gputils 0.13.2

James Bowman и Craig Franklin

5 мая 2005

Оглавление

Часть 1 4
Введение 4
1.1 Используемые режимы 4
1.1.1 Абсолютный Asm режим 4
1.1.2 Настраиваемый Asm режим 4
1.1.3 Какой режим лучше? 4
1.2 Поддерживаемые процессоры 5
Часть 2 7
gpasm 7
2.1 Запуск gpasm 7
2.1.1 Использование gpasm с «make» 8
2.1.2 Работа с ошибками 8
2.2 Синтаксис 8
2.2.1 Структура файла 8
2.2.2 Выражения 9
2.2.3 Числа 10
2.2.4 Препроцессор 11
2.2.5 Файлы заголовков процессора 12
2.3 Директивы 12
2.3.1 Генерация кода 12
2.3.2 Конфигурация 12
2.3.3 Условное ассемблирование 12
2.3.4 Макросы 12
2.3.5 $ 13
2.3.6 Предложения по структурированию вашего кода 14
2.3.7 Директивы суммарно 14
2.3.8 Расширения высокого уровня 23
2.4 Инструкции 26
2.4.1 Набор инструкций суммарно 26
2.5 Errors/Warnings/Messages 29
2.5.1 Ошибки 30
2.5.2 Предупреждения 31
2.5.3 Сообщения 32
Часть 3 34
gplink 34
3.1 Запуск gplink 34
3.2 gplink выводы 34

3.3 Скрипты линковщика 35
3.4 Стеки 35
Часть 4 36
gplib 36
4.1 Запуск gplib 36
4.2 Создание архива 36
4.3 Другие gplib операции 37
4.4 Формат архива 37
Часть 5 38
5.1 gpdasm 38
5.1.1 Запуск gpdasm 38
5.1.2 Комментарии при дизассемблировании 38
5.2 gpstrip 39
5.2.1 Запуск gpstrip 39
5.3 gpvc 39
5.3.1 Запуск gpvc 39
5.4 gpvo 40
5.4.1 Запуск gpvo 40
Указатель 42

Часть 1

Введение

gputils – это набор инструментов для работы с Microchip (TM) PIC микроконтроллерами. Он включает **gpasm**, **gplink** и **gplib**. Каждое из этих средств предназначено для замены в рамках **open source** соответствующего инструмента Microchip (TM). Это руководство описывает основы использования инструментов. Для более детального знакомства с микроконтроллерами обратитесь к руководствам по тем продуктам PICmicro, которые вы используете.

Данный документ является частью **gputils**.

1.1 Используемые режимы

gputils может использоваться в двух разных случаях: абсолютном режиме asm и настраиваемом asm режиме.

1.1.1 Абсолютный Asm режим

В абсолютном asm режиме, как источник ассемблерного языка, файл непосредственно конвертируется в hex-файл с помощью **gpasm**. Это метод абсолютный, поскольку окончательные адреса жестко закодированы в исходном файле.

1.1.2 Настраиваемый Asm режим

В настраиваемом asm режиме исходный ассемблерный код микроконтроллера разделен на отдельные модули. Каждый модуль ассемблируется в объект с помощью **gpasm**. Этот объект может размещаться «где угодно» в памяти микроконтроллера. А затем используется **gplink** для разрешения символьных ссылок, присвоения окончательных адресов и правки машинного кода с окончательными адресами. Вывод из **gplink** – это абсолютный выполняемый объект.

1.1.3 Какой режим лучше?

Абсолютный режим проще понять и использовать. Он требует только одного инструмента – **gpasm**. Большинство примеров на Microchip web-сайте используют абсолютный режим. Зачем же тогда использовать перемещаемый?

• Код можно писать без заботы об адресах. Что делает его легким в написании и повторном использовании.
• Объекты могут быть архивированы с созданием библиотеки, которую тоже проще использовать повторно.
• Перекомпиляция проекта может быть более быстрой, поскольку вы компилируете только часть изменений.
• Файлы могут иметь локальные имена размещения. Пользователь выбирает, какие символы глобальны.

Большинство инструментов разработки используют перемещаемые объекты по этим причинам. Немного о том, что не относится обычно к микроконтроллерным средствам. Их приложения так малы, что абсолютный режим непрактичен. Для PIC перемещаемый режим имеет один большой недостаток. Под вопросом управление банками и страницами.

1.2 Поддерживаемые процессоры

gputils в настоящее время поддерживает следующие процессоры:

eeprom8	gen	p10f200	p10f202	p10f204	p10f206
p12c508	p12c508a	p12c509	p12c509a	p12c671	p12c672
p12ce518	p12ce519	p12ce673	p12ce674	p12cr509a	p12f508
p12f509	p12f629	p12f635	p12f675	p12f683	p14000
p16c5x	p16cxx	p16c432	p16c433	p16c505	p16c52
p16c54	p16c54a	p16c54b	p16c54c	p16c55	p16c55a
p16c554	p16c557	p16c558	p16c56	p16c56a	p16c57
p16c57c	p16c58a	p16c58b	p16c61	p16c62	p16c62a
p16c62b	p16c620	p16c620a	p16c621	p16c621a	p16c622
p16c622a	p16c63	p16c63a	p16c64	p16c64a	p16c642
p16c65	p16c65a	p16c65b	p16c66	p16c662	p16c67
p16c71	p16c710	p16c711	p16c712	p16c715	p16c716
p16c717	p16c72	p16c72a	p16c73	p16c73a	p16c73b
p16c74	p16c745	p16c747	p16c74a	p16c74b	p16c76
p16c765	p16c77	p16c770	p16c771	p16c773	p16c774
p16c781	p16c782	p16c84	p16c923	p16c924	p16c925
p16c926	p16ce623	p16ce624	p16ce625	p16cr54	p16cr54a
p16cr54b	p16cr54c	p16cr56a	p16cr57a	p16cr57b	p16cr57c
p16cr58a	p16cr58b	p16cr62	p16cr620a	p16cr63	p16cr64
p16cr65	p16cr72	p16cr83	p16cr84	p16f505	p16f54
p16f57	p16f59	p16f627	p16f627a	p16f628	p16f628a
p16f630	p16f636	p16f639	p16f648a	p16f676	p16f684
p16f685	p16f687	p16f688	p16f689	p16f690	p16f716
p16f72	p16f73	p16f737	p16f74	p16f76	p16f767
p16f77	p16f777	p16f785	p16f818	p16f819	p16f83
p16f84	p16f84a	p16f87	p16f870	p16f871	p16f872
p16f873	p16f873a	p16f874	p16f874a	p16f876	p16f876a
p16f877	p16f877a	p16f88	p16f913	p16f914	p16f916
p16f917	p16hv540	p17cxx	p17c42	p17c42a	p17c43
p17c44	p17c752	p17c756	p17c756a	p17c762	p17c766
p17cr42	p17cr43	p18cxx	p18c242	p18c252	p18c442
p18c452	p18c601	p18c658	p18c801	p18c858	p18f1220
p18f1320	p18f2220	p18f2320	p18f2331	p18f2410	p18f242
p18f2420	p18f2431	p18f2439	p18f2455	p18f248	p18f2480
p18f2510	p18f2515	p18f252	p18f2520	p18f2525	p18f2539

```
p18f2550    p18f258     p18f2580    p18f2585    p18f2610    p18f2620
p18f2680    p18f2681    p18f4220    p18f4320    p18f4331    p18f4410
p18f442     p18f4420    p18f4431    p18f4439    p18f4455    p18f448
p18f4480    p18f4510    p18f4515    p18f452     p18f4520    p18f4525
p18f4539    p18f4550    p18f458     p18f4580    p18f4585    p18f4610
p18f4620    p18f4680    p18f4681    p18f6310    p18f6390    p18f6410
p18f6490    p18f64j15   p18f6520    p18f6525    p18f6585    p18f65j10
p18f65j15   p18f6620    p18f6621    p18f6627    p18f6680    p18f66j10
p18f66j15   p18f6720    p18f6722    p18f67j10   p18f8310    p18f8390
p18f8410    p18f8490    p18f84j15   p18f8520    p18f8525    p18f8585
p18f85j10   p18f85j15   p18f8620    p18f8621    p18f8627    p18f8680
p18f86j10   p18f86j15   p18f8720    p18f8722    p18f87j10   rf509af
rf509ag     rf675f      rf675h      rf675k      sx18  sx20
sx28
```

