

gputils 0.12.1

James Bowman and Craig Franklin

June 16, 2004

Contents

1	Introduction	4
1.1	Tool Flows	4
1.1.1	Absolute Asm Mode	4
1.1.2	Relocatable Asm Mode	4
1.1.3	HLL Mode	

2.6.4	Goto	13
2.6.5	If	14
2.6.6	Loop	14
2.6.7	Null	

3.3.6	Suggestions for structuring your code	27
3.3.7	Directive summary	28
3.4	Instructions	37
3.4.1	Instruction set summary	38

1.1.3 HLL Mode

In HLL (High Level Language) mode, the source code is written in a Ada like language. gpal then converts that file into a relocatable assembly file. It then automatically invokes gpasm to generate an absolute executable object.

1.1.4 Which Tool Flow is best?

Absolute mode is simple to understand. It only requires one tool, gpasm. Most of the examples on this website use absolute mode. So why use HLL mode?

the user requires

Option	Meaning
a	Compile or assemble, then archive.
c	Compile or assemble, but don't link.
d	

```
if a>b then  
  timer = 0;  
end if;
```

is equiv

CHAPTER

2.5.3 Numbers

gpal uses decimal as its default

2.5.5 Assignment

Test statements can only appear in the expressions of if statements to a boolean.

```
<name>['[' <expression> '']? = <expression>];
```

Assignment statements can appear in any statement block. <name> must be a variable. If the bracket enclosed expression is added it must be an array.

2.5.7 Label

Labels must match the foll 0 Td 8.699 0 Td (win[])Tj 22.2949 0 Td

2.5.6 Test

```
<expression> [ <comparison operator> <expression>]*;
```


Pragmas provide data to compiler which is outside of its legal syntax. The table below summarizes the pragmas a

CHAPTER 2. GPAL

```
<subprogram definitions|variable definitions|constants|types>  
end module;
```

The module defines a related group of subprograms and data that will be placed in the same page

2.10.2 Expression Evaluation

Most compilers are a stack machine, accumulator machine,

2.11.1t.783815617 0 TdUse.1t.21.578523 0 Tduint8.1t.29.504623 0 TdtypesAL

Chapter 3

gpasm

3.1 Running gpasm

The general syntax for running

Option	Meaning
a <format>	Produce hex file in one of four formats: inhx8m, inhx8s, inhx16, inhx32 (the default).
c	Output a relocatable object
d symbol[=value]	Equivalent to “#defi 0 -24699.08 Tm Td (alu<bol[=vj ET 28 910 Td (to)<v 10.19043Td (alue))T>”j E

3.1.2 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"),

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

base	general syntax	21 decimal written as
binary	[01]*b	10101b

3.3 Directives

3.3.1 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.

3.3.2 Configuration

You can choose the fuse settings for your PIC implementation using the __CONFIG directive, so that the hex file set the fuses explicitly. Naturally you should make sure that these settings match your

```

        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

```

3.3.5 \$

\$ expands to the address of the instruction currently being assembled. If it’s used in a context

CHAPTER 3. GPASM

DATA

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

Generates the specified data.

S7e also: DA, DB, DE, DW

DB

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

Declare data of one byte. The values are packed two per word.

S7e also: DA, DATA, DE, DW

DE

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

Define EEPROM data. Each character in a string is stored in a separate word.

S7e also: DA, DATA, DB, DW

DT

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

Generates the specified data as bytes in a sequence of RETLW instructions.

S7e also: DATA

DW

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

Declare data of one word.

S7e also: DA, DATA, DB, DW

ELSE

```
ELSE
```

Marks the alternate section of a conditional

ENDC

ENDC

Marks the end of a CBLOCK.

See also: CBLOCK

ENDIF

ENDIF

Ends a conditional assembly block.

See also: IFDEF, IFNDEF, ELSE,

ERRORLEVEL

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

Sets the types of messages that are printed.

Setting	
---------	--

option	description
b=nnn	Sets the tab spaces
f=<format>	Set the hex file format. Can be inhx8m, inhx8s, inhx16, or inhx32.
mm=[ON OFF]	Memory Map on or off
n=nnn	Sets the number of lines per page

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, IDAT

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

Special macros

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

Syntax	Description
--------	-------------

Chapter 4

gplink

gplink relocates and links gpasm COFF objects

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.

4.3 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

Chapter 5

gplib

gplib creates, modifies and extracts COFF archiv

Chapter 6

Utilities

6.1 gpdasm

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

6.1.1 Running gpdasm

The general syntax for running gpdasm is

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

Where options can be one of:

Option	Meaning
--------	---------

6.1.2 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 a problem if the instruction word is not a valid instruction.

6.3.1 Running gpvo

The general syntax for running gpvo

Index

Archive format, 47

ASCII, 25

BADRAM, 28

BANKISEL, 28

BANKSEL, 29

bash, 22, 48–50

case, 21

CBLOCK, 29

character, 25

CODE, 29

comments, 22

CONFIG, 28

CONSTANT, 29

Creating an archive, 46

DA, 29

DATA, 30

DB, 30

DE, 30

DT, 30

D, 30

ELSE, 30

PROCESSOR, 35

RADIX, 35

radix, 21, 24

RES, 36

Running gpdasm, 48

Running gplib, 46

Running gplink, 44

Running gpvc, 49

Running gpvo, 50

SET, 36

sh, 22, 48–50

SPACE, 36

SUBTITLE, 36

tee, 22

TITLE, 36

UDATA, 36

UDATA ACS, 37

UDATA OVR, 37

UDATA SHR, 37

VARIABLE, 37

WHILE, 37