gputils 1.4.0

James Bowman, Craig Franklin, David Barnett, Borut Ražem and Molnár Károly

Sep 27, 2014

Contents

1 Introduction **3**
- 1.1 Tool Flows . . . 3
  - 1.1.1 Absolute Asm Mode . . . 3
  - 1.1.2 Relocatable Asm Mode . . . 3
  - 1.1.3 Which Tool Flow is best? . . . 4
- 1.2 Supported processors . . . 4

2 gpasm **8**
- 2.1 Running gpasm . . . 8
  - 2.1.1 MPASM(X) compatibility mode . . . 10
  - 2.1.2 Using gpasm with "make" . . . 10
  - 2.1.3 Dealing with errors . . . 10
- 2.2 Syntax . . . 10
  - 2.2.1 File structure . . . 10
  - 2.2.2 Expressions . . . 11
  - 2.2.3 Numbers . . . 12
  - 2.2.4 Preprocessor . . . 13
  - 2.2.5 Processor header files . . . 13
  - 2.2.6 Predefined constants . . . 14
- 2.3 Directives . . . 15
  - 2.3.1 Code generation . . . 15
  - 2.3.2 Configuration . . . 15
  - 2.3.3 Conditional assembly . . . 15
  - 2.3.4 Macros . . . 15
  - 2.3.5 $ . . . 16
  - 2.3.6 Suggestions for structuring your code . . . 17
  - 2.3.7 Directive summary . . . 17
  - 2.3.8 High level extensions . . . 28
- 2.4 Instructions . . . 31
  - 2.4.1 Instruction set summary . . . 31
- 2.5 Errors/Warnings/Messages . . . 35
  - 2.5.1 Errors . . . 36
  - 2.5.2 Warnings . . . 38
  - 2.5.3 Messages . . . 40

3 gplink **41**
3.1 Running gplink . . . 41
3.2 MPLINK compatibility mode . . . 42
3.3 gplink outputs . . . 42
3.4 Linker scripts . . . 42
3.5 Stacks . . . 43
3.6 Optimization . . . 43
3.6.1 Level 0 . . . 43
3.6.2 Level 1 (default) . . . 43
3.6.3 Level 2 . . . 43
3.6.4 Level 3 . . . 43

4 gplib **44**
4.1 Running gplib . . . 44
4.2 Creating an archive . . . 44
4.3 Other gplib operations . . . 45
4.4 Archive format . . . 45

5 Utilities **46**
5.1 gpdasm . . . 46
5.1.1 Running gpdasm . . . 46
5.1.2 Comments on Disassembling . . . 47
5.2 gpstrip . . . 48
5.2.1 Running gpstrip . . . 48
5.3 gpvc . . . 48
5.3.1 Running gpvc . . . 48
5.4 gpvo . . . 49
5.4.1 Running gpvo . . . 49

Chapter 1

Introduction

gputils is a collection of tools for Microchip (TM) PIC microcontrollers. It includes gpasm, gplink, and gplib. Each tool is intended to be an open source replacement for a corresponding Microchip (TM) tool. This manual covers the basics of running the tools. For more details on a microcontroller, consult the manual for the specific PICmicro product that you are using.

This document is part of gputils.

gputils is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

gputils is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with gputils; see the file COPYING. If not, write to the Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

1.1 Tool Flows

gputils can be used in two different ways: absolute asm mode and relocatable asm mode.

1.1.1 Absolute Asm Mode

In absolute asm mode, an assembly language source file is directly converted into a hex file by gpasm. This method is absolute because the final addresses are hard coded into the source file.

1.1.2 Relocatable Asm Mode

In relocatable asm mode, the microcontroller assembly source code is divided into separate modules. Each module is assembled into an object using gpasm. That object can be placed "anywhere" in microcontroller's memory. Then gplink is used to resolve symbols references, assign final address, and to patch the machine code with the final addresses. The output from gplink is an absolute executable object.

1.1.3 Which Tool Flow is best?

Absolute mode is simple to understand and to use. It only requires one tool, gpasm. Most of the examples on Microchip's website use absolute mode. So why use relocatable mode?

- Code can be written without regard to addresses. This makes it easier to write and reuse.
- The objects can be archived to create a library, which also simplifies reuse.
- Recompiling a project can be faster, because you only compile the portions that have changed.
- Files can have local name spaces. The user chooses what symbols are global.

Most develpment tools use relocatable objects for these reasons. The few that don't are generally microcontroller tools. Their applications are so small that absolute mode isn't impractical. For PICs, relocatable mode has one big disadvantage. The bank and page control is a challenge.

1.2 Supported processors

gputils currently supports most processors supported by MPLABX v2.10. This includes the following processors:

p10f200	p10f202	p10f204	p10f206	p10f220	p10f222
p10f320	p10f322	p10lf320	p10lf322	p12c508	p12c508a
p12c509	p12c509a	p12c671	p12c672	p12ce518	p12ce519
p12ce673	p12ce674	p12cr509a	p12f508	p12f509	p12f510
p12f519	p12f520	p12f529t39a	p12f529t48a	p12f609	p12f615
p12f617	p12f629	p12f635	p12f675	p12f683	p12f752
p12f1501	p12f1571	p12f1572	p12f1612	p12f1822	p12f1840
p12hv609	p12hv615	p12hv752	p12lf1501	p12lf1571	p12lf1572
p12lf1552	p12lf1612	p12lf1822	p12lf1840	p12lf1840t39a	p12lf1840t48a
p16c5x	p16c52	p16c54	p16c54a	p16c54b	p16c54c
p16c55	p16c55a	p16c56	p16c56a	p16c57	p16c57c
p16c58a	p16c58b	p16c61	p16c62	p16c62a	p16c62b
p16c63	p16c63a	p16c64	p16c64a	p16c65	p16c65a
p16c65b	p16c66	p16c67	p16c71	p16c72	p16c72a
p16c73	p16c73a	p16c73b	p16c74	p16c74a	p16c74b
p16c76	p16c77	p16c84	p16c432	p16c433	p16c505
p16c554	p16c557	p16c558	p16c620	p16c620a	p16c621
p16c621a	p16c622	p16c622a	p16c642	p16c662	p16c710
p16c711	p16c712	p16c715	p16c716	p16c717	p16c745
p16c765	p16c770	p16c771	p16c773	p16c774	p16c781
p16c782	p16c923	p16c924	p16c925	p16c926	p16ce623
p16ce624	p16ce625	p16cr54	p16cr54a	p16cr54b	p16cr54c
p16cr56a	p16cr57a	p16cr57b	p16cr57c	p16cr58a	p16cr58b
p16cr62	p16cr63	p16cr64	p16cr65	p16cr72	p16cr83

