255).
duration	duration of the output signal, variable (8 bit) or constant

description: The PWM command generates a certain amount of pwm signals which are square pulses with various duty cycles. The corresponding pin is set to output at the beginning of the routine and set to input at the end. This is done automatically to put this pin on high impedance (TRI-State).

example:

REM generates a pwm signal that is filtered by a RC combination. This is the way to generate a analogue voltage.
REM (128 / 255) * 5V. Everytime 20 cycles are generated to charge/discharge the capacitor.

```
START:
        SERIN ra.0,4800,var_a       REM read Byte seriel ( e.g 128)

        PWM rb,0,var_a,20           REM Output of analog tension corresponding
                                    REM to the receiving BYTE
                                    REM of the capacitor  of the RC-link
                                    REM at rb,0 load up to 2,5V
                                    REM (128 / 255) *5V.  20 charging cycles will be generated.
GOTO START
```

RANDOM

syntax: RANDOM *var*

function: generates a pseudo random number

var (8/16 bit) variable, contains the generated value

description: The RANDOM command generates a pseudo random number which is stored in *var* . The initial value is the old contents of *var* . The new value is calculated by a special algorithem using the value of the program counter. If the rtcc register is in use, its value is also used for the new random number. *var* can be a 8-bit- or 16-bit-variable.

example:

```
START:
        RANDOM var_s                    REM generates a 16 bit random
        IF var_s < 2000 THEN GOTO less
        IF var_s < 10000 THEN GOTO more
        GOTO start
LESS:
        LOFREQ ra.0,var_s,20            REM generates a sound
        GOTO start
MORE:
        SOUND ra.0,var_s,20             REM generates a sound
        GOTO start
```

remark:

It depends mainly on the rtcc (tmr0) register how randomly the numbers will be generated. If CLOCK is actived the generated random number is ok. Otherwise you should initialize the option register if possible. Reset the t0cs-bit. The result is a increment of rtcc at each instruction cycle. When you use the 12C5xx the compiler switch TOCS_INT can be set.

RCTIME

syntax: RCTIME *port, pin, level, var*

function: measures the charging or discharging time of a capacitor.

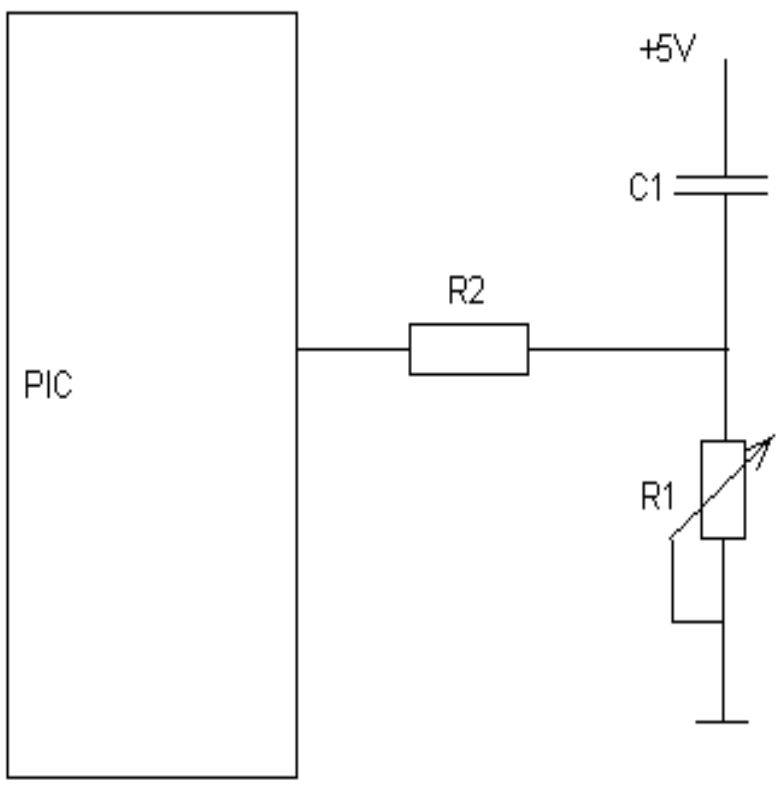
port, pin	(8 bit) pin that is connected to the RC circuit.
level	The measuring keeps on as long as this *level* is at pin.
var	(8 Bit) variable for the result

description: This command measures the charging or discharging time of a capacitor. Because the lower threshold is smaller than the upper it is recommended to design the hardware as shown in the schematic. The 10 ohms resistor is for protection in case of discharging. There is no time reference, the result depends on the xtal frequency. The experience has shown that a reference to real time is not necessary. In case you need it (e.g. to display the value on a lcd), you may calculate it by your own (just a few BASIC lines).

example:

```
START:
        HIGH ra.0               REM charge the capacitor
        WAIT 1                  REM time for charging
        RCTIME ra.0,1,var_a     REM measure the time, until the level drops to zero.
```

RCTIME (cont.)



fq = 4MHz	fq = 2MHz
C1 = 100nF... 200nF	C1 = 100nF
R1 = 10k	R1 = 22k
R2 = 10 Ohm	R2 = 300 Ohm

The 10 ohms resistor avoids shortening in the discarge cycle. The result is relative to the xtal frequency. In most cases no absolute time measuring is needed. On the other hand it needs only a few program lines to do this in basic.

READDATA

syntax: READDATA *var1 (, var2, var3,...)*

function: Reads the next item out of the data list. This data field can have 2048 entries (depending on memory capacity).

var1 8-bit variable or 8-bit-constant

description: Very often a large amount of constant values are necessary. In former compiler releases you had to use LOOKUP and LOOKDWN statements. But these can handle only up to about 100 items. DATA break through this barrier. Read an item with READDATA increments an internal pointer which controls the access. To reset or set this pointer to a specified line is the only pointer modification you need.

example:

```
START:
        RESTORE                     REM set read pointer points to first item
        READDATA var__a             REM read first item (here 65)
        RESTORE label_1             REM set read pointer -> pointer points to the 5th item
        READDATA var_a              REM read 5. item, herer 12
        READDATA var_b,var_c,var_d  REM read next three items

        GOTO START
        DATA 65,66,67,"F"
LABEL_1:
        DATA 12,45,32,17,25
```

remarks: see also DATA , RESTORE

(!! not for 12C5xx and 16C5x !!)

You can also manipulate the data pointer like:

LET datptr_=datptr_ + offset

READ

syntax: READ *addr,var*

function: reads a memory cell at address *addr* out of the internal EEPROM and stores the value in *var*

addr (8 bit) address to read, variable or constant.
var (8 bit) variable with the result

description: The READ command reads one of the additional memory cells (eeprom) in a device like PIC16C83, PIC16C84, PIC16F83, PIC16F84, 16F87 and put the result into *var*

example:

```
            DEFINE ADR=$30          REM ADR to location 30H
            DEFINE VALUE=$31        REM VALUE
            DEFINE I1 = $31
START:
            FOR I1=0 TO 63          REM only 64 bytes
            LET ADR=I1              REM writes a value in each
            LET VALUE=I1*2          REM memory cell
            WRITE ADR,VALUE
            NEXT I1
            LET ADR=2               REM now read the third entry
            READ ADR,VALUE          REM into VALUE (must be 4)
```

remarks:
(only for PICs with internal eeprom data memory)

REM

syntax: REM or ' (single quotation mark)

function: the contents of the line after REM or ' will be ignored by the compiler.

description: Useful for comments or debugging.

example:

```
LET Length=5          REM your comment for this line
```

RES

syntax: RES

function: set the corresponding bit of a variable or a port to zero.

description: In contrast to the LOW command the RES command do not affect the tris register. Therefore this command can access to regular variables and is faster then LOW.

example:

```
START:
        RES var_a,0       REM bit 0 in variable var_a is cleared
```

RESTORE

(not for 12C5x and 16C5x)

syntax: RESTORE {label

function: The internal data pointer is reset and points to the first item or is set to any label and points to an item within the data list.

Description: Very often a large amount of constant values are necessary. In former compiler releases you had to use LOOKUP and LOOKDWN statements. But these can handle only up to about 100 items. DATA break through this barrier. Read an item with READDATA increments an internal pointer which controls the access. To reset or set this pointer to a specified line is the only pointer modification you need.

example:

```
START:
        RESTORE                     REM reads pointer, points to first item
        READDATA var_a               REM reads first item (here 65)
        RESTORE label_1              REM pointer points to the 5th item
        READDATA var_a               REM read 5. item, here  12
        READDATA var_b,var_c, var_d   REM read next three items

        GOTO START
        DATA  65,66,67,"F"
LABEL_1:
        DATA 12,45,32,17,25
```

remarks: see also DATA , READDATA

RETURN

syntax: RETURN

function: Terminate a subroutine and returns to the routine superior.

description: Every subroutine must be terminate by a RETURN command so that the program can return to the command which follows the GOSUB.

example:

```
START:
        FOR var_a = 0 TO 5
        GOSUB reading       REM check an external signal
        NEXT var_a
        END

READING:
        IF RA.0 = 1 THEN GOTO rememb
        PULSOUT RA.1,5                  REM no signal, generates a puls
        RETURN

REMEMB:
        LET var_b = 1                   REM remembers the signal with the help of variable var_b
        RETURN
```

remarks:

REVERS

syntax: REVERS *port,pin*

function: defines this pin as output and toggle the actual output signal

port i/o port, variable (8) or constant
pin i/o pin, variable (8) or constant

description: The command REVERS inverts the output signal of an i/o pin. Is this pin configuered as an input, the tris register is changed so that this pin becomes an output.

example:

```
START:
        LET var_a=6             REM address of port B
        OUTP ra.0               REM pin 0 is output
        OUTPUT var_a, 0         REM pin 0 is output, give out zero
        REVERS var_a,0          REM set RB,0 to one
        REVERS var_a,0          REM reset RB,0 to zero
```

remarks: REVERS should not be used for regular variables, only for port pins. To invert a bit within a variable use the TOGGLE command.

SERIN

syntax:

SERIN *port,pin,baud,var 1,.var 2. ..., var n*
SERIN *port,pin,baud,timeout=x,var 1,.var 2. ..., var n*
SERIN *port,pin,baud,parity,var 1,.var 2. ..., var n*
SERIN *port,pin,baud,parity,timeout=x,var 1,.var 2. ..., var n*
SERIN *port,pin,baud,int,var 1,.var 2. ..., var n*

function: reads data serially from an i/o pin

port	i/o input pin, (8 bit) variable or constant
pin	i/o input pin, (8 Bit) variable or constant
baud	(300 - 9600) baud rate, variable or constant
parity	EVEN, EVEN7, ODD oder ODD7
timeout	(8/16 bit) defines how many cycles the program will wait for the start bit if timeout occurs, bit 3 of the internal ERR byte is set
int	enables the RCIE bit for generating an interrupt
var	(8 bit) variable, containing the received data byte

description: This command reads at *port, pin* any data byte with the defined baud rate. The argument *timeout* can only be used when uart is not used. If *timeout* is not used, the program remains in a loop until the start bit is detected. If timeout is defined the waiting loop is execute n-times. If no start bit is detected bit 3 in ERR is set. Timeout can be a variable or constant.
The time to execute one loop in the waiting loop is about 10 * 4/fq; therefore it is a relative time, depending on the xtal frequency.
Int can only be used in interrupt driven input routines. If you want to read a serial data stream within a interrupt service routine, this routine should be entered when the uart has received a data byte. To enabling interrupt generation *int* must be active.

example:

```
START:
        LET baud=329            REM at 6,144 MHz for 4800 baud
        LET baud=407            REM at 6,144 MHz for 2400 baud
        LET pin=2
        LET var="A"
        SEROUT rb,pin,baud,var
```

remarks:
You only can change the baud rate and the parity of the parameters of the serial transmission. At the SERIN software routine the patity-bits 8E1 and 8O1 will be ignored.

8 data-bits, no parity, 1 stopp bit (8N1).
8 data-bits, eveb parity, 1 stopp bit (8E1).
8 data-bits, odd parity, 1 stopp bit (8O1).
7 data-bits, even parity, 1 stopp bit (7E1).
7 data-bits, odd parity, 1 stopp bit (7O1).

You can chose a bit rate up to 4800 baud at a xtal frequency of 4 Mhz, at higher frequencies corresponding more, at lower corresponding less.

SERIN (cont.)

see: SETBAUD

attention !!!

What do you have to do when the parameter BAUD is a variable?

Having a SERIN- and SEROUT-command, the parameters can be variables. PIN have to be a 8-bit-variable, BAUD a 16-bit-variable. The parameter BAUD will be randomized in a delay-time which depends on the PIC-kernel, xtal frequency and the baud rate.! It must be calculated before because it can't be done during the runtime. This value must be given to the variable instead of the absolute baud rate. The compiler does it itself when the baud rate is given as a constant. The supplied program BAUDCALC.EXE will do this for you.

Interrupt operation:

If you want to replace the interrupt routine with the help of the SERIN-command, of course you have to set the corresponding interrupt-enable-bits, all ahead GIE and RCIE. At some PICs you have to set the PEIE-bit, because here the UART-interrupts are merged with other interrupts to the "peripheral interrupt". See argument *int*

If the SERIN command is not within the interrupt service routine (ISR) don't use argument *int*

remarks: The data format for SERIN and SEROUT is fixed except the baud rate.

8 data bits, no parity, 1 stopp bit (8N1).

At a xtal frequency of 4 MHz baud rates up to 9600 bps are realizable. If the frequency is less, only a lower baud rate may be reached.

SEROUT

syntax: SEROUT *port,pin,baud,var 1, var 2,...,var n*
SEROUT port,pin,baud,parity,var 1, var 2,... var_n

function: sends data bytes serially out of a pin

port	(8 bit) io output port, variable or constant
pin	(8 bit) io output pin, variable or constant
baud	(300 - 9600) sends the data bytes with the defined baud rate, variable or constant
parity	EVEN, EVEN7, ODD oder ODD7
var	(8-Bit) variable or constant to be sent.

description: The variables or constants are sent serially via *port, pin* with the defined baud rate.

example:

```
START:
        SERIN ra.0,4800,var_b          REM reads a byte with 4800 bps
        LET VAR_b = var_b-1
        SEROUT rb.0,2400,var_b         REM sends variable B with 2400 bps
```

Remarks: The data format for SERIN and SEROUT is fixed except the baud rate.

8 data bits, no parity, 1 stopp bit (8N1).
8 Datenbits, even parity, 1 stopp bit (8E1).
8 Datenbits, odd parity, 1 stopp bit (8O1).
7 Datenbits, even parity, 1 stopp bit (7E1).
7 Datenbits, odd parity, 1 stopp bit (7O1).

Baud rates up to 9600 bps are realizable at a xtal frequency of 4 MHz. If the frequency is less, only a lower baud rate may be reached.

see: SERBAUD

attention !!!

What do you have to do when the parameter BAUD is a variable?

Having a SERIN- and SEROUT-command, the parameters can be variables. PIN have to be a 8-bit-variable, BAUD a 16-bit-variable. The parameter BAUD will be randomized in a delay-time which depends on the PIC-kernel, xtal frequency and the baud rate.! It must be calculated before because it can't be done during the runtime. This value must be given to the variable instead of the absolute baud rate. The compiler does it itself when the baud rate is given as a constant. The supplied program BAUDCALC.EXE will do this for you.

example:

```
        LET baud=329            REM at 6,144 MHz for 4800 baud
        LET baud=407            REM at 6,144 MHz for 2400 baud
        LET pin=2
        LET var="A"
        SEROUT rb,pin,baud,var
```

SET

syntax:

SET *var,bit*

or

SET *port,pin*

function: sets the corresponding bit of a variable or port pin to one.

var, port, pin must be byte variables or byte constants

description: In contrast to the HIGH command SET will not change the tris register. Therefore it is useable within regular variables.

example:

```
START:
        SET var_A,0     REM bit 0 of variable A is set to one
```

SETBAUD

syntax: SETBAUD *value*

function: Changes baud rate.

baud (300 - 9600) fixes the baud rate (Konstante)

description: This command is only necessary if the SERIN command is interrupt driven by the internal UART. This allows to use different baud rate on the same seriell interface, e.g. at first a slower baud rate is selected and later a higher baud rate is used. In a interrupt driven SERIN command the baud rate must be changed before the next byte is received.

example:

```
START:
        SETBAUD 9600
```

remarks: only at UART-operation !!!

SLEEP

syntax: SLEEP *duration*

function: The cpu enters the SLEEP mode up to several seconds.

duration (8/16-Bit) duration of the sleep mode in seconds, variable or constant

description: The cpu enters the sleep mode for a specified time. During the sleep mode the power consumption is very low. All functions are turned off except the watchdog timer. This timer interval is set to 2,3s. After this period the cpu wakes up and checks if the total time is passed. If not, the cpu enters the sleep mode again. If the total time is over, program execution is continued at the command which follows the SLEEP command. If you are using with command along with the PIC12C50x or PIC16C5x a short puls occurs at each output pin because the cpu is woken up by a reset that changes all tris registers to one's.

example:

```
START:
        SLEEP 1565      REM sleeps for about one hour
                        REM 3600 / 2.3 = 1565

              .....
        GOTO xyz        REM program continues here after an hour
```

remarks: *duration* is a value from 1 to 65535, therefore the sleeping time is 18h maximum with a resolution of about 2,3 s.

SLEEP uses the watchdog-timer (must be active) and the prescaler. If CLOCK is active at the same time the prescaler will be changed into the watchdog-timer.

Important hint:

All PICs with 12 bit core (PIC 12C5x and 16C5x) the SLEEP instruction should be placed in the first part of the program (exactly within the first 256 assembler instructions).

SOUND

syntax: SOUND *port,pin,frq,duration*

function: generates a frequency for the *duration* at *port,pin*

port	(8 bit) i/o port, variable or constant
pin	(8 bit) i/o pin, variable or constant
frq	(8/16-Bit) frequency in Hertz, must be a constant
duration	(8/16-Bit) duration of sound in milli seconds, must be a constant

description: The SOUND command generates tones a the specified i/o pin

example:

```
START:

        SOUND ra.0,10000,20    REM generates a tone for 20 ms with the frequency of 10kHz at pin RA.0

        GOTO START
```

remarks: The SOUND command is optimized to 5kHz at 4Mhz xtal frequency. Applications with frequencies form 1000Hz to 10kHz. The error is about 6% (1kHz), 7% (10kHz)and less then 1% at 5kHz.

