



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



## Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

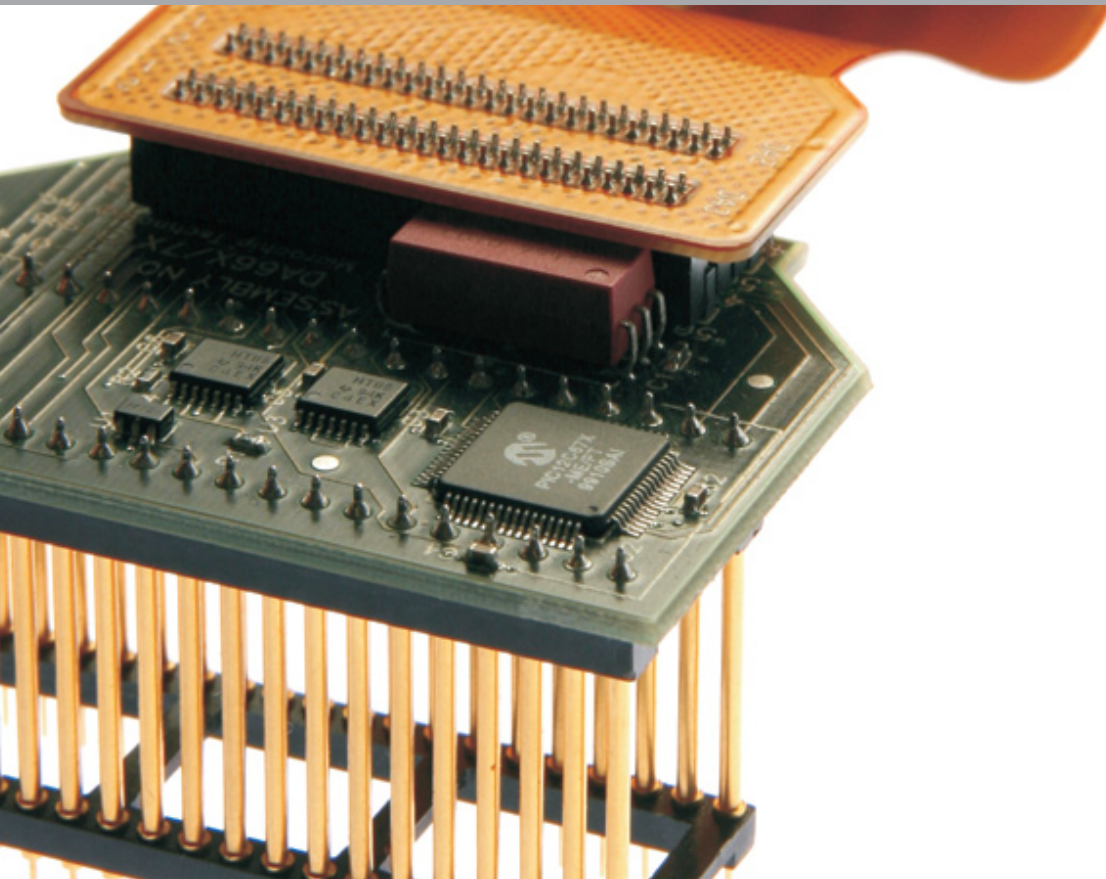
Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China





# HI-TECH C<sup>®</sup> Tools for the PIC10/12/16 MCU Family





# HI-TECH C Tools for the PIC10/12/16 MCU Family

HI-TECH Software.

Copyright (C) 2008 HI-TECH Software.  
All Rights Reserved. Printed in Australia.

Produced on: July 29, 2008

HI-TECH Software Pty. Ltd.  
ACN 002 724 549  
45 Colebard Street West  
Acacia Ridge QLD 4110  
Australia

email: [hitech@htsoft.com](mailto:hitech@htsoft.com)  
web: <http://microchip.htsoft.com>  
ftp: <ftp://www.htsoft.com>