p16cr84	p16cr620a	p16cxx			
p16f54	p16f57	p16f59	p16f72	p16f73	p16f74
p16f76	p16f77	p16f83	p16f84	p16f84a	p16f87
p16f88	p16f505	p16f506	p16f526	p16f527	p16f570
p16f610	p16f616	p16f627	p16f627a	p16f628	p16f628a
p16f630	p16f631	p16f636	p16f639	p16f648a	p16f676
p16f677	p16f684	p16f685	p16f687	p16f688	p16f689
p16f690	p16f707	p16f716	p16f720	p16f721	p16f722
p16f722a	p16f723	p16f723a	p16f724	p16f726	p16f727
p16f737	p16f747	p16f753	p16f767	p16f777	p16f785
p16f818	p16f819	p16f870	p16f871	p16f872	p16f873
p16f873a	p16f874	p16f874a	p16f876	p16f876a	p16f877
p16f877a	p16f882	p16f883	p16f884	p16f886	p16f887
p16f913	p16f914	p16f916	p16f917	p16f946	p16f1454
p16f1455	p16f1458	p16f1459	p16f1503	p16f1507	p16f1508
p16f1509	p16f1512	p16f1513	p16f1516	p16f1517	p16f1518
p16f1519	p16f1526	p16f1527	p16f1613	p16f1614	p16f1618
p16f1703	p16f1704	p16f1705	p16f1707	p16f1708	p16f1709
p16f1713	p16f1716	p16f1717	p16f1718	p16f1719	p16f1782
p16f1783	p16f1784	p16f1786	p16f1787	p16f1788	p16f1789
p16f1823	p16f1824	p16f1825	p16f1826	p16f1827	p16f1828
p16f1829	p16f1829lin	p16f1847	p16f1933	p16f1934	p16f1936
p16f1937	p16f1938	p16f1939	p16f1946	p16f1947	p16f1xxx
p16hv540	p16hv610	p16hv616	p16hv753	p16hv785	
p16lf707	p16lf720	p16lf721	p16lf722	p16lf722a	p16lf723
p16lf723a	p16lf724	p16lf726	p16lf727	p16lf1454	p16lf1455
p16lf1458	p16lf1459	p16lf1503	p16lf1507	p16lf1508	p16lf1509
p16lf1512	p16lf1513	p16lf1516	p16lf1517	p16lf1518	p16lf1519
p16lf1526	p16lf1527	p16lf1554	p16lf1559	p16lf1613	p16lf1614
p16lf1618	p16lf1619	p16lf1703	p16lf1704	p16lf1705	p16lf1707
p16lf1708	p16lf1709	p16lf1713	p16lf1716	p16lf1717	p16lf1718
p16lf1719	p16lf1782	p16lf1783	p16lf1784	p16lf1786	p16lf1787
p16lf1788	p16lf1789	p16lf1823	p16lf1824	p16lf1824t39a	p16lf1825
p16lf1826	p16lf1827	p16lf1828	p16lf1829	p16lf1847	p16lf1902
p16lf1903	p16lf1904	p16lf1906	p16lf1907	p16lf1933	p16lf1934
p16lf1936	p16lf1937	p16lf1938	p16lf1939	p16lf1946	p16lf1947
p16lf18313	p16lf18323				
p17c42	p17c42a	p17c43	p17c44	p17c752	p17c756
p17c756a	p17c762	p17c766	p17cr42	p17cr43	p17cxx
p18c242	p18c252	p18c442	p18c452	p18c601	p18c658
p18c801	p18c858	p18cxx			
p18f13k22	p18f13k50	p18f14k22	p18f14k22lin	p18f14k50	p18f23k20
p18f23k22	p18f24j10	p18f24j11	p18f24j50	p18f24k20	p18f24k22
p18f24k50	p18f25j10	p18f25j11	p18f25j50	p18f25k20	p18f25k22
p18f25k50	p18f25k80	p18f26j11	p18f26j13	p18f26j50	p18f26j53
p18f26k20	p18f26k22	p18f26k80	p18f27j13	p18f27j53	p18f43k20

```
p18f43k22    p18f44j10    p18f44j11    p18f44j50    p18f44k20    p18f44k22
p18f45j10    p18f45j11    p18f45j50    p18f45k20    p18f45k22    p18f45k50
p18f45k80    p18f46j11    p18f46j13    p18f46j50    p18f46j53    p18f46k20
p18f46k22    p18f46k80    p18f47j13    p18f47j53    p18f63j11    p18f63j90
p18f64j11    p18f64j15    p18f64j90    p18f65j10    p18f65j11    p18f65j15
p18f65j50    p18f65j90    p18f65j94    p18f65k22    p18f65k80    p18f65k90
p18f66j10    p18f66j11    p18f66j15    p18f66j16    p18f66j50    p18f66j55
p18f66j60    p18f66j65    p18f66j90    p18f66j93    p18f66j94    p18f66j99
p18f66k22    p18f66k80    p18f66k90    p18f67j10    p18f67j11    p18f67j50
p18f67j60    p18f67j90    p18f67j93    p18f67j94    p18f67k22    p18f67k90
p18f83j11    p18f83j90    p18f84j11    p18f84j15    p18f84j90    p18f85j10
p18f85j11    p18f85j15    p18f85j50    p18f85j90    p18f85j94    p18f85k22
p18f85k90    p18f86j10    p18f86j11    p18f86j15    p18f86j16    p18f86j50
p18f86j55    p18f86j60    p18f86j65    p18f86j72    p18f86j90    p18f86j93
p18f86j94    p18f86j99    p18f86k22    p18f86k90    p18f87j10    p18f87j11
p18f87j50    p18f87j60    p18f87j72    p18f87j90    p18f87j93    p18f87j94
p18f87k22    p18f87k90    p18f95j94    p18f96j60    p18f96j65    p18f96j94
p18f96j99    p18f97j60    p18f97j94    p18f242      p18f248      p18f252
p18f258      p18f442      p18f448      p18f452      p18f458      p18f1220
p18f1230     p18f1320     p18f1330     p18f2220     p18f2221     p18f2320
p18f2321     p18f2331     p18f2410     p18f2420     p18f2423     p18f2431
p18f2439     p18f2450     p18f2455     p18f2458     p18f2480     p18f2510
p18f2515     p18f2520     p18f2523     p18f2525     p18f2539     p18f2550
p18f2553     p18f2580     p18f2585     p18f2610     p18f2620     p18f2680
p18f2681     p18f2682     p18f2685     p18f4220     p18f4221     p18f4320
p18f4321     p18f4331     p18f4410     p18f4420     p18f4423     p18f4431
p18f4439     p18f4450     p18f4455     p18f4458     p18f4480     p18f4510
p18f4515     p18f4520     p18f4523     p18f4525     p18f4539     p18f4550
p18f4553     p18f4580     p18f4585     p18f4610     p18f4620     p18f4680
p18f4681     p18f4682     p18f4685     p18f6310     p18f6390     p18f6393
p18f6410     p18f6490     p18f6493     p18f6520     p18f6525     p18f6527
p18f6585     p18f6620     p18f6621     p18f6622     p18f6627     p18f6628
p18f6680     p18f6720     p18f6722     p18f6723     p18f8310     p18f8390
p18f8393     p18f8410     p18f8490     p18f8493     p18f8520     p18f8525
p18f8527     p18f8585     p18f8620     p18f8621     p18f8622     p18f8627
p18f8628     p18f8680     p18f8720     p18f8722     p18f8723
p18lf13k22   p18lf13k50   p18lf14k22   p18lf14k50   p18lf23k22   p18lf24j10
p18lf24j11   p18lf24j50   p18lf24k22   p18lf24k50   p18lf25j10   p18lf25j11
p18lf25j50   p18lf25k22   p18lf25k50   p18lf25k80   p18lf26j11   p18lf26j13
p18lf26j50   p18lf26j53   p18lf26k22   p18lf26k80   p18lf27j13   p18lf27j53
p18lf43k22   p18lf44j10   p18lf44j11   p18lf44j50   p18lf44k22   p18lf45j10
p18lf45j11   p18lf45j50   p18lf45k22   p18lf45k50   p18lf45k80   p18lf46j11
p18lf46j13   p18lf46j50   p18lf46j53   p18lf46k22   p18lf46k80   p18lf47j13
p18lf47j53   p18lf65k80   p18lf66k80
p14000
eeprom8      eeprom16
```

```
gen
hcs1365         hcs1370
mcv08a          mcv14a          mcv18a          mcv28a
ps500           ps810
rf509af         rf509ag         rf675f          rf675h          rf675k
sx18            sx20            sx28            sx48            sx52
```

Chapter 2

gpasm

2.1 Running gpasm

The general syntax for running gpasm is

```
gpasm [options] asm-file
```

Where options can be one of:

Option	Long option	Meaning
-a <format>	–hex-format <format>	Produce hex file in one of four formats: inhx8 (the default).
-c	–object	Output a relocatable object in the older versio format.
-C	–old-coff	Output a relocatable object in the old Microch
-d	–debug	Output debug messages.
-D symbol[=value]	–define symbol[=value]	Equivalent to "#define <symbol> <value>".
-e [ON|OFF]	–expand [ON|OFF]	Expand macros in listing file.
-f	–full-address	Show full address in .lst file at the memory ma
-g	–debug-info	Use debug directives for COFF.
-h	–help	Display the help message.
-i	–ignore-case	Ignore case in source code. By default gpas "FOOYA" as being different.
-j	–sdcc-dev16-list	Help to the extension of the pic16devices.txt f ing by itself, displays the all '16e' devices. A shows only the specified device.
-I <directory>	–include <directory>	Specify an include directory.
-k	–error	Enables the creation of the error file.
-l[12[ce]|14[ce]|16[ce]]	–list-chips[=(12[ce]|14[ce]|16[ce]])]	List the supported processors based on various
-L	–force-list	Ignore nolist directives.
-m	–dump	Memory dump.
	–mpasm-compatible	MPASM(X) compatibility mode.
-M	–deps	Output a dependency file.
-n	–dos	Use DOS style newlines (CRLF) in hex file. win32 systems.
-o <file>	–output<file>	Alternate name of output files. Option effect c – If the "-c" option included in the command l file.*o*, file.*lst*, file.*err* (The "file.*o*" should specified.) – If the "-c" option *not* included in the comma file.*hex*, file.*cod*, file.*lst*, file.*err* (The "file.*hex*" should specified.)
-p <processor>	–processor <processor>	Select target processor.
-P <file>	–preprocess <file>	Write preprocessed asm file to <file>.
-q	–quiet	Quiet.
-r <radix>	–radix <radix>	Set the radix, i.e. the number base that gpas numbers.<radix> can be one of "oct", "dec" ten, and sixteen respectively. Default is "hex"
-s[12[ce]|14[ce]|16[ce]]	–list-processor-properties[=([12[ce]|14[ce]|16[ce]])]	Lists properties of the processors. Using by its or group of the devices. Along with the '-p specified device.
-u	–absolute	Use absolute paths.
-v	–version	Print gpasm version information and exit.
-w [0 | 1 | 2]	–warning [0|1|2]	Set the message level.
-y	–extended	Enable 18xx extended mode.

Unless otherwise specified, gpasm removes the ".asm" suffix from its input file, replacing it with ".lst" and ".hex" for the list and hex output files respectively. On most modern operating systems case is significant in filenames. For this reason you should ensure that filenames are named consistently, and that the ".asm" suffix on any source file is in lower case.

gpasm always produces a ".lst" file. If it runs without errors, it also produces a ".hex" file or a ".o" file.