Часть 2

gpasm

2.1 Запуск gpasm

Основной синтаксис для запуска **gpasm**

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

Где опция может быть одной из:

Опция	Значение
a <format>	Производит hex-файл в одном из четырех форматов: inhx8m, inhx8s, inhx16, inhx32 (по умолчанию)
c	Выводит перемещаемый объект
d	Выводит сообщение об ошибке
D symbol[=value]	Эквивалентно «#define <symbol> <value>»
e [ON|OFF]	Расширяет макросы в файл листинга
g	Используется для директив отладки для COFF
h	Отображает сообщение помощи (help)
i	Игнорирует чувствительность к регистру. По умолчанию в **gpasms** обращения «fooYa» и «FOOYA» воспринимаются различно.
I <directory>	Задает директорию включения (include)
l	Выводит список поддерживаемых процессоров
L	Игнорировать директивы nolist
m	Дамп памяти
n	Использовать DOS стиль новой линии (CRLF) в hex-файле. Эта опция не разшена на win32 системах.
o <file>	Альтернативное имя выходного hex-файла
p<processor>	Выбор целевого процессора
q	Выйти
r <radix>	Установить radix, то есть, основание системы счисления, которую

	gpasm использует, когда интерпретирует числа. <radix> может быть одной из «oct», «dec» и «hex» для основания восемь, десять и шестнадцать соответственно. По умолчанию это «hex».
v	Выводит информацию о версии **gpasm** и выходит
w [0 | 1|2]	Устанавливает уровень сообщений
y	Разрешает 18xx расширенный режим

Пока иное не задано **gpasm** удаляет суффикс «.asm» из входного файла, замещая его «.lst» и «.hex» для выходных файлов list и hex соответственно. На большинстве современных операционных систем это важно в именах файлов. Из этих соображений вы должны обеспечить, чтобы имена файлов образовывались правильно, и чтобы суффикс «.asm» любого исходного кода был в нижнем регистре.

gpasm всегда производит «.lst» файл. Если запуск прошел без ошибок, он также производит «.hex» файл или «.o» файл.

2.1.1 Использование gpasm с «make»

На большинстве операционных систем вы можете построить проект, используя утилиту make. Для использования **gpasm** с make вы можете получить «makefile» подобно этому:

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

Это перестроит «tree.hex», когда бы либо «tree.asm», либо «treedef.inc» файлы ни менялись. Более общий пример использования **gpasm** с makefile – это пример, включенный в исходный дистрибутив **gpasm**.

2.1.2 Работа с ошибками

gpasm не создает специально файл ошибок. Это может вызвать проблемы, если вы хотите сохранять запись ошибок, или если ваш ассемблер производит столь много ошибок, что они переполняют весь экран. Чтобы работать с этим, если ваш командный процессор «sh», «bash» или «ksh», вы можете сделать что-нибудь в этом роде:

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

Это перенаправит стандартный вывод ошибок на стантдартный вывод (2>&1), затем создаст канал этого вывода в «tee», который скопируется в «tree.err», а затем отобразится на дисплее.

2.2 Синтаксис

2.2.1 Структура файла

gpasm файлы источника состоят из ряда строк. Строки могут содержать метку (начинающуюся в столбце 1) или операцию (начинающуюся в столбце, следующем за 1), и то и другое, или ничего. Комментарии следуют за символом «;» и рассматриваются как новая линия. Метки могут быть любой последовательностью букв A-z,

цифр 0-9 и подчеркиваний («_»); они не могут начинаться с цифр. Метки могут сопровождаться двоеточием («:»).

Операция – это единственный идентификатор (те же правила, что и для метки выше), сопровождаемый пробелом и отделенными запятыми параметрами. Например, следующее допустимо для исходных строк:

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

2.2.2 Выражения

gpasm поддерживает полный набор операторов, базируемых на наборе операторов C. Операторы в следующей таблице собраны в группы эквивалентной значимости, но группы расположены в порядке увеличения важности. Когда **gpasm** вычисляет операторы эквивалентной значимости, он всегда проходит слева направо.

Оператор	Описание
=	присваивание
||	логическое или
&&	логическое и
&	побитовое и
|	побитовое или
^	побитовое исключающее или
<	меньше чем
>	больше чем
==	равно
!=	не равно
>=	больше чем или равно
<=	меньше чем или равно
<<	сдвиг влево
>>	сдвиг вправо
+	сложение
-	вычитание
*	умножение
/	деление
%	по модулю

UPPER	старший байт
HIGH	старший байт
LOW	младший байт
-	отрицание
!	логическое не
~	побитовое не

Любому символу, появляющемуся в столбце 1 может присваиваться значение с использованием оператора присваивания (=) в предыдущей таблице. Дополнительно любое значение, прежде присвоенное, может модифицироваться с использованием одного из операторов в таблице ниже. Каждый из этих операторов оценивает текущее значение символа и затем назначает новое значение, определяемое оператором.

Оператор	Описание
=	присваивание
++	увеличение на 1 (инкремент)
--	уменьшение на 1 (декремент)
+=	инкремент
-=	декремент
*=	умножение
/=	деление
%=	по модулю
<<=	сдвиг влево
>>=	сдвиг вправо
&=	побитовое и
|=	побитовое или
^=	побитовое исключающее или

2.2.3 Числа

gpasm дает вам несколько способов задания чисел. Вы можете использовать синтаксис, который использует начальный символ для задания основания счисления. Следующая таблица обобщает альтернативы. Отметьте C-стиль опции для обозначения шестнадцатеричных чисел.

Основание	Основной синтаксис	Десятичное 21 пишется как
binary	B'[01]*’	B'10101’
octal	O'[0-7]*’	O'25’
decimal	D'[0-9]*’	D'21’

binary	B'[01]*’	B'10101’
hex	H'[0-F]*’	H'15’
hex	0x[0-F]*	0x15

Когда вы пишете число без специального префикса, как, например, 45, **gpasm** использует текущую систему счисления (основание) для интерпретации числа. Вы можете изменить эту систему счисления директивой RADIX или опцией «-r» в командной линии **gpasm**. По умолчанию система счисления шестнадцатеричная.

