PIC18XXX Tools Quick Chart

MPLAB® C18 Compiler Software Installation

To install the MPLAB C18 Compiler, run the setup program from the CD-ROM. A series of dialogs will step through the setup process. When installing MPLAB C18 for the first time, the default installation directory is `C:\mcc18`.

Environment Variables

Use these settings either through the `AUTOEXEC.BAT` file or your DOS properties so that when using MPLAB C18 from the command line, the path to the executables and header files will not have to be specified.

`PATH=C:\MCC18\BIN;`<additional paths for other apps>

Allows MPLAB C18 and the MPLINK™ linker to be executed at the command shell prompt from any directory.

```
PATH=C:\MCC18\MPASM;C:\MCC18\BIN;C:\Program Files\MPLAB IDE\MCHIP_Tools;%PATH%
```

Allows the MPASM™ assembler to be executed at the command shell prompt from any directory.

```
SET MCC_INCLUDE=c:\mcc18\h
```

Specifies the default search path for include files.

Help Resources

Refer to the Troubleshooting section of MPLAB IDE Help and the Microchip web site (www.microchip.com) for:

- On-line support
- Latest development tool downloads and updates, data sheets, application notes, user's guides, articles and sample programs
- Web Conference, Design Tips, Device Errata
- Microchip Change Notification System

Development Systems Information Line and Technical Support:

1-800-755-2345 for U.S. and most of Canada
1-480-792-7302 for the rest of the world.

Key to PIC18XXX Instruction Set

Field	Description
f, fs, fd	8-bit file register address
r	0, 1 or 2 for FSR0, FSR1, FSR2 register
b	3-bit value representing bit position 0-7
a	Access bit, 0=Access reg, 1=Use BSR
d	Destination bit, 0=WREG, 1=f
kb, kk, kc	4-, 8- and 12-bit literal value, respectively
nn	8-bit relative offset (signed, 2's complement)
nd	11-bit relative offset (signed, 2's complement)
mm	20-bit program memory address

Literal Instructions

Mnemonic	Description	Function
ADDLW kk	ADD literal to WREG	W+kk → W
ANDLW kk	AND literal with WREG	W .AND. kk → W
CLRWDT	Clear Watchdog Timer	0 → WDT, 0 → WDT postscaler, 1 → TO,1 → PD
DAW	Decimal Adjust WREG	if W<3:0> >9 or DC=1, W<3:0>+6→W<3:0>, else W<3:0> → W<3:0>; if W<7:4> >9 or C=1, W<7:4>+6→W<7:4>, else W<7:4> → W<7:4>
IORLW kk	Inclusive OR literal with WREG	W .OR. kk → W
LFSR r,kc	Load 12-bit Literal to FSR (second word)	kc → FSRr
MOVLB kb	Set BSR bank	kb → BSR
MOVLW kk	Move literal to WREG	kk → W
MULLW kk	Multiply lit with WREG	W * kk → PRODH:PRODL
SUBLW kk	Subtract W from literal	kk–W → W
XORLW kk	Excl OR lit with WREG	W .XOR. kk → W

Core Memory Instructions

Mnemonic	Description	Function
TBLRD*	Table Read (no change to TBLPTR)	Prog Mem (TBLPTR) → TABLAT
TBLRD*+	Table Read (post-increment TBLPTR)	Prog Mem (TBLPTR) → TABLAT TBLPTR +1 → TBLPTR
TBLRD*-	Table Read (post-decrement TBLPTR)	Prog Mem (TBLPTR) → TABLAT TBLPTR -1 → TBLPTR
TBLRD+*	Table Read (pre-increment TBLPTR)	TBLPTR +1 → TBLPTR Prog Mem (TBLPTR) → TABLAT
TBLWT*	Table Write (no change to TBLPTR)	TABLAT → Prog Mem(TBLPTR)
TBLWT*+	Table Write (post-increment TBLPTR)	TABLAT → Prog Mem(TBLPTR) TBLPTR +1 → TBLPTR
TBLWT*-	Table Write (post-decrement TBLPTR)	TABLAT → Prog Mem(TBLPTR) TBLPTR -1 → TBLPTR
TBLWT+*	Table Write (pre-increment TBLPTR)	TBLPTR +1 → TBLPTR TABLAT → Prog Mem(TBLPTR)