# Contents

<b>Table of Contents</b>	<b>3</b>
<b>List of Tables</b>	<b>17</b>
<b>1 Introduction</b>	<b>19</b>
1.1 Typographic conventions	19
<b>2 PICC Command-line Driver</b>	<b>21</b>
2.1 Invoking the Compiler	22
2.1.1 Long Command Lines	23
2.2 The Compilation Sequence	24
2.2.1 Single-step Compilation	25
2.2.2 Generating Intermediate Files	26
2.2.3 Special Processing	27
2.2.3.1 Printf check	28
2.2.3.2 Assembly Code Requirements	28
2.3 Runtime Files	28
2.3.1 Library Files	29
2.3.1.1 Standard Libraries	30
2.3.2 Runtime Startup Module	30
2.3.2.1 Initialization of Data psects	31
2.3.2.2 Clearing the Bss Psects	32
2.3.3 The Powerup Routine	33
2.3.4 The <code>printf</code> Routine	33
2.4 Debugging Information	35
2.4.1 Output File Formats	35
2.4.2 Symbol Files	36
2.4.3 MPLAB-specific information	36

2.5	Compiler Messages	37
2.5.1	Messaging Overview	37
2.5.2	Message Language	38
2.5.3	Message Type	38
2.5.4	Message Format	39
2.5.5	Changing Message Behaviour	41
2.5.5.1	Disabling Messages	41
2.5.5.2	Changing Message Types	42
2.6	PICC Driver Option Descriptions	42
2.6.1	-C: Compile to Object File	43
2.6.2	-Dmacro: Define Macro	43
2.6.3	-Efile: Redirect Compiler Errors to a File	44
2.6.4	-Gfile: Generate Source-level Symbol File	44
2.6.5	-Ipath: Include Search Path	45
2.6.6	-Llibrary: Scan Library	45
2.6.7	-L-option: Adjust Linker Options Directly	46
2.6.8	-Mfile: Generate Map File	47
2.6.9	-Nsize: Identifier Length	48
2.6.10	-Ofile: Specify Output File	48
2.6.11	-P: Preprocess Assembly Files	48
2.6.12	-Q: Quiet Mode	48
2.6.13	-S: Compile to Assembler Code	49
2.6.14	-Umacro: Undefine a Macro	49
2.6.15	-V: Verbose Compile	49
2.6.16	-X: Strip Local Symbols	49
2.6.17	--ASMLIST: Generate Assembler .LST Files	50
2.6.18	--BANKQUAL=selection: Set Compiler Response to Bank Selection Qualifiers	50
2.6.19	--CALLGRAPH=type: Select call graph type	50
2.6.20	--CHECKSUM=start-end@destination<, specs>: Calculate a checksum	51
2.6.21	--CHIP=processor: Define Processor	51
2.6.22	--CHIPINFO: Display List of Supported Processors	52
2.6.23	--CODEOFFSET: Offset Program Code to Address	52
2.6.24	--CR=file: Generate Cross Reference Listing	52
2.6.25	--DEBUGGER=type: Select Debugger Type	52
2.6.26	--DOUBLE=type: Select kind of Double Types	53
2.6.27	--ECHO: Echo command line before processing	53
2.6.28	--ERRFORMAT=format: Define Format for Compiler Messages	53

2.6.29	--ERRORS= <i>number</i> : Maximum Number of Errors	53
2.6.30	--FILL= <i>opcode</i> : Fill Unused Program Memory	54
2.6.31	--FLOAT= <i>type</i> : Select kind of Float Types	54
2.6.32	--GETOPTION= <i>app, file</i> : Get Command-line Options	54
2.6.33	--HELP<= <i>option</i> >: Display Help	54
2.6.34	--IDE= <i>type</i> : Specify the IDE being used	54
2.6.35	--LANG= <i>language</i> : Specify the Language for Messages	55
2.6.36	--MEMMAP= <i>file</i> : Display Memory Map	55
2.6.37	--MSGDISABLE= <i>messagelist</i> : Disable Warning Messages	55
2.6.38	--MSGFORMAT= <i>format</i> : Set Advisory Message Format	56
2.6.39	--NODEL: Do not remove temporary files	56
2.6.40	--NOEXEC: Don't Execute Compiler	56
2.6.41	--OBJDIR: Specify a directory for intermediate files	56
2.6.42	--OPT<= <i>type</i> >: Invoke Compiler Optimizations	56
2.6.43	--OUTDIR: Specify a directory for output files	56
2.6.44	--OUTPUT= <i>type</i> : Specify Output File Type	57
2.6.45	--PASS1: Compile to P-code	58
2.6.46	--PRE: Produce Preprocessed Source Code	58
2.6.47	--PROTO: Generate Prototypes	58
2.6.48	--RAM= <i>lo-hi, &lt;lo-hi, ...&gt;</i> : Specify Additional RAM Ranges	59
2.6.49	--ROM= <i>lo-hi, &lt;lo-hi, ...&gt; tag</i> : Specify Additional ROM Ranges	60
2.6.50	--RUNTIME= <i>type</i> : Specify Runtime Environment	60
2.6.51	--SCANDEP: Scan for Dependencies	62
2.6.52	--SERIAL= <i>hexcode@address</i> : Store a Value at this Program Memory Address	62
2.6.53	--SETOPTION= <i>app, file</i> : Set The Command-line Options for Application	62
2.6.54	--STRICT: Strict ANSI Conformance	63
2.6.55	--SUMMARY= <i>type</i> : Select Memory Summary Output Type	63
2.6.56	--TIME: Report time taken for each phase of build process	63
2.6.57	--VER: Display The Compiler's Version Information	64
2.6.58	--WARN= <i>level</i> : Set Warning Level	64
2.6.59	--WARNFORMAT= <i>format</i> : Set Warning Message Format	64
<b>3</b>	<b>C Language Features</b>	<b>65</b>
3.1	ANSI Standard Issues	65
3.1.1	Implementation-defined behaviour	65
3.2	Processor-related Features	65
3.2.1	Stack	65