See also LOFREQ

SWAP

syntax: SWAP *var*

function: exchanges the upper and lower half byte (nibble) of the 8 bit variable *var*

description: SWAP exchanges the upper and lower half byte of a 8 bit variable. This is useful for bcd arithmetics or bit handling.

example:

```
START:
        LET var_a=$6
        SWAP var_a      REM after SWAP: A1=$60
```

TOGGLE

syntax: TOGGLE *port,pin*

function: inverts the output signal at the specified pin

port i/o port, variable (8 bit) or constant
pin i/o pin, variable (8 bit) or constant

description: The TOGGLE inverts the actual signal at output pin. The tris register is not changed.

example:

```
START:
        LET var_a=6         REM address of port RB
        OUTP ra.0           REM pin 0 is output
        OUTPUT ra, 0        REM output level is low
        TOGGLE a1,0         REM output level is high
        TOGGLE var_a        REM output level is low again
```

remarks: With the TOGGLE command you can also invert bits in a regular variable because the tris register is not affected.

TRIS

syntax: TRIS *port,value*

function: writes *value* into the tris register of the specified *port* (8 bit)

port i/o port, variable (8 bit) or constant
value direction of the signal path (1 becomes input; 0 becomes output), can be variable (8 bit) or constant

description: The tris register which defines the mode of the i/o *port* (RA, RB, RC, RD, RE, GPIO) is loaded with the value. Within value every bit is corresponding with an i/o pin e.g. bit 0 belongs to Rx0. If the bit is set to 1, the pin becomes an input while set to 0 the pin becomes output.

example:

```
LET var_a=%00001111      REM bit RA0 to RA3 becomes input
                         REM bit RA4 to RA7 becomes output
LET var_b=5
TRIS var_b,var_a
```

remarks: TRIS is a byte command.

TXDDELAY

syntax: TXDDELAY *const*

function: Pause a short time after each transmitted byte.

const 8 bit constant (default = 125)

describtion: In case of a slow data receiver this command slows down the byte transfer rate, not the baud rate which is the bit transfer rate. A simple software delay routine is executed, while const is the repeatition value. At 4 MHz and 9600 baud a value of 125 delays for about 500us which is good for the most applications.

example:

```
        TXDDELAY 255              REM max. delay time
START:
        SEROUT RC,6,9600,temp     REM send TEMP twice with 1ms between each byte.
```

remarks: TXDDELAY is necessary only if hardware usart is used. If a software routine is implemented or forced (define serout=soft) it is slow enough.

WAIT

syntax: WAIT *msec*

function: delays program execution for a certain time

msec (8/16-Bit) time to wait in ms, variable or constant

description: WAIT delays the program for a certain time without entering the sleep mode. Therefore no reduction of power consumption arrives. The accuracy of delay time depends on the kind of oscillator type (crystal, ceramic or rc).

example:

```
START:
        WAIT 1000    REM waits for about 1 s without power saving
        . . .
       GOTO xyz     REM after 1 sec program execution
REM continues hier
```

remarks: The range of the delay time starts at 1ms and ends at 65,535s (the final value depends on the xtal frequency and can be less then 65 sec because the 16 bit value is too small for longer times at higher frequencies.

WRITE

(o n l y 1 2 E 5 1 x , 1 2 E 6 7 x , 1 6 E 6 2 x , 1 6 X8 x , P I C1 6 F 8 7 x)

syntax: WRITE *addr,value*

function: writes a value into the additional eeprom

addr (8-Bit) address where value is stored, variable or constant
value (8-Bit) variable or constant, which is stored

description: WRITE writes the contents of the variable *var* or the constant at address *address* into the eeprom. The size of eeprom is upto 256 byte. Because the writing of a byte needs about 20 ms the program checks the corresponding flag wether the writing is still going on or not. This check is done before the byte is written. In doing so program execution continues after starting the write sequence. If you write or read bytes within interval greater then 20 ms, no additional delays are caused by this command.

example

```
                DEFINE ADR=$30          REM ADR to location 30H
                DEFINE VALUE=$31        REM VALUE
                DEFINE I1 = $31
START:
                FOR I1=0 TO 63          REM only 64 bytes
                LET ADR=I1              REM writes a value in each
                LET VALUE=I1*2          REM memory cell
                WRITE ADR,VALUE
                NEXT I1
                LET ADR=2               REM now read the third entry
                READ ADR,VALUE          REM into VALUE (must be 4)
```

remarks: using 12E51x, 12E67x, 16E62x, ... it must be garenteed that the time between two WRITE commands is at least 10msec because they have the internal EEPROM as an independant I2C Die. Therefore the runtime-library can't check the end of the writing cycle.

ASSEMBLER (general)

Indroduction

The assembler iL_ASS16 translates the SRC file which is generated by the compiler iL_BAS16 into the OBJ file. Most of the syntax is compatible to Microchips notation. Some special instructions outputed by iL_BAS16 will be translated into a sequence of assembler instructions.
iL_ASS16 is also useable as a standalone application.

Invoking the assembler iL_ASS16

Usually the assembler will be invoked by the assembler topic in the iL_EDy enviroment. When pressing the compiler button the assembler will start automatically if no error is found by the compiler. Invoking iL_ASS16 out of the explorer an i/o-error is the result because of a missing parameter (filename without extension).

The extention SRC is inserted by the assembler. The assemblers outputs a file with the extension OBJ. The default file format is INTELHEX8 or INTELHEX16. This format is recognized and loaded automatically by the corresponding simulator. The source code inclusive commend and OP-code is deposit in the LST-file. In case of error the output of the message takes place at the screen as well as at the ERR file. OBJ-, SYM- and LST then will not be created. Additionally a DEBUG file will be created when the assembler will be started directly after a compiling operation with the BASIC compiler iL_BAS16. With this the simulater iL_SIM16 will be able later on to run the BASIC source code step by step.
INTELHEX16 is selected when compiler switch /M_OBJ is on or with the help of the $LIST command.

Assembler directives

Assembler instructions

The label has to start in the first column of the line, at least one leading blank is necessary for the instruction. Comments are marked by a colon (**;**)

Commands:

Some instructions are not for the assembled program but for the assembler itself. These instructions are called directives. The shareware version doesn't support the DEVICE-instruction. Therefore in this version the default line will be chosen automatically.

```
DEVICE          16C83,XT_OSC,WDT_OFF,PROJECT_OFF
```

The format of an assembler line is:

label instruction argument ;comment

a) label

The label must begin at the first place of the line. Is there no label before a commend you need at least one blank before the command. Commets will be introduced by a semicolon (**;**).

attention !

If there is only a symbol or a label in a line the actual program counter reading will be allocated to this value. An error message doesn't appears.

example:

```
          ORG    15h
COUNT
```

Value 15h will be allocated to the symbol COUNT. Therefore you only should use this notation in jump labels. Symbols must be defined in the same line with EQU.

b) command

Beside the actually mnemonic commands of the program the assembler also knows further commands. These commands shall influence the sequence of the assembling operation. You can find the commands in chapter 4. The assembler instructions are:

ORG *nnn*

The following code assembled by iL_ASS16 ist stored at the address *nnn*

EQU symbolname value

iL_ASS16 replaces the symbolname by value, for example:

```
BAUD   EQU   10h
```

LIST INHX8 or INHX16

iL_ASS16 produces a OBJ file in Intel-Hex-8 format or Intel-Hex-16 format. The extension is always .OBJ

LIST /M_OBJ

The produced obj file can be loaded into microchips picstart programer.

Assembler directives (cont.)

LIST C=xxx
Fixes the line width in the LST-file. Pay attention that the assembler adds 18 columns at the left side. There you can find the line numbers of the source file, may be marked with an "I". In case the INCLUDE-file is read you will find the address of the codes and the opcode itself. Only after that the symbol column anf the rest will follow.

LIST BIN
LIST BINX
An additional file in binary format with the extension BIN will be produced. This file only includes the dates for the program code. It doesn't give any information about the type of module or the bits of the configuration areas. The file area is missing at the 16X8x module. The switch BIN places the dates in the following manner highbyte - lowbyte in the binary-file.
BINX exchanges low and high byte. All arguments can be listed, seperated by colons.

LIST OBJ2HEX
Creates the ending HEX instead og OBJ. A lot of programming tools need the ending HEX.
Different LIST-instructions can be seperated in the same line by commas. (In basic-syntax the command is $LIST).

IF *condition*
ELSEIF
ENDIF
If *condition* is true (=0) the following lines up to ELSEIF or ENDIF will be assembled. If *condition* is not true (=1) all lines between IF and ELSEIF will be ignored, the lines between ELSEIF end ENDIF will be assembled. IF clauses may not be nested.

IDENT *nnnn*
The digits *nnnn* are stored into the id area of the controller (not available with picstart).

OLDVAR
The elder assembler versions predefined some symbols, f.g. INDIRECT, RTCC, ... This function is dropped with version 5.0. If you want to use these predefined symbols you have to set the assembler switch OLDVAR. Otherwise the user has to define these symbols again. (In basic syntax the switch ist $OLDVAR).

DEVICE
Defines the type of controller e.g.
12C508 12C509 12E518 12E519 12F629 12C671 12C672 12F675 16C53 16C54 16C55 16C56 16C57 16C58 16C61 16C62 16C63 16C64 16C65 16C66 16C620 16C621 16C622 16E623 16E624 16E625 16C71 16C72 16C73 16C74 16C76 16C77 16F818 16F819 16C83 16C84 16F83 16F84 16F873 16F874 16F876 16F877
(This list increases permamently!)
You can also write e.g. PIC16C84 instead 16C84
In case of A-type PICs add suffix A to device name e.g. 16C65A.

The oscillator type is selected by
LP_OSC oscillator typ LOW POWER
XT_OSC " xtal or resonator
HS_OSC " HIGH SPEED
RC_OSC " RC circuit

Assembler directives (cont.)

IRC_OSC " internal RC (12Cxxx, etc)
ERC_OSC " external RC (12Cxxx, etc)

WDT_ON watchdog on
WDT_OFF watchdog off

PROTECT_ON code protection on (no readout of program memory)
PROTECT_OFF code protection off (program memory can be read out)

PWRTE_ON power up timer on (not every PIC)
PWRTE_OFF power up timer off (not every PIC)

MCLR_INT internal reset generation (12Cxxx, etc)
MCLR_EXT reset circuit is connect to pin MCLR (12Cxxx, etc)

INCLUDE filename.ext

With the help of the INCULDE-files you can write definite program modules, e.g. definition of symbols, in a separate file. They will be read during assemblation and handled as if they stand directely in the main-file. So you can relieve the main file and get more clearness. Include files are not allowed to have include devices for their own. The INCLUDE-file will be marked in the LST file. Because INCLUDE is a command it isn't allowed to take the position of a label!
(In the BASIC-source text $INCLUDE is used).

c) arguments

In addition to decimal-, hex- and binary-format you also can use ASCII characters which must be enclosed by apostrophe (').
You can enter numbers in decimal format, hexadecimal format by adding the suffix H and
binary format with a suffix B. A ascii character is set in quotes.

e.g.

10	64H	01100101B	'A'
dec.	hex	binary	ascii 65 (41H)

You also can calculate arguments but only one calculation respectively algebraic signs can be done. No blanks are allowed. Between a sign and the argument no blank is accepted.

write: VAR1+VAR2
VAR1 + VAR2
VAR1+ VAR2
VAR1 +VAR2

wrong: VAR1 + VAR2
!VAR1 + VAR2 (=two operators)
!VAR1

Available operators are:

+ addition or sign
-- subtraction or sign
* multplication
/ division
& logical AND
| logical OR

Assembler directives (cont.)

^ logical EXCLUSIVE-OR
<< shift left (var1 << 4)
>> shift right. .(var1 >> 2)
! negate
$ $ is replaced by the actual value of program counter ($+3)

If the result is higher than 255 (-128, +127) a error message is generated, execpt: CALL, GOTO, ORG und EQU.

The assembler knows some predefined symbols when OLDVAR is set.

INDIRECT	for file 0 (register for indirecte adressing)
RTCC	for file 1 (real time clock/counter)
PC	for file 2 (programmcounter)
STATUS	for file 3 (statusregister, flags)
FSR	for file 4 (file select register)
RA	for file 5 (port A)
RB	for file 6 (port B)
RC	for file 7 (port C), if 16C55 or 16C57
TRUE	0
FALSE	1

In addition with the statusregister the following symbols are defined:

C	Carry /Borrow
Z	Zero
DC	DigitCarry /Borrow
PD	Power Down
TO	Time Out
PA0	Page Preselect Bit 0 (16C5x)
PA1	Page Preselect Bit 1 (16C5x)
PA2	Page Preselect Bit 2 (16C5x)
RP0	Register Page Select direct (16C71, 16C84)
RP1	Register Page Select direct (16C71, 16C84)
IRP	Register Page Select indirect (16C71, 16C84)

Attention !
The shareware-version does not know the device instruction so the default values take place.
e.g. DEVICE 16C83,XT_OSC,WDT_OFF,PROJECT_OFF

PIC assembler basic instruction set

```
ADDWF f,d INCF f,d TRIS f
ANDLW    k         INCFSZ  f,d       XORLW  k
ANDWF    f,d       IORLW   k         XORWF  f,d
 BCF      f,b       IORWF   f,d
BSF      f,b       MOVF    f,d
BTFSC    f,b       MOVLW   k         ADDLW  k
BTFSS    f,b       MOVWF   f         RETFIE
CALL     k         NOP               RETURN
CLRF     f         OPTION            SUBLW  k
 CLRW               RETLW   k
 CLRWDT             RLF     f,d
 COMF     f,d       RRF     f,d
DECF     f,d       SLEEP
 DECFSZ   f,d       SUBWF   f,d
 GOTO     k         SWAPF   f,d
```

Hint: f=file register, d=dest. (W or 0 ->result to w; 1=result to f). Default of d is 1

ADDWF f,d change Z,C,DC
Add W and f. result to W, if d=0, else in f

ANDLW k change Z
logical AND of W and k, result to W

ANDWF f,d change Z
logical AND of W and f, result to W, if d=0, else in f

BCF f,b change -
clears bit b in file register f

BSF f,b change -
set bit b in file register f

BTFSC f,b change -
test bit b in file register f, skip next instruction if bit=0

BTFSS f,b change -
test bit b in file register f, skip next instruction if bit=1

CALL k change -
call subroutine on address k. Return address is put on top of stack. PIC16C5x have a two level stack and the destination address must be within 0 ... FFH on each page. All other PIC (16C71 and 16C84) have a eight level deep stack and no restriction in the destination address of the subroutines.

CLRF f,d change Z
clears file register f

CLRW change Z
clears W

PIC assembler basic instruction set (cont.)

CLRWDT change TO=PD=1
resets watchdog timer

COMF f,d change Z
calculate one's complement from file register f. Result to W, if d=0, else in f

DECF f,d change Z
decrements file register f, result to W, if d=0, else in f

DECFSZ f,d change -
decrements file register f, skip next instruction if result is 0. Result is stored in W, if d=0 else in f

GOTO k change -
Branch to address k. At 12C5xx and 16C5x k is only 9 bit wide and accesses a range from 0 to 1FFH. To get to other program pages it is necessary to set PA0, PA1 and PA2 in the status register accordinly. All other PICs have a page size of 2048 (0..7FFH). To get to other program pages the pclath register must set accordingly.

INCF f,d change Z
Increment the contents of file registers f. Result to W if d=0, else in f

INCFSZ f,d change -
Increment file registers f. Skip next instruction if result i 0. Result to W, if d=0

IORLW k change Z
Logical OR of W and k

IORWF f,d change Z
Logical OR of W and file register f. Result to W, if d=0

MOVF f,d change Z
Move contents of f to itself or to W (allows setting zero flag if result is 0)

MOVLW k change -
Move literal (constant) k to W

MOVWF f change -
Move W to file register f

NOP change -
No operation

OPTION change -
Move W to option register

RETLW k change -
Returns from subroutine. Literal k is loaded into W (e.g. for tables)

RLF f,d change C
Rotate file register to left througth carry. Result is in W, if d=0 else in f-register
RRF f,d change C
Rotate file register to right througth carry. Result to W if d=0 else in f-register

PIC assembler basic instruction set (cont.)

SLEEP change PD=0, TO=1
Reduces power consumption. Nearly every function is stopped. To stop sleep mode reset is needed for 12C5x and 16C5x. Other PICs (16C6x, 16C7x and 16C8x) can use an interrupt to end sleep mode.

SUBWF f,d change Z,C,DC
Subtract W from f. Result to W if d=0, else to f-register.
ATTENTION! Carry is handled as borrow bit (it is inverted to the general opinion)

SWAP f,d change -
Swaps the two nibbles of the contents in file register f. Result is in W if d=0

TRIS f change -
W is moved to the tris register to defines the port pins a inputs or outputs

XORLW k change Z
Logical EXCLUSIVE OR of W and k.

XORWF f,d change Z
Logical EXCLUSIVE OR of W and file reg. f. Result to W if d=0

PICs with 14 bit cores has four additional instructions:
ADDLW k **change C,DC,Z**
add literal to W
RETFIE change -
Return from interrupt, set gie flag
RETURN change -
Return from subroutine without changing W register
SUBLW k change C,DC,Z
Subtract W from literal

In all 14 bit core PICs the option and tris register are placed within the regular register set. For downward compatibility the instructions option and tris are still available but should not be used.

ATTENTION!!
Don't confuse with the BASIC instruction TRIS.

The subtract instruction set and reset the carry flag not as you usually think. It is a borrow bit. The reason is how the subtraction is done by the processor. The PIC makes a addition with the two's complement. This is:

```
        CLRF 10         f10=0
        MOVLW 1         w=1
        SUBWF 10        f10=0-1=0+FF=FF                 result is negative, but C=0
or
        MOVLW 0FFh
        MOVWF 10        f10=FF
        CLRW            w=0
        SUBWF 10        f10=FF-0 = FF     result is positive, but C=1
```

Simulator (general)

1. Introduction

Important!!!

There is a important difference between iL_SIM16STD and iL_SIM16PRO. iL_SIM16PRO supports the same devices as iL_BAS16PRO does. iL_SIM16STD supports only a part of them, that means every PIC as iL_BAS16STD does. the standard version can't simulate a program which has several memory pages. Which one is supported by which version is noted in appendix III.

IL_SIM is a tool to simulate all functions of Microchips microcontrollers. The standard version supports PIC12C508, PIC16C54, PIC16C55, PIC16C64, PIC16C71, PIC16C84 and PIC16F84 (actual list see www.iL-online.de). The professional version supports also these devices with more program memory, where calls und jumps over program pages are necessary. Both programs run under DOS in protected mode (386 or higher). For easy handling, the contents of all registers and parts are displayed on screen. This is the simulators screen. An available additional hardware (iL_VIEW16 or iL_HARD16) permits a connection of the simulator to the hardware!