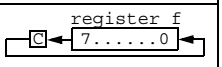
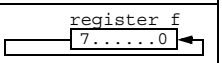
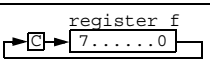
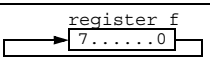
Control Instructions

Mnemonic		Description	Function
BC	nn	Relative Branch if Carry	if C=1, PC+2+2*nn→PC
BN	nn	Relative Branch if Negative	if N=1, PC+2+2*nn→PC
BNC	nn	Relative Branch if Not Carry	if C=0, PC+2+2*nn→PC
BNN	nn	Relative Branch if Not Negative	if N=0, PC+2+2*nn→PC
BNOV	nn	Relative Branch if Not Overflow	if OV=0, PC+2+2*nn→PC
BNZ	nn	Relative Branch if Not Zero	if Z=0, PC+2+2*nn→PC
BOV	nn	Relative Branch if Overflow	if OV=1, PC+2+2*nn→PC
BRA	nd	Unconditional Relative Branch	PC+2+2*nd→PC
BZ	nn	Relative Branch if Zero	if Z=1, PC+2+2*nn→PC
CALL	mm,s	Absolute Subroutine Call (second word)	PC+4 → TOS, mm → PC<20:1>, if s=1, W → WS, STATUS → STATUSS, BSR → BSRS
GOTO	mm	Absolute Branch (second word)	mm → PC<20:1>
NOP		No Operation	No operation
POP		Pop Top/stack	TOS-1 → TOS
PUSH		Push Top/stack	PC +2 → TOS
RCALL	nd	Relative Subroutine Call	PC+2 → TOS, PC+2+2*nd→PC
RESET		Generate a Reset (same as MCLR reset)	same as $\overline{\text{MCLR}}$ reset
RETFIE	s	Return from interrupt (and enable interrupts)	TOS → PC, 1 → GIE/GIEH or PEIE/GIEL, if s=1, WS → W, STATUSS → STATUS, BSRS → BSR
RETLW	kk	Return from subroutine, store literal in W	TOS → PC, kk → W
RETURN	s	Return from subroutine	TOS → PC, if s=1, WS → W, STATUSS → STATUS, BSRS → BSR
SLEEP		Enter Sleep Mode	0 → WDT, 0 → WDT postscaler, 1 → TO, 0 → PD

Bit Instructions

Mnemonic		Description	Function
BCF	f,b,a	Bit Clear f	0 → f<b>
BSF	f,b,a	Bit Set f	1 → f<b>
BTFSC	f,b,a	Bit test f, skip if clear	if f<b>=0, PC+4→PC
BTFSS	f,b,a	Bit test f, skip if set	if f<b>=1, PC+4→PC
BTG	f,b,a	Bit Toggle f	~f<b> → f<b>