2.1.1 MPASM(X) compatibility mode

When the –mpasm-compatibe command line option is specified gpasm tries to mimic MPASM(X) behaviour:

- code_pack section doesn't generate line symbols in coff file; the consequence is missing object file info in the generated linker listing file.
- x << n is actually x << (n % (sizeof(int) * 8)) * on x86 architectures, so 0x1234 << 32 results 0x1234 which is wrong but compatible with MPASM(X)

2.1.2 Using gpasm with "make"

On most operating systems, you can build a project using the make utility. To use gpasm with make, you might have a "makefile" like this:

```
tree.hex: tree.asm treedef.inc
        gpasm tree.asm
```

This will rebuild "tree.hex" whenever either of the "tree.asm" or "treedef.inc" files change. A more comprehensive example of using gpasm with makefiles is included as example1 in the gpasm source distribution.

2.1.3 Dealing with errors

gpasm doesn't specifically create an error file. This can be a problem if you want to keep a record of errors, or if your assembly produces so many errors that they scroll off the screen. To deal with this if your shell is "sh", "bash" or "ksh", you can do something like:

```
gpasm tree.asm 2>&1 | tee tree.err
```

This redirects standard error to standard output ("2>&1"), then pipes this output into "tee", which copies it input to "tree.err", and then displays it.

2.2 Syntax

2.2.1 File structure

The gpasm source files consist of a series of lines. Lines can contain a label (starting in column 1) or an operation (starting in any column after 1), both, or neither. Comments follow a ";" character, and are

treated as a newline. Labels may be any series of the letters A-z, digits 0-9, and the underscore ("_"); they may not begin with a digit. Labels may be followed by a colon (":").

An operation is a single identifier (the same rules as for a label above) followed by a space, and a comma-separated list of parameters. For example, the following are all legal source lines:

```
                              ; Blank line
loop    sleep                 ; Label and operation
        incf    6,1           ; Operation with 2 parameters
        goto    loop          ; Operation with 1 parameter
```

2.2.2 Expressions

The gpasm expressions are implemented using 32-bit arithmetic. gpasm supports a full set of operators, based on the C operator set. The operators in the following table are arranged in groups of equal precedence, but the groups are arranged in order of increasing precedence. When gpasm encounters operators of equal precedence, it always evaluates from left to right.

Operator	Description
=	assignment
||	logical or
&&	logical and
&	bitwise and
|	bitwise or
^	bitwise exclusive-or
<	less than
>	greater than
==	equals
!=	not equals
>=	greater than or equal
<=	less than or equal
<<	left shift
>>	right shift
+	addition
-	subtraction
*	multiplication
/	division
%	modulo
UPPER	upper byte
HIGH	high byte
LOW	low byte
-	negation
!	logical not
~	bitwise no

Any symbol appearing in column 1 may be assigned a value using the assignment operator (=) in the previous table. Additionally, any value previously assigned may be modified using one of the operators in the table below. Each of these operators evaluates the current value of the symbol and then assigns a new value based on the operator.

Operator	Description
=	assignment
++	increment by 1
–	decrement by 1
+=	increment
-=	decrement
*=	multiply
/=	divide
%=	modulo
<<=	left shift
>>=	right shift
&=	bitwise and
|=	bitwise or
^=	bitwise exclusive-or

2.2.3 Numbers

gpasm gives you several ways of specifying numbers. You can use a syntax that uses an initial character to indicate the number's base. The following table summarizes the alternatives. Note the C-style option for specifying hexadecimal numbers.

base	general syntax	21 decimal written as
binary	B'[01]*'	B'10101'
octal	O'[0-7]*'	O'25'
decimal	D'[0-9]*'	D'21'
hex	H'[0-F]*'	H'15'
hex	0x[0-F]*	0x15

When you write a number without a specifying prefix such as "45", gpasm uses the current radix (base) to interpret the number. You can change this radix with the RADIX directive, or with the "-r" option on gpasm's command-line. The default radix is hexadecimal.

If you do not start hexadecimal numbers with a digit, gpasm will attempt to interpret what you've written as an identifier. For example, instead of writing C2, write either 0C2, 0xC2 or H'C2'.

Case is not significant when interpreting numbers: 0ca, 0CA, h'CA' and H'ca' are all equivalent.

Several legacy mpasm number formats are also supported. These formats have various shortcomings, but are still supported. The table below summarizes them.

base	general syntax	21 decimal written as
binary	[01]*b	10101b
octal	q'[0-7]*'	q'25'
octal	[0-7]*o	25o
octal	[0-7]*q	25q
decimal	0-9]*d	21d
decimal	.[0-9]*	.21
hex	[0-F]*h	15h

You can write the ASCII code for a character X using 'X', or A'X'.

2.2.4 Preprocessor

A line such as:

```
include foo.inc
```

Will make gpasm fetch source lines from the file "foo.inc" until the end of the file, and then return to the original source file at the line following the include.

Lines beginning with a "#" are preprocessor directives, and are treated differently by gpasm. They may contain a "#define", or a "#undefine" directive.

Once gpasm has processed a line such as:

```
#define X Y
```

Every subsequent occurrence of X is replaced with Y, until the end of file or a line

```
#undefine X
```

appears. The preprocessor will replace an occurance of #v(expression) in a symbol with the value of "expression" in decimal. In the following expression:

```
number equ 5
label_#v((number + 1) * 5)_suffix equ 0x10
```

The gpasm will place the symbol "label_30_suffix" with a value of 0x10 in the symbol table. The preprocessor in gpasm is only *like* the C preprocessor; its syntax and semantic is rather different from that of the C preprocessor.

2.2.5 Processor header files

gputils distributes the Microchip processor header files. These files contain processor specific data that is helpful in developing PIC applications. The location of these files is reported in the gpasm help message. Use the INCLUDE directive to utilize the appropriate file in your source code. Only the name of the file is required. gpasm will search the default path automatically.

2.2.6 Predefined constants

The device ID:

```
__18F4520 -- If the type of the processor is p18f4520.
```

The following constants exist *only* in the "gpasm" mode (if the command line is *not* included the "–mpasm-compatible" option).

End address of the Lower Access RAM area (*only* in the Extended 16 bit core: pic18xxx):

```
__ACC_RAM_LOW_END  -- e.g. 0x5F, 0x7F
```

The bounds of the Common RAM area, *if exist* this memory (in the 12 and 14 bit core: pic12xxx, pic16xxx):
(This exists, only in the bank 0, the same can be achieved from other banks.)

```
__COMMON_RAM_START -- Start address of the Common RAM area   (e.g. 0x07, 0x0C, 0x70)
                      (The implementation of the Common RAM inconsistent in some old processor
__COMMON_RAM_END   -- End address of the Common RAM area     (e.g. 0x0F, 0x2F, 0x7F)
                      (The implementation of the Common RAM inconsistent in some old processor
__COMMON_RAM_MAX   -- Maximum address of the Common RAM area (e.g. 0x6F) (This rarely exists.)
```

The bounds of the Linear RAM area, *if exist* this memory (*only* in the Enhanced 14 bit core: pic16xxx):

```
__LINEAR_RAM_START -- Start address of the Linear RAM area (e.g. 0x2000, 0x2000)
__LINEAR_RAM_END   -- End address of the Linear RAM area   (e.g. 0x202F, 0x23EF)
```

The bounds of the EEPROM area, *if exist* this memory:

```
__EEPROM_START     -- Start address of the EEPROM area (e.g. 0x0400, 0x2100, 0xF000, 0xF00000)
__EEPROM_END       -- End address of the EEPROM area   (e.g. 0x043F, 0x21FF, 0xF0FF, 0xF000FF)
```

The bounds of the program memory:

```
__CODE_START       -- This is the first address where may be still placed a code.
__CODE_END         -- This is the last address where may be still placed a code.
```

Addresses of the vectors:

```
__VECTOR_RESET     -- Address of the Reset Vector (all pic)
__VECTOR_INT       -- Address of the Interrupt Vector (14 bit core: pic12xxx, pic16xxx)
__VECTOR_INT_EXT   -- Address of the External Interrupt Vector (pic17xxx)
__VECTOR_INT_TMR0  -- Address of the Timer0 Interrupt Vector (16 bit core: pic17xxx)
__VECTOR_INT_T0CKI -- Address of the T0CKI Pin Interrupt Vector (16 bit core: pic17xxx)
__VECTOR_INT_PERI  -- Address of the Peripheral Interrupt Vector (16 bit core: pic17xxx)
__VECTOR_INT_HIGH  -- Address of the High Priority Interrupt Vector (Extended 16 bit core: pic
__VECTOR_INT_LOW   -- Address of the Low Priority Interrupt Vector (Extended 16 bit core: pic1
```

2.3 Directives

2.3.1 Code generation

In absolute mode, use the ORG directive to set the PIC memory location where gpasm will start assembling code. If you don't specify an address with ORG, gpasm assumes 0x0000. In relocatable mode, use the CODE directive.

2.3.2 Configuration

You can choose the fuse settings for your PIC implementation using the __CONFIG, __FUSES and CONFIG (PIC18 MCUs) directive, so that the hex file set the fuses explicitly. Naturally you should make sure that these settings match your PIC hardware design. CONFIG and __CONFIG directives cannot both be used in the same project. __CONFIG and __FUSES directives are deprecated for 16-bit PIC MCU devices (PIC18FXXX), CONFIG directive should be used instead.

In the "gpasm" mode (if the command line is *not* included the "–mpasm-compatible" option), in the case of each processor (PIC12XXX, PIC14XXX, PIC18XXX) it is possible to use the CONFIG directive.