Если вы не начинаете шестнадцатеричное число с цифры, **gpasm** сделает попытку интерпретировать то, что вы написали как идентификатор. Например, вместо написанного C2, запишет либо 0C2, 0xC2 или H'C2'.

Случай не важный при интерпретации чисел: 0ca, 0CA, h'CA’ и H'ca’ все эквивалентны.

Несколько допустимых числовых форматов **mpasm** также поддерживаются. Эти форматы имеют разные недостатки, но все еще поддерживаются. Таблица ниже суммирует их.

Основание	Основной синтаксис	Десятичное 21 пишется как
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

Вы можете записать ASCII код для символа X, используя 'X' или A'X'.

2.2.4 Препроцессор

Строка вида:

```
include foo.inc
```

заставит **gpasm** извлечь строки файла «foo.inc» до конца файла, а затем вернуться к оригинальному файлу исходного кода к строке, следующей за include.

Строки, начинающиеся с «#» – это директивы процессора, и обрабатываются по разному **gpasm**. Они могут содержать «#define» или «#undefine» директивы.

Когда **gpasm** работает, строки вида:

```
#define X Y
```

каждое последовательное обнаружение X заменят на Y, пока не будет достигнут конец файла или строка

```
#undefine X
```

появится.

Препроцессор будет замещать обнаруженные #v(выражение) в символе со значением «выражения» в

десятичном формате. В следующих выражениях:

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

gpasm заменит символ «label_30_suffix» ” значением 0x10 в таблице символов.

Препроцессор в **gpasm** только похож на препроцессор C; его синтаксис довольно отличен от препроцессора C. **gpasm** использует простой внутренний препроцессор для реализации «include», «#define» и «#undefine».

2.2.5 Файлы заголовков процессора

gputils распространяет файлы заголовков Microchip процессора. Эти файлы содержат специфические данные процессора, которые полезны при разработке приложений PIC. Местоположение этих файлов сообщается в **gpasm** help сообщении. Используйте директиву INCLUDE для использования подходящего файла в вашем исходном коде. Требуется только имя файла. **gpasm** найдет путь по умолчанию автоматически.

2.3 Директивы

2.3.1 Генерация кода

В абсолютном режиме используйте директиву ORG для установки памяти PIC в место, где **gpasm** начнет ассемблировать код. Если вы не зададите адрес с помощью ORG, **gpasm** присвоит 0x0000. В перемещаемом режиме используйте директиву CODE.

2.3.2 Конфигурация

Вы можете выбрать защитные установки для вашей PIC реализации, используя директиву __CONFIG, так что hex-файл явно установит защиту. Конечно вы должны удостовериться, что эти установки соответствуют вашей аппартной PIC разработке.

Директивы __MAXRAM и __BADRAM определят, какое положение RAM допустимо. Эти директивы, по большей части, используются в специальных файлах конфигурации процессора.

2.3.3 Условное ассемблирование

Директивы IF, IFNDEF, IFDEF, ELSE и ENDIF позволяют вам ассемблировать некоторые секции кода только, если встречается условие. Сами по себе они не приводят к генерации **gpasm** какого-либо PIC кода. Пример в разделе 2.3.4 демонстрирует условное ассемблирование.

2.3.4 Макросы

gpasm поддерживает простую схему макроса; вы можете определить и использовать макросы подобные этому:

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

Более полезный пример некоторых макросов для использования:

```
; 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
```

Этот рекурсивный макрос генерирует код для умножения W на постоянную «factor», и сохранения результата в «reg».
Так что написание:

```
scale tmp,D'10’
```

равносильно написанию:

```
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 $

$ распространяется на адреса инструкций, в настоящее время ассемблируемых. Если используется в контексте другом, чем инструкция, таком как условия, распространяется на адреса следующих обнаруженных инструкций, пока ассемблер инкрементирует текущие адреса после ассемблирования инструкции. С $ можно обходиться как и с любыми другими числами:

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

и может быть использовано для записи циклов без меток.

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

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

2.3.6 Предложения по структурированию вашего кода

Вложенные IF операции могут быстро стать запутанными. Отступы – это один из путей сделать код яснее. Другой способ – добавить фигурные скобки к IF, ELSE и ENDIF, подобным образом:

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

После того, как вы сделали это, вы можете использовать ваш текстовый редактор с show-matching-brace для проверки частей IF структуры. В vi эта команда – «%», в emacs это M-C-f и M-C-b.

2.3.7 Директивы суммарно

__BADRAM

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

Инструктирует **gpasm**, что нужно генерировать ошибку, если как-то используется заданное место RAM. Специфицирует диапазон адресов с помощью <lo>-<hi>. См. любой файл заголовков определенного процессора в качестве примера.

См. такеж: __MAXRAM

__CONFIG

```
__CONFIG <expression>
```

Устанавливает конфигурацию защиты PIC процессора.

__IDLOCS

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

Устанавливает местоположения идентификации PIC процессора. Для 12 и 14 битовых процессоров устанавливается четыре id места в шестнадцатеричное значение выражения. Для 18cxx устройств id место expression1 устанавливается в шестнадцатеричное значение expression2.

__MAXRAM

```
__MAXRAM <expression>
```

Инструктирует **gpasm**, что попытка использовать любое место RAM выше заданного должно считаться ошибкой. См. любой файл заголовка конкретного процессора в качестве примера.

См. также: __BADRAM

BANKISEL

```
BANKISEL <label>
```

Эта директива генерирует код выбора банка для косвенного доступа к адресам обозначенным <label>. Директива не доступна для всех устройств. Она доступна только для 14 битовых и 16 битовых устройств. Для 14 битовых устройств код выбора банка будет установлен/очищен IRP битом регистра STATUS. Она использует MOVLB или MOVLR в 16 битовых устройствах.

См. также: BANKSEL, PAGESEL

BANKSEL

```
BANKSEL <label>
```

Эта директива генерирует код выбора банка для банка содержащего <label>. Код выбора банка будет устанавливать/очищать биты в FSR для 12 битовых устройств. Он будет устанавливать/очищать биты в регистре STATUS для 14 битовых устройств. Она будет использовать MOVLB или MOVLR в 16 битовых устройствах. MOVLB будет использоваться для улучшенных 16 битовых устройств.

См. также: BANKISEL, PAGESEL

CBLOCK

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

Маркирует начало блока константами <label>. **gpasm** размещает значения для символов в блок, начинающийся со значения <expression>, заданного CBLOCK. Опционное значение <increment> оставляет пространство после <label> перед следующей <label>.

См. также: EQU

CODE

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

Только для перемещаемого режима. Создает секцию нового машинного кода в выходном объектном файле, <label> задает имя секции. Если <label> не специфицирует имя, будет использовано имя по умолчанию «.code». <expression> не обязательно и задает абсолютный адрес секции.

См. также: IDATA, UDATA

CONSTANT

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

Постоянно присваивает значение, полученное при вычислении <expression> символу <label>. Похоже на SET и VARIABLE, исключая то, что не может быть изменено после присваивания.

См. также: EQU, SET, VARIABLE

DA

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

Сохраняет Strings (строки) в программной памяти. Данные сохраняются как одно 14 битовое слово, представляющее два 7 битовых ASCII символа.

См. также: DT

DATA

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

Генерирует заданные данные.
См. также: DA, DB, DE, DW

DB

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

Декларирует байт данных. Значения упаковываются по два в слово.
См. также: DA, DATA, DE, DW

DE

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

Определяет данные EEPROM. Каждый символ – это строка, сохраняемая в отдельном слове.
См. также: DA, DATA, DB, DW

DT

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

Генерирует заданные данные как байты в последовательности RETLW инструкций.
См. также: DATA

DW

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

Декларирует данные одного слова.
См. также: DA, DATA, DB, DW

ELSE