3.2.2	Configuration Fuses	66
3.2.3	ID Locations	66
3.2.4	Bit Instructions	67
3.2.5	EEPROM Access	67
3.2.5.1	The <i>eeeprom</i> variable qualifier	67
3.2.5.2	The <code>__EEPROM_DATA()</code> macro	68
3.2.5.3	EEPROM Access Functions	68
3.2.5.4	EEPROM Access Macros	69
3.2.6	Flash Runtime Access	70
3.2.6.1	Flash Access Macros	70
3.2.6.2	Flash Access Functions	71
3.2.7	Baseline PIC special instructions	71
3.2.7.1	The <code>OPTION</code> instruction	71
3.2.7.2	The <code>TRIS</code> instructions	71
3.2.7.3	Calibration Space	72
3.2.7.4	Oscillator calibration constants	72
3.3	Supported Data Types and Variables	73
3.3.1	Radix Specifiers and Constants	73
3.3.2	Bit Data Types and Variables	75
3.3.3	8-Bit Integer Data Types and Variables	76
3.3.4	16-Bit Integer Data Types	76
3.3.5	24-Bit Integer Data Types	77
3.3.6	32-Bit Integer Data Types and Variables	77
3.3.7	Floating Point Types and Variables	77
3.3.8	Structures and Unions	78
3.3.8.1	Bit-fields in Structures	79
3.3.8.2	Structure and Union Qualifiers	80
3.3.9	Standard Type Qualifiers	80
3.3.9.1	Const and Volatile Type Qualifiers	80
3.3.10	Special Type Qualifiers	81
3.3.10.1	Persistent Type Qualifier	81
3.3.10.2	Near Type Qualifier	82
3.3.10.3	Bank1, Bank2 and Bank3 Type Qualifiers	82
3.3.11	Eeprom Type Qualifier	82
3.3.12	Pointer Types	83
3.3.12.1	Combining Type Qualifiers and Pointers	83
3.3.12.2	Data Pointers	84
3.3.12.3	Pointers to Const	86
3.3.12.4	Pointers to Both Memory Spaces	87

3.4	Storage Class and Object Placement	88
3.4.1	Local Variables	88
3.4.1.1	Auto Variables	88
3.4.1.2	Static Variables	89
3.4.2	Absolute Variables	89
3.4.3	Objects in Program Space	89
3.5	Functions	90
3.5.1	Function Argument Passing	90
3.5.2	Function Return Values	91
3.5.2.1	8-Bit Return Values	91
3.5.2.2	16-bit and 32-bit values	91
3.5.2.3	Structure Return Values	91
3.6	Function Calling Convention	92
3.7	Operators	93
3.7.1	Integral Promotion	93
3.7.2	Shifts applied to integral types	94
3.7.3	Division and modulus with integral types	95
3.8	Psects	95
3.8.1	Compiler-generated Psects	95
3.9	Interrupt Handling in C	97
3.9.1	Interrupt Functions	97
3.9.1.1	Midrange Interrupt Functions	98
3.9.1.2	Context Saving on Interrupts	98
3.9.1.3	Midrange Context Saving	98
3.9.1.4	Context Restoration	99
3.9.2	Function Duplication	99
3.9.2.1	Implicit Calls to Library Routines	100
3.10	Mixing C and Assembler Code	100
3.10.1	External Assembly Language Functions	100
3.10.2	#asm, #endasm and asm()	102
3.10.3	Accessing C objects from within Assembly Code	103
3.10.3.1	Equivalent Assembly Symbols	103
3.10.3.2	Accessing special function register names from assembler	103
3.10.4	Interaction between Assembly and C Code	104
3.10.4.1	Absolute Psects	104
3.10.4.2	Undefined Symbols	105
3.11	Preprocessing	106
3.11.1	Preprocessor Directives	106
3.11.2	Predefined Macros	106



