

# Оглавление

<b>Об авторе</b> .....	12
<b>О техническом рецензенте</b> .....	13
<b>Предисловие от издательства</b> .....	14
<b>Введение</b> .....	15
<b>Прежде чем начать</b> .....	17
<b>Глава 1. Самая первая программа</b> .....	19
Редактирование, ассемблирование, связывание и запуск (или отладка).....	20
Структура программы на ассемблере .....	25
Раздел section .data .....	25
Раздел section .bss.....	26
Раздел section .txt.....	27
Резюме.....	29
<b>Глава 2. Двоичные и шестнадцатеричные числа и регистры</b> .....	30
Краткий вводный курс по двоичным числам.....	30
Целые числа .....	31
Числа с плавающей точкой .....	32
Краткий вводный курс по регистрам.....	32
Регистры общего назначения .....	33
Регистр счетчика команд (rip) .....	34
Регистр флагов .....	34
Регистры xmm и ymm .....	35
Резюме.....	35
<b>Глава 3. Анализ программ с помощью отладчика: GDB</b> .....	36
Начало отладки .....	36
Двигаемся дальше .....	42
Некоторые дополнительные команды отладчика GDB .....	44
Немного улучшенная версия программы hello, world .....	45
Резюме.....	47

---

<b>Глава 4. Следующая программа: Alive and Kicking</b> .....	48
Анализ программы alive .....	49
Вывод.....	53
Резюме.....	56
<b>Глава 5. Ассемблер основан на логике</b> .....	57
Логический оператор NOT .....	57
Логический оператор OR .....	57
Логический оператор XOR.....	58
Логический оператор AND.....	58
Резюме.....	59
<b>Глава 6. Отладчик Data Display Debugger</b> .....	60
Работа с отладчиком DDD .....	60
Резюме.....	63
<b>Глава 7. Переходы и циклы</b> .....	64
Установка SimpleASM .....	64
Использование SASM .....	64
Резюме.....	72
<b>Глава 8. Память</b> .....	73
Обследование памяти .....	73
Резюме.....	80
<b>Глава 9. Целочисленная арифметика</b> .....	81
Основы использования целочисленной арифметики .....	81
Изучение арифметических инструкций .....	84
Резюме.....	87
<b>Глава 10. Стек</b> .....	88
Изучение работы стека .....	88
Наблюдение за стеком.....	91
Резюме.....	93
<b>Глава 11. Арифметика с плавающей точкой</b> .....	94
Сравнение чисел с обычной и двойной точностью .....	94
Кодирование с применением чисел с плавающей точкой .....	96
Резюме.....	98
<b>Глава 12. Функции</b> .....	99
Создание простой функции .....	99

---

Еще о функциях .....	100
Резюме.....	102
<b>Глава 13. Выравнивание стека и фрейм стека .....</b>	<b>103</b>
Выравнивание стека.....	103
Более подробно о фреймах стека .....	105
Резюме.....	106
<b>Глава 14. Внешние функции .....</b>	<b>107</b>
Создание и связывание функций.....	107
Расширенная версия makefile .....	110
Резюме.....	111
<b>Глава 15. Соглашения о вызовах функций.....</b>	<b>112</b>
Аргументы функций.....	113
Схема стека .....	116
Сохранение регистров.....	118
Резюме.....	120
<b>Глава 16. Операции с битами .....</b>	<b>121</b>
Основные положения.....	121
Арифметика .....	126
Резюме.....	129
<b>Глава 17. Работа с битами .....</b>	<b>130</b>
Другие способы изменения битов.....	130
Переменная bitflags.....	132
Резюме.....	133
<b>Глава 18. Макрокоманды .....</b>	<b>134</b>
Создание макроса.....	134
Использование objdump.....	136
Резюме.....	137
<b>Глава 19. Ввод и вывод в консоли .....</b>	<b>138</b>
Использование средств ввода/вывода .....	138
Обработка переполнений .....	140
Резюме.....	143
<b>Глава 20. Файловый ввод/вывод .....</b>	<b>144</b>
Использование системных вызовов.....	144
Обработка файла .....	145

