HI-TECH

C

Руководство пользователя

Компилятор Z80 CP/M HI-TECH C V3.09 предоставляется бесплатно для любого использования, частного или коммерческого, строго как есть. Никакие гарантии или поддержка продукта не предоставляются и не подразумеваются.

Вы можете использовать это программное обеспечение для чего угодно, если вы подтверждаете, что авторское право на это программное обеспечение принадлежит HI-TECH Software.

Клайд Смит-Стаббс

Содержание

Компилятор HI-TECH C

Руководство пользователя

Март 1989

1 Введение

Компилятор HI-TECH C представляет собой комплект программного обеспечения, которое переводит программы, написанные на языке C в исполняемый машинный код программы. Доступны версии, компилирующие программы для работы в операционной системе, или которые производят программы для выполнения во встроенных системах без операционной системы.

1.1. Функциональные возможности

Некоторые функциональные возможности HI-TECH C:

- Единственная команда компилирует, ассемблирует и связывает все программы.
- Компилятор выполняет строгую проверку типов и выдает предупреждения о различных конструкциях, которые могут представлять собой ошибки программирования.
- Сгенерированный код является очень маленьким и быстрым в исполнении.
- Полная библиотека времени выполнения обеспечивает реализацию всех стандартных операций ввода-вывода C и другие функции.
- Для всех процедур времени выполнения предоставляется исходный код.
- Включен мощный универсальный макроассемблер.
- Программы могут быть сгенерированы для выполнения в операционной системе или настроены для загрузки в ПЗУ.

PC-DOS/MS-DOS
CP/M-86
Concurrent DOS
Atari ST
Xenix
Unix
CP/M-80

Таблица 1. Поддерживаемые системы

1.2. Системные требования

Компиляторы HI-TECH C работают под операционными системами, перечисленными в Таблице 1. Убедитесь, что имеющаяся версия компилятора соответствует используемой системе. Обратите внимание, что в целом в вашей системе должен быть жесткий диск или два гибких диска (можно использовать один гибкий диск емкостью 800 Кбайт или более). Настоятельно рекомендуется присутствие жесткого диска. Обратите внимание, что родной компилятор CP/M-80 не имеет всех функциональных возможностей, описанных в данном руководстве, поскольку он не обновлялся после V3.09 из-за ограничений памяти. Кросс-компилятор Z80 поддерживает все функции, описанные здесь и может использоваться для создания программ для выполнения в CP/M-80.

1.3. Использование данного руководства

Документация, поставляемая с компилятором HI-TECH C, состоит из двух отдельных справочников объединенных вместе. Учебник, который вы читаете в настоящее время, рассматривает все версии компиляторов (отражая переносимую природу компилятора). Отдельное пособие охватывает машинно-зависимые аспекты вашего компилятора, например установку.

Это руководство предполагает, что вы уже знакомы с языком C. Если это не так, вы должны иметь по крайней мере один справочник, касающийся C, многие из которых доступны в большинстве компьютерных магазинов, например, "Книга о C" Келли и Поля. Другими подходящими материалами являются "Программирование в ANSI C" С.Кочан и "Язык программирования C" Кернигана и Ритчи. Вы должны прочесть главу "*Приступая к работе*" в этом руководстве и главу "*Установка*" в руководстве для конкретного компьютера. Они предоставят вам достаточную информацию, для работы с ознакомительными примерами из используемого вами справочника по языку C.

Если у вас есть начальное понимание языка C, остальная часть данного руководства предоставит вам информацию, для изучения более сложных аспектов C.

Большая часть руководства описывает реализации всех компиляторов HI-TECH C. Отдельное пособие предоставляется для макроассемблера для вашей конкретной машины и другой машинно-зависимой информации.

2 Приступая к работе

При использовании компилятора в системе с жестким диском вам необходимо установить компилятор перед его использованием. Смотрите главу "*Установка*" для получения более подробной информации. При использовании системы с гибкими дисками, как правило, вы должны иметь копию дистрибутивного диска #1 в дисководе A: и сохранять ваши файлы на диск в дисководе B:. Опять смотрите главу "*Установка*" для получения дополнительной информации.

```
main() {
      pntf("Hello, world\n");
}
```

Рисунок 1 Пример программы на языке C

Прежде, чем скомпилировать вашу программу она должна содержаться в файле с расширением .C (или типом, т.е. частью имени файла после '.'). Например, вы можете ввести программу, показанную на Рис. 1 в файл с названием HELLO.C. Вам нужен текстовый редактор, чтобы сделать это. Обычно подойдет любой текстовый редактор, который может создать простой файл ASCII (т.е. не файл типа "текстового процессора"). При использовании редакторов, подобных Wordstar, вы должны использовать "режим недокумента". Если у вас есть программа в таком файле все, что требуется для компиляции, это дать команду C, например, для компиляции HELLO.C, просто введите команду

```
C -V HELLO.C
```

Кросс-компиляторы (т.е. компиляторы, которые работают в одной системе, но создают код для конкретной целевой системы), имеют различные названия драйвера компилятора, например, драйвер кросс-компилятора для 68HC11, называется C68.

Если вы используете систему, основанную на гибких дисках (или систему CP/M) возможно необходимо определить, где искать команду C, например, если команда C находится на диске в дисководе A: и вы работаете с B:, введите команду

```
A:C -V HELLO.C
```

Компилятор выдаст начальное сообщение и затем приступит к исполнению различных проходов компилятора в последовательности, необходимой для компиляции программы. Если вы используете систему, основанную на гибких дисках, в которой компилятор не помещается на одном диске, вам будет предложено сменить диски, всякий раз, когда компилятор не сможет найти проход. В этом случае вы должны вставить копию следующего дистрибутивного диска в дисковод A: и нажать RETURN.

По мере выполнения компилятором каждого прохода, командная строка для этого прохода будет выведена на экран. Это вызвано тем, что используется параметр -V. Он означает - многословный, и, если бы он не был задан, компиляция была бы тихой за исключением начального сообщения. Сообщения об ошибках могут быть перенаправлены в файл при помощи символа перенаправления стандартного вывода, например

```
> somefile
```

После завершения компиляции компилятор выйдет в командный уровень. Вы заметите, что несколько временных файлов, создаваемых во время компиляции, будут удалены, и на диске останется (кроме исходного файла HELLO.C) только исполняемый файл. Имя этого исполняемого файла будет HELLO.EXE для MS-DOS, HELLO.PRG для Atari ST, HELLO.COM для CP/M-80 и HELLO.CMD для CP/M-86. Для кросс-компиляторов он называется HELLO.HEX или HELLO.BIN в зависимости от выходного формата по умолчанию для конкретного компилятора. Чтобы выполнить эту программу, просто введите

```
HELLO
```

и вы должны быть вознаграждены сообщением "Hello, world!" на вашем экране. Если вы будете использовать кросс-компилятор, то вы будете должны поместить программу в ППЗУ или загрузить на целевую систему, чтобы выполнить ее. Кросс-компиляторы не производят программы, исполняемые на хост-системе.

Есть и другие параметры, которые могут использоваться с командой C, но вы не обязаны использовать их без необходимости. Если вы новичок в языке C, то рекомендуем ввести и скомпилировать несколько простых программ (например, взятых из одного из учебников по языку C упомянутых выше) прежде, чем исследовать другие возможности компилятора HI-TECH C.

Есть одно исключение из упомянутого выше. Если вы компилируете программу, которая использует арифметику с плавающей точкой (вещественные числа) вы **должны** указать компилятору, где искать библиотеку операций с плавающей точкой. Это выполняется с помощью параметра -LF в **конце** командной строки, например:

```
C -V FLOAT.C –LF
```

3 Структура компилятора

Компилятор выполняет нескольких проходов. Каждый проход реализован в виде отдельной программы. Обратите внимание, что у пользователя нет необходимости вызвать каждый проход индивидуально, так как команда C выполняет каждый проход автоматически. Обратите внимание, что зависящие от машины проходы по разному называются для каждого процессора, например в их имени присутствует 86 для 8086 и 68K для 68000.

Имеются следующие проходы:

CPP Препроцессор - обрабатывает макросы и условную компиляцию

P1 Проход синтаксического и семантического анализа. Он пишет промежуточный код для чтения генератором кода.

CGEN, **CG86** и т.д. Генератор кода - производит ассемблерный код.

OPTIM, **OPT86** и т.д. Оптимизатор кода - при желании может быть опущен, для сокращения времени компиляции ценой создания большего по размеру и медленного при исполнении кода.

ZAS, **AS86** и т.д. Ассемблер - фактически макроассемблер общего назначения.

LINK Редактор связей - соединяет объектные файлы с библиотеками.

OBJTOHEX Эта утилита преобразует вывод LINK в соответствующий формат исполняемого файла (например, .EXE, .PRG или .HEX).

Проходы вызываются в заданном порядке. Каждый проход считывает файл и записывает в файл для чтения его преемником. Каждый промежуточный файл имеет определенный формат. CPP производит код C без макроопределений и с использованием расширенных макросов. P1 записывает файл, содержащий программу в промежуточном коде. CGEN переводит его в ассемблерный код. AS производит объектный код в двоичном формате, содержащем байты кода вместе с информацией о символах и перемещении. LINK принимает объектные файлы и библиотеки объектных файлов и пишет другой объектный файл. Он может быть в абсолютной форме, или может сохранять информацию о перемещении и быть введен с помощью другой команды LINK.

Имеются также другие утилиты:

LIBR Создает и поддерживает библиотеки объектных модулей.

CREF Производит списки перекрестных ссылок программы на C или ассемблере.

4 Особенности использования

HI-TECH C был разработан для удобного использования. Единственная команда скомпилирует, ассемблирует и скомпонует программу C. Синтаксис команды C следующий:

C [параметры] файлы [библиотеки]

Параметры - ноль или более параметров, каждый из которых содержит тире ('-'), одну ключевую букву, и возможно параметр, после ключевой буквы без пробелов. **Файлы** - один или несколько исходных файлов C, ассемблерных исходных файлов или объектных файлов. **Библиотеки** могут быть нолем или большим количеством имен библиотек или сокращенной формой -lname, которая будет расширена до имени библиотеки libname.lib.

Команда C, в зависимости от определений заданных параметрами, скомпилирует любые указанные исходные файлы C, ассемблирует их в объектный код, если не предусмотрено иное, ассемблирует любые заданные исходные файлы на языке ассемблера, и затем соединит результаты ассемблирования с любыми указанными объектными фалами или библиотеками.

Если команда C будет вызвана без параметров, то она предложит ввести командную строку. Эта командная строка может быть расширена, вводом символа наклонной черту влево ('\') в конце строки. Будет запрошена следующая строка. Если стандартный ввод команд будет из файла (например, вводя C < afile), то командные строки будут считаны из этого файла. В файле может быть задана более чем одна строка, если каждая строка будет завершаться наклонной чертой влево. Обратите внимание, что этот механизм не работает в пакетном файле MS-DOS, т.е. командный файл для команды C должен быть отдельным файлом. В MS-DOS нет механизма для обеспечения длинных командных строк или стандартного ввода из пакетного файла.

Команда C распознает следующие параметры:

-S Оставляет результаты компиляции любых файлов C как вывод ассемблера. Исходный код C будет перемежаться в виде комментариев с кодом ассемблера.

-C Оставляет результаты всех компиляций и ассемблирования в виде объектных файлов. Редактор связей не вызывается. Это позволяет вызвать редактор связей отдельно, или с помощью команды C на более позднем этапе.

-CR Создает список перекрестных ссылок. -CR самостоятельно оставит необработанную информацию о перекрестных ссылках во временном файле, позволяя пользователю выполнить CREF явно, с указанием имени файла, например, -CRFRED.CRF заставит CREF быть вызванным для обработки первичной информации в указанный файл, в данном случае FRED.CRF.

-CPM Только для кросс-компилятора Z80. Создает CP/M-80 файл COM. Если параметр -CPM не задан, кросс-компилятор Z80 использует startoff модуль времени выполнения ПЗУ и производит шестнадцатеричный или бинарный образ. Если задан параметр -CPM, при компоновке используется startoff код времени выполнения CP/M-80 и создает CP/M-80 файл с расширением COM.

-O Вызывает оптимизатор для всего скомпилированного кода. Также предписывает ассемблеру осуществить оптимизацию переходов.

-OOUTFILE Определяет имя для создаваемого исполняемого файла. По умолчанию имя для исполняемого файла является производным от имени первого исходного или объектного файла, определенного компилятору. Этот параметр позволяет переопределить значение по умолчанию. Если в заданном имени файла отсутствует точка ('.'), будет добавлено подходящее для конкретной операционной системы расширение, например, -OFRED генерирует файл FRED.EXE в MS-DOS или FRED.CMD в CP/M-86. Для кросс-компиляторов он также обеспечивает средство определения выходного формата, например, определение выходного файла, PROG.BIN заставит компилятор генерировать двоичный файл, а при определении PROG.HEX - генерировать шестнадцатеричный файл.

-V Многословный: каждый шаг компиляции будет отражен, по мере его выполнения.

-I Определяет дополнительный префикс имени файла для использования при поиске файлов #include. В CP/M префикс по умолчанию 0:A: (код пользователя 0, дисковод A). В MS-DOS префикс по умолчанию A:\HITECH\. В системах Unix и Xenix префиксом по умолчанию служит /usr/hitech/include/. Обратите внимание, что в MS-DOS к любому имени каталога, заданному в параметре -I должна быть добавлена заключительная наклонная черта влево. Например, -I\FRED\, а не -I\FRED. В Unix должна быть добавлена завершающая наклонная черта.

-D Определяет символ для препроцессора: например, -DCPM определит символ CPM как будто с помощью #define CPM 1.

-U Сбрасывает ранее определенный символ. Обратное действие -D.

-F Запрашивает редактор связей произвести файл символов для использования с отладчиком.

-R Только для компилятора Z80 CP/M этот параметр будет соединять код для выполнения перенаправление ввода-вывода командной строки и подстановочного расширения в именах файлов. См. описание _getargs() в *Приложении 2* для уточнения синтаксиса перенаправления.

-X Очищает локальные символы из любых файлов, скомпилированных, ассемблированных или соединенных. Останутся только глобальные символы.

-M Предписывает редактору связей создать карту ссылок.

-A Этот параметр только для Z80, заставит компилятор производить исполняемую программу, которая, при выполнении, переместит себя к вершине TPA (области транзитных программ). Это позволяет писать программы, которые могут выполнить другие программы под собой. Обратите внимание, что программа, скомпилированная таким способом, автоматически не сбросит адрес базовой дисковой операционной система в расположении 6, чтобы защитить себя. Это должно быть сделано непосредственно самой программой.

Для кросс-компиляторов он обеспечивает способ определения адресов для редактора связей, в которых должна быть соединена скомпилированная программа. Формат параметра -AROMADR, RAMADR,

RAMSIZE. ROMADR является адресом ПЗУ в системе и куда будут помещены исполняемый код и инициализируемые данные. RAMADR - начальный адрес ОЗУ и куда будет помещена psect bss, т.е. неинициализированные данные. RAMSIZE – размер памяти, доступной программе, и используется для установки вершины стека.

Компилятор для 6801/6301/68HC11 в параметре -A принимает четвертое значение, которое является адресом четырехбайтовой прямой области страниц, названной ctemp, которую скомпилированный код использует в качестве временной памяти. Если в параметре -A адрес ctemp опущен, он по умолчанию принимает значение адреса 0. Обычно это приемлемо, однако некоторые разновидности 6801 (как 6303) имеют порты ввода-вывода, отображенные в памяти по адресу 0, и для них прямую страницу ОЗУ начинают по адресу $80.

Для большой модели памяти компилятора 8051 параметр -A принимает форму -AROMADR, INTRAM, EXTRAM, EXTSIZE. ROMADR является адресом ПЗУ в системе. INTRAM - начальный адрес внутреннего ОЗУ, куда будет помещена psect rbss. Внутренний стек 8051 начнется после окончания psect rbss. EXTRAM - начальный адрес внешнего ОЗУ, куда будет помещена psect bss. EXTSIZE - размер внешнего ОЗУ, доступного программе, и используется для установки вершины внешнего стека.

-B Для компиляторов, которые поддерживают более одной "модели памяти", этот параметр используется для выбора модели памяти, для которой должен быть сгенерирован код. Формат этого параметра: -Bx, где x - одна или более букв, определяющих используемую модель памяти. Для 8086 этот параметр используется для выбора используемой, одной из пяти моделей памяти: **Tiny** (крошечной), **Small** (маленькой), **Medium** (средней), **Compact** (компактной) или **Large** (большой).

Для компилятора 8051 этот параметр используется для выбора используемой из трех моделей памяти: **Small** (маленькой), **Medium** (средней) или **Large** (большой). Только для компилятора 8051, этот параметр может также использоваться, для выбора статического распределения автоматических переменных, добавляя A в конце параметра -B. Например, -Bsa выбирает маленькую модель со статическим выделением всех переменных, в то время как -Bm выберет среднюю модель с автоматическими переменными, динамически выделяемыми в стеке.

-E По умолчанию компилятор 8086 инициализирует заголовок исполняемого файла, для запроса 64 Кбайтного сегмента данных во время выполнения. Это может быть переопределено параметром -E. Он принимает аргумент (обычно в шестнадцатеричном представлении), который является числом **байтов** (не разделов) выделяемых программе во время выполнения.
Например, -E0ffff0h запросит Мегабайт. Поскольку так много доступно не будет, система выделит столько сможет.

-W Этот параметр задает уровень предупреждений, т.е. он определяет, насколько придирчив компилятор к допустимым, но сомнительным преобразованиям типов и т.п. -W0, позволяет все предупреждающие сообщения (значение по умолчанию), -W1 подавит сообщение "Func() declared implicit int" (Func() неявно объявлена целой). -W3 реко-

мендуется для компиляции кода, первоначально написанного с помощью другого, менее строгого, компилятора. `-W9` подавит все предупреждающие сообщения.

`-H` Этот параметр генерирует файл символов для использования с отладчиками. Формат файла символов описан в другом месте. По умолчанию имя файла символов `l.sym`. Альтернативное имя может быть определено с помощью параметра, например, `-Hsymfile.abc`.

`-G` Как и `-H`, параметр `-G` также генерирует файл символов, но тот, который содержит информацию о номере строки и файле для отладчика на уровне исходного кода. Как и в `-H` может быть определено имя файла. Если он используется в сочетании с `-O`, выполняется только частичная оптимизация, чтобы избежать запутывание отладчика.

`-P` Для родных компиляторов, работающих под управлением `DOS`, `CP/M-86` и `Atari ST` доступно выполнение профилирования. Этот параметр генерирует код для включения выполнения профилирования во время выполнения программы. Также должен быть определен параметр `-H` для предоставления таблицы символов профилировщику `EPROF`.

`-Z` Только для компиляторов версии 5.xx. Параметр `-Z` используется для выбора глобальной оптимизации сгенерированного кода. Для компиляторов 8086 и 6801/6301/68HC11 единственным допустимым параметром `-Z` является `-Zg`. Для компилятора 8051 допустимы параметры `-Z`: `-Zg`, который вызывает глобальную оптимизацию, `-Zs`, который оптимизирует размер и `-Zf`, который оптимизирует скорость. Параметры `s` и `f` могут использоваться с параметром `g`, таким образом, параметры `-Zgf` и `-Zgs` допустимы. Оптимизация скорости и размера взаимоисключающие, т.е. параметры `s` и `f` не могут использоваться вместе.

`-1` Только для компилятора 8086. Запрашивает генерацию кода, который использует дополнительные инструкции процессора 80186. Программа, скомпилированная с параметром `-1`, не выполнится на процессорах 8086 или 8088. Для компиляторов 68000 сгенерирует инструкции для процессора 68010.

`-2` Как и `-1`, но для 80286 и 68020.

`-11` Для компилятора 6801/HC11 этот параметр запросит генерацию инструкций, определенных для процессора 68HC11.

`-6301` Для компилятора 6801/HC11 этот параметр запросит генерацию инструкций, определенных для процессоров 6301/6303.

Некоторые примеры использования команды C:

```
c prog.c
c -mlink.map prog.c x.obj -lx
c -S prog.c
c -O -C -CRprog.crf prog.c prog2.c
c -v -Oxfile.exe afile.obj anfile.c -lf
```

В вышеупомянутых примерах использован верхний и нижний регистр (для привлечения внимания, что компилятор не различает регистр), хотя параметры, определяющие имена, например, -D, по сути, чувствительны к регистру.

Рассмотрим вышеупомянутые примеры по порядку. Первый скомпилирует исходный файл на языке C prog.c и соединит его со стандартной библиотекой C. Второй пример скомпилирует файл prog.c и соединит его с объектным файлом x.obj и библиотекой libx.lib. Распределение адресов соединенной программы будет записано в файл link.map.

Третий пример скомпилирует файл prog.c, оставляя ассемблерный вывод в файле prog.as. Он не собирает этот файл и не вызывает редактор связей. Следующий пример компилирует prog.c и prog2.c, вызывая оптимизатор для обоих файлов, но не выполняет компоновку. Список перекрестных ссылок будет помещен в файл prog.crf.

Последний пример относится к версии компилятора 8086. Он выполняет компиляцию с многословным параметром и компилирует anfile.c без оптимизации в объектный код, создавая anfile.obj, затем afile.obj и anfile.obj будут соединены вместе с библиотекой операций с плавающей точкой (используя параметр -LF) и стандартной библиотекой для создания исполняемой программы xfile.exe (предполагается, что это осуществляется в системе MS-DOS). Можно ожидать, что эта программа использует вычисления с плавающей запятой, если это не так, то параметр -LF был бы не нужен.

Если несколько исходных файлов на C или на ассемблере будут заданы в команде C, то имя каждого файла будет распечатано на консоли, по мере их обработки. Если какие-либо фатальные ошибки произойдут во время компиляции или ассемблирования исходных файлов, то последующие исходные файлы будут обработаны, но редактор связей не будет вызван.

Другие команды, могут выдаваться пользователем, а не автоматически командой C:

ZAS	Ассемблер Z80.
AS86	Ассемблер 8086.
LINK	Редактор связей.
LIBR	Обслуживания библиотек.
OBJTOHEX	Конвертор объектных файлов в шестнадцатеричные.
CREF	Генератор перекрестных ссылок.

В основном, эти команды принимают тот же вид командной строки, что и команда C, т.е. ноль или более параметров (обозначенных предшествующим символом '-'), затем один или более аргументов файлов. Если редактор связей или библиотекарь будут вызваны без параметров, то они запросят командную строку. Это позволяет вводить командные строки, содержащие более 128 байт. Ввод также может быть получен из файла при помощи возможностей перенаправления (см. _getargs() в списке библиотечных функций). Смотрите описание команды C выше. Более подробно эти команды описаны в соответствующих им руководствах.

5 Специфические особенности

Компилятор HI-TECH C имеет много функций, которые, в основном совместимы с другими компиляторами C и способствуют более надежным методам программирования.

5.1. Совместимость со стандартом ANSI C

На момент написания проект стандарта ANSI для языка C находился на завершающей стадии, хотя еще не являлся официальным стандартом. Соответственно, невозможно требовать соблюдения этого стандарта, однако, HI-TECH C включает большинство новых и измененных функций из проекта стандарта ANSI. Именно в этом смысле, большинство людей понимают его "совместимость с ANSI".

5.2. Контроль соответствия типов

Предыдущие компиляторы C использовали нестрогий подход при проверке типов. Это характерно для Unix компилятора C, который позволяет практически произвольное смешение типов в выражениях. Компилятор HI-TECH C выполняют намного более строгую проверку типов, хотя в большинстве случаев выдаются только предупреждающие сообщения, позволяющие продолжить компиляцию, если пользователь знает, что ошибки безвредны. Это происходит, например, когда целочисленное значение присваивается переменной указателя. Сгенерированный код будет почти наверняка соответствовать намерениям пользователя, однако, если на самом деле он представляет ошибку в исходном коде, пользователю предлагается проверить и исправить его при необходимости.

5.3. Имена членов

В ранних компиляторах C имена членов в разных структурах должны были быть разными, за исключением определенных обстоятельств. HI-TECH C, как и большинство последних реализаций C, позволяет пересекаться именам членов в различных структурах и объединениях. Имя члена распознается только в контексте выражения, тип которого соответствует структуре, в которой определен член. На практике это означает, что имя элемента будет распознаваться только справа от оператора '.' или '->', в котором выражение слева от оператора имеет тип структуры или указатель на структуру те же, в которой было объявлено имя элемента. Это позволяет не только без конфликтов повторно использовать имена структуры более чем в одной структуре, но и осуществлять строгую проверку использования членов. Распространенной ошибкой в других компиляторах C является использование имени члена с указателем структуры неправильного типа, или еще хуже с переменной, которая является указателем на простой тип.

Однако, есть возможность избежать это, если пользователь желает использовать в качестве указателя на структуру то, что не декларируется в качестве такового. Это осуществляется использованием преобразования типа. Например, предположим, что необходимо получить доступ к устройству ввода-вывода, отображенному в память, состоящему из нескольких регистров. Объявления и использование могут выглядеть как фрагмент кода на Рис. 2.

```
struct io_dev {
        short   io_status;    /* Состояние */
        char    io_rxdata;    /* rx data */
        char    io_txdata;    /* tx data */
};

#define RXRDY   01            /* rx ready */
#define TXRDY   02            /* tx ready */

/* Определение (абсолютного) адреса устройства */

#define DEVICE  ((struct io_dev *)0xFF00)

send_byte(c) char c; {
        /* Ожидание готовности передатчика */
        while(!(DEVICE->io_status & TXRDY))
                continue;
        /* Отправить байт данных */
        DEVICE->io_txdata = c;
}
```

Рисунок 2 Использование преобразования типа для абсолютного адреса

В этом примере рассматриваемое устройство имеет 16-битный порт состояния и два 8-битных порта данных. Адрес устройства (т.е. адрес его порта состояния) задан в виде (шестнадцатеричного) значения `0FF00`. Этот адрес преобразуется в указатель структуры требуемого типа, чтобы разрешить использование имен членов структуры. Сгенерированный код будет использовать необходимые абсолютные ссылки на память для доступа к устройству.

Некоторые примеры правильного и неправильного использования имен членов показаны на Рис. 3.

```
struct fred {
    char      a;
    int       b;
} s1, *s2;

struct bill {
     float    c;
     long     b;
} x1, *x2;

main() {

    s1.c = 2;      /* неверно - c не является элементом fred */
    s1.a = 2;      /* верно */
    s2.a = 2;      /* неверно - s2 является указателем */
    x2->b = 24L;   /* верно */
    s2->b = x2->b; /* верно, но отметьте преобразование
                      типа long в int */
}
```

Рисунок 3 Примеры использования элементов

5.4. Беззнаковые типы

HI-TECH C реализуют все версии целочисленных типов без знака, т.е. unsigned, char, short, int и long. Если беззнаковая величина сдвигается вправо, сдвиг будет выполняться как логический сдвиг, т.е. нули заносятся в самые правые биты. Аналогично сдвиги вправо величины со знаком, расширяют знак в самые правые биты.

5.5. Арифметические операции

На машинах, где арифметические операции могут быть выполнены более эффективно в длинах короче, чем int, операнды короче, чем int не будут расширены до длины int без крайней необходимости.

Например, если при сложении двух величин char, и сохранении результата в другой величине char, необходимо выполнить арифметику только в 8 битах, в связи с тем, что любое переполнение в верхних 8 битах будет потеряно.

Однако, если сумма двух char сохраняется в int, сложение должно быть выполнено в 16 битах, чтобы гарантировать корректный результат.

В соответствии с проектом стандарта ANSI, операции с плавающей точкой, вместо удвоенных величин будут выполняться с меньшей точностью, а не преобразовываться в двойную точность, а затем обратно.

5.6. Операции со структурами

HI-TECH C в полном объеме реализуют операторы присваивания структур, передачу структур в качестве параметров и функции возвращающие структуры. Пример приведенный на Рис. 4 является функцией, возвращающей структуру. Также показаны некоторые корректные (и недопустимые) использования функций.

```
struct bill {
    char    a;
    int     b;
}

afunc() {
    struct bill     x;
    return x;
}

main() {
    struct bill     a;

    a = afunc();            /* правильно */
    pf("%d", afunc().a);    /* правильно */

    /* недопустимо, afunc() не может быть присвоено,
       поэтому и не может afunc().a   */

    afunc().a = 1;

    /* недопустимо, по той же причине */

    afunc().a++;
}
```

Рисунок 4 Пример функции, возвращающей структуру

5.7. Перечислимые типы

HI-TECH C поддерживает перечислимые типы. Они обеспечивают структурированный способ определения именованных констант.

Область применения перечислимых типов более ограниченная, по сравнению с Unix компилятором C, но более широкая, чем разрешенная в LINT. В частности выражение перечислимого типа может использоваться в размерности массива, в качестве индекса массива или операнда оператора switch. Перечислимые типы могут использоваться в арифметических операциях, и выражения перечислимого типа могут могут сравниваться как для равенство, так и с операторами отношения. Пример использования перечислимого типа показан на Рис. 5.

```
/* a представляет 0, b -> 1 */
enum fred { a, b, c = 4 };

main() {
    enum fred x, y, z;

    x = z;
    if(x < z) func();
    x = (enum fred)3;
    switch(z) {
        case a:
        case b:
        default:
    }
}
```

Рисунок 5 Использование перечислимых типов

5.8. Синтаксис инициализации

Керниган и Ритчи в книге "Язык программирования C" утверждают, что в инициализаторе в некоторых контекстах пары фигурных скобок могут быть опущены. Проект стандарта ANSI предусматривает, что соответствующая программа на C должна или включать все фигурные скобки в инициализаторе или все их опускать. HI-TECH C позволяют опустить любые пары фигурных скобок, если препроцессор компилятора может определить размер любых инициализируемых массивов, и при условии, что не возникает неоднозначность в определении, какие фигурные скобки опущены. Чтобы избежать неоднозначности, если присутствуют любые пары скобок, то должны присутствовать любые скобки, содержащие эти скобки. Компилятор сообщит "initialization syntax" (синтаксис инициализации), если будет присутствовать неоднозначность.

5.9. Прототипы функции

Новая функциональная возможность C, включенная в предлагаемый стандарт ANSI, известная как "прототипы функции", предоставляет C средство проверки параметров, т.е. они позволяют компилятору проверять во время компиляции, что фактические параметры, предоставленные вызову функции, не противоречат формальным параметрам, ожидаемых функцией. Эта функциональная возможность позволяет программисту включать в объявление функции (или внешнее объявление или фактическое определение) типы параметров этой функции.

Например, фрагмент кода, показанный на Рис. 6, показывает два прототипа функции.

```
void fred(int, long, char *);
char *bill(int a, short b, ...) {
    return a;
}
```

Рисунок 6 Прототипы функции

Первый прототип - внешнее объявление функции `fred()`, которая принимает один целочисленный параметр, один параметр в виде длинного целого и один параметр, который является указателем на символ (байт). Любое использование `fred()`, пока декларация прототипа находится в области видимости, заставит фактические параметры быть проверенными по числу и типу с заданными в прототипе, например, если будут предоставлены только два параметра, или предоставлено целочисленное значение для третьего параметра, компилятор сообщит об ошибке.

Во втором примере, функция `bill()` принимает два или более параметра. Первый и второй будут преобразованы в `int` и `short` соответственно, в то время как остальные (если присутствуют) могут иметь любой тип. Символ многоточие (`...`) указывает компилятору, что ноль или более параметров любого типа могут следовать за двумя параметрами. Символ многоточие должен быть последним в списке параметров и не может появиться в качестве единственного параметра в прототипе.

Все прототипы функций должны согласовываться точно, однако допустимо определение функции в старом стиле, т.е. только с названиями параметров в круглых скобках, с последующими объявлениями прототипа, если число и тип параметров согласованы. В этом случае важно, что бы определение функции находилось в области видимости объявления прототипа.

Доступ к неуказанным параметрам (т.е. параметрам, предоставленным, где многоточие появилась в прототипе), должен осуществляться через макросы, определенные в заголовочном файле `<stdarg.h>`. Он определяет макросы `va_start`, `va_arg` и `va_end`. См. `va_start` в перечне библиотечных функции, для получения дополнительной информации.

Обратите внимание, что использование функции, имеющей соответствующий прототип, если этот прототип не находится в области видимости является серьезной ошибкой, т.е. прототип **должен** быть объявлен (возможно в заголовочном файле), прежде чем функция будет вызвана. Несоблюдение этого правила может привести к странному поведению программы. Каждый раз, когда функция вызывается без явного объявления, `HI-TECH C` выдает предупреждающее сообщение ("`func() declared implicit int`"). Хорошей практикой является объявление всех функций и глобальных переменных в одном или более заголовочных файлах, которые включаются везде, где функции определены или упомянуты.

5.10. Void и указатель на Void

Тип `void` может использоваться для указания компилятору, что функция не возвращает значение. Любое использование возвращаемого значения из `void` функции будет отмечено как ошибка.

Тип `void *`, т.е. указатель на `void`, может использоваться в качестве указателя "универсального" типа. Он предназначен, чтобы помочь в написании средств общего назначения выделения памяти и т.п., на которую возвращается указатель, который может быть присвоен указателю на другую переменную некоторого другого типа.

Компилятор разрешает, без приведения типа и не сообщая об ошибке преобразование void * в указатель любого другого типа и наоборот. Программисту рекомендуется осторожно использовать это средство и гарантировать, что любое значение void * применимо в качестве указателя на любой другой тип, например, выравнивание любого такого указателя, должно быть пригодно для хранения любого объекта.

5.11. Квалификаторы типов

Стандарт ANSI C ввел в C понятие квалификаторов типа. Это ключевые слова, уточняют тип, к которому они применены. Квалификаторами типа, определенными ANSI C, являются const и volatile. HI-TECH C также реализуют несколько других квалификаторов типа. Дополнительными квалификаторами являются:

```
far
near
interrupt
fast interrupt
port
```

Не все версии компиляторов реализуют все дополнительные квалификаторы. Смотрите машинно-зависимые разделы для получения дополнительной информации.