The __MAXRAM and __BADRAM directives specify which RAM locations are legal. These directives are mostly used in processor-specific configuration files.

2.3.3 Conditional assembly

The IF, IFNDEF, IFDEF, ELSE and ENDIF directives enable you to assemble certain sections of code only if a condition is met. In themselves, they do not cause gpasm to emit any PIC code. The example in section 2.3.4 for demonstrates conditional assembly.

2.3.4 Macros

gpasm supports a simple macro scheme; you can define and use macros like this:

```
any     macro parm
        movlw parm
        endm
...
        any   33
```

A more useful example of some macros in use is:

```
; Shift reg left
slf     macro   reg
        clrc
        rlf     reg,f
endm

; Scale W by ‘‘factor’’.  Result in ‘‘reg’’, W unchanged.
scale   macro   reg, factor
        if (factor == 1)
```

```
            movwf reg                ; 1 X is easy
        else
            scale   reg, (factor / 2) ; W * (factor / 2)
            slf     reg,f             ; double reg
            if ((factor & 1) == 1)    ; if lo-bit set ..
                addwf   reg,f         ; .. add W to reg
            endif
        endif
endm
```

This recursive macro generates code to multiply W by a constant “factor”, and stores the result in “reg”. So writing:

```
        scale   tmp,D'10'
```

is the same as writing:

```
        movwf   tmp      ; tmp = W
        clrc
        rlf     tmp,f    ; tmp = 2 * W
        clrc
        rlf     tmp,f    ; tmp = 4 * W
        addwf   tmp,f    ; tmp = (4 * W) + W = 5 * W
        clrc
        rlf     tmp,f    ; tmp = 10 * W
```

2.3.5 $

$ expands to the address of the instruction currently being assembled. If it’s used in a context other than an instruction, such as a conditional, it expands to the address the next instruction would occupy, since the assembler’s idea of current address is incremented after an instruction is assembled. $ may be manipulated just like any other number:

```
$
$ + 1
$ - 2
```

and can be used as a shortcut for writing loops without labels.

```
LOOP:  BTFSS flag,0x00
       GOTO LOOP

       BTFSS flag,0x00
       GOTO  $ - 1
```

2.3.6 Suggestions for structuring your code

Nested IF operations can quickly become confusing. Indentation is one way of making code clearer. Another way is to add braces on IF, ELSE and ENDIF, like this:

```
IF (this) ; {
  ...
ELSE      ; }{
  ...
ENDIF     ; }
```

After you've done this, you can use your text editor's show-matching-brace to check matching parts of the IF structure. In vi this command is "%", in emacs it's M-C-f and M-C-b.

2.3.7 Directive summary

__BADRAM

```
__BADRAM <expression> [, <expression>]*
```

Instructs gpasm that it should generate an error if there is any use of the given RAM locations. Specify a range of addresses with <lo>-<hi>. See any processor-specific header file for an example.
See also: __MAXRAM

__BADROM

```
__BADROM <expression> [, <expression>]*
```

Instructs gpasm that it should generate an error if there is any use of the given ROM locations. Specify a range of addresses with <lo>-<hi>. See any processor-specific header file for an example.
See also: __MAXROM

__CONFIG

```
__CONFIG <expression>
```

Sets the PIC processor's configuration fuses.
See also: CONFIG, __FUSES

NOTE: __CONFIG and __FUSES directives are deprecated for 16-bit PIC MCU devices (PIC18FXXX), CONFIG directive should be used instead.

__FUSES

```
__FUSES <expression>
```

Alias for __CONFIG. Sets the PIC processor's configuration fuses
See also: CONFIG

NOTE: __CONFIG and __FUSES directives are deprecated for 16-bit PIC MCU devices (PIC18FXXX), CONFIG directive should be used instead.

__IDLOCS

```
__IDLOCS <expression> or __IDLOCS <expression1>,<expression2>
```

Sets the PIC processor's identification locations. For 12 and 14 bit processors, the four id locations are set to the hexadecimal value of expression. For 18cxx devices idlocation expression1 is set to the hexadecimal value of expression2.

See also: IDLOCS

__MAXRAM

```
__MAXRAM <expression>
```

Instructs gpasm that an attempt to use any RAM location above the one specified should be treated as an error. See any processor specific header file for an example.

See also: __BADRAM

__MAXROM

```
__MAXROM <expression>
```

Instructs gpasm that an attempt to use any ROM location above the one specified should be treated as an error. See any processor specific header file for an example.

See also: __BADROM

BANKISEL

```
BANKISEL <label>
```

This directive generates bank selecting code for indirect access of the address specified by <label>. The directive is not available for all devices. It is only available for 14 bit and 16 bit devices. For 14 bit devices, the bank selecting code will set/clear the IRP bit of the STATUS register. It will use MOVLB or MOVLR in 16 bit devices.

See also: BANKSEL, PAGESEL

BANKSEL

```
BANKSEL <label>
```

This directive generates bank selecting code to set the bank to the bank containing <label>. The bank selecting code will set/clear bits in the FSR for 12 bit devices. It will set/clear bits in the STATUS register for 14 bit devices. It will use MOVLB or MOVLR in 16 bit devices. MOVLB will be used for enhanced 16 bit devices.

See also: BANKISEL, PAGESEL

CONFIG

```
CONFIG <expression>[, <expression>]*
```

Sets configuration fuses only on 16-bit PIC MCU devices (PIC18FXXX) if the command line is included the "–mpasm-compatible" option.

In “gpasm” mode (if the command line is *not* included the "–mpasm-compatible" option), in the case of each processor (PIC12XXX, PIC14XXX, PIC18XXX) it is possible to use this directive.

See also: __CONFIG

CBLOCK

```
CBLOCK [<expression>]
       <label>[:<increment>][,<label>[:<increment>]]
ENDC
```

Marks the beginning of a block of constants <label>. gpasm allocates values for symbols in the block starting at the value <expression> given to CBLOCK. An optional <increment> value leaves space after the <label> before the next <label>.

See also: EQU

CODE

```
<label> CODE <expression>
```

Only for relocatable mode. Creates a new machine code section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name “.code” will be used. <expression> is optional and specifies the absolute address of the section.

See also: IDATA, UDATA, CODE_PACK

CODE_PACK

```
<label> CODE_PACK <expression>
```

Only for relocatable mode. Creates a new byte-packed machine code section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name “.code” will be used. <expression> is optional and specifies the absolute address of the section.

See also: IDATA, UDATA, CODE

CONSTANT

```
CONSTANT <label>=<expression> [, <label>=<expression>]*
```

Permanently assigns the value obtained by evaluating <expression> to thesymbol <label>. Similar to SET and VARIABLE, except it can not be changed once assigned.

See also: EQU, SET, VARIABLE

DA

```
<label> DA <expression> [, <expression]*
```

Stores Strings in program memory. The data is stored as one 14 bit word representing two 7 bit ASCII characters.
See also: DT

DATA

```
DATA <expression> [, <expression]*
```

Generates the specified data.
See also: DA, DB, DE, DW

DB

```
<label> DB <expression> [, <expression]*
```

Declare data of one byte. The values are packed two per word.
See also: DA, DATA, DE, DW

DE

```
<label> DE <expression> [, <expression]*
```

Define EEPROM data. Each character in a string is stored in a separate word.
See also: DA, DATA, DB, DW

DT

```
DT <expression> [, <expression]*
```

Generates the specified data as bytes in a sequence of RETLW instructions.
See also: DATA

DW

```
<label> DW <expression> [, <expression]*
```

Declare data of one word.
See also: DA, DATA, DB, DW

ELSE

```
ELSE
```

Marks the alternate section of a conditional assembly block.
See also: IF, IFDEF, IFNDEF, ELSE, ENDIF

END

```
END
```

Marks the end of the source file.

ENDC

```
ENDC
```

Marks the end of a CBLOCK.
See also: CBLOCK

ENDIF

```
ENDIF
```

Ends a conditional assembly block.
See also: IFDEF, IFNDEF, ELSE, ENDIF

ENDM

```
ENDM
```

Ends a macro definition.
See also: MACRO

ENDW

```
ENDW
```

Ends a while loop.
See also: WHILE

EQU

```
<label> EQU <expression>
```

Permanently assigns the value obtained by evaluating <expression> to the symbol <label>. Similar to SET and VARIABLE, except it can not be changed once assigned.
See also: CONSTANT, SET

ERROR

```
ERROR <string>
```

Issues an error message.
See also: MESSG

ERRORLEVEL

```
ERRORLEVEL {0 | 1 | 2 | +<msgnum> | -<msgnum>}[, ...]
```

Sets the types of messages that are printed.

Setting	Affect
0	Messages, warnings and errors printed.
1	Warnings and error printed.
2	Errors printed.
-<msgnum>	Inhibits the printing of message <msgnum>.
+<msgnum>	Enables the printing of message <msgnum>.