```
ELSE
```

Маркирует альтернативную секцию условного ассемблерного блока.
См. также: IF, IFDEF, IFNDEF, ELSE, ENDIF

END

```
END
```

Маркирует end исходного файла.

ENDC

```
ENDC
```

Маркирует end для CBLOCK.
См. также: CBLOCK

ENDIF

```
ENDIF
```

Заканчивает условный ассемблерный блок.
См. также: IFDEF, IFNDEF, ELSE, ENDIF

ENDM

```
ENDM
```

Заканчивает определение макроса.
См. также: MACRO

ENDW

```
ENDW
```

Заканчивает цикл while.
См. также: WHILE

EQU

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

Постоянно присваивает значение полученное вычислением выражения <expression> символу <label>. Похоже на SET и VARIABLE, исключая невозможность изменения после присваивания.
См. также: CONSTANT, SET

ERROR

```
ERROR <string>
```

Вызывает сообщение об ошибке.
См. также: MESSG

ERRORLEVEL

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

Sets the types of messages that are printed.

Установки	Влияние
0	Выводятся сообщения, предупреждения, ошибки.
1	Выводятся предупреждения и ошибки.
2	Выводятся ошибки.
-<msgnum>	Препятствует выводу сообщения <msgnum>.
+<msgnum>	Разрешает вывод сообщения <msgnum>.

См. также: LIST

EXTERN

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

Только для перемещаемого режима. Декларирует новы символ, который определен в другом объектном файле.
См. также: GLOBAL

EXITM

```
EXITM
```

Немедленно возвращает из макро-расширения в процессе ассемблирования.
См. также: ENDM

EXPAND

```
EXPAND
```

Расширяет макрос в файле листинга.
См. также: ENDM

FILL

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

Генерирует обнаружение <count> слова программы или байта <expression>. Если выражение заключено в скобки, выражение – строка ассемблера.
См. также: DATA DW ORG

GLOBAL

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

Только для перемещаемого режима. Объявляет символ, как глобальный.
См. также: GLOBAL

IDATA

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

Только для перемещаемого режима. Создает секцию новых начальных данных в выходном объектном файле. <label> задает имя секции. Если <label> не задано, будет использовано имя по умолчанию «.idata». <expression> не обязательно и специфицирует абсолютный адрес секции. Память для данных локализуется и начальные данные помещаются в ROM. Пользователь должен обеспечить код для загрузки данных в память.
См. также: CODE, UDATA

IF

```
IF <expression>
```

Начинает ассемблерный условный блок. Если значение, полученное вычислением <expression> – true (то есть, не

нулевое), код, относящийся к следующим ELSE и ENDIF ассемблируется. Если значение – false (то есть, нуль), код не ассемблируется до соответствующих ELSE или ENDIF.

См. также: IFDEF, IFNDEF, ELSE, ENDIF

IFDEF

```
IFDEF <symbol>
```

Начинает условный блок ассемблера. Если <symbol> появляется в таблице символов, **gpasm** ассемблирует следующий код.

См. также: IF, IFNDEF, ELSE, ENDIF

IFNDEF

```
IFNDEF <symbol>
```

Начинает условный ассемблерный блок. Если <symbol> не появляется в таблице символов, **gpasm** ассемблирует следующий код.

См. также: IF, IFNDEF, ELSE, ENDIF

LIST

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

Разрешает вывод в файл листинга (.lst). Все аргументы интерпретируются как десятичные, не зависимо от текущих установок системы счисления. «list n=0» может использоваться для предотвращения прерывания страницы в секции кода файла листинга. Другие опции показаны в таблице ниже:

Опция	Описание
b=nnn	Устанавливает пространство таблицы
f=<format>	Устанавливает формат hex-файла. Может быть inhx8m, inhx8s, inhx16 или inhx32.
mm=[ON|OFF]	Карта памяти on или off
n=nnn	Устанавливает число строк на страницу
p = <symbol>	Устанавливает текущий процессор
pe = <symbol>	Устанавливает текущий процессор и разрешает 18xx расширенный режим
r=[oct | dec|hex]	Устанавливает систему счисления
st = [ON |OFF]	Дамп таблицы символов on или off
w=[0 | 1| 2]	Устанавливает уровень сообщения
x=[ON|OFF]	Макро расширение on или off

См. также: NOLIST, RADIX, PROCESSOR

LOCAL

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

Объявляет <symbol> как локальный для макроса, который в настоящий момент определяется. Это означает, что дальнейшее обнаружение <symbol> в определении макроса сошлется на локальную переменную, с границами и временем жизни, ограниченными выполнением макроса.

См. также: MACRO, ENDM

MACRO

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

Объявляет макрос с именем <label>. **gpasm** замещает любые обнаружения <symbol> в определении макроса с параметрами, заданными в обращении к макросу.

См. также: LOCAL, ENDM

MESSG

```
MESSG <string>
```

Записывает <string> в файл листинга и в стандартный вывод ошибок.

См. также: ERROR

NOEXPAND

```
NOEXPAND
```

Выключает расширение макроса в файле листинга.

См. также: EXPAND

NOLIST

```
NOLIST
```

Запрещает вывод файла листинга.

См. также: LIST

ORG

```
ORG <expression>
```

Устанавливает место, в которое будут помещены инструкции. Если исходный файл не задает адрес с помощью ORG, **gpasm** принимает, что ORG равно нулю.

PAGE

```
PAGE
```

Приводит к переходу файла листинга на следующую страницу.

См. также: LIST

PAGESEL

```
PAGESEL <label>
```

Эта директива генерирует страницу выбранного кода, чтобы установить биты страницы на страницу,

содержащую назначенную <label>. Страница выделенного кода будет устанавливать/очищать биты в STATUS длч 12 битовых и 14 битовых устройств. Для 16 битовых устройств генерируется MOVLW и MOVWF для модификации PCLATH. Директива игнорируется для улучшенных 16 битовых устройств.

См. также: BANKISEL, BANKSEL

PROCESSOR

```
PROCESSOR <symbol>
```

Выбирает целевой процессор. См. раздел ?? с более детальным описанием.

См. также: LIST

RADIX

```
RADIX <symbol>
```

Выбирает предопределенную систему счисления из «oct» для восьмеричной, «dec» для десятичной и «hex» для шестнадцатеричной. **gpasm** использует эту систему счисления для интерпретации чисел, не имеющих явной системы счисления.

См. также: LIST

RES

```
RES <mem_units>
```

Приводит к тому, что указатель на место в памяти будет перведен к <mem_units>. Может быть использован для резервирования хранилища данных.

См. также: FILL, ORG

SET

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

Временно присваивает значение полученное вычислением <expression> символу <label>.

См. также: SET

SPACE