---

Условное ассемблирование .....	152
Инструкции для обработки файлов.....	152
Резюме.....	153
<b>Глава 21. Командная строка .....</b>	<b>154</b>
Доступ к аргументам командной строки.....	154
Отладка программы с аргументами командной строки .....	155
Резюме.....	157
<b>Глава 22. Использование ассемблера в коде C.....</b>	<b>158</b>
Создание файла исходного кода на языке C.....	158
Создание файла исходного кода на ассемблере .....	160
Резюме.....	163
<b>Глава 23. Встроенный ассемблер.....</b>	<b>164</b>
Простой встроенный ассемблерный код .....	164
Расширенный встроенный ассемблерный код.....	166
Резюме.....	169
<b>Глава 24. Строки .....</b>	<b>170</b>
Обработка строк .....	170
Сравнение и сканирование строк .....	174
Резюме.....	178
<b>Глава 25. Предъявите ваш идентификатор .....</b>	<b>179</b>
Использование инструкции <code>cpuid</code> .....	179
Использование инструкции <code>test</code> .....	181
Резюме.....	183
<b>Глава 26. SIMD.....</b>	<b>184</b>
Скалярные данные и упакованные данные.....	184
Невыровненные и выровненные данные .....	186
Резюме.....	187
<b>Глава 27. Работа с битами регистра <code>mxcsr</code> .....</b>	<b>189</b>
Анализ программы.....	194
Резюме.....	196
<b>Глава 28. Выравнивание для SSE.....</b>	<b>197</b>
Пример без выравнивания .....	197
Пример с выравниванием.....	200
Резюме.....	203

<b>Глава 29. SSE-инструкции для работы с упакованными целыми числами</b> .....	204
SSE-инструкции для работы с целыми числами .....	204
Анализ исходного кода.....	206
Резюме.....	206
<b>Глава 30. Обработка строк средствами SSE</b> .....	207
Управляющий байт <code>imm8</code> .....	208
Использование управляющего байта <code>imm8</code> .....	209
Биты 0 и 1 .....	209
Биты 2 и 3.....	209
Биты 4 и 5.....	210
Бит 6.....	211
Зарезервированный бит 7.....	211
Флаги .....	211
Резюме.....	212
<b>Глава 31. Поиск символа в строке</b> .....	213
Определение длины строки .....	213
Поиск в строках .....	216
Резюме.....	219
<b>Глава 32. Сравнение строк</b> .....	220
Строки с неявно заданной длиной.....	220
Строки с явно заданной длиной.....	222
Резюме.....	226
<b>Глава 33. Перемешиваем данные</b> .....	227
Основные принципы операций перемешивания .....	227
Перемешивание в случайном порядке .....	231
Перемешивание в обратном порядке .....	233
Перемешивание вращением.....	234
Перемешивание байтов .....	234
Резюме.....	236
<b>Глава 34. SSE-инструкции: маски строк</b> .....	237
Поиск символов .....	237
Поиск символов из заданного диапазона.....	243
Поиск подстроки.....	246
Резюме.....	249

---

<b>Глава 35. AVX</b> .....	250
Проверка поддержки AVX .....	250
Пример программы с использованием AVX .....	252
Резюме .....	256
<b>Глава 36. Операции с матрицами с использованием AVX</b> .....	257
Пример исходного кода для операций с матрицами .....	257
Вывод матрицы: <code>printm4x4</code> .....	264
Умножение матриц: <code>multi4x4</code> .....	265
Обращение матрицы: <code>inverse4x4</code> .....	268
Теорема Гамильтона–Кэли .....	268
Алгоритм Фаддеева–Леверье .....	268
Исходный код .....	269
Резюме .....	273
<b>Глава 37. Транспонирование матриц</b> .....	274
Пример исходного кода для транспонирования матриц .....	274
Версия с использованием неупакованных данных .....	277
Версия с применением перемешивания .....	282
Резюме .....	285
<b>Глава 38. Оптимизация производительности</b> .....	286
Производительность вычисления транспонированной матрицы .....	286
Производительность вычисления следа матрицы .....	292
Резюме .....	297
<b>Глава 39. Приветствуем мир Windows</b> .....	298
Начинаем изучение .....	298
Пишем код в Windows .....	300
Отладка .....	302
Системные вызовы .....	302
Резюме .....	303
<b>Глава 40. Использование Windows API</b> .....	304
Вывод в консоли .....	304
Создание окон Windows .....	307
Резюме .....	308
<b>Глава 41. Функции в Windows</b> .....	309
Использование более четырех аргументов функции .....	309
Обработка значений с плавающей точкой .....	314
Резюме .....	316