File Register Instructions

Mnemonic		Description	Function
ADDWF	f,d,a	ADD WREG to f	W+f → dest
ADDWFC	f,d,a	ADD WREG and Carry bit to f	W+f+C → dest
ANDWF	f,d,a	AND WREG with f	W .AND. f → dest
CLRF	f,a	Clear f	0 → f
COMF	f,d,a	Complement f	~f → dest
CPFSEQ	f,a	Compare f with WREG, skip if f=WREG	f–W, if f=W, PC+4 → PC else PC+2 → PC
CPFSGT	f,a	Compare f with WREG, skip if f > WREG	f–W, if f > W, PC+4 → PC else PC+2 → PC
CPFSLT	f,a	Compare f with WREG, skip if f < WREG	f–W, if f < W, PC+4 → PC else PC+2 → PC
DECF	f,d,a	Decrement f	f–1 → dest
DECFSZ	f,d,a	Decrement f, skip if 0	f–1 → dest, if dest=0, PC+4 → PC else PC+2 → PC
DCFSNZ	f,d,a	Decrement f, skip if not 0	f–1 → dest, if dest ≠ 0, PC+4 → PC else PC+2 → PC
INCF	f,d,a	Increment f	f+1 → dest
INCFSZ	f,d,a	Increment f, skip if 0	f+1 → dest, if dest=0, PC+4 → PC else PC+2 → PC
INFSNZ	f,d,a	Increment f, skip if not 0	f+1 → dest, if dest ≠ 0, PC+4 → PC else PC+2 → PC
IORWF	f,d,a	Inclusive OR WREG with f	W .OR. f → dest
MOVF	f,d,a	Move f	f → dest
MOVFF	fs,fd	Move fs (first word) to fd (second word)	fs → fd
MOVWF	f,a	Move WREG to f	W → f
MULWF	f,a	Multiply WREG with f	W * f → PRODH:PRODL
NEGF	f,a	Negate f	~f + 1 → f
RLCF	f,d,a	Rotate f left through Carry	
RLNCF	f,d,a	Rotate f left (no carry)	
RRCF	f,d,a	Rotate f right through Carry	
RRNCF	f,d,a	Rotate f right (no carry)	
SETF	f,a	Set f	0xFF → f
SUBFWB	f,d,a	Subtract f from WREG with Borrow	W - f - $\overline{\text{C}}$ → dest
SUBWF	f,d,a	Subtract WREG from f	f - W → dest
SUBWFB	f,d,a	Subtract WREG from f with Borrow	f - W - $\overline{\text{C}}$ → dest
SWAPF	f,d,a	Swap nibbbles of f	f<3:0> → dest<7:4>, f<7:4> → dest<3:0>
TSTFSZ	f,a	Test f, skip if 0	if f=0, PC+4 → PC else PC+2 → PC
XORWF	f,d,a	Exclusive OR WREG with f	W .XOR. f → dest

Two Word Instructions

The PIC18XXX instruction set consists of mainly single word (two byte) and a few double word (four byte) instructions. The second word of every two word instruction always has a value of 0xFn for the first byte. Such instructions always execute as a NOP. This allows a “skip” instruction, such as `BTFSC` to be used before any two word instruction. If the skip is taken, it will skip over the first word of a two word instruction to the second word, execute a NOP and continue on with the next instruction.

FAST Interrupts and FAST CALLs

Bit 8 in the `CALL` instruction determines whether the `WREG`, `STATUS` and `BSR` registers are automatically saved on the `FAST` hardware stack (`fast=1`). Use:

```
call mysub,FAST
```

then use:

```
return FAST
```

to let the CPU automatically save and restore `WREG`, `STATUS` and `BSR`. Bit 1 in the `RETURN` instruction is set to one for `FAST` returns. Note that this special stack is only one level deep, and `FAST CALL`s and `FAST` Interrupts cannot be nested. If `FAST` interrupts are used, `FAST CALL`s must be avoided.

MPLAB C18 Data Types

Type	Bit Width	Range
`void`	–	none
`char`	8	-128 to 127
`unsigned char`	8	0 to 255
`int`	16	-32,768 to 32,767
`unsigned int`	16	0 to 65,535
`short`	16	-32,768 to 32,767
`unsigned short`	16	0 to 65,535
`short long`	24	-8,388,608 to 8,388,607
`unsigned short long`	24	0 to 16,777,215
`long`	32	-2,147,483,648 to 2,147,483,647
`unsigned long`	32	0 to 4,294,967,295
`float`	32	1.7549435E-38 to 6.80564693E+38
`double`	32	1.7549435E-38 to 6.80564693E+38

Note: A plain `char` is signed by default.
A plain `char` may be unsigned by default via the -k command-line option.

MPLAB C18 Floating-Point Format

The MPLAB C18 format for floating-point numbers is a modified form of the IEEE 754 format. The difference between the MPLAB C18 format and the IEEE 754 format consists of a rotation of the top nine bits of the representation. A left rotate will convert from the IEEE 754 format to the MPLAB C18 format. A right rotate will convert from the MPLAB C18 format to the IEEE 754 format.