See also: LIST

EXTERN

```
EXTERN <symbol> [ , <symbol> ]*
```

Only for relocatable mode. Delcare a new symbol that is defined in another object file.
See also: GLOBAL

EXITM

```
EXITM
```

Immediately return from macro expansion during assembly.
See also: ENDM

EXPAND

```
EXPAND
```

Expand the macro in the listing file.
See also: ENDM

FILL

```
<label> FILL <expression>,<count>
```

Generates <count> occurrences of the program word or byte <expression>. If expression is enclosed by parentheses, expression is a line of assembly.
See also: DATA DW ORG

GLOBAL

```
GLOBAL <symbol> [ , <symbol> ]*
```

Only for relocatable mode. Delcare a symbol as global.
See also: GLOBAL

IDATA

```
<label> IDATA <expression>
```

Only for relocatable mode. Creates a new initialized data section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name ".idata" will be used. <expression> is optional and specifies the absolute address of the section. Data memory is allocated and the initialization data is placed in ROM. The user must provide the code to load the data into memory.
See also: CODE, UDATA

IDLOCS

```
IDLOCS <expression> or IDLOCS <expression1>,<expression2>
```

In "gpasm" mode (if the command line is *not* included the "–mpasm-compatible" option), in the case of PIC18FXXX processors it is possible to use this directive.

The expressions may be are as follows: string ("abcdef"), character ('a'), constant (0x37), symbol (NUM1)

These may be used simultaneously: IDLOCS "abcd", 0x76, 'D', NUM1

The length of IDLOCS can be at most 8 bytes.

See also: CONFIG, __IDLOCS

IF

```
IF <expression>
```

Begin a conditional assembly block. If the value obtained by evaluating <expression> is true (i.e. non-zero), code up to the following ELSE or ENDIF is assembled. If the value is false (i.e. zero), code is not assembled until the corresponding ELSE or ENDIF.
See also: IFDEF, IFNDEF, ELSE, ENDIF

IFDEF

```
IFDEF <symbol>
```

Begin a conditional assembly block. If <symbol> appears in the symbol table, gpasm assembles the following code.
See also: IF, IFNDEF, ELSE, ENDIF

IFNDEF

```
IFNDEF <symbol>
```

Begin a conditional assembly block. If <symbol>does not appear in the symbol table, gpasm assembles the following code.

See also: IF, IFNDEF, ELSE, ENDIF

LIST

```
LIST <expression> [ , <expression> ] *
```

Enables output to the list (".lst") file. All arguments are interpreted as decimal regardless of the current radix setting. "list n=0" may be used to prevent page breaks in the code section of the list file. Other options are listed in the table below:

option	description
b=nnn	Sets the tab spaces
f=<format>	Set the hex file format. Can be inhx8m, inhx8s, inhx16, or inhx32.
m=<expression>	Set the maximum ROM address.
mm=[ON|OFF]	Memory Map on or off
n=nnn	Sets the number of lines per page
p = <symbol>	Sets the current processor
pe = <symbol>	Sets the current processor and enables the 18xx extended mode
r= [oct | dec | hex]	Sets the radix
st = [ON | OFF]	Symbol table dump on or off
w=[0 | 1| 2]	Sets the message level.

See also: NOLIST, RADIX, PROCESSOR

LOCAL

```
LOCAL <symbol>[[=<expression>], [<symbol>[=<expression>]]*]
```

Declares <symbol> as local to the macro that's currently being defined. This means that further occurrences of <symbol> in the macro definition refer to a local variable, with scope and lifetime limited to the execution of the macro.

See also: MACRO, ENDM

MACRO

```
<label> MACRO [ <symbol> [ , <symbol> ]* ]
```

Declares a macro with name <label>. gpasm replaces any occurrences of <symbol> in the macro definition with the parameters given at macro invocation.

See also: LOCAL, ENDM

MESSG

```
MESSG <string>
```

Writes <string> to the list file, and to the standard error output.
See also: ERROR

NOEXPAND

```
NOEXPAND
```

Turn off macro expansion in the list file.
See also: EXPAND

NOLIST

```
NOLIST
```

Disables list file output.
See also: LIST

ORG

```
ORG <expression>
```

Sets the location at which instructions will be placed. If the source file does not specify an address with ORG, gpasm assumes an ORG of zero.

PAGE

```
PAGE
```

Causes the list file to advance to the next page.
See also: LIST

PAGESEL

```
PAGESEL <label>
```

This directive will generate page selecting code to set the page bits to the page containing the designated <label>. The page selecting code will set/clear bits in the STATUS for 12 bit and 14 bit devices. For 16 bit devices, it will generate MOVLW and MOVWF to modify PCLATH. The directive is ignored for enhanced 16 bit devices.
See also: BANKISEL, BANKSEL, PAGESELW

PAGESELW

```
PAGESELW <label>
```

This directive will generate page selecting code to set the page bits to the page containing the designated <label>. The page selecting code will generate MOVLW and MOVWF to modify PCLATH. The directive is ignored for enhanced 16 bit devices.

See also: BANKISEL, BANKSEL, PAGESEL

PROCESSOR

```
PROCESSOR <symbol>
```

Selects the target processor. See section **??** for more details.

See also: LIST

RADIX

```
RADIX <symbol>
```

Selects the default radix from "oct" for octal, "dec" for decimal or "hex" for hexadecimal. gpasm uses this radix to interpret numbers that don't have an explicit radix.

See also: LIST

RES

```
RES <mem_units>
```

Causes the memory location pointer to be advanced <mem_units>. Can be used to reserve data storage.

See also: FILL, ORG

SET

```
<label> SET <expression>
```

Temporarily assigns the value obtained by evaluating <expression> to the symbol <label>.

See also: SET

SPACE

```
SPACE <expression>
```

Inserts <expression> number of blank lines into the listing file.

See also: LIST

SUBTITLE

```
SUBTITLE <string>
```

This directive establishes a second program header line for use as a subtitle in the listing output. <string> is an ASCII string enclosed by double quotes, no longer than 60 characters.

See also: TITLE

TITLE

```
TITLE <string>
```

This directive establishes a program header line for use as a title in the listing output. <string> is an ASCII string enclosed by double quotes, no longer than 60 characters.

See also: SUBTITLE

UDATA

```
<label> UDATA <expression>
```

Only for relocatable mode. Creates a new uninitialized data section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name ".udata" will be used. <expression> is optional and specifies the absolute address of the section.

See also: CODE, IDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

UDATA_ACS

```
<label> UDATA_ACS <expression>
```

Only for relocatable mode. Creates a new uninitialized accessbank data section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name ".udata_acs" will be used. <expression> is optional and specifies the absolute address of the section.

See also: CODE, IDATA, UDATA

UDATA_OVR

```
<label> UDATA_OVR <expression>
```

Only for relocatable mode. Creates a new uninitialized overlaid data section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name ".udata_ovr" will be used. <expression> is optional and specifies the absolute address of the section.

See also: CODE, IDATA, UDATA

UDATA_SHR

```
<label> UDATA_SHR <expression>
```

Only for relocatable mode. Creates a new uninitialized sharebank data section in the output object file. <label> specifies the name of the section. If <label> is not specified the default name ".udata_shr" will be used. <expression> is optional and specifies the absolute address of the section.

See also: CODE, IDATA, UDATA

VARIABLE

```
VARIABLE <label>[=<expression>, <label>[=<expression>]]*
```

Delcares variable with the name <label>. The value of <label> may later be reassigned. The value of <label> does not have to be assigned at declaration.

See also: CONSTANT

WHILE

```
WHILE <expression>
```

Performs loop while <expression> is true.

See also: ENDW

2.3.8 High level extensions

gpasm supports several directives for use with high level languages. These directives are easily identified because they start with ".". They are only available in relocatable mode.

These features are advanced and require knowledge of how gputils relocatable objects work. These features are intended to be used by compilers. Nothing prevents them from being used with assembly.

.DEF

```
.DEF <symbol> [, <expression> ]*
```

Create a new COFF <symbol>. Options are listed in the table below:

option	description
absolute	Absolute symbol keyword
class=nnn	Sets the symbol class (byte sized)
debug	Debug symbol keyword
extern	External symbol keyword
global	Global symbol keyword
size=nnn	Reserve words or bytes for the symbol
static	Static Symbol keyword
type=nnn	Sets the symbol type (short sized)
value=nnn	Sets the symbol value

This directive gives the user good control of the symbol table. This control is necessary, but if used incorrectly it can have many undesirable consequences. It can easily cause errors during linking or invalid machine code. The user must fully understand the operation of gputils COFF symbol table before modifying its contents.

For best results, only one of the single keywords should be used. The keyword should follow the symbol name. The keyword should then be followed by any expressions that directly set the values. Here is an example:

.def global_clock, global, type = T_ULONG, size = 4

See also: .DIM

.DIM

```
.DIM <symbol>, <number>, <expression> [, <expression> ] *
```

Create <number> auxiliary symbols, attached to <symbol>. Fill the auxilary symbols with the values specified in <expression>. The expressions must result in byte sized values when evaluated or be strings. The symbol must be a COFF symbol.

This directive will generate an error if the symbol already has auxiliary symbols. This prevents the user from corrupting automatically generated symbols.

Each auxiliary symbol is 18 bytes for Microchip COFF version 1 or 20 bytes for Microchip COFF version 2. If the byte size of contents specified by the expressions is greater then symbol size, several auxiliary symbols are generated. The last auxiliary symbol is zero byte padded to the auxiliary symbol length.

gpasm does not use auxilary symbols. So the contents have no effect on its operation. However, the contents may be used by gplink or a third party tool.