---

<b>Глава 42. Функции с переменным числом аргументов .....</b>	<b>317</b>
Функции с переменным числом аргументов в Windows .....	317
Обработка смешанных значений .....	319
Резюме .....	320
<b>Глава 43. Работа с файлами в Windows.....</b>	<b>321</b>
Резюме.....	324
<b>Послесловие. Что дальше? .....</b>	<b>325</b>
<b>Предметный указатель .....</b>	<b>326</b>

## Об авторе



**Йо Ван Гуй (Jo Van Hoey)** обладает 40-летним опытом работы в сфере информационных технологий, выполняя разнообразные функции в многочисленных ИТ-компаниях с использованием различных компьютерных платформ. Недавно он уволился из компании IBM, где являлся менеджером по работе с клиентами, использующими ПО для мейнфреймов. Йо всегда интересовался проблемами безопасности в области ИТ, а знание языка ассемблера представляет собой весьма важный профессиональный навык при защите ИТ-инфраструктуры от атак и вредоносных программ.



# О техническом рецензенте



**Пол Коэн (Paul Kohan)** присоединился к компании Intel Corporation в те далекие дни, когда только еще создавалась архитектура x86, начиная с микропроцессора 8086, и уволился из Intel после 26 лет работы, связанной с продажами, маркетингом и управлением. В настоящее время Пол сотрудничает с компанией Douglas Technology Group, деятельность которой сосредоточена на издании книг от имени Intel и других корпораций. Кроме того, Пол читает курс, превращающий учеников средней школы и студентов младших курсов в реальных уверенных в себе предпринимателей, сотрудничая с Young Entrepreneurs Academy (YEA), а также является членом транспортного суда в городе Бивертон (шт. Орегон) и совета директоров нескольких некоммерческих организаций.

# Предисловие от издательства

## ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте [www.dmkpress.com](http://www.dmkpress.com), зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com); при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу [http://dmkpress.com/authors/publish\\_book/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

## СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com), и мы исправим это в следующих тиражах.

## НАРУШЕНИЕ АВТОРСКИХ ПРАВ

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

# Введение

Изучение программирования на ассемблере может оказаться обескураживающим, но совсем не потому, что это язык, не прощающий ошибок, ведь компьютер будет «одобрять» ваши действия при каждом удобном случае. А если это не так, то, возможно, где-то в программе скрывается необнаруженная ошибка, которая «укусит» вас во время выполнения программы. Сверх всего прочего кривая сложности обучения весьма крута, язык загадочный и не сразу понятный, официальная документация Intel ошеломляюще велика, а доступные инструменты разработки обладают весьма специфическими особенностями.

Эта книга научит вас программировать на ассемблере, начиная с самых простых программ и постепенно осваивая путь к овладению программированием с использованием расширенной системы команд Advanced Vector Extensions (AVX). Прочитав эту книгу полностью, вы сможете писать и читать код на ассемблере, ассемблерный код, объединенный с языками высокого уровня, поймете, что такое AVX и многое другое. Цель этой книги – показать, как используются инструкции языка ассемблера. Это не руководство по стилю программирования или по оптимизации производительности кода. После того как вы освоите базовые знания об ассемблере, можно будет продолжить обучение по теме оптимизации кода. Эта книга не должна быть вашей первой книгой по программированию: если вы никогда раньше не программировали, то отложите эту книгу на время и изучите основы программирования на каком-либо языке высокого уровня, например на C.

Весь исходный код, используемый здесь, доступен по ссылке Download Source Code на сайте [www.apress.com/9781484250754](http://www.apress.com/9781484250754). Исходный код в книге представлен в настолько простом виде, насколько это вообще возможно, т. е. без каких-либо графических пользовательских интерфейсов, без излишеств («бантиков и рюшечек»), без средств проверки на ошибки. Добавление всех этих замечательных функциональных возможностей увело бы нас от истинной цели: изучение языка ассемблера.