3.11.3	Pragma Directives . . . . .	106
3.11.3.1	The #pragma inline Directive . . . . .	106
3.11.3.2	The #pragma jis and nojis Directives . . . . .	109
3.11.3.3	The #pragma pack Directive . . . . .	109
3.11.3.4	The #pragma printf_check Directive . . . . .	110
3.11.3.5	The #pragma regsused Directive . . . . .	110
3.11.3.6	The #pragma switch Directive . . . . .	111
3.11.3.7	The #pragma warning Directive . . . . .	111
3.12	Linking Programs . . . . .	113
3.12.1	Replacing Library Modules . . . . .	114
3.12.2	Signature Checking . . . . .	114
3.12.3	Linker-Defined Symbols . . . . .	116
3.13	Standard I/O Functions and Serial I/O . . . . .	116
<b>4</b>	<b>Macro Assembler</b> . . . . .	<b>117</b>
4.1	Assembler Usage . . . . .	117
4.2	Assembler Options . . . . .	118
4.3	HI-TECH C Assembly Language . . . . .	120
4.3.1	Statement Formats . . . . .	120
4.3.2	Characters . . . . .	121
4.3.2.1	Delimiters . . . . .	121
4.3.2.2	Special Characters . . . . .	121
4.3.3	Comments . . . . .	121
4.3.3.1	Special Comment Strings . . . . .	121
4.3.4	Constants . . . . .	122
4.3.4.1	Numeric Constants . . . . .	122
4.3.4.2	Character Constants and Strings . . . . .	122
4.3.5	Identifiers . . . . .	122
4.3.5.1	Significance of Identifiers . . . . .	123
4.3.5.2	Assembler-Generated Identifiers . . . . .	123
4.3.5.3	Location Counter . . . . .	123
4.3.5.4	Register Symbols . . . . .	124
4.3.5.5	Symbolic Labels . . . . .	124
4.3.6	Expressions . . . . .	125
4.3.7	Program Sections . . . . .	125
4.3.8	Assembler Directives . . . . .	127
4.3.8.1	GLOBAL . . . . .	127
4.3.8.2	END . . . . .	127
4.3.8.3	PSECT . . . . .	129

4.3.8.4	ORG	131
4.3.8.5	EQU	131
4.3.8.6	SET	131
4.3.8.7	DB	132
4.3.8.8	DW	132
4.3.8.9	DS	132
4.3.8.10	DABS	132
4.3.8.11	FNADDR	133
4.3.8.12	FNARG	133
4.3.8.13	FNBREAK	133
4.3.8.14	FNCALL	134
4.3.8.15	FNCONF	134
4.3.8.16	FNINDIR	134
4.3.8.17	FNSIZE	135
4.3.8.18	FNROOT	135
4.3.8.19	IF, ELSIF, ELSE and ENDIF	135
4.3.8.20	MACRO and ENDM	136
4.3.8.21	LOCAL	137
4.3.8.22	ALIGN	138
4.3.8.23	REPT	138
4.3.8.24	IRP and IRPC	138
4.3.8.25	PROCESSOR	139
4.3.8.26	SIGNAT	139
4.3.9	Assembler Controls	140
4.3.9.1	COND	140
4.3.9.2	EXPAND	141
4.3.9.3	INCLUDE	141
4.3.9.4	LIST	141
4.3.9.5	NOCOND	141
4.3.9.6	NOEXPAND	142
4.3.9.7	NOLIST	142
4.3.9.8	NOXREF	142
4.3.9.9	PAGE	142
4.3.9.10	SPACE	142
4.3.9.11	SUBTITLE	142
4.3.9.12	TITLE	142
4.3.9.13	XREF	142