The first line shows the number and date of release, controller type, contents of the configuration byte, like the type of oscillator (RC, HS, LP, XT, IRC, ERC), watchdog on or off (WDT_ON, WDT_OFF) and the status of the protection bit (PROTECT_ON, PROTECT_OFF). In the first column the program counter '>' (=PC) points to the next code to be execute. The lower 8 bits are found also in register F02, the upper bits in the status register as PA0 to PA2 (12Cxxx and 16C5x) or in the register PCLATH (12C6xx, 16C6x, 16C7x, 16X8x). Usually the focus of the disassembler listing is the PC and will keep visable execpt if you are modifying the program by hand. The second column the opcode, followed by the labels (symbols) and the mnemonics.

```
iL_SIM16   Ver. 5.2    29.04.96        16C54   XT-OSC  WDT-OFF  PROTECT-OFF (c)iL
================================================================================
 >0000 000  START    NOP                          W-REG. : 0 0 0 0 0 0 0 0 00
  0001 030           MOVWF  10                    S-REG. : 0 0 0 1t1p0z0d0c
  0002 040           CLRW                         PgSel. : 00 RTCC: 00 WD: --
  0003 071           CLRF   11                    PC     :1FF     FSR: 00
  0004 0B3           SUBWF  12                    Option : - - 1 1 1 1 1 1
  0005 092           SUBWF  12,W                 >Port A :         0i0i0i0i
  0006 0F3           DECF   13                    PORT B : 0i0i0i0i0i0i0i0i
  0007 0D3           DECF   13,W                  Port C : 0i0i0i0i0i0i0i0i
  0008 134           IORWF  14                    F00: 00  F0B: 00   F16: 00
  0009 114           IORWF  14,W                  F01: 00  F0C: 00   F17: 00
  000A 175           ANDWF  15                    F02:1FF  F0D: 00   F18: 00
  000B 155           ANDWF  15,W                  F03: 18  F0E: 00   F19: 00
  000C 1B6           XORWF  16                    F04: 00  F0F: 00   F1A: 00
  000D 196           XORWF  16,W                  F05: 00  F10: 00   F1B: 00
  000E 1F7           ADDWF  17                    F06: 00  F11: 00   F1C: 00
  000F 1D7           ADDWF  17,W                  F07: 00  F12: 00   F1D: 00
  0010 238           MOVF   18                    F08: 00  F13: 00   F1E: 00
  0011 218           MOVF   18,W                  F09: 00  F14: 00   F1F: 00
  0012 279           COMF   19                    F0A: 00  F15: 00
Quarz: 4.194304 MHz     Laufzeit: 0.000us         RTCC-Pin 0   Stop      0,0V
                                                                        ?

================================================================================
F1-Help F2-Rset F3-RgSel+ F4-RegMod F5-PortSel F6-WMod F7-SMod F8-Trace F9-Setp

ALT     F2-Load F3-PicSel F4-PrgMod F5-OMod  F6-LRst F7-Pattrn F8-PtOnOf F9-Go
```

Simulator (general) (cont.)

```
CTRL F2-ExRes F3-RgSel- F4-DiSig F5-RtccTg F6-ASig F7-ATrg F8-IgCmd F9-Step-

 SHIFT           F3-Brkpt F4-Bank   F5-Value  F6-BStr F7-BsOnOf F8-HxAsc F9-Go-
```

The functions of the different areas of the screen are:

In the headline there are the program name, number of version, the date, the at the moment active type of processor, also the information about the configuration-byte in the processor. It is shown the type of oscillator(RC,HS,LP,XT, IRC,ERC), the state of the watchdog timer (WDT_ON, WDT_OFF) and the readout protection (PROTECT_ON, PROTECT_OFF).

The right half of the screen shows the registers and their values. To change the value just click the desired register and overwrite the old value. The bits in the status register are marked: t means time out flag, p is power down flag, z is the zero flag, d is the digit carry flag (often called half carry) and c means carry flag. The upper three bits are not marked because their meaning is different and depends on the controller type. Their meaning depends either on the program counter (PA0 to PA2) or is used to select the ram bank RP0 and RP1 or for indirect memory access IRP.

The value of file register F01 can be incremented by a slope on the rtcc pin. The prescaler is shared by the rtcc and the watchdog timer. A change on the fly is possible e.g. during normal operation it is used along with the rtcc and before the chip is set in sleep mode (power down) you assign it to the watchdog to get longer reset intervals. The assignment and the prescaler factor is set in the OPTION register. If the rtcc counts from 255 to 0 (overflow) the timeout flag is set, and if in sleep mode the contoller is waked up.

Because of the compact command words (harvard architecture) the arguments in such a command word are limited. This means that there is no linear access to the file registers (data memory). So this memory is devided in seperate banks. The size of such a bank depends on the type of controller. PIC12C5xx and PIC16C5x have just 12 bits in a command word so the memory bank is only 32 bytes (00H to 1FH). Because the lower 16 Bytes are very very importend, this part will not be switched away, so if you have select the second page (Bit 5 in FSR is '1') and you read the contents of file register 1, you get the value of the rtcc. This means you select memory blocks from 10H to 1FH, 30H to 3FH, 50H to 5FH and 70H to 7FH, if available in this chip. Another way is used with the contoller types PIC12C6xx, PIC16C6x, PIC16C7x and PIC16XFx. These types of contoller use 14 bits in a command word. The bits to select the bank are not in the FSR register but in the status register. This is really complex so please refer microchips data sheets.

All type of pic controllers have a hardware stack. A stack is a memory (other cpu's uses regular data memory) in which the contoller writes very importent values. PICs use the stack for keeping the return adresses of a subroutine. The user cannot access this part of memory in the PIC. Take care, in the PIC12C5xx and PIC16C5x the stack is only two levels deep. A CALL in a CALL is ok but a CALL in a CALL in a CALL is too much an ends in program confusion. All other PICs have an eight level deep stack, what a waste, but anyway it must be enough.

Most of the hardware configuration is done by writing the OPTION register. The lower three bits defines the prescaler factor. Take care it is different if assigned to the rtcc or the watchdog. Bit 3 is the prescaler assignment, bit 4 selects the slope on which the rtcc will be incremented (bit 3 and bit 5 must be 0), bit 5 select the trigger source for the rtcc or prescaler. The meaning of the upper two bits depends on the controller type.

Simulator (general) (cont.)

RBUP	INTEDG	RTS	RTE	PSA	PS2	PS1	PS0

bit 7 RBUP some device has weak pull up at port RB, '1' with pull up
'0' without pull up

bit 6 INTEDG '0' falling edge interrupts program
'1' rising edge interrupts program

bit 5 RTS '0' rtcc is incremented by the internal source (Fosc/4)
'1' rtcc is incremented by a falling or rising edge at the rtcc pin

bit 4 RTE '0' rtcc is incremented by the rising edge at the rtcc pin
'1' rtcc is incremented by the falling edge at the rtcc pin

bit 3 PSA '0' prescaler is assigned to the rtcc register
'1' prescaler is assigned to the watchdog register

bit 2	bit 1	bit 0	rtcc	wdt
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Fig. 2 option register

The ports are shown in binary format. Each bit is marked as 'i' for input, 'o' for output, 's' for signal source (square wave) and 'd' for data source (TTL level of a RS232 input). Along with the PIC12C67x and PIC16C7x there are also an 'a' for analogue input and a 'r' for reference input. 'i' and 'o' are defined in the TRIS register where a '0' correspondens to an output, a '1' is input. On reset all pins are defined as inputs (remember if you wake up a PIC form sleep). 'a' and 'r' are defined in the register ADCON0 and ADCON1. 's' and 'd' are functions or the simulator for easy pin stimulus. In the function data source handshake is also available. But the assigned pin for this function is not marked because it is an ordinary pin which must be polled by software. Another function of this simulator is the analogue signal at the rtcc pin. This is a 'quasi' analogue signal because the rtcc pin is just a schmitt trigger input. So it is enough to define a rising slope, the time of remaing high and the falling slope. The time of remaining low is the rest to 100%. You may define the frequency and the hysteresis on the rtcc pin. Once started it runs until stopped. No synchronization is done between the slopes and the running program.

All timing , including the signal sources and runtime counter, is calulated on base of the actual xtal frequency. This value is setable between 1Hz and 20Mhz. For accessing the eeprom data space in PIC16X8x you must run through the procedure described in the data sheet. These datas are stored and loaded by the simulator in EEPROM.DAT.

The analogue signal sources (max 8) stored in ANALOG.DAT.

Getting started

Please copy all files on the disks root directory on your harddisk after having created a new directory (e.g. PICTOOLS). If you have bought the basic compiler iL_BAS16 too, copy the files of that disk into the same directory. On DOS-prompt type in il_SIM16 or il_SIM16 *filename*. Entering the optional filename the appropriate file will be loaded immediately after il_SIM16 is started. Do not type in the file extension because it is added automatically. The assumed extension is LST for the listfile.

Other parameters are
/comx,
/Q=1.5,
/P=stimulusfile.sti
/T=iotracefile.trc
/L=500
/E.

comx is a number from 0 to 4. It defines the com port where iL_VIEW16 (hardware in circuit simulator) is attatched. /0 means no iL_VIEW16 is used.

/Q= defines the xtal frequency

/P= invokes the pattern file (stimulus data)

/T= while simulation runs the changes at the io pins are recorded in this file (e.g. for documentation)

/L= defines the end time on which the simulator automatically stops

/E= after stopping the simulation, the program exit and return to the caller.

/B= nnn after nnn us the data source (serial) starts.

This parameter are very useful for batch files.

After program exit, the pic type, com port and colors are saved in iL_SIM16.CFG. Upon starting, this values are used if no parameters are set (parameter have higher priority). iL_SIM16.CFG is a ascii file with the first line for the com port (0 to 4), the second is 'm' for monochorme display and 'c' for color display. The third line contains the latest used pic type. The following lines are the colors for the screen. The values are adequate to those of BORLANDs PASCAL compiler.

Line	color for	default
4	back ground	3
5	regular text	15 (white)
6	head line	1
7	foot line	1
8	messages/inputs	4
9	assembler lines	5
10	register	5
11	window	1

For easier working you may use the editor ED16X. Now you can write your source program and assemble or compile it with just tying ALT F1 (assembler) or ALT F4 (compiler). To invoke the simulator you've to press ALT F2.

Quit the program
After pressing ESC you are asked if you really want to quit. Press 'Y' to exit the program.

Simulator commands

All commands will be invoked by pressing the appropriate F-Key. On the screen bottom line you can see the commands that are supported. There are another three command levels which you switched by pressing permanently the ALT-,CTRL- or SHIFT_key. Click the bottom line with the right mouse key and the commands will be scrolled. Click on the appropriate function with the left mouse key starts this function. The middle mouse key is used like the ESC key.

F1-Help.
Invokes the help-screen with its 11 pages to turn over the page-up and page-down key. To quit the Help-screen press ESC. To turn over by using the mouse position the cursor at the arrows in the frame of the auxiliary window and press the left mouse button.

F2-Rset.
Reset the CPU like on power on. The program counter is set to the highest available value (all one's) if the selected controller is PIC12C5xx or PIC16C5x, with all other PIC controllers the program counter is set to zero. The i/o-pins are defined as inputs. The selected ram bank is the first, etc (see also CTRL-F2).

F3-Rg-Sel+.
Move the file pointer to the next position. Only the marked register file can be changed. Using mouse control, it will do when you position the mouse at the corresponding register and press the left mouse button. So the pointer is set at this position. Clicking at this position for a second time you active automatically the function F4 and you can change the contents of this register.
Hint: See also F4 and CTRL F3.

F4-RegMod.
A new value can be written in the marked file. You have to input in Hex-format.
Hint: See also F3 and CTRL F3.

F5-PortSet.
Moves the port pointer to the next position. Only the pointed port can be changed. You toggle the bits 0-7 by pressing the appropriate key 0-7. Mouse click toggles the pin.

F6-WMod.
Allows you to change the W-register.

F7-SMod.
Allows you to change one of the bits in the statusregister. The entry is e.g. Z=1 to set the zeroflag or C=0 to reset the carryflag. The abbrivations for the flags are:
C = carrybit
D = digitcarry
Z = zerobit
P = powerdown
T = time out
The page-select-bitsPA0-PA2 or register-page-bits RP0, RP1 and IR don't have a lable (identification) but they are changable.

Having a mouse you just have to put it at the desired bit and press the left mouse button. The bit will be inverted at that.

F8-Trace.
The command pointed by the program counter is simulated and then the display is updated.

F9-Step.

Simulator commands (cont.)

A temporary breakpoint is set to the next command and the command pointed by the program counter is simulated; useful for GOTO's and CALL's.

ALT F2-Load.
After enter the filename (without extension) the file will be loaded and a reset is executed. ALT F2 without filename reloads the current file.

ALT F3-CpuSel.
Change the Processor (not available in the Shareware-version where only the 16C54 is supported). Using mouse control you can change the type by positioning the mouse at the shown prozessor in the top line and press the left mouse button.

ALT F4-PrgMod.
You are asked to type in the appropriate address. Now you are able to change the code (Entry in hex format). You quit this function by pressing the ENTER key only. Using hte mouse just set the mouse cursor at the corresponding command and press the left mouse button.

ALT F5-O-Mod
The bits of the option register are listed and may be toggled by typing the bit numer (0 to 7) or just click it on.
Allows you to change part of the option register.
PSX = 0 ... 7 Value for the prescaler
PSA = 0.... 1 prescale to 0=rtcc 1=wdt
RTE = 0 ... 1...rtcc triggerslope 0=riging edge 1= falling edge
RTS = 0 ... 1 ..rtcc-clock from 0= intern 1= rtcc-pin.

ALT F6-LRst.
Resets the runtime counter to 0us. the runtime is indicated in us (10E-6) with a resolution of 1 ns. Using a 4,194304 MHz-quartz the cycle-time runs by this at 954 ns.

ALT F7-Pattern
Opens a window for entering end editing the pattern file. You may input up to 500 times and the io levels to set at this time. The first line is the enable line which means that only those pins will be changed whose corresponding bit is set to '1' (input). If this bit is set to '0' (output) the output driver is stronger than the input form the pattern file. The following lines contains the time (in us) when the new io information should be set to the pin and the level that is set to the pins. MCLR input in internally stored as RA5.

ALT F8-PtOnOf.
Starts and stops the work of the pattern file.

ALT F9-Go.
Starts a continuous simulation. To stop the simulation press ESC. Several keys are still scanned. Key 0-7 toggles the appropriate bit in the selected part, F5, CTRL-F5, Shift-F6 and Shift-F7 can still be used. so that the running program meets different conditions on the i/o-pins and the rtcc-pin.

CTRL F2-ExRes
Corresponds to manual reset along with an enclosed operating voltage. The bits TO and WD in the status register are set accordingly. So you can differ a manuel reset, which ends the sleep mode, from a switch-on of the operating voltage.

CTRL F2- AnaIn (only 12X67x 16X7x and 1687x)

Simulator commands (cont.)

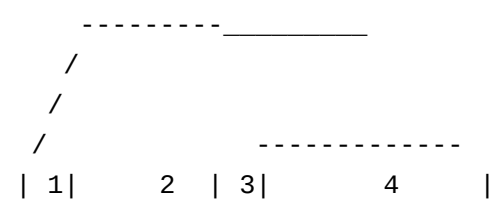
You define the maximum level and the frequency of an analogue signal for a desired analogue i/o pin. If the frequency is higher than 0Hz you may define the signal shape. Some shapes are predefined but you may also define one by your own. In this case you have to input up to 64 points which were spreaded linear by 4 points.

CTRL F3-RgSel-.
Moves the register file pointer to the previous position.
HInt: See also F3 and F4.

CTRL F4-DiSig.
Allows you to "connect" a squarewave generator to any i/o-pin. The frequency is selectable from 1Hz to 100kHz in steps of 1Hz. The duty cycle can also be changed. E.g. to connect this generator with 10kHz, duty-cycle 1:4 to pin RA 3 please type in:
RA3 (---means disconnect)
10.000 (Entry in thousands of Hz)
1:5 (High-phase, low-phase)

CTRL F5-RtccTg.
Toggles the information on the RTCC pin. This can influence the rtcc register and/or the prescales if assigned to rtcc.

CTRL F6-AnSig.
An analogue source can "connect" to the RTCC pin. The frequency can also be set between 1Hz and 100kHz. The waveform is trapezoid with variable rising ramp, high peri-ode, falling rampramp and low periode. The timing is entered in percentage with the total period (1+2+3+4) is 100%. There are only 3 values to enter, the forth value is the rest missing to 100%. Typical values are:
Triangle (sqm) 50,0 ; 50,0.
Square wave (1:1) 0,50 ; 0,50
Sinus (1.Nährung) 30,40,30,0
" " 15,20,15,50



The threshold of the rtcc-pin and the hysteresis are not fixed.The analogue source can be triggered. Manualtrigger by CTRL-F7 or by the logic level of an i/o pin. This trigger source can be any i/o pin, the polarity is selectable, too. As long as the triggered condition is true the generator starts with a new period after finish the old one (free running function generator). Becomes the trigger condition false within an actual period, this period will be finished before the generator is stopped.

CTRL F7-AnTrig.
The function generator is enabled for one period. If once enabled at least one periode is send out. Is the trigger still enable, free running mode is enabled.

CTRL F8-IgCmd.
The command pointed by the program counter will be ignored. Has the same result as setting the program counter to the next address.Having a mouse it is set to the desired adress in the adress column and press the right mouse button.

CTRL F9-Step.
The same function as F9 except that there is no screen refresh and therefore the program runs quite faster.

Shift F3-Brkpt.

Simulator commands (cont.)

Allows you to define a breakpoint to a certain address. This breakpoint is only effective with Go (F9 or Shift F9). To clear the breakpoint enter '-' instead of the address.

Shift F4-Bank.
If the type 16C57 is active you can select the available banks.

Shift F5-Value
Five function must be select by pressing key 1 to 5.
1 Enter a new xtal frequency in Mhz.
2. Define the upper und lower threshold of the rtcc pin
3. Turn the watchdog on or off.
4. If iL_VIEW16 is attatched and aktive, only the simulator reads and writes this hardware. If your program has less io transfers, the changing at your hardware is not visable on screen. This value defines the numbers of codes that are executed before the simulator is forced to read the io pins. A good value is 100.
5. If the simulator runs in the fastest mode, no screen refresh is done (it cost the majority of time). This value defines the numbers of codes that are execueted before the simulator is forced to refresh the screen.