See also: .DEF

.DIRECT

```
.DIRECT <command>, <string>
```

Provides a mechanism for direct communication from the program to the debugging environment. This method has no impact on the executable. The symbols will appear in both the COFF files and the COD files.

Each directive creates a new COFF symbol ".direct". An auxilary symbol is attached that contains <command> and <string>. The string must be less than 256 bytes. The command must have a value 0 to 255. There are no restrictions on the content, however these messages must conform to the debugging environment. The typical values are summarized in the table below:

ASCII command	description
a	User defined assert
A	Assembler/Compiler defined assert
e	User defined emulator commands
E	Assembler/Compiler defined emulator commands
f	User defined printf
F	Assembler/Compiler defined printf
l	User defined log command
L	Assembler/Compiler/Code verification generated log command

The symbols also contain the address where the message was inserted into the assembly. The symbols, with the final relocated addresses, are available in executable COFF. The symbols are also written to the COD file. They can be viewed using gpvc.

See also: .DEF, .DIM

.EOF

```
.EOF
```

This directive causes an end of file symbol to be placed in the symbol table. Normally this symbol is automatically generated. This directive allows the user to manually generate the symbol. The directive is only processed if the "-g" command line option is used. When that option is used, the automatic symbol generation is disabled.

See also: .EOF, .FILE, .LINE

.FILE

```
.FILE <string>
```

This directive causes a file symbol to be placed in the symbol table. Normally this symbol is automatically generated. This directive allows the user to manually generate the symbol. The directive is only processed if the "-g" command line option is used. When that option is used, the automatic symbol generation is disabled.

See also: .EOF, .FILE, .LINE

.IDENT

```
.IDENT <string>
```

Creates an .ident COFF symbol and appends an auxiliary symbol. The auxiliary symbol points to an entry in the string table. The entry contains <string>. It is an ASCII comment of any length. This symbol has no impact on the operation of gputils. It is commonly used to store compiler versions.

See also: .DEF, .DIM

.LINE

```
.LINE <expression>
```

This directive causes and COFF line number to be generated. Normally they are automatically generated. This directive allows the user to manually generate the line numbers. The directive is only processed if the "-g" command line option is used. When that option is used, the automatic symbol generation is disabled. The <expression> is always evaluated as decimal regardless of the current radix setting.

See also: .EOF, .FILE, .LINE

.TYPE

```
.TYPE <symbol>, <expression>
```

This directive modifies the COFF type of an existing <symbol>. The symbol must be defined. The type must be 0 to 0xffff. Common types are defined in coff.inc.

COFF symbol types default to NULL in gpasm. Although the type has no impact linking or generating an executable, it does help in the debug environment.

See also: .DEF

2.4 Instructions

2.4.1 Instruction set summary

12 Bit Devices (PIC12C5XX)

Syntax	Description
ADDLW <imm8>	Add immediate to W
ADDWF <f>,<dst>	Add W to <f>, result in <dst>
ANDLW <imm8>	And W and literal, result in W
ANDWF <f>,<dst>	And W and <f>, result in <dst>
BCF <f>,<bit>	Clear <bit> of <f>
BSF <f>,<bit>	Set <bit> of <f>
BTFSC <f>,<bit>	Skip next instruction if <bit> of <f> is clear
BTFSS <f>,<bit>	Skip next instruction if <bit> of <f> is set
CALL <addr>	Call subroutine
CLRF <f>,<dst>	Write zero to <dst>
CLRW	Write zero to W
CLRWDT	Reset watchdog timer
COMF <f>,<dst>	Complement <f>, result in <dst>
DECF <f>,<dst>	Decrement <f>, result in <dst>
DECFSZ <f>,<dst>	Decrement <f>, result in <dst>, skip if zero
GOTO <addr>	Go to <addr>
INCF <f>,<dst>	Increment <f>, result in <dst>
INCFSZ <f>,<dst>	Increment <f>, result in <dst>, skip if zero
IORLW <imm8>	Or W and immediate
IORWF <f>,<dst>	Or W and <f>, result in <dst>
MOVF <f>,<dst>	Move <f> to <dst>
MOVLW <imm8>	Move literal to W
MOVWF <f>	Move W to <f>
NOP	No operation
OPTION	
RETLW <imm8>	Load W with immediate and return
RLF <f>,<dst>	Rotate <f> left, result in <dst>
RRF <f>,<dst>	Rotate <f> right, result in <dst>
SLEEP	Enter sleep mode
SUBWF <f>,<dst>	Subtract W from <f>, result in <dst>
SWAPF <f>,<dst>	Swap nibbles of <f>, result in <dst>
TRIS	
XORLW	Xor W and immediate
XORWF	Xor W and <f>, result in <dst>

12 Bit Devices Enhanced Instruction Set

Syntax	Description
MOVLB <k>	Move literal to BSR register
RETFIE	Return from interrupt
RETURN	Return, maintain W

14 Bit Devices (PIC16CXX)

Syntax	Description
ADDLW <imm8>	Add immediate to W
ADDWF <f>,<dst>	Add W to <f>, result in <dst>
ANDLW <imm8>	And immediate to W
ANDWF <f>,<dst>	And W and <f>, result in <dst>
BCF <f>,<bit>	Clear <bit> of <f>
BSF <f>,<bit>	Set <bit> of <f>
BTFSC <f>,<bit>	Skip next instruction if <bit> of <f> is clear
BTFSS <f>,<bit>	Skip next instruction if <bit> of <f> is set
CALL <addr>	Call subroutine
CLRF <f>,<dst>	Write zero to <dst>
CLRW	Write zero to W
CLRWDT	Reset watchdog timer
COMF <f>,<dst>	Complement <f>, result in <dst>
DECF <f>,<dst>	Decrement <f>, result in <dst>
DECFSZ <f>,<dst>	Decrement <f>, result in <dst>, skip if zero
GOTO <addr>	Go to <addr>
INCF <f>,<dst>	Increment <f>, result in <dst>
INCFSZ <f>,<dst>	Increment <f>, result in <dst>, skip if zero
IORLW <imm8>	Or W and immediate
IORWF <f>,<dst>	Or W and <f>, result in <dst>
MOVF <f>,<dst>	Move <f> to <dst>
MOVLW <imm8>	Move literal to W
MOVWF <f>	Move W to <f>
NOP	No operation
OPTION	
RETFIE	Return from interrupt
RETLW <imm8>	Load W with immediate and return
RETURN	Return from subroutine
RLF <f>,<dst>	Rotate <f> left, result in <dst>
RRF <f>,<dst>	Rotate <f> right, result in <dst>
SLEEP	Enter sleep mode
SUBLW	Subtract W from literal
SUBWF <f>,<dst>	Subtract W from <f>, result in <dst>
SWAPF <f>,<dst>	Swap nibbles of <f>, result in <dst>
TRIS	
XORLW	Xor W and immediate
XORWF	Xor W and <f>, result in <dst>

14 Bit Devices Enhanced Instruction Set

Syntax	Description
ADDFSR <n>, <k>	Add Literal <k> to FSR<n>
ADDWFC <f>,<dst>	Add with Carry W and <f>
ASRF <f>,<dst>	Arithmetic Right Shift
BRA <k>	Relative Branch
BRW	Relative Branch with W
CALLW	Call Subroutine with W
LSLF <f>,<dst>	Logical Left Shift
LSRF <f>,<dst>	Logical Right Shift
MOVIW ++FSR<n>	Move Indirect FSR<n> to W with preincrement
MOVIW –FSR<n>	Move Indirect FSR<n> to W with predecrement
MOVIW FSR<n>++	Move Indirect FSR<n> to W with postincrement
MOVIW FSR<n>–	Move Indirect FSR<n> to W with postdecrement
MOVIW <k>[<n>]	Move INDFn to W, Indexed Indirect
MOVWI ++FSR<n>	Move W to Indirect FSR<n> with preincrement
MOVWI –FSR<n>	Move W to Indirect FSR<n> with predecrement
MOVWI FSR<n>++	Move W to Indirect FSR<n> with postincrement
MOVWI FSR<n>–	Move W to Indirect FSR<n> with postdecrement
MOVWI <k>[<n>]	Move W to INDF<n>, Indexed Indirect
MOVLB <k>	Move literal to BSR
MOVLP <k>	Move literal to PCLATH
RESET	Software device Reset
SUBWFB <f>,<dst>	Subtract with Borrow W from <f>

Ubicom Processors

For Ubicom (Scenix) processors, the assembler supports the following instructions, in addition to those listed under “12 Bit Devices” above.

Syntax	Description
BANK <imm3>	
IREAD	
MODE <imm4>	
MOVMW	
MOVWM	
PAGE <imm3>	
RETI	
RETIW	
RETP	
RETURN	

Special Macros

There are also a number of standard additional macros. These macros are:

Syntax	Description
ADDCF <f>,<dst>	Add carry to <f>, result in <dst>
B <addr>	Branch
BC <addr>	Branch on carry
BZ <addr>	Branch on zero
BNC <addr>	Branch on no carry
BNZ <addr>	Branch on not zero
CLRC	Clear carry
CLRZ	Clear zero
SETC	Set carry
SETZ	Set zero
MOVFW <f>	Move file to W
NEGF <f>	Negate <f>
SKPC	Skip on carry
SKPZ	Skip on zero
SKPNC	Skip on no carry
SKPNZ	Skip on not zero
SUBCF <f>,<dst>	Subtract carry from <f>, result in <dst>
TSTF <f>	Test <f>

2.5 Errors/Warnings/Messages

gpasm writes every error message to two locations:

- the standard error output
- the list file (".lst")

The format of error messages is:

```
Error <src-file> <line> : <code> <description>
```

where:

<src-file> is the source file where gpasm encountered the error

<line> is the line number

<code> is the 3-digit code for the error, given in the list below

<description> is a short description of the error. In some cases this contains further information about the error.