Floating-Point Standard	Byte 3	Byte 0	Byte 1	Byte 2
IEEE 754	$seeeeeee_1$	$e_0ddd\ dddd_{16}$	$dddd\ dddd_8$	$dddd\ dddd_0$
MPLAB C18	$eeeeeeee_0$	$sddd\ dddd_{16}$	$dddd\ dddd_8$	$dddd\ dddd_0$

Legend: s = sign bit, d = mantissa, e = exponent

Common Variable Modifiers

Modifier	Use
const	Variable will not be modified
far	Variable is paged/banked regardless of memory model selected
extern	Variable is allocated in another module
near	Variable is not paged/banked regardless of memory model selected
ram	Locate object in data memory
rom	Locate object in program memory
static	Variable is retained unchanged between executions of the defining block.
volatile	Variable may change from other sources (e.g., input port)

Data Storage Format

Endian refers to the ordering of bytes in a multi-byte value. MPLAB C18 stores data in little-endian format. Bytes at lower addresses have lower significance (the value is stored “little-end-first”). For example:

```
#pragma idata test = 0x0200
long ltemp = 0xAABBCCDD;
```

results in a memory layout as follows:

Itemp Address	0x0200	0x0201	0x0202	0x0203
Itemp Contents	0xDD	0xCC	0xBB	0xAA

Pointer Sizes

Pointer Type	Example	Size
Data memory	`char * dmp;` `near char * npmp;`	16 bits
Near pgm memory	`rom near char * npmp;`	16 bits
Far pgm memory	`rom far char * fpmp;`	24 bits

Instruction Macros

These macros are provided for efficient use of some of the PIC18XXX instructions directly from C code:

Instruction[1]	Macro Action
`Nop()`	Execute a no operation.
`ClrWdt()`	Clear the watchdog timer.
`Sleep()`	Execute a `SLEEP` instruction.
`Reset()`	Execute a device reset.
`Rlcf(var, dest, access)` [2,3]	Rotate `var` to the left through the carry bit.
`Rlncf(var, dest, access)` [2,3]	Rotate `var` to the left without affecting the carry bit.
`Rrcf(var, dest, access)` [2,3]	Rotate `var` to the right through carry bit.
`Rrncf(var, dest, access)` [2,3]	Rotate `var` to the right without affecting the carry bit.
`Swapf(var, dest, access)` [2,3]	Swap the upper and lower nibble of `var`.

Note 1: Using any of these macros in a function affects the ability of the MPLAB C18 compiler to perform optimizations on that function.

2: `var` must be an 8-bit quantity (i.e., char) and not located on the stack.

3: If `dest` is 0, the result is stored in WREG, and if `dest` is 1, the result is stored in `var`. If `access` is 0, the access area will be selected, overriding the BSR value. If `access` is 1, then the bank will be selected according to the BSR value.

MPLAB C18 Interrupts

To create an interrupt service routine no additional libraries are required. Follow these steps:

- Create a `code` section at the interrupt vector that contains a `goto isr` statement, either using inline assembly or a separate assembly file.
- Declare the interrupt routine in the source code using one of the following statements:

High-priority interrupts – `W`, `BSR` and `STATUS` are saved in shadow registers.

```
#pragma interrupt <isr> [save=symbol-list]
```

Low-priority interrupts – `W`, `BSR` and `STATUS` are saved on the software stack.

```
#pragma interruptlow <isr> [save=sym-list]
```

If your ISR calls non-ISR functions, the temporary data section must be saved. This is done using the section qualifier on the `save=` keyword.

```
#pragma interruptlow <isr>
                    |save=section(".tmpdata"|
```

Compiler Managed Resources at Interrupts

MPLAB C18 will save some registers automatically when an interrupt occurs. In order to make sure that other registers are saved and restored properly use the `save=` construct in the `#pragma interrupt` declaration.

Compiler-Managed Resource	Primary Use(s)	Auto Saved
`PC`	execution control	x
`WREG`	intermediate calculations	x
`STATUS`	calculation results	x
`BSR`	bank selection	x
`PROD`	multiplication results, return values, intermediate calculations	
`section.tmpdata`	intermediate calculations	
`FSR0`	pointers to RAM	x
`FSR1`	stack pointer	x
`FSR2`	frame pointer	x
`TBLPTR`	accessing values in program memory	
`TABLAT`	accessing values in program memory	
`PCLATH`	function pointer invocation	
`PCLATU`	function pointer invocation	
`section MATH_DATA`	arguments, return values and temporary locations for math library functions	

Note: Compiler temporary variables for non-ISR functions are placed in an access qualified `udata` section named `.tmpdata`. Interrupt service routines each create a separate section for temporary data storage, so, section `.tmpdata` doesn't need to be saved if the ISR makes no function calls.