ClkSel
Here you can put in another quartz fequency. The cycle time will be adapted correspondently.
RtccHyst.
It permits to set the lower or upper threshold at the Rtcc-pin. Default value is 0.8V and 4.3V, which corresponds approximately to the data sheet. Is thevalue of the input 0 first the rated voltage must go up to 4.3V, that an "1" can be recognized. Corrspondingling the rated voltage must sink to 0.8V to recognize a "0".
WDtog
Switch on respectively switch off of the watchdogtimer. Using a mouse you can change the watchdogtimer by setting the mouse at the shown prosessor at the upper line and press the left mouse button.
I/O-access
This function just becomes relevant when the haredware adapter iL_VIEW16 or iL_HARD16 are connected and activ. You can tell the simulater after howmany commands it has to read in the state of the adapter. You can put in values beteween 1 and 32000. The value 100 it a good compromise.
Compulsory refresh
Usually you have no refreshment of the screen in a SHIFT F10 GO-mode. Here you can determine, that the screen contents will be updated after 100 done coomands.

SHIFT F6-BStr.
Allows you to 'attatch' a data stream to an input pin. This serial data stream is defined by the baud rate, parity, numbers of bits ant CTS pin for handshake. The text string is limited to 16 characters. Characters below 20H (blank) must be entered as a hexadecimal digit with a leading '#' character. The appropriate pin is marked by a 'd'.

Shift F7-BsOnOff
Starts the data stream with sending the bits to the selected pin (under respect of RTS, if defined, 1=stop).

SHIFT F8-HxAsc
The appearence of the registers are switched over to ascii or back to hexadecimal.

Shift F9-GO
Invokes the simulation without screen refresh. The simulation is stopped by a breakpoint or ESC-key. The screen is refreshed anyway, if any external states are changing.

Simulator commands (cont.)

SHIFT F10-Go--
In this mode the simulator runs on its maximum speed because any screen refresh is suppressed. Only after a specified numbers of cycles a single refresh is done.
To reset the analogue and digital signal generator press CTRL-F10 or ALT-F10. Now the periode starts at the first point.

RUN UNTIL
This function can invoked by mouse click only. Set mouse pointer to the first coloumn of the address coloumn and press left mouse button. A temporary brakpoint is set and the simulation starts with maximum speed. The program is simulated until it reaches the breakpoint. If a breakpoint was already set at this address the breakpoint will be first reset before simulation starts.

BASIC high level debugging
To simulate BASIC programs the simulator needs the LST file generated by iL_BAS16 and iL_ASS16. After loading the simulator and the LST file the assembler code is visible first. Press F10 to switch to the BASIC lines.

For testing BASIC programs you can use following commands:

TRACE (F8)
The highlighted BASIC line is executed.

STEP (F9)
A temporary breakpoint is set on the line which follows the highlighted line. Then simulation starts with maximum speed. This command is useful to "execute" subroutines fast.

IGNORE-CMD (CTRL-F8)
The highlighted BASIC line is not executed but skipped.

GO (SHIFT-F9)
Starts the Simulation at the highlighted BASIC line and executes line by line.

BaudCalc

BaudCalc is a utility program to calculate the delay constants for SERIN and SEROUT. Normaly you set the baud rate in these two instructions as a constant e.g. 1200 or 4800. But sometimes it is very interessting to change the baud rate without using two separate instructions. This can be necessary when you have to compress your program to fit in a smaller PIC. For this the parameter baud rate must be a variable. But the real value that is used by the runtime library isn't that value. The compiler calculates the value under respect of the xtal frequency, the used core and the baud rate value. There is no linear proportion which makes ist difficult to calculate. For a PC and the compiler itself it is no problem. But to calculate this value within the runtime library would take to much memory space and time. So it is quiet better to keep this calculation out ouf the PIC. This is done by BaudCalc. Enter the desired baud rate, the used core (depends on which PIC) and the xtal frequency of the target hardware. Then press 'CALCULATION' and the value that results in the desired baud rate will be displayed.

```
LET var_baud=205     'baud rate value is a variable
SERIN rb,0,var_baud,var_get
```

DEFAULT.EQU

```
;Stand (Release) (ddmmyy) 22.07.2003
;--------------------------------------------------------------------------

;--------------------------------------------------------------------------
:12C508
:12C509
:12E518
 :12E519
INDIRECT  EQU  00h       DC        EQU  1         ARG3      EQU  0Eh
TMR0      EQU  01h       C         EQU  0         ARG4      EQU  10h
RTCC      EQU  01h       FSR       EQU  04h       ARG5      EQU  12h
PCL       EQU  02h       OSCCAL_   EQU  05h       ERRVAR    EQU  13h
PC        EQU  02h       RB        EQU  06h       ERR_      EQU  13h
STATUS    EQU  03h       GPIO      EQU  06h       DATPTR_   EQU  14h
GPWUF     EQU  7         GP_       EQU  06h       RBTRIS_   EQU  16h
PA0       EQU  5         ERG       EQU  08h       :
TO        EQU  4         ARG0      EQU  08h
PD        EQU  3         ARG1      EQU  0Ah
Z         EQU  2         ARG2      EQU  0Ch
;--------------------------------------------------------------------------
:12C671
:12C672
:12E673
 :12E674
INDIRECT  EQU  00h       INTF      EQU  0BH,1     PCON      EQU  8Eh
TMR0      EQU  01h       GPIF      EQU  0BH,0     POR       EQU  8Eh,1
RTCC      EQU  01h       PIR1      EQU  0CH       OSCCAL_   EQU  8Fh
PCL       EQU  02h       ADIF      EQU  0Ch,6     ADCON1    EQU  9Fh
PC        EQU  02h       ADRES     EQU  1Eh       PCFG2     EQU  9Fh,2
STATUS    EQU  03h       ADCON0    EQU  1Fh       PCFG1     EQU  9Fh,1
IRP       EQU  7         ADCS1     EQU  1Fh,7     PCFG0     EQU  9Fh,0
RP1       EQU  6         ADCS0     EQU  1Fh,6     PUPOPW    EQU  20h
RP0       EQU  5         CHS1      EQU  1Fh,4     DATPTR_   EQU  21h
TO        EQU  4         CHS0      EQU  1Fh,3     ERG       EQU  70h
PD        EQU  3         GO_DONE   EQU  1Fh,2     ARG0      EQU  70h
Z         EQU  2         ADON      EQU  1Fh,0     ARG1      EQU  72h
DC        EQU  1         OPTION_R  EQU  81H       ARG2      EQU  74h
C         EQU  0         GPUP      EQU  81H,7     ARG3      EQU  76h
FSR       EQU  04H       INTEDG    EQU  81H,6     ARG4      EQU  78h
GPIO      EQU  05h       T0CS      EQU  81H,5     ARG5      EQU  7Ah
RA        EQU  05h       T0SE      EQU  81H,4     ERR_      EQU  7Bh
PCLATH    EQU  0Ah       PSA       EQU  81H,3     ERRVAR    EQU  7Bh
INTCON    EQU  0BH       PS2       EQU  81H,2     PUPOPF    EQU  7Ch
GIE       EQU  0BH,7     PS1       EQU  81H,1     PUPOPS    EQU  7DH
PEIE      EQU  0BH,6     PS0       EQU  81H,0     PUPOPP    EQU  7EH
T0IE      EQU  0BH,5     TRISGP    EQU  85h       TIMERX    EQU  7FH
INTE      EQU  0BH,4     TRISA     EQU  85h       :
GPIE      EQU  0BH,3     PIE1      EQU  8Ch
T0IF      EQU  0BH,2     ADIE      EQU  8Ch,6
;--------------------------------------------------------------------------
 :12F629
```

DEFAULT.EQU (cont.)

```
INDIRECT EQU 00h TMR1CS EQU 10H,1 WPU0 EQU 95H,0
TMR0      EQU  01h       TMR1ON    EQU  10H,0     IOCB      EQU  96H
RTCC      EQU  01h       CMCON     EQU  19H       IOCB5     EQU  96H,5
PCL       EQU  02h       COUT      EQU  19H,6     IOCB4     EQU  96H,4
PC        EQU  02h       CINV      EQU  19H,4     IOCB3     EQU  96H,3
STATUS    EQU  03h       CIS       EQU  19H,3     IOCB2     EQU  96H,2
IRP       EQU  7         CM2       EQU  19H,2     IOCB1     EQU  96H,1
RP1       EQU  6         CM1       EQU  19H,1     IOCB0     EQU  96H,0
RP0       EQU  5         CM0       EQU  19H,0     VRCON     EQU  99H
TO        EQU  4         ADFM      EQU  1Fh,7     VREN      EQU  99H,7
PD        EQU  3         VCFG      EQU  1Fh,6     VRR       EQU  99H,5
Z         EQU  2         CHS1      EQU  1Fh,3     VR3       EQU  99H,3
DC        EQU  1         CHS0      EQU  1Fh,2     VR2       EQU  99H,2
C         EQU  0         GO_DONE   EQU  1Fh,1     VR1       EQU  99H,1
FSR       EQU  04H       ADON      EQU  1Fh,0     VR0       EQU  99H,0
GPIO      EQU  05h       OPTION_R  EQU  81H       EEDTA     EQU  9AH
RA        EQU  05h       GPUP      EQU  81H,7     EEADR     EQU  9BH
PCLATH    EQU  0Ah       INTEDG    EQU  81H,6     EECON1    EQU  9CH
INTCON    EQU  0BH       T0CS      EQU  81H,5     WRERR     EQU  9CH,3
GIE       EQU  0BH,7     T0SE      EQU  81H,4     WREN      EQU  9CH,2
PEIE      EQU  0BH,6     PSA       EQU  81H,3     WR        EQU  9CH,1
T0IE      EQU  0BH,5     PS2       EQU  81H,2     RD        EQU  9CH,0
INTE      EQU  0BH,4     PS1       EQU  81H,1     EECON2    EQU  9DH
GPIE      EQU  0BH,3     PS0       EQU  81H,0     PUPOPW    EQU  4Eh
T0IF      EQU  0BH,2     TRISGP    EQU  85H       DATPTR_   EQU  4Fh
INTF      EQU  0BH,1     TRISA     EQU  85H       ERG       EQU  50h
GPIF      EQU  0BH,0     PIE1      EQU  8CH       ARG0      EQU  50h
PIR1      EQU  0CH       EEIE      EQU  8CH,7     ARG1      EQU  52h
EEIF      EQU  0CH,7     ADIE      EQU  8Ch,6     ARG2      EQU  54h
ADIF      EQU  0CH,6     CMIE      EQU  8CH,3     ARG3      EQU  56h
CMIF      EQU  0CH,3     TMR1IE    EQU  8CH,0     ARG4      EQU  58h
TMR1IF    EQU  0CH,0     PCON      EQU  8EH       ARG5      EQU  5Ah
TMR1L     EQU  0EH       POR       EQU  8EH,1     ERR_      EQU  5Bh
TMR1H     EQU  0FH       BOD       EQU  8EH,0     ERRVAR    EQU  5Bh
T1CON     EQU  10H       OSCCAL_   EQU  90H       PUPOPF    EQU  5Ch
TMR1GE    EQU  10H,6     WPU       EQU  95H       PUPOPS    EQU  5DH
T1CKPS1   EQU  10H,5     WPU5      EQU  95H,5     PUPOPP    EQU  5EH
T1CKPS0   EQU  10H,4     WPU4      EQU  95H,4     TIMERX    EQU  5FH
T1OSCEN   EQU  10H,3     WPU2      EQU  95H,2     :
T1SYNC    EQU  10H,2     WPU1      EQU  95H,1
;--------------------------------------------------------------------------
 :12F675
INDIRECT  EQU  00h       CINV      EQU  19H,4     IOCB1     EQU  96H,1
TMR0      EQU  01h       CIS       EQU  19H,3     IOCB0     EQU  96H,0
RTCC      EQU  01h       CM2       EQU  19H,2     VRCON     EQU  99H
PCL       EQU  02h       CM1       EQU  19H,1     VREN      EQU  99H,7
PC        EQU  02h       CM0       EQU  19H,0     VRR       EQU  99H,5
STATUS    EQU  03h       ADRESH    EQU  1Eh       VR3       EQU  99H,3
IRP       EQU  7         ADCON0    EQU  1Fh       VR2       EQU  99H,2
RP1       EQU  6         ADFM      EQU  1Fh,7     VR1       EQU  99H,1
RP0       EQU  5         VCFG      EQU  1Fh,6     VR0       EQU  99H,0
```

DEFAULT.EQU (cont.)

```
TO EQU 4 CHS1 EQU 1Fh,3 EEDTA EQU 9AH
PD        EQU  3         CHS0      EQU  1Fh,2     EEADR     EQU  9BH
Z         EQU  2         GO_DONE   EQU  1Fh,1     EECON1    EQU  9CH
DC        EQU  1         ADON      EQU  1Fh,0     WRERR     EQU  9CH,3
C         EQU  0         OPTION_R  EQU  81H       WREN      EQU  9CH,2
FSR       EQU  04H       GPUP      EQU  81H,7     WR        EQU  9CH,1
GPIO      EQU  05h       INTEDG    EQU  81H,6     RD        EQU  9CH,0
RA        EQU  05h       T0CS      EQU  81H,5     EECON2    EQU  9DH
PCLATH    EQU  0Ah       T0SE      EQU  81H,4     ADRESL    EQU  9EH
INTCON    EQU  0BH       PSA       EQU  81H,3     ANSEL     EQU  9Fh
GIE       EQU  0BH,7     PS2       EQU  81H,2     ADCON1    EQU  9FH
PEIE      EQU  0BH,6     PS1       EQU  81H,1     ADCS2     EQU  9FH,6
T0IE      EQU  0BH,5     PS0       EQU  81H,0     ADCS1     EQU  9FH,5
INTE      EQU  0BH,4     TRISGP    EQU  85H       ADCS0     EQU  9FH,4
GPIE      EQU  0BH,3     TRISA     EQU  85H       ANS3      EQU  9FH,3
T0IF      EQU  0BH,2     PIE1      EQU  8CH       ANS2      EQU  9FH,2
INTF      EQU  0BH,1     EEIE      EQU  8CH,7     ANS1      EQU  9Fh,1
GPIF      EQU  0BH,0     ADIE      EQU  8Ch,6     ANS0      EQU  9Fh,0
PIR1      EQU  0CH       CMIE      EQU  8CH,3     PUPOPW    EQU  4Eh
EEIF      EQU  0CH,7     TMR1IE    EQU  8CH,0     DATPTR_   EQU  4Fh
ADIF      EQU  0CH,6     PCON      EQU  8EH       ERG       EQU  50h
CMIF      EQU  0CH,3     POR       EQU  8EH,1     ARG0      EQU  50h
TMR1IF    EQU  0CH,0     BOD       EQU  8EH,0     ARG1      EQU  52h
TMR1L     EQU  0EH       OSCCAL_   EQU  90H       ARG2      EQU  54h
TMR1H     EQU  0FH       WPU       EQU  95H       ARG3      EQU  56h
T1CON     EQU  10H       WPU5      EQU  95H,5     ARG4      EQU  58h
TMR1GE    EQU  10H,6     WPU4      EQU  95H,4     ARG5      EQU  5Ah
T1CKPS1   EQU  10H,5     WPU2      EQU  95H,2     ERR_      EQU  5Bh
T1CKPS0   EQU  10H,4     WPU1      EQU  95H,1     ERRVAR    EQU  5Bh
T1OSCEN   EQU  10H,3     WPU0      EQU  95H,0     PUPOPF    EQU  5Ch
T1SYNC    EQU  10H,2     IOCB      EQU  96H       PUPOPS    EQU  5DH
TMR1CS    EQU  10H,1     IOCB5     EQU  96H,5     PUPOPP    EQU  5EH
TMR1ON    EQU  10H,0     IOCB4     EQU  96H,4     TIMERX    EQU  5FH
CMCON     EQU  19H       IOCB3     EQU  96H,3     :
COUT      EQU  19H,6     IOCB2     EQU  96H,2
;-------------------------------------------------------------------------
 :16C505
INDIRECT  EQU  00h       C         EQU  0         ARG4      EQU  10h
TMR0      EQU  01h       FSR       EQU  04h       ARG5      EQU  12h
RTCC      EQU  01h       OSCCAL_   EQU  05h       ERRVAR    EQU  13h
PCL       EQU  02h       PORTB     EQU  06h       ERR_      EQU  13h
PC        EQU  02h       RB        EQU  06h       DATPTR_   EQU  14h
STATUS    EQU  03h       PORTC     EQU  07H       RATRIS_   EQU  15h
GPWUF     EQU  7         RC        EQU  07h       RBTRIS_   EQU  16h
PA0       EQU  5         ERG       EQU  08h       RCTRIS_   EQU  17h
TO        EQU  4         ARG0      EQU  08h       :
PD        EQU  3         ARG1      EQU  0Ah
Z         EQU  2         ARG2      EQU  0Ch
DC        EQU  1         ARG3      EQU  0Eh
;-------------------------------------------------------------------------
:16C53
```