Error messages are suitable for parsing by emacs' "compilation mode". This chapter lists the error messages that gpasm produces.

2.5.1 Errors

101 ERROR directive

A user-generated error. See the ERROR directive for more details.

108 Illegal character.

gpasm encountered an illegal character in a source file.

109 Unmatched (

110 Unmatched)

113 Symbol not previously defined.

gpasm encountered an unrecognized symbol.

114 Divide by zero.

gpasm encountered a divide by zero.

115 Duplicate label or redefining symbol that cannot be redefined.

116 Address label duplicated or different in second pass.

Label resolved to a different address on gpasm's second pass.

117 Address wrapped around 0.

118 Overwriting previous address contents.

gpasm was instructed to write different values into the same address.

120 Call or jump not allowed at this address (must be in low half of page)

gpasm was instructed to write different values into the same address.

121 Illegal label.

gpasm encountered an illegal label.

123 Illegal directive (Not Valid for this processor).

The specified directive is not valid for this processor.

124 Illegal Argument.

gpasm encountered an illegal argument in an expression.

125 Illegal Condition.

An illegal condition like a missing ENDIF or ENDW has been encountered.

126 Argument out of range.

The expression has an argument that was out of range.

127 Too many arguments.

gpasm encountered an expression with too many arguments.

128 Missing argument(s).

gpasm encountered an expression with at least one missing argument.

129 Expected

Expected a certain type of argument.

130 Processor type previously defined.

The processor is being redefined.

131 Processor type is undefined.

The processor type has not been defined.

132 Unknown processor.

The selected processor is not valid. Check the processors listed in section **??**.

133 Hex file format INHX32 required.

An address above 32K was specified.

135 Macro name missing.

A macro was defined without a name.

136 Duplicate macro name.

A macro name was duplicated.

140 WHILE must terminate within 256 iterations.

gpasm encountered an infinite loop or a loop with too many iterations (more than 256).

143 Illegal nesting.

145 Unmatched ENDM.

ENDM found without a macro definition.

149 Directive only allowed when generating an object file.

Attempt to use relocatable-mode directive when generating HEX file directly.

151 Operand contains unresolvable labels or is too complex.

Labels must be resolvable to a relocatable address plus a constant.

152 Executable code and data must be defined in an appropriate section.

Code or data defined in an invalid section.

154 Each object file section must be contiguous.

156 Operand must be an address label.

gpasm encountered a non-label operand where an address label was expected.

157 ORG at odd address.

ORG directive must take an even address as the start of an absolute section.

159 Cannot use FILL Directive with odd number of bytes.

In PIC18CXX devices the number of bytes must be even.

163 __CONFIG directives must be contiguous.

164 __IDLOC directives must be contiguous.

168 Square brackets required around offset operand.

175 __IDLOCS directives must be listed in ascending order.

176 An error with the CONFIG directive occured.

177 You cannot mix CONFIG and __CONFIG directives.

CONFIG and __CONFIG directives cannot both be used in the same project.

180 RES directive cannot reserve odd number of bytes in PIC18 absolute mode.

2.5.2 Warnings

201 Symbol not previously defined.

The symbol being #undefined was not previously defined.

202 Argument out of range. Least significant bits used.

The argument does not fit in the allocated space.

203 Found opcode in column 1.

Opcodes should be indented to distinguish them from labels.

205 Found directive in column 1.

Directives should be indented to distinguish them from labels.

206 Found call to macro in column 1.

Macro calls should be indented to distinguish them from labels.

207 Found label after column 1.

Labels should be unindented to distinguish them from directives and opcodes.

209 Missing quote.

Inserted close quote after quoted string.

211 Extraneous arguments on the line.

Extra arguments were found on the line.

212 Expected.

215 Processor superseded by command line.

The processor was specified on the command line and in the source file. The command line has precedence.

216 Radix superseded by command line.

The radix was specified on the command line and in the source file. The command line has precedence.

217 Hex file format specified on command line.

The hex file format was specified on the command line and in the source file. The command line has precedence.

218 Expected dec, oct, hex. Will use hex.

gpasm encountered an invalid radix.

219 Invalid RAM location specified.

gpasm encountered an invalid RAM location as specified by the __MAXRAM and __BADRAM directives.

220 Address exceeds maximum range for this processor.

Data emitted past maximum ROM address.

222 Error messages can not be disabled.

Error messages can not be disabled using the ERRORLEVEL directive.

223 Redefining processor.

The processor is being reselected by the LIST or PROCESSOR directive.

224 Use of this instruction is not recommanded.

Use of the TRIS and OPTION instructions is not recommended for a PIC16CXX device.

226 Destination address must be word aligned.

228 Invalid ROM location specified.

gpasm encountered an invalid ROM location as specified by the __MAXROM and __BADROM directives.

2.5.3 Messages

301 User Message

User message, invoked with the MESSG directive.

302 Register in operand not in bank 0. Ensure bank bits are correct.

Accessing a register outside of bank 0. User must select the appropriate bank with banksel or similar directives.

303 Program word too large. Truncated to core size.

gpasm has encounter a program word larger than the core size of the selected device.

304 ID Locations value too large. Last four hex digits used.

The ID locations value specified is too large.

305 Using default destination of 1 (file).

No destination was specified so the default location was used.

306 Crossing page boundary – ensure page bits are set.

ROM address crossed boundary between pages. User must select appropriate page with pagesel or similar directives when using call or goto directives.

307 Setting page bits.

308 Warning level superceded by command line value.

The warning level was specified on the command line and in the source file. The command line has precedence.

309 Macro expansion superceded by command line value.

Macro expansion was specified on the command line and in the source file. The command line has precedence.

310 Superceding current maximum RAM and RAM map

312 Page or Bank selection not needed for this device.

This device does not use special page or bank selection code.

313 CBLOCK constants will start with a value of 0.

First CBLOCK has no initial value. Assuming a value of 0.

316 W register modified.

Hidden use of the W register overwrites previous value. User may need to save and restore the original value.

318 Special Instruction Mnemonic used.

Using special instruction mnemonic which may map to one or several instructions.

Chapter 3

gplink

gplink relocates and links gpasm COFF objects and generates an absolute executable COFF.

3.1 Running gplink

The general syntax for running gplink is

```
gplink [options] [objects] [libraries]
```

Where options can be one of:

Option	Long option	Meaning
-a <format>	–hex-format <format>	Produce hex file in one of four formats: inhx8m, inhx8s, inhx16, inhx32 (the default)
-c	–object	Output an executable object
-d	–debug	Display debug messages
-f <value>	–fill <value>	Fill unused unprotected program memory with <value>
-h	–help	Show the help message
-I <directory>	–include <directory>	Specify an include directory
-l	–no-list	Disable the list file output
-m	–map	Output a map file
	–mplink-compatible	MPLINK compatibility mode
-o <file>	–output <file>	Alternate name of hex output file
-O <level>	–optimize <level>	Optimization level
-q	–quiet	Quiet
-r	–use-shared	Attempt to relocate unshared data sections to shared memory if relocation fails
-s <file>	–script <file>	Specify linker script
-t <size>	–stack <size>	Create a stack section
-u	–macro <symbol[=value]>	Add macro value for script.
-v	–version	Print gplink version information and exit
-w	–processor-mismatch	Disable processor mismatch warning.

3.2 MPLINK compatibility mode

When the –mplink-compatibe command line option is specified gpl tries to mimic MPLINK behaviour:

- .cinit initialized data section is generated always, even if initialized data is not defined
- .cinit initialized data section is generated at the lowest possible address in the nonvolatile program memory (ROM)

3.3 gplink outputs

gplink creates an absolute executable COFF. From this COFF a hex file and cod file are created. The executable COFF is only written when the "-c" option is added. This file is useful for simulating the design with mpsim. The cod file is used for simulating with gpsim.

gplink can also create a map file. The map file reports the final addresses gplink has assigned to the COFF sections. This is the same data that can be viewed in the executable COFF with gpvo.

3.4 Linker scripts

gplink requires a linker script. This script tells gplink what memory is available in the target processor. A set of Microchip generated scripts are installed with gputils. These scripts were intended as a starting

point, but for many applications they will work as is.

If the user does not specify a linker script, gplink will attempt to use the default script for the processor reported in the object file. The default location of the scripts is reported in the gplink help message.

3.5 Stacks

gplink can create a stack section at link time using a stack directive in the linker script. The same feature can be utilized with a -t option on the command line. gplink will create the section and two symbols. _stack points to the beginning of the stack section and _stack_end points to the end.

3.6 Optimization

gplink is an optimizing linker. There are four different optimzation levels. Each level includes all optimizations of lower levels. Increasing the level typically increases the link time required.

3.6.1 Level 0

No optimizations.