```
SPACE <expression>
```

Вставляет число <expression> чистых строк в файл листинга.

См. также: LIST

SUBTITLE

```
SUBTITLE <string>
```

Эта директива создает вторую строку программного заголовка для использования в качестве подзаголовка выходного листинга. <string> – это строка ASCII заключенная в двойные кавычки, не длиннее 60 символов.

См. также: TITLE

TITLE

```
TITLE <string>
```

Эта директива создает строку программного заголовка для использования в качестве заголовка выходного листинга. <string> – это ASCII строка, заключенная в двойные кавычки, не длиннее 60 символов.

См. также: SUBTITLE

UDATA

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

Только для перемещаемого режима. Создает новую секцию не инициализированных данных в выходном объектном файле. <label> задает имя секции. Если <label> не задано, будет использовано имя по умолчанию «.udata». <expression> не обязательно и задает абсолютный адрес секции.

См. также: CODE, IDATA, UDATA_ACS, UDATA_OVR, UDATA_SHR

UDATA_ACS

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

Только для перемещаемого режима. Создает новую секцию не инициализированных банка доступа данных в выходном объектном файле. <label> задает имя секции. Если <label> не задано, будет использовано имя по умолчанию «.udata_acs». <expression> не обязательно и задет абсолютный адрес секции.

См. также: CODE, IDATA, UDATA

UDATA_OVR

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

Только для перемещаемого режима. Создает новую секцию не инициализированных перезаписываемых данных в выходном объектном файле. <label> задает имя секции. Если <label> не задано, будет использовано имя по умолчанию «.udata_ovr». <expression> не обязательно и задает абсолютный адрес секции.

См. также: CODE, IDATA, UDATA

UDATA_SHR

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

Только для перемещаемого режима. Создает новую секцию не инициализированных общего банка данных в выходном объектном файле. <label> задает имя секции. Если <label> не задано, будет использовано имя по умолчанию «.udata_shr». <expression> не обязательно и задает абсолютный адрес секции.

См. также: CODE, IDATA, UDATA

VARIABLE

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

Объявляет переменную с именем <label>. Значение <label> может позже быть присвоено заново. Значение <label> не должно присваиваться при объявлении.

См. также: CONSTANT

WHILE

```
WHILE <expression>
```

Создает цикл пока <expression> истинно.

См. также: ENDW

2.3.8 Расширения высокого уровня

gpasm поддерживает несколько директив для использования с языками высокого уровня. Эти директивы легко идентифицируются, поскольку начинаются с «.». Они доступны только в перемещаемом режиме.

Эти возможности расширенные и требуют знания того, как работают перемещаемые объекты **gputils**. Эти возможности предназначены для использования компиляторами. Ничто не мешает им использоваться с ассемблером.

.DEF

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

Создает новый COFF <symbol>. Опции перечислены в таблице ниже:

Опция	Описание
absolute	Ключевое слово абсолютного символа
class=nnn	Устанавливает символ класса (размер в байтах)
debug	Ключевое слово символа ошибоки
extern	Ключевое слово внешнего символа
global	Ключевое слово глобального символа
size=nnn	Резервирование слов или байт для символа
static	Ключевое слово статического символа
type=nnn	Установка типа символа (укороченного)
value=nnn	Установка значения символа

Эти директивы дают пользователю хороший способ управления таблицей символов. Контроль необходим, но если используется неправильно, то может привести ко многим нежелательным последствиям. Он может легко стать причиной ошибок в процессе линковки или неправильный машинный код. Пользователь должен полностью понимать операцию **gputils** COFF таблицы символов, прежде чем модифицировать ее содержимое.

Для достижения лучших результатов только одно ключевое слово должно использоваться. Ключевое слово должно сопровождаться именем символа. Затем за ключевым словом должно идти любое выражение, которое непосредственно устанавливает значение. Вот пример:

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

См. также: .DIM

.DIM

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

Создает <number> вспомогательных символов, прикрепленных к <symbol>. Заполняет вспомогательные символы

значениями заданными в <expression>. Выражения должны получаться в значениях, размещаемых в байтах, когда вычисляются или быть строковыми. Символ должен быть COFF символом.

Эта директива генерирует ошибку, если символ уже имеет вспомогательные символы. Это защищает пользователя от неприятностей с автоматически генерируемыми символами.

Каждый вспомогательный символ имеет 18 байт. Так что содержимое, заданное выражениями, должно быть меньше или равно 18 * <number>.

gpasm не использует вспомогательные символы. Так что содержимое не сказывается на операциях программы. Однако содержимое может быть использовано **gplink** или средствами других производителей.

См. также: .DEF

.DIRECT

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

Предоставляет механизм для прямой связи из программы с отладочным окружением. Метод не имеет влияния на исполняемые. Символы будут появляться в обоих файлах COFF и COD.

Каждая директива создает новый COFF символ «.direct». Вспомогательный символ, который прикрепляется, содержит <command> и <string>. Строка должна быть меньше, чем 256 байт. Команда должна иметь значение от 0 до 255. Нет ограничений на содержимое, однако эти сообщения должны согласовываться с отладочной средой. Типичные значения приведены в таблице ниже:

ASCII команда	Описание
a	Определенное пользователем утверждение
A	Assembler/Compiler определенное утверждение
e	Определенные пользователем команды эмулятора
E	Assembler/Compiler определенные команды эмулятора
f	Определенное пользователем printf
F	Assembler/Compiler определенное printf
l	Определенная пользователем log команда
L	Assembler/Compiler/Code генерированный log проверки команды

Символы также содержат адрес, где сообщение было вставлено в ассемблере. Символы с окончательно перемещенными адресами, доступны в исполняемом COFF. Символы также записываются в COD файл. Они могут просматриваться с помощью **gpvc**.

См. также: .DEF, .DIM

.EOF

```
.EOF
```

Директива приводит к тому, что символ конца файла будет помещен в таблицу символов. Обычно этот символ

генерируется автоматически. Директива позволяет пользователю вручную генерировать символ. Директива проходит только, если используется опция «-g» командной строки. Когда эта опция используется, запрещается автоматическая генерация символа.

См. также: .EOF, .FILE, .LINE

.FILE

```
.FILE <string>
```

Директива приводит к тому, что символ файла размещается в таблице символов. Обычно этот символ генерируется автоматически. Директива позволяет пользователю вручную генерировать символ. Директива проходит только, если используется опция «-g» командной строки. Когда эта опция используется, запрещается автоматическая генерация символа.

См. также: .EOF, .FILE, .LINE

.IDENT

```
.IDENT <string>
```

Создает символ .ident COFF и прибавляет вспомогательный символ. Вспомогательный символ указывает на вход в строковую таблицу. Вход содержит <string>. Это ASCII комментарий любой длины. Этот символ не влияет на операцию **gputils**. Обычно он используется для хранения версии компилятора.

См. также: .DEF, .DIM

.LINE