DEFAULT.EQU (cont.)

```
:16C54
:16C55
:16C56
:16C57
 :16C58
INDIRECT  EQU  00h       C         EQU  0         ERRVAR    EQU  13h
TMR0      EQU  01h       FSR       EQU  04h       ERR_      EQU  13h
RTCC      EQU  01h       PORTA     EQU  05h       DATPTR_   EQU  14h
PCL       EQU  02h       RA        EQU  05h       RATRIS_   EQU  15h
PC        EQU  02h       PORTB     EQU  06h       RBTRIS_   EQU  16h
STATUS    EQU  03h       RB        EQU  06h       :
 PA2       EQU  7         ERG       EQU  08h         :16C55
 PA1       EQU  6         ARG0      EQU  08h         :16C57
PA0       EQU  5         ARG1      EQU  0Ah       RC        EQU  07h
TO        EQU  4         ARG2      EQU  0Ch       RCTRIS_   EQU  17h
PD        EQU  3         ARG3      EQU  0Eh       :
Z         EQU  2         ARG4      EQU  10h
DC        EQU  1         ARG5      EQU  12h
;--------------------------------------------------------------------------
:16C554
:16C556
 :16C558
INDIRECT  EQU  00h       PCLATH    EQU  0Ah       TRISB     EQU  86H
TMR0      EQU  01h       INTCON    EQU  0Bh       PCON      EQU  8Eh
RTCC      EQU  01h       GIE       EQU  0Bh,7     POR       EQU  8Eh,1
PCL       EQU  02h       T0IE      EQU  0Bh,5     PUPOPW    EQU  20H
PC        EQU  02h       INTE      EQU  0Bh,4     PUPOPS    EQU  21H
STATUS    EQU  03h       RBIE      EQU  0Bh,3     PUPOPP    EQU  22H
IRP       EQU  7         T0IF      EQU  0Bh,2     PUPOPF    EQU  23H
RP1       EQU  6         INTF      EQU  0Bh,1     ERG       EQU  24h
RP0       EQU  5         RBIF      EQU  0Bh,0     ARG0      EQU  24h
TO        EQU  4         OPTION_R  EQU  81h       ARG1      EQU  26h
PD        EQU  3         RBUP      EQU  81H,7     ARG2      EQU  28h
Z         EQU  2         INTEDG    EQU  81H,6     ARG3      EQU  2Ah
DC        EQU  1         T0CS      EQU  81H,5     ARG4      EQU  2Ch
C         EQU  0         T0SE      EQU  81H,4     ARG5      EQU  2Eh
FSR       EQU  04H       PSA       EQU  81H,3     ERRVAR    EQU  2Fh
PORTA     EQU  05H       PS2       EQU  81H,2     ERR_      EQU  2Fh
RA        EQU  05H       PS1       EQU  81H,1     DATPTR_   EQU  30h
PORTB     EQU  06h       PS0       EQU  81H,0     :
RB        EQU  06h       TRISA     EQU  85H
;--------------------------------------------------------------------------
 :16C61
INDIRECT  EQU  00h       PCLATH    EQU  0Ah       TRISA     EQU  85H
TMR0      EQU  01h       INTCON    EQU  0Bh       TRISB     EQU  86H
RTCC      EQU  01h       GIE       EQU  0Bh,7     PUPOPW    EQU  0CH
PCL       EQU  02h       PEIE      EQU  0Bh,6     PUPOPS    EQU  0DH
PC        EQU  02h       T0IE      EQU  0Bh,5     PUPOPP    EQU  0EH
STATUS    EQU  03h       INTE      EQU  0Bh,4     PUPOPF    EQU  0Fh
IRP       EQU  7         RBIE      EQU  0Bh,3     ERG       EQU  10h
RP1       EQU  6         T0IF      EQU  0Bh,2     ARG0      EQU  10h
```

DEFAULT.EQU (cont.)

```
RP0 EQU 5 INTF EQU 0Bh,1 ARG1 EQU 12h
TO        EQU  4         RBIF      EQU  0Bh,0     ARG2      EQU  14h
PD        EQU  3         OPTION_R  EQU  81h       ARG3      EQU  16h
Z         EQU  2         RBUP      EQU  81H,7     ARG4      EQU  18h
DC        EQU  1         INTEDG    EQU  81H,6     ARG5      EQU  1Ah
C         EQU  0         T0CS      EQU  81H,5     ERRVAR    EQU  1Bh
FSR       EQU  04H       T0SE      EQU  81H,4     ERR_      EQU  1Bh
PORTA     EQU  05H       PSA       EQU  81H,3     TIMERX    EQU  1CH
RA        EQU  05H       PS2       EQU  81H,2     DATPTR_   EQU  1Dh
PORTB     EQU  06H       PS1       EQU  81H,1     :
RB        EQU  06H       PS0       EQU  81H,0
;--------------------------------------------------------------------------
:16C62
:16C62A
:16C64
 :16C64A
INDIRECT  EQU  00H       T1SYNC    EQU  10H,2     RCIE      EQU  8CH,5
TMR0      EQU  01H       TMR1CS    EQU  10H,1     TXIE      EQU  8CH,4
RTCC      EQU  01H       TMR1ON    EQU  10H,0     SSPIE     EQU  8CH,3
PCL       EQU  02H       TMR2      EQU  11H       CCP1IE    EQU  8CH,2
PC        EQU  02h       T2CON     EQU  12H       TMR2IE    EQU  8CH,1
STATUS    EQU  03H       TOUTPS3   EQU  12H,6     TMR1IE    EQU  8CH,0
IRP       EQU  7         TOUTPS2   EQU  12H,5     PCON      EQU  8EH
RP1       EQU  6         TOUTPS1   EQU  12H,4     POR       EQU  8EH,1
RP0       EQU  5         TOUTPS0   EQU  12H,3     BOR       EQU  8EH,0
TO        EQU  4         TMR2ON    EQU  12H,2     PR2       EQU  92H
PD        EQU  3         T2CKPS1   EQU  12H,1     SSPADD    EQU  93H
Z         EQU  2         T2CKPS0   EQU  12H,0     SSPSTAT   EQU  94H
DC        EQU  1         SSPBUF    EQU  13H       SMP       EQU  94H,7
C         EQU  0         SSPCON    EQU  14H       CKE       EQU  94H,6
FSR       EQU  04H       WCOL      EQU  14H,7     D_A       EQU  94H,5
PORTA     EQU  05H       SSPOV     EQU  14H,6     P         EQU  94H,4
RA        EQU  05H       SSPEN     EQU  14H,5     S         EQU  94H,3
PORTB     EQU  06H       CKP       EQU  14H,4     R_W       EQU  94H,2
RB        EQU  06H       SSPM3     EQU  14H,3     UA        EQU  94H,1
PORTC     EQU  07H       SSPM2     EQU  14H,2     BF        EQU  94H,0
RC        EQU  07H       SSPM1     EQU  14H,1     PUPOPW    EQU  20H
PCLATH    EQU  0AH       SSPM0     EQU  14H,0     PUPOPS    EQU  21H
INTCON    EQU  0BH       CCPR1L    EQU  15H       PUPOPP    EQU  22H
GIE       EQU  0BH,7     CCPR1H    EQU  16H       PUPOPF    EQU  23h
PEIE      EQU  0BH,6     CCP1CON   EQU  17H       ERG       EQU  24h
T0IE      EQU  0BH,5     CCP1X     EQU  17H,5     ARG0      EQU  24h
INTE      EQU  0BH,4     CCP1Y     EQU  17H,4     ARG1      EQU  26h
RBIE      EQU  0BH,3     CCP1M3    EQU  17H,3     ARG2      EQU  28h
T0IF      EQU  0BH,2     CCP1M2    EQU  17H,2     ARG3      EQU  2Ah
INTF      EQU  0BH,1     CCP1M1    EQU  17H,1     ARG4      EQU  2Ch
RBIF      EQU  0BH,0     CCP1M0    EQU  17H,0     ARG5      EQU  2Eh
PIR1      EQU  0CH       OPTION_R  EQU  81H       ERRVAR    EQU  2Fh
PSPIF     EQU  0CH,7     RBUP      EQU  81H,7     ERR_      EQU  2Fh
ADIF      EQU  0CH,6     INTEDG    EQU  81H,6     TIMERX    EQU  30H
RCIF      EQU  0CH,5     T0CS      EQU  81H,5     DATPTR_   EQU  31h
```

DEFAULT.EQU (cont.)

```
TXIF EQU 0CH,4 T0SE EQU 81H,4 :
 SSPIF     EQU  0CH,3     PSA       EQU  81H,3      :16C64
 CCP1IF    EQU  0CH,2     PS2       EQU  81H,2      :16C64A
TMR2IF    EQU  0CH,1     PS1       EQU  81H,1     PORTD     EQU  08H
TMR1IF    EQU  0CH,0     PS0       EQU  81H,0     RD        EQU  08H
TMR1L     EQU  0EH       TRISA     EQU  85H       PORTE     EQU  09H
TMR1H     EQU  0FH       TRISB     EQU  86H       RE        EQU  09H
T1CON     EQU  10H       TRISC     EQU  87H       TRISD     EQU  88H
T1CKPS1   EQU  10H,5     PIE1      EQU  8CH       TRISE     EQU  89H
T1CKPS0   EQU  10H,4     PSPIE     EQU  8CH,7     :
T1OSCEN   EQU  10H,3     ADIE      EQU  8CH,6
;--------------------------------------------------------------------------
:16C63
:16C65
:16C65A
:16C66
:16C66A
:16C67
 :16C67A
INDIRECT  EQU  00H       TMR2ON    EQU  12H,2     SSPIE     EQU  8CH,3
TMR0      EQU  01H       T2CKPS1   EQU  12H,1     CCP1IE    EQU  8CH,2
RTCC      EQU  01H       T2CKPS0   EQU  12H,0     TMR2IE    EQU  8CH,1
PCL       EQU  02H       SSPBUF    EQU  13H       TMR1IE    EQU  8CH,0
PC        EQU  02h       SSPCON    EQU  14H       PIE2      EQU  8DH
STATUS    EQU  03H       WCOL      EQU  14H,7     EEIE      EQU  8DH,4
IRP       EQU  7         SSPOV     EQU  14H,6     BCLIE     EQU  8DH,3
RP1       EQU  6         SSPEN     EQU  14H,5     CCP2IE    EQU  8DH,0
RP0       EQU  5         CKP       EQU  14H,4     PCON      EQU  8EH
TO        EQU  4         SSPM3     EQU  14H,3     POR       EQU  8EH,1
PD        EQU  3         SSPM2     EQU  14H,2     BOR       EQU  8EH,0
Z         EQU  2         SSPM1     EQU  14H,1     PR2       EQU  92H
DC        EQU  1         SSPM0     EQU  14H,0     SSPADD    EQU  93H
C         EQU  0         CCPR1L    EQU  15H       SSPSTAT   EQU  94H
FSR       EQU  04H       CCPR1H    EQU  16H       SMP       EQU  94H,7
PORTA     EQU  05H       CCP1CON   EQU  17H       CKE       EQU  94H,6
RA        EQU  05H       CCP1X     EQU  17H,5     D_A       EQU  94H,5
PORTB     EQU  06H       CCP1Y     EQU  17H,4     P         EQU  94H,4
RB        EQU  06H       CCP1M3    EQU  17H,3     S         EQU  94H,3
PORTC     EQU  07H       CCP1M2    EQU  17H,2     R_W       EQU  94H,2
RC        EQU  07H       CCP1M1    EQU  17H,1     UA        EQU  94H,1
PCLATH    EQU  0AH       CCP1M0    EQU  17H,0     BF        EQU  94H,0
INTCON    EQU  0BH       RCSTA     EQU  18H       TXSTA     EQU  98H
GIE       EQU  0BH,7     SPEN      EQU  18H,7     CSRC      EQU  98H,7
PEIE      EQU  0BH,6     RX9       EQU  18H,6     TX9       EQU  98H,6
T0IE      EQU  0BH,5     SREN      EQU  18H,5     TXEN      EQU  98H,5
INTE      EQU  0BH,4     CREN      EQU  18H,4     SYNC      EQU  98H,4
RBIE      EQU  0BH,3     ADDEN     EQU  18H,3     BRGH      EQU  98H,2
T0IF      EQU  0BH,2     FERR      EQU  18H,2     TRMT      EQU  98H,1
INTF      EQU  0BH,1     OERR      EQU  18H,1     TX9D      EQU  98H,0
RBIF      EQU  0BH,0     RX9D      EQU  18H,0     SPBRG     EQU  99H
PIR1      EQU  0CH       TXREG     EQU  19H       PUPOPW    EQU  20H
```

DEFAULT.EQU (cont.)

```
PSPIF EQU 0CH,7 RCREG EQU 1AH PUPOPS EQU 21H
ADIF      EQU  0CH,6     CCPR2L    EQU  1BH       PUPOPP    EQU  22H
RCIF      EQU  0CH,5     CCPR2H    EQU  1CH       PUPOPF    EQU  23h
TXIF      EQU  0CH,4     CCP2CON   EQU  1DH       ERG       EQU  24h
SSPIF     EQU  0CH,3     CCP2X     EQU  1DH,5     ARG0      EQU  24h
CCP1IF    EQU  0CH,2     CCP2Y     EQU  1DH,4     ARG1      EQU  26h
TMR2IF    EQU  0CH,1     CCP2M3    EQU  1DH,3     ARG2      EQU  28h
TMR1IF    EQU  0CH,0     CCP2M2    EQU  1DH,2     ARG3      EQU  2Ah
PIR2      EQU  0DH       CCP2M1    EQU  1DH,1     ARG4      EQU  2Ch
EEIF      EQU  0DH,4     CCP2M0    EQU  1DH,0     ARG5      EQU  2Eh
BCLIF     EQU  0DH,3     OPTION_R  EQU  81H       ERRVAR    EQU  2Fh
CCP2IF    EQU  0DH,0     RBUP      EQU  81H,7     ERR_      EQU  2Fh
TMR1L     EQU  0EH       INTEDG    EQU  81H,6     TIMERX    EQU  30H
TMR1H     EQU  0FH       T0CS      EQU  81H,5     DATPTR_   EQU  31h
T1CON     EQU  10H       T0SE      EQU  81H,4     :
 T1CKPS1   EQU  10H,5     PSA       EQU  81H,3     :16C65
 T1CKPS0   EQU  10H,4     PS2       EQU  81H,2     :16C65A
 T1OSCEN   EQU  10H,3     PS1       EQU  81H,1     :16C67
 T1SYNC    EQU  10H,2     PS0       EQU  81H,0     :16C67A
TMR1CS    EQU  10H,1     TRISA     EQU  85H       PORTD     EQU  08H
TMR1ON    EQU  10H,0     TRISB     EQU  86H       RD        EQU  08H
TMR2      EQU  11H       TRISC     EQU  87H       PORTE     EQU  09H
T2CON     EQU  12H       PIE1      EQU  8CH       RE        EQU  09H
TOUTPS3   EQU  12H,6     PSPIE     EQU  8CH,7     TRISD     EQU  88H
TOUTPS2   EQU  12H,5     ADIE      EQU  8CH,6     TRISE     EQU  89H
TOUTPS1   EQU  12H,4     RCIE      EQU  8CH,5     :
TOUTPS0   EQU  12H,3     TXIE      EQU  8CH,4
;--------------------------------------------------------------------------
:16C620
:16C621
 :16C622
INDIRECT  EQU  00h       INTF      EQU  0Bh,1     VRCON     EQU  9FH
TMR0      EQU  01h       RBIF      EQU  0Bh,0     VREN      EQU  9FH,7
RTCC      EQU  01h       PIR1      EQU  0Ch       VROE      EQU  9FH,6
PCL       EQU  02h       CMIF      EQU  0Ch,6     VRR       EQU  9FH,5
PC        EQU  02h       CMCON     EQU  1FH       VR3       EQU  9FH,3
STATUS    EQU  03h       C2OUT     EQU  1FH,7     VR2       EQU  9FH,2
IRP       EQU  7         C1OUT     EQU  1FH,6     VR1       EQU  9FH,1
RP1       EQU  6         CIS       EQU  1FH,3     VR0       EQU  9FH,0
RP0       EQU  5         CM2       EQU  1FH,2     PUPOPW    EQU  20H
TO        EQU  4         CM1       EQU  1FH,1     PUPOPS    EQU  21H
PD        EQU  3         CM0       EQU  1FH,0     PUPOPP    EQU  22H
Z         EQU  2         OPTION_R  EQU  81h       PUPOPF    EQU  23h
DC        EQU  1         RBUP      EQU  81H,7     ERG       EQU  24h
C         EQU  0         INTEDG    EQU  81H,6     ARG0      EQU  24h
FSR       EQU  04H       T0CS      EQU  81H,5     ARG1      EQU  26h
PORTA     EQU  05H       T0SE      EQU  81H,4     ARG2      EQU  28h
RA        EQU  05H       PSA       EQU  81H,3     ARG3      EQU  2Ah
PORTB     EQU  06H       PS2       EQU  81H,2     ARG4      EQU  2Ch
RB        EQU  06H       PS1       EQU  81H,1     ARG5      EQU  2Eh
PCLATH    EQU  0Ah       PS0       EQU  81H,0     ERRVAR    EQU  2Fh
```

DEFAULT.EQU (cont.)

```
INTCON EQU 0Bh TRISA EQU 85H ERR_ EQU 2Fh
GIE       EQU  0Bh,7     TRISB     EQU  86H       TIMERX    EQU  30H
PEIE      EQU  0Bh,6     PIE1      EQU  8CH       DATPTR_   EQU  31h
T0IE      EQU  0Bh,5     CMIE      EQU  8CH,6     :
INTE      EQU  0Bh,4     PCON      EQU  8EH
RBIE      EQU  0Bh,3     POR       EQU  8EH,1
T0IF      EQU  0Bh,2     BO        EQU  8EH,0
;--------------------------------------------------------------------------
:16E623
:16E624
 :16E625
INDIRECT  EQU  00h       RBIF      EQU  0Bh,0     EESDA     EQU  90H,1
TMR0      EQU  01h       PIR1      EQU  0Ch       EEVDD     EQU  90H,0
RTCC      EQU  01h       CMIF      EQU  0Ch,6     VRCON     EQU  9FH
PCL       EQU  02h       CMCON     EQU  1FH       VREN      EQU  9FH,7
PC        EQU  02h       C2OUT     EQU  1FH,7     VROE      EQU  9FH,6
STATUS    EQU  03h       C1OUT     EQU  1FH,6     VRR       EQU  9FH,5
IRP       EQU  7         CIS       EQU  1FH,3     VR3       EQU  9FH,3
RP1       EQU  6         CM2       EQU  1FH,2     VR2       EQU  9FH,2
RP0       EQU  5         CM1       EQU  1FH,1     VR1       EQU  9FH,1
TO        EQU  4         CM0       EQU  1FH,0     VR0       EQU  9FH,0
PD        EQU  3         OPTION_R  EQU  81h       PUPOPW    EQU  70H
Z         EQU  2         RBUP      EQU  81H,7     PUPOPS    EQU  71H
DC        EQU  1         INTEDG    EQU  81H,6     PUPOPP    EQU  72H
C         EQU  0         T0CS      EQU  81H,5     PUPOPF    EQU  73h
FSR       EQU  04H       T0SE      EQU  81H,4     ERG       EQU  74h
PORTA     EQU  05H       PSA       EQU  81H,3     ARG0      EQU  74h
RA        EQU  05H       PS2       EQU  81H,2     ARG1      EQU  76h
PORTB     EQU  06H       PS1       EQU  81H,1     ARG2      EQU  78h
RB        EQU  06H       PS0       EQU  81H,0     ARG3      EQU  7Ah
PCLATH    EQU  0Ah       TRISA     EQU  85H       ARG4      EQU  7Ch
INTCON    EQU  0Bh       TRISB     EQU  86H       ARG5      EQU  7Eh
GIE       EQU  0Bh,7     PIE1      EQU  8CH       ERRVAR    EQU  7Fh
PEIE      EQU  0Bh,6     CMIE      EQU  8CH,6     ERR_      EQU  7Fh
T0IE      EQU  0Bh,5     PCON      EQU  8EH       TIMERX    EQU  20H
INTE      EQU  0Bh,4     POR       EQU  8EH,1     DATPTR_   EQU  21h
RBIE      EQU  0Bh,3     BO        EQU  8EH,0     :
T0IF      EQU  0Bh,2     EEINTF    EQU  90H
INTF      EQU  0Bh,1     EESCL     EQU  90H,2
;--------------------------------------------------------------------------
:16F627
 :16F628
INDIRECT  EQU  00h       CCP1Y     EQU  17h,4     POR       EQU  8EH,1
TMR0      EQU  01h       CCP1M3    EQU  17h,3     BO        EQU  8EH,0
RTCC      EQU  01h       CCP1M2    EQU  17h,2     PR2       EQU  92h
PCL       EQU  02h       CCP1M1    EQU  17h,1     TXSTA     EQU  98h
PC        EQU  02h       CCP1M0    EQU  17h,0     CSRC      EQU  98h,7
STATUS    EQU  03h       RCSTA     EQU  18h       TX9       EQU  98h,6
IRP       EQU  7         SPEN      EQU  18h,7     TXEN      EQU  98h,5
RP1       EQU  6         RX9       EQU  18h,6     SYNC      EQU  98h,4
RP0       EQU  5         SREN      EQU  18h,5     BRGH      EQU  98h,2
```