<b>5</b>	<b>Linker and Utilities</b>	<b>143</b>
5.1	Introduction	143
5.2	Relocation and Psects	143
5.3	Program Sections	144
5.4	Local Psects	144
5.5	Global Symbols	144
5.6	Link and load addresses	145
5.7	Operation	145
5.7.1	Numbers in linker options	146
5.7.2	<i>-Aclass=low-high,...</i>	147
5.7.3	<i>-Cx</i>	147
5.7.4	<i>-Cpsect=class</i>	148
5.7.5	<i>-Dclass=delta</i>	148
5.7.6	<i>-Dsymfile</i>	148
5.7.7	<i>-Eerrfile</i>	148
5.7.8	<i>-F</i>	148
5.7.9	<i>-Gspec</i>	148
5.7.10	<i>-Hsymfile</i>	149
5.7.11	<i>-H+symfile</i>	149
5.7.12	<i>-Jerrcount</i>	149
5.7.13	<i>-K</i>	150
5.7.14	<i>-I</i>	150
5.7.15	<i>-L</i>	150
5.7.16	<i>-LM</i>	150
5.7.17	<i>-Mmapfile</i>	150
5.7.18	<i>-N, -Ns and -Nc</i>	150
5.7.19	<i>-Ooutfile</i>	150
5.7.20	<i>-Pspec</i>	151
5.7.21	<i>-Qprocessor</i>	152
5.7.22	<i>-S</i>	152
5.7.23	<i>-Sclass=limit[, bound]</i>	152
5.7.24	<i>-Usymbol</i>	153
5.7.25	<i>-Vavmap</i>	153
5.7.26	<i>-Wnum</i>	153
5.7.27	<i>-X</i>	153
5.7.28	<i>-Z</i>	153
5.8	Invoking the Linker	154
5.9	Compiled Stack Operation	154
5.9.1	Parameters involving Function Calls	155

5.10	Map Files	157
5.10.1	Generation	157
5.10.2	Contents	157
5.10.2.1	General Information	158
5.10.2.2	Call Graph Information	159
5.10.2.3	Psect Information listed by Module	165
5.10.2.4	Psect Information listed by Class	167
5.10.2.5	Segment Listing	167
5.10.2.6	Unused Address Ranges	168
5.10.2.7	Symbol Table	168
5.11	Librarian	169
5.11.1	The Library Format	169
5.11.2	Using the Librarian	170
5.11.3	Examples	171
5.11.4	Supplying Arguments	171
5.11.5	Listing Format	172
5.11.6	Ordering of Libraries	172
5.11.7	Error Messages	172
5.12	Objtohex	172
5.12.1	Checksum Specifications	174
5.13	Cref	174
5.13.1	-Fprefix	175
5.13.2	-Hheading	175
5.13.3	-Llen	175
5.13.4	-Ooutfile	175
5.13.5	-Pwidth	176
5.13.6	-Sstoplist	176
5.13.7	-Xprefix	176
5.14	Cromwell	176
5.14.1	-Pname[,architecture]	176
5.14.2	-N	178
5.14.3	-D	178
5.14.4	-C	178
5.14.5	-F	178
5.14.6	-Okey	179
5.14.7	-Ikey	179
5.14.8	-L	179
5.14.9	-E	179
5.14.10	-B	179

5.14.11 -M	179
5.14.12 -V	179
5.15 Hexmate	180
5.15.1 Hexmate Command Line Options	181
5.15.1.1 specifications,filename.hex	182
5.15.1.2 + Prefix	182
5.15.1.3 -ADDRESSING	182
5.15.1.4 -BREAK	183
5.15.1.5 -CK	183
5.15.1.6 -FILL	184
5.15.1.7 -FIND	185
5.15.1.8 -FIND...,DELETE	186
5.15.1.9 -FIND...,REPLACE	186
5.15.1.10 -FORMAT	186
5.15.1.11 -HELP	187
5.15.1.12 -LOGFILE	188
5.15.1.13 -MASK	188
5.15.1.14 -Ofile	188
5.15.1.15 -SERIAL	188
5.15.1.16 -SIZE	189
5.15.1.17 -STRING	189
5.15.1.18 -STRPACK	190
<b>A Library Functions</b>	<b>191</b>
__CONFIG	192
__EEPROM_DATA	193
__IDLOC	194
__IDLOC7	195
__DELAY	196
ABS	197
ACOS	198
ASCTIME	199
ASIN	201
ASSERT	202
ATAN	203
ATAN2	204
ATOF	205
atoi	206
ATOL	207