```
.LINE <expression>
```

Директива приводит к тому, что генерируются номера строк COFF. Обычно они генерируются автоматически. Эта директива позволяет пользователю вручную генерировать номера строк. Директива проходит только, если используется опция «-g» командной строки. Когда эта опция используется, автоматическая генерация запрещена. <expression> всегда вычисляется как десятичное не зависимо от текущих установок системы счисления.

См. также: .EOF, .FILE, .LINE

.TYPE

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

Эта директива модифицирует COFF тип содержащегося <symbol>. Символ должен быть определен. Тип должен быть от 0 до 0xffff. Обычно типы определены в coff.inc.

По умолчанию типы символа COFF определены в NULL в **gpasm**. Хотя тип не влияет на компоновку или генерацию исполняемого файла, он помогает в отладочном окружении.

См. также: .DEF

2.4 Инструкции

2.4.1 Набор инструкций суммарно

12 битовые устройства (PIC12C5XX)

Синтаксис	Описание
ADDLW <imm8>	Добавить операнд к W
ADDWF <f>,<dst>	Добавить W к <f>, результат в <dst>
ANDLW <imm8>	And W и слово, результат в W
ANDWF <f>,<dst>	And W и <f>, результат в <dst>
BCF <f>,<bit>	Очистить <bit> в <f>
BSF <f>,<bit>	Установить <bit> в <f>
BTFSC <f>,<bit>	Пропустить следующую инструкцию, если <bit> в <f> сброшен
BTFSS <f>,<bit>	Пропустить следующую инструкцию, если <bit> в <f> установлен
CALL <addr>	Вызывать подпрограмму
CLRF <f>,<dst>	Записать нуль в <dst>
CLRW	Записать нуль в W
CLRWDT	Сбросить сторожевой таймер (watchdog)
COMF <f>,<dst>	Дополнение <f>, результат в <dst>
DECF <f>,<dst>	Декремент <f>, результат в <dst>
DECFSZ <f>,<dst>	Декремент <f>, результат в <dst>, пропустить, если нуль
GOTO <addr>	Перейти к <addr>
INCF <f>,<dst>	Инкремент <f>, результат в <dst>
INCFSZ <f>,<dst>	Инкремент <f>, результат в <dst>, пропустить, если нуль
IORLW <imm8>	Or W и операнд
IORWF <f>,<dst>	Or W и <f>, результат в <dst>
MOVF <f>,<dst>	Переместить <f> в <dst>
MOVLW <imm8>	Переместить слово в W
MOVWF <f>	Переместить W в <f>
NOP	Нет операции
OPTION	
RETLW <imm8>	Загрузить в W операнд и вернуться
RLF <f>,<dst>	Циклический сдвиг <f> влево, результат в <dst>
RRF <f>,<dst>	Циклический сдвиг <f> вправо, результат в <dst>

SLEEP	Ввести режим sleep (ожидание)
SUBWF <f>,<dst>	Вычесть W из <f>, результат в <dst>
SWAPF <f>,<dst>	Обменять полубайты <f>, результат в <dst>
TRIS	
XORLW	Xor W и операнд
XORWF	Xor W и <f>, результат в <dst>

14 битовые устройства (PIC16CXX)

Синтаксис	Описание
ADDLW <imm8>	Добавить операнд к W
ADDWF <f>,<dst>	Сложить W с <f>, результат в <dst>
ANDLW <imm8>	Добавить операнд к W
ANDWF <f>,<dst>	And W и <f>, результат в <dst>
BCF <f>,<bit>	Очистеть <bit> в <f>
BSF <f>,<bit>	Установить <bit> в <f>
BTFSC <f>,<bit>	Пропустить следующую инструкцию, если <bit> в <f> сброшен
BTFSS <f>,<bit>	Пропустить следующую инструкцию, если <bit> в <f> установлен
CALL <addr>	Вызвать подпрограмму
CLRF <f>,<dst>	Записать ноль в <dst>
CLRW	Записать ноль в W
CLRWDT	Сбросить сторожевой таймепр (watchdog)
COMF <f>,<dst>	Дополнение <f>, результат в <dst>
DECF <f>,<dst>	Decrement <f>, result in <dst>
DECFSZ <f>,<dst>	Декремент <f>, результат в <dst>, пропустить, если ноль
GOTO <addr>	Перейти к <addr>
INCF <f>,<dst>	Инкремент <f>, результат в <dst>
INCFSZ <f>,<dst>	Инкремент <f>, результат в <dst>, пропустить, если ноль
IORLW <imm8>	Or W и операнд
IORWF <f>,<dst>	Or W и <f>, результат в <dst>
MOVF <f>,<dst>	Переместить <f> в <dst>
MOVLW <imm8>	Переместить слово в W

MOVWF <f>	Переместить W в <f>
NOP	Нет операции
OPTION	
RETFIE	Возвращение из прерывания
RETLW <imm8>	Загрузить W операндом и вернуться
RETURN	Возвращение из подпрограммы
RLF <f>,<dst>	Циклический сдвиг <f> влево, результат в <dst>
RRF <f>,<dst>	Циклический сдвиг <f> вправо, результат в <dst>
SLEEP	Ввести режим sleep
SUBLW	Вычесть W из слова
SUBWF <f>,<dst>	Вычесть W из <f>, результат в <dst>
SWAPF <f>,<dst>	Поменять полубайты <f>, результат в <dst>
TRIS	
XORLW	Xor W и операнд
XORWF	Xor W и <f>, результат в <dst>

Ubicom процессоры

Для Ubicom (Scenix) процессоров ассемблер поддерживает следующие инструкции в дополнение к тем, что перечислены под «12 битовые устройства» выше.

Синтаксис	Описание
BANK <imm3>	
IREAD	
MODE <imm4>	
MOVMW	
MOVWM	
PAGE <imm3>	
RETI	
RETIW	
RETP	
RETURN	

Специальные макросы

Есть также некоторое количество стандартных дополнительных макросов. Эти макросы:

Синтаксис	Описание
ADDCF <f>,<dst>	Добавить перенос в <f>, результат в <dst>
B <addr>	Переход
BC <addr>	Переход по переносу
BZ <addr>	Переход по нулю
BNC <addr>	Переход по не переносу
BNZ <addr>	Переход по не нулю
CLRC	Сбросить перенос
CLRZ	Сбросить ноль
SETC	Установить перенос
SETZ	Установить ноль
MOVFW <f>	Переместить файл в W
NEGF <f>	Отрицание <f>
SKPC	Пропуск по переносу
SKPZ	Пропуск по нуль
SKPNC	Пропуск по не перенос
SKPNZ	Пропуск по не нуль
SUBCF <f>,<dst>	Вычесть перенос из <f>, результат в <dst>
TSTF <f>	Проверить <f>

2.5 Errors/Warnings/Messages

gpasm записывает каждое сообщение об ошибках в два места:

- стандартный вывод ошибок
- файл листинга (.lst)

Формат сообщений об ошибках:

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

где:

<src-file> – исходный файл, где **gpasm** встречает ошибку

<line> – номер строки

<code> – 3х-значный код ошибки, данный в списке ниже

<description> – краткое описание ошибки. В некоторых случаях содержит дальнейшую информацию об ошибке.

Сообщения об ошибках подходят для анализа emacs в «режиме компиляции». В этой части описаны сообщения об ошибках, производимые **gpasm**.

2.5.1 Ошибки

101 ERROR directive (директива ошибок)

Генерируемая пользователем ошибка. См. директиву ERROR, где описано более детально.

114 Divide by zero (деление на ноль)

gpasm столкнулся с делением на ноль.

115 Duplicate Label (дублирование метки)

Дублирование метки или переопределение символа, который не может быть переопределен.

124 Illegal Argument (неверный аргумент)

gpasm столкнулся с неверным аргументом в выражении.

125 Illegal Condition (неверное условие)

Неверное условие подобно пропущенному ENDIF или ENDW было встречено.

126 Argument out of Range (аргумент выходит из пределов)

Выражение имеет аргумент, который вышел за пределы.

127 Too many arguments (слишком много аргументов)

gpasm столкнулся с выражением со слишком большим количеством аргументов.

128 Missing argument(s) (пропущен аргумент(ы))

gpasm столкнулся с выражением, в котором пропущен, по меньшей мере, один аргумент.

129 Expected (ожидалось)

Ожидался некоторый тип аргумента.