При построении объявлений, с использованием квалификаторов типа, очень легко запутаться относительно правильной семантики объявления. Несколько практических правил сделают это проще. Во-первых, когда квалификатор типа появляется в левой части объявления, он может появиться с любым спецификатором класса памяти и основным типом в любом порядке, например.

```
static void interrupt func();
```

семантически то же как

```
interrupt static void func();
```

Если классификатор появляется в этом контексте, он применяется к основному типу объявления. Если классификатор появляется справа от одного или нескольких '*' (звездочка) модификаторов указателя, тогда вы должны прочитать объявление справа налево, например.

```
char * far fred;
```

должен быть прочитан как "fred является far указателем на char". Это означает, что fred квалифицируется как far, а не char, на который он указывает. С другой стороны

```
char far * bill;
```

должен быть прочитан как "bill является указателем на far char", т.е. char, на который указывает bill, располагается в дальнем адресном пространстве. В контексте 8086 компиляторов это означает, что bill 32-битный указатель, в то время как fred 16-битный указатель. Вы услышите bill, ссылается как "дальний указатель", однако терминология "указатель на far" является предпочтительней.

5.12. Встроенный ассемблер

Существует два метода включения встроенного ассемблерного кода в программы C. Первый позволяет разместить несколько строк ассемблера в любом месте программы. Он осуществляется с помощью директив препроцессора #asm и #endasm. Любые строки между этими двумя директивами непосредственно копируются в ассемблерный файл, производимый компилятором. Альтернативно можно использовать конструкцию asm("строка")[1] в любом месте, где ожидается оператор C. Строка непосредственно копируется в ассемблерный файл. При использовании встроенного ассемблера необходимо соблюдать осторожность, так как он может взаимодействовать со сгенерированным кодом компилятора.

5.13. Директивы pragma

Предварительный стандарт ANSI C предусматривает директивы препроцессора #pragma, позволяюшие компилятору управлять различными аспектами процесса компиляции. В настоящее время HI-TECH C поддерживают только одну директиву pragma pack. Она позволяет контролировать порядок, в котором элементы распределяются внутри структуры. По умолчанию некоторые компиляторы (особенно компиляторы 8086 и 68000) выравнивают элементы структуры на четные границы, чтобы оптимизировать доступ машине. Иногда требуется переопределить это для достижения определенного расположения в структуре. Директива pragma pack позволяет устанавливать максимальный коэффициент упаковки. Например, #pragma pack(1) сообщает компилятору, что между элементами структуры никакие заполнения не вставляются, т.е. что все элементы должны быть выровнены по границам, кратным 1. Аналогично #pragma pack(2) позволит выравнивание на границы, кратные 2. Ни в коем случае не используйте pragma pack для выравнивания большего, чем использовалось бы для этого типа данных в любом случае.

В программе могут использоваться несколько директив pragma pack. Действие директивы остается в силе, до изменения другой директивой или до конца файла. Не используйте директиву pragma pack, прежде чем будут включены такие файлы, как <stdio.h>, поскольку это приведет к неправильным объявлениям структур данных библиотеки времени выполнения.

6 Зависимости от машины

HI-TECH C устраняет многие машино-зависимые аспекты C, так как он единообразно реализует такие функции, как unsigned char. Однако существуют некоторые области, где язык C остается зависимым от машины. Программисты должны знать о них и принимать их во внимание при написании переносимого кода.

Самой очевидной зависимостью от машины является переменный размер типов C. На некоторых машинах int будет 16 бит на других, он может быть 32 бита. HI-TECH C соответствует следующим правилам, которые являются обычной практикой в большинстве компиляторов C.

char	не менее 8 бит;	**int**	совпадает с short или long;
short	не менее 16 бит;	**float**	не менее 32 бита;
long	не менее 16 бит;	**double**	имеет размер не менее, чем float.

[1] Не реализована в компиляторе V3.09 для CP/M

Из-за переменной ширины `int` рекомендуется, по мере возможности вместо `int` использовать `short` или `long`. Исключением из этого правила является случай, когда требуется величина соответствующая естественному размеру слова машины.

Еще одной областью зависящей от машины является порядок байтов. Порядок байтов в `short` или `long` может значительно различаться между машинами. Нет простого способа решения этой проблемы, кроме как избегать кода, зависящего от конкретного упорядочивания. В частности следует избегать записи всей структуры в файл (с помощью `fwrite()`), если файл не используется только для считывания той же программой и последующего удаления. Разные компиляторы используют различный размер заполнения между элементами структуры, хотя он может быть изменен с помощью конструкции `#pragma pack(n)`.

6.1. Предопределенные макросы

Одним из методов, с помощью которого можно управлять неизбежной зависимостью от машины, является использование предопределенных макросов, обеспеченных каждым компилятором для идентификации целевого процессора и операционной системы (если имеется). Они определяются драйвером компилятора и могут быть проверены с помощью директив препроцессора условной компиляции.

Макросы, определенные различными компиляторами, перечислены в Таблице 2. Они могут использоваться, как показано в примере под Таблицей 2.

Макрос	Определен для
i8051	Семейства процессоров 8051
i8O86	Семейства процессоров 8086
i8096	Семейства процессоров 8096
z80	Процессора z80 и производных
m68000	Семейства процессоров m68000
m6800	Процессоров 6801, 68HC11 и 6301
m6809	Процессора m6809
DOS	MS-DOS и PC-DOS
CPM	CP/M-80 и CP/M-86
TOS	Atari ST

Таблица 2 Предопределенные макросы

```
#if    DOS
char * filename = "c:file";
#endif /* DOS */
#if    CPM
char* filename = "0:B:afile";
#endif /* CPM */
```

7 Проверка и сообщения об ошибках

Об ошибках может сообщить любой проход компилятора, однако ассемблер и оптимизатор практически не встречаются с ошибками в сгенерированном коде. Ниже приводятся типы ошибок, производимые каждым проходом. Обычно любая ошибка будет обозначена именем исходного файла, в котором она встретились, и номером строки, в котором она была обнаружена. P1 также обозначит имя функции внутри, которой была обнаружена ошибка.

Ошибки могут быть перенаправлены в файл с помощью обычного синтаксиса, т.е. символ "больше" ('>') за которым следует имя файла, в который должны быть записаны ошибки. Конечно, именем файла может, быть имя устройства, например, LST: в CP/M или PRN в MS-DOS.

CPP сообщает об ошибках, касающихся макроопределений и расширений, а также условной компиляции.

P1 является проходом, который сообщает о большинстве ошибок. Он выполняет синтаксическую и семантическую проверку ввода, и сообщит о встреченных фатальных и предупреждающих ошибках. Синтаксические ошибки будут обычно выражаться как "symbol expected" (ожидается символ) или "symbol unexpected" (неожиданный символ). Семантические ошибки могут быть связаны с необъявленными, повторно или неправильно объявленными переменными. P1 также сообщит об определениях переменных, которые являются неиспользованными или не имеющими ссылок. Эти ошибки являются предупреждениями, как и большинство ошибок проверки типов .

Если P1 обнаруживает ошибки, он выводит на экран исходную строку, содержащую ошибку, и ниже отображает сообщение об ошибке и стрелку вверх, указывающую на точку, в которой компилятор обнаружил ошибку. В некоторых случаях фактическая причина ошибки может быть в начале строки или даже на предыдущей строке.

CGEN может иногда сообщать об ошибках, обычно предупреждения, и главным образом связанных с необычными комбинациями типов с константами, например при проверке условия, что величина без знака меньше, чем ноль. CGEN производит одну неустранимую ошибку "can't generate code" (не удается сгенерировать код) для этого выражения, которая означает, что скомпилированное в настоящее время выражение, в некотором роде слишком сложное, чтобы произвести для него код. Обычно она может быть преодолена путем переписывания исходного кода. Такие ошибки встречаются редко, и будут происходить только для необычных конструкций.

Редактор связей сообщит о неопределенных или многократно определенных символах. Обратите внимание, что объявления переменных в заголовочном файле, который включен более чем в один исходный файл, должны быть объявлены как extern, чтобы избежать ошибки многократного определения символа. Затем эти символы должны быть определены в одном и только одном исходном файле.

Полный список сообщений об ошибках включен в приложении.

8 Стандартные библиотеки

8.1. Стандартный ввод-вывод

C является языком, который не определяет средства ввода-вывода в самом языке. Все операции ввода-вывода осуществляются с помощью библиотеки подпрограмм. На практике это приводит к обработке ввода-вывода, которая не менее удобна, чем в любом другом языке с возможностью дополнительной настройки ввода-вывода для конкретного приложения. Например, возможно обеспечить подпрограмму, чтобы заменить стандартную getchar() (которая получает один символ из стандартного ввода) на специальную getchar(). Это особенно полезно при написании кода, который должен работать на специальной аппаратной конфигурации при поддержании высокого уровня совместимости со "стандартным" вводом-выводом C.

Фактически есть библиотека стандартного ввода-вывода (STDIO), которая определяет переносимый набор подпрограмм ввода-вывода. Это подпрограммы, которые обычно используются любой прикладной программой C. Эти подпрограммы, вместе с другими библиотечными подпрограммами, подробно описаны в последующем разделе руководства.

8.2. Совместимость

Библиотеки, поставляемые с HI-TECH C, хорошо совместимы с библиотеками ANSI, а также UNIX V7 на стандартном уровне ввода-вывода и на уровне системных вызовов UNIX. Стандартная библиотека ввода-вывода является полной, и во всех отношениях соответствует стандартной библиотеке ввода-вывода UNIX. Библиотечные подпрограммы, реализующие UNIX-подобные системные вызовы, как функции максимально приближены к этим системным вызовам, однако существуют некоторые системные вызовы UNIX, которые не могут быть смоделированы в других системах, например, операция link(). Несмотря на это, основные низкоуровневые подпрограммы ввода-вывода, т.е. open, close, read, write и lseek идентичны эквивалентам UNIX. Это означает, что многие программы, написанные для запуска в UNIX, даже если они не используют стандартные операции ввода-вывода, с незначительными изменениями будут работать при компиляции с HI-TECH C.

8.3. Библиотеки для встраиваемых систем

Кросс-компиляторы, предназначенные для получения кода для целевых систем без операционных систем, поставляются в комплекте с библиотеками, реализующими подмножество функций STDIO. Не предусмотрены функции, имеющие дело с файлами. Включены printf(), scanf() и т.д. Они работают, вызывая две низкоуровневых функции putch() и getch(), которые обычно отправляют и получают символы через последовательный порт. Они используются, когда компилятор ориентирован на однокристальный микрокомпьютер со встроенным UART. Для всех этих функций предоставляется исходный код, позволяя пользователю изменять его для адресации различных последовательных портов.

8.4. Двоичный ввод-вывод

В некоторых операционных системах, особенно CP/M, файлы обрабатываются по-разному в зависимости от того, содержат они ASCII (т.е. печатаемые) или двоичные данные. MS-DOS также страдает от этой проблемы, не из-за отсутствия чего-либо в самой операционной системе, а скорее из-за наследия от CP/M, в результате чего многие программы, помещают избыточный символ Ctrl-Z в конце файлов. К сожа-

лению, не существует способа определения, какой тип данных содержится в файле (за исключением, возможно, по имени или расширению файла, и это не надежно). Чтобы преодолеть эту трудность, существует дополнительный символ, который может быть включен в стоку mode в вызов fopen(). Чтобы открыть файл для ввода-вывода в формате ASCII используется вызов fopen():

```
fopen("filename.ext", "r")      /* для чтения */
fopen("filename.ext", "w")      /* для записи */
```

Чтобы открыть файл для двоичного ввода-вывода, к параметру может быть добавлен символ 'b'.

```
fopen("filename.ext", "rb")
fopen("filename.ext", "wb")
```

Дополнительный символ сообщает библиотеке STDIO, что этот файл должен быть обработан строго в двоичном режиме. В CP/M или MS-DOS, для файла, открытого в режиме ASCII, подпрограммы STDIO будут выполнять следующую специальную обработку символов:

newline ('\n') при выводе преобразуется в возврат каретки / новая строка;
Return ('\r') игнорируется при вводе;
Ctrl-Z при вводе интерпретируется как конец файла (EOF) и, только в CP/M, добавляется при закрытии файла.

Специальные действия, выполняемые с ASCII файлами, гарантируют, что файл записывается в формате, совместимом с другими программами обработки текстовых файлов, устраняя необходимость любой специальной обработки программой пользователя - файл выглядит в пользовательской программе, как будто он UNIX-подобный текстовый файл.

Ни одно из этих специальных действий не выполняется с файлом, открытым в двоичном режиме. Это требуется при обработке любых двоичных данных, чтобы гарантировать, что не вставляются сомнительные байты, и не встречаются преждевременные EOF.

Так как символ двоичного режима является дополнительным к нормальному символьному режиму, его использование является вполне совместимо с UNIX C. При компиляции в UNIX, дополнительный символ игнорируется.

Упоминание здесь термина "поток" является уместным. Поток используется по отношению к библиотечным подпрограммам STDIO для обозначения источника или приемника байтов (символов), которыми управляют эти подпрограммы. Таким образом, указатель FILE, переданный как параметр подпрограммам STDIO, может рассматриваться как описатель соответствующего потока. Поток может рассматриваться в качестве безликой последовательности байтов, поступающих из или отправляемых в устройство или файл или даже некоторый другой неопределенный источник. Указатель FILE не следует путать с "дескрипторами файлов", используемыми низкоуровневыми функциями ввода-вывода open(), close(), read() и write(). Они образуют независимую группу функций ввода-вывода, которые выполняют не буферизированное чтение и запись в файлы.

8.5. Библиотека операций с плавающей точкой

HI-TECH C поддерживают операции с плавающей точкой как часть языка, однако реализация Z80 обеспечивает только одинарную точность. Двойная точность с плавающей точкой разрешена, но не отличаются от `float`. Кроме того, стандартная библиотека LIBC.LIB, не содержит подпрограмм с плавающей точкой. Они были выделены в другую библиотеку LIBF.LIB. Это означает, что если эта библиотека не просматривается, подпрограммы поддержки с плавающей точкой не включаются в программу, таким образом, предотвращая любое увеличение размера за поддержку с плавающей точкой, если она не используется. Это особенно важно для `printf` и `scanf`, и поэтому LIBF.LIB содержит версии `printf` и `scanf`, которые действительно поддерживают форматы с плавающей точкой.

Таким образом, если используется операции с плавающей точкой, в команде C должен использоваться параметр `-LF` **после** исходных и/или объектных файлов. Например:

```
C -V -O x.c y.c z.obj –LF
```

9 Стилистические соображения

Хотя изложение стандарта кодирования в C не является целью этого руководства, некоторые комментарии относительно использования некоторых функций HI-TECH C могут быть полезными.

9.1. Имена элементов

Несмотря на то, что HI-TECH C позволяет использовать одинаковые имена элементов структуры или объединения в нескольких структурах или объединениях, это не допускается другими компиляторами C. Чтобы обеспечить переносимость кода, рекомендуется, использовать различные имена элементов, и полезным способом гарантировать это является использование в имени каждого элемента префикса из одной или двух букв, производных от имени самой структуры. Пример приведен на Рис. 7.

```
struct tree_node {
    struct tree_node *      t_left;
    struct tree_node *      t_right;
    short                   t_operator;
};
```

Рисунок 7 Именование элементов

Поскольку HI-TECH C требует использования всех промежуточных имен при обращении к элементу, вложенному в несколько структур, некоторые простые макроопределения могут служить сокращением. Пример приведен на Рис. 8.

```
struct tree_node {
    short       t_operator;
    union {
        struct tree_node *  t_un_sub[2];
        char *              t_un_name;
        long                t_un_val;
    }   t_un;
};

#define t_left  t_un.t_un_sub[0]
#define t_right t_un.t_un_sub[1]
#define t_name  t_un.t_un_name
#define t_val   t_un.t_un_val
```

Рисунок 8 Сокращение имени элемента

Это позволяет различные компоненты структуры называть краткими именами, гарантируя переносимость и обеспечивать корректное определение структуры.

9.2. Использование `int`

Рекомендуется по мере возможности избегать типа `int`, отдавая предпочтение типам `short` или `long`. Это вызвано тем, что тип `int` имеет переменной размер, тогда как `short` обычно 16 бит и `long` 32 бита в большинстве реализаций C.

9.3. Объявления extern

Некоторые компиляторы позволяют неинициализированной глобальной переменной быть объявленной больше чем в одном месте с повторными определениями, разрешаемыми редактором связей в предположении, что все определения относятся к одному и тому же объекту. HI-TECH C в частности запрещает это, так как это может привести к трудно уловимым ошибкам. Вместо этого глобальные переменные могут быть объявлены внешними (extern) везде, где вы пожелаете и должны быть определены в одном и только одном месте. Как правило, это реализуется объявлением глобальных переменных как extern в заголовочном файле, и определением каждой переменной в файле наиболее тесно связанном с этой переменной. Такое использование является переносимым практически во всех других реализациях C.

10 Модели памяти

Многие процессоры, поддерживаемые компиляторами HI-TECH C, могут иметь несколько адресных пространств, доступных программе. Обычно одно адресное пространство более экономично к доступу, чем другое, большее адресное пространство. При этом желательно иметь возможность адаптировать использование памяти программы для достижения максимальной экономии в адресации (таким образом, уменьшая размер программы и увеличивая скорость) при предоставлении доступа к достаточному размеру памяти, требуемому программе.

Эта концепция различных адресных пространств не относится к K&R или ANSI C (за исключением возможности использования отдельных адресных пространств для кода и данных). Без каких-либо расширений самого языка можно создать несколько моделей памяти для данного процессора, выбираемых во время компиляции. Это имеет эффект выбора одного метода адресации для всех данных и/или кода. Это позволяет выбрать модель для конкретной программы в зависимости от требования к памяти программы.

Однако, во многих программах только одна или две структуры данных являются достаточно большими, и должны быть помещены в большее адресное пространство. Выбор "large" (большой) модели памяти для всей программы делает всю программу больше и медленнее только, чтобы позволить несколько больших структур данных. Это можно преодолеть, разрешив индивидуальный подбор адресного пространства для каждой структуры данных. К сожалению, это влечет за собой расширения языка, не желательный подход. Чтобы свести к минимуму последствия таких расширений, они должны удовлетворить следующим критериям:

1. Насколько возможно расширения должны быть совместимы с общепринятой практикой.
2. Расширения должны соответствовать машинно-независимой модели, для максимальной переносимости между процессорами и операционными системами.

Эти цели были достигнуты в HI-TECH C посредством следующей модели:

Каждая модель памяти определяет три адресных пространства каждое для кода и данных. Эти адресные пространства известны как пространства near (близкое), far (далекое) и default (по умолчанию). Любой объект, квалифицированный ключевым словом near, будет помещен в адресное пространство near, любой объект, квалифицированный ключевым словом far, должен быть помещен в адресное пространство far, и все другие объекты должны быть помещены в адресное пространство по умолчанию. Адресное пространство near должно быть (возможно, неподходящим) подпространством адресного пространства по умолчанию, в то время как адресное пространство по умолчанию должно быть (возможно, неподходящим) подпространством адресного пространства far. Должно быть, до трех видов указателей, соответствующих этим трем адресным пространствам, каждый способный к адресации объекта в его собственном адресном пространстве или подпространстве этого адресного пространства.

Это подразумевает, что адрес объекта может быть преобразован в указатель в большее адресное пространство, например, для near объекта его адрес можно преобразовать в указатель на far, но удаленный объект, может быть не в состоянии адресоваться указателем near.

На практике адресное пространство по умолчанию, как правило, в точности соответствуют адресным пространствам near или far. Если все три адресных про-

странства соответствуют одной и той же памяти, то возможна только одна модель памяти. Это происходит с процессором 68000. Если код по умолчанию и пространство данных каждое могут соответствовать адресным пространствам `near` или `far`, то в общей сложности возможно четыре модели памяти. Так обстоит дело с процессором 8086.

Ключевые слова `far` и `near` поддерживаются всеми компиляторами HI-TECH C, но точное соответствие адресных пространств определяется индивидуальными особенностями каждого процессора и выбором модели памяти (если есть выбор). Однако, код написанный с использованием этих ключевых слов, будет переносим, при условии, что он удовлетворяет ограничениям модели описанной выше.

Эта модель также хорошо соответствует другим реализациям, использующим ключевые слова `near` и `far`, несмотря на то, что такие реализации, кажется, формально не были разработаны для переносимой модели.

11 Что-то пошло не так, как надо

Компилятор может производить разнообразные сообщения об ошибках. Большинство из них касаются ошибок в исходном коде (разного рода синтаксические ошибки и так далее), но некоторые представляют собой ограничения, в частности, памяти. Два прохода, которые, скорее всего, будут затронуты ограничениями памяти, являются генератор кода и оптимизатор. Генератор кода выдаст сообщение "No room" (нет места), если он исчерпает динамическую память. Они обычно могут быть устранены путем упрощения выражения в строке, указанной в сообщении об ошибке. Чем сложнее выражение, тем больше требуется памяти для хранения дерева представляющего его. Также поможет сокращение количества символов, используемых в программе.

Обратите внимание, что эта ошибка отличается от сообщения, "Can't generate code" (не удается генерировать код) для этого выражения, которое указывает, что выражение в некоторых случаях слишком сложное для обработки генератором кода. Это сообщение встречается довольно редко и может быть устранено путем изменения выражения, например, вычисления промежуточного значения во временную переменную.

Оптимизатор читает ассемблерный код для всей функции в память одновременно. Очень большие функции не поместятся, выдавая сообщение об ошибке "Optim: out of memory in _func" (недостаточно памяти в _func), где func - имя ответственной функции. В этом случае функция должна быть разбита на более мелкие функции. Это происходит только с функциями, содержащими несколько сотен строк исходного кода C. Рекомендуется ограничивать размер функции не более чем 50 строк каждая.

Если проход выходит с сообщением "Error closing file" (ошибка закрытия файла), или "Write error on file" (Ошибка при записи в файл), это обычно означает, что недостаточно места на текущем диске.

Если вы используете редактор обработки текстов, такой как Wordstar, убедитесь, что вы используете режим "не документ" или любой другой соответствующий режим. Отредактированный файл не должен содержать любые символы с установленным старшим битом, и в конце строки должен присутствовать символ перевода строки. Строки должны быть длиной не более 255 символов.

При использовании операций с плавающей точкой убедитесь, чтобы вы использовали флаг -LF в **конце** командной строки, чтобы активизировать поиск библиотеки с плавающей точкой. Это заставит соединить с программой версии printf и scanf с поддержкой плавающей точки, а также специальные программы с плавающей точкой.

Если будет использоваться версия printf без поддержки плавающей точки с вещественным форматом, таким как %f то, она просто распечатает букву f.

Если редактор связей выдает сообщение "Undefined symbol" (неопределенный символ) для некоторого символа, о котором вы ничего не знаете, возможно, что это библиотечная подпрограмма, которая не была найдена во время поиска в библиотеке из-за неправильного упорядочивания библиотеки. В этом случае вы можете выполнить поиск в библиотеке дважды. Например, для стандартной библиотеки добавьте -LC в конце командной строки C или -LF для библиотеки с плавающей точкой. Если вы указали библиотеку по имени, просто повторите ее имя.

12 Ассемблер Z80 Справочное руководство

12.1. Введение

Ассемблер, включенный в систему компилятора HI-TECH C, является полнофункциональным перемещающим макроассемблером, принимающим мнемоники Zilog. Эти мнемоники и синтаксис языка ассемблера Z80 описаны в "Руководстве Ассемблера Z80", опубликованном Zilog, и включены в конце этого руководства в качестве справочного материала. Ассемблер реализует определенные расширения допустимых операндов, и некоторые дополнительные псевдо-операции, которые описаны здесь. Ассемблер также принимает дополнительные коды операций для процессоров Hitachi 64180 и Z180.

12.2. Использование

Ассемблер называется ZAS и вызывается следующим образом:

```
ZAS параметры файлы ...
```

Файлы - один или несколько исходных файлов на языке ассемблера, которые будут ассемблированы, но обратите внимание, что все файлы ассемблируются как один, а не как отдельные файлы. Чтобы ассемблировать файлы отдельно, необходимо вызывать ассемблер для каждого файла индивидуально. **Параметры** - ноль или более параметров из следующего списка:

-N Игнорировать арифметическое переполнение в выражениях. Параметр `-N` подавляет нормальную проверку арифметического переполнения. Ассемблер следует "Руководству Ассемблера Z80" в его обработке переполнения, и в некоторых случаях это может привести к ошибке, когда в действительности выражение оценивается не так, как предполагает пользователь. Этот параметр может использоваться для переопределения проверки переполнения.

-J Попытаться оптимизировать переходы к ответвлениям. Параметр `-J` запросит ассемблер попытаться ассемблировать переходы и условные переходы в виде относительных ветвей, где это возможно. Будут оптимизированы только условные переходы с эквивалентными ответвлениями, и только переходы к ответвлениям, в которых цель находится в пределах ветви. Обратите внимание, что использование этого параметра замедляет ассемблирование, из-за необходимости ассемблера, выполнить дополнительный проход по входному коду.

-U Рассматривать неопределенные символы как внешние. Параметр `-U` подавит сообщения об ошибках, касающиеся неопределенных символов. Такие символы обрабатываются как внешние в любом случае. Использование этого параметра не изменяет сгенерированный объектный код, а просто служит для подавления сообщений об ошибках.

-Ofile Поместить объектный код в файл. Имя объектного файла по умолчанию формируется из имени первого исходного файла. Любой суффикс или тип файла, т.е. что-либо после самой правой точки (`'.'`) в имени удаляется, и добавляется суффикс `.obj`. Таким образом, команда

```
ZAS file1.as file2.z80
```

произведет объектный файл, названный `file1.obj`. Использование параметра `-O` переопределяет это соглашение по умолчанию, позволяя

произвольно назвать объектный файл. Например:

```
ZAS -ox.obj file1.obj
```

поместит объектный код в `x.obj`.

-Llist Поместить листинг ассемблирования в файл `list`, или на стандартный вывод, если `list` пустой. Файл `list` может быть произведен с помощью параметра `-L`. Если в параметре присутствует имя файла, то файл `list` будет создан с этим именем, иначе листинг будет записан в стандартный вывод (т.е. консоль). В качестве имен файлов `list`, приемлемы имена `CON:` и `LST:`

-Wwidth Отформатировать листинг для принтера заданной ширины. Параметр `-W` определяет ширину, к которой должен быть отформатирован листинг. Например.

```
ZAS -Llst: -W80 x.as
```

выведет листинг, отформатированный для принтера на `80` столбцов на устройство печати.

-C Это параметр запрашивает `ZAS` произвести информацию о перекрестных ссылках в файл. Файл будет называться `xxx.crf`, где `xxx` - начальная часть имени первого исходного файла. Затем необходимо выполнить утилиту `CREF`, чтобы превратить эту информацию в отформатированный листинг.

12.3. Язык ассемблера

Как упоминалось выше, язык ассемблера, принимаемый `ZAS`, основывается на мнемониках Zilog. У вас должен быть некоторый справочник, такой как "Руководство Ассемблера Z80". Ниже описаны те области, где `ZAS` отличается, или имеет расширения, по сравнению со стандартным ассемблером Zilog.

12.3.1. Символы

Символы (метки), поддерживаемые ассемблером могут иметь любую длину, и все символы являются значимыми. Символы, используемые для формирования символа (символического имени) могут быть выбраны из алфавитных букв верхнего и нижнего регистра, цифр `0-9`, а также специальных символов подчеркивания ('_'), доллара ("$") и знака вопроса ('?'). Первый символ не может быть числом. Верхний и нижний регистр различаются. Все приведенные ниже символы являются допустимыми и уникальными.

```
An_identifier
an_identifier
an_identifier1
$$$
?$_123455
```

Обратите внимание, что символ `$` является специальным (представляющим текущее местоположение), и не может использоваться в качестве метки. Не могут использоваться любые мнемоники кодов операций или псевдоопераций, имена регистров или имена кодов условий. Вы должны отметить названия дополнительных кодов условий, описанных ниже.

12.3.1.1. Временные метки

Ассемблер реализует систему временных меток, полезных для использования в локализованном участке кода. Они помогают избежать необходимости генерации имен меток, на которые ссылаются только в непосредственной близости от их определения, например при реализации цикла.

Временная метка принимает форму строки цифр. Ссылка на такую метку требует ту же строку цифр, плюс добавленную букву `b` или `f` для обозначения ссылки назад или ссылки вперед соответственно. Ниже приведен пример использования таких меток.

```
entry_point:    ;На эту ссылаются из далека
    ld   b,10
1:  dec  c
    jr   nz,2f ; если не 0, переход вперед к 2:
    ld   c,8
    djnz 1b    ; Уменьшение на 1 и переходим назад к 1:
    jr   1f    ; это не переход к той же
               ; самой метке djnz
2:  call fred  ; переход сюда от jr nz,2f
1:  ret        ; переход сюда от jr 1f
```

Строка цифр может быть любым положительным десятичным числом от `0` до `65535`. Значение временной метки может снова быть использовано любое число раз. В случае, если сделана ссылка, например, на `1b`, она будет ссылаться на ближайшую метку `1:` найденную ниже текущего положения в файле. Аналогично `23f` сошлется на первую метку `23:` найденную выше текущего положения в файле.

12.3.2. Константы

Константы могут быть введены в одном из оснований 2, 8, 10 или 16. По умолчанию используется основание 10. Константы в других системах счисления могут быть обозначены завершающим символом из следующего набора:

Символ	Основание	Наименование
B	2	двоичное
O	8	восьмеричное
Q	8	восьмеричное
o	8	восьмеричное
q	8	восьмеричное
H	16	шестнадцатеричное
h	16	шестнадцатеричное

Шестнадцатеричные константы могут также быть определены в стиле `C`, например `LD A, 0x21`. Обратите внимание, что `b` в нижнем регистре не может использоваться, для обозначения двоичного числа, так как `1b` - ссылка назад на временную метку `1:`.

12.3.2.1. Символьные константы

Символьная константа - единственный символ, заключенный в одинарные кавычки (`'`). Многосимвольные константы могут использоваться только в качестве операнда в псевдооперации `DEFM`.

12.3.2.2. Константы с плавающей точкой

Константа с плавающей точкой в обычной нотации (например, `1.234` или `1234e-3`) может использоваться в качестве операнда в псевдооперации `DEFF`.

12.3.2.3. Константы кодов операций

В качестве константы в выражении может использоваться любой код операции Z80. Значение кода операции в этом контексте будет байтом, в который ассемблировался бы код операции при использовании обычным способом. Если код операции будет 2-байтовым кодом операции (префиксный байт `CB` или `ED`), то будет использоваться только второй байт кода операции. Это особенно полезно при создании векторов перехода. Например:

```
ld   a,jp          ;Команда перехода
ld   (0),a         ; 0 - переход к "теплой" загрузке
ld   hl,boot       ;сделанной здесь
ld   (1),hl
```

12.3.3. Выражения

Выражения строятся в основном, как описано в "Руководстве Ассемблера Z80".

12.3.3.1. Операторы

В выражениях могут использоваться следующие операторы:

Оператор	Значение	Оператор	Значение
`&`	Поразрядная операция ‘И’	`.or.`	Поразрядное логическое ‘ИЛИ’
`*`	Умножение	`.shl.`	Сдвиг влево
`+`	Сложение	`.shr.`	Сдвиг вправо
`-`	Вычитание	`.ult.`	Без знака меньше, чем
`.and.`	Поразрядная операция ‘И’	`.ugt.`	Без знака больше чем
`.eq.`	Тест равенства	`.xor.`	Исключающее ‘ИЛИ’
`.gt.`	Со знаком больше, чем	`/`	Деление
`.high.`	Старший байт операнда	`<`	Со знаком меньше, чем
`.low.`	Младший байт операнда	`=`	Равно
`.lt.`	Со знаком меньше, чем	`>`	Со знаком больше, чем
`.mod.`	Деление по модулю	`^`	Поразрядное логическое ‘ИЛИ’
`.not.`	Побитовое дополнение		

Операторы, начинающиеся с точки "`.`" должны быть разделены пробелами, таким образом, `label .and. 1` допустимо, а `label.and.1` нет.

12.3.3.2. Перемещаемость

`ZAS` производит перемещаемый объектный код. Это означает, что во время ассемблирования нет необходимости указывать место расположения кода в памяти. Это можно сделать при помощи псевдооперации `ORG`, однако предпочтительный подход должен использовать программные секции или `psect`. `psect` представляет собой именованную секцию программы, в которой код или данные могут быть определены во время ассемблирования. Все части `psect` последовательно загружаются в память, даже если они определены в разных файлах, или в том же файле, но разделены кодом из другой `psect`.

Например, следующий код загружает несколько исполняемых инструкций в psect, названную text и некоторые байты данных в psect data.

```
        psect text, global

alabel:
    ld    hl,astring
    call  putit
    ld    hl,anotherstring

    psect data, global
astring:
    defm  'A string of chars'
    defb  0
anotherstring:
    defm  'Another string'
    defb  0

    psect text

putit:
    ld    a,(hl)
    or    a
    ret   z
    call  outchar
    inc   hl
    jr    putit
```

Обратите внимание, несмотря на то, что два блока кода в psect text разделены блоком psect data, эти два блока psect text будут непрерывны при загрузке редактором связей. Инструкция "ld hl,anotherstring" передаст управление к метке "putit:" во время выполнения. Фактическое расположение в памяти этих двух блоков psect определяется редактором связей. См. *Описание редактора связей* для получения информации о том, как определяются адреса psect.

Метка, определенная в psect, как говорят, перемещаема, т.е. ее фактический адрес в памяти не определен во время ассемблирования. Обратите внимание, что это не так, если метка находится в psect по умолчанию (неименованной), или в psect объявленной абсолютной (см. *Описание псевдооперации PSECT* ниже). Любые метки, объявленные в абсолютной psect, будут абсолютными, то есть их адрес будет определяться ассемблером.

В версии ZAS, поставляемой с HI-TECH C версии 7 или выше, перемещаемые выражения можно свободно комбинировать в выражениях. Более старые версии ZAS допускали лишь ограниченные арифметические операции в перемещаемых выражениях.

12.3.4. Псевдооперации

Псевдооперации основываются на описанные в "Руководстве Ассемблера Z80", с некоторыми дополнениями.