18F452i Linker Script

Linker scripts tell MPLINK which areas of memory are available for data and program code. Here is a linker script for debugging a PIC18F452 application with MPLAB ICD 2.

```
CODEPAGE NAME=vectors START=0x0      END=0x29     PROTECTED
CODEPAGE NAME=page    START=0x2A     END=0x7DBF
CODEPAGE NAME=debug   START=0x7DC0   END=0x7FFF   PROTECTED
CODEPAGE NAME=idlocs  START=0x200000 END=0x200007 PROTECTED
CODEPAGE NAME=config  START=0x300000 END=0x30000D PROTECTED
CODEPAGE NAME=devid   START=0x3FFFFE END=0x3FFFFF PROTECTED
CODEPAGE NAME=eedata  START=0xF00000 END=0xF000FF PROTECTED

ACCESSBANK NAME=accessram START=0x0    END=0x7F
DATABANK   NAME=gpr0      START=0x80   END=0xFF
DATABANK   NAME=gpr1      START=0x100  END=0x1FF
DATABANK   NAME=gpr2      START=0x200  END=0x2FF
DATABANK   NAME=gpr3      START=0x300  END=0x3FF
DATABANK   NAME=gpr4      START=0x400  END=0x4FF
DATABANK   NAME=gpr5      START=0x500  END=0x5F3
DATABANK   NAME=dbgspr    START=0x5F4  END=0x5FF  PROTECTED
ACCESSBANK NAME=accesssfr START=0xF80  END=0xFFF  PROTECTED

SECTION    NAME=CONFIG    ROM=config
```

This linker script is for use with MPLAB ICD 2, so the area in program memory assigned to the `CODEPAGE` area `debug` and the area in RAM noted by the `DATABANK` area `dbgspr` are marked `PROTECTED`.

Locating Code

Following a `#pragma code` directive, all generated code will be assigned to the specified code section until another `#pragma code` directive is encountered. An absolute code section allows the location of code to a specific address. For example:

```
#pragma code my_code=0x2000
```

will locate the code section `my_code` at program memory address 0x2000. If the address is left blank, the linker will choose from available free blocks of code space.

Locating Data

Data can be placed in either data or program memory with the MPLAB C18 compiler. To locate data in RAM, it can either be uninitialized data (`udata`) or initialized data (`idata`). When using intialized data, all the data is stored in program memory and then moved to RAM before the `main` application function at `main` is executed (this is done in the object file `c018i.o`). The following declares a section for statically allocated uninitialized data (`udata`) at absolute address 0x120:

```
#pragma udata my_new_data_section=0x120
```

Data that is placed in on-chip program memory can be read but not written without additional user-supplied code. The `rom` keyword tells the compiler that the data should be placed in program memory. The compiler will allocate this data into the current `romdata` type section. For example:

```
#pragma romdata const_table
 const rom char my_const_array[10]=
             {0,1,2,3,4,5,6,7,8,9};
       /* Resume allocation of romdata
          into the default section */
#pragma romdata
```

MPLAB C18 In-line Assembly

MPLAB C18 has an internal assembler with a syntax similar to the MPASM assembler, except that comments must be in the C (`/* */`) or C++ (`//`) style. The block of assembly code must begin with `_asm` and end with `_endasm`. For example:

```
_asm
              /* User assembly code */
   MOVLW 10   // Move decimal 10 to count
   MOVWF count, 0
               /* Loop until count is 0 */
start:
   DECFSZ count, 1, 0
   GOTO done
   BRA start
done:
_endasm
```

Note that with in-line assembly, the access bit and the destination bit must be explicitly entered for each instruction.