DEFAULT.EQU (cont.)

```
TO EQU 4 CREN EQU 18h,4 TRMT EQU 98h,1
PD        EQU  3         ADDEN     EQU  18h,3     TX9D      EQU  98h,0
Z         EQU  2         FERR      EQU  18h,2     SPBRG     EQU  99h
DC        EQU  1         OERR      EQU  18h,1     EEDTA     EQU  9Ah
C         EQU  0         RX9D      EQU  18h,0     EEADR     EQU  9Bh
FSR       EQU  04H       TXREG     EQU  19h       EECON1    EQU  9Ch
PORTA     EQU  05H       RCREG     EQU  1Ah       WRERR     EQU  9Ch,3
RA        EQU  05H       CMCON     EQU  1FH       WREN      EQU  9Ch,2
PORTB     EQU  06H       C2OUT     EQU  1FH,7     WR_       EQU  9Ch,1
RB        EQU  06H       C1OUT     EQU  1FH,6     RD_       EQU  9Ch,0
PCLATH    EQU  0Ah       C2INV     EQU  1Fh,5     EECON2    EQU  9Dh
INTCON    EQU  0Bh       C1INV     EQU  1Fh,4     VRCON     EQU  9FH
GIE       EQU  0Bh,7     CIS       EQU  1FH,3     VREN      EQU  9FH,7
PEIE      EQU  0Bh,6     CM2       EQU  1FH,2     VROE      EQU  9FH,6
T0IE      EQU  0Bh,5     CM1       EQU  1FH,1     VRR       EQU  9FH,5
INTE      EQU  0Bh,4     CM0       EQU  1FH,0     VR3       EQU  9FH,3
RBIE      EQU  0Bh,3     OPTION_R  EQU  81h       VR2       EQU  9FH,2
T0IF      EQU  0Bh,2     RBUP      EQU  81H,7     VR1       EQU  9FH,1
INTF      EQU  0Bh,1     INTEDG    EQU  81H,6     VR0       EQU  9FH,0
RBIF      EQU  0Bh,0     T0CS      EQU  81H,5     PUPOPS    EQU  6DH
PIR1      EQU  0Ch       T0SE      EQU  81H,4     PUPOPP    EQU  6EH
EEIF      EQU  0Ch,7     PSA       EQU  81H,3     PUPOPF    EQU  6Fh
CMIF      EQU  0Ch,6     PS2       EQU  81H,2     ERG       EQU  70h
RCIF      EQU  0Ch,5     PS1       EQU  81H,1     ARG0      EQU  70h
TXIF      EQU  0Ch,4     PS0       EQU  81H,0     ARG1      EQU  72h
CCP1IF    EQU  0Ch,2     TRISA     EQU  85H       ARG2      EQU  74h
TMR2IF    EQU  0Ch,1     TRISB     EQU  86H       ARG3      EQU  76h
TMR1IF    EQU  0Ch,0     PIE1      EQU  8CH       ARG4      EQU  78h
TMR1L     EQU  0Eh       EEIE      EQU  8Ch,7     ARG5      EQU  7Ah
TMR1H     EQU  0Fh       CMIE      EQU  8CH,6     ERRVAR    EQU  7Bh
T1CON     EQU  10h       RCIE      EQU  8Ch,5     ERR_      EQU  7Bh
TMR2      EQU  11h       TXIE      EQU  8Ch,4     TIMERX    EQU  7CH
T2CON     EQU  12h       CCP1IE    EQU  8Ch,2     DATPTR_   EQU  7Dh
CCPR1L    EQU  15h       TMR2IE    EQU  8Ch,1     PUPOPW    EQU  7FH
CCPR1H    EQU  16h       TMR1IE    EQU  8Ch,0     :
CCP1CON   EQU  17h       PCON      EQU  8EH
CCP1X     EQU  17h,5     OSCF      EQU  8Eh,3
;--------------------------------------------------------------------------
:16C71
:16C710
 :16C711
INDIRECT  EQU  00H       CHS0      EQU  08h,3     PS0       EQU  81H,0
TMR0      EQU  01H       GO_DONE   EQU  08h,2     TRISA     EQU  85H
RTCC      EQU  01H       ADIF      EQU  08h,1     TRISB     EQU  86H
PCL       EQU  02H       ADON      EQU  08h,0     ADCON1    EQU  88H
PC        EQU  02h       ADRES     EQU  09h       PCFG1     EQU  88H,1
STATUS    EQU  03H       PCLATH    EQU  0Ah       PCFG0     EQU  88H,0
IRP       EQU  7         INTCON    EQU  0BH       PUPOPW    EQU  0CH
RP1       EQU  6         GIE       EQU  0BH,7     PUPOPS    EQU  0DH
RP0       EQU  5         ADIE      EQU  0BH,6     PUPOPP    EQU  0EH
TO        EQU  4         T0IE      EQU  0BH,5     PUPOPF    EQU  0FH
```

DEFAULT.EQU (cont.)

```
PD EQU 3 INTE EQU 0BH,4 ERG EQU 10h
Z         EQU  2         RBIE      EQU  0BH,3     ARG0      EQU  10h
DC        EQU  1         T0IF      EQU  0BH,2     ARG1      EQU  12h
C         EQU  0         INTF      EQU  0BH,1     ARG2      EQU  14h
FSR       EQU  04H       RBIF      EQU  0BH,0     ARG3      EQU  16h
PORTA     EQU  05H       OPTION_R  EQU  81H       ARG4      EQU  18h
RA        EQU  05H       RBUP      EQU  81H,7     ARG5      EQU  1Ah
PORTB     EQU  06H       INTEDG    EQU  81H,6     ERRVAR    EQU  1Bh
RB        EQU  06H       T0CS      EQU  81H,5     ERR_      EQU  1Bh
ADCON0    EQU  08h       T0SE      EQU  81H,4     TIMERX    EQU  1CH
ADCS1     EQU  08h,7     PSA       EQU  81H,3     DATPTR_   EQU  1Dh
ADCS0     EQU  08h,6     PS2       EQU  81H,2     :
CHS1      EQU  08h,4     PS1       EQU  81H,1
;--------------------------------------------------------------------------
 :16C715
INDIRECT  EQU  00H       INTE      EQU  0BH,4     TRISA     EQU  85H
TMR0      EQU  01H       RBIE      EQU  0BH,3     TRISB     EQU  86H
RTCC      EQU  01H       T0IF      EQU  0BH,2     ADCON1    EQU  9FH
PCL       EQU  02H       INTF      EQU  0BH,1     PCFG1     EQU  9FH,1
PC        EQU  02h       RBIF      EQU  0BH,0     PCFG0     EQU  9FH,0
STATUS    EQU  03H       PIR1      EQU  0CH       PUPOPS    EQU  6DH
IRP       EQU  7         ADIF      EQU  0CH,6     PUPOPP    EQU  6EH
RP1       EQU  6         ADCON0    EQU  1Fh       PUPOPF    EQU  6Fh
RP0       EQU  5         ADCS1     EQU  1Fh,7     ERG       EQU  70h
TO        EQU  4         ADCS0     EQU  1Fh,6     ARG0      EQU  70h
PD        EQU  3         CHS1      EQU  1Fh,4     ARG1      EQU  72h
Z         EQU  2         CHS0      EQU  1Fh,3     ARG2      EQU  74h
DC        EQU  1         GO_DONE   EQU  1Fh,2     ARG3      EQU  76h
C         EQU  0         ADON      EQU  1Fh,0     ARG4      EQU  78h
FSR       EQU  04H       ADRES     EQU  1Eh       ARG5      EQU  7Ah
PORTA     EQU  05H       OPTION_R  EQU  81H       ERRVAR    EQU  7Bh
RA        EQU  05H       RBUP      EQU  81H,7     ERR_      EQU  7Bh
PORTB     EQU  06H       INTEDG    EQU  81H,6     TIMERX    EQU  7CH
RB        EQU  06H       T0CS      EQU  81H,5     DATPTR_   EQU  7Dh
PCLATH    EQU  0Ah       T0SE      EQU  81H,4     PUPOPW    EQU  7FH
INTCON    EQU  0BH       PSA       EQU  81H,3     :
GIE       EQU  0BH,7     PS2       EQU  81H,2
ADIE      EQU  0BH,6     PS1       EQU  81H,1
T0IE      EQU  0BH,5     PS0       EQU  81H,0
;--------------------------------------------------------------------------
 :16C72
INDIRECT  EQU  00H       TMR2      EQU  11H       TRISA     EQU  85H
TMR0      EQU  01H       T2CON     EQU  12H       TRISB     EQU  86H
RTCC      EQU  01H       TOUTPS3   EQU  12H,6     TRISC     EQU  87H
PCL       EQU  02H       TOUTPS2   EQU  12H,5     PIE1      EQU  8CH
PC        EQU  02h       TOUTPS1   EQU  12H,4     ADIE      EQU  8CH,6
STATUS    EQU  03H       TOUTPS0   EQU  12H,3     SSPIE     EQU  8CH,3
IRP       EQU  7         TMR2ON    EQU  12H,2     CCP1IE    EQU  8CH,2
RP1       EQU  6         T2CKPS1   EQU  12H,1     TMR2IE    EQU  8CH,1
RP0       EQU  5         T2CKPS0   EQU  12H,0     TMR1IE    EQU  8CH,0
TO        EQU  4         SSPBUF    EQU  13H       PCON      EQU  8EH
```

DEFAULT.EQU (cont.)

```
PD EQU 3 SSPCON EQU 14H POR EQU 8EH,1
Z         EQU  2         WCOL      EQU  14H,7     BOR       EQU  8EH,0
DC        EQU  1         SSPOV     EQU  14H,6     PR2       EQU  92H
C         EQU  0         SSPEN     EQU  14H,5     SSPADD    EQU  93H
FSR       EQU  04H       CKP       EQU  14H,4     SSPSTAT   EQU  94H
PORTA     EQU  05H       SSPM3     EQU  14H,3     SMP       EQU  94H,7
RA        EQU  05H       SSPM2     EQU  14H,2     CKE       EQU  94H,6
PORTB     EQU  06H       SSPM1     EQU  14H,1     D_A       EQU  94H,5
RB        EQU  06H       SSPM0     EQU  14H,0     P         EQU  94H,4
PORTC     EQU  07H       CCPR1L    EQU  15H       S         EQU  94H,3
RC        EQU  07H       CCPR1H    EQU  16H       R_W       EQU  94H,2
PCLATH    EQU  0AH       CCP1CON   EQU  17H       UA        EQU  94H,1
INTCON    EQU  0BH       CCP1X     EQU  17H,5     BF        EQU  94H,0
GIE       EQU  0BH,7     CCP1Y     EQU  17H,4     ADCON1    EQU  9FH
PEIE      EQU  0BH,6     CCP1M3    EQU  17H,3     ADFM      EQU  9FH,7
T0IE      EQU  0BH,5     CCP1M2    EQU  17H,2     PCFG3     EQU  9FH,3
INTE      EQU  0BH,4     CCP1M1    EQU  17H,1     PCFG2     EQU  9FH,2
RBIE      EQU  0BH,3     CCP1M0    EQU  17H,0     PCFG1     EQU  9FH,1
T0IF      EQU  0BH,2     ADRES     EQU  1EH       PCFG0     EQU  9FH,0
INTF      EQU  0BH,1     ADCON0    EQU  1FH       PUPOPW    EQU  20H
RBIF      EQU  0BH,0     ADCS1     EQU  1FH,7     PUPOPS    EQU  21H
PIR1      EQU  0CH       ADCS0     EQU  1FH,6     PUPOPP    EQU  22H
ADIF      EQU  0CH,6     CHS2      EQU  1FH,5     PUPOPF    EQU  23h
SSPIF     EQU  0CH,3     CHS1      EQU  1FH,4     ERG       EQU  24h
CCP1IF    EQU  0CH,2     CHS0      EQU  1FH,3     ARG0      EQU  24h
TMR2IF    EQU  0CH,1     GO_DONE   EQU  1FH,2     ARG1      EQU  26h
TMR1IF    EQU  0CH,0     ADON      EQU  1FH,0     ARG2      EQU  28h
TMR1L     EQU  0EH       OPTION_R  EQU  81H       ARG3      EQU  2Ah
TMR1H     EQU  0FH       RBUP      EQU  81H,7     ARG4      EQU  2Ch
T1CON     EQU  10H       INTEDG    EQU  81H,6     ARG5      EQU  2Eh
T1CKPS1   EQU  10H,5     T0CS      EQU  81H,5     ERRVAR    EQU  2Fh
T1CKPS0   EQU  10H,4     T0SE      EQU  81H,4     ERR_      EQU  2Fh
T1OSCEN   EQU  10H,3     PSA       EQU  81H,3     TIMERX    EQU  30H
T1SYNC    EQU  10H,2     PS2       EQU  81H,2     DATPTR_   EQU  31h
TMR1CS    EQU  10H,1     PS1       EQU  81H,1     :
TMR1ON    EQU  10H,0     PS0       EQU  81H,0
;--------------------------------------------------------------------------
:16C73
:16C73A
:16C74
 :16C74A
INDIRECT  EQU  00H       TOUTPS0   EQU  12H,3     TRISA     EQU  85H
TMR0      EQU  01H       TMR2ON    EQU  12H,2     TRISB     EQU  86H
RTCC      EQU  01H       T2CKPS1   EQU  12H,1     TRISC     EQU  87H
PCL       EQU  02H       T2CKPS0   EQU  12H,0     TRISD     EQU  88H
PC        EQU  02H       SSPBUF    EQU  13H       TRISE     EQU  89H
STATUS    EQU  03H       SSPCON    EQU  14H       PIE1      EQU  8CH
IRP       EQU  7         WCOL      EQU  14H,7     PSPIE     EQU  8CH,7
RP1       EQU  6         SSPOV     EQU  14H,6     ADIE      EQU  8CH,6
RP0       EQU  5         SSPEN     EQU  14H,5     RCIE      EQU  8CH,5
TO        EQU  4         CKP       EQU  14H,4     TXIE      EQU  8CH,4
```

DEFAULT.EQU (cont.)

```
PD EQU 3 SSPM3 EQU 14H,3 SSPIE EQU 8CH,3
Z         EQU  2         SSPM2     EQU  14H,2     CCP1IE    EQU  8CH,2
DC        EQU  1         SSPM1     EQU  14H,1     TMR2IE    EQU  8CH,1
C         EQU  0         SSPM0     EQU  14H,0     TMR1IE    EQU  8CH,0
FSR       EQU  04H       CCPR1L    EQU  15H       PIE2      EQU  8DH
PORTA     EQU  05H       CCPR1H    EQU  16H       CCP2IE    EQU  8DH,0
RA        EQU  05H       CCP1CON   EQU  17H       PCON      EQU  8EH
PORTB     EQU  06H       CCP1X     EQU  17H,5     POR       EQU  8EH,1
RB        EQU  06H       CCP1Y     EQU  17H,4     BOR       EQU  8EH,0
PORTC     EQU  07H       CCP1M3    EQU  17H,3     PR2       EQU  92H
RC        EQU  07H       CCP1M2    EQU  17H,2     SSPADD    EQU  93H
PORTD     EQU  08H       CCP1M1    EQU  17H,1     SSPSTAT   EQU  94H
RD        EQU  08H       CCP1M0    EQU  17H,0     D_A       EQU  94H,5
PORTE     EQU  09H       RCSTA     EQU  18H       P         EQU  94H,4
RE        EQU  09H       SPEN      EQU  18H,7     S         EQU  94H,3
PCLATH    EQU  0AH       RX9       EQU  18H,6     R_W       EQU  94H,2
INTCON    EQU  0BH       SREN      EQU  18H,5     UA        EQU  94H,1
GIE       EQU  0BH,7     CREN      EQU  18H,4     BF        EQU  94H,0
PEIE      EQU  0BH,6     FERR      EQU  18H,2     TXSTA     EQU  98H
T0IE      EQU  0BH,5     OERR      EQU  18H,1     CSRC      EQU  98H,7
INTE      EQU  0BH,4     RX9D      EQU  18H,0     TX9       EQU  98H,6
RBIE      EQU  0BH,3     TXREG     EQU  19H       TXEN      EQU  98H,5
T0IF      EQU  0BH,2     RCREG     EQU  1AH       SYNC      EQU  98H,4
INTF      EQU  0BH,1     CCPR2L    EQU  1BH       BRGH      EQU  98H,2
RBIF      EQU  0BH,0     CCPR2H    EQU  1CH       TRMT      EQU  98H,1
PIR1      EQU  0CH       CCP2CON   EQU  1DH       TX9D      EQU  98H,0
PSPIF     EQU  0CH,7     CCP2X     EQU  1DH,5     SPBRG     EQU  99H
ADIF      EQU  0CH,6     CCP2Y     EQU  1DH,4     ADCON1    EQU  9FH
RCIF      EQU  0CH,5     CCP2M3    EQU  1DH,3     PCFG2     EQU  9FH,2
TXIF      EQU  0CH,4     CCP2M2    EQU  1DH,2     PCFG1     EQU  9FH,1
SSPIF     EQU  0CH,3     CCP2M1    EQU  1DH,1     PCFG0     EQU  9FH,0
CCP1IF    EQU  0CH,2     CCP2M0    EQU  1DH,0     PUPOPW    EQU  20H
TMR2IF    EQU  0CH,1     ADRES     EQU  1EH       PUPOPS    EQU  21H
TMR1IF    EQU  0CH,0     ADCON0    EQU  1FH       PUPOPP    EQU  22H
PIR2      EQU  0DH       ADCS1     EQU  1FH,7     PUPOPF    EQU  23h
CCP2IF    EQU  0DH,0     ADCS0     EQU  1FH,6     ERG       EQU  24h
TMR1L     EQU  0EH       CHS2      EQU  1FH,5     ARG0      EQU  24h
TMR1H     EQU  0FH       CHS1      EQU  1FH,4     ARG1      EQU  26h
T1CON     EQU  10H       CHS0      EQU  1FH,3     ARG2      EQU  28h
T1CKPS1   EQU  10H,5     GO_DONE   EQU  1FH,2     ARG3      EQU  2Ah
T1CKPS0   EQU  10H,4     ADON      EQU  1FH,0     ARG4      EQU  2Ch
T1OSCEN   EQU  10H,3     OPTION_R  EQU  81H       ARG5      EQU  2Eh
T1SYNC    EQU  10H,2     RBUP      EQU  81H,7     ERRVAR    EQU  2Fh
TMR1CS    EQU  10H,1     INTEDG    EQU  81H,6     ERR_      EQU  2Fh
TMR1ON    EQU  10H,0     T0CS      EQU  81H,5     TIMERX    EQU  30H
TMR2      EQU  11H       T0SE      EQU  81H,4     DATPTR_   EQU  31h
T2CON     EQU  12H       PSA       EQU  81H,3     :
TOUTPS3   EQU  12H,6     PS2       EQU  81H,2
TOUTPS2   EQU  12H,5     PS1       EQU  81H,1
TOUTPS1   EQU  12H,4     PS0       EQU  81H,0
;--------------------------------------------------------------------------
```