12.3.4.1. DEFB, DB

За этой псевдооперацией должен следовать список выражений разделенных запятой, которые будут ассемблированы в область последовательных байтов. Каждое выражение должно иметь значение в диапазоне от -128 до 255 включительно. DB может использоваться в качестве синонима для DEFB. Пример:

```
DEFB  10, 20, 'a', 0FFH
DB    'hello world',13,10,0
```

12.3.4.2. DEFF

Эта псевдооперация ассемблирует, константы с плавающей точкой в 32 битный формат констант с плавающей точкой HI-TECH C. Например:

```
pi: DEFF  3.14159
```

12.3.4.3. DEFW

Эта работает аналогично DEFB, за исключением того, что она ассемблирует выражения в слова без ограничения значений. Например:

```
DEFW  -1, 3664H, 'A', 3777Q
```

12.3.4.4. DEFS

Псевдооперация DEFS резервирует ячейки памяти, не инициализируя их. Ее операндом является абсолютное выражение, определяющее число байтов, которые будут зарезервированы. Это выражение добавляется к текущему счетчику адреса. Однако, обратите внимание, что редактор связей области зарезервированные DEFS может инициализировать нулями, если они находятся в середине программы. Пример:

```
DEFS  20h    ; Резервирует 32 байта памяти
```

12.3.4.5. EQU

Псевдооперация устанавливает значение символа слева от EQU равным значению выражения справа. Не допускается устанавливать значение символа, который уже определен. Пример:

```
SIZE      equ   46
```

12.3.4.6. DEFL

Псевдооперация идентична EQU за исключением того, что она может переопределить существующие символы. Пример:

```
SIZE      defl  48
```

12.3.4.7. DEFM

Псевдооперация DEFM должна сопровождаться строкой символов заключенных в одинарные кавычки. Значения ASCII этих символов ассемблируются в последовательные ячейки памяти. Пример:

```
DEFM  'A string of funny *@$ characters'
```

12.3.4.8. END

Конец ассемблирования определяется концом исходного файла, или псевдооперацией END. Псевдооперация END при необходимости может сопровождаться выражением, которое будет определять начальный адрес программы. Это фактически бесполезно для CP/M. Только один стартовый адрес может быть определен в программе, и редактор связей будет жаловаться, если их будет больше. Пример:

```
END    somelabel
```

12.3.4.9. COND, IF, ELSE, ENDC

Условный блок представлен псевдооперацией COND. Операнд COND должен быть абсолютным выражением. Если его значение равно false (ноль), то код после COND до соответствующей псевдооперации ENDC не будет ассемблироваться. Пары COND/ENDC могут быть вложены. IF может использоваться в качестве синонима COND. Псевдооперация ELSE может быть включена в блок COND/ENDC, например:

```
IF     CPM
call   5
ELSE
call   os_func
ENDC
```

12.3.4.10. ELSE

См. COND.

12.3.4.11. ENDC

См. COND.

12.3.4.12. ENDM

См. MACRO.

12.3.4.13. PSECT

Эта псевдооперация обеспечивает спецификацию перемещаемых секций программы. Ее параметры - имя psect, за которым может следовать список флагов psect. Имя psect - символ, который строится по тем же правилам что и метки, однако psect может иметь то же имя, что и метка без конфликта. Имена psect распознаются только после псевдооперации PSECT. Флаги psect следующие:

ABS	psect является абсолютной
GLOBAL	psect является глобальной
LOCAL	psect не является глобальной
OVRLD	psect будет накладываться редактором связей
PURE	psect только для чтения

Если psect является глобальной, то компоновщик объединит ее с любыми другими глобальными psect с тем же именем из других модулей. Локальные psect будут обрабатываться как разные из любой другой psect из другого модуля. По умолчанию psect являются глобальными.

Редактор связей по умолчанию объединяет код в psect из различных модулей. Если psect определена как OVRLD, то редактор связей перекроет вклад каждого модуля в этой psect. Это особенно полезно при компоновке модулей, которые инициализируют, например, векторы прерываний.

Флаг PURE сообщает редактору связей, что во время выполнения psect должна быть доступна только для чтения. Полноценность этого флага зависит от воз-

можности редактора связей осуществить это требование. CP/M терпит фиаско в этом отношении.

Флаг ABS делает psect абсолютной. Секция psect будет загружена по адресу 0. Это полезно для статической инициализации векторов прерываний и таблицы переходов. Примеры:

```
PSECT     text, global, pure
PSECT     data, global
PSECT     vectors, ovrld
```

12.3.4.14. GLOBAL

В случае следования за GLOBAL более одного символа (разделенных запятой), они будут рассматриваться ассемблером как внутренние или внешние глобальные символы в зависимости от того, определены они в текущем модуле или нет. Пример:

```
GLOBAL      label1, putchar, _printf
```

12.3.4.15. ORG

Псевдооперация ORG устанавливает для текущей psect по умолчанию (абсолютной) счетчик адреса равным ее операнду, который должен быть абсолютным выражением. Пример:

```
ORG   100H
```

12.3.4.16. MACRO

Эта псевдооперация определяет макрос. Ей должно предшествовать или следовать за ней имя макроса, затем при необходимости список разделенных запятой формальных параметров. Строки кода после псевдооперации MACRO, до следующей псевдо-операции ENDM, будут сохранены в качестве тела макроса. Имя макроса может впоследствии использоваться в части кода операции ассемблерного оператора, сопровождаемого фактическими параметрами. В этом месте будет вставлен текст тела макроса с любыми используемыми формальными параметрами, которые будут заменены на соответствующие фактические параметры. Например:

```
print     MACRO string
    psect data
999:      db    string,'$'
    psect text
    ld    de,999b
    ld    c,9
    call  5
    ENDM
```

При использовании, этот макрос расширится до 3-х инструкций из тела макроса, с фактическими параметрами для функции и аргумента. Таким образом

```
print 'hello world'
```

расширяется до

```
psect data
999:      db    'hello world','$'
    psect text
    ld    de,999b
    ld    c,9
    call  5
```

Параметры макроса могут быть включены в угловые скобки ('<' и '>'), чтобы передать произвольный текст, включая символы-разделители как запятые в качестве единственного параметра. Например, предположим, вы хотите использовать макрос `print`, определенный выше, чтобы распечатать строку, которая включает символы перевода строки и возврат каретки. Макро-вызов:

```
print 'hello world',13,10
```

был бы ошибкой, потому что `13` и `10` рассматриваются как дополнительные параметры и игнорируются. Чтобы передать строку, которая содержит запятые как единый параметр, можно написать:

```
print <'hello world',13,10>
```

который заставит текст `'hello world',13,10` передаваться как единственный параметр. Он расширится до следующего кода:

```
psect data
999:    db    'hello world',13,10,'$'
    psect text
    ld    de,999b
    ld    c,9
    call  5
```

ZAS поддерживает две формы объявления макросов для совместимости с более старыми версиями ZAS и другими ассемблерами Z80. Имя макроса может быть объявлено или в поле метки перед псевдооперацией `MACRO`, или в поле операнда после псевдооперации `MACRO`. Таким образом, эти два объявления `MACRO` эквивалентны:

```
bdos    MACRO func,arg
    ld    de,arg
    ld    c,func
    call  5
    ENDM
```

и

```
    MACRO bdos,func,arg
    ld    de,arg
    ld    c,func
    call  5
    ENDM
```

12.3.4.17. LOCAL

Псевдооперация LOCAL позволяет для каждого расширения макроса определять уникальные метки. У любых символов, перечисленных после директивы LOCAL, будет уникальный сгенерированный ассемблером символ заменяющий их при расширении макроса. Например:

```
copy      MACRO source,dest,count
    LOCAL nocopy
    push  af
    push  bc
    ld    bc,source
    ld    a,b
    or    c
    jr    z,nocopy
    push  de
    push  hl
    ld    de,dest
    ld    hl,source
    ldir
    pop   hl
    pop   de
nocopy:   pop   bc
    pop   af
    ENDM
```

при расширении будет включать уникальную метку, сгенерированную ассемблером вместо nocopy. Например,

```
copy (recptr),buf,(recsize)
```

расширяется до:

```
    push  af
    push  bc
    ld    bc,(recsize)
    ld    a,b
    or    c
    jr    z,??0001
    push  de
    push  hl
    ld    de,buf
    ld    hl,(recptr)
    ldir
    pop   hl
    pop   de
??0001:   pop   bc
    pop   af
```

при втором вызове, метка nocopy расширится до ??0002.

12.3.4.18. REPT

Псевдооперация REPT определяет временный макрос, определяет временный макрос, который затем расширяется столько раз, сколько определено его параметром. Например:

```
REPT  3
ld    (hl),0
inc   hl
ENDM
```

расширяется до:

```
ld    (hl),0
inc   hl
ld    (hl),0
inc   hl
ld    (hl),0
inc   hl
```

12.3.5. IRP и lRPC

Директивы IRP и IRPC подобны REPT, однако вместо того, чтобы повторить блок постоянное число раз, он повторяется один раз для каждого элемента из списка параметров. В случае IRP список представляет собой обычный список параметров макроса, в случае IRPC, это - последовательные символы из строки. Например:

```
IRP   string,<'hello world',13,10>,'arg2'
    LOCAL str
    psect data
str:      db    string,'$'
    psect text
    ld    c,9
    ld    de,str
    call  5
    ENDM
```

расширяется до:

```
    psect data
??0001:   db    'hello world',13,10,'$'
    psect text
    ld    c,9
    ld    de,??0001
    call  5
    psect data
??0002:   db    'arg2','$'
    psect text
    ld    c,9
    ld    de,??0002
    call  5
```

Обратите внимание на использование меток LOCAL и угловых скобок таким же образом, как и с обычными макросами.

Использование IRPC лучше всего продемонстрировать на следующем примере:

```
IRPC  char,ABC
ld    c,2
ld    e,'char'
call  5
ENDM
```

расширяется до:

```
ld    c,2
ld    e,'A'
call  5
ld    c,2
ld    e,'B'
call  5
ld    c,2
ld    e,'C'
call  5
```

12.3.6. Расширенные коды условий

Ассемблер распознает несколько кодов дополнительных условий. Это:

Код	Эквивалент	Значение
alt	m	Арифметически меньше, чем
llt	c	Логически меньше, чем
age	p	Арифметически больше или равно
lge	nc	Логически больше или равно
di		Используется после `ld a,i` для тестирования состояния флага разрешения прерывания (включен или отключен соответственно).
ei		

12.4. Директивы ассемблера

Директива ассемблера представляет собой строку в исходном файле, которая не производит кода, а изменяет поведение ассемблера. Каждая директива распознается по присутствию звездочки в первом столбце строки, непосредственно за которой следует слово, в котором только первый символ имеет значение. Строка, содержащая саму директиву, никогда не появляется в листинге. Директивами являются:

***Title**

Использует текст после директивы как заголовок для листинга.

***Heading**

Использует текст после директивы в качестве подзаголовка для включения в листинг. Также вызывает `*Eject`.

***List**

Может содержать `ON` или `OFF` для включения или отключения листинга соответственно. Обратите внимание, что эта директива может использоваться в макросе или включаемом файле, для управления распечаткой этого макроса или включаемого файла. Предыдущее состояние распечатки будет восстановлено на выходе от макроса или включаемого файла.

***Include**

Файл, названный после директивы, будет включен в ассемблирование в этой точке.

***Eject**

Новая страница будет запущена в листинге в этой точке. Символ перевода формата в источнике будет иметь тот же эффект.

Некоторые примеры использования этих директив:

```
*Title Widget Control Program
*Heading Initialization Phase
*Include widget.i
```

12.5. Сообщения об ошибках

Для каждой обнаруженной ошибки при ассемблировании сообщение об ошибке записывается в стандартный поток ошибок. Это сообщение идентифицирует имя файла, номер строки и описывает ошибку. Кроме того, строка в листинге, в которой произошла ошибка, помечается с помощью одного символа обозначающего ошибку. Символы и соответствующие сообщения:

A:	Absolute expression required	(Требуется абсолютное выражение)
B:	Bad arg to *L	(Некорректный аргумент для `*L`)
	Bad arg to IM	(Некорректный аргумент для `IM`)
	Bad bit number	(Некорректный номер бита)
	Bad character constant	(Некорректная символьная константа)
	Bad jump condition	(Некорректное условие перехода)
D:	Directive not recognized	(Не распознанная директива)
	Digit out of range	(Цифра вне диапазона)
E:	EOF inside conditional	(`EOF` в условном выражении)
	Expression error	(Ошибка в выражении)
G:	Garbage after operands	(Неверная информация после операндов)
	Garbage on end of line	(Неверная информация в конце строки)
I:	Index offset too large	(Слишком большое смещение индекса)
J:	Jump target out of range	(Точка перехода вне досягаемости)
L:	Lexical error	(Лексическая ошибка)
M:	Multiply defined symbol	(Многократно определенный символ)
O:	Operand error	(Ошибка операнда)
P:	Phase error	(Ошибка фазы)
	Psect may not be local and global	(`psect` не может быть локальной и глобальной)
R:	Relocation error	(Ошибка перемещения)
S:	Size error	(Ошибка размера)
	Syntax error	(Синтаксическая ошибка)
U:	Undefined symbol	(Неопределенный символ)
	Undefined temporary label	(Неопределенная временная метка)
	Unterminated string	(Незавершенная строка)

12.6. Система команд Z80/Z180/64180

Остаток этой главы посвящен полному перечислению системы команд для Z80, Z180, 64180 и процессоров NSC800. Z180 и 64180 выполнит все инструкции Z80, несмотря на то, что временные характеристики отличаются.

Код операции	Мнемоника	Операнды	Действие
8E	ADC	A,(HL)	Сложение с переносом A и памяти в (HL)
DD 8E ii	ADC	A,(IX+ii)	Сложение с переносом A, и памяти в (IX+ii)
FD 8E ii	ADC	A,(IY+ii)	Сложение с переносом A, и памяти в (IY+ii)
8F	ADC	A,A	Сложение A с регистром с учетом переноса
88	ADC	A,B	
89	ADC	A,C	
8A	ADC	A,D	
8B	ADC	A,E	
8C	ADC	A,H	
8D	ADC	A,L	
CE nn	ADC	A,nn	Сложение с переносом A с nn
ED 4A	ADC	HL,BC	Сложение HL с регистром с учетом перено-
ED 5A	ADC	HL,DE	са
ED 6A	ADC	HL,HL	
ED 7A	ADC	HL,SP	
86	ADD	A,(HL)	Сложение с A памяти в (HL)
DD 86 ii	ADD	A,(IX+ii)	Сложение с A памяти в (IX+ii)
FD 86 ii	ADD	A,(IY+ii)	Сложение с A памяти в (IY+ii)
87	ADD	A,A	Сложение A с регистром
80	ADD	A,B	
81	ADD	A,C	
82	ADD	A,D	
83	ADD	A,E	
84	ADD	A,H	
85	ADD	A,L	
C6 nn	ADD	A,nn	Сложение A непосредственно с nn
09	ADD	HL,BC	Сложение HL с регистром
19	ADD	HL,DE	
29	ADD	HL,HL	
39	ADD	HL,SP	
DD 09	ADD	IX,BC	Сложение IX с регистром
DD 19	ADD	IX,DE	
DD 29	ADD	IX,IX	
DD 39	ADD	IX,SP	
FD 09	ADD	IY,BC	Сложение IY с регистром
FD 19	ADD	IY,DE	
FD 29	ADD	IY,IY	
FD 39	ADD	IY,SP	
A6	AND	(HL)	Логическое ‘И' A с памятью в (HL)
DD A6 ii	AND	(IX+ii)	Логическое ‘И' A с памятью в (IX+ii)
FD A6 ii	AND	(IY+ii)	Логическое ‘И' A с памятью в (IY+ii)

Код операции	Мнемоника	Операнды	Действие
A7	AND	A	Логическое 'И' A с регистром
A0	AND	B	
A1	AND	C	
A2	AND	D	
A3	AND	E	
A4	AND	H	
A5	AND	L	
E6 nn	AND	nn	Логическое 'И' A непосредственно с nn
CB 46	BIT	0,(HL)	Проверка бита 0 памяти в (HL)
DD CB ii 46	BIT	0,(IX+ii)	Проверка бита 0 памяти в (IX+ii)
FD CB ii 46	BIT	0,(IY+ii)	Проверка бита 0 памяти в (IY+ii)
CB 47	BIT	0,A	Проверка бита 0 регистра
CB 40	BIT	0,B	
CB 41	BIT	0,C	
CB 42	BIT	0,D	
CB 43	BIT	0,E	
CB 44	BIT	0,H	
CB 45	BIT	0,L	
CB 4E	BIT	1,(HL)	Проверка бита 1 памяти в (HL)
DD CB ii 4E	BIT	1,(IX+ii)	Проверка бита 1 памяти в (IX+ii)
FD CB ii 4E	BIT	1,(IY+ii)	Проверка бита 1 памяти в (IY+ii)
CB 4F	BIT	1,A	Проверка бита 1 регистра
CB 48	BIT	1,B	
CB 49	BIT	1,C	
CB 4A	BIT	1,D	
CB 4B	BIT	1,E	
CB 4C	BIT	1,H	
CB 4D	BIT	1,L	
CB 56	BIT	2,(HL)	Проверка бита 2 памяти в (HL)
DD CB ii 56	BIT	2,(IX+ii)	Проверка бита 2 памяти в (IX+ii)
FD CB ii 56	BIT	2,(IY+ii)	Проверка бита 2 памяти в (IY+ii)
CB 57	BIT	2,A	Проверка бита 2 регистра
CB 50	BIT	2,B	
CB 51	BIT	2,C	
CB 52	BIT	2,D	
CB 53	BIT	2,E	
CB 54	BIT	2,H	
CB 55	BIT	2,L	
CB 5E	BIT	3,(HL)	Проверка бита 3 памяти в (HL)
DD CB ii 5E	BIT	3,(IX+ii)	Проверка бита 3 памяти в (IX+ii)
FD CB ii 5E	BIT	3,(IY+ii)	Проверка бита 3 памяти в (IY+ii)
CB 5F	BIT	3,A	Проверка бита 3 регистра
CB 58	BIT	3,B	
CB 59	BIT	3,C	
CB 5A	BIT	3,D	
CB 5B	BIT	3,E	
CB 5C	BIT	3,H	
CB 5D	BIT	3,L	

Код операции	Мнемоника	Операнды	Действие
CB 66	BIT	4,(HL)	Проверка бита 4 памяти в (HL)
DD CB ii 66	BIT	4,(IX+ii)	Проверка бита 4 памяти в (IX+ii)
FD CB ii 66	BIT	4,(IY+ii)	Проверка бита 4 памяти в (IY+ii)
CB 67	BIT	4,A	Проверка бита 4 регистра
CB 60	BIT	4,B	
CB 61	BIT	4,C	
CB 62	BIT	4,D	
CB 63	BIT	4,E	
CB 64	BIT	4,H	
CB 65	BIT	4,L	
CB 6E	BIT	5,(HL)	Проверка бита 5 памяти в (HL)
DD CB ii 6E	BIT	5,(IX+ii)	Проверка бита 5 памяти в (IX+ii)
FD CB ii 6E	BIT	5,(IY+ii)	Проверка бита 5 памяти в (IY+ii)
CB 6F	BIT	5,A	Проверка бита 5 регистра
CB 68	BIT	5,B	
CB 69	BIT	5,C	
CB 6A	BIT	5,D	
CB 6B	BIT	5,E	
CB 6C	BIT	5,H	
CB 6D	BIT	5,L	
CB 76	BIT	6,(HL)	Проверка бита 6 памяти в (HL)
DD CB ii 76	BIT	6,(IX+ii)	Проверка бита 6 памяти в (IX+ii)
FD CB ii 76	BIT	6,(IY+ii)	Проверка бита 6 памяти в (IY+ii)
CB 77	BIT	6,A	Проверка бита 6 регистра
CB 70	BIT	6,B	
CB 71	BIT	6,C	
CB 72	BIT	6,D	
CB 73	BIT	6,E	
CB 74	BIT	6,H	
CB 75	BIT	6,L	
CB 7E	BIT	7,(HL)	Проверка бита 7 памяти в (HL)
DD CB ii 7E	BIT	7,(IX+ii)	Проверка бита 7 памяти в (IX+ii)
FD CB ii 7E	BIT	7,(IY+ii)	Проверка бита 7 памяти в (IY+ii)
CB 7F	BIT	7,A	Проверка бита 7 регистра
CB 78	BIT	7,B	
CB 79	BIT	7,C	
CB 7A	BIT	7,D	
CB 7B	BIT	7,E	
CB 7C	BIT	7,H	
CB 7D	BIT	7,L	
CD ll hh	CALL	hhll	Вызов подпрограммы
DC ll hh	CALL	C,hhll	Вызов, если carry установлен
FC ll hh	CALL	M,hhll	Вызов, если минус
D4 ll hh	CALL	NC,hhll	Вызов, если carry сброшен
C4 ll hh	CALL	NZ,hhll	Вызов, если не ноль
F4 ll hh	CALL	P,hhll	Вызов, если плюс
EC ll hh	CALL	PE,hhll	Вызов, если четный
E4 ll hh	CALL	PO,hhll	Вызов, если нечетный
CC ll hh	CALL	Z,hhll	Вызов, если ноль

Код операции	Мнемоника	Операнды	Действие
3F	CCF		Инверсия флага переноса
BE	CP	(HL)	Сравнение с A памяти в (HL)
DD BE ii	CP	(IX+ii)	Сравнение с A памяти в (IX+ii)
FD BE ii	CP	(IY+ii)	Сравнение с A памяти в (IY+ii)
BF	CP	A	Сравнение A с регистром
B8	CP	B	
B9	CP	C	
BA	CP	D	
BB	CP	E	
BC	CP	H	
BD	CP	L	
FE nn	CP	nn	Сравнение с A непосредственно nn
ED A9	CPD		Сравнение с декрементом Сравнить регистр A с памятью (HL), декремент HL и BC. Флаг Z отражает сравнение, флаг P/V очищен, если BC 0.
ED B9	CPDR		Блочное сравнение с декрементом Сравнить регистр A с памятью (HL), декремент HL и BC, если BC не ноль, и A не равен (HL) то повторить. После завершения флаг Z установлен, если соответствие было найдено, флаг P/V очищен, если BC ноль.
ED A1	CPI		Сравнение с инкрементом Сравнить регистр A с памятью (HL), инкремент HL, декремент BC. Флаг Z отражает сравнение, флаг P/V очищен, если BC 0.
ED B1	CPIR		Блочное сравнение с инкрементом Сравнить регистр A с памятью (HL), инкремент HL, декремент BC, если BC не ноль, и A не равен (HL) то повторить. После завершения флаг Z установлен, если соответствие было найдено, флаг P/V очищен, если BC ноль.
2F	CPL		Инверсия регистра A
27	DAA		Десятичная коррекция регистра A
35	DEC	(HL)	Декремент памяти в (HL)
DD 35 ii	DEC	(IX+ii)	Декремент памяти в (IX+ii)
FD 35 ii	DEC	(IY+ii)	Декремент памяти в (IY+ii)
3D	DEC	A	Декремент 8 битного регистра
05	DEC	B	
0D	DEC	C	
15	DEC	D	
1D	DEC	E	
25	DEC	H	
2D	DEC	L	
0B	DEC	BC	Декремент 16 битного регистра
1B	DEC	DE	
2B	DEC	HL	
DD 2B	DEC	IX	
FD 2B	DEC	IY	
3B	DEC	SP	

Код операции	Мнемоника	Операнды	Действие
F3	DI		Запретить прерывания
10 rr	DJNZ	rr	Декремент B и переход если не 0
FB	EI		Разрешить прерывания
E3	EX	(SP),HL	Обмен HL с вершиной стека
DD E3	EX	(SP),IX	Обмен IX с вершиной стека
FD E3	EX	(SP),IY	Обмен IY с вершиной стека
08	EX	AF,AF'	Обмен AF с AF'
EB	EX	DE,HL	Обмен DE с HL
D9	EXX		Обмен BC, DE, HL с BC', DE', HL'
76	HALT		Приостановить выполнение
ED 46	IM	0	Установить режим прерывания 0
ED 56	IM	1	Установить режим прерывания 1
ED 5E	IM	2	Установить режим прерывания 2
DB pp	IN	A,(pp)	Загрузить A из порта (pp) Вывод pp на младшую половину адресной шины и регистра A на старшую половину адресной шины.
ED 78	IN	A,(C)	Загрузить регистр из порта (BC) Вывод регистра C на младшую половину адресной шины и регистра B на старшую половину адресной шины. Используйте это на процессорах с декодированием 16 битного адреса ввода-вывода.
ED 40	IN	B,(C)	
ED 48	IN	C,(C)	
ED 50	IN	D,(C)	
ED 58	IN	E,(C)	
ED 60	IN	H,(C)	
ED 68	IN	L,(C)	
ED 38 pp	IN0	A,(pp)	Z180: Загрузить регистр из порта (00pp). Вывод pp на младшую половину адресной шины и 00H на старшую половину адресной шины. Используйте эту инструкцию вместо IN A,(pp) с процессорами Z180 и 64180.
ED 00 pp	IN0	B,(pp)	
ED 08 pp	IN0	C,(pp)	
ED 10 pp	IN0	D,(pp)	
ED 18 pp	IN0	E,(pp)	
ED 20 pp	IN0	H,(pp)	
ED 28 pp	IN0	L,(pp)	
34	INC	(HL)	Инкремент памяти в (HL)
DD 34 ii	INC	(IX+ii)	Инкремент памяти в (IX+ii)
FD 34 ii	INC	(IY+ii)	Инкремент памяти в (IY+ii)
3C	INC	A	Инкремент 8 битного регистра
04	INC	B	
0C	INC	C	
14	INC	D	
1C	INC	E	
24	INC	H	
2C	INC	L	
03	INC	BC	Инкремент 16 битного регистра
13	INC	DE	
23	INC	HL	
DD 23	INC	IX	
FD 23	INC	IY	
33	INC	SP	
ED AA	IND		Ввод с декрементом. Ввод порта (BC) и запись результата по адресу (HL). Декремент регистров B и HL. Установить флаг Z, если B - ноль.

Код операции	Мнемоника	Операнды	Действие
ED BA	INDR		Блочный ввод с декрементом. Ввод порта (BC) и запись результата по адресу (HL). Декремент регистров B и HL, повторить если B не ноль.
ED A2	INI		Ввод с инкрементом. Ввод порта (BC) и запись результата по адресу (HL). Декремент B, инкремент HL. Установить флаг Z, если B - ноль.
ED B2	INIR		Блочный ввод с инкрементом. Ввод порта (BC) и запись результата по адресу (HL). Декремент B, инкремент HL, повторить если B не ноль.
C3 ll hh	JP	hhll	Переход
E9	JP	(HL)	Переход по адресу в HL
DD E9	JP	(IX)	Переход по адресу в IX
FD E9	JP	(IY)	Переход по адресу в IY
DA ll hh	JP	C,hhll	Переход, если carry установлен
FA ll hh	JP	M,hhll	Переход, если минус
D2 ll hh	JP	NC,hhll	Переход, если carry сброшен
C2 ll hh	JP	NZ,hhll	Переход, если не ноль
F2 ll hh	JP	P,hhll	Переход, если плюс
EA ll hh	JP	PE,hhll	Переход, если четный
E2 ll hh	JP	PO,hhll	Переход, если нечетный
CA ll hh	JP	Z,hhll	Переход, если ноль
18 rr	JR	rr	Относительный переход
38 rr	JR	C,rr	Относительный переход, если C установлен
30 rr	JR	NC,rr	Относительный переход, если C очищен
20 rr	JR	NZ,rr	Относительный переход, если не ноль
28 rr	JR	Z,rr	Относительный переход, если ноль
02	LD	(BC),A	Сохранить A по адресу из (BC)
12	LD	(DE),A	Сохранить A по адресу из (DE)
77	LD	(HL),A	Сохранить регистр по адресу из (HL)
70	LD	(HL),B	
71	LD	(HL),C	
72	LD	(HL),D	
73	LD	(HL),E	
74	LD	(HL),H	
75	LD	(HL),L	
36 nn	LD	(HL),nn	Сохранить nn по адресу из (HL)
DD 77 ii	LD	(IX+ii),A	Сохранить регистр по адресу из (IX+ii)
DD 70 ii	LD	(IX+ii),B	
DD 71 ii	LD	(IX+ii),C	
DD 72 ii	LD	(IX+ii),D	
DD 73 ii	LD	(IX+ii),E	
DD 74 ii	LD	(IX+ii),H	
DD 75 ii	LD	(IX+ii),L	

Код операции	Мнемоника	Операнды	Действие
DD 36 ii nn	LD	(IX+ii),nn	Сохранить nn по адресу из (IX+ii)
FD 77 ii	LD	(IY+ii),A	Сохранить регистр по адресу из (IY+ii)
FD 70 ii	LD	(IY+ii),B	
FD 71 ii	LD	(IY+ii),C	
FD 72 ii	LD	(IY+ii),D	
FD 73 ii	LD	(IY+ii),E	
FD 74 ii	LD	(IY+ii),H	
FD 75 ii	LD	(IY+ii),L	
FD 36 ii nn	LD	(IY+ii),nn	Сохранить nn по адресу из (IX+ii)
32 ll hh	LD	(hhll),A	Сохранить A в память
ED 43 ll hh	LD	(hhll),BC	Сохранить BC в память
ED 53 ll hh	LD	(hhll),DE	Сохранить DE в память
22 ll hh	LD	(hhll),HL	Сохранить HL в память
DD 22 ll hh	LD	(hhll),IX	Сохранить IX в память
FD 22 ll hh	LD	(hhll),IY	Сохранить IY в память
ED 73 ll hh	LD	(hhll),SP	Сохранить SP в память
3A ll hh	LD	A,(hhll)	Загрузка A из памяти
0A	LD	A,(BC)	Загрузка A из адреса в (BC)
1A	LD	A,(DE)	Загрузка A из адреса в (DE)
7E	LD	A,(HL)	Загрузка A из адреса в (HL)
DD 7E ii	LD	A,(IX+ii)	Загрузка A из адреса в (IX+ii)
FD 7E ii	LD	A,(IY+ii)	Загрузка A из адреса в (IY+ii)
7F	LD	A,A	Загрузка A из регистра
78	LD	A,B	
79	LD	A,C	
7A	LD	A,D	
7B	LD	A,E	
7C	LD	A,H	
7D	LD	A,L	
ED 57	LD	A,I	Загрузка A из регистра I
ED 5F	LD	A,R	Загрузка A из регистра R
3E nn	LD	A,nn	Загрузка в A непосредственно nn
46	LD	B,(HL)	Загрузка B из адреса в (HL)
DD 46 ii	LD	B,(IX+ii)	Загрузка B из адреса в (IX+ii)
FD 46 ii	LD	B,(IY+ii)	Загрузка B из адреса в (IY+ii)
47	LD	B,A	Загрузка B из регистра
40	LD	B,B	
41	LD	B,C	
42	LD	B,D	
43	LD	B,E	
44	LD	B,H	
45	LD	B,L	
06 nn	LD	B,nn	Загрузить в B непосредственно nn
ED 4B ll hh	LD	BC,(hhll)	Загрузка BC из памяти
01 ll hh	LD	BC,hhll	Загрузка в BC непосредственно hhll
4E	LD	C,(HL)	Загрузка C из адреса в (HL)
DD 4E ii	LD	C,(IX+ii)	Загрузка C из адреса в (IX+ii)
FD 4E ii	LD	C,(IY+ii)	Загрузка C из адреса в (IY+ii)

Код операции	Мнемоника	Операнды	Действие
4F	LD	C,A	Загрузка C из регистра
48	LD	C,B	
49	LD	C,C	
4A	LD	C,D	
4B	LD	C,E	
4C	LD	C,H	
4D	LD	C,L	
0E nn	LD	C,nn	Загрузка в C непосредственно nn
56	LD	D,(HL)	Загрузка D из адреса в (HL)
DD 56 ii	LD	D,(IX+ii)	Загрузка D из адреса в (IX+ii)
FD 56 ii	LD	D,(IY+ii)	Загрузка D из адреса в (IY+ii)
57	LD	D,A	Загрузка D из регистра
50	LD	D,B	
51	LD	D,C	
52	LD	D,D	
53	LD	D,E	
54	LD	D,H	
55	LD	D,L	
16 nn	LD	D,nn	Загрузка в D непосредственно nn
ED 5B ll hh	LD	DE,(hhll)	Загрузка DE из памяти
11 ll hh	LD	DE,hhll	Загрузка в DE непосредственно hhll
5E	LD	E,(HL)	Загрузка E из адреса в (HL)
DD 5E ii	LD	E,(IX+ii)	Загрузка E из адреса в (IX+ii)
FD 5E ii	LD	E,(IY+ii)	Загрузка E из адреса в (IY+ii)
5F	LD	E,A	Загрузка E из регистра
58	LD	E,B	
59	LD	E,C	
5A	LD	E,D	
5B	LD	E,E	
5C	LD	E,H	
5D	LD	E,L	
1E nn	LD	E,nn	Загрузка в E непосредственно nn
66	LD	H,(HL)	Загрузка H из адреса в (HL)
DD 66 ii	LD	H,(IX+ii)	Загрузка H из адреса в (IX+ii)
FD 66 ii	LD	H,(IY+ii)	Загрузка H из адреса в (IY+ii)
67	LD	H,A	Загрузка H из регистра
60	LD	H,B	
61	LD	H,C	
62	LD	H,D	
63	LD	H,E	
64	LD	H,H	
65	LD	H,L	
26 nn	LD	H,nn	Загрузка в H непосредственно nn
2A ll hh	LD	HL,(hhll)	Загрузка HL из памяти
21 ll hh	LD	HL,hhll	Загрузка в HL непосредственно hhll
ED 47	LD	I,A	Загрузка регистра I из A
DD 2A ll hh	LD	IX,(hhll)	Загрузка IX из памяти
DD 21 ll hh	LD	IX,hhll	Загрузка в IX непосредственно hhll
FD 2A ll hh	LD	IY,(hhll)	Загрузка IY из памяти
FD 21 ll hh	LD	IY,hhll	Загрузка в IY непосредственно hhll

Код операции	Мнемоника	Операнды	Действие
6E	LD	L,(HL)	Загрузка L из адреса в (HL)
DD 6E ii	LD	L,(IX+ii)	Загрузка L из адреса в (IX+ii)
FD 6E ii	LD	L,(IY+ii)	Загрузка L из адреса в (IY+ii)
6F	LD	L,A	Загрузка L из регистра
68	LD	L,B	
69	LD	L,C	
6A	LD	L,D	
6B	LD	L,E	
6C	LD	L,H	
6D	LD	L,L	
2E nn	LD	L,nn	Загрузка в L непосредственно nn
ED 4F	LD	R,A	Загрузка регистра R из A
ED 7B ll hh	LD	SP,(hhll)	Загрузка SP из памяти, косвенная
F9	LD	SP,HL	Загрузка SP из HL
DD F9	LD	SP,IX	Загрузка SP из IX
FD F9	LD	SP,IY	Загрузка SP из IY
31 ll hh	LD	SP,hhll	Загрузка SP из памяти, непосредственная
ED A8	LDD		Загрузка с декрементом. Копирование памяти из (HL) в (DE), декремент HL, DE и BC, очистка флага P/V если BC - 0
ED B8	LDDR		Загрузка блока с декрементом. Копирование памяти из (HL) в (DE), декремент HL, DE и BC, повторить, если BC не 0
ED A0	LDI		Загрузка с инкрементом. Копирование памяти из (HL) в (DE), инкремент HL и DE, декремент BC, очистка флага P/V если BC - 0
ED B0	LDIR		Загрузка блока с инкрементом. Копирование памяти из (HL) в (DE), инкремент HL и DE, декремент BC, повторить, если BC не 0
ED 4C	MLT	BC	Z180: умножение BC = B x C
ED 5C	MLT	DE	Z180: умножение DE = D x E
ED 6C	MLT	HL	Z180: умножение HL = H x L
ED 7C	MLT	SP	Z180: умножение SP = SPH x SPL
ED 44	NEG		Дополнение до двух регистра A
00	NOP		Нет операции
B6	OR	(HL)	Логическое 'ИЛИ' A с памятью в (HL)
DD B6 ii	OR	(IX+ii)	Логическое 'ИЛИ' A с памятью в (IX+ii)
FD B6 ii	OR	(IY+ii)	Логическое 'ИЛИ' A с памятью в (IY+ii)
B7	OR	A	Логическое 'ИЛИ' A с регистром
B0	OR	B	
B1	OR	C	
B2	OR	D	
B3	OR	E	
B4	OR	H	
B5	OR	L	
F6 nn	OR	nn	Логическое 'ИЛИ' A непосредственно с nn

Код операции	Мнемоника	Операнды	Действие
ED BB	OTDR		Вывод блока с декрементом. Вывод памяти (HL) в порт (BC), декремент HL и B, повторить если B не 0
ED 9B	OTDMR		Z180: Вывод блока с декрементом. Как и OTDR, но выводит 00H на старшую половину адресной шины. Используйте вместо OTDR с процессорами Z180 и 64180.
ED B3	OTIR		Вывод блока с инкрементом. Вывод памяти (HL) в порт (BC), инкремент HL, декремент B, повторить если B не 0
ED 93	OTIMR		Z180: Вывод блока с инкрементом. Как и OTIR, но выводит 00H на старшую половину адресной шины. Используйте вместо OTIR с процессорами Z180 и 64180.
D3 pp	OUT	(pp),A	Вывод A в порт (pp). Выходные pp на низкую половину адресной шины, регистр A на старшую половину адресной шины и на шину данных.
ED 79 ED 41 ED 49 ED 51 ED 59 ED 61 ED 69	OUT OUT OUT OUT OUT OUT OUT	(C),A (C),B (C),C (C),D (C),E (C),H (C),L	Вывод регистра в порт (BC) Вывод регистра C на младшую половину адресной шины, регистра B на старшую половину адресной шины. Используйте эту инструкцию вместо OUT (pp),A с процессорами с 16 разрядным декодированием ввода-вывода.
ED 39 pp ED 01 pp ED 09 pp ED 11 pp ED 19 pp ED 21 pp ED 29 pp	OUT0 OUT0 OUT0 OUT0 OUT0 OUT0 OUT0	(pp),A (pp),B (pp),C (pp),D (pp),E (pp),H (pp),L	Z180: Вывод регистра в порт (00pp), вывод 00H в старший байт адресной шины. Используйте это вместо OUT (pp),A с процессорами Z180 и 64180.
ED AB	OUTD		Вывод с декрементом. Вывод памяти (HL) в порт (BC), декремент HL и B, установить флаг Z если B - 0.
ED 8B	OTDM		Z180: Вывод с декрементом. Как OUTD, но выводит 00H на старшую половину адресной шины, используйте вместо OUTD с процессорами Z180 и 64180.
ED A3	OUTI		Вывод с инкрементом. Вывод памяти (HL) в порт (BC), инкремент HL, декремент B, установить флаг Z если B - 0.
ED 83	OTIM		Z180: Вывод с инкрементом. Как OUTI, но выводит 00H на старшую половину адресной шины, используйте вместо OUTI с процессорами Z180 и 64180.