Теоретическая часть сведена к необходимому минимуму: немного информации о двоичных (бинарных) числах, краткое описание логических операторов и кое-что об основах линейной алгебры. Здесь не рассматриваются операции преобразования чисел с плавающей точкой. Если требуется преобразовать двоичные или шестнадцатеричные числа, то найдите веб-сайт, который сделает это за вас. Не теряйте времени на вычисления вручную. Помните о главной цели: изучение ассемблера.

Исходный ассемблерный код представлен в виде завершенных программ, так что вы можете протестировать их на своем компьютере, поэкспериментировать, изменять их и «ломать»...

Кроме того, будут рассматриваться инструментальные средства, которыми можно воспользоваться, и потенциальные проблемы при использовании этих

инструментов. Выбор правильных инструментов чрезвычайно важен для преодоления крутой кривой сложности обучения. Иногда будут встречаться ссылки на книги, статьи, прочие документы и веб-сайты, которые могут оказаться полезными при обучении или содержать более подробные описания.

Автор не намеревался предоставить исчерпывающий курс по всем инструкциям ассемблера. Это невозможно сделать в одной книге (взгляните на размеры справочных руководств компании Intel). Читателю предоставлена возможность попробовать на практике основные компоненты, чтобы получить представление о том, куда двигаться дальше. При работе с этой книгой вы получите знания, необходимые для самостоятельного дальнейшего глубокого изучения определенных предметных областей ИТ. После завершения чтения этой книги вы сможете изучать справочные руководства компании Intel и понимать (во всяком случае, пытаться понять) смысл их содержимого.

Основная часть книги содержит информацию о применении ассемблера в Linux, потому что это самая простая и удобная платформа для изучения языка ассемблера. В заключительной части книги представлено несколько глав, описывающих методику использования ассемблера в Windows. Вы сами убедитесь в том, что, вооружившись ассемблером Linux, гораздо проще осваивать ассемблер в Windows.

Существует несколько трансляторов ассемблера, доступных для использования с процессорами Intel, например FASM, MASM, GAS, NASM и YASM (список далеко не полный). В этой книге используется транслятор NASM, поскольку он многоплатформенный: доступен для Linux, Windows и macOS. Кроме того, он обладает относительно большой пользовательской базой. Но не стоит беспокоиться; если вы знаете один ассемблер, то с легкостью освоите любой другой «диалект».

Исходный код в книге тщательно проверен и протестирован. Но если обнаружатся какие-либо опечатки в тексте или ошибки в программах, мы не несем за это никакой ответственности. Автор обвиняет в этом двух своих котов, которые любят прыгать на клавиатуру во время его работы.

Идеи и мнения, представленные в этой книге, принадлежат лично автору и не всегда представляют позицию, стратегии или мнения компании IBM.

# Прежде чем начать

Для чтения этой книги вы должны знать некоторые основные темы.