130 Processor type previously defined (тип процессора прежде определялся)

Процессор переопределялся.

131 Undefined processor (не определенный процессор)

Тип процессора не был определен.

132 Unknown processor (неизвестный процессор)

Выбранный процессор не правильный. Проверьте список процессоров в разделе «??».

133 Hex file format INHX32 required (требовался hex-файл формат INHX32)

Адрес выше 32K был задан.

135 Macro name missing (пропущено имя макроса)

Макрос был определен без имени.

136 Duplicate macro name (дублированное имя макроса)

Имя макроса было продублировано.

145 Unmatched ENDM (не соответствующее ENDM)

Обрнаружено ENDM без определения макроса.

159 Odd number of FILL bytes (нечетное число FILL байт)

В устройствах PIC18CXX число байт должно быть четным.

2.5.2 Предупреждения

201 Symbol not previously defined (символ прежде не определен)

Сделана попытка #undefine символ, который не был ранее определен (#define).

202 Argument out of range (аргумент вне пределов)

Аргумент не находится в пределах выделенного пространства.

211 Extraneous arguments (чуждые аргументы)

Не стандартные аргументы были обнаружены в строке.

215 Processor superseded by command line (процессор заменен в командной строке)

Процессор был задан в командной строке и в исходном файле. Предпочтение отдано командной строке.

216 Radix superseded by command line (система счисления заменена в командной строке)

Система счисления была задана в командной строке и в исходном файле. Предпочтение командной строке.

217 Hex format superseded by command line (hex формат задан в командной строке)

Формат hex-файла задан в командной строке и в исходном файле. Предпочтение командной строке.

218 Expected DEC, OCT, HEX. Will use HEX. (ожидается DEC, OCT, HEX. Будет использовано HEX)

gpasm встретил неправильную систему счисления.

219 Invalid RAM location specified (задано неверное место RAM)

gpasm встретил неверное местоположение RAM, заданное директивами __MAXRAM и __BADRAM.

222 Error messages can not be disabled (сообщение об ошибках не может быть запрещено)

Error messages не могут быть запрещены директивой ERRORLEVEL.

223 Redefining processor (переопределение процессора)

Процессор был заново выбран директивой LIST или PROCESSOR.

224 Use of this instruction is not recommanded (использование этой директивы не рекомендуется)

Использование инструкций TRIS и OPTION не рекомендовано для PIC16CXX устройств.

2.5.3 Сообщения

301 User Message (сообщение пользователя)

Сообщение пользователя, вызванное директивой MESSG.

303 Program word too large. Truncated to core size. (слово программы большое, усечено до нужного размера)

gpasm встретил слово программы, большее, чем допустимый размер для выбранного устройства.

304 ID Locations value too large. Last four hex digits used. (значение ID места слишком велико, используются последние четыре hex цифры)

Заданное значение местоположения ID слишком велико.

305 Using default destination of 1 (file) (используется место назначения по умолчанию 1, файл)

Место назначения не было задано, так что используется место по умолчанию.

308 Warning level superseded by command line (уровень предупреждений заменен в командной строке)

Уровень предупреждений был задан в командной строке и исходном файле. Предпочтение командной строке.

309 Macro expansion superseded by command line (макро расширение заменено в командной строке)

Макро расширение было задано в командной строке и исходном файле. Предпочтение командной строке.

Часть 3

gplink

gplink перемещает и связывает (link) **gpasm** COFF объекты и генерирует абсолютный испольняемый COFF.

3.1 Запуск gplink

Основной синтаксис для запуска **gplink**

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

Где опции могут быть одной из:

Опция	Значение
a	Производит hex-файо в одном из четырех форматов: inhx8m, inhx8s, inhx16, inhx32 (по умолчанию)
c	Выводит исполняемый объект
d	Отображает сообщения об ошибках
f <value>	Заполняет неиспользуемое незащищенное пространство памяти значением <value>
h	Показывает help сообщение
I <directory>	Задет директорию include (включения)
l	Запрещает вывод файла листинга
m	Выводит map файл (картирование)
o <file>	Альтернативное имя выходного hex-файла
q	Выход
r	Попытаться переместить не обобщенную секцию данных в общую память, если перемещение прервано
s <file>	Задет скрипт линковщика
-t <value>	Создает секцию стека
v	Выводит информацию версии **gplink** и выходит

3.2 gplink выводы

gplink создает абсолютный исполняемый COFF. Из этого COFF создаются hex и cod файлы. Исполняемый COFF записывается, только если опция «-c» добавлена. Этот файл полезено для сиуляции и разработки с

помощью **mpsim**. Cod-файл используется для симуляции с **gpsim**.

gplink может также создавать map-файл. map-файл сообщает об окончательных адресах, присвоенных **gplink** секциям COFF. Это те же данные, что могут быть просмотрены в исполняемом COFF с помощью **gpvo**.

3.3 Скрипты линковщика

gplink требует скрипта линковщика. Этот скрипт говорит **gplink**, какая память доступна в целевом процессоре. Множество сгенерированных Microchip скриптов устанавливается с **gputils**. Эти скрипты предназначены быть отправной точкой, но для многих приложений они будут работать, как есть.

Если пользователь не задал скрипт линковщика, **gplink** постарается использовать скрипт по умолчанию для процессора, обозначенного в объектном файле. Местоположение по умолчанию для скриптов сообщается в **gplink** help сообщении.

3.4 Стеки

gplink может создавать секцию стека в процессе линковки, используя директиву стека в скрипте линковщика. Та же возможность может быть использована с опцией «-t» в командной строке. **gplink** создаст секцию и два символа. _stack указывает на начало секции стека, а _stack_end указывает на конец.

Часть 4

gplib

gplib создает, модифицирует и распаковывает COFF арихвы. Это позволяет относительной группе объектов комбинироваться в один файл. Затем этот единственный файл передается **gplink**.

4.1 Запуск gplib

Основной синтаксис запуска **gplib**

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

Где опции могут быть одной из:

Опция	Значение
c	Создать новую библиотеку
d	Удалить члена из библиотеки
h	Показать help сообщение
n	Не добавлять индекс символа
q	Покинуть режим
r	Добавить или заменить члена в библиотеке
s	Перечислить глобальные символы в библиотеке
t	Перечислить членов библиотеки
v	Вывести информацию верстии **gplib** и выйти
x	Извлечь члена из библиотеки