Код операции	Мнемоника	Операнды	Действие
F1	POP	AF	Извлечение из стека AF
C1	POP	BC	Извлечение из стека BC
D1	POP	DE	Извлечение из стека DE
E1	POP	HL	Извлечение из стека HL
DD E1	POP	IX	Извлечение из стека IX
FD E1	POP	IY	Извлечение из стека IY
F5	PUSH	AF	Занесение в стек AF
C5	PUSH	BC	Занесение в стек BC
D5	PUSH	DE	Занесение в стек DE
E5	PUSH	HL	Занесение в стек HL
DD E5	PUSH	IX	Занесение в стек IX
FD E5	PUSH	IY	Занесение в стек IY
CB 86	RES	0,(HL)	Очистка бита 0 в памяти (HL)
DD CB ii 86	RES	0,(IX+ii)	Очистка бита 0 в памяти (IX+ii)
FD CB ii 86	RES	0,(IY+ii)	Очистка бита 0 в памяти (IY+ii)
CB 87	RES	0,A	Очистка бита 0 в регистре
CB 80	RES	0,B	
CB 81	RES	0,C	
CB 82	RES	0,D	
CB 83	RES	0,E	
CB 84	RES	0,H	
CB 85	RES	0,L	
CB 8E	RES	1,(HL)	Очистка бита 1 в памяти (HL)
DD CB ii 8E	RES	1,(IX+ii)	Очистка бита 1 в памяти (IX+ii)
FD CB ii 8E	RES	1,(IY+ii)	Очистка бита 1 в памяти (IY+ii)
CB 8F	RES	1,A	Очистка бита 1 в регистре
CB 88	RES	1,B	
CB 89	RES	1,C	
CB 8A	RES	1,D	
CB 8B	RES	1,E	
CB 8C	RES	1,H	
CB 8D	RES	1,L	
CB 96	RES	2,(HL)	Очистка бита 2 в памяти (HL)
DD CB ii 96	RES	2,(IX+ii)	Очистка бита 2 в памяти (IX+ii)
FD CB ii 96	RES	2,(IY+ii)	Очистка бита 2 в памяти (IY+ii)
CB 97	RES	2,A	Очистка бита 2 в регистре
CB 90	RES	2,B	
CB 91	RES	2,C	
CB 92	RES	2,D	
CB 93	RES	2,E	
CB 94	RES	2,H	
CB 95	RES	2,L	
CB 9E	RES	3,(HL)	Очистка бита 3 в памяти (HL)
DD CB ii 9E	RES	3,(IX+ii)	Очистка бита 3 в памяти (IX+ii)
FD CB ii 9E	RES	3,(IY+ii)	Очистка бита 3 в памяти (IY+ii)

Код операции	Мнемоника	Операнды	Действие
CB 9F	RES	3,A	Очистка бита 3 в регистре
CB 98	RES	3,B	
CB 99	RES	3,C	
CB 9A	RES	3,D	
CB 9B	RES	3,E	
CB 9C	RES	3,H	
CB 9D	RES	3,L	
CB A6	RES	4,(HL)	Очистка бита 4 в памяти (HL)
DD CB ii A6	RES	4,(IX+ii)	Очистка бита 4 в памяти (IX+ii)
FD CB ii A6	RES	4,(IY+ii)	Очистка бита 4 в памяти (IY+ii)
CB A7	RES	4,A	Очистка бита 4 в регистре
CB A0	RES	4,B	
CB A1	RES	4,C	
CB A2	RES	4,D	
CB A3	RES	4,E	
CB A4	RES	4,H	
CB A5	RES	4,L	
CB AE	RES	5,(HL)	Очистка бита 5 в памяти (HL)
DD CB ii AE	RES	5,(IX+ii)	Очистка бита 5 в памяти (IX+ii)
FD CB ii AE	RES	5,(IY+ii)	Очистка бита 5 в памяти (IY+ii)
CB AF	RES	5,A	Очистка бита 5 в регистре
CB A8	RES	5,B	
CB A9	RES	5,C	
CB AA	RES	5,D	
CB AB	RES	5,E	
CB AC	RES	5,H	
CB AD	RES	5,L	
CB B6	RES	6,(HL)	Очистка бита 6 в памяти (HL)
DD CB ii B6	RES	6,(IX+ii)	Очистка бита 6 в памяти (IX+ii)
FD CB ii B6	RES	6,(IY+ii)	Очистка бита 6 в памяти (IY+ii)
CB B7	RES	6,A	Очистка бита 6 в регистре
CB B0	RES	6,B	
CB B1	RES	6,C	
CB B2	RES	6,D	
CB B3	RES	6,E	
CB B4	RES	6,H	
CB B5	RES	6,L	
CB BE	RES	7,(HL)	Очистка бита 7 в памяти (HL)
DD CB ii BE	RES	7,(IX+ii)	Очистка бита 7 в памяти (IX+ii)
FD CB ii BE	RES	7,(IY+ii)	Очистка бита 7 в памяти (IY+ii)
CB BF	RES	7,A	Очистка бита 7 в регистре
CB B8	RES	7,B	
CB B9	RES	7,C	
CB BA	RES	7,D	
CB BB	RES	7,E	
CB BC	RES	7,H	
CB BD	RES	7,L	
C9	RET		Возврат из подпрограммы

Код операции	Мнемоника	Операнды	Действие
D8	RET	C	Возврат, если carry установлен
F8	RET	M	Возврат, если минус
D0	RET	NC	Возврат, если carry сброшен
C0	RET	NZ	Возврат, если не ноль
F0	RET	P	Возврат, если плюс
E8	RET	PE	Возврат, если четный
E0	RET	PO	Возврат, если нечетный
C8	RET	Z	Возврат, если ноль
ED 4D	RETI		Возврат из прерывания
ED 45	RETN		Возврат из немаскируемых прерываний
CB 16	RL	(HL)	Сдвиг влево с переносом памяти в (HL)
DD CB ii 16	RL	(IX+ii)	Сдвиг влево с переносом памяти в (IX+ii)
FD CB ii 16	RL	(IY+ii)	Сдвиг влево с переносом памяти в (IY+ii)
CB 17	RL	A	Сдвиг влево с переносом регистра
CB 10	RL	B	
CB 11	RL	C	
CB 12	RL	D	
CB 13	RL	E	
CB 14	RL	H	
CB 15	RL	L	
17	RLA		Сдвиг влево с переносом A
CB 06	RLC	(HL)	Сдвиг влево памяти в (HL)
DD CB ii 06	RLC	(IX+ii)	Сдвиг влево памяти в (IX+ii)
FD CB ii 06	RLC	(IY+ii)	Сдвиг влево памяти в (IY+ii)
CB 07	RLC	A	Сдвиг влево регистра
CB 00	RLC	B	
CB 01	RLC	C	
CB 02	RLC	D	
CB 03	RLC	E	
CB 04	RLC	H	
CB 05	RLC	L	
07	RLCA		Сдвиг влево A
ED 6F	RLD		Обмен полубайтов влево
CB 1E	RR	(HL)	Сдвиг вправо с переносом памяти в (HL)
DD CB ii 1E	RR	(IX+ii)	Сдвиг вправо с переносом памяти в (IX+ii)
FD CB ii 1E	RR	(IY+ii)	Сдвиг вправо с переносом памяти в (IY+ii)
CB 1F	RR	A	Сдвиг вправо с переносом регистра
CB 18	RR	B	
CB 19	RR	C	
CB 1A	RR	D	
CB 1B	RR	E	
CB 1C	RR	H	
CB 1D	RR	L	
1F	RRA		Сдвиг вправо с переносом A
CB 0E	RRC	(HL)	Сдвиг вправо памяти в (HL)
DD CB ii 0E	RRC	(IX+ii)	Сдвиг вправо памяти в (IX+ii)
FD CB ii 0E	RRC	(IY+ii)	Сдвиг вправо памяти в (IY+ii)

Код операции	Мнемоника	Операнды	Действие
CB 0F	RRC	A	Сдвиг вправо регистра
CB 08	RRC	B	
CB 09	RRC	C	
CB 0A	RRC	D	
CB 0B	RRC	E	
CB 0C	RRC	H	
CB 0D	RRC	L	
0F	RRCA		Сдвиг вправо A
ED 67	RRD		Обмен полубайтов вправо
C7	RST	00H	CALL 0000H
CF	RST	08H	CALL 0008H
D7	RST	10H	CALL 0010H
DF	RST	18H	CALL 0018H
E7	RST	20H	CALL 0020H
EF	RST	28H	CALL 0028H
F7	RST	30H	CALL 0030H
FF	RST	38H	CALL 0038H
9E	SBC	A,(HL)	Вычитание из A с заемом памяти в (HL)
DD 9E ii	SBC	A,(IX+ii)	Вычитание из A с заемом памяти в (IX+ii)
FD 9E ii	SBC	A,(IY+ii)	Вычитание из A с заемом памяти в (IY+ii)
9F	SBC	A,A	Вычитание из A с заемом регистра
98	SBC	A,B	
99	SBC	A,C	
9A	SBC	A,D	
9B	SBC	A,E	
9C	SBC	A,H	
9D	SBC	A,L	
DE nn	SBC	A,nn	Вычитание из A с заемом nn
ED 42	SBC	HL,BC	Вычитание из HL с заемом регистра
ED 52	SBC	HL,DE	
ED 62	SBC	HL,HL	
ED 72	SBC	HL,SP	
37	SCF		Установка флага переноса.
CB C6	SET	0,(HL)	Установка бита 0 памяти в (HL)
DD CB ii C6	SET	0,(IX+ii)	Установка бита 0 памяти в (IX+ii)
FD CB ii C6	SET	0,(IY+ii)	Установка бита 0 памяти в (IY+ii)
CB C7	SET	0,A	Установка бита 0 регистра
CB C0	SET	0,B	
CB C1	SET	0,C	
CB C2	SET	0,D	
CB C3	SET	0,E	
CB C4	SET	0,H	
CB C5	SET	0,L	
CB CE	SET	1,(HL)	Установка бита 1 памяти в (HL)
DD CB ii CE	SET	1,(IX+ii)	Установка бита 1 памяти в (IX+ii)
FD CB ii CE	SET	1,(IY+ii)	Установка бита 1 памяти в (IY+ii)

Код операции	Мнемоника	Операнды	Действие
CB CF	SET	1,A	Установка 1 бита регистра
CB C8	SET	1,B	
CB C9	SET	1,C	
CB CA	SET	1,D	
CB CB	SET	1,E	
CB CC	SET	1,H	
CB CD	SET	1,L	
CB D6	SET	2,(HL)	Установка 2 бита памяти в (HL)
DD CB ii D6	SET	2,(IX+ii)	Установка 2 бита памяти в (IX+ii)
FD CB ii D6	SET	2,(IY+ii)	Установка 2 бита памяти в (IY+ii)
CB D7	SET	2,A	Установка 2 бита регистра
CB D0	SET	2,B	
CB D1	SET	2,C	
CB D2	SET	2,D	
CB D3	SET	2,E	
CB D4	SET	2,H	
CB D5	SET	2,L	
CB DE	SET	3,(HL)	Установка 3 бита памяти в (HL)
DD CB ii DE	SET	3,(IX+ii)	Установка 3 бита памяти в (IX+ii)
FD CB ii DE	SET	3,(IY+ii)	Установка 3 бита памяти в (IY+ii)
CB DF	SET	3,A	Установка 3 бита регистра
CB D8	SET	3,B	
CB D9	SET	3,C	
CB DA	SET	3,D	
CB DB	SET	3,E	
CB DC	SET	3,H	
CB DD	SET	3,L	
CB E6	SET	4,(HL)	Установка 4 бита памяти в (HL)
DD CB ii E6	SET	4,(IX+ii)	Установка 4 бита памяти в (IX+ii)
FD CB ii E6	SET	4,(IY+ii)	Установка 4 бита памяти в (IY+ii)
CB E7	SET	4,A	Установка 4 бита регистра
CB E0	SET	4,B	
CB E1	SET	4,C	
CB E2	SET	4,D	
CB E3	SET	4,E	
CB E4	SET	4,H	
CB E5	SET	4,L	
CB EE	SET	5,(HL)	Установка 5 бита памяти в (HL)
DD CB ii EE	SET	5,(IX+ii)	Установка 5 бита памяти в (IX+ii)
FD CB ii EE	SET	5,(IY+ii)	Установка 5 бита памяти в (IY+ii)
CB EF	SET	5,A	Установка 5 бита регистра
CB E8	SET	5,B	
CB E9	SET	5,C	
CB EA	SET	5,D	
CB EB	SET	5,E	
CB EC	SET	5,H	
CB ED	SET	5,L	
CB F6	SET	6,(HL)	Установка 6 бита памяти в (HL)
DD CB ii F6	SET	6,(IX+ii)	Установка 6 бита памяти в (IX+ii)
FD CB ii F6	SET	6,(IY+ii)	Установка 6 бита памяти в (IY+ii)

Код операции	Мнемоника	Операнды	Действие
CB F7	SET	6,A	Установка 6 бита регистра
CB F0	SET	6,B	
CB F1	SET	6,C	
CB F2	SET	6,D	
CB F3	SET	6,E	
CB F4	SET	6,H	
CB F5	SET	6,L	
CB FE	SET	7,(HL)	Установка 7 бита памяти в (HL)
DD CB ii FE	SET	7,(IX+ii)	Установка 7 бита памяти в (IX+ii)
FD CB ii FE	SET	7,(IY+ii)	Установка 7 бита памяти в (IY+ii)
CB FF	SET	7,A	Установка 7 бита регистра
CB F8	SET	7,B	
CB F9	SET	7,C	
CB FA	SET	7,D	
CB FB	SET	7,E	
CB FC	SET	7,H	
CB FD	SET	7,L	
CB 26	SLA	(HL)	Сдвиг влево арифм. памяти в (HL)
DD CB ii 26	SLA	(IX+ii)	Сдвиг влево арифм. памяти в (IX+ii)
FD CB ii 26	SLA	(IY+ii)	Сдвиг влево арифм. памяти в (IY+ii)
CB 27	SLA	A	Сдвиг влево арифм. регистра
CB 20	SLA	B	
CB 21	SLA	C	
CB 22	SLA	D	
CB 23	SLA	E	
CB 24	SLA	H	
CB 25	SLA	L	
ED 76	SLP		Z180: Низкая мощность/спящий режим
CB 2E	SRA	(HL)	Сдвиг вправо арифм. памяти в (HL)
DD CB ii 2E	SRA	(IX+ii)	Сдвиг вправо арифм. памяти в (IX+ii)
FD CB ii 2E	SRA	(IY+ii)	Сдвиг вправо арифм. памяти в (IY+ii)
CB 2F	SRA	A	Сдвиг вправо арифм. регистра
CB 28	SRA	B	
CB 29	SRA	C	
CB 2A	SRA	D	
CB 2B	SRA	E	
CB 2C	SRA	H	
CB 2D	SRA	L	
CB 3E	SRL	(HL)	Сдвиг вправо логич. памяти в (HL)
DD CB ii 3E	SRL	(IX+ii)	Сдвиг вправо логич. памяти в (IX+ii)
FD CB ii 3E	SRL	(IY+ii)	Сдвиг вправо логич. памяти в (IY+ii)
CB 3F	SRL	A	Сдвиг вправо логич. регистра
CB 38	SRL	B	
CB 39	SRL	C	
CB 3A	SRL	D	
CB 3B	SRL	E	
CB 3C	SRL	H	
CB 3D	SRL	L	

Код операции	Мнемоника	Операнды	Действие
96	SUB	(HL)	Вычитание из A памяти в (HL)
DD 96 ii	SUB	(IX+ii)	Вычитание из A памяти в (IX+ii)
FD 96 ii	SUB	(IY+ii)	Вычитание из A памяти в (IY+ii)
97	SUB	A	Вычитание регистра из A
90	SUB	B	
91	SUB	C	
92	SUB	D	
93	SUB	E	
94	SUB	H	
95	SUB	L	
D6 nn	SUB	nn	Вычитание из A nn
ED 34	TST	(HL)	Z180: Сравнение A с памятью в (HL)
ED 3C	TST	A	Z180: Сравнение A с регистром
ED 04	TST	B	
ED 0C	TST	C	
ED 14	TST	D	
ED 1C	TST	E	
ED 24	TST	H	
ED 2C	TST	L	
ED 64 nn	TST	nn	Z180: Сравнение A с nn
ED 74 pp	TSTIO	pp	Z180: Сравнение A с портом
AE	XOR	(HL)	Исключающее 'ИЛИ' A с памятью в (HL)
DD AE ii	XOR	(IX+ii)	Исключающее 'ИЛИ' A с памятью в (IX+ii)
FD AE ii	XOR	(IY+ii)	Исключающее 'ИЛИ' A с памятью в (IY+ii)
AF	XOR	A	Исключающее 'ИЛИ' A с регистром
A8	XOR	B	
A9	XOR	C	
AA	XOR	D	
AB	XOR	E	
AC	XOR	H	
AD	XOR	L	
EE nn	XOR	nn	Исключающее 'ИЛИ' A с nn

Регистры ввода-вывода Z180/64180

Имя	Адрес	Регистр
CNTLA0	0x00	Регистр управления A ASCI[2] канал 0
CNTLA1	0x01	Регистр управления A ASCI канал 1
CNTLB0	0x02	Регистр управления B ASCI канал 0
CNTLB1	0x03	Регистр управления B ASCI канал 1
STAT0	0x04	Регистр состояния ASCI канал 0
STAT1	0x05	Регистр состояния ASCI канал 1
TDR0	0x06	Регистр передачи данных ASCI, канал 0
TDR1	0x07	Регистр передачи данных ASCI, канал 1
TSR0	0x08	Регистр приема данных ASCI, канал 0
TSR1	0x09	Регистр приема данных ASCI, канал 1

[2] ASCI сокращение от Asynchronous Serial Communications Interface (Асинхронный последовательный интерфейс связи)

Имя	Адрес	Регистр
CNTR	0x0A	Регистр управления CSI/0[3]
TRDR	0x0B	Регистр передачи/приема данных CSI/0
TMDR0L	0x0C	Регистр таймера данных, канал 0L
TMDR0H	0x0D	Регистра таймера данных, канал 0H
RLDR0L	0x0E	Регтстр таймера перезагрузки, канал 0L
RLDR0H	0x0F	Регтстр таймера перезагрузки, канал 0H
TCR	0x10	Регистр таймера управления
TMDR1L	0x14	Регистр таймера данных, канал 1L
TMDR1H	0x15	Регистр таймера данных, канал 1H
RLDR1L	0x16	Регистр таймера перезагрузки, канал 1L
RLDR1H	0x17	Регистр таймера перезагрузки, канал 1H
FRC	0x18	Автономный счетчик (Free running counter)
SAR0L	0x20	Регистр исходного адреса DMA[4], канал 0L
SAR0H	0x21	Регистр исходного адреса DMA, канал 0H
SAR0B	0x22	Регистр исходного адреса DMA, канал 0B
DAR0L	0x23	Регистр адреса назначения DMA, канал 0L
DAR0H	0x24	Регистр адреса назначения DMA, канал 0H
DAR0B	0x25	Регистр адреса назначения DMA, канал 0B
BCR0L	0x26	Регистр счетчика байтов DMA, канал 0L
BCR0H	0x27	Регистр счетчика байтов DMA, канал 0H
MAR1L	0x28	Регистр адресов памяти DMA, канал 1L
MAR1H	0x29	Регистр адресов памяти DMA, канал 1H
MAR1B	0x2A	Регистр адресов памяти DMA, канал 1B
IAR1L	0x2B	Регистр адресов ввода-вывода DMA, канал 1L
IAR1H	0x2C	Регистр адресов ввода-вывода DMA, канал 1H
BCR1L	0x2E	Регистр счетчика байтов DMA, канал 1L
BCR1H	0x2F	Регистр счетчика байтов DMA, канал 1H
DSTAT	0x30	Регистр состояния DMA
DMODE	0x31	Регистр режима DMA
DCNTL	0x32	Регистр управления ожиданием DMA
IL	0x33	Interrupt vector low register
ITC	0x34	Регистр управления INT/TRAP
RCR	0x36	Регистр управления регенерацией
CBR	0x38	MMU[5] common base register
BBR	0x39	MMU bank base register
CBAR	0x3A	MMU common/bank area register
OMCR	0x3E	Регистр управления режимом работы
ICR	0x3F	Регистр управления вводом-выводом

[3] CSI/O сокращение от Clocked Serial I/O (синхронизированный последовательный ввод-вывод)
[4] DMA сокращение от direct memory access (прямой доступ к памяти)
[5] MMU сокращение от memory management unit (устройство управления памятью)

13 Редактор связей. Справочное руководство

HI-TECH C включает в себя перемещающий ассемблер и редактор связей, которые обеспечивают раздельную компиляцию исходных файлов на языке C. Это означает, что программа может быть разделена на несколько исходных файлов, каждый из которых может быть сведен к обозримому размеру для простоты редактирования и компиляции, затем каждый объектный файл компилируется отдельно и наконец все объектные файлы, соединяются в единственную исполняемую программу.

Ассемблер описан в руководстве для конкретной машины. Это приложение описывает теоретические основы и использование редактора связей.

13.1. Перемещение и перемещаемые секции (psect)

Основная задача редактора связей состоит в объединении нескольких перемещаемых объектных файлов в один. Объектные файлы называются перемещаемыми, поскольку файлы содержат в себе достаточно информации, чтобы любые ссылки на программу или адреса данных (например, адрес функции) в файле могли быть скорректированы с учетом того, где файл, в конечном счете, будет находиться в памяти после процесса компоновки. Поэтому файл называется перемещаемым. Перемещение может принимать две основные формы. Перемещение по имени, т.е. перемещение по окончательному значению глобального символа, или перемещение psect, т.е. перемещение по базовому адресу конкретного участка кода, например, фрагмента кода, содержащего фактические исполняемые инструкции.

13.1.1. Программные секции

Любой объектный файл может содержать байты, которые будут храниться в памяти в одной или нескольких программных секциях, которые будут упоминаться как psect. Эти psect представляют собой логические группы определенных типов байтов кода в программе. Программная секция, содержащая исполняемые инструкции обычно называют psect **text**. Остальные секции psect с инициализированными данными, называются просто psect **data** и psect с неинициализированными данными, называется psect **bss**.

На самом деле редактор связей обработает любое число psect, и в принципе может использовать большее количество в специальных приложениях. Однако компилятор C использует только эти три, упомянутые, и имена **text**, **data** и **bss** просто выбраны для идентификации. Редактор связей не придает специального значения имени psect.

Различие между psect data и bss можно проиллюстрировать, рассматривая две внешних переменные. Одна инициализируется значением 1, и другая не инициализируется. Первая будет помещена в psect data и вторая в psect bss. При запуске программы psect bss всегда очищается к 0, таким образом, вторая переменная во время выполнения будет инициализирована 0. Первая, однако, займет место в программном файле, и будет содержать свое инициализированное значение 1 при запуске. Вполне возможно изменить значение переменной в psect data во время выполнения, однако лучше этого не делать, поскольку это приводит к более согласованному использованию переменных и допускает повторно выполняемые и пригодные для записи в ПЗУ программы.

В секцию psect text помещаются все исполнимые инструкции. В CP/M-80 psect text обычно начинается у основания TPA, с которого начинается выполнение.

Обычно psect data следуют за psect text, и последней располагается bss. Секция bss не занимает места в программном (.COM) файле. Этот порядок psect может быть переопределен параметрами редактора связей. Это особенно полезно при создании кода для специального оборудования.

В MS-DOS и CP/M-86 psect упорядочены аналогичным образом, но так как процессоры 8086 имеют сегментные регистры, обеспечивающие перемещение, и text и data psect начинаются в 0, даже если они будут загружены в память одна за другой. Это позволяет 64 Кбайт для кода и 64 Кбайт для данных и стека. В исполняемый файл (.EXE или .CMD) помещается необходимая операционной системе информация для загрузки программы в память.

13.1.2. Локальные перемещаемые секции и большая модель памяти

Так как для практических целей на 8086 psect ограничены 64 Кб, чтобы позволить больше 64 Кбайт кода компилятор использует локальные psect. Секцию psect считают локальной, если у директивы .psect есть флаг LOCAL. Любое число локальных psect могут быть связаны из разных модулей не объединяясь, даже если они имеют одинаковое имя. Однако следует отметить, что ни одна локальная psect не может иметь такое же имя, как глобальная psect.

Все ссылки на локальную psect в том же модуле (или в той же библиотеке) будут рассматривать как ссылки на ту же psect. Между модулями, однако, две локальные psect с тем же именем рассматривают как различные. Чтобы позволить коллективную ссылку к локальным psect с помощью параметра -P (описана ниже), локальная psect может иметь имя класса связанного с ним. Это достигается с помощью флага CLASS в директиве .psect.

13.2. Глобальные символы

Редактор связей обрабатывает только символы, которые были объявлены ассемблером глобальными. На исходном уровне C это означает все имена, которые имеют внешний класс хранения и которые не объявлены как static. Эти символы могут ссылаться на другие модули, кроме того, в котором они определены. Задача редактора связей соотносить определение глобального символа со ссылками на него.

13.3. Использование

Команда вызова редактора связей имеет следующую форму:

```
LINK параметры файлы ...
```

Параметры - ноль или больше параметров редактора связей, каждый из которых каким-то образом изменяет поведение редактора связей. **Файлы** - один или несколько объектных файлов, и ноль или больше имен библиотек. Параметры распознаются в верхнем или нижнем регистре. Редактор связей распознает следующие параметры:

-R	Оставляет вывод перемещаемым.
-L	Сохраняет информацию об абсолютном перемещении, -LM сохраняет только информацию о перемещении сегмента.
-I	Игнорирует неопределенные символы.
-N	Сортирует символы по адресу.
-Caddr	Производит двоичный выходной файл со смещением addr.

-S Удаляет информацию о символах из выходного файла.

-X Подавляет локальные символы в выходном файле.

-Z Подавляет тривиальные (сгенерированные компилятором) символы в выходном файле.

-Oname Называет имя выходного файла.

-Pspec spec является спецификацией расположения psect.

-Mname Записывает распределение карты памяти компоновки в файл с именем name.

-Usymbol Делает symbol изначально неопределенным.

-Dfile Записывает файл символов

-Wwidth Определяет ширину карты

Рассмотрим каждый из них в отдельности:

Параметр -R дает редактору связей команду оставлять выходной файл (названый с помощь параметра -O или l.obj по умолчанию) перемещаемым. Это нормально потому, что существуют дополнительные файлы, которые будут связаны, а вывод этой компоновки будет использоваться впоследствии редактором связей в качестве входных данных. Без этого параметра редактор связей сделает выходной файл абсолютным, в котором все перемещаемые адреса преобразуются в абсолютные ссылки. Этот параметр не может использоваться с параметрами -L или -C.

Параметр -L заставляет редактор связей выводить нулевую информацию о перемещении, даже если файл будет абсолютным. Эта информация позволяет само перемещающимся программам знать, какие адреса должны быть перемещены во время выполнения. Этот параметр не может использоваться с параметром -C . Для создания исполняемого файла (т.е. файла .COM) необходимо использовать программу objtohex. Если используется параметр -LM, то сохраняется только информация о перемещении сегмента. Он используется в сочетании с большой моделью памяти. Objtohex будет использовать информацию о перемещении (при вызове с параметром -L), чтобы вставить адреса перемещения сегмента в исполняемый файл.

Параметр -I используется для связывания кода, который содержит символы, не определенные в любом другом модуле. Обычно он используется при разработке программы сверху вниз, при наличии в коде ссылок на подпрограммы, до написания кода самих подпрограмм.

При получении карты распределения памяти компоновки с помощью параметра -M таблица символов по умолчанию отсортирована по именам символов. Для сортировки по адресам может использоваться параметр -N.

Вывод редактора связей - по умолчанию объектный файл. Для создания исполняемой программы, он должен быть преобразован в исполнимый образ. Для CP/M это - .COM файл, который является просто образом исполняемой программы, как она должна выглядеть в памяти, начиная с адреса 100H. Редактор связей произведет такой файл с помощью параметра -C100H. Форматы файлов для других приложений, требующих файл с двоичный образом также могут производиться с помощью параметра -C. Адрес после -C может быть задан в десятичном виде (по умолчанию), восьмеричном виде (при помощи суффикса o или O) или шестнадцатеричном виде (при помощи суффикса h или H).

Обратите внимание, что из-за сложности форматов исполняемого файла для MS-DOS и CP/M-86, LINK не производит их (.EXE и соответственно .CMD) непосред-

ственно. Компилятор автоматически выполняет OBJTOHEX с подходящими параметрами, чтобы генерировать файла в корректном формате.

Параметры -S, -X и -Z, которые бессмысленны, когда используется параметр -C, соответственно удалят все символы, все локальные символы или все тривиальные локальные символы из выходного файла. Тривиальные символы - это символы, созданные компилятором, и состоящие из однобуквенных символов, за которыми следует строка цифр.

Если используется параметр -C по умолчанию выходному файлу присваивается имя 1.obj или 1.bin. Оно может быть переопределено с помощью параметра -Oname. Выходной файл будет называться name в данном случае. Обратите внимание, что к имени никакой суффикс не добавляется. Файл будет называться точно в соответствии с аргументом параметра.

Для некоторых специализированных приложений, например, создании кода для встроенного микропроцессора, редактору связей необходимо указать, по каким адресам должны быть расположены различные psect. Это достигается с помощью параметра -P. Далее следует спецификация, состоящая из списка разделенных запятой имен psect, каждого с необязательной спецификацией адреса. В отсутствие спецификации адреса для перечисленной psect она будет связана с предыдущей psect. Например

```
-Ptext=0c000h,data,bss=8000h
```

Это заставит psect text быть расположенной в 0C000H, psect data, начнется в конце psect text и psect bss, начнется с 8000H. Это может быть процессор с ПЗУ в 0C000H и ОЗУ в 8000H.

Если адрес компоновки, то есть адрес, по которому код будет адресоваться во время выполнения, а также адрес загрузки, то есть адрес смещения внутри выходного файла, различны (например, для 8086) можно указать адрес загрузки отдельно от адреса компоновки. Например:

```
-Ptext=100h/0,data=0C000h/
```

Эта спецификация заставит секцию text быть соединенной для выполнения в 100h, но загруженной в выходном файле в 0, в то время как секция data будет соединена для 0C000h, но загружена в файл рядом с psect text. Обратите внимание, что, если наклонная черта ('/') опущена, адрес загрузки совпадает с адресом компоновки, а если наклонная черта присутствует, но за ней отсутствует адрес, psect будет загружена после предыдущей psect.