3.6.2 Level 1 (default)

Weak Symbols

A weak symbol is an external symbol declaration that isn't used. These symbols are typically created by declaring functions or data that isn't used. Including these symbols might lead to extra objects being extracted from archives for symbol resolution. That will increase the data and program memory used. This optimzation removes all weak symbols when the object file is read by the linker.

3.6.3 Level 2

Dead Sections

A dead section is any section that doesn't have relocations pointing to its symbols. This means the code or data in the section is never accesed. This optimization removes the section and its symbols to reduce program and data memory. This optimization will not remove any absolute sections.

3.6.4 Level 3

No optimizations.

Chapter 4

gplib

gplib creates, modifies and extracts COFF archives. This allows a related group of objects to be combined into one file. Then this one file is passed to gplink.

4.1 Running gplib

The general syntax for running gplib is

```
gplib [options] library [member]
```

Where options can be one of:

Option	Long option	Meaning
-c	–create	Create a new library
-d	–delete	Delete member from library
-h	–help	Show the help message
-n	–no-index	Don't add the symbol index
-q	–quiet	Quiet mode
-r	–replace	Add or replace member from library
-s	–symbols	List global symbols in libary
-t	–list	List member in library
-v	–version	Print gplib version information and exit
-x	–extract	Extract member from library

4.2 Creating an archive

The most common operation is to create a new archive:

```
gplib -c math.a mult.o add.o sub.o
```

This command will create a new archive “math.a” that contains “mult.o add.o sub.o”.

The name of the archive “math.a” is arbitrary. The tools do not use the file extension to determine file type. It could just as easily been “math.lib” or “math”.

When you use the library, simply add it to the list of object passed to gplink. gplink will scan the library and only extract the archive members that are required to resolve external references. So the application won’t necessarily contain the code of all the archive members.

4.3 Other gplib operations

Most of the other are useful , but will be used much less often. For example you can replace individual archive members, but most people elect to delete the old archive and create a new one.

4.4 Archive format

The file format is a standard COFF archive. A header is added to each member and the unmodified object is copied into the archive.

Being a standard archive they do include a symbol index. It provides a simple why to determine which member should be extract to resolve external references. This index is not included in mplib archives. So using gplib archives with Microchip Tools will probably cause problems unles the “-n” option is added when the archive is created.

Chapter 5

Utilities

5.1 gpdasm

gpdasm is a disassembler for gputils. It converts hex files generated by gpasm and gplink into disassembled instructions.

5.1.1 Running gpdasm

The general syntax for running gpdasm is

```
gpdasm [options] hex-file
```

Where options can be one of:

Option	Long option	Meaning
-c	–mnemonics	Decode the special mnemonics.
-h	–help	Display the help message.
-i	–hex-info	Show the informations of the input hex file
-j	–mov-fsrn	In the MOVIW or MOVWI instructions show as base the FSRn register instead of the INDFn. [INDFn]
-k <file>	–label-list <file>	A file which lists the names and addresses of the labels in the disassembled program code. (With the -n, -o and -s options.)
-l	–list-chips	List the supported processors.
-m	–dump	Memory dump of the input hex file.
-n	–show-names	For some case of SFR, shows the name of instead of the address. In addition shows the labels also.
-o	–show-config	Show the CONFIG and IDLOCS - or __idlocs - directives.
-p<processor>	–processor <processor>	Select the processor.
-s	–short	Print short form output. (Creates a compilable source. See also the -k, -n and -o options.)
-v	–version	Print the gpdasm version information and exit.
-y	–extended	Enable 18xx extended mode.
	–strict	Disassemble only opcodes generated by gpasm in case of instructions with several opcodes.

gpdasm doesn't specifically create an output file. It dumps its output to the screen. This helps to reduce the risk that a good source file will be unintentionally overwritten. If you want to create an output file and your shell is "sh", "bash" or "ksh", you can do something like:

```
gpdasm test.hex > test.dis
```

This redirects standard output to the file "test.dis".

5.1.2 Comments on Disassembling

- The gpdasm only uses a hex file as an input. Because of this it has no way to distinguish between instructions and data in program memory.
- If gpdasm encounters an unknown instruction it uses the DW directive and treats it as raw data.
- There are DON'T CARE bits in the instruction words. Normally, this isn't a problem. It could be, however, if a file with data in the program memory space is disassembled and then reassembled. Example: gpdasm will treat 0x0060 in a 14 bit device as a NOP. If the output is then reassembled, gpasm will assign a 0x0000 value. The value has changed and both tools are behaving correctly.
- In this case helps the "-k <list_file>" or "–label-list" <list_file> command line options (with the -n, -o and -s options). The description of the format of the list file can be found in the *gpdasm_sample.ulist*
  For example: gpdasm -nos *-k test.ulist* -p12f1822 test.hex > test.dis

5.2 gpstrip

gpstrip manipulates the sections and symbol tables of gputils object files.

5.2.1 Running gpstrip

The general syntax for running gpstrip is

```
gpstrip [options] object-file
```

Where options can be one of:

Option	Long option	Meaning
-g	–strip-debug	Strip debug symbols
-h	–help	Show the help message
-k <symbol>	–keep-symbol <symbol>	Keep symbol
-n <symbol>	–strip-symbol <symbol>	Remove symbol
-o <file>	–output <file>	Alternate output file
-p	–preserve-dates	Preserve dates
-r <section>	–remove-section <section>	Remove section
-s	–strip-all	Remove all symbols
-u	–strip-unneeded	Remove all symbols not needed for relocations
-v	–version	Show version
-V	–verbose	Verbose mode
-x	–discard-all	Remove non-global symbols

5.3 gpvc

gpvc is cod file viewer for gputils. It provides an easy way to view the contents of the cod files generated by gpasm and gplink.

5.3.1 Running gpvc

The general syntax for running gpvc is

```
gpvc [options] cod-file
```

Where options can be one of:

Option	Long option	Meaning
-a	–all	Display all information
-d	–directory	Display directory header
-h	–help	Show the help message.
-l	–listing	Display source listing
-m	–message	Display debug message area
-r	–rom	Display ROM
-s	–symbols	Display symbols
-v	–version	Print gpvc version information and exit

gpvc doesn't specifically create an output file. It dumps its output to the screen. If you want to create an output file and your shell is "sh", "bash" or "ksh", you can do something like:

```
gpvc test.cod > test.dump
```

This redirects standard output to the file "test.dump".

5.4 gpvo

gpvo is COFF object file viewer for gputils. It provides an easy way to view the contents of objects generated by gpasm and gplink.

5.4.1 Running gpvo

The general syntax for running gpvo is

```
gpvo [options] object-file
```

Where options can be one of:

Option	Long option	Meaning
-b	–binary	Binary data
-c	–mnemonics	Decode special mnemonics
-f	–file	File header
-h	–help	Show the help message
-n	–no-names	Suppress filenames
-s	–section	Section data
-t	–symbol	Symbol data
-v	–version	Print gpvo version information and exit
-x <file>	–export <file>	Export symbols to an include file.
-y	–extended	Enable 18xx extended mode

gpvo doesn't specifically create an output file. It dumps its output to the screen. If you want to create an output file and your shell is "sh", "bash" or "ksh", you can do something like:

```
gpvo test.obj > test.dump
```

This redirects standard output to the file "test.dump".

Index

.DEF, 28
.DIM, 29
.DIRECT, 29
.EOF, 30
.FILE, 30
.IDENT, 30
.LINE, 30
.TYPE, 30

Archive format, 45
ASCII, 13

__BADRAM, 17
__BADROM, 17
BANKISEL, 18
BANKSEL, 18
bash, 10, 47, 49

case, 9
CBLOCK, 19
character, 13
CODE, 19
CODE_PACK, 19
comments, 10
CONFIG, 19
__CONFIG, 17
CONSTANT, 19
Creating an archive, 44

DA, 20
DATA, 20
DB, 20
DE, 20
DT, 20
DW, 20

ELSE, 20
END, 21
ENDC, 21
ENDIF, 21
ENDM, 21
ENDW, 21
EQU, 21
ERROR, 21
error file, 10
ERRORLEVEL, 22
EXITM, 22
EXTERN, 22

FILL, 22
__FUSES, 17

GLOBAL, 23
GNU, 3
gpasm options, 8
gpdasm, 46
gpvc, 48
gpvo, 48, 49

hex file, 9

IDATA, 23
IDLOCS, 23
__IDLOCS, 18
IF, 23
IFDEF, 23
IFNDEF, 24
include, 13

ksh, 10, 47, 49

labels, 10
License, 3
LIST, 24
LOCAL, 24

MACRO, 24

make, 10
__MAXRAM, 18
__MAXROM, 18
MESSG, 25

NO WARRANTY, 3
NOEXPAND, 25
NOLIST, 25

operators, 11
ORG, 25
Other gplib operations, 45

PAGE, 25
PAGESEL, 25
PAGESELW, 26
PROCESSOR, 26

RADIX, 26
radix, 9, 12
RES, 26
Running gpdasm, 46
Running gplib, 44
Running gplink, 41
Running gpvc, 48
Running gpvo, 48, 49

SET, 26
sh, 10, 47, 49
SPACE, 26
SUBTITLE, 27

tee, 10
TITLE, 27

UDATA, 27
UDATA ACS, 27
UDATA OVR, 27
UDATA SHR, 27

VARIABLE, 28

WHILE, 28