DEFAULT.EQU (cont.)

```
:16C76
 :16C77
INDIRECT  EQU  00H      SSPCON    EQU  14H      TXIE      EQU  8CH,4
TMR0      EQU  01H      WCOL      EQU  14H,7    SSPIE     EQU  8CH,3
RTCC      EQU  01H      SSPOV     EQU  14H,6    CCP1IE    EQU  8CH,2
PCL       EQU  02H      SSPEN     EQU  14H,5    TMR2IE    EQU  8CH,1
PC        EQU  02H      CKP       EQU  14H,4    TMR1IE    EQU  8CH,0
STATUS    EQU  03H      SSPM3     EQU  14H,3    PIE2      EQU  8DH
IRP       EQU  7        SSPM2     EQU  14H,2    EEIE      EQU  8DH,4
RP1       EQU  6        SSPM1     EQU  14H,1    BCLIE     EQU  8DH,3
RP0       EQU  5        SSPM0     EQU  14H,0    CCP2IE    EQU  8DH,0
TO        EQU  4        CCPR1L    EQU  15H      PCON      EQU  8EH
PD        EQU  3        CCPR1H    EQU  16H      POR       EQU  8EH,1
Z         EQU  2        CCP1CON   EQU  17H      BOR       EQU  8EH,0
DC        EQU  1        CCP1X     EQU  17H,5    PR2       EQU  92H
C         EQU  0        CCP1Y     EQU  17H,4    SSPADD    EQU  93H
FSR       EQU  04H      CCP1M3    EQU  17H,3    SSPSTAT   EQU  94H
PORTA     EQU  05H      CCP1M2    EQU  17H,2    SMP       EQU  94H,7
RA        EQU  05H      CCP1M1    EQU  17H,1    CKE       EQU  94H,6
PORTB     EQU  06H      CCP1M0    EQU  17H,0    D_A       EQU  94H,5
RB        EQU  06H      RCSTA     EQU  18H      P         EQU  94H,4
PORTC     EQU  07H      SPEN      EQU  18H,7    S         EQU  94H,3
RC        EQU  07H      RX9       EQU  18H,6    R_W       EQU  94H,2
PCLATH    EQU  0AH      SREN      EQU  18H,5    UA        EQU  94H,1
INTCON    EQU  0BH      CREN      EQU  18H,4    BF        EQU  94H,0
GIE       EQU  0BH,7    ADDEN     EQU  18H,3    TXSTA     EQU  98H
PEIE      EQU  0BH,6    FERR      EQU  18H,2    CSRC      EQU  98H,7
T0IE      EQU  0BH,5    OERR      EQU  18H,1    TX9       EQU  98H,6
INTE      EQU  0BH,4    RX9D      EQU  18H,0    TXEN      EQU  98H,5
RBIE      EQU  0BH,3    TXREG     EQU  19H      SYNC      EQU  98H,4
T0IF      EQU  0BH,2    RCREG     EQU  1AH      BRGH      EQU  98H,2
INTF      EQU  0BH,1    CCPR2L    EQU  1BH      TRMT      EQU  98H,1
RBIF      EQU  0BH,0    CCPR2H    EQU  1CH      TX9D      EQU  98H,0
PIR1      EQU  0CH      CCP2CON   EQU  1DH      SPBRG     EQU  99H
PSPIF     EQU  0CH,7    CCP2X     EQU  1DH,5    ADCON1    EQU  9FH
ADIF      EQU  0CH,6    CCP2Y     EQU  1DH,4    ADFM      EQU  9FH,7
RCIF      EQU  0CH,5    CCP2M3    EQU  1DH,3    PCFG3     EQU  9FH,3
TXIF      EQU  0CH,4    CCP2M2    EQU  1DH,2    PCFG2     EQU  9FH,2
SSPIF     EQU  0CH,3    CCP2M1    EQU  1DH,1    PCFG1     EQU  9FH,1
CCP1IF    EQU  0CH,2    CCP2M0    EQU  1DH,0    PCFG0     EQU  9FH,0
TMR2IF    EQU  0CH,1    ADRES     EQU  1EH      PUPOPS    EQU  6DH
TMR1IF    EQU  0CH,0    ADCON0    EQU  1FH      PUPOPP    EQU  6EH
PIR2      EQU  0DH      ADCS1     EQU  1FH,7    PUPOPF    EQU  6Fh
EEIF      EQU  0DH,4    ADCS0     EQU  1FH,6    ERG       EQU  70h
BCLIF     EQU  0DH,3    CHS2      EQU  1FH,5    ARG0      EQU  70h
CCP2IF    EQU  0DH,0    CHS1      EQU  1FH,4    ARG1      EQU  72h
TMR1L     EQU  0EH      CHS0      EQU  1FH,3    ARG2      EQU  74h
TMR1H     EQU  0FH      GO_DONE   EQU  1FH,2    ARG3      EQU  76h
T1CON     EQU  10H      ADON      EQU  1FH,0    ARG4      EQU  78h
T1CKPS1   EQU  10H,5    OPTION_R  EQU  81H      ARG5      EQU  7Ah
T1CKPS0   EQU  10H,4    RBUP      EQU  81H,7    ERRVAR    EQU  7Bh
```

DEFAULT.EQU (cont.)

```
T1OSCEN EQU 10H,3 INTEDG EQU 81H,6 ERR_ EQU 7Bh
T1SYNC    EQU  10H,2     T0CS      EQU  81H,5     TIMERX    EQU  7CH
TMR1CS    EQU  10H,1     T0SE      EQU  81H,4     DATPTR_   EQU  7Dh
TMR1ON    EQU  10H,0     PSA       EQU  81H,3     PUPOPW    EQU  7FH
TMR2      EQU  11H       PS2       EQU  81H,2     :
 T2CON     EQU  12H       PS1       EQU  81H,1      :16C77
TOUTPS3   EQU  12H,6     PS0       EQU  81H,0     PORTD     EQU  08H
TOUTPS2   EQU  12H,5     TRISA     EQU  85H       RD        EQU  08H
TOUTPS1   EQU  12H,4     TRISB     EQU  86H       PORTE     EQU  09H
TOUTPS0   EQU  12H,3     TRISC     EQU  87H       RE        EQU  09H
TMR2ON    EQU  12H,2     PIE1      EQU  8CH       TRISD     EQU  88H
T2CKPS1   EQU  12H,1     PSPIE     EQU  8CH,7     TRISE     EQU  89H
T2CKPS0   EQU  12H,0     ADIE      EQU  8CH,6     :
SSPBUF    EQU  13H       RCIE      EQU  8CH,5
;--------------------------------------------------------------------------
:16C83
:16F83
:16C84
 :16F84
INDIRECT  EQU  00H       INTCON    EQU  0BH       WRERR     EQU  88H,3
TMR0      EQU  01H       GIE       EQU  0BH,7     WREN      EQU  88H,2
RTCC      EQU  01H       EEIE      EQU  0BH,6     WR!       EQU  88H,1
PCL       EQU  02H       T0IE      EQU  0BH,5     RD!       EQU  88H,0
PC        EQU  02H       INTE      EQU  0BH,4     EECON2    EQU  89h
STATUS    EQU  03H       RBIE      EQU  0BH,3     PUPOPW    EQU  0CH
IRP       EQU  7         T0IF      EQU  0BH,2     PUPOPS    EQU  0DH
RP1       EQU  6         INTF      EQU  0BH,1     PUPOPP    EQU  0EH
RP0       EQU  5         RBIF      EQU  0BH,0     PUPOPF    EQU  0FH
TO        EQU  4         OPTION_R  EQU  81H       ERG       EQU  10h
PD        EQU  3         RBUP      EQU  81H,7     ARG0      EQU  10h
Z         EQU  2         INTEDG    EQU  81H,6     ARG1      EQU  12h
DC        EQU  1         T0CS      EQU  81H,5     ARG2      EQU  14h
C         EQU  0         T0SE      EQU  81H,4     ARG3      EQU  16h
FSR       EQU  04H       PSA       EQU  81H,3     ARG4      EQU  18h
PORTA     EQU  05H       PS2       EQU  81H,2     ARG5      EQU  1Ah
RA        EQU  05H       PS1       EQU  81H,1     ERRVAR    EQU  1Bh
PORTB     EQU  06H       PS0       EQU  81H,0     ERR_      EQU  1BH
RB        EQU  06H       TRISA     EQU  85H       TIMERX    EQU  1CH
EEDTA     EQU  08h       TRISB     EQU  86H       DATPTR_   EQU  1Dh
EEADR     EQU  09h       EECON1    EQU  88H       :
PCLATH    EQU  0Ah       EEIF      EQU  88H,4
;--------------------------------------------------------------------------
:16F818
 :16F819
INDIRECT  EQU  00H       T2CKPS1   EQU  12H,1     IRCF2     EQU  8FH,6
TMR0      EQU  01H       T2CKPS0   EQU  12H,0     IRCF1     EQU  8FH,5
RTCC      EQU  01H       SSPBUF    EQU  13H       IRCF0     EQU  8FH,4
PCL       EQU  02H       SSPCON    EQU  14H       IOFS      EQU  8FH,2
PC        EQU  02H       WCOL      EQU  14H,7     OSCTUNE   EQU  90H
STATUS    EQU  03H       SSPOV     EQU  14H,6     PR2       EQU  92H
IRP       EQU  7         SSPEN     EQU  14H,5     SSPADD    EQU  93H
```

DEFAULT.EQU (cont.)

```
RP1 EQU 6 CKP EQU 14H,4 SSPSTAT EQU 94H
RP0       EQU  5         SSPM3     EQU  14H,3     SMP       EQU  94H,7
TO        EQU  4         SSPM2     EQU  14H,2     CKE       EQU  94H,6
PD        EQU  3         SSPM1     EQU  14H,1     D_A       EQU  94H,5
Z         EQU  2         SSPM0     EQU  14H,0     P         EQU  94H,4
DC        EQU  1         CCPR1L    EQU  15H       S         EQU  94H,3
C         EQU  0         CCPR1H    EQU  16H       R_W       EQU  94H,2
FSR       EQU  04H       CCP1CON   EQU  17H       UA        EQU  94H,1
PORTA     EQU  05H       CCP1X     EQU  17H,5     BF        EQU  94H,0
RA        EQU  05H       CCP1Y     EQU  17H,4     ADRESL    EQU  9EH
PORTB     EQU  06H       CCP1M3    EQU  17H,3     ADCON1    EQU  9FH
RB        EQU  06H       CCP1M2    EQU  17H,2     ADFM      EQU  9FH,7
 PCLATH    EQU  0AH       CCP1M1    EQU  17H,1     ADCS2     EQU  9FH,6
INTCON    EQU  0BH       CCP1M0    EQU  17H,0     PCFG3     EQU  9FH,3
GIE       EQU  0BH,7     ADRESH    EQU  1EH       PCFG2     EQU  9FH,2
PEIE      EQU  0BH,6     ADCON0    EQU  1FH       PCFG1     EQU  9FH,1
T0IE      EQU  0BH,5     ADCS1     EQU  1FH,7     PCFG0     EQU  9FH,0
INTE      EQU  0BH,4     ADCS0     EQU  1FH,6     EEDTA     EQU  10CH
RBIE      EQU  0BH,3     CHS2      EQU  1FH,5     EEADR     EQU  10DH
T0IF      EQU  0BH,2     CHS1      EQU  1FH,4     EEDATH    EQU  10EH
INTF      EQU  0BH,1     CHS0      EQU  1FH,3     EEADRH    EQU  10FH
RBIF      EQU  0BH,0     GO_DONE   EQU  1FH,2     EECON1    EQU  18CH
PIR1      EQU  0CH       ADON      EQU  1FH,0     EEPGD     EQU  18CH,7
ADIF      EQU  0CH,6     OPTION_R  EQU  81H       FREE      EQU  18CH,4
SSPIF     EQU  0CH,3     RBUP      EQU  81H,7     WRERR     EQU  18CH,3
CCP1IF    EQU  0CH,2     INTEDG    EQU  81H,6     WREN      EQU  18CH,2
TMR2IF    EQU  0CH,1     T0CS      EQU  81H,5     WR_       EQU  18CH,1
TMR1IF    EQU  0CH,0     T0SE      EQU  81H,4     RD_       EQU  18CH,0
PIR2      EQU  0DH       PSA       EQU  81H,3     EECON2    EQU  18DH
EEIF      EQU  0DH,4     PS2       EQU  81H,2     PUPOPS    EQU  6DH
TMR1L     EQU  0EH       PS1       EQU  81H,1     PUPOPP    EQU  6EH
TMR1H     EQU  0FH       PS0       EQU  81H,0     PUPOPF    EQU  6Fh
T1CON     EQU  10H       TRISA     EQU  85H       ERG       EQU  70h
T1CKPS1   EQU  10H,5     TRISB     EQU  86H       ARG0      EQU  70h
T1CKPS0   EQU  10H,4     PIE1      EQU  8CH       ARG1      EQU  72h
T1OSCEN   EQU  10H,3     ADIE      EQU  8CH,6     ARG2      EQU  74h
T1SYNC    EQU  10H,2     SSPIE     EQU  8CH,3     ARG3      EQU  76h
TMR1CS    EQU  10H,1     CCP1IE    EQU  8CH,2     ARG4      EQU  78h
TMR1ON    EQU  10H,0     TMR2IE    EQU  8CH,1     ARG5      EQU  7Ah
TMR2      EQU  11H       TMR1IE    EQU  8CH,0     ERRVAR    EQU  7Bh
T2CON     EQU  12H       PIE2      EQU  8DH       ERR_      EQU  7Bh
TOUTPS3   EQU  12H,6     EEIE      EQU  8DH,4     TIMERX    EQU  7CH
TOUTPS2   EQU  12H,5     PCON      EQU  8EH       DATPTR_   EQU  7Dh
TOUTPS1   EQU  12H,4     POR       EQU  8EH,1     PUPOPW    EQU  7FH
TOUTPS0   EQU  12H,3     BOR       EQU  8EH,0     :
TMR2ON    EQU  12H,2     OSCCON    EQU  8FH
;-------------------------------------------------------------------------
:16F818
:16F819
INDIRECT  EQU  00H       T2CKPS0   EQU  12H,0     IRCF0     EQU  8FH,4
TMR0      EQU  01H       SSPBUF    EQU  13H       IOFS      EQU  8FH,2
```