Чтобы определить адреса компоновки и загрузки для локальных psect, может использоваться название группы, к которой принадлежат psect, вместо имени глобальной psect. У локальных psect тогда будет адрес компоновки, определенный параметром -P и адреса загрузки, увеличенные вверх от указанного адреса загрузки.

Параметр -Mname просит, чтобы карта распределения памяти компоновки, содержащая информацию с таблицей символов и адресов загрузки модулей была записана в файл с именем name. Если имя опущено, то карта будет записана в стандартный вывод. Может использоваться параметр -W для определения желаемой ширины карты.

Параметр -U позволяет указать редактору связей символ, который должен быть первоначально внесен в таблицу символов как неопределенный. Это полезно при загрузке исключительно из библиотек. Может использоваться несколько параметров -U.

Если необходимо использовать отладчик с соединяемой программой, полезно создать файл символов. Параметр -Dfile запишет такой файл символов с именем file или Lsym, если файл не будет задан. Файл символов состоит из списка адресов и символов, по одному в каждой строке.

13.4. Примеры

Ниже приведены некоторые примеры использования редактора связей. Однако, обратите внимание, что обычно нет необходимости вызывать редактор связей явно, так как он вызывается автоматически командой C.

```
LINK -MMAP -C100H START.OBJ MAIN.OBJ A:LIBC.LIB
```

Эта команда связывает файлы START.OBJ и MAIN.OBJ с библиотекой A:LIBC.LIB. На самом деле, только необходимые модули из библиотеки будут связаны с ними. Вывод в формате .COM должен быть помещен в файл с именем по умолчанию L.BIN. Карта должна быть записана в файл с именем MAP. Обратите внимание, что файл START.OBJ должен содержать загрузочный код, и при запуске программы фактически будет выполняться код с самым младшим адресом в этом файле, так как он будет находится в 100H.

```
LINK -X -R -OX.OBJ FILE1.OBJ FILE2.OBJ A:LIBC.LIB
```

Файлы FILE1.OBJ и FILE2.OBJ будут соединены с любыми необходимыми подпрограммами из A:LIBC.LIB, и оставлены в файле X.OBJ. Этот файл останется перемещаемым. Неопределенные символы не вызовут ошибку. Файл X.OBJ, вероятно, позже будет объектом другого вызова LINK. Все локальные символы будут удалены из выходного файла, тем самым, экономя пространство.

13.5. Вызов редактора связей

Редактор связей вызывается командой LINK, и обычно находится в CP/M на диске A:, или в каталоге A:\HITECH\ в MS-DOS. Он может быть вызван без параметров. В этом случае он запросит ввод из стандартного ввода. Если стандартный ввод будет файлом, то приглашение к вводу не будет напечатано. Входные данные, предоставленные таким образом, могут содержать символы нижнего регистра, так как CP/M преобразует всю командную строку в верхний регистр по умолчанию. Это полезно с параметрами -U и -P. Обычно этот способ вызова полезен, если число параметров LINK большое. Даже если список файлов слишком длинный, чтобы поместиться в одной строке, могут быть включены строки продолжения, поместив символ наклонной черты влево ('\') в конце предыдущей строки. Таким образом, могут быть заданы команды LINK практически неограниченной длины.

14 Библиотекарь

Программа библиотекарь, LIBR, имеет функцию объединения нескольких объектных файлов в один файл, известный как библиотека. Объединение нескольких таких объектных модулей преследует несколько целей:

1. сокращение числа соединяемых файлов;
2. ускорение доступа;
3. сокращение используемого места на диске .

Чтобы сделать концепцию библиотеки полезной, редактор связей должен обрабатывать модули в библиотеке иначе, чем объектные файлы. Если редактору связей указан объектный файл, он будет связан с последним связанным модулем. Однако, модуль из библиотеки, будет соединен только в том случае, если он определяет один или несколько символов, ранее известных, но не заданных редактору связей. Таким образом, модули из библиотеки будут связаны только при необходимости. Поскольку выбор модулей для связывания производится на первом проходе редактора связей, и поиск в библиотеке осуществляется линейным способом, можно упорядочить модули в библиотеке, чтобы создать специальные эффекты при соединении. Подробнее об этом будет сказано позже.

14.1. Формат библиотеки

Модули в библиотеке в основном только объединяются, но в начале библиотеки сохраняется каталог модулей и символов в библиотеке. Так как этот каталог меньше, чем сумма модулей, редактор связей может ускорить поиск в библиотеке, так как на первом проходе ему нужно читать только каталог, а не все модули. На втором проходе ему нужно читать только необходимые модули, осуществляя поиск не затрагивая остальные. Все это минимизирует дисковый ввод-вывод при компоновке.

Нужно отметить, что формат библиотеки приспособлен исключительно к объектным модулям и не является механизмом архивации общего назначения, используемым некоторыми другими системами компиляторов. Это имеет преимущество в том, что формат может быть оптимизирован в направлении ускорения процесса связывания.

14.2. Использование

Программа библиотекаря называется LIBR, и формат команды выглядит следующим образом:

```
LIBR k file.lib file.obj ...
```

Интерпретируя это, **LIBR** - имя программы, **k** - ключевая буква, обозначающая, запрашиваемую функцию библиотекаря (замена, извлечение или удаление модулей, перечисление модулей или символов), **file.lib** является именем файла обрабатываемой библиотеки и **file.obj** - ноль или более имен объектных файлов.

Ключевые буквы:

r замена модулей;
d удаление модулей;
x извлечение модулей;
m список имен модулей;
s список модулей с символами.

При замене или извлечении модулей, параметры file.obj - имена модулей, которые будут заменены или извлечены. Если параметры отсутствуют, то все модули в библиотеке будут заменены или извлечены соответственно. Добавление файла в библиотеку выполняется, запросив библиотекарь заменить его в библиотеке. Так как он отсутствует, модуль будет добавлен в библиотеку. Если используется ключ r и библиотека не существует, то она будет создана.

С ключевой буквой d названные объектные файлы будут удалены из библиотеки. В данном случае, отсутствие заданных имен объектных файлов является ошибкой.

Ключевые буквы m и s перечислят названные модули и, в случае использования ключевой буквы s, символы, определенные или упомянутые внутри (библиотекарем обрабатываются только глобальные символы). Как и с ключевыми буквами r и x, пустой список модулей означает все модули в библиотеке.

14.3. Примеры

Ниже приведены некоторые примеры использования библиотекаря.

LIBR m file.lib

Перечень всех модулей в библиотеке file.lib.

LIBR s file.lib a.obj b.obj c.obj

Перечень глобальных символов в модулях a.obj, b.obj и c.obj

LIBR r file.lib 1.obj 2.obj

Заменить модуль 1.obj в файле file.lib на содержимое объектного файла 1.obj и повторить для 2.obj. Если объектный модуль еще не присутствует в библиотеке, добавить его в конец.

LIBR x file.lib

Извлечь, без удаления, все модули из file.lib и записать их в виде объектных файлов на диск.

LIBR d file.lib a.obj b.obj 2.obj

Удалить объектные модули a.obj, b.obj и 2.obj из библиотеки file.lib.

14.4. Задание параметров

Поскольку часто необходимо передать LIBR много объектных файлов в виде параметров, а командные строки CP/M и MS-DOS ограничены 127 символами, LIBR принимает команды из стандартного ввода, если не заданы параметры командной строки. Если стандартный ввод будет присоединен к консоли, то LIBR выведет приглашение. Может быть введено несколько строк, используя наклонную черту влево в качестве символа продолжения в конце строки. Если стандартный ввод будет перенаправлен из файла, то LIBR возьмет ввод от файла без приглашения. Например:

```
LIBR
libr> r file.lib 1.obj 2.obj 3.obj \
libr> 4.obj 5.obj 6.obj
```

выполнит так же, как будто .obj файлы были введены в командной строке. Приглашение libr> было распечатано самой программой LIBR, остальной текст был набран в качестве входных данных.

```
LIBR <lib.cmd
```

Программа LIBR считает ввод из lib.cmd и выполнит команду, найденную там. Это позволяет передавать к LIBR команды практически неограниченной длины.

14.5. Формат листинга

Запрос LIBR перечислить имена модуля просто выведет список имен по одному в каждой строке на стандартный вывод. Ключ s приведет к тому же, списку символов после каждого имени модуля. Каждому символу будет предшествовать буква D или U, представляя определение или ссылку на символ соответственно. Параметр -W может использоваться для определения ширины бумаги для этой операции. Например, LIBR -w80 s file.lib перечислит все модули в file.lib с их глобальными символами с выводом отформатированным для принтера или дисплея с 80 столбцами.

14.6. Упорядочивание библиотек

Библиотекарь создает библиотеки с модулями в порядке, в котором они заданы в командной строке. При обновлении библиотеки порядок модулей сохраняется. Любые новые модули, добавляемые в библиотеку после ее создания, будут добавляться в конце.

Порядок следования модулей в библиотеке имеет важное значение для редактора связей. Если библиотека содержит модуль, который ссылается на символ, определенный в другом модуле в этой же библиотеке, модуль, определяющий символ, должен появиться после модуля, ссылающегося на символ.

14.7. Сообщения об ошибках

Программа LIBR выводит различные сообщения об ошибках, большинство из которых являются серьезной ошибкой, а некоторые представляют собой безобидное явление, о котором, тем не менее, будет сообщаться, если не использовался параметр -w. В этом случае все предупреждающие сообщения будут подавлены.

15 OBJTOHEX

Редактор связей HI-TECH способен к созданию простых двоичных файлов или объектных файлов в качестве выхода. Любой другой требуемый формат должен быть произведен, выполнив утилиту OBJTOHEX. Она позволяет преобразовать созданные редактором связей объектные файлы во множество разных форматов, включая различные шестнадцатеричные форматы. Программа вызывается следующим образом:

OBJTOHEX параметры входной_файл выходной_файл

Все параметры необязательные. Если выходной файл опущен, то по умолчанию он будет `l.hex` или `l.bin` в зависимости от того, используется ли параметр `-b`. Входным файлом по умолчанию является `l.obj`.

Возможны следующие параметры:

-Baddr Произвести вывод двоичного образа. Он подобен параметру `-C` редактора связей. Если `addr` поприсутствует, то начало образа в файле будет смещено на величину `addr`. Если `addr` опущен, то первый байт в файле будет инициализирован младшим байтом. Можно задать `addr` в десятичном, восьмеричном или шестнадцатеричном виде. Основание по умолчанию - десятичное, буквы суффикса `o` или `O` указывают восьмеричное, `h` или `H` указывают шестнадцатеричное основание. Таким образом `-B100H` произведет файл в формате `.COM`.

-I Включать в вывод записи символов в шестнадцатеричном формате Intel. У каждой записи символа есть форма, подобная объектной записи, но с различным типом записи. Байты данных в записи являются именем символа, а адрес является значением символа. Это полезно для загрузки в ПЗУ отладчиков.

-C Читать спецификацию контрольной суммы из стандартного ввода. Спецификация контрольной суммы описана ниже. Обычно спецификация находится в файле.

-Estack Этот параметр производит файл `.EXE` в формате MS-DOS. Необязательный параметр `stack` определяет максимальный размер стека, выделяемый программе при выполнении. По умолчанию программе будет выделен максимально доступный стек в пределах 64K данных. Если присутствует параметр `stack`, то размер стека не превысит значение параметра. Это полезно для ограничения объема памяти, используемой программой. Параметр `stack` принимает ту же форму, что и параметр `-B` выше.

-Sstack Этот параметр произведет файл `.CMD` для CP/M-86. Аргумент стека совпадает с параметром `-E`.

-Astack Он используется при создании файлов `a.out` формата для систем Unix (в частности Venix-86). Если аргумент `stack` будет ноль, то размер сегмента данных будет 64 Кбайт, иначе стек будет помещен ниже сегмента данных, а его размер установлен равным `stack`. Он должен быть согласован с соответствующими аргументами параметра `-p` редактора связей.

-M Этот флаг даст OBJTOHEX команду производить вывод в шестнадцате-

ричном формате 'S' Motorola.

-L Этот параметр используется при создании программ большой модели. Редактор связей будет использоваться с параметром -LM для сохранения информации о перемещении сегмента в объектном файле. Использование параметра -L в OBJTOHEX заставит его преобразовывать эту информацию о перемещении сегмента в надлежащие данные в исполняемом файле для использования при загрузке программы. Операционная система или код запуска среды выполнения будут использовать данные перемещения для корректирования ссылки сегмента на основе того, где в памяти программа фактически загружена. Если параметр -L будет сопровождаться именем символа, то информация перемещения будет храниться в адресе, представленном этим символом в выходном файле, например, _Bbss заставит быть сохраненным в основе psect bss (_Bbss, определен редактором связей, в качестве адреса загрузки psect bss). Если используется специальный символ Doshdr, тогда информация о перемещении, будет сохранена в заголовке файла .EXE. Это допустимо только в сочетании с параметром -E.

-S Параметр -S дает OBJTOHEX команду записать файл символов. Имя файла символов задается после -S, например, -Sxx.sym.

Если не будет специально указан другой формат, OBJTOHEX будет производить файл в шестнадцатеричном формате Intel. Это подходит для загрузки по линии связи, программирования ППЗУ и т.д. Формат HP, полезен для передачи кода в HP64000 для эмуляции или программирования ППЗУ.

Спецификация контрольной суммы позволяет автоматизированное вычисление контрольной суммы. Спецификация контрольной суммы принимает форму нескольких строк, каждая строка, описывает одну контрольную сумму. Синтаксис строки контрольной суммы следующий:

```
addr1-addr2 where1-where2 +offset
```

Все addr1, addr2, where1, where2 и offset являются шестнадцатеричными числами без обычного суффикса H. Такая спецификация говорит, что байты с addr1 до addr2 включительно должны быть суммированы, и сумма помещена в расположения начиная с where1 до where2 включительно. Для 8-разрядной контрольной суммы эти два адреса должны совпадать. Поскольку контрольная сумма сохраняет сначала младший байт, where1 должен быть меньше, чем where1 и наоборот. +offset является необязательным, но, если присутствует, значение offset будет использоваться для инициализации контрольной суммы. Иначе она будет обнулена. Например:

```
0005-1FFF 3-4 +1FFF
```

суммирует байты с 5 по 1FFFH включительно, затем добавить 1FFFH к сумме. 16 разрядная контрольная сумма будет помещена в расположения 3 и 4, младший байт в 3. Контрольная сумма инициализируется значением 1FFFH, чтобы обеспечить защиту от всех нулевых ПЗУ или ПЗУ неуместных в памяти. Во время выполнения проверки этой контрольной суммы будет добавлен последний адрес ПЗУ для проверки суммированием в контрольной сумме. Для рассматриваемого ПЗУ он должен быть 1FFFH. Однако, инициализируемое значение может использоваться в любой желаемой форме.

16 CREF

Утилита списка перекрестных ссылок CREF используется для форматирования необработанной информации о перекрестных ссылках, произведенной компилятором или ассемблером в отсортированный список. Файл с необработанными перекрестными ссылками создается компилятором с помощью параметра -CR. Ассемблер генерирует файл с необработанными перекрестными ссылками с помощью параметра -C (ассемблеры Z80 или 8086) или при помощи директивы OPT CRE (модификации ассемблеров 6800) или строки управления REF (ассемблер 8096). Общая форма команды CREF следующая:

```
CREF параметры файлы
```

где **параметры** - ноль или более параметров, описанных ниже и **файлы** один или несколько необработанных файлов перекрестных ссылок. Утилита CREF принимает следующие параметры:

-Ooutfile Позволяет указать имя выходного файла. По умолчанию листинг будет записан в стандартный поток вывода и может быть перенаправлен обычным способом. Кроме того, с помощью параметра -O имя выходного файла может быть указано, например -Oxxx.lst.

-Pwidth Этот параметр позволяет задавать ширину, с учетом которой листинг должен быть отформатирован, например -P132 будет форматировать листинг для принтера со 132 столбцами. Значение по умолчанию - 80 столбцов.

-Llength Определяет длину бумаги, на которой должен производиться листинг, например, если листинг должен быть напечатан на бумаге 55 строк вы будете использовать параметр -L55. Значение по умолчанию - 66 строк.

-Xprefix Параметр -X позволяет исключить символы из листинга, основываясь на префиксе, переданном в качестве аргумента к параметру -X. Например, если желательно исключить все символы, начиная с последовательности символов xyz, то используется параметр -Xxyz. Если в последовательности символов, встречается цифра, то она будет соответствовать любой цифре в символе, например -XX0 исключит любые символы, начинающиеся с буквы X и последующей цифры.

-F Параметр -F исключает из листинга любые ссылки из файлов с полным именем пути. Полное имя пути означает: имя файла, начинающееся с наклонной черты ('/'), или обратной наклонной черты ('\'), или имени файла, начинающегося с префикса из номера пользователя/буквы диска CP/M, например, 0:A:. Она предназначена для исключения из листинга любых ссылок символов, полученных из стандартных заголовочных файлов, например, использование параметра -F исключит любые ссылки из заголовочного файла STDIO.H.

-Hstring Параметр -H принимает строку в качестве аргумента, которая будет использоваться в качестве заголовка в листинге. По умолчанию заголовком является имя первого файла с необработанной информацией о перекрестных ссылках.

-Sstoplist Параметр -S должен иметь в качестве аргумента имя файла, содержащего список символов, которые не должны появляться в перекрест-

ных ссылках. Может быть задано несколько стоп-листов с помощью нескольких параметров -S.

Утилита CREF принимает подстановочные имена файлов и перенаправление ввода-вывода. Длинные командные строки могут быть предоставлены при вызове CREF без параметров и ввода командной строки в ответ на приглашение cref>. Обратная наклонная черта в конце строки будет означать, что далее следуют дополнительные командные строки.

17 Отладчик. Справочное руководство[6]

HI-TECH C поставляется с интерактивным средством отладки, ориентированным на программы C. Это не отладчик "исходного уровня", то есть он ничего не знает об исходном коде C, однако он умеет обрабатывать символы C и показывать последовательность вызова функции C.

Структура команды строится по образцу отладчика Unix, известного как adb. Он предоставляет средства для отображения памяти в различных основаниях или в виде инструкций, установки и удаления контрольных точек, которые могут иметь счетчик повторений и/или команду, связанную с ними. Можно установить контрольную точку, которая остановит выполнение только если определенное условие истинно.

Отладчик может использоваться с любым файлом .COM, однако для того, чтобы воспользоваться преимуществами символических объектов, необходимо создать файл символов, как правило, с помощью параметра -F в команде C. В этом файле по одной строке на символ содержатся имя символа и его шестнадцатеричное значение.

17.1. Вызов отладчика

Отладчик вызывается командой **DEBUG**. У нее может быть ноль, один или два параметра файла. Первый параметр - имя файла в формате .COM для отладки, второе имя файла символов. Если имя файла символов опущено, то символы будут не доступны. Если будет опущен файл формата .COM, то код не будет загружен. Некоторые примеры:

```
DEBUG fred.com
DEBUG bill.com l.sym
```

17.2. Структура времени выполнения

При выполнении **DEBUG** перемещает себя ниже BDOS, позволяя отлаживаемой программе как обычно быть загруженной с начала TPA. Таблица символов, если загружена, растет вниз от начала перемещенного отладчика. Запись BDOS в расположении 5 изменяется, чтобы отразить начало таблицы символов, а не начало BDOS. Таким образом таблица символов и отладчик не перезаписываются стеком отлаживаемой программы.

Контрольные точки вставляются в код как инструкции RST 8. Переход помещается в расположение 8 к обработчику прерывания отладчика. Это печально, если оказывается что ваша система использует RST 8 для прерываний, но это не более вероятно, чем то, что, она использует любое другое местоположение перезапуска.

17.3. Команды

Основной синтаксис команды отладчика:

адрес, счетчик команда модификатор дополнительные_условия

Это может показаться немного неясным, так что читайте дальше. Поля **адрес** и **счетчик** являются выражениями в их самой простой форме просто шестнадцатеричные числа. Поля **адрес** и **счетчик** не являются обязательными, но если счетчик должен быть указан без адреса, должна присутствовать запятая.

---

[6] Раздел не входил в оригинал руководчтва. Добавлен из файла debugman.txt

Поле **Команда** является однобуквенным символом, определяющем действие команды. Поле **модификатор** еще один символ, который более конкретно определяет команду. Поле **дополнительные_условия** зависит от конкретной команды, и обычно опускается.

17.3.1. Выражения

Выражения могут состоять из:

`.` Значение текущего расположения (не обязательно текущего PC или последней контрольной точки, это - внутреннее текущее значение).

`SYMBOL` Значение символа, просматриваемого в таблице символов. Если символ не будет найден, то будет разыскиваться тот же ожидаемый символ с предшествующим подчеркиванием. Это позволяет упоминать символы C без начального символа подчеркивания, добавляемого компилятором.

`INTEGER` Шестнадцатеричное целое число. Оно должно начинаться с цифры, иначе отладчик будет думать, что это - символ.

`<REGNAME` Она дает содержимое указанного регистра Z80. Имена регистров являются обычными именами Z80, только в нижнем регистре. См. команду **$r** ниже.

`(EXPR)` Могут использоваться круглые скобки для включения в выражение, чтобы изменить порядок вычисления.

`*EXPR` Содержание слова по адресу EXPR. Это является косвенностью, аналогичная косвенному оператору C.

`-EXPR` Отрицательное значение EXPR.

`~EXPR` Поразрядное дополнение EXPR.

`E1+E2` Сумма E1 и E2.

`E1-E2` Значение E1 минус значение E2.

`E1*E2` Умножение E1 на E2.

`E1%E2` E1 деленное на E2.

`E1&E2` Операция AND E1 с E2.

Применяются обычные правида приоритетов. Могут использоваться круглые скобки для изменения порядка вычислений.

17.3.2. Символы команд

Основным используемым символом команды является **/**. Он используется лля вывода на экран памяти в различных основаниях или в виде инструкции. Точный формат определяется символом модификатора или используемым ранее форматом, если символ модификатора опущен.

Модификаторами являются:

i Печать инструкции Z80.

h или **b** Печать в виде шестнадцатеричных байтов.

o Печать в виде восьмеричных байтов.

d Печать в виде десятичных байтов.

H или **W** Печать в виде шестнадцатеричных слов.

O Восьмеричных слов.

D Десятичных слов.

c Печать каждого байта в виде символа ASCII.

C Печать как символы ASCII, если печатаемые, в противном случае как **@x**, где **x** - соответствующий буквенный символ, например, **@C** для 3.

s Печатать строки символов до нуля.

a Печать адреса как символьное значение.

Если адрес определен, отображение начинается с указанного адреса. Будут распечатаны в количестве счетчика байты, инструкции, слова, строки или что-то другое, или один элемент, если счетчик опущен. Например:

```
fred,10/b
123/i
```

Первая команда распечатывает 16 байтов от адреса символа **fred**. Вторая отображает одну инструкцию из расположения **123H**.

Когда выдается такая команда, значение точки временно увеличивает общее количество отображаемых байтов. Последующая команда, состоящая исключительно из RETURN или ПЕРЕВОДА СТРОКИ, сделает временный инкремент точки постоянным, и выполнится команда **/**. Таким образом RETURN может использоваться, чтобы продвинуться вперед в память, выводя на экран память в том же формате как последнея команда **/**.

Команда **/** может также использоваться для изменения памяти. Команда **/w EXPR** запишет значение EXPR в память в текущем расположении (точка). Будут записаны или слово или байт в зависимости от последнего формата, используемого для команды **/**. Команда **/w** не может быть задана, если текущий формат отличается от типа слова или байта. Таким образом, память не может быть изменена в формате I. Когда-нибудь будет встроенный встроенный ассемблер, которого пока нет.

Команда **]** походит на команду **/**, за исключением того, что она выводит на экран порты ввода-вывода, а не содержание памяти. Она может использоваться только с форматами **h**, **b**, **o** или **d**. Кроме того, **]w** может использоваться, чтобы записать в порт ввода-вывода.

Команды **$** имеет различные модификаторы, описаные ниже:

c Печать трассировки стека C. Обратите внимание, что это не является надежным при использовании на оптимизированной программе, поскольку оптимизатор изменяет стек манипулируя кодом. Там может оказаться меньше аргументов, чем на самом деле. Параметры long будут всегда появляться как два целочисленных параметра.

b Отображаются установленные точки останова

s Установите предел для соответствий символов к заданному адресу. Это определяет максимальное значение смещения при печати значений типа sym+offset.

w Установка ширины терминала при адресации. Значение по умолчанию - 80 десятичное.

r Отображение содержания всех регистров Z80.

Команда **:** имеет следующие модификаторы:

r Выполнить программу по адресу. Если адрес опущен, попробует с символа start. Если он не будет найден, то отладчик будет жаловаться. С помощью этой команды **дополнительные_условия** будут предоставлены как строка параметров программы, т.е. в буфере по умолчанию в 80H. Вы должны гарантировать, что между параметрами **r** и **дополнительные_условия** имеется пробел. Например:

```
100:r arg1 arg2
```

c Продолжить программу с адреса или содержимого PC, если адрес опущен. Используется после контрольной точки.

s Как в **c**, но выполняет число инструкций определенное параметром счетчик, или 1, если счетчик опущен. Таким образом этот один шаг отлаживаемой программы.

b Устанавливает контрольную точку по адресу. Если параметр счетчик присутствует, то остановка в контрольной точке не произойдет, пока она не будет достигнута несколько раз, определяемых счетчиком. Дополнительное_условие может быть командой, выполняемой каждый раз при достижении контрольной точки. Если команда устанавливает точку к нулю, то программа остановится в контрольной точке, даже если счетчик не равен нулю.

. Устанавливает временную контрольную точку по адресу и продолжает выполнение. Когда следующая контрольная точка встречается, эта временная точка останова будет удалена.

d Очистить контрольную точку по адресу.

Команда **>** позволяет изменить значения регистров. Могут быть определены и регистровые пары и регистры. Также может быть изменен флаг прерываний. 0 означает выключен, 1 означает включен.

17.4. Пример

Ниже приведен пример использования отладчика:

```
A>type tst.c

main() {
    int i, j;
    scanf("%d", &i);
    printf("%d\n", j);                Здесь ошибка: вместо j должно быть i.
}

A>c -f tst.c                          Компилируем с запросом файла символов
A>debug tst.com l.sym                 По умолчанию имя этого файла - l.sym.
ZDEBUG
: printf/I                            Дизассемблируем начиная с printf.
_printf:        call    csv
:                                     Шаг вниз с помощью клавиши RETURN.
_printf+3:      push    ix
: :b                                  Устанавливаем здесь точку останова.
: :r                                  Запускаем программу без аргументов.
123                                   Функция scanf ждет ввода - вводим 123
   Breakpoint   _printf+3
_printf+3:      push    ix            Остановка в точке останова.
: $c                                  Получаем трассировку стека.
_printf(1872,0)                       1872 - строка формата "d\n"
_main()
: main/I                              Теперь смотрим main().
_main:          call    csv
: cr                                  Шаг вниз с помощью клавиши RETURN.
_main+3:        ld      hl,FFFC
: cr
```

```
_main+6:        add     hl,sp
: ,10                                   Дизассемблируем 16 инструкций.
_main+7:        ld      sp,hl
_main+8:        push    ix
_main+A:        pop     hl
_main+B:        dec     hl
_main+C:        dec     hl
_main+D:        push    hl
_main+E:        ld      hl,186F
_main+11:       push    hl
_main+12:       call    _scanf
_main+15:       ld      hl,4
_main+18:       add     hl,sp
_main+19:       ld      sp,hl
_main+1A:       ld      l,(ix+-4)       Здесь загружается j
_main+1D:       ld      h,(ix+-3)
_main+20:       push    hl              и помещается в стек.
_main+21:       ld      hl,1872
: main+1a/I
_main+1A:       ld      l,(ix+-4)
: /h                                    Смотрим байты в hex виде.
_main+1A:       DD                      Индексация префиксного байта.
: cr                                    Шаг вниз с помощью клавиши RETURN.
_main+1B:       6E
: cr
_main+1C:       FC                      Смещение индекса = -4.
: /w 0fe                                Изменяем на -2.
:
_main+1D:       DD
:
_main+1E:       66
:
_main+1F:       FD
: /w 0ff                                Изменяем старший байт на -1, чтобы ад-
                                        ресовать i вместо j.
: :r                                    Выполняем программу снова,
123                                     опять вводим то же число.
   Breakpoint   _printf+3
_printf+3:      push    ix
: $c
_printf(1872,7B)
_main()
: 7b=d                                  7b был параметр выше.
123                                     Теперь у нас есть правильное значение.
: :c                                    Продолжаем программу которая теперь
123                                     распечатывает правильное значение.
A>
```

Установка *Lucifer* на целевой компьютер

Для использования *Lucifer* в вашей целевой системе, необходимо скомпилировать монитор *Lucifer* TARGET.C и поместить его в ПЗУ. Если ваша система Z80 уже имеет монитор в ПЗУ, можно также загрузить целевой код *Lucifer* в ОЗУ. Если у вас есть одна из плат Z80, поддерживаемая поставляемым целевым кодом, вы сможете использовать прилагаемую программу монитора *Lucifer* без изменений. Однако, если ваша система Z80 не является одной из поддерживаемых, вам потребуется изменить целевой код перед его использованием. Области, которые обычно требуют модификации, это драйверы последовательного порта и код одного шага. При поставке одношаговый код сконфигурирован для использования одношаговой немаскируемых прерываний (NMI) на плате JED STD-801.

Для компиляции целевой программы используйте команду:

```
ZC -0 -OTARGET.HEX -Arom,ram,ramsize TARGET.C
```

где

rom	адрес в ПЗУ, где будет находиться монитор
ram	адрес в оперативной памяти, где будут расположены глобальные переменные и стек монитора
ramsize	размер области стека и глобальных переменных монитора

Пример

Для компиляции TARGET.C для работы по адресу 0F000H с 800H байтами оперативной памяти расположенной с адреса 7800H:

```
ZC -0 -OTARGET.HEX –AF000,7800,800 TARGET.C
```

Компилятор создаст TARGET.HEX, шестнадцатеричный файл формата Intel, готовый для помещения в ПЗУ.

Модификация целевой программы

Большинство изменений в TARGET.C будут сделаны для функций последовательного ввода-вывода, putch(), getch() и init_uart(), которые предварительно сконфигурированы для использования EXAR 88681 UART на плате JED STD-801. В системах, где нет доступных одношаговых прерываний, одношаговое прерывание может быть достигнуто путем установки точки останова в конечной точке каждой инструкции перед ее выполнением. Для условных ветвей, доступны две стратегии:

1. Предварительно оценить условие и определить, какой путь будет выбран.
2. Установить точку останова в **обеих** возможных конечных точках инструкции.

Если вы изменили TARGET.C для работы на системе Z80, которую мы в настоящее время не поддерживаем, отправьте нам измененный код, чтобы помочь в создании нашей библиотеки поддерживаемых систем.

Поддержка вызовов BDOS CP/M

Предоставленный целевой код *Lucifer* будет эмулировать вызовы BDOS CP/M, которые при необходимости обрабатывают консольный ввод-вывод. Если TARGET.C скомпилировать с флагом -DCPMCALLS, будет включен код эмуляции BDOS. Эмуляция BDOS полезна в системах, которые не поддерживают консольное устройство.

Чтобы скомпилировать TARGET.C с теми же адресами, что и раньше, но с включенной эмуляцией BDOS:

```
ZC -0 -OTARGET.HEX –AF000,7800,800 TARGET.C -DCPMCALLS
```

Если TARGET.C находит инструкцию JP, уже установленную в местоположение 5, он перенаправит JP к своему собственному обработчику и сохранит старый вектор BDOS для использования с вызовами BDOS, который он не поддерживает.
Поддерживаемые вызовы CP/M:

1 Ввод с консоли;
2 Вывод на консоль;
6 Прямой ввод/вывод с консоли;
10 Чтение буфера консоли;
11 Получение статуса консоли.

Все остальные вызовы передаются старому обработчику BDOS. Если обработчик BDOS не найден в местоположении 5, любые нереализованные вызовы BDOS заставляют целевой код печатать сообщение об ошибке через последовательный порт. Было бы целесообразно изменить TARGET.C, включив в него всю эмуляцию BDOS CP/M, включая доступ к файлам в хост-системе через последовательный канал. Мы не реализовали поддержку удаленного файлового сервера, так как в большинстве случаев, если требуется доступ к файлу, пользователь все равно будет работать в системе CP/M.

Приложение 1 Сообщения об ошибках

Ниже перечислены сообщения об ошибках, выдаваемые компилятором. Каждое сообщение сопровождается именем программы, выдавшей его и некоторым дополнительным описанием причины сообщения или рекомендацией.

'.' expected after '..' **P1**

('. ' *ожидается после* '. .')