BSEARCH	208
CEIL	210
CGETS	211
CLRWDT	213
COS	214
COSH	215
CPUTS	216
CTIME	217
DI	218
DIV	219
EEPROM_READ	220
EVAL_POLY	221
EXP	222
FABS	223
FLASH_COPY	224
FLASH_ERASE	226
FMOD	228
FLOOR	229
FREXP	230
FTOA	231
GETCH	232
GETCHAR	233
GETS	234
GET_CAL_DATA	235
GMTIME	236
ISALNUM	238
ISDIG	240
ITOA	241
LABS	242
LDEXP	243
LDIV	244
LOCALTIME	245
LOG	247
LONGJMP	248
LTOA	250
MEMCHR	251
MEMCMP	253
MEMCPY	255
MEMMOVE	257

---

MEMSET	258
MKTIME	259
MODF	261
POW	264
PUTCH	268
PUTCHAR	269
PUTS	271
QSORT	272
RAM_TEST_FAILED	274
RAND	275
ROUND	277
SETJMP	280
SIN	282
SPRINTF	283
SQRT	284
SRAND	285
STRCAT	286
STRCAT	287
STRCHR	289
STRCHR	291
STRCMP	293
STRCPY	295
STRCPY	296
STRCSPN	298
STRLEN	299
STRNCAT	300
STRNCAT	302
STRNCMP	304
STRNCPY	306
STRNCPY	308
STRPBRK	310
STRPBRK	311
STRRCHR	312
STRRCHR	313
STRSPN	315
STRSTR	316
STRSTR	317
STRTOD	318
STRTOL	320

STRTOK	322
STRTOK	324
TAN	326
TIME	327
TOLOWER	329
TRUNC	330
UDIV	331
ULDIV	332
UNGETCH	333
UTOA	334
VA_START	335
XTOI	337
<b>B Error and Warning Messages</b>	<b>339</b>
1...	339
138...	347
184...	353
226...	361
268...	370
311...	377
354...	385
398...	394
443...	399
487...	407
595...	414
668...	418
711...	423
757...	430
808...	437
855...	442
905...	448
971...	453
1032...	458
1118...	463
1237...	468
<b>C Chip Information</b>	<b>475</b>
<b>Index</b>	<b>481</b>



# List of Tables

2.1	PICC input file types	22
2.2	Support languages	38
2.3	Messaging environment variables	39
2.4	Messaging placeholders	40
2.5	Compiler responses to bank qualifiers	50
2.6	Default values for filling unprogrammed code space	51
2.7	Selectable debuggers	53
2.8	Floating point selections	54
2.9	Supported IDEs	55
2.10	Supported languages	55
2.11	Optimization Options	57
2.12	Output file formats	57
2.13	Runtime environment suboptions	61
2.14	Memory Summary Suboptions	63
3.1	Basic data types	73
3.2	Radix formats	74
3.3	Floating-point formats	78
3.4	Floating-point format example IEEE 754	78
3.5	Integral division	95
3.6	Preprocessor directives	107
3.7	Predefined macros	108
3.9	Pragma directives	109
3.10	Valid Register Names	110
3.11	switch types	111
3.12	Supported standard I/O functions	116



4.1	ASPIC command-line options	118
4.2	ASPICstatement formats	121
4.3	ASPIC numbers and bases	122
4.4	ASPIC operators	126
4.5	ASPIC assembler directives	128
4.6	PSECT flags	129
4.7	ASPIC assembler controls	140
4.8	LIST control options	141
5.1	Linker command-line options	145
5.1	Linker command-line options	146
5.2	Librarian command-line options	170
5.3	Librarian key letter commands	170
5.4	OBJTOHEX command-line options	173
5.5	CREF command-line options	175
5.6	CROMWELL format types	177
5.7	CROMWELL command-line options	177
5.8	-P option architecture arguments for COFF file output.	178
5.9	Hexmate command-line options	181
5.10	Hexmate Checksum Algorithm Selection	184
5.11	INHX types used in -FORMAT option	187
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	475
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	476
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	477
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	478
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	479
C.1	Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family	480

# Chapter 1

## Introduction

### 1.1 Typographic conventions

Different fonts and styles are used throughout this manual to indicate special words or text. Computer prompts, responses and filenames will be printed in `constant-spaced` type. When the filename is the name of a standard header file, the name will be enclosed in angle brackets, e.g. `<stdio.h>`. These header files can be found in the `INCLUDE` directory of your distribution.

Samples of code, C keywords or types, assembler instructions and labels will also be printed in a `constant-space` type. Assembler code is printed in a font similar to that used by C code.