DEFAULT.EQU (cont.)

```
RTCC EQU 01H SSPCON EQU 14H OSCTUNE EQU 90H
PCL       EQU  02H       WCOL      EQU  14H,7     PR2       EQU  92H
PC        EQU  02H       SSPOV     EQU  14H,6     SSPADD    EQU  93H
STATUS    EQU  03H       SSPEN     EQU  14H,5     SSPSTAT   EQU  94H
IRP       EQU  7         CKP       EQU  14H,4     SMP       EQU  94H,7
RP1       EQU  6         SSPM3     EQU  14H,3     CKE       EQU  94H,6
RP0       EQU  5         SSPM2     EQU  14H,2     D_A       EQU  94H,5
TO        EQU  4         SSPM1     EQU  14H,1     P         EQU  94H,4
PD        EQU  3         SSPM0     EQU  14H,0     S         EQU  94H,3
Z         EQU  2         CCPR1L    EQU  15H       R_W       EQU  94H,2
DC        EQU  1         CCPR1H    EQU  16H       UA        EQU  94H,1
C         EQU  0         CCP1CON   EQU  17H       BF        EQU  94H,0
FSR       EQU  04H       CCP1X     EQU  17H,5     ADRESL    EQU  9EH
PORTA     EQU  05H       CCP1Y     EQU  17H,4     ADCON1    EQU  9FH
RA        EQU  05H       CCP1M3    EQU  17H,3     ADFM      EQU  9FH,7
 PORTB     EQU  06H       CCP1M2    EQU  17H,2     ADCS2     EQU  9FH,6
RB        EQU  06H       CCP1M1    EQU  17H,1     PCFG3     EQU  9FH,3
PCLATH    EQU  0AH       CCP1M0    EQU  17H,0     PCFG2     EQU  9FH,2
INTCON    EQU  0BH       ADRESH    EQU  1EH       PCFG1     EQU  9FH,1
GIE       EQU  0BH,7     ADCON0    EQU  1FH       PCFG0     EQU  9FH,0
PEIE      EQU  0BH,6     ADCS1     EQU  1FH,7     EEDTA     EQU  10CH
T0IE      EQU  0BH,5     ADCS0     EQU  1FH,6     EEADR     EQU  10DH
INTE      EQU  0BH,4     CHS2      EQU  1FH,5     EEDATH    EQU  10EH
RBIE      EQU  0BH,3     CHS1      EQU  1FH,4     EEADRH    EQU  10FH
T0IF      EQU  0BH,2     CHS0      EQU  1FH,3     EECON1    EQU  18CH
INTF      EQU  0BH,1     GO_DONE   EQU  1FH,2     EEPGD     EQU  18CH,7
RBIF      EQU  0BH,0     ADON      EQU  1FH,0     FREE      EQU  18CH,4
PIR1      EQU  0CH       OPTION_R  EQU  81H       WRERR     EQU  18CH,3
ADIF      EQU  0CH,6     RBUP      EQU  81H,7     WREN      EQU  18CH,2
SSPIF     EQU  0CH,3     INTEDG    EQU  81H,6     WR_       EQU  18CH,1
CCP1IF    EQU  0CH,2     T0CS      EQU  81H,5     RD_       EQU  18CH,0
TMR2IF    EQU  0CH,1     T0SE      EQU  81H,4     EECON2    EQU  18DH
TMR1IF    EQU  0CH,0     PSA       EQU  81H,3     PRTTAB    EQU  68H
PIR2      EQU  0DH       PS2       EQU  81H,2     PUPOPS    EQU  6DH
EEIF      EQU  0DH,4     PS1       EQU  81H,1     PUPOPP    EQU  6EH
TMR1L     EQU  0EH       PS0       EQU  81H,0     PUPOPF    EQU  6Fh
TMR1H     EQU  0FH       TRISA     EQU  85H       ERG       EQU  70h
T1CON     EQU  10H       TRISB     EQU  86H       ARG0      EQU  70h
T1CKPS1   EQU  10H,5     PIE1      EQU  8CH       ARG1      EQU  72h
T1CKPS0   EQU  10H,4     ADIE      EQU  8CH,6     ARG2      EQU  74h
T1OSCEN   EQU  10H,3     SSPIE     EQU  8CH,3     ARG3      EQU  76h
T1SYNC    EQU  10H,2     CCP1IE    EQU  8CH,2     ARG4      EQU  78h
TMR1CS    EQU  10H,1     TMR2IE    EQU  8CH,1     ARG5      EQU  7Ah
TMR1ON    EQU  10H,0     TMR1IE    EQU  8CH,0     ERRVAR    EQU  7Bh
TMR2      EQU  11H       PIE2      EQU  8DH       ERR_      EQU  7Bh
T2CON     EQU  12H       EEIE      EQU  8DH,4     TIMERX    EQU  7CH
TOUTPS3   EQU  12H,6     PCON      EQU  8EH       DATPTR_   EQU  7Dh
TOUTPS2   EQU  12H,5     POR       EQU  8EH,1     PUPOPW    EQU  7FH
TOUTPS1   EQU  12H,4     BOR       EQU  8EH,0     :
TOUTPS0   EQU  12H,3     OSCCON    EQU  8FH
TMR2ON    EQU  12H,2     IRCF2     EQU  8FH,6
```

DEFAULT.EQU (cont.)

```
T2CKPS1 EQU 12H,1 IRCF1 EQU 8FH,5
;--------------------------------------------------------------------------
:16F870
:16F871
 :16F872
INDIRECT  EQU  00H      TOUTPS0   EQU  12H,3    POR       EQU  8EH,1
TMR0      EQU  01H      TMR2ON    EQU  12H,2    BOR       EQU  8EH,0
RTCC      EQU  01H      T2CKPS1   EQU  12H,1    SSPCON2   EQU  91H
PCL       EQU  02H      T2CKPS0   EQU  12H,0    GCEN      EQU  91H,7
PC        EQU  02H      SSPBUF    EQU  13H      ACKSTAT   EQU  91H,6
STATUS    EQU  03H      SSPCON    EQU  14H      ACKDT     EQU  91H,5
IRP       EQU  7        WCOL      EQU  14H,7    ACKEN     EQU  91H,4
RP1       EQU  6        SSPOV     EQU  14H,6    RCEN      EQU  91H,3
RP0       EQU  5        SSPEN     EQU  14H,5    REN       EQU  91H,2
TO        EQU  4        CKP       EQU  14H,4    RSEN      EQU  91H,1
PD        EQU  3        SSPM3     EQU  14H,3    SEN       EQU  91H,0
Z         EQU  2        SSPM2     EQU  14H,2    PR2       EQU  92H
DC        EQU  1        SSPM1     EQU  14H,1    SSPADD    EQU  93H
C         EQU  0        SSPM0     EQU  14H,0    SSPSTAT   EQU  94H
FSR       EQU  04H      CCPR1L    EQU  15H      SMP       EQU  94H,7
PORTA     EQU  05H      CCPR1H    EQU  16H      CKE       EQU  94H,6
RA        EQU  05H      CCP1CON   EQU  17H      D_A       EQU  94H,5
PORTB     EQU  06H      CCP1X     EQU  17H,5    P         EQU  94H,4
RB        EQU  06H      CCP1Y     EQU  17H,4    S         EQU  94H,3
PORTC     EQU  07H      CCP1M3    EQU  17H,3    R_W       EQU  94H,2
RC        EQU  07H      CCP1M2    EQU  17H,2    UA        EQU  94H,1
PCLATH    EQU  0AH      CCP1M1    EQU  17H,1    BF        EQU  94H,0
INTCON    EQU  0BH      CCP1M0    EQU  17H,0    ADRESL    EQU  9EH
GIE       EQU  0BH,7    ADRESH    EQU  1EH      ADCON1    EQU  9FH
PEIE      EQU  0BH,6    ADCON0    EQU  1FH      ADFM      EQU  9FH,7
T0IE      EQU  0BH,5    ADCS1     EQU  1FH,7    PCFG3     EQU  9FH,3
INTE      EQU  0BH,4    ADCS0     EQU  1FH,6    PCFG2     EQU  9FH,2
RBIE      EQU  0BH,3    CHS2      EQU  1FH,5    PCFG1     EQU  9FH,1
T0IF      EQU  0BH,2    CHS1      EQU  1FH,4    PCFG0     EQU  9FH,0
INTF      EQU  0BH,1    CHS0      EQU  1FH,3    EEDTA     EQU  10CH
RBIF      EQU  0BH,0    GO_DONE   EQU  1FH,2    EEADR     EQU  10DH
PIR1      EQU  0CH      ADON      EQU  1FH,0    EEDATH    EQU  10EH
PSPIF     EQU  0CH,7    OPTION_R  EQU  81H      EEADRH    EQU  10FH
ADIF      EQU  0CH,6    RBUP      EQU  81H,7    EECON1    EQU  18CH
RCIF      EQU  0CH,5    INTEDG    EQU  81H,6    EEPGD     EQU  18CH,7
TXIF      EQU  0CH,4    T0CS      EQU  81H,5    WRERR     EQU  18CH,3
SSPIF     EQU  0CH,3    T0SE      EQU  81H,4    WREN      EQU  18CH,2
CCP1IF    EQU  0CH,2    PSA       EQU  81H,3    WR_       EQU  18CH,1
TMR2IF    EQU  0CH,1    PS2       EQU  81H,2    RD_       EQU  18CH,0
TMR1IF    EQU  0CH,0    PS1       EQU  81H,1    EECON2    EQU  18DH
PIR2      EQU  0DH      PS0       EQU  81H,0    PUPOPS    EQU  6DH
EEIF      EQU  0DH,4    TRISA     EQU  85H      PUPOPP    EQU  6EH
BCLIF     EQU  0DH,3    TRISB     EQU  86H      PUPOPF    EQU  6Fh
CCP2IF    EQU  0DH,0    TRISC     EQU  87H      ERG       EQU  70h
TMR1L     EQU  0EH      PIE1      EQU  8CH      ARG0      EQU  70h
TMR1H     EQU  0FH      PSPIE     EQU  8CH,7    ARG1      EQU  72h
```

DEFAULT.EQU (cont.)

```
T1CON EQU 10H ADIE EQU 8CH,6 ARG2 EQU 74h
T1CKPS1   EQU  10H,5     RCIE      EQU  8CH,5     ARG3      EQU  76h
T1CKPS0   EQU  10H,4     TXIE      EQU  8CH,4     ARG4      EQU  78h
T1OSCEN   EQU  10H,3     SSPIE     EQU  8CH,3     ARG5      EQU  7Ah
T1SYNC    EQU  10H,2     CCP1IE    EQU  8CH,2     ERRVAR    EQU  7Bh
TMR1CS    EQU  10H,1     TMR2IE    EQU  8CH,1     ERR_      EQU  7Bh
TMR1ON    EQU  10H,0     TMR1IE    EQU  8CH,0     TIMERX    EQU  7CH
TMR2      EQU  11H       PIE2      EQU  8DH       DATPTR_   EQU  7Dh
T2CON     EQU  12H       EEIE      EQU  8DH,4     PUPOPW    EQU  7FH
TOUTPS3   EQU  12H,6     BCLIE     EQU  8DH,3     :
TOUTPS2   EQU  12H,5     CCP2IE    EQU  8DH,0
TOUTPS1   EQU  12H,4     PCON      EQU  8EH
;--------------------------------------------------------------------------
:16F873
:16F873A
:16F874
:16F874A
:16F876
:16F876A
:16F877
 :16F877A
INDIRECT  EQU  00H       CCP1CON   EQU  17H       SSPADD    EQU  93H
TMR0      EQU  01H       CCP1X     EQU  17H,5     SSPSTAT   EQU  94H
RTCC      EQU  01H       CCP1Y     EQU  17H,4     SMP       EQU  94H,7
PCL       EQU  02H       CCP1M3    EQU  17H,3     CKE       EQU  94H,6
PC        EQU  02H       CCP1M2    EQU  17H,2     D_A       EQU  94H,5
STATUS    EQU  03H       CCP1M1    EQU  17H,1     P         EQU  94H,4
IRP       EQU  7         CCP1M0    EQU  17H,0     S         EQU  94H,3
RP1       EQU  6         RCSTA     EQU  18H       R_W       EQU  94H,2
RP0       EQU  5         SPEN      EQU  18H,7     UA        EQU  94H,1
TO        EQU  4         RX9       EQU  18H,6     BF        EQU  94H,0
PD        EQU  3         SREN      EQU  18H,5     TXSTA     EQU  98H
Z         EQU  2         CREN      EQU  18H,4     CSRC      EQU  98H,7
DC        EQU  1         ADDEN     EQU  18H,3     TX9       EQU  98H,6
C         EQU  0         FERR      EQU  18H,2     TXEN      EQU  98H,5
FSR       EQU  04H       OERR      EQU  18H,1     SYNC      EQU  98H,4
PORTA     EQU  05H       RX9D      EQU  18H,0     BRGH      EQU  98H,2
RA        EQU  05H       TXREG     EQU  19H       TRMT      EQU  98H,1
PORTB     EQU  06H       RCREG     EQU  1AH       TX9D      EQU  98H,0
RB        EQU  06H       CCPR2L    EQU  1BH       SPBRG     EQU  99H
PORTC     EQU  07H       CCPR2H    EQU  1CH       ADRESL    EQU  9EH
RC        EQU  07H       CCP2CON   EQU  1DH       ADCON1    EQU  9FH
PCLATH    EQU  0AH       CCP2X     EQU  1DH,5     ADFM      EQU  9FH,7
INTCON    EQU  0BH       CCP2Y     EQU  1DH,4     PCFG3     EQU  9FH,3
GIE       EQU  0BH,7     CCP2M3    EQU  1DH,3     PCFG2     EQU  9FH,2
PEIE      EQU  0BH,6     CCP2M2    EQU  1DH,2     PCFG1     EQU  9FH,1
T0IE      EQU  0BH,5     CCP2M1    EQU  1DH,1     PCFG0     EQU  9FH,0
INTE      EQU  0BH,4     CCP2M0    EQU  1DH,0     EEDTA     EQU  10CH
RBIE      EQU  0BH,3     ADRESH    EQU  1EH       EEADR     EQU  10DH
T0IF      EQU  0BH,2     ADCON0    EQU  1FH       EEDATH    EQU  10EH
INTF      EQU  0BH,1     ADCS1     EQU  1FH,7     EEADRH    EQU  10FH
```

DEFAULT.EQU (cont.)

```
RBIF EQU 0BH,0 ADCS0 EQU 1FH,6 EECON1 EQU 18CH
PIR1      EQU  0CH       CHS2      EQU  1FH,5     EEPGD     EQU  18CH,7
PSPIF     EQU  0CH,7     CHS1      EQU  1FH,4     WRERR     EQU  18CH,3
ADIF      EQU  0CH,6     CHS0      EQU  1FH,3     WREN      EQU  18CH,2
RCIF      EQU  0CH,5     GO_DONE   EQU  1FH,2     WR_       EQU  18CH,1
TXIF      EQU  0CH,4     ADON      EQU  1FH,0     RD_       EQU  18CH,0
SSPIF     EQU  0CH,3     OPTION_R  EQU  81H       EECON2    EQU  18DH
CCP1IF    EQU  0CH,2     RBUP      EQU  81H,7     PUPOPS    EQU  6DH
TMR2IF    EQU  0CH,1     INTEDG    EQU  81H,6     PUPOPP    EQU  6EH
TMR1IF    EQU  0CH,0     T0CS      EQU  81H,5     PUPOPF    EQU  6Fh
PIR2      EQU  0DH       T0SE      EQU  81H,4     ERG       EQU  70h
EEIF      EQU  0DH,4     PSA       EQU  81H,3     ARG0      EQU  70h
BCLIF     EQU  0DH,3     PS2       EQU  81H,2     ARG1      EQU  72h
CCP2IF    EQU  0DH,0     PS1       EQU  81H,1     ARG2      EQU  74h
TMR1L     EQU  0EH       PS0       EQU  81H,0     ARG3      EQU  76h
TMR1H     EQU  0FH       TRISA     EQU  85H       ARG4      EQU  78h
T1CON     EQU  10H       TRISB     EQU  86H       ARG5      EQU  7Ah
T1CKPS1   EQU  10H,5     TRISC     EQU  87H       ERRVAR    EQU  7Bh
T1CKPS0   EQU  10H,4     PIE1      EQU  8CH       ERR_      EQU  7Bh
T1OSCEN   EQU  10H,3     PSPIE     EQU  8CH,7     TIMERX    EQU  7CH
T1SYNC    EQU  10H,2     ADIE      EQU  8CH,6     DATPTR_   EQU  7Dh
TMR1CS    EQU  10H,1     RCIE      EQU  8CH,5     PUPOPW    EQU  7FH
TMR1ON    EQU  10H,0     TXIE      EQU  8CH,4     :
 TMR2      EQU  11H       SSPIE     EQU  8CH,3     : 16F871
 T2CON     EQU  12H       CCP1IE    EQU  8CH,2     : 16F874
 TOUTPS3   EQU  12H,6     TMR2IE    EQU  8CH,1     : 16F874A
 TOUTPS2   EQU  12H,5     TMR1IE    EQU  8CH,0     : 16F877
 TOUTPS1   EQU  12H,4     PIE2      EQU  8DH       : 16F877A
TOUTPS0   EQU  12H,3     EEIE      EQU  8DH,4     PORTD     EQU  08H
TMR2ON    EQU  12H,2     BCLIE     EQU  8DH,3     RD        EQU  08H
T2CKPS1   EQU  12H,1     CCP2IE    EQU  8DH,0     PORTE     EQU  09H
T2CKPS0   EQU  12H,0     PCON      EQU  8EH       RE        EQU  09H
SSPBUF    EQU  13H       POR       EQU  8EH,1     TRISD     EQU  88H
SSPCON    EQU  14H       BOR       EQU  8EH,0     TRISE     EQU  89H
WCOL      EQU  14H,7     SSPCON2   EQU  91H       :
 SSPOV     EQU  14H,6     GCEN      EQU  91H,7     : 16F873A
 SSPEN     EQU  14H,5     ACKSTAT   EQU  91H,6     : 16F874A
 CKP       EQU  14H,4     ACKDT     EQU  91H,5     : 16F876A
 SSPM3     EQU  14H,3     ACKEN     EQU  91H,4     : 16F877A
SSPM2     EQU  14H,2     RCEN      EQU  91H,3     CMCON     EQU  9CH
SSPM1     EQU  14H,1     REN       EQU  91H,2     CVRCON    EQU  9DH
SSPM0     EQU  14H,0     RSEN      EQU  91H,1     :
CCPR1L    EQU  15H       SEN       EQU  91H,0
CCPR1H    EQU  16H       PR2       EQU  92H
;--------------------------------------------------------------------------
```

Error codes

label, symbol or keyword already defined
symbol table full
= expected
TO expected
too many FOR-NEXT nested
wrong FOR-NEXT nested
NEXT without FOR
label not defined
too many GOSUBs nested
pin number not available
port not available
only variables allowed or wrong variable
variable already defined
too much or too less arguments
THEN expected
only 8 bit variables and values allowed
syntax error
(maximale) Zeitangaben nicht aufl"sbar
must be a boolean constant value
symbol not defined
command for PIC16C8x only
command for PIC16C7x only
AD converter not configured
only 16 bits variables
command for PIC16C7x and PIC16C8x only
target address beyond available area
program too big to fit
watchdog is off
too many functions in run time library. (page to small)
error in lcd initialisation
ENDASM missing
error in I2C initialisation
already initialized
format error
only constant values allowed
only 8 bits variables
wrong interrupt handling
warning! time base <> 1msec within WAIT or 100?s within DELAY
compiling file
building symbol table
linking library
saving...
no error found by compiler
file not found
compiling line
array error
no direct transfer from array to array allowed
concatening too many lines (200 characters max.)
matrix keyboard not initialized
error in line
unkown CPU

Error codes (cont.)

warnung in line
hey, a little bit more faster? It is your responsibility.
too many data items
this variable must be in bank 0
instruction wrong
file "*.PIC" not found
file "*.EQU" not found
instruction not for 12X5xx and 16C5x
xtal frquency too low
constant exceeds limit
ARITH32 variable not defined

Supported PICs

iL_BAS16SES:
PIC 16F628 and 16F84

iL_BAS16SEP:
PIC 12F629, 16F627, 16F628, 16F84, 16F877

iL_BAS16STD:
see www.iL-online.de

iL_BAS16PRO
see www.iL-online.de

FAQs

For FAQs please see www.iL-online.de