Символ многоточие должен иметь три точки

actuals too long **CPP**

(*фактические слишком длинные*)

Уменьшить длину параметров макросов

argument list conflicts with prototype **P1**

(*список параметров конфликтует с прототипом*)

Список параметров в определении функции должен согласоваться с прототипом, если он существует

argument redeclared **P1**

(*параметр объявлен повторно*)
Этот параметр объявлен дважды

arithmetic overflow in constant **CGEN**

(*арифметическое переполнение в константе*)

Оценка этого константного выражения привела к арифметическому переполнению. Это может быть или не быть настоящей ошибкой.

array index out of bounds **P1**

(*индекс массива вне границ*)

Индексное выражение массива вычисляется в константу, которая меньше ноля или больше или равна размерности массива

Assertion **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

attempt to modify const object **P1**

(*попытка изменить объект const*)

Предпринята попытка присвоить или иным образом изменить объект, обозначенный как 'const'

bad bitfield type **P1**

(*некорректный тип битового поля*)

Битовые поля должны иметь тип 'int'

Bad conval **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

Bad dimensions **CGEN**

(*некорректные размерности*)

Массив имеет плохие размерности - вероятно ноль

Bad element count expr **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

bad formal **CPP**

(*некорректный формальный параметр*)

Проверьте синтаксис определения макроса

bad include syntax **CPP**

(*некорректный синтаксис include*)

Используйте только "" и <> для включаемых файлов

Bad int. code **CGEN**

(*некорректный внутренний код*)

Файл с промежуточным кодом поврежден - возможно, исчерпав дисковое пространство

Bad -M option **CGEN**

(*некорректный параметр -M*)

Неизвестный параметр `-M` передается в генератор кода

Bad mod '+' for how = c **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

bad object code format **LINK**

(*плохой формат объектного кода*)

Этот файл поврежден или не допустимый объектный файл

Bad op d to swaplog **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

Bad op n to revlog **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

bad origin format in spec **LINK**

(*плохой формат источника в спецификации*)

Недопустимый адрес в параметре `-p`

bad '-p' format **LINK**

(*плохой формат '-p'*)

Указан недопустимый параметр `-p`

Bad pragma c **CGEN**

(*некорректная прагма c*)

Генератору кода была передана прагма, о которой он не знает

Bad putwsize **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

bad storage class **PI, CGEN**

(*неверный класс памяти*)

Указанный класс памяти недопустим

Bad U usage **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

Bit field too large (n bits) **CGEN**

(*Слишком большое битовое поле (n бит)*)

Битовое поле не может быть больше чем `int`

Cannot get memory **LINK**

(*не удается получить память*)

Редактор связей исчерпал динамическую память

Can't be both far and near **P1**

(*far и near не могут быть одновременно*)

Ключевые слова 'far' и 'near' не могут появиться в одном спецификаторе типа

can't be long **P1**

(*не может быть long*)

char и short не могут быть long

can't be register **P1**

(*не может быть register*)

extern или static переменные не могут быть register

can't be short **P1**

(*не может быть short*)

float и char не могут быть short

can't be unsigned **P1**

(*не может быть unsigned*)

float не может быть unsigned

can't call an interrupt function **P1**

(*не удается вызвать функцию прерывания*)

Функция определенная 'interrupt' может быть вызвана только аппаратно, а не вызовом обычной функции

Can't create filename **CGEN**

(*не удается создать имя файла*)

Указанный файл не может быть создан

Can't create xref file **P1**

(*не удается создать файл xref*)

Не удалось создать указанный файл перекрестных ссылок

Can't create **CPP**

(*не удается создать*)

Выходной файл не может быть создан

Can't create **LINK**

(*не удается создать*)

Редактор связей не может создать файл

Can't find include file **CPP**

(Не удается найти включают файл include)

Проверьте и исправьте имя файла include - пробелы в именах файлов не допустимы

Can't find register for bits **CGEN**

(Не удается найти регистр для битов)

Внутренняя ошибка - обратитесь в HI-TECH

Can't generate code for this **CGEN**

(*Не удается создать код для этого*)

Генератор кода неспособен генерировать код выражения для этого выражения - упрощение выражения (например, вычисление значений во временные переменные) как правило, исправлять ее, иначе свяжитесь с HI-TECH

can't have array of functions **P1**

(не может быть массива из функций)

Нельзя иметь массив функций – вы можете иметь массив указателей на функции

Can't have 'port' variable **CGEN**

(*не может быть переменной 'port'*)

Вы не можете объявить переменную с квалификатором 'port' - Вы можете использовать port только для квалификации указателей или преобразования типа постоянных значений

can't have storage class **P1**

(*не может иметь класс памяти*)

Класс памяти не может появиться в параметре прототипа

can't initialise auto aggregates **P1**

(*не удается инициализировать агрегаты auto*)

Вы не можете инициализировать структуру или массив внутри функции, если они не являются статическими

can't initialize arg **P1**

(*не удается инициализировать параметр*)

Параметр не может иметь инициализатор

can't mix proto and non-proto args **P1**

(*нельзя смешивать прототип и не прототип параметры*)

Вы не можете смешать прототип и не прототип параметры в одном определении функции

Can't open filename **CGEN**

(*не удается открыть файл*)

Указанный файл не может быть открыт для чтения

Can't open **LINK**

(*не удается открыть*)

Редактор связей не может открыть файл

Can't seek **LINK**

(*Невозможно найти*)

Редактор связей не смог выполнить поиск в файле

can't take address of register **P1**

(*не удается получить адрес регистра*)

Вы не можете получить адрес переменной регистровой переменной

can't take sizeof func **CGEN**

(*Невозможно получить размер функции*)

Вы не можете получить размер функции. Вы можете получить размер вызова функции

can't take this address **P1**

(*не удается получить этот адрес*)

Выражение не имеет адреса

'case' not in switch **P1**

(*'case' не в switch*)

Метка 'case' допустима только в switch

char const too long **P1**

(*символьная константа слишком длинная*)

Символьная константа может содержать только один символ

close error (disk space?) **P1**

(*ошибка закрытия (дисковое пространство?)*)

Возможно, недостаточно места на диске

common symbol psect conflict **LINK**

(*общий символ psect конфликт*)

Общий символ определен более чем в одной psect

constant conditional branch **CGEN**

(*постоянный условный переход*)

У вас есть конструкция программы, тестирующая константное выражение, например, while(1). Вы должны заменить ее на более эффективную - for(;;)

constant expression required **P1**

(*требуется константное выражение*)

Требуется константное выражение , например, в размерности массива

constant operand to || or && **CGEN**

(*константа операнд к || или &&*)

У логического оператора есть постоянный операнд, который был оптимизирован

declarator too complex **P1**

(*слишком сложный оператор объявления*)

Это объявление слишком сложное для обработки компилятором

default case redefined **P1**

(переопределен выбор по умолчанию)

В инструкции switch допускается только одна метка "default"

'default' not in switch **P1**

(*default вне switch*)

Метка 'default' разрешена только в switch

digit out of range **P1**

(*цифра вне диапазона*)

Восьмеричная константа не может содержать 7 или 8, и десятичная константа не может содержать A-F

dimension required **P1**

(*требуется размерность*)

В объявлении массива размерность требуется для всех индексов кроме старшего

Division by zero **CGEN**

(*Деление на ноль*)

В этом выражении попытка деления на ноль

Duplicate case label n **CGEN**

(*Двойная метка case n*)

В этом switch есть две метки case, имеющие одинаковое значение

Duplicate -d flag **LINK**

(*дублирование флага -d*)

Редактор связей может иметь только один флаг -d

duplicate label **PI**

(*дублирование метки*)

Эта метка определена дважды

Duplicate -m flag **LINK**

(*дублирование флага -m*)

Редактор связей может иметь только один флаг -m

duplicate qualifier **P1**

(*дублирование квалификатора*)

Тот же квалификатор появляется несколько раз в этом описателе типа

entry point multiply defined **LINK**

(*Точка входа определена многократно*)

Программа может иметь только одну точку входа (начальный адрес)

EOF in #asm **P1**

(*EOF в #asm*)

Конец файла обнаружен, после #asm и прежде чем был замечен #endasm

Error closing output file **CGEN,CPP**

(*Ошибка закрытия выходного файла*)

Вероятно означает, что вы исчерпали место на диске

excessive -I file ignored **CPP**

(*излишняя -I файл игнорируется*)

Используйте меньшее количество параметров -I

expand - bad how **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

expand - bad which **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

exponent expected **P1**

(*ожидается экспонента*)

В константе с плавающей точкой после 'e' или 'E' ожидается экспонента. Экспонента должна содержать только +, - и цифры 0-9

Expression error **CGEN**

(*Ошибка выражения*)

Внутренняя ошибка - обратитесь в HI-TECH

expression generates no code **CGEN**

(*выражение не генерирует кода*)

Это выражение не имеет никаких побочных эффектов и таким образом не генерирует кода. Оно было оптимизировано

expression syntax **P1**

(*синтаксис выражения*)

Выражение сформировано неправильно

expression too complex **P1**

(*выражение слишком сложное*)

Выражение имеет слишком много вложенных скобок или других вложенных конструкций

Fixup overflow referencing **LINK**

(*сегментное переполнение сылки*)

Редактор связей переместил ссылку на psect или символ, и перемещенный адрес слишком большой, чтобы вписаться в пространство, например, перемещенный однобайтовый адрес превышает 256, или перемещенный 16-битный адрес превышает 65536

float param coerced to double **P1**

(*параметр float приведен к double*)

Этот параметр float был преобразован в double - прототип будет отвергать это приведение

function() declared implicit int **P1**

(*function() объявлена неявным int*)

Эта функция была вызвана без явного объявления. Полезно явно объявить все функции, предпочтительно с помощью прототипа. Это позволит избежать многих потенциальных ошибок, когда ваша программа состоит из более, чем одного исходного файла

function does not take arguments **P1**

(*функция не принимает параметры*)

Прототип для этой функции указывает, что она не принимает параметры

function or function pointer **P1**

(*функция или указатель на функцию*)

Для вызова функции требуется идентификатор функции или указатель на функцию.

functions can't return arrays **P1**

(*функции не могут возвращать массивы*)

Функция не может возвратить массив - она может возвращать указатель

functions can't return functions **P1**

(*функции не могут возвращать функции*)

Функция не может возвращать функцию - она может возвращать указатель на функцию

hex digit expected **P1**

(*ожидается шестнадцатеричная цифра*)

После '`0x`' ожидается шестнадцатеричная цифра

identifier is a structure tag **P1**

(*идентификатор является именем структуры*)

Имя структуры используется в контексте, где ожидается имя другого типа, например, выражение `struct fred`, где `fred` был ранее объявлен как `union fred`.

identifier is a union tag **P1**

(*идентификатор является именем объединения*)

Аналогисна вышеупомянутой ошибке

identifier is an enum tag **P1**

(*идентификатор является именем перечисления*)

Аналогисна вышеупомянутой ошибке

identifier: large offset **CGEN**

(*идентификатор: большое смещение*)

Только Z80: Этот идентификатор имеет большое смещение в фрейме стека, и таким образом доступ к нему неэффективен. В функции любые массивы должны быть объявлены после любых простых переменных

identifier redeclared **P1**

(*идентификатор переопределен*)

Идентификатор был переопределен с разными атрибутами

identifier redefined **P1**

(*идентификатор повторно объявлен*)

Идентификатор был определен дважды

If-less else **CPP**

(*if- отсутствует else*)

Проверьте использование `#if`

If-less endif **CPP**

(*if- отсутствует endif*)

Проверьте использование `#if`

illegal '#' directive **P1**

(*недопустимая директива '#'*)

директива #, переданная через первый проход, неизвестна. Если это происходит с `#include`, она может быть вызвана предыдущим, включаемым файлом, не имеющего <CR><LF> или символа новая строка а последней строке.

Illegal character in preprocessor if **CPP**

(*Недопустимый символ в препроцессоре if*)

Проверить наличие странных символов

illegal character **P1**

(*Недопустимый символ*)

Обнаружен символ, неизвестный компилятору. Указанное значение восьмеричное значение символа

illegal conversion between pointer **P1**

(*недопустимое преобразование между указателем*)

Выражение заставляет один тип указателя преобразовать в другой несовместимый тип

illegal conversion of integer to **P1**

(*недопустимое преобразование целого числа к*)

Целое число используется вместо указателя - ожидается указатель

illegal conversion of pointer to **P1**

(*недопустимое преобразование указателя к*)

Указатель используется вместо целого числа – ожидается целое число

illegal conversion **P1**

(*недопустимое преобразование*)

Преобразование типов здесь недопустимо

Illegal flag **LINK**

(*недопустимый флаг*)

Этот параметр недопустим

illegal function qualifier(s) **P1**

(*недопустимый квалификатор(ы) функции*)

Функция не может иметь квалификатор '`const`'

illegal initialization **P1**

(*недопустимая инициализация*)

Инициализация этой переменной недопустима

Illegal number **CPP**

(*Недопустимое число*)

Проверьте синтаксис числа

illegal type for array dimension **P1**

(*недопустимый тип размерности массива*)

Размерность массива должна быть целочисленной величиной

illegal type for index expression **P1**

(*недопустимый тип индексного выражения*)

Индекс массива должен быть простым целочисленным выражением

illegal type for switch expression **P1**

(*недопустимый тип выражения в switch*)

Выражение в 'switch' должно быть целочисленным

illegal use of void expression **P1**

(*недопустимое использование выражения void*)

Выражения void не могут использоваться в любом случае

implicit conversion of float to **P1**

(*неявное преобразование float к*)

Значение с плавающей точкой было преобразовано в целочисленное целое число - может произойти усечение

implicit return at end of non-void **P1**

(*неявный возврат в конце функции non-void*)

Функции с типом, отличным от void имеет функцию с возвратом без оператора return

implict signed to unsigned **P1**

(*неявное преобразование signed в unsigned*)

Здесь может произойти нежелательное расширение знака, при преобразовании добавьте явное преобразование типа, чтобы выполнить преобразование, которое вы хотите

inappropriate break/continue **P1**

inappropriate 'else' **P1**

(*неуместные break/continue или 'else'*)

'else' появился без соответствия 'if'

inconsistent storage class **P1**

(*неправильный класс памяти*)

Только один класс памяти может быть определен в объявлении

inconsistent type **P1**

(*неправильный тип*)

в объявлении может быть определен только один основной тип

initialisation illegal in arg list **P1**

(*недопустимая инициализация в списке параметров*)

Вы не можете инициализировать параметр функции

initialisation syntax **P1**

(*синтаксис инициализации*)

Синтаксис этой инициализации недопустим

initializer in 'extern' declaration **P1**

(*инициализатор в объявлении 'extern'*)

Объявление с ключевым словом 'extern' имеет инициализатор. Это не разрешено, поскольку объявление extern не резервирует место в памяти

integer constant expected **P1**

(*ожидается целочисленная константа*)

Здесь ожидается целочисленная константа

integer expression required **P1**

(*требуется целочисленное выражение*)

Здесь требуется целочисленное выражение

integral type required **P1**

(*требуется целочисленный тип*)

Здесь требуется целочисленный тип

large offset **CGEN**

(*Большое смещение*)

Только Z80: Этот идентификатор имеет большое смещение в фрейме стека, и таким образом доступ к нему неэффективен. В функции любые массивы должны быть объявлены после любых простых переменных

Line too long **P1**

(*Слишком длинная строка*)

Исходная строка слишком длинная, или не имеет <CR><LF> или символа новой строки в конце

local psect conflicts with global **LINK**

(*локальный psect конфликтует с глобальным*)

Локальная psect не может иметь то же имя psect, которое присвоено глобальной psect

logical type required **P1**

(*требуется логический тип*)

Логический тип (т. е. целочисленный тип) является обязательным предметом условного выражения

lvalue required **P1**

(*требуется левое выражение*)

Левое выражение (lvalue), т.е. что-то, что может быть присвоено, требуется после '&' или с левой стороны от присваивания

macro recursion **CPP**

(*макро-рекурсия*)

Макрос препроцессора попытался развернуть себя. Это создало бы бесконечную рекурсию

member is not a member of the **P1**

(*элемент не является элементом*)

Этот элемент не входит в структуру или объединение структуры/ объединения, с которым она используется

members cannot be functions **P1**

(*элементы не могут быть функциями*)

Элемент не может быть функцией - он может быть указателем на функцию

Missing arg to -u **LINK**

(*Отсутствует аргумент у -u*)

`-u` требует аргумент

Missing arg to -w **LINK**

(*Отсутствует аргумент у -w*)

`-w` требует аргумент

missing) **CPP**

(*отсутствует)*)

Поместите корректную) в выражение

Missing number after % in -p option **LINK**

(*Отсутствует число после % в параметре -p*)

После `%` в параметре `-p` должно быть число

Missing number after pragma 'pack' **P1**

(*Отсутствует число после прагмы 'pack'*)

Корректный синтаксис - `#pragma pack(n)`, где `n` равняется 1, 2 или 4.

module has code below file base **LINK**

(*у модуля есть код ниже начала файла*)

Был определен параметр `-C`, но у программы есть код ниже адреса, определенного как начало двоичного файла

multiply defined symbol **LINK**

(*многократное определение символа*)

Символ определен несколько раз

name is a union, struct or enum **P1**

(*имя - объединение, структура или перечисление*)

Имя объединения, структуры или перечисления было повторно использовано в другом контексте

No case labels **CGEN**

(*отсутствуют метки case*)

У этого `switch` нет меток `case`

no identifier in declaration **P1**

(*в объявлении отсутствует идентификатор*)

Это объявление должно содержать в себе идентификатор

No room **CGEN**

(*нет памяти*)

Генератор кода исчерпал динамическую память. Вы должны сократить количество символов и/или упростить выражения

No source file **CPP**

(*отсутствует файл источника*)

Исходный файл не найден – проверьте орфографию, пути к каталогам и т.д.

no space **CPP**

(*нет места*)

Сократите число/размер макроопределений

no start record: entry point **LINK**

(*отсутствует запись запуска: точка входа*)

Ни один из объектных файлов, переданных редактору связей, не содержал начальный адрес запуска. Редактор связей установил значение начального адреса равным `0`

Non-constant case label **CGEN**

(*метка case не является константой*)

Эта метка `case` не вычисляется к целочисленной константе

non-void function returns no value **P1**

(*отличноя от void функция не возвращает значение*)

Функция, которая должна возвращать значение имеет оператор '`return`' без значения

not a variable identifier **P1**

(*Идентификатор не является переменной*)

Идентификатор не является переменной - он может быть, например меткой или именем структуры

not an argument **P1**

(*не является параметром*)

Этот идентификатор отсутствует в списке параметров этой функции

only functions may be qualified **P1**

(*могут быть квалифицированы только функции*)

Квалификатор типа '`interrupt`' может быть применен только к функциям, не переменным.

only functions may be void **P1**

(*только функции могут быть void*)

Только функции, не переменные, могут быть объявлены `void`

only lvalues may be assigned to **P1**

(*может быть присвоено только левой части (lvalue)*)

Вы попытались изменить или изменили выражение, которое не идентифицирует место хранения

only register storage class allowed **P1**

(*допустим только класс памяти register*)

Параметр может только быть `auto` или `register`

operands of operator not same **P1**

(*операнды и оператор разные*)

операнд названного оператора в указателе типа выражения оба являются указателями, но имеют разные типы.

operands of operator not same type **P1**

(*операнды и оператор разного типа*)

Операнды названом операторе в выражении имеют несовместимые типы

pointer required **P1**

(*требуется указатель*)

Требуется указатель после '*' (знак косвенной операции)

popreg - bad reg **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

portion of expression has no effect **CGEN**

(*часть выражения не имеет никакого эффекта*)

Часть этого выражения не оказывает никакого влияния на его значение и не имеет побочных эффектов

probable missing '}' in previous block **P1**

(*в предыдущем блоке вероятно отсутствует '}'*)

Выявлено объявление там, где ожидается выражение. Вероятной причиной этого является то, что пропущена закрывающая '}' в функции выше этой точки.

psect cannot be in classes a and b **LINK**

(*psect не может быть в классах a и b*)

Psect может быть только в одном классе

psect exceeds max size **LINK**

(*psect превышает максимальный размер*)

Эта psect больше, чем указанный максимальный размер

psect is absolute **LINK**

(psect является абсолютной)

Эта psect является абсолютной и не может иметь адрес ссылки, указанный в параметре -p

Psect not loaded on 0xhexnum **LINK**

(*Psect, не загружена на 0xhexnum*)

Эта psect должна быть загружена на определенной границе

Psect not relocated on 0xhexnum **LINK**

(*Psect не перемещена на 0xhexnum*)

Эта psect должна быть связана на конкретной границе

psect origin multiply defined **LINK**

(*начало определено несколько раз*)

Этой psect определили адрес ссылки несколько раз

pushreg - bad reg **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

redundant & applied to array **P1**

(*избыточный & примененный к массиву*)

Тип массива имеет оператор '&', примененный к нему. Он был проигнорирован, так как использование массива неявно дает его адрес

regused - bad arg to G **CGEN**

Внутренняя ошибка - обратитесь в HI-TECH

signatures do not match **LINK**

(сигнатуры *не совпадают*)

Внешняя функция объявлена с неправильным прототипом. Например, если параметр объявлен как `long` в объявлении `extern`, а в действительности он `int`, произойдет несоответствие сигнатуры.

signed bitfields not supported **P1**

(битовые поля со знаком не поддерживаются)

Поддерживаются только беззнаковые битовые поля

simple type required **P1**

(*требуется простой тип*)

Здесь нельзя использовать тип массив или структура.

Sizeof yields 0 **CGEN**

(*sizeof дает 0*)

Размер объекта вычисляется к нулю, в контексте, где это недопустимо, например, инкрементирование указателя на объект нулевой длины.

storage class illegal **P1**

(*недопустимый класс памяти*)

Класс памяти не может быть определен здесь

struct/union member expected **P1**

(*ожидается элемент структуры/объединения*)

После '`.`' или '`->`' требуется элемент структуры или объединения

struct/union redefined **P1**

(*структура/объединение переопределена*)

Эта структура или объединение были определены дважды

struct/union required **P1**

(*требуется структура/объединение*)

Перед '`.`' требуется идентификатор структуры или объединения

Switch on long! **CGEN**

(*Switch на long*)

Переключение на выражение типа `long` не поддерживается

symbol cannot be global **LINK**

(*символ не может быть глобальным*)

Стек, имя файла или символы номера строки не могут быть глобальной переменной

Syntax error in checksum list **LINK**

(*Синтаксическая ошибка в списке контрольной суммы*)

Список контрольной суммы является недопустимым

token too long **CPP**

(*лексема слишком длинная*)

Сократите лексему (например, идентификатор)

too few arguments **P1**

(*слишком мало параметров*)

Прототип этой функции перечисляет больше параметров, чем предоставлено

too many arguments **P1**

(*слишком много параметров*)

Было предоставлено больше параметров, чем перечислено в прототипе этой функции

Too many cases in switch **CGEN, PI**

(*слишком много case в операторе switch*)

В этом операторе `switch` слишком много `case`

too many -D options **CPP**

(*слишком много параметров -D*)

Используйте меньше параметров `-D`

too many defines **CPP**

(*слишком много определяет*)

Сократите количество макроопределений

Too many errors **CGEN**

(*слишком много ошибок*)

`CGEN` прекратил обработку, из-за слишком большого количества ошибок.

too many formals **CPP**

(*много формальных параметров*)

Сократите количество параметров в этом макроопределении

Too many initializers **CGEN**

(*слишком много* инициализаторов)

Есть слишком много инициализаторов для этого объекта

Too many psects **LINK**

(*слишком много psect*)

Слишком много psects для таблицы символов

Too many symbols **LINK**

(*слишком много символов*)

Слишком много символов для таблицы символов редактора связей

too many -U options **CPP**

(*слишком много параметров -U*)

Используйте меньше параметров `-U`

too much defining **CPP**

(*слишком много определяющих*)

Сократите количество/размер макросов

too much indirection **P1**

(*слишком много операций косвенной адресации*)

В этом объявлении слишком много '*'

too much pushback **CPP**

(*слишком много возвратов назад*)

Упростите используемые макросы

type conflict **P1**

(*конфликт типов*)

В этом выражении существует конфликт типов, например, попытка присвоить структуру простому типу

type specifier reqd. for proto arg **P1**

(*для параметра прототипа требуется спецификатор типа*)

Параметр прототипа должен иметь основной тип

undefined control **CPP**

(*не определено управление*)

Проверьте использование **#**

undefined enum tag **P1**

(*не определен тег enum*)

Этот тег перечислимого типа не был определен

undefined identifier **P1**

(*не определен идентификатор*)

Этот идентификатор не был определен перед использованием

undefined struct/union **P1**

(*не определена структура/объединение*)

Используемая структура или объединение не были определены

undefined symbol **LINK**

(*символ не определен*)

Ниже представлен список неопределенных символов. Если некоторые из символов, должны быть в связанной библиотеке, это может быть вызвано проблемами в упорядочивании библиотеки. В этом случае перестройте библиотеку с корректным упорядочиванием или определите библиотеку несколько раз в команде редактора связей

unexpected EOF **P1**

(*неожиданный EOF*)

Конец файла встретился в середине конструкции C. Обычно, это вызвано отсутствием закрывающей '}' ранее в программе.

Unknown predicate **CGEN**

(*неизвестный предикат*)

Внутренняя ошибка - обратитесь в HI-TECH

unknown psect **LINK**

(*неизвестная psect*)

В программе отсутствует psect заданная параметром `-p`. Проверьте правильность написания и проверьте режим ввода - верхний регистр не соответствует нижнему регистру

unreachable code **P1**

(*недостижимый код*)

Этот участок кода никогда не может быть выполнен, поскольку отсутствуют возможные пути доступа к нему

Unreasonable include nesting **CPP**

(*чрезмерная вложенность include*)

Сократите количество `include` файлов

Unreasonable matching depth **CGEN**

(*чрезмерная глубина соответствия*)

Внутренняя ошибка - обратитесь в HI-TECH

unterminated macro call **CPP**

(*незавершенный макровызов*)

Возможно не хватает)

void function cannot return value **P1**

(*функция void не может возвращать значение*)

Функция объявленная `void` не может возвращать значение

Write error (out of disk space?) **LINK**

(*Ошибка записи (недостаточно места на диске?)*)

Вероятно означает, что диск заполнен

Приложение 2 Функции стандартной библиотеки

Функции доступные пользовательским программам в стандартной библиотеке `libc.lib` перечислены ниже по категориям с кратким комментарием, затем в алфавитном порядке с более подробным описанием. Раздел *Синтаксис* в подробном описании каждой функции приводит примерный вид написания функции в исходном файле, определяющем ее. Если приведен `include` файл, подразумевается, что этот `include` файл, должен быть включен в любой исходный файл, использующий эту функцию.

В случае когда включаемый файл не указан, обычно необходимо включить объявление функции `extern` в любой модуль источника использующего ее, чтобы обеспечить правильный тип функции. Например, если используется функция `lseek()`, форма объявления

```
extern long lseek();
```

должна присутствовать в самом исходном файле или в `include` файле, включенном в исходный файл. Это гарантирует, оповещение компилятора о том, что `lseek()` возвращает длинное значение `long`, а не `int` по умолчанию.

Там, где делается ссылка на `stdio`, это означает группу функций перечисленных под заголовком Стандартный ввод-вывод ниже. У них всех есть одна общая черта. Они работают с указателями на определенный тип данных под названием `FILE`. Такой указатель часто упоминается как потоковый указатель. Понятие потока имеет важнейшее значение для этих подпрограмм. По сути, поток является источником или приемником байтов данных. Для операционной системы и библиотечных подпрограмм этот поток безликий, т.е. не подразумевающий и не предполагающий любую структуру записей. Однако, некоторые подпрограммы тем не менее признают символы конца строки.

Стандартный ввод-вывод

`fopen(name, mode)`	Открытие файла для ввода-вывода
`freopen(name, mode, stream)`	Повторное открытие существующего потока
`fdopen(fd, mode)`	Привязка потока к дескриптору файла
`fclose(stream)`	Закрытие открытого файла
`fflush(stream)`	Очистка буфера данных
`getc(stream)`	Чтение байта из потока
`fgetc(stream)`	Аналогична `getc()`
`ungetc(c, stream)`	Возврат символа назад в поток
`putc(c, stream)`	Запись байта в поток
`fputc(c, stream)`	Аналогична `putc()`
`getchar()`	Чтение байта из стандартного ввода
`putchar(c)`	Запись байта в стандартный вывод
`getw(stream)`	Чтение слова из потока
`putw(w, stream)`	Вывод слова в поток
`gets(s)`	Чтение строки из стандартного ввода
`fgets(s, n, stream)`	Чтение строки из потока
`puts(s)`	Вывод строки на стандартный вывод
`fputs(s, stream)`	Вывод строки в поток
`fread(buf, size, cnt, stream)`	Двоичное чтение из потока
`fwrite(buf, size, cnt, stream)`	Двоичная запись в поток
`fseek(stream, offs, wh)`	Позиционирование произвольного доступа
`ftell(stream)`	Текущая позиция чтения/записи в файле
`rewind(stream)`	Перемещение файлового указателя в начало

`setvbuf(stream, buf, mode, size)`	Включение/отключение буферизации потока
`fprintf(stream, fmt, args)`	Форматированный вывод в поток
`printf(fmt, args)`	Форматипованный стандартный вывод
`sprintf(buf, fmt, args)`	Форматированный вывод в строку
`vfprintf(stream, fmt, va_ptr)`	Форматированный вывод в поток
`vprintf(fmt, va_ptr)`	Форматированный стандартный вывод
`vsprintf(buf, fmt, va_ptr)`	Форматированный вывод в строку
`fscanf(stream, fmt, args)`	Форматированный ввод из потока
`scanf(fmt, args)`	Форматированный стандартный ввод
`sscanf(buf, fmt, va_ptr)`	Форматированный ввод из сторки
`vfscanf(stream, fmt, va_ptr)`	Форматированный ввод из потока
`vscanf(fmt, args)`	Форматированный стандартный вывод
`vsscanf(buf, fmt, va_ptr)`	Форматированный ввод из сторки
`feof(stream)`	Проверка наличия конца файла
`ferror(stream)`	Определение наличия ошибок в потоке
`clrerr(stream)`	Сброс статуса ощибок в потоке
`fileno(stream)`	Определение дескриптора файла потока
`remove(name)`	Удаление файла

Обработка строк

`atoi(s)`	Преобразование десятичного ASCII в целое
`atol(s)`	Преобразование десятичного ASCII в длинное целое
`atof(s)`	Преобразование десятичного ASCII в действительное
`xtoi(s)`	Преобразование шестнадцатеричного ASCII в целое
`memchr(s, c, n)`	Поиск символа в блоке памяти
`memcmp(s1, s2, n)`	Сравнение `n` байт в памяти
`memcpy(s1, s2, n)`	Копирование `n` байт из `s2` в `s1`
`memmove(s1, s2, n)`	Копирование `n` байт из `s2` в `s1`
`memset(s, c, n)`	Присвоение `n` байтам начиная с `s` значения `c`
`strcat(s1, s2)`	Добавление строки 2 к строке 1
`strncat(s1, s2, n)`	Добавление более `n` символов к строке 1
`strcmp(s1, s2)`	Сравнение строк
`strncmp(s1, s2, n)`	Сравнение `n` байт строк
`strcpy(s1, s2)`	Копирование `s2` в `s1`
`strncpy(s1, s2, n)`	Копирование `n` символов из строки `s2` в строку `s1`
`strerror(errnum)`	Формирование сообщения об ошибке по коду ошибки
`strlen(s)`	Длинна строки
`strchr(s, c)`	Поиск символа в строке
`strrchr(s, c)`	Поиск самого правого символа в строке
`strspn(s1, s2)`	Длина начального участка строки s1, содержащей только символы строки s2
`strcspn(s1, s2)`	Длина первой части строки s1 не содержащей никаких символов строки s2
`strstr(s1, s2)`	Поиск первого вхождение `s2` в `s1`
`open(name, mode)`	Открытие файла
`close(fd)`	Закрытие файла
`creat(name)`	Создание файла
`dup(fd)`	Дублирование дескриптора файла
`lseek(fd, offs, wh)`	Позиционирование произвольного доступа
`read(fd, buf, cnt)`	Чтение из файла
`rename(name1, name2)`	Переименование файла
`unlink(name)`	Удаление файла из каталога
`write(fd, buf, cnt)`	Запись в файл

`isatty(fd)`	Проверка символьного устройства
`stat(name, buf)`	Запросм информации о файле
`chmod(name, mode)`	Установка атрибутов файла

Проверка символов

`isalpha(c)`	Истина, если `c` является буквой алфавита
`isupper(c)`	Буква в верхнем регистре
`islower(c)`	Буква в нижнем регтстре
`isdigit(c)`	Цифра
`isalnum(c)`	Буквенноцифровой символ
`isspace(c)`	Пробел, табуляция, новая строка, возврат или прогон листа
`ispunct(c)`	Символ пунктуации
`isprint(c)`	Пачатаемый символ
`isgraph(c)`	Печатаемый символ, не являющийся пробелом
`iscntrl(c)`	Управляющий символ
`isascii(c)`	Символ ASCII (`0`-`127`)

С плавающей точкой

`cos(f)`	Функция косинус
`sin(f)`	Функция синус
`tan(f)`	Функция тангенс
`acos(f)`	Функция арккосинус
`asin(f)`	Функция арксинус
`atan(f)`	Функция арктангенс
`exp(f)`	Экспонента числа `f`
`log(f)`	Натуральный логарифм `f`
`log10(f)`	Десятичный логарифм `f`
`pow(x,y)`	Возведение `x` в степень `y`
`sqrt(f)`	Квадратный корень
`fabs(f)`	Абсолютное значение числа с плавающей точкой
`ceil(f)`	Наименьшее целое значение >= `f`
`floor(f)`	Наибольшее целое значение <= `f`
`sinh(f)`	Гиперболический синус
`cosh(f)`	Гиперболический косинус
`tanh(f)`	Гиперболический тангенс
`frexp(y, p)`	Разделить на мантиссу и экспоненту
`ldexp(y, i)`	Загрузить новую экспоненту

Консольный ввод-вывод

`getch()`	Получить один символ
`getche()`	Получить один символ без эха
`putch(c)`	Поместить один символ
`ungetch(c)`	Вставтавить символ назад
`kbhit()`	Проверить нажатие клавиши
`cgets(s)`	Получить строку с консоли
`cputs(s)`	Послать строку на консоль

Функции даты и времени

`time(p)`	Получить текущую дату/время
`gmtime(p)`	Получить всемирное время в развернутом структурном представлении
`localtime(p)`	Получить локальное время в развернутом структурном представлении

asctime(t)	Преобразовать развернутое структурное представление времени в строку ASCII
ctime(p)	Преобразовать время в строку ASCII

Прочие

execl(name, args)	Выполнить другую программу
execv(name, argp)	Выполнить другую программу
spawnl(name, arg, ...)	Выполнить подпрограмму
spawnv(name, argp)	Выполнить подпрограмму
system(s)	Выполнить системную команду
atexit(func)	Устанавливает func, выполняющуюся при завершении
exit(status)	Прервать выполнение
_exit(status)	Прервать выполнение немедленно
getuid()	Получить идентификатор пользователя (CP/M)
setuid(uid)	Установить идентификатор пользователя (CP/M)
chdir(s)	Сменить каталог (MS-DOS)
mkdir(s)	Создать каталог (MS-DOS)
rmdir(s)	Удалить каталог (MS-DOS)
getcwd(drive)	Получить текущий рабочий каталог (MS-DOS)
signal(sig, func)	Установить ловушку для состояния прерывания
brk(addr)	Установка выделения памяти
sbrk(incr)	Настройка выделения памяти
malloc(cnt)	Выделение динамической памяти
free(ptr)	Освобождение динамической памяти
realloc(ptr, cnt)	Перераспределение динамической памяти
calloc(cnt, size)	Обнуление динамической памяти
perror(s)	Печать сообщения об ошибке
qsort(base, nel, width, func)	Быстрая сортировка
srand(seed)	Инициализация генератора случайных чисел
rand()	Получить следующее случайное число
setjmp(buf)	Установка дальнего перехода
longjmp(buf, val)	Дальний переход
_getargs(buf, name)	Подстановка символов и перенаправление ввода-вывода
inp(port)	Чтение из порта
outp(port, data)	Запись данных в порт
bdos(func, val)	Выполнить вызов BDOS (CP/M)
msdos(func, val, val,...)	Выполнить вызов MS-DOS
msdoscx(func, val, val,...)	Альтернативный вызов MS-DOS
intdos(ip, op)	Выполнить прерывание DOS
intdosx(ip, op, sp)	Выполнить прерывание DOS
segread(sp)	Получить значения сегментных регистров
int86(int, ip, op)	Выпонить программное прерывание
int86x(int, ip, op, sp)	Выпонить программное прерывание
bios(n, c)	Вызвать точку входа BIOS (CP/M)
ei()	Разрешить прерывания
di()	Запретить прерывания
set_vector(vec, func)	Установить вектор прерывания
assert(e)	Подтверждение отсутствия ошибок
getenv(s)	Получить строку среды (MS-DOS)

ACOS, **ASIN**, **ATAN**, **ATAN2**

Синтаксис:

```
#include <math.h>

double acos(double f)
double asin(double f)
double atan(double f)

double atan2(double x, double y)
```

Описание:

Эти функции являются обратными тригонометрическими функциями `cos`, `sin` и `tan`. Функции `acos` и `asin` не определены для аргументов, абсолютное значение которых больше 1.0. Возвращаемое значение в радианах, и всегда в диапазоне от - π/2 до +π/2, за исключением `cos()`, которая возвращает значение в диапазоне от `0` до π. Функция `atan2()` возвращает обратное значение `tan x/y`, но использует знаки его аргументов для возвращаемого значения в диапазоне от -π до +π.