- **Вы должны уметь устанавливать и настраивать программное обеспечение виртуализации (VMware, VirtualBox или аналогичное ПО).** Если эти программы вам неизвестны, то загрузите бесплатное ПО Oracle VirtualBox (<https://www.virtualbox.org>), установите его и научитесь его использовать, установив, например, Ubuntu Desktop Linux как гостевую операционную систему (ОС). ПО виртуализации позволяет устанавливать различные гостевые ОС на основном компьютере, и если вы что-то напутали в такой гостевой ОС, то можете ее удалить и установить заново. Или если имеются мгновенные снимки состояния системы, то можно вернуться к предыдущей версии гостевой ОС. Другими словами, при экспериментах с гостевой ОС не будет нанесено никакого вреда основной операционной системе. В интернете можно найти огромное количество ресурсов, описывающих VirtualBox и другие решения с использованием ПО виртуализации.
- **Вы должны обладать базовыми знаниями об использовании интерфейса командной строки Linux.** В книге используется Ubuntu Desktop Linux и командная строка этой ОС, начиная с главы 1. Если хотите, можете работать в другом дистрибутиве Linux, но при этом необходимо убедиться в том, что имеется возможность установить все инструментальные средства, применяемые в данной книге (NASM, GCC, GDB, SASM и т. д.). Требуются следующие базовые знания: как установить ОС, как устанавливается дополнительное ПО, как запустить терминал с приглашением (промптом) командной строки, а также как создавать, перемещать, копировать и удалять каталоги и файлы в командной строке. Также необходимо уметь использовать утилиты tar, grep, find, ls, time и т. п. Вы должны знать, как запустить и использовать текстовый редактор. Не требуется никаких продвинутых знаний о Linux, нужны только самые простые, базовые навыки выполнения задач, для того чтобы следовать описаниям, приведенным в этой книге. Если вы незнакомы с ОС Linux, то немного поработайте в ней, чтобы освоить ее использование. В интернете существует множество небольших по объему, но качественных руководств для начинающих (например, <https://www.guru99.com/unix-linux-tutorial.html>). Вы сами убедитесь в том, что после изучения ассемблера на компьютере с ОС Linux освоение ассемблера в других ОС станет не таким уж трудным делом.
- **Вы должны обладать некоторыми базовыми знаниями в области программирования на языке C.** В книге в некоторых случаях применяются функции на языке C для упрощения примеров ассемблерного кода. Кроме того, будет показано, как организовать интерфейс с языком высокого уровня, таким как C. Если вы не знаете C, но намерены в полной мере освоить содержимое этой книги, то рекомендуется изучить пару

бесплатных курсов по С, например [tutorialspoint.com](http://tutorialspoint.com). Нет необходимости проходить полный курс, просто внимательно изучите несколько программ на этом языке. В дальнейшем всегда можно вернуться к курсу по С, чтобы узнать больше подробностей.

## ЗАЧЕМ НУЖНО ИЗУЧАТЬ АССЕМБЛЕР

Знание ассемблера дает некоторые преимущества:

- вы узнаете, как работает (центральный) процессор (ЦПУ) и оперативная память;
- вы узнаете, как совместно работают компьютер и операционная система;
- вы поймете, как компиляторы языков высокого уровня генерируют код на машинном языке, а эти знания могут помочь писать более эффективный код;
- вы получите более эффективные средства для анализа ошибок в программах;
- вы получите огромное удовольствие, когда, наконец, ваша программа заработает правильно;
- и причина, по которой я написал эту книгу: если необходимо исследовать вредоносное ПО, то в вашем распоряжении есть только машинный код без исходного кода. Если вы хорошо понимаете код ассемблера, то сможете проанализировать вредоносную программу, выполнить необходимые действия и принять превентивные меры.

## РУКОВОДСТВА КОМПАНИИ INTEL

Справочные руководства компании Intel содержат все, что может потребоваться при программировании микропроцессоров Intel. Но информация весьма сложна для освоения начинающими программистами. По мере чтения данной книги вы обнаружите, что описания в этих руководствах Intel постепенно становятся все более понятными. В книге часто встречаются ссылки на эти солидные тома информации.

Справочные руководства Intel можно найти здесь:

<https://software.intel.com/en-us/articles/intel-sdm>.

Только не пытайтесь распечатать их на бумаге – пожалейте деревья, которые вы можете уничтожить. Бегло просмотрите руководства, чтобы убедиться в том, насколько исчерпывающими, подробными и формализованными документами они являются. Попытка изучения ассемблера по этим руководствам может оказаться обескураживающе неудачной. Особый интерес для нас представляет Volume 2 (том 2), в котором вы найдете подробные описания программных инструкций ассемблера.

Полезный источник информации можно найти здесь: <https://www.felixcloutier.com/x86/index.html>. На этом сайте представлен список всех инструкций с краткими описаниями их использования. Если предоставленная здесь информация окажется недостаточной, то вы всегда можете вернуться к справочным руководствам Intel или обратиться к вашему надежному другу Google.