4.2 Создание архива

Наиболее общая операция – это создание нового архива:

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

Эта команда создаст новый арихв «math.a», который содержит «mult.o add.o sub.o».

Имя архива «math.a» произвольно. Инструмент не использует расширение файла для определения типа файла. Это свободно может быть «math.lib» или «math».

Когда вы используете библиотеку, просто добавьте ее к списку объектов, передаваемых в **gplink**. **gplink** просканирует библиотеку и извлечет только членов архива, которые требуются для разрешения внешних ссылок. Так что в приложении не будет содержать код всех членов архива.

4.3 Другие gplib операции

Большинство остальных полезно, но будет использоваться гораздо реже. Например, вы можете заместить отдельных членов архива, но большинство людей выберет удаление старого архива и создание нового.

4.4 Формат архива

Формат файла – это стандартный COFF архив. Заголовок добавляется к каждому члену и не модифицированный объект копируется в архив.

Будучи стандартным архивом, он включает индекс символа. Последний просто служит для определения, какой член архива будет извлекаться для разрешения внешних ссылок. Этот индекс не включается в **mplib** архивы. Так что использование **gplib** архивов с Microchip Tools будет, возможно, приводить к появлению проблем, если не использовать опцию «-n» при создании архива.

Часть 5

Утилиты

5.1 gpdasm

gpdasm – это дизассемблер для **gputils**. Он конвертирует hex-файлы, генерируемые **gpasm** и **gplink** в дизассемблерные инструкции.

5.1.1 Запуск gpdasm

Основной синтаксис запуска **gpdasm**

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

Где опции могут быть одной из:

Опция	Значение
c	Декодировать специальную мнемонику
h	Отобразить help сообщение
i	Отобразить информацию hex-файла
l	Список поддерживаемых процессоров
m	Дамп памяти hex-файла
p<processor>	Выбрать процессор
s	Вывести короткую форму вывода
v	Вывести информацию версии **gpdasm** и выйти
y	Разрешить расширенный режим 18xx

gpdasm специально не создает выходной файл. Он делает снимок памяти (dump) своего вывода на экране. Это помогает уменьшить риск того, что хороший исходный файл будет непреднамеренно перезаписан. Если вы хотите создать выходной файл и ваш командный язык – это «sh, bash» или «ksh», вы можете сделать что-то в этом роде:

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

Это перенаправит стандартный вывод в файл «test.dis».

5.1.2 Комментарии при дизассемблировании

- **gpdasm** использует только hex-файл в качестве ввода. Поскольку так, он не имеет способа различить инструкцию и данные в памяти программы.

- Если **gpdasm** встречает неизвестную инструкцию, он использует директиву DW и рассматривает ее как необработанные данные.
- Есть DON'T CARE биты в словах инструкций. Обычно это не проблема. Это может ею быть, однако, если файл с данными в пространстве программной памяти дизассемблирован и затем вновь ассемблирован. Например: **gpdasm** будет рассматривать 0x0060 в 14 битовом устройстве, как NOP. Если вывод затем вновь ассемблируется, **gpasm** присвоит значение 0x0000. Значение изменилось, а оба инструмента вели себя корректно.

5.2 gpstrip

gpstrip манипулирует секциями и таблицами символов объектных файлов **gputils** .

5.2.1 Запуск gpstrip

Основной синтаксис запуска **gpstrip**

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

Где опции могут быть одной из:

Опция	Значение
g	Очистить отладочные символы
h	Показать help сообщение
k	Сохранить символ
n	Удалить символ
o	Альтернативный выходной файл
p	Защищитить данные
r	Удалить секцию
s	Удалить все символы
u	Удалить все символы не нужные для премещения
v	Показать версию
V	Многословный режим
x	Удалить не глобальные символы

5.3 gpvc

gpvc – это обозреватель cod-файла для **gputils**. Он предоставляет легкий путь для просмотра содержимого cod-файлов, сгенерированных **gpasm** и **gplink**.

5.3.1 Запуск gpvc

Основной синтаксис запуска **gpvc**

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

Где опции могут быть одной из:

Опция	Значение
a	Отобразить всю информацию
d	Отобразить заголовок директории
s	Отобразить символы
h	Показать help сообщение
r	Отобразить ROM
l	Отобразить исходный листинг
m	Отобразить пространство отладочных сообщений
v	Вывести информацию версии **gpvc** и выйти

gpvc специально не создает выходного файла. Он делает снимок памяти (dump) своего вывода на экране. Если вы хотите создать выходной файл и ваш командный процессор – это «sh, bash» или «ksh», вы можете сделать что-нибудь подобное:

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

Это перенаправит стандартный вывод в файл «test.dump».

5.4 gpvo

gpvo – это обозреватель COFF объектных файлов для **gputils**. Он предоставляет легкий способ посмотреть содержимое объектов, генерированных **gpasm** и **gplink**.

5.4.1 Запуск gpvo

Основной синтаксис запуска **gpvo**

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

Где опции могут быть одной из:

Опция	Значение
b	Двоичные данные
c	Декодировать специальные мнемоники
f	Заголовок файла
h	Показать help сообщение
n	Подавить имена файлов
s	Данные секции

b	Двоичные данные
t	Символьные данные
v	Вывести информацию версии **gpvo** и выйти
y	Разрешить расширенный режим 18xx

gpvo специально не создает выходного файла. Он делает снимок памяти (dump) своего вывода на экране. Если вы хотите создать выходной файл и ваш командный процессор – это «sh, bash» или «ksh», вы можете сделать что-нибудь подобное:

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

Это перенаправит стандартный вывод в файл «test.dump».

Указатель

.DEF, 23
.DIM, 23
.DIRECT, 24
.EOF, 24
.FILE, 25
.IDENT, 25
.LINE, 25
.TYPE, 25

Архива формат, 37
ASCII, 11

BADRAM, 14
BANKISEL, 14
BANKSEL, 15
bash, 8, 40, 41

CBLOCK, 15
CODE, 15
CONFIG, 14
CONSTANT, 15
Создание архива, 36

DA, 15
DATA, 16
DB, 16
DE, 16
DT, 16
DW, 16

ELSE, 16
END, 16
ENDC, 16
ENDIF, 17
ENDM, 17

ENDW, 17
EQU, 17
ERROR, 17
ERRORLEVEL, 17
EXITM, 18
EXTERN, 18

FILL, 18

GLOBAL, 18
GNU, 3
gpasm опции, 7
gpdasm, 38
gpvc, 39
gpvo, 40

hex-файл, 7

IDATA, 18
IDLOCS, 14
IF, 18
IFDEF, 19
IFNDEF, 19

LIST, 19
LOCAL, 19

MACRO, 20
MAXRAM, 14
MESSG, 20

NOEXPAND, 20
NOLIST, 20

ORG, 20
Другие gplib операции, 37

PAGE, 20
PAGESEL, 20
PROCESSOR, 21

RADIX, 21
RES, 21
Запуск gpdasm, 38
Запуск gplib, 36
Запуск gplink, 34
Запуск gpvc, 39
Запуск gpvo, 40

SET, 21
SPACE, 21
SUBTITLE, 21

TITLE, 21

UDATA, 22
UDATA ACS, 22
UDATA OVR, 22
UDATA SHR, 22

VARIABLE, 22

WHILE, 22