См. также

`sin`, `cos`, `tan`

ATEXIT

Синтаксис:

```
#include <stdlib.h>

int atexit(void (*func)(void));
```

Описание:

Функция `atexit()` регистрирует функцию, на которую указывает **`func`** для вызова без параметров при нормальном завершении программы. Функция `atexit()` возвращает ноль, если регистрация произошла успешно, ненулевое значение в случае сбоя. При завершении работы программы все функции, зарегистрированные `atexit()` вызываются в порядке обратном их регистрации.

См. также

`exit`

ASCTIME

Синтаксис:

```
#include <time.h>
char *asctime(time_t t)
```

Описание:

Функция `asctime()` принимает развернутое структурное представление времени, на которое указывает ее параметр **`t`**, и возвращает строку из 26 символов, описывающих текущую дату и время в формате

```
Sun Sep 16 01:03:52 1973\n\0
```

Обратите внимание на символ новой строки в конце строки. Ширина каждого поля в строке фиксирована.

См. также

`ctime`, `time`, `gmtime`, `localtime`

ASSERT

Синтаксис:

```
#include <assert.h>

void assert(int e)
```

Описание:

Этот макрос используется для отладки. Основной метод использования заключается в размещении утверждений свободно на протяжении всего кода в точках, где правильная работа кода изначально зависит от определенных условий истинности. Макрос `assert()` может использоваться, чтобы гарантировать во время выполнения, что это предположение выполняется. Например, следующий оператор утверждает, что указатель `tp` не равен `NULL`:

```
assert(tp);
```

Если во время выполнения выражение принимает значение `false`, то программа будет прервана с сообщением, идентифицирующим исходный файл и номер строки с макросом, и выражение, используемое в качестве его параметра. Более подробное обсуждение использования `assert`, невозможно в ограниченном объеме, но оно тесно связано с методами доказательства правильности программ.

ATOF, **ATOI**, **ATOL**

Синтаксис:

```
#include <math.h>

double atof(char *s)
int    atoi(char *s)

#include <stdlib.h>

long   atol(char *s)
```

Описание:

Эти подпрограммы преобразуют десятичное число в строке параметра **s** в вещественное значение двойной точности, целое или длинное целое соответственно. Начальные пробелы пропускаются. В случае `atof()`, число может быть в экспоненциальном представлении.

BDOS (только CP/M)

Синтаксис:

```
#include  <cpm.h>

char  bdos(int func, int arg)
short bdoshl(int func, int arg)  (только CP/M-80)
```

Описание:

Функция `bdos()` вызывает BDOS CP/M с параметрами **func** в регистре C (CL для CP/M-86) и **arg** в регистре DE (DX). Возвращаемое значение - байт, возвращенный BDOS в регистре A (AX). Функция `bdoshl()` аналогичная, за исключением того, что возвращаемое значение является значением, возвращаемым BDOS в HL. Значения констант для различных функций BDOS определены в файле `cpm.h`.

Следует избегать использования этих функций кроме программ, которые предназначены для операционной системы CP/M. Стандартные подпрограммы ввода-вывода более предпочтительны, поскольку они переносимы.

См. также

bios, msdos

BIOS (только CP/M)

Синтаксис:

```
#includ <cpm.h>

char bios(int n, int a1, int a2)
```

Описание:

Эта функция вызовет **n** -ю точку входа BIOS ("холодная" загрузка = 0, "теплая" загрузка = 1, и т.д.) с регистром BC (CX) установленным равным параметру **a1** и DE (DX) установленным равным параметру **a2**. Возвращаемое значение - содержимое регистра A (AX) после вызова BIOS. В CP/M-86 используется функция bdos 50 для выполнения вызова BIOS. Эта функция не должна использоваться, если возможно, так как она непереносима. Нет даже никакой гарантии переносимости вызовов bios() между различными системами CP/M.

См. также

bdos

CALLOC

Синтаксис:

```
#include <stdlib.h>

char *calloc(size_t cnt, size_t size)
```

Описание:

Функция calloc() пытается получить непрерывный блок динамической памяти, которая вместит **cnt** объектов, каждый длиной **size**. Блок заполняется нулями. Возвращается указатель на блок, или 0, если не удалось выделить память.

См. также

brk, sbrk, malloc, free

CGETS, **CPUTS**

Синтаксис:

```
#include <conio.h>

char *cgets(char *s)
void cputs(char *s)
```

Описание:

Функция cputs() будет читать, одну строку ввода из консоли в буфер, переданный ей в качестве параметра. Она делает это путем повторных вызовов getche(). Функция cputs() записывает строку своего параметра в консоль, выводя символы возврата каретки перед каждой новой строкой в строке. Она неоднократно вызывает putch().

См. также

getch, getche, putch

CHDIR

Синтаксис:

```
#include <sys.h>

int chdir(char *s)
```

Описание:

Эта функция доступна только в MS-DOS. Она изменяет текущий рабочий каталог на путь, предоставленный в качестве параметра. Это имя пути может быть абсолютным, как в A:\FRED или относительным, как ..\SOURCES. Возвращаемое значение -1 указывает, что запрашиваемое изменение не может быть выполнено.

См. также

mkdir, rmdir, getcwd

CHMOD

Синтаксис:

```
#include <stat.h>

int chmod(char *name, int)
char *name; int mode;
```

Описание:

Эта функция изменяет атрибуты файла (или режимы) названного файла. Параметр **name** может иметь любое допустимое имя файла. Параметр **mode** может включать все биты, определенные в файле stat.h кроме тех, которые касаются расширения файла, например, S_IFDIR. Однако, обратите внимание, что не все биты могут быть изменены во всех операционных системах, например, ни DOS, ни CP/M не разрешают сделать файлы нечитаемыми, таким образом, даже если режим не включает S_IREAD, файл все равно будет читаем (и функция stat() по-прежнему будет возвращать флаг S_IREAD).

См. также

stat, creat

CLOSE

Синтаксис:

```
#include <unixio.h>

int close(int fd)
```

Описание:

Эта подпрограмма закрывает файл, связанный с дескриптором файла **fd**, который был ранее получен из вызова open(). Функция close() возвращает 0 при успешном завершении, или -1 в противном случае.

См. также

open, read, write, seek

CLRERR, **CLREOF**

Синтаксис:

```
#include <stdio.h>

void clrerr(FILE * stream)
void clreof(FILE * stream)
```

Описание:

Это макросы, определенные в файле `stdio.h`, сбрасывают для указанного потока **stream** флаги ошибки и конца файла соответственно. Они должны использоваться с осторожностью. Основное допустимое использование для очистки состояния `EOF` при вводе из терминального устройства, где допустимо, продолжать чтение после обнаружения конца файла.

См. также

`fopen`, `fclose`

COS

Синтаксис:

```
#include <math.h>
double cos(double f)
```

Описание:

Эта функция вычисляет косинус аргумента.

См. также

`sin`, `tan`, `asin`, `acos`, `atan`

COSH, **SINH**, **TANH**

Синтаксис:

```
#include <math.h>
double cosh(double f)
double sinh(double f)
double tanh(double f)
```

Описание:

Эти функции реализуют гиперболические тригонометрические функции.

CREAT

Синтаксис:

```
#include <stat.h>
int creat(char *name, int mode)
```

Описание:

Эта подпрограмма пытается создать файл с именем **name**. Если файл существует и доступен для записи, он будет удален и создан заново. Возвращаемое значение равно `-1`, если создать не удалось, или небольшое неотрицательное число в случае успеха. Это число является ценным маркером, используемым впоследствии при записи в файл или закрытии файла. Параметр **mode** используется для инициализации атрибутов создаваемого файла. Допустимые биты совпадают с используемыми функцией `chmod()`, но для совместимости с `Unix` рекомендуется использовать `mode 0666` или `0600`. В `CP/M` параметр `mode` игно-

рируется - единственным способом установить атрибуты файлов является использование функции chmod().

См. также

open, close, read, write, seek, stat, chmod

CTIME

Синтаксис:

```
#include <time.h>
char *ctime(time_t t)
```

Описание:

Функция ctime() преобразует время в секундах, указанное в параметре **t** в строку той же формы, которая описана для asctime. Таким образом, следующая программа распечатает текущие время и дату:

```
#include <time.h>

main() {
    time_t t;

    time(&t);
    pnntf("%s", ctime(&t));
}
```

См. также

gmtime, localtime, asctime, time

DIV, LDIV

Синтаксис:

```
#include <stdlib.h>
div_t  div(int numer, int denom)
ldiv_t ldiv(long numer, long denom)
```

Описание:

Функция div() вычисляет частное и остаток от деления **numer** на **denom**. Функция div() возвращает структуру типа div_t, содержащую частное и остаток. Функция ldiv() аналогична div() за исключением того, что принимает аргументы типа long и возвращает структуру типа ldiv_t. Типы div_t и ldiv_t определены в файле <stdlib.h> следующим образом:

```
typedef struct {
    int quot, rem;
} div_t;

typedef struct {
    long quot, rem;
} ldiv_t;
```

DI, EI

Синтаксис:

```
void ei(void);
void di(void);
```

Описание:

Функции ei() и di() включают и отключают прерывания соответственно.

DUP

Синтаксис:

```
#include <unixio.h>

int dup(int fd)
```

Описание:

Учитывая дескриптор файла, такой как возвращенный `open()`, эта подпрограмма возвращает другой файловый дескриптор, относящийся к тому же открытому файлу. Возвращает значение -1, если параметр **`fd`** является некорректным дескриптором или не относится к открытому файлу.

См. также

`open`, `close`, `creat`, `read`, `write`

EXECL, **EXECV**

Синтаксис:

```
#include <sys.h>

int execl(char *name, char *pname, ...)
int execv(char *name, char **ppname)
```

Описание:

Функции `execl()` и `execv()` загружают и выполняют программу, определенную строкой **`name`**. Функция `execl()` принимает параметры для программы из завершенного нулем списка параметров в строке. Функции `execv()` передается указатель на массив строк. Массив должен заканчиваться нулем. Если указанная программа найдена и может быть считана, вызов не возвращается. Таким образом, любой возврат из этих подпрограмм может рассматриваться как ошибка.

См. также

`spawnl`, `spawnv`, `system`

EXIT

Синтаксис:

```
#include <stdlib.h>

void exit(int status)
```

Описание:

Этот вызов закроет все открытые файлы и выйдет из программы. В `CP/M` это значит возврат в уровень `CCP`. Статус сохраняется в фиксированном месте для исследования другими программами. Это полезно только если выполнение программы было фактически вызвано другой программой, которая перехватывает "теплую" загрузку. В `CP/M` значение статуса сохраняется по адресу `80H`. Этот вызов никогда не вернется.

См. также

`atexit`

_EXIT

Синтаксис:

```
#include <stdlib.h>

void _exit(int status)
```

Описание:

Эта функция вызовет немедленный выход из программы без нормального сброса буферов `stdio`, который выполняется `exit()`.

См. также

`exit`

EXP, **LOG**, **LOG10**, **POW**

Синтаксис:

```
#include <math.h>

double exp(double f)
double log(double f)
double log10(double f)
double pow(double x, y)
```

Описание:

Функция `exp()` возвращает значение экспоненты своего аргумента, `log()` вычисляет натуральный логарифм **f** и `log10()` - десятичный логарифм. Функция `pow()` возвращает значение **x** возведенного в степень **y**.

FABS, **CEIL**, **FLOOR**

Синтаксис:

```
#include <math.h>

double fabs(double f)
double ceil(double f)
double floor(double f)
```

Описание:

Эти подпрограммы возвращают соответственно абсолютное значение **f**, наименьшее целое значение не меньше чем `f` и наибольшее целое значение не большее чем `f`.

FCLOSE

Синтаксис:

```
#include <stdio.h>

int fclose(FILE *stream)
```

Описание:

Эта подпрограмма закрывает указанный поток ввода-вывода **stream**. Поток должен быть маркером, возвращенным предыдущим вызовом `fopen()`. При успешном завершении возвращается `NULL`, иначе `EOF`.

См. также

`fopen`, `fread`, `fwrite`

FEOF, **FERROR**

Синтаксис:

```
#include <stdio.h>

feof(FILE *stream)
ferror(FILE *stream)
```

Описание:

Эти макросы проверяют состояние битов EOF и ERROR соответственно для указанного потока **stream**. Каждый будет истиной, если соответствующий флаг установлен. Макросы определены в файле stdio.h. Поток должен быть маркером, возвращенным предыдущим вызовом fopen().

См. также

fopen, fclose

FFLUSH

Синтаксис:

```
#include <stdio.h>

int fflush(FILE *stream)
```

Описание:

Функция fflush() выведет в файл на диске, или другое открытое в настоящее время устройство в указанном потоке **stream** содержимое связанного с ним буфера. Она обычно используется для сбрасывания буферизированного стандартного вывода в интерактивных приложениях.

См. также

fopen, fclose

FGETC

Синтаксис:

```
#include <stdio.h>

int fgetc(FILE *stream)
```

Описание:

Функция fgetc() возвращает следующий символ из входного потока **stream**. Если встречается конец файла, то вместо него будет возвращен EOF. Именно по этой причине функция объявлена как int. Целочисленный EOF не является допустимым байтом, таким образом, конец файла отличается от считанного байта из файла, все биты которого равны 1. Функция fgetc()не является макро версией getc().

См. также

fopen, fclose, fputc, getc, putc

FGETS

Синтаксис:

```
#include <stdio.h>
char *fgets(char *s, size_t n, char *stream)
```

Описание:

Функция `fgets()` помещает в буфере **s** до **n**-1 символов из входного потока **stream**. Если во вводе появится символ новой строки, прежде чем будет считано корректное число символов, то сразу произойдет возврат из `fgets()`. Символ новой строки останется в буфере. В любом случае буфер будет завершен нулем. Успешный вызов `fgets()` возвратит свой первый параметр. При достижении конца файла или ошибке возвращается `NULL`.

FILENO

Синтаксис:

```
fileno(FILE *stream)
```

Описание:

`fileno()` является макросом из файла `stdio.h`, который возвращает дескриптор файла, связанный с потоком **stream**. Обычно, он используется, когда необходимо выполнить некоторую низкоуровневую операцию с файлом, открытым как поток `stdio`.

См. также

`fopen`, `fclose`, `open`, `close`

FOPEN

Синтаксис:

```
#include <stdio.h>
FILE *fopen(char *name, char *mode);
```

Описание:

Функция `fopen()`пытается открыть файл для чтения или записи (или того и другого) согласно предоставленной строки **mode**. Строка режима интерпретируется следующим образом:

r Файл открывается для чтения, если он существует. Если файл не существует, то вызов завершится с ошибкой.

r+ Если файл существует, он открывается для чтения и записи. Если файл не существует, то вызов завершится с ошибкой.

w Файл создается, если он не существует, или усекается, если он есть. Затем он открывается для записи.

w+ Файл создается, если он не существует, или усекается, если он есть. Файл открывается для чтения и записи.

a Файл создается, если он еще не существует, и открывается для записи. Все записи будут динамично принудительно приводить в конец файла, таким образом этот режим известен, как режим добавления.

a+ Файл создается, если он еще не существует, и открывается для чтения и записи. Все записи в файл будут динамично приводить к концу файла, т.е. в то время как любая часть файла может быть считана, все записи будут располагаться в конце файла, и не перезапишут существующих данных. Вызов `fseek()` в попытке произвести запись в файл в любом другом месте не будет эффективным.

Модификатор "b" может быть добавлен к любому из вышеупомянутых режимов, например, "r+b" или "rb+" эквивалентны. Добавление модификатора "b" заставит файл быть открытым в двоичном, а не в ASCII режиме. Открытие в режиме ASCII гарантирует, что текстовые файлы считаны способом, совместимым с основанными на Unix соглашениями для программ C, то есть, что текстовые файлы содержат строки, разделенные символами новой строки. Специальный режим чтения или записи символов меняется в зависимости от операционной системы, но включает в себя некоторые или все из следующих действий:

NEWLINE (перевод строки), преобразуется в возврат каретки, перевод строки при выводе.

RETURN игнорируется при вводе, вставляется перед NEWLINE при выводе.

Ctrl-Z Обозначает EOF при вводе. Добавляется fclose при выводе при необходимости в CP/M.

Открытие файла в двоичном режиме позволяет каждый символ прочитать как он записан, но так как точный размер файла не известен CP/M, файл может содержать больше байтов, чем было записано в него. Смотрите open() для описание того, что является именем файла.

При использовании одного из режимов чтения/записи (с символом '+' в строке), несмотря на то, что они разрешают читать и писать в тот же поток, невозможно произвольно смешивать вызовы ввода и вывода в один и тот же поток. В любой момент времени поток, открытый в режиме "+", будет в состоянии ввода или вывода. Состояние может быть изменено, только когда связанный с ним буфер пуст, что гарантируется только сразу после вызова fflush() или одной из функций позиционирования файла fseek() или rewind(). Буфер также будет пуст после обнаружения с EOF при чтении двоичного потока, но рекомендуется, использовать явный вызов, fflush(), чтобы гарантировать эту ситуацию. Таким образом, после чтения из потока вы должны вызвать fflush() или fseek() прежде, чем попытаться записать в этот поток, и наоборот.

См. также

fclose, fgetc, fputc, freopen

FPRINTF

Синтаксис:

```
#include <stdio.h>

fprintf(FILE *stream, char *fmt, ...);
vfprintf(FILE *stream, va_list va_arg);
```

Описание:

Функция fprintf() выполняет отформатированную печать в указанный поток **stream**. Обратитесь к printf() для получения подробной информации о доступных форматах. Функция vfprintf() похожа на fprintf(), но принимает указатель переменного списка аргументов, а не список аргументов. См. описание va_start() для получения дополнительной информации о переменных списках аргументов.

См. также

printf, fscanf, sscanf

FPUTC

Синтаксис:

```
#include <stdio.h>
int fputc(int c, FILE *stream)
```

Описание:

Символ **c** записывается в предоставленный поток **stream**. Это не макро версия putc(). Символ возвращается, если он был успешно записан, в противном случае возвращается EOF. Обратите внимание, что "записать в поток" может означать только, поместить символ в буфер, связанный с потоком.

См. также

putc, fgetc, fopen, fflush

FPUTS

Синтаксис:

```
#include <stdio.h>
int fputs(char *s, FILE *stream)
```

Описание:

Завершенная нулем строка **s** записывается в поток **stream**. Символ новой строки не добавляется (см. puts()). В случае ошибки возвращается EOF.

См. также

puts, fgets, fopen, fclose

FREAD

Синтаксис:

```
#include <stdio.h>
int fread(void *buf, size_t size, size_t cnt, FILE *stream)
```

Описание:

До **cnt** объектов, каждый длиной **size**, читается в память **buf** из потока **stream**. Возвращаемое значение - число считанных объектов. Если ни один не будет считан, то будет возвращен 0. Обратите внимание, что возвращаемое значение меньше cnt, но больше 0, может не представлять ошибку (см. fwrite()). Никакого выравнивание по границе слова в потоке не предполагается или необходимо. Чтение выполняется через последовательныевызовы getc().

См. также

fwrite, fopen, fclose, getc

FREE

Синтаксис:

```
#include <stdlib.h>
void free(void *ptr)
```

Описание:

Функция free() освобождает блок памяти с адресом **ptr**, который ранее должен был быть получен с помощью вызова malloc() или calloc().

См. также

malloc, calloc

FREOPEN

Синтаксис:

```
#include <stdio.h>

FILE *freopen(char *name, char *mode, FILE *stream)
```

Описание:

Функция `freopen()` закрывает заданный поток **`stream`** (если он открыт), а затем вновь открывает поток, прикрепленный к файлу, описанный **`name`**. Режим открытия задается **`mode`**. Она либо возвращает параметр `stream`, в случае успеха, или `NULL`, в противном случае. См `fopen()` для получения дополнительной информации.

См. также

`fopen`, `fclose`

FREXP, **LDEXP**

Синтаксис:

```
#include <math.h>

double frexp(double f, int *p)
double ldexp(double f, int i)
```

Описание:

Функция `frexp()` раскладывает число с плавающей точкой на нормализованную дробную часть и целочисленную степень двойки. Целое число сохраняется в целочисленный объект, на который указывает **`p`**. Возвращаемое ей значение `x` находится в интервале (`0.5`, `1.0`) или ноль, и **`f`** равняется значению `x` умноженному на `2` и возведенному в степень, хранящуюся в `*p`. Если `f` - ноль, обе части результата равны нулю. `ldexp()` выполняет обратное действие. Целое число `i` добавляется к экспоненте плавающей точки `f` и возвращается результирующее значение.

FSCANF

Синтаксис:

```
#include <stdio.h>

int fscanf(FILE *stream, char *fmt, ...)
```

Описание:

Эта подпрограмма выполняет отформатированный ввод из указанного потока **`stream`**. См. `scanf()` для полного описания поведения подпрограммы. Функция `vfscanf()` подобна `fscanf()`, но принимает указатель на переменный список параметров, а не список параметров. См. описание `va_start()` для получения дополнительной информации о переменных списках параметров.

См. также

`scanf`, `sscanf`, `fopen`, `fclose`

FSEEK

Синтаксис:

```
#include <stdio.h>
int fseek(FILE *stream, long offs, int wh)
```

Описание:

Функция `fseek()` позиционирует "файловый указатель" (т.е. указатель на следующий символ, который будет считан или записан) указанного потока **`stream`** следующим образом:

wh	**Результирующее расположение**
0	`offs` начало файла
1	`offs+` текущая позиция
2	`offs+` конец файла

Нужно отметить, что **`offs`** это значение со знаком. Таким образом, 3 предусмотренные режима обеспечивают позиционирование относительно начала файла, текущего указателя файла и конца файла соответственно. Если запрос позиционирование не может быть удовлетворен, то возвращается `EOF`. Однако, обратите внимание, что позиционирование за пределы конца файла разрешено, но приведет к индикации `EOF`, если предпринимается попытка считать там данные. Вполне допустимо, записать данные вне предшествующего конца файла. Функция `fseek()` правильно обрабатывает любые буферизированные данные.

См. также

`lseek`, `fopen`, `fclose`

FTELL

Синтаксис:

```
#include <stdio.h>
long ftell(FILE *stream)
```

Описание:

Эта функция возвращает текущую позицию воображаемого указателя чтения/записи, связанного с потоком **`stream`**. Это позиция относительно начала файла следующего считываемого или записываемого в файл байта.

См. также

`fseek`

FWRITE

Синтаксис:

```
#include <stdio.h>
int fwrite(void *buf, size_t size, size_t cnt, FILE *stream)
```

Описание:

`cnt` объектов длиной **`size`** байтов будут записаны из памяти в **`buf`** из указанного потока **`stream`**. Возвращается целочисленное количество записанных объектов, или `0`, если ни один не мог быть записан. Любое возвращаемое значение, не равное `cnt`, должно рассматриваться как ошибка (см. `fread()`).

См. также

`firead`, `fopen`, `fclose`

_GETARGS

Синтаксис:

```
#include <sys.h>
char ** _getargs(char *buf, char *name)
extern int _argc_;
```

Описание:

Эта подпрограмма выполняет перенаправление ввода-вывода (только CP/M) и подстановочное расширение. В MS-DOS перенаправление ввода-вывода выполняется операционной системой. Она вызывается из кода запуска, чтобы воздействовать на командную строку, если для команды C используется параметр -R, но также может быть вызвана из кода, написанного пользователем. Если параметр **buf** равен нулю, она читает строки текста из стандартного ввода. Если стандартный ввод будет терминалом (обычно консоль), то параметр **name** будет записываться в стандартный поток ошибок как подсказка. Если параметр buf будет не ноль, то он будет использоваться в качестве источника строки для обработки. Возвращенное значение - указатель на массив строк, точно как был бы указан параметр argv функции main(). Число строк в массиве может быть получено из глобальной переменной **_argc_**. Пример обычного использования этой функции:

```
#include <sys.h>

main(argc, argv) char ** argv; {

    extern char ** _getargs();
    extern int     _argc_;

    if(argc == 1) {   /* параметры отсутствуют */
        argv = _getargs(0, "myname");
        argc = _argc_;
    }
    .
    .
    .
}
```

Для каждого слова в обработанном буфере будет одна строка в массиве. Для включения пробелов в слова могут использоваться одинарные (') или двойные (") кавычки. Если какие-либо подстановочные символы (? или *) появляются в слове без кавычек, оно будет расширено в строку слов, по одному для каждого файла, соответствующего слову. Для этого расширения действуют обычные соглашения CP/M. В CP/M любые вхождения символов перенаправления > и < вне кавычек обрабатываются следующим образом:

> name	стандартный вывод будет перенаправлен в файл с именем name.
< name	стандартный ввод будет перенаправлен из файл с именем name.
>> name	стандартный вывод будет добавляться в файл с именем name.

Между символами > или < и именем файла пробельные символы необязательны, однако символ перенаправления, за которым не следует имя файла, является ошибкой. Также возникает ошибка, если файл не может быть открыт для ввода или создан для вывода. Добавление перенаправления (>>) создаст файл, если он не существует. Если источником обрабатываемого текста является стандартный ввод, могут быть предоставлены несколько строк, заканчи-

вая каждую строку (кроме последней) символом обратной косой черты (\). Он служит символом продолжения. Обратите внимание, что символ новой строки после обратной косой черты игнорируется и не обрабатывается как пробел.

GETC

Синтаксис:

```
#include <stdio.h>
int getc(FILE *stream)
FILE *stream;
```

Описание:

Один символ считывается и возвращается из указанного потока **stream**. В конце файла или при ошибке возвращается EOF. Это - макро-версия fgetc() и определена в файле stdio.h.

GETCH, GETCHE, UNGETCH, PUTCH

Синтаксис:

```
#include <conio.h>

char  getch(void)
char  getche(void)
void  putch(int c)
```

Описание:

Функция getch() возвращает очередной символ, считанный с консоли, но не выводит его на экран. Функция getche() подобна, но выводит этот символ на экран. Функция ungetch() возвращает обратно один символ, таким образом, следующий вызов getch() или getche() возвратит этот символ. Функция putch() выводит символ **c** на экран консоли, предварительно добавив символ возврата каретки, если символ является новой строкой.

См. также

cgets, cputs

GETCHAR

Синтаксис:

```
#include <stdio.h>
int getchar(void)
```

Описание:

getchar() является операцией getc(stdin). Это макрос, определенный в файле stdio.h. Обратите внимание, что при нормальных обстоятельствах getchar() не возвратится, пока в консоли не будет введен возврат каретки. Чтобы немедленно получить единственный символ с консоли, используйте подпрограмму getch().

См. также

getc, fgetc, freopen, fclose

GETCWD (только MS-DOS)

Синтаксис:

```
#include <sys.h>

char *getcwd(int drive)
```

Описание:

getcwd() возвращает путь к текущему рабочему каталогу на указанном диске, где drive == 0 представляет текущий диск, drive == 1 представляет диск A:, drive == 2 представляет диск B: и т.д. Возвращаемое значение - указатель на статическую область памяти, которая будет перезаписана при следующем вызове getcwd().

См. также

chdir

GETENV

Синтаксис:

```
#include <stdlib.h>

char *getenv(char *s)
extern char **environ;
```

Описание:

Функция getenv() осуществляет поиск первого соответствия представленного параметра в таблице строк среды окружения и возвращает часть строки этой среды. Например, если среда содержит строку

```
COMSPEC=A:\COMMAND.COM
```

то getenv("COMSPEC") возвратит A:\COMMAND.COM. Глобальная переменная **environ** является указателем на массив указателей на строки среды, завершенный нулевым указателем. В MS-DOS этот массив инициализируется во время запуска из предоставленного указателя среды, во время выполнения программы. В CP/M такая среда не предусмотрена, поэтому, первый вызов getenv() попытается открыть файл в текущей области пользователя на текущем диске с названием ENVIRON. Этот файл должен содержать определения для любых переменных среды, которые должны быть доступны программе, например.

```
HITECH=0:C:
```

Определение каждой переменной должно быть на отдельной строке, состоящей из имени переменной (обычно все в верхнем регистре), затем без пробела следует знак равенства ('=') после чего значение, которое будет присвоено этой переменной.

GETS

Синтаксис:

```
#include <stdio.h>

char *gets(char *s)
```

Описание:

Функция gets() читает строку из стандартного ввода в буфер **s**, удаляя новую строку (см. fgets()). Буфер оканчивается нулевым символом. Она возвращает свой параметр или NULL в конце файла.

См. также

fgets, freopen

GETUID (только CP/M)

Синтаксис:

```
#include <sys.h>

int getuid(void)
```

Описание:

Функция getuid() возвращает текущий код пользователя. В CP/M текущий код пользователя определяет код пользователя, связанный с открытым или создаваемым файлом, если он не был переопределен явным указанием номера пользователя в префиксе имени файла.

См. также

setuid, open

GETW

Синтаксис:

```
#include <stdio.h>

int getw(FILE *stream)
```

Описание:

Функция getw() возвращает одно слово (16 битов для Z80 и 8086) из определенного потока **stream**. В конце файла возвращается EOF, но так как он является совершенно нормальным словом для тестирования конца файла, должен использоваться макрос feof(). При чтении слова никакое специальное выравнивание в файле не требуется, поскольку чтение выполняется двумя последовательными вызовами getc(). Однако, порядок байтов не определен. Как правило, читаемое слово должно быть записано с помощью putw().

См. также

putw, getc, fopen, fclose

GMTIME, **LOCALTIME**

Синтаксис:

```
#include <time.h>

struct tm *gmtime(time_t *t)
struct tm *localtime(time_t *t)
```

Описание:

Эти функции преобразуют время, на которое указывает параметр t, который задается в секундах, начиная с 00:00:00 1 января 1970 года, в развернутое структурное представление, сохраняемое в структуре, определенной в файле time.h. Функция gmtime(), выполняет прямое преобразование, в то время как localtime() учитывает содержимое глобальной целочисленной переменной time_zone (часовой пояс). Она должна содержать число минут, на которые местный часовой пояс расположен западнее Гринвича. Поскольку в MS-DOS фактически отсутствует способ заранее определить это значение, по умолчанию localtime() возвращает тот же результат, что и gmtime().

См. также

ctime, asctime, time

INP, **OUTP**

Синтаксис:

```
char inp(unsigned port)

void outp(unsigned, unsigned data)
```

Описание:

Эти подпрограммы читают и пишут байты в и из портов ввода-вывода. inp() возвращает байт данных, прочитанный из указанного порта и outp() выводит байт данных в указанный порт.

INT86, **INT86X**, **INTDOS**, **INTDOSX**

Синтаксис:

```
#include <dos.h>

int int86(int intno, union REGS *inregs, union REGS *outregs)
int int86x(int intno, union REGS inregs, union REGS outregs,
           struct SREGS *segregs)
int intdos(union REGS *inregs, union REGS *outregs)
int intdosx(union REGS *inregs, union REGS *outregs,
            struct SREGS *segregs)
```

Описание:

Эти функции позволяют вызывать программные прерывания из программ C. int86() и int86x() выполняют программное прерывание, определенное параметром **intno**, в то время как intdos() и intdosx() выполняют прерывание 21 (шестн.), которое является системным прерыванием MS-DOS. Указатель **inregs** должен указать на объединение, содержащее значения для каждого из регистров общего назначения, которые будут установлены при выполнении прерывания, и значения регистров при возврате копируются в объединение, на которое указывает **outregs**. Версии вызовов x также принимают указатель на объединение, определяющий значения сегментного регистра, кото-

рые будут установлены при выполнении прерывания, хотя фактически только ES и DS устанавливаются из этой структуры.

См. также

segread

ISALNUM, **ISALPHA**, **ISDIGIT**, **ISLOWER** и другие

Синтаксис:

```
#include <ctype.h>

isalnum(char c)
isalpha(char c)
isascii(char c)
iscntrl(char c)
isdigit(char c)
islower(char c)
isprint(char c)
isgraph(char c)
ispunct(char c)
isspace(char c)
isupper(char c)
char c;
```

Описание:

Эти макросы, определенные в файле ctype.h, проверяют предоставленный символ на принадлежность к одной из нескольких перекрывающихся групп символов. Обратите внимание, что все макросы кроме isascii определены для **c**, если isascii(c) является истиной.

isalnum(c)	c является алфавитно-цифровым
isalpha(c)	c является A-Z или a-z
isascii(c)	c является 7-битным символом ASCII
iscntrl(c)	c является управляющим символом
isdigit(c)	c является десятичной цифрой
islower(c)	c является a-z
isprint(c)	c является печатаемым символом
isgraph(c)	c является печатаемым символом, но не пробелом
ispunct(c)	c является знаком пунктуации
isspace(c)	c является пробелом, табуляцией или новой строкой
isupper(c)	c является A-Z
isxdigit(c)	c является 0-9 или a-f или A-F

См. также

toupper, tolower, toascii

ISATTY

Синтаксис:

```
#include <unixio.h>
int isatty(int fd)
```

Описание:

Тестирует тип файла, связанного с **fd**. Она возвращает true, если файл присоединен к терминальному устройству. Обычно она используется для тестирования осуществления стандартного ввода из файла или консоли. Для тестирования потоков stdio используйте isatty(fileno(stream)).