Particularly useful points and new terms will be emphasized using *italicized type*. When part of a term requires substitution, that part should be printed in the appropriate font, but in *italics*. For example: `#include <filename.h>`.



## Chapter 2

# PICC Command-line Driver

PICC is the driver invoked from the command line to perform all aspects of compilation, including C code generation, assembly and link steps. It is the recommended way to use the compiler as it hides the complexity of all the internal applications used in the compilation process and provides a consistent interface for all compilation steps.

This chapter describes the steps the driver takes during compilation, files that the driver can accept and produce, as well as the command-line options that control the compiler's operation.



---

**WHAT IS “THE COMPILER”?** Throughout this manual, the term “the compiler” is used to refer to either all, or some subset of, the collection of applications that form the HI-TECH C PRO for the PIC10/12/16 MCU Family package. Often it is not important to know, for example, whether an action is performed by the parser or code generator application, and it is sufficient to say it was performed by “the compiler”.

It is also reasonable for “the compiler” to refer to the command-line driver (or just “driver”), PICC, as this is the application executed to invoke the compilation process. Following this view, “compiler options” should be considered command-line driver options, unless otherwise specified in this manual.

Similarly “compilation” refers to all, or some part of, the steps involved in generating source code into an executable binary image.

---

Table 2.1: PICC input file types

File Type	Meaning
.c	C source file
.p1	p-code file
.lpp	p-code library file
.as	Assembler source file
.obj	Relocatable object code file
.lib	Relocatable object library file
.hex	Intel HEX file

## 2.1 Invoking the Compiler

This chapter looks at how to use PICC as well as the tasks that it and the internal applications perform during compilation.

PICC has the following basic command format:

```
PICC [options] files [libraries]
```

It is conventional to supply *options* (identified by a leading *dash* “-” or *double dash* “-”) before the filenames, although this is not mandatory.

The formats of the options are discussed below in Section 2.6, and a detailed description of each option follows.

The *files* may be any mixture of C and assembler source files, and precompiled intermediate files, such as relocatable object (.obj) files or p-code (.p1) files. The order of the files is not important, except that it may affect the order in which code or data appears in memory, and may affect the name of some of the output files.

*Libraries* is a list of either object code or p-code library files that will be searched by the linker. The -L option, see Section 2.6.6, can also be used to specify library files to search.

PICC distinguishes source files, intermediate files and library files solely by the *file type* or *extension*. Recognized file types are listed in Table 2.1. This means, for example, that an assembler file must always have a .as extension. Alphabetic case of the extension is not important from the compiler’s point of view.



**MODULES AND SOURCE FILES:** A *C source file* is a file on disk that contains all or part of a program. C source files are initially passed to the preprocessor by the driver. A *module* is the output of the preprocessor, for a given source file, after inclusion of any header files (or other source files) which are specified by `#include` preprocessor



directives. These modules are then passed to the remainder of the compiler applications. Thus, a module may consist of several source and header files. A module is also often referred to as a *translation unit*. These terms can also be applied to assembly files, as they too can include other header and source files.

---

Some of the compiler's output files contain project-wide information and are not directly associated with any one particular input file, e.g. the map file. If the names of these project-wide files are not specified on the command line, the basename of these files is derived from the first C source file listed on the command line. If there are no files of this type being compiled, the name is based on the first input file (regardless of type) on the command line. Throughout this manual, the basename of this file will be called the *project name*.

Most IDEs use project files whose names are user-specified. Typically the names of project-wide files, such as map files, are named after the project, however check the manual for the IDE you are using for more details.

### 2.1.1 Long Command Lines

The PICC driver is capable of processing command lines exceeding any operating system limitation. To do this, the driver may be passed options via a command file. The command file is read by using the @ symbol which should be immediately followed (i.e. no intermediate space character) by the name of the file containing the command line arguments.

The file may contain blank lines, which are simply skipped by the driver. The command-line arguments may be placed over several lines by using a *space* and *backslash* character for all non-blank lines, except for the last line.

The use of a command file means that compiler options and project filenames can be stored along with the project, making them more easily accessible and permanently recorded for future use.