Configuration Bits

The `#pragma romdata CONFIG` directive is used to set the current romdata section to the section named `CONFIG`. The configuration for the device can be specified using the `_CONFIG_DECL` macro and the `#defines` located in the processor-specific header file.

```
#include <p18c452.h>
#pragma romdata CONFIG
_CONFIG_DECL
    (_CP_ON_1L,
     _OSCS_ON_1H & _OSC_LP_1H,
       _PWRT_ON_2L & _BOR_OFF_2L &
     _BORV_42_2L,
     _WDT_OFF_2H & _WDTPS_1_2H,
     _CCP2MUX_OFF_3H,
     _CONFIG4L_DEFAULT);
#pragma romdata
void main (void)
{
  ...
}
```

Return Values

Functions that return values will return them in different registers depending upon the return value size:

Return Value Size	Return Value Location
8 bits	`WREG`
16 bits	`PRODH:PRODL`
24 bits	`(AARGB2+2):(AARGB2+1):AARGB2`
32 bits	`(AARGB3+3):(AARGB3+2):(AARGB3+1):ARGB3`
> 32 bits	On the stack, `FSR0` points to the return value

PIC18XXX Library Files

File	Use
`clib.lib`	Standard C routines, math routines.
`c018i.o`	Startup code with initialized data support.
`c018iz.o`	Startup code with initialized data support that clears unused RAM.
`c018.o`	Startup code without initialized data support.
`p18xxxx.lib`	Peripheral library routines and SFR definitions.

`xxxx` = Processor type (e.g., C452 for PIC18C452)

MPLAB ICD 2 Alerts

PLL

Care should be taken when programming the Phase Locked Loop oscillator (PLL). The PLL only changes when power is first applied to the chip. When programming the PLL for the first time, remove power from the PIC18FXXX part after programming and reapply for the PLL to be enabled. When reprogramming the device from PLL mode to another mode, first reprogram with PLL off, then remove power and reapply.

Flash Memory Blocks

For a range of program memory, the Start Address must be set to the beginning of an 8-byte block. The End Address must be set to the end of an 8-byte block, i.e., a Start Address of `0x10` and an End Address of `0x1F`. If a programming error is received due to an incorrect End Address, click the Connect button, correct the End Address and click the Program button again.

PIC18FXX20

All AVDD and AVSS pins must be connected for the device to program.

General Alerts

SLEEP

Do not single step into, set a breakpoint on or break/halt during execution of a `SLEEP` instruction. If this happens, select Debugger>Reinitialize ICE Hardware in order to wake up the processor module. In code, use a Watchdog Timer time-out or other suitable method to wake the processor from `SLEEP` mode.

Interrupts While Single Stepping

Interrupts will not work when single stepping through code. Interrupts will work only when running.

$\overline{\text{MCLR}}$ While Single Stepping

Initiating a master clear on the $\overline{\text{MCLR}}$ pin will not reset the processor when in step mode.

Emulator Unimplemented GPRs

Some unimplemented General Purpose Registers in the emulator can be written. Therefore, their read values are not guaranteed to be zero (as is the case in the actual device).

Low Voltage Emulation

In-circuit emulation is limited to 2.5 to 5.5 volts.

Table Write Results in MPLAB IDE Windows

If performing table writes, "Upload Program Memory from ICE" must be selected before the Program Memory window will be modified.

Table Reads of Breakpoint Locations

If performing table reads, a software breakpoint will be a `TRAP` instruction, so these locations will not read correctly when performing program memory reads. This will affect any run-time checksum routines. It is recommended that run-time checksums be disabled while debugging.

Additional Reference Documents

PICmicro 18C MCU Family Reference Manual (DS39500)

MPLAB C18 C Compiler Getting Started (DS51295)

MPLAB C18 C Compiler User's Guide (DS51288)

MPLAB C18 C Compiler Libraries (DS51297)

Embedded Design with the PIC18F452 Microcontroller, by John B. Peatman, Prentice Hall, (c) 2003 Pearson Edication, Inc., ISBN 0-13-046213-6.

Microchip Technology Inc. • 2355 West Chandler Blvd.
Chandler, AZ 85224-6199 • 480-792-7200
www.microchip.com