KBHIT

Синтаксис:

```
#include <conio.h>
int kbhit(void)
```

Описание:

Эта функция возвращает 1, если клавиша символа была нажата на клавиатуре консоли, 0 в противном случае. Обычно символ затем считывается с помощью getch().

См. также

getch, getche

LONGJMP

Синтаксис:

```
#include <setjmp.h>
void longjmp(jmp_buf buf, int val)
```

Описание:

Функция longjmp(), в сочетании с setjmp(), обеспечивает механизм для нелокальных переходов. Чтобы использовать это средство, нужно вызвать setjmp() с параметром jmp_buf на уровне некоторой внешней функции. Вызов из setjmp() возвращает 0. Чтобы вернуться к этому уровню выполнения, она может быть вызвана с тем же параметром jmp_buf из внутреннего уровня выполнения. Однако следует отметить, что функция, которая вызывается setjmp() должна быть активной во время вызова longjmp(). Нарушение этого правила вызовет аварийную ситуацию, из-за использования стека, содержащего недопустимые данные. Параметр **val** longjmp() будет значением, очевидно возвращенным из setjmp(). Оно должно обычно быть ненулевым, чтобы отличить его от подлинного вызова setjmp(). Например:

```
#include <setjmp.h>

static jmp_buf  jb_err;

main() {
  if(setjmp(jb_err)) {
    printf("An error occured0);
    exit(1);
  }
  a_func();
}
```

```
a_func() {
  if(do_whatever() != 0)       longjmp(jb_err, 1);
  if(do_something_else() != 0) longjmp(jb_err, 2);
}
```

Вызовы `longjmp()` выше никогда не вернется, а вызов `setjmp()` будет возвращаться, но с возвращаемым значением, равным параметру `longjmp()`.

См. также

`setjmp`

LSEEK

Синтаксис:

```
#include <unixio.h>

long lseek(int fd, long offs, int wh)
```

Описание:

Эта функция работает аналогично `fseek()`, однако она выполняет это с помощью не буферизированных низкоуровневых дескрипторов файлов ввода-вывода, а не потоков `stdio`. Она также возвращает результирующий указатель местоположения. Таким образом, `lseek(fd, 0L, 1)` возвращает текущее расположение указателя, не перемещая его. При ошибке возвращается `-1`.

См. также

`open`, `close`, `read`, `write`

MALLOC

Синтаксис:

```
#include <stdlib.h>

void *malloc(size_t cnt)
```

Описание:

Функция `malloc()` пытается выделить **`cnt`** байтов памяти из "кучи", динамически распределяемой области памяти. Если вызов успешный, она возвращает указатель на выделенный блок, иначе возвращается `0`. Выделенная память может быть освобождена с помощью `free()` или изменена в размере используя `realloc()`. Функция `malloc()` вызывает `sbrk()`, чтобы получить память, и она в свою очередь вызывает `calloc()`. Функция `malloc()` не очищает память, которую она получает.

См. также

`calloc`, `free`, `realloc`

MEMSET, **MEMCPY**, **MEMCMP**, **MEMMOVE**

Синтаксис:

```
#include <string.h>

void  memset(void s, char c, size_t n)
void *memcpy(void *d, void *s, size_t n)
int   memcmp(void *s1, void *s2, size_t n)
void *memmove(void *s1, void *s2, size_t n)
void *memchr(void *s, int c, size_t n)
```

Описание:

Функция `memset()` инициализирует **n** байтов памяти символом **c**, начинающейся в расположении, на которое указывает **s**. Функция `memcpy()`, копирует **n** байтов памяти, начинающейся в расположения, на которое указывает **s** в блок памяти, на которую указывает **d**. Результат копирования перекрывающихся блоков не определен. Функция `memcmp()` сравнивает два блока памяти, длиной **n**, и возвращает значение со знаком, аналогичное `strncmp()`. В отличие от `strncmp()` сравнение не останавливается на нулевом символе. Для сравнения используется упорядоченная последовательность `ASCII`, но эффект включения не `ASCII` символов в блоки памяти на результат возвращаемого значения неопределенен. Функция `memmove()` похожа на `memcpy()` за исключением копирования перекрывающихся блоков, которые обрабатываются правильно. Функция `memchr()` находит первое вхождение **c** (преобразованного в unsigned char) в первых **n** символах объекта, на который указывает **s**.

См. также

`strncpy`, `strncmp`, `strchr`

MKDIR, **RMDIR**

Синтаксис:

```
#include <sys.h>

int mkdir(char *s)
int rmdir(char *s)
```

Описание:

Эти функции позволяют создавать (`mkdir()`) и удалять (`rmdir()`) подкаталоги в операционной системе `MS-DOS`. Параметр **s** может быть произвольным путем, и возвращаемое значение равно `-1`, если создание или удаление были неудачны.

См. также

`chdir`

MSDOS, **MSDOSCX**

Синтаксис:

```
#include <dos.h>

long msdos(int ax, int dx, int ex, int bx, int si, int di)
long msdoscx(int ax, int dx, int ex, int bx, int si, int di)
```

Описание:

Эти функции предоставляют прямой доступ к системным вызовам MS-DOS. Параметры помещаются в регистры, подразумеваемые их именами, в то время как возвращаемое значение будет содержимым AX и DX (для `msdos()`) или содержимым DX и CX (для `msdoscx()`). Должно быть предоставлено только необходимое число параметров, например, если только AH и DX определяют величину, то требуются только 2 параметра. Следующая часть кода выводит символ прогона страницы на принтер.

```
msdos(0x500, '\f');
```

Обратите внимание, что номер системного вызова (в этом случае 5) должен быть умножен на `0x100`, так как MS-DOS ожидает номер вызова в AH, старшем байте AX.

См. также

intdos, intdosx, int86, int86x

OPEN

Синтаксис:

```
#include <unixio.h>

int open(char *name, int mode)
```

Описание:

Функция `open()` является основным средством открытия файлов для чтения и записи. Осуществляется поиск файла с указанным именем **name**, и если найден, он открывается для чтения, записи или того и другого. Режим **mode** кодируется следующим образом:

Режим	Значение
0	Открытие только для чтения
1	Открытие только для записи
2	Открытие для чтения и записи

Файл должен уже существовать - если это не так, должен использоваться вызов `creat()` для его создания. При успешном открытии возвращается дескриптор файла. Он является неотрицательным целым числом, которое может использоваться впоследствии для обращения к открытому файлу. Если открыть файл не удалось, возвращается `-1`. Синтаксис имени файла CP/M:

```
[uid:][drive:]name.type
```

где **uid** десятичное число от 0 до 15, **drive** - буква диска от A до P или от a до p, **name** - имя из 1-8 символов и **type** - расширение от 0 до 3 символов. Хотя существует несколько ограничений для символов, относящихся к имени и расширению файла, рекомендуется, чтобы они были ограничены буквенно-цифровыми и стандартными печатаемым символами. Использование странных символов может вызвать проблемы при доступе и/или удалении файла.

Один или оба uid: и drive: могут быть опущены. Если предоставлены оба, uid: должен быть на первом месте. Обратите внимание, что **[** и **]** являются только мета-символами. Некоторые примеры:

```
fred.dat
file.c
0:xyz.com
0:a:file1.p
a:file2.
```

Если опущен uid:, файл будет разыскиваться с uid, равным текущему номеру пользователя, которое возвращает getuid(). Если опущен drive:, файл будет разыскиваться на выбранном в настоящее время диске. Распознаются следующие специальные имена файлов:

lst: Получает доступ к устройству печати - только для записи
pun: Получает доступ к устройству перфорации - только для записи
rdr: Получает доступ к считывающему устройству - только для чтения
con: Получает доступ к системной консоли - чтение-запись

Имена файлов могут быть в любом регистре - они преобразуются в верхний регистр во время обработки имени.

Имена файлов MS-DOS могут быть любым допустимыми в MS-DOS 2xx именами файлов, например:

```
fred.nrk
A:\HITECH\STDIO.H
```

Специальные имена устройств (например, CON, LST) также распознаются. Они не требуют (и не должны иметь) завершающего двоеточия.

См. также

close, fopen, fclose, read, write, creat

PERROR

Синтаксис:

```
#include <stdio.h>

void perror(char *s)
```

Описание:

Эта подпрограмма распечатает в поток stderr, параметр **s**, затем содержательное сообщение с подробным описанием последней ошибки, возвращенной вызовами открытия, закрытия, чтения, или записи. К сожалению, CP/M не предоставляет исчерпывающую информацию, об ошибке, кроме случая произвольного чтения или записи. Поэтому эта подпрограмма имеет ограниченную ценность в CP/M. MS-DOS предоставляет намного больше информации, однако, и использования perror() после вызовов обработки файла MS-DOS, безусловно, обеспечит полезную диагностику.

См. также

open, close, read, write

PRTNTF, **VPRINTF**

Синтаксис:

```
#include <stdio.h>

int printf(char *fmt, ...)
int vprintf(char *fmt, va_list va_arg)
```

Описание:

printf() является программой форматированного вывода, воздействующей на stdout. Существуют аналогичные подпрограммы, воздействующие на заданный поток (fprintf()) или буфер строки (sprintf()). Подпрограмме printf() передается строка формата, за которым следует список из нуля или более параметров. В строке формата располагаются спецификации преобразования, каждая из которых используется для печати одного из значений в списке параметров. Каждая спецификация преобразования имеет форму **%m.nc**, где символ процента **%** представляет преобразование, за которым следует необязательная спецификация ширины **m**. **n** является дополнительной спецификацией точности (предваряется точкой), и **c** является буквой, определяющей тип преобразования. Знак "минус" ('-') предшествующий m указывает выравнивание преобразуемого значения в поле по левому краю, а не по правому. Когда ширина поля больше, чем требуется для преобразования, дополнение пробелами выполняется слева или справа, согласно определения. Если определено числовое преобразование с выравниванием вправо и первая цифра m равна 0, то дополнение будет выполняться с помощью нолей, а не пробелов.

Если символ * будет использоваться вместо десятичной константы, например, в формате %*d, один целочисленный параметр будет взят из списка, чтобы обеспечить это значение. Типы преобразования:

f Плавающая точка - m является общей шириной, и n - число цифр после десятичной точки. Если n опущено его значение по умолчанию равняется 6.

e Печатать соответствующего параметра в экспоненциальном представлении. В остальном аналогичен f.

g Используется формат e или f, в зависимости от того какой обеспечивает максимальную точность при минимальной ширине.

oxXud Целочисленное преобразование в системе счисления 8, 16, 16, 10 и 10 соответственно. Преобразование со знаком в случае d, без знака в противном случае. Значение точности определяет общее количество распечатываемых цифр и может использоваться, чтобы вызвать печать предшествующих нулей. Например, %8.4x распечатает, по крайней мере, 4 шестнадцатеричных цифры в поле шириной 8 символов. Предшествование ключевой буквы l указывает, что параметр значения является длинным целым или значением без знака. Буква X распечатывает шестнадцатеричные числа, используя прописные буквы A-F, а не a-f, как было бы распечатано при использовании x.

s печатать строки - значением параметра, как предполагается, является символьный указатель. В поле шириной из m символов будет распечатано не более n символов строки.

c Параметр, как предполагается, является единственным символом и распечатывается без преобразований.

Любые другие символы, используемые в качестве спецификации преобразования, будут напечатаны. Таким образом, **%%** будет производить одинарный знак процента. Некоторые примеры:

printf("Total = %4d%%", 23)
создаст 'Total = 23%'

printf("Size is %lx" , size)
Когда size имеет тип данных long, выводит size в шестнадцатеричном представлении.

printf("Name = %.8s", "a1234567890")
создаст 'Name = a1234567'

printf("xx%*d", 3, 4)
создаст 'xx 4'

При ошибке printf возвращает EOF, 0 в противном случае. vprintf() подобна printf(), но принимает указатель переменного списка параметров вместо списка параметров. См. описание va_start() для получения дополнительной информации о переменных списках параметров.

См. также
fprintf, sprintf

PUTC

Синтаксис:
```
#include <stdio.h>

int putc(int c, FILE *stream)
```

Описание:
putc() является макро-версией fputc() и определен в файле stdio.h. См. fputc() для описания ее поведения.

См. также
fputc, getc, fopen, fclose

PUTCHAR

Синтаксис:
```
#include <stdio.h>

int putchar(int c)
```

Описание:
putchar() является операцией aputc() в stdout, определена в stdio.h.

См. также
putc, getc, freopen, fclose

PUTS

Синтаксис:

```
#include <stdio.h>

int puts(char *s)
```

Описание:

puts() пишет строку **s** в поток stdout, добавляя новую строку. Ноль, завершающий строку, не копируется. При возникновении ошибки возвращается EOF.

См. также

fputs, gets, freopen, fclose

PUTW

Синтаксис:

```
#include <stdio.h>

int putw(int w, FILE *stream)
```

Описание:

putw() копирует слово **w** в заданный поток **stream**. Она возвращает w, за исключением ошибки, в этом случае возвращается EOF. Так как это обычное целое число, должна использоваться функция ferror() для проверки на наличие ошибок.

См. также

getw, fopen, fclose

QSORT

Синтаксис:

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width, int (*func)())
```

Описание:

Функция qsort() является реализацией алгоритма быстрой сортировки. Она сортирует массив из **nel** элементов, каждый длиной **width** байт, расположенных в памяти последовательно начиная с адреса **base**. **func** является указателем на функцию, используемую qsort() для сравнения элементов. Она вызывает func с указателями на два элемента для сравнения. Если первый элемент считается больше, равен или меньше второго, тогда func() должна возвратить значение больше 0, равно 0 или меньше 0 соответственно.

```
static short array[100];

#define SIZE sizeof array/sizeof array [0]

a_func(p1, p2) short *p1, *p2; {
    return *p1 - *p2;
}

sort_em() {
    qsort(array, SIZE, sizeof array [0], afunc);
}
```

Это программа сортирует массив в порядке возрастания значений. Обратите внимание на использование выражения sizeof, чтобы сделать код независимым от размера short, или числа элементов в массиве.

RAND

Синтаксис:

```
#include <stdlib.h>

int rand(void)
```

Описание:

rand() является генератором псевдослучайных чисел. Она возвращает целое число в диапазоне от 0 до 32767, которое при каждом вызове изменяется псевдослучайным способом.

См. также

srand

READ

Синтаксис:

```
#include <unixio.h>

int  read(int fd, void *buf, size_t cnt)
```

Описание:

Функция read() читает из файла, связанного с **fd** до **cnt** байтов в буфер, расположенный в **buf**. Она возвращает число фактически прочитанных байтов. Возврат 0 указывает конец файла. Возврат отрицательного значения означает ошибку. Значение fd должно быть получено из предыдущего вызова open(). Функция read() может вернуть меньше байтов, чем требуется, например, при чтении из консоли, в этом случае read() считает одну входную строку.

См. также

open, close, write

REALLOC

Синтаксис:

```
void  *realloc(void *ptr, size_t cnt)
```

Описание:

Функция realloc() освобождает блок памяти в **ptr**, который должен был быть получен предыдущим вызовом malloc(), calloc() или realloc(), затем пытается выделить **cnt** байт динамической памяти, и если удачно, то скопирует содержимое блока памяти в располагающийся в ptr новый блок. В лучшем случае realloc() скопирует количество байтов, которые были в старом блоке, но если новый блок будет меньше, то будут скопированы только cnt байтов. Если блок не может быть выделен, возвращается 0 .

См. также

malloc, calloc, realloc

REMOVE

Синтаксис:

```
#include <stdio.h>

int remove(char *s)
```

Описание:

remove() попытается удалить файл из каталога, названный параметром **s**. Возвращаемое значение -1 указывает, что попытка не удалась.

См. также

unlink

RENAME

Синтаксис:

```
#include <stdio.h>

int rename(char *name1, char *name2)
```

Описание:

Файл с именем **name1** будет переименован в **name2**. Возвращается -1, если переименование не удалось. Обратите внимание, что переименования с различными номерами пользователя, или дисками не допускаются.

См. также

open, close, unlink

REWIND

Синтаксис:

```
#include <stdio.h>

int rewind(FILE *stream)
```

Описание:

Эта функция попытается изменить местоположение указателя чтения/записи назначенного потока **stream** к началу файла. Возвращаемое значение -1 указывает, что попытка не была успешна, возможно потому, что поток связан не с файлом произвольного доступа, таким как символьное устройство.

См. также

fseek, ftell

SBRK

Синтаксис:

```
char *sbrk(int incr)
```

Описание:

Функция sbrk() увеличивает текущий адрес самой верхней ячейки памяти, выделенной программе на **incr** байт. Она возвращает указатель на предыдущее значение адреса самой верхней ячейки памяти. Таким образом, sbrk(0) возвращает указатель на текущий адрес самой верхней ячейки памяти, не изменяя его значение. Если для удовлетворения запроса недостаточно памяти, возвращается -1.

См. также

brk, malloc, calloc, realloc, free

SCANF

Синтаксис:

```
#include <stdio.h>

int scanf(char *fmt, ...)
int vscanf(char *fmt, va_list ap)
```

Описание:

Функция `scanf()` выполняет форматированный ввод ("де-редактирование") из потока `stdin`. Аналогичные функции доступны для потоков и для строк. Функция `vscanf()` подобна, но принимает указатель на список аргументов, а не список дополнительных параметров. Этот указатель должен быть предварительно инициализирован с помощью `va_start()`. Входные преобразования выполняются согласно строке **`fmt`**. Обычно символ в строке формата должен соответствовать символу на вводе. Однако, пробелу в строке формата будет соответствовать ноль или более "пробельных" символов на вводе, т.е. символов пробела, табуляции или новой строки. Спецификация преобразования принимает форму символа `%`, за которым необязательно следует символ подавления присвоения ('*'), затем необязательное числовое значение максимальной ширины поля, за которым следует символ спецификации преобразования. Каждая спецификация преобразования, если она не включает символ подавления присвоения, присвоит значение переменной, на которую указывает следующий параметр. Таким образом, если присутствуют две спецификации преобразования в строке `fmt`, должно быть два дополнительных указателя параметров. Символы преобразования следующие:

`o x d` Пропустит пробельные символы, затем преобразует число к основанию системы счисления 8, 16 или 10 соответственно. Если была предоставлена ширина поля, возьмет символы из ввода в количестве не более этого числа. При вводе распознается предшествующий знак "минус".

`f` Пропустит пробельные символы, затем преобразует вещественное число в стандартном или в экспоненциальном представлении. Ширина поля применяется как указано выше.

`s` Пропустит пробельные символы, затем скопирует последовательность максимальной длины из непробельных символов. Параметр указателя должен быть указателем на `char`. Ширина поля будет ограничивать количество скопированных символов. Результирующая строка будет завершена `0`.

`c` Копирует следующий символ из ввода. Параметр указателя, как предполагается, является указатель на `char`. Если ширина поля определена, то будут скопированы символы в количестве не более этого числа. Это отличается от формата `s` тем, что пробельные символы не прерывают последовательность символов.

Символам преобразования `o`, `x`, `u`, `d` и `f` может предшествовать `l`, чтобы указать, что соответствующий параметр указателя является указателем на `long` или `double` при необходимости. Предшествующий `h` означает, что параметр указателя является указателем на `short`, а не `int`. Функция `scanf()` возвращает число успешных преобразований. Если до выполнения любых преобразований встречается конец файла, возвращается `EOF`.

Некоторые примеры:

scanf("%d%s", &a, &s)
при вводе " 12s", 12 будет присвоено a и "s" - s.

scanf("%3cd %lf', &c, &f)
при вводе " abcd -3.5", " abc" будет присвоено c, и -3.5 - f.

См. также
fscanf, sscanf, printf, va_arg

SEGREAD

Синтаксис:
```
#include <dos.h>

int segread(struct SREGS *segregs)
```

Описание:
Функция segread() копирует значения сегментных регистров в структуру, на которую указывает **segregs**.

См. также
int86, int86x, intdos, intdosx

SETJMP

Синтаксис:
```
#include <setjmp.h>

int setjmp(jmp_buf buf)
```

Описание:
Функция setjmp() используется с longjmp() для нелокальных переходов. Для получения дополнительной информации см. longjmp().

См. также
longjmp

SETUID (только CP/M)

Синтаксис:
```
#include <sys.h>

void setuid(int uid)
```

Описание:
Функция setuid() устанавливает текущий код пользователя в **uid**. Значение uid должно быть числом в диапазоне 0-15.

См. также
getuid

SETVBUF, **SETBUF**

Синтаксис:

```
#include  <stdio.h>

int setvbuf(FILE *stream, char *buf, int mode, size_t size);
void setbuf(FILE *stream, char *buf)
```

Описание:

Функция `setvbuf()` позволяет изменить поведение буферизации потока `stdio`. Она заменяет функцию `setbuf()`, которая сохранена для обратной совместимости. Параметры `setvbuf()` следующие: **`stream`** определяет затронутый поток `stdio`. **`buf`** - указатель на буфер, который будет использоваться для всех последующих операций ввода-вывода в этот поток. Если `buf` равен `0`, то подпрограмма при необходимости выделит буфер из "кучи" размером `BUFSIZ`, как определено в `<stdio.h>`. Параметр **`mode`** может принять значения `_IONBF` для полного отключения буферизации, `_IOFBF` для полной буферизации или `_IOLBF` для построчной буферизации. Полная буферизация означает, что связанный буфер будет сброшен, только когда заполнен, в то время как построчная буферизация означает, что буфер будет сброшен в конце каждой строки или когда будет требоваться ввод из другого потока `stdio`, **`size`** - размер предоставленного буфера. Например:

```
setvbuf(stdout, my_buf, _IOLBF, sizeof my_buf);
```

Если буфер предоставляется вызывающей стороной, то этот буфер останется связанным с этим потоком даже после вызовов `fclose()`, `fopen()` до изменения другим вызовов `setvbuf()`.

См. также

`fopen`, `freopen`, `fclose`

SET_VECTOR

Синтаксис:

```
#include <intrpt.h>

typedef interrupt void (*isr)();
isr set_vector(isr *vector, isr func);
```

Описание:

Эта подпрограмма позволяет инициализировать вектор прерываний. Первый параметр должен быть адресом вектора прерывания (не номером вектора, а фактическим адресом) приведенным к указателю на **`isr`**, который является `typedef` указателем на функцию обработки прерываний. Второй параметр должен быть функцией, на которую будет указывать вектор прерывания. Он должен быть объявлен, используя спецификатор типа `interrupt`. Возвращаемое значение `set_vector()` является предыдущим содержанием вектора.

См. также

`di()`, `ei()`

SIGNAL

Синтаксис:

```
#include <signal.h>

void (*signal)(int sig, void (*func)());
```

Описание:

Функция signal() обеспечивает механизм для перехвата нажатий Ctrl-C (Ctrl-BREAK в MS-DOS) введенных в консоли во время ввода-вывода. В CP/M консоль опрашивается при каждом выполнении вызова ввода-вывода, в то время как в MS-DOS опрос зависит от установок команды BREAK. Если будет обнаружен Ctrl-C, то будет выполняться некоторое действие. Действие по умолчанию - немедленный выход. Оно может быть изменено с помощью signal(). Параметр **sig** функции signal может в настоящее иметь значение только SIGINT, означающее условие прерывания. Параметр **func** может иметь одно из значений - SIGDFL, представляющий действие по умолчанию, SIGIGN для полного игнорирования Ctrl-C или адрес функции, которая будет вызвана с одним параметром, номером пойманного сигнала, при обнаружении Ctrl-C. Поскольку единственным поддерживаемым сигналом является SIGINT, он всегда будет значением параметра вызванной функции.

См. также

exit

SIN

Синтаксис:

```
#include <math.h>

double sin(double f);
```

Описание:

Эта функция возвращает значение синуса своего аргумента.

См. также

cos, tan, asin, acos, atan

SPAWNL, **SPAWNV**, **SPAWNVE**

Синтаксис:

```
int spawnl(char *n, char *argv0, ...);
int spawnv(cahr *n, char **v)
int spawnve(char *n, char **v, char **e)
```

Описание:

Эти функции загрузят и выполнят подпрограмму, названную параметром **n**. Соглашения о вызовах аналогичны функциям execl() и execv(), с той разницей, что функции spawn возвращаются в программу вызова после завершения подпрограммы, в то время как функции exec возвращаются, только если программа не могла быть выполнена. Функция spawnve() принимает список среды в том же формате, в каком предоставляется список аргументов выполняемой программе в качестве ее среды.

См. также

execl, execv

SPRINTF

Синтаксис:

```
#include <stdio.h>
int sprintf(char *buf, char *fmt, ...);
int vsprintf(char *buf, char *fmt, valist ap);
```

Описание:

Функция sprintf() действует аналогично printf(), за исключением того, что вместо помещения преобразованных выходных данных в поток stdout, символы помещаются в буфер buf. Результирующая строка будет завершена 0, и будет возвращено число символов в буфере. Функция vsprintf принимает в качестве параметра указатель, а не список параметров.

См. также

printf, fprintf, sscanf

SQRT

Синтаксис:

```
#include <math.h>

double sqrt(double f)
```

Описание:

sqrt() реализует функцию квадратного корня используя приближение Ньютона.

См. также

exp

SSCANF

Синтаксис:

```
#include <stdio.h>

int sscanf(char *buf, char *fmt,...);
int vsscanf(char *buf, char *fmt, va_list ap);
```

Описание:

Функция sscanf() действует аналогично scanf(), за исключением того, что вместо преобразований, взятых из stdin, они принимаются из строки buf.

См. также

scanf, fscanf, sprintf

SRAND

Синтаксис:

```
#include <stdlib.h>

void srand(int seed)
```

Описание:

Функция srand() инициализирует генератор случайных чисел, к которому обращается rand() с заданным началом. Это обеспечивает механизм изменения начальной точки псевдослучайной последовательности, заданной rand(). На z80 хорошим способом получения по-настоящему случайного числа явля-

ется обновленное значение регистра. В противном случае можно задействовать время ответа от консоли.

См. также

rand

STAT

Синтаксис:

```
#include <stat.h>

int stat(char *name, struct stat *statbuf)
```

Описание:

Эта подпрограмма возвращает информацию о файле с именем **name**. Возвращаемые сведения зависят от операционной системы, но могут включать атрибуты файла (например, только для чтения), размер файла в байтах и время изменения файла и/или время доступа. Параметр name должен быть именем файла и может включать пути в DOS, коды пользователя в CP/M, и т.д. Параметр **statbuf** должен быть адресом структуры определенной в файле stat.h, которая будет заполнена информацией о файле. Строение структуры stat следующее:

```
struct stat {
    short  st_mode;   /* Флаги */
    long   st_atime;  /* Время доступа */
    long   st_mtime;  /* Время изменения */
    long   st_size;   /* Размер файла */
};
```

Время доступа и время изменения (в DOS они оба установлены во время изменения) находятся в секундах с 00:00:00 1 января 1970. Может использоваться функция ctime() для преобразования его в читаемое значение. Размер файла не требует пояснений. Биты флагов следующие:

Флаг	Значение
S_IFMT	Маска типа файла
S_IFDIR	Файл является каталогом
S_IFREG	Обычный файл
S_IREAD	Читаемый файл
S_IWRITE	Записываемый файл
S_IEXEC	Выполняемый файл
S_HIDDEN	Скрытый файл
S_SYSTEM	Системный файл
S_ARCHIVE	Файл был записан в

Функция stat возвращает 0 в случае успеха, -1 в случае неудачи, например, если файл не может быть найден.

См. также

ctime, creat, chmod

STRCAT, **STRCMP**, **STRCPY**, **STRLEN** и другие

Синтаксис:

```
#include <string.h>
char *strcat(char *s1, char *s2);
int   strcmp(char *s1, char *s2);
char *strcpy(char *s1, char *s2);
int   strlen(char *s);
char *strncat(char *s1, char *s2, size_t n);
int   strncmp(char *s1, char *s2, size_t n);
char *strncpy(char *s1, char *s2, size_t n);
```

Описание:

Эти функции обеспечивают операции со строками завершенными нулем. Функция `strcat()` добавляет строку `s2` в конец строки `s1`. Строка в `s1` будет завершена нулем. Само собой разумеется, буфер в `s1` должен быть достаточно большим.

Функция `strcmp()` сравнивает две строки и возвращает число, больше чем `0`, `0` или меньше чем `0` согласно тому, больше ли `s1`, равна или меньше, чем `s2`. Сравнение выполняется с учетом упорядочивания кодов `ASCII` с первым старшим символом.

Функция `strcpy()` копирует `s2` в буфер `s1`, его завершает ноль.

Функция `strlen()` возвращает длину `s1`, не включая завершающий ноль. Функции `strncat()`, `strncmp()` и `strncpy()` соединят, сравнят и скопируют `s2` и `s1` таким же образом, как их одноименные аналоги выше, но с участием не более **`n`** символов. Для `strncpy()` результирующая строка может быть не завершена нулем.

STRCHR, **STRRCHR**

Синтаксис:

```
#include <string.h>
char *strchr(char *s, int c)
char *strrchr(char *s, int c)
```

Описание:

Эти функции определяют местоположение символа **`c`** в строке **`s`**. В случае `strchr()` будет возвращен указатель на первый символ, найденный с начала строки, в то время как `strrchr()` ищет назад от конца строки. Если символ в строке не существует – возвращается нулевой указатель `NULL`.

См. также

`strlen`, `strcmp`, `strcpy`, `strcat`

SYSTEM

Синтаксис:

```
#include <sys.h>
int system(char *s)
```

Описание:

При выполнении в `MS-DOS` `system()` передаст строку параметров в командный процессор для выполнения с помощью строки окружения `COMSPEC`. Статус выхода командного процессора будет возвращен из вызова `system()`. Например,

чтобы установить скорость передачи последовательного порта в машине с MS-DOS:

```
system("MODE C0M1:96,N,8,1,P");
```

Эта функция не будет работать CP/M-86, так как у нее нет применимого интерпретатора команд. В MS-DOS и CP/M используется системный вызов CLI.

См. также

spawnl, spawnv

TAN

Синтаксис:

```
#include <math.h>

double tan(double f);
```

Описание:

Это функция тангенса.

См. также

sin, cos, asin, acos, atan

TIME

Синтаксис:

```
#include <time.h>

time_t time(time_t *t)
```

Описание:

Эта функция возвращает текущее время в секундах с 00:00:00 1 января 1970. Если параметр **t** не равен NULL, то же значение хранится в объекте, на который указывает t. Точность этой функции естественно зависит от операционной системы, имеющей корректное время. Эта функция не работает в CP/M-86 или CP/M 2.2, но действительно работает в Concurrent CP/M и CP/M+.

См. также

ctime, gmtime, localtime, asctime

TOUPPER, TOLOWER, TOASCII

Синтаксис:

```
#include <ctype.h>

char toupper(int c);
char tolower(int c);
char toascii(int c);
char c;
```

Описание:

Функция toupper() преобразует свой алфавитный параметр **c** в нижнем регистре в верхний регистр, функция tolower() выполняет обратное преобразование, и toascii() возвращает результат, который гарантируется в диапазоне 0-0177. Функции toupper() и tolower() возвращают свои параметры, если они не являются алфавитными символами.

См. также

islower, isupper, isascii и др.

UNGETC

Синтаксис:

```
#include <stdio.h>

int ungetc(int c, FILE *stream)
```

Описание:

Функция ungetc() попытается поместить символ **c** обратно в указанный поток stream, так, что последующий вызов getc() возвратит символ. Максимально позволен один уровень возврата, и если поток не буферизован, даже это может быть не возможно. Если функция ungetc() не может быть выполнена, возвращается EOF.

См. также

getc

UNLINK

Синтаксис:

```
int unlink(char *name)
```

Описание:

Функция unlink() удалит названный файл, то есть сотрет файл из его каталога. Смотрите open() для уточнения правил построения имени файла. В случае успеха возвращается 0, и -1, если файл не существует, или не может быть удален.

См. также

open, close, rename, remove

VA_START, **VA_ARG**, **VA_END**

Синтаксис:

```
#include   <stdarg.h>

void va_start(va_list ap, parmN);
type va_arg(ap, type);
void va_end(va_list ap);
```

Описание:

Эти макросы предоставляются для обеспечения доступа переносимым способом к параметрам функции, представленных в прототипе символом многоточие (...), где тип и число параметров, предоставленных функции, не известны во время компиляции. Самый правый параметр функции (обозначенный parmN) играет важную роль в этих макросах, так как он является начальной точкой для доступа к последующим параметрам. В функции, принимающей переменное число параметров, должна быть объявлена переменная типа va_list, затем вызывается макрос va_start с этой переменной и именем parmN. Он инициализирует переменную, чтобы позволить последующим вызовам макроса va_arg получать доступ к последовательным параметрам. Каждый вызов va_arg требует два параметра. Ранее определенную переменную и имя типа, следующего параметра. Обратите внимание, что любые параметры доступные таким образом будут расширены в соответствии с соглашениями по умолчанию к int, unsigned int или double. Например, если передан символьный параметр, к нему должен получить доступ va_arg(ap, int), так как символ будет расширен до int. Ниже приведен пример функции, при-

нимающей один целочисленный параметр, сопровождаемый многими другими параметрами. В этом примере функция ожидает, что последующие параметры будут указателями на `char`, но отметьте, что компилятор не знает об этом, и программисты обязаны гарантировать, что предоставлены корректные параметры.

```
#include <stdarg.h>

prf(int n, ...) {
  va_list ap;
  va_start(ap, n);
  while(n--)
  puts(va_arg(ap, char *));
  va_end(ap);
}
```

WRITE

Синтаксис:

```
#include <unixio.h>
int write(int fd, void *buf, size_t cnt)
```

Описание:

Функция `write()` записывает из буфера **`buf`** до **`cnt`** байтов в файл, связанный с дескриптором файла **`fd`**. Возвращается число фактически записанных байтов. При ошибке возвращается `EOF` или значение меньше чем `cnt`. В любом случае любое возвращаемое значение, не равное `cnt`, должно быть обработано как ошибка (см. `read()`).

См. также

`open`, `close`, `read`

Предметный указатель

S

T

U

V

W