# Глава 1

## Самая первая программа

Многие поколения разработчиков начали свою карьеру программиста с изучения способа вывода на экран компьютерного дисплея фразы `hello, world`. Эта традиция была заложена в 1970-е гг. Брайаном Керниганом (Brian W. Kernighan) в книге, которую он написал вместе с Деннисом Ритчи (Dennis Ritchie) «The C Programming Language» («Язык программирования С»). Керниган принимал участие в разработке языка программирования С в компании Bell Labs. С тех пор язык С значительно изменился, но он остается языком, с которым должен быть знаком каждый уважающий себя программист. Большинство «новейших», «модных» языков программирования в той или иной степени являются наследниками языка С. Язык С иногда называют переносимым языком ассемблера, и как программист, стремящийся к овладению ассемблером, вы должны знать С. Уважая традицию, начнем с программы на ассемблере, выводящей фразу `hello, world` на экран. В листинге 1.1 показан исходный код версии на языке ассемблера этой программы, которую мы будем анализировать далее в этой главе.

### Листинг 1.1. *hello.asm*

```
;hello.asm
section .data
    msg db "hello, world",0
section .bss
section .text
    global main
main:
    mov    rax, 1      ; 1 = запись.
    mov    rdi, 1      ; 1 = в поток стандартного вывода stdout.
    mov    rsi, msg    ; Выводимая строка в регистре rsi.
    mov    rdx, 12     ; Длина строки без конечного 0.
    syscall           ; Вывод строки.
    mov    rax, 60     ; 60 = код выхода из программы.
    mov    rdi, 0      ; 0 = код успешного завершения программы.
    syscall           ; Выход из программы.
```

## РЕДАКТИРОВАНИЕ, АССЕМБЛИРОВАНИЕ, СВЯЗЫВАНИЕ И ЗАПУСК (ИЛИ ОТЛАДКА)

Существует множество хороших текстовых редакторов, как бесплатных, так и коммерческих. Следует искать редактор, поддерживающий подсветку синтаксиса для версии ассемблера NASM 64-bit. В большинстве случаев придется скачать и установить некоторый подключаемый модуль (plugin) или дополнительный пакет, обеспечивающий подсветку синтаксиса.

**Примечание.** В этой книге мы будем писать код для версии Netwide Assembler (NASM). Существуют и другие версии ассемблера, например YASM, FASM, GAS или MASM компании Microsoft. И как обычно в мире ИТ, иногда возникают оживленные дискуссии о том, какая версия ассемблера является самой лучшей. В этой книге используется версия NASM, потому что она доступна для ОС Linux, Windows и macOS, а кроме того, из-за наличия большого сообщества пользователей NASM. Справочное руководство по NASM можно найти здесь: [www.nasm.us](http://www.nasm.us).

В этой книге будет использоваться текстовый редактор gedit с установленным дополнительным файлом (модулем) подсветки синтаксиса ассемблера. Gedit – стандартный текстовый редактор, доступный в системе Linux<sup>1</sup>, – здесь используется Ubuntu Desktop 18.04.2 LTS. Файл (модуль) поддержки подсветки синтаксиса можно найти здесь: <https://wiki.gnome.org/action/show/Projects/GtkSourceView/LanguageDefinitions>. Загрузите файл *asm-intel.lang*, скопируйте его в каталог */usr/share/gtksourceview\*/language-specs/*, заменив символ звездочки (\*) на номер версии, установленной в вашей системе. При первом запуске gedit можно выбрать поддерживаемый язык программирования, в нашем случае Assembler (Intel), в нижней части окна gedit.

На экране файл *hello.asm*, приведенный в листинге 1.1, будет выглядеть так, как показано на рис. 1.1.

```

1 ; hello.asm
2 section .data
3     msg db     "hello, world",0
4 section .bss
5 section .text
6     global main
7 main:
8     mov     rax, 1           ; 1 = write
9     mov     rdi, 1           ; 1 = to stdout
10    mov     rsi, msg         ; string to display in rsi
11    mov     rdx, 12          ; length of the string, without 0
12    syscall                ; display the string
13    mov     rax, 60          ; 60 = exit
14    mov     rdi, 0           ; 0 = success exit code
15    syscall                ; quit

```

Рис. 1.1. Содержимое файла *hello.asm* в текстовом редакторе gedit

<sup>1</sup> Автор не совсем точен – в Linux gedit доступен только при наличии установленной рабочей среды GNOME. Для