---

#### TUTORIAL

---

**USING COMMAND FILES** A command file `xyz.cmd` is constructed with your favorite text editor and contains both the options and file names that are required to compile your project as follows:

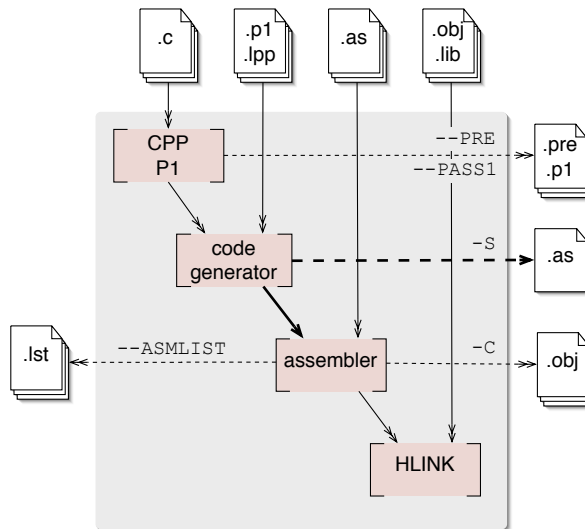
```
--chip=16F877A -m \  
--opt=all -g \  
main.c isr.c
```

After it is saved, the compiler may be invoked with the command:

```
PICC @xyz.cmd
```

---

Figure 2.1: Flow diagram of the initial compilation sequence



## 2.2 The Compilation Sequence

PICC will check each file argument and perform appropriate actions on each file. The entire compilation sequence can be thought of as the initial sequence up to the link stage, and the final sequence which takes in the link step and any post link steps required.

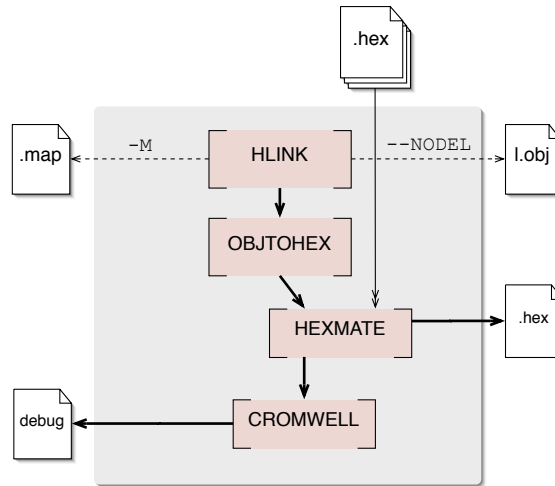
Graphically the compilation steps up to the link stage are illustrated in Figure 2.1. This diagram shows all possible input files along the top; intermediate and transitional files, along the right side; and useful compiler output files along the left. Generated files are shown along with the options that are used to generate and preserve these. All the files shown on the right, can be generated and fed to the compiler in a subsequent compile step; those on the left are used for debug purposes and cannot be used as an input to any subsequent compilation.

The individual compiler applications are shown as boxes. The C preprocessor, CPP, and parser, P1, have been grouped together for clarity.

The thin, multi-arrowed lines indicate the flow of multiple files — one for each file being processed by the relevant application. The thick single-arrowed lines indicate a single file for the project being compiled. Thus, for example, when using the `--PASS1` driver option, the parser produces one `.p1` file for each C source file that is being compiled as part of the project, but the code generator produces only one `.as` file from all `.c`, `.p1` and `.lpp` input files which it is passed.

Dotted lines indicate a process that may require an option to create or preserve the indicated file.

Figure 2.2: Flow diagram of the final compilation sequence



The link and post-link steps are graphically illustrated in Figure 2.2.

This diagram shows `.hex` files as additional input file type not considered in the initial compilation sequence. These files can be merged into the `.hex` file generated from the other input files in the project by an application called `HEXMATE`. See Section 5.15 for more information on this utility.

The output of the linker is a single absolute object file, called `l.obj`, that can be preserved by using the `--NODEL` driver option. Without this option, this temporary file is used to generate an output file (e.g. a `HEX` file) and files used for debugging by development tools (e.g. `COFF` files) before it is deleted. The file `l.obj` can be used as the input to `OBJTOHEX` if running this application manually, but it cannot be passed to the driver as an input file as it absolute and cannot be further processed.

### 2.2.1 Single-step Compilation

The command-line driver, `PICC`, can compile any mix of input files in a single step. All source files will be re-compiled regardless of whether they have been changes since that last time a compilation was performed.

Unless otherwise specified, a default output file and debug file are produced. All intermediate files (`.p1` and `.obj`) remain after compilation has completed, but all other transitional files are deleted, unless you use the `--NODEL` option which preserves all generated files. Note some generated files may be in a temporary directory not associated with your project and use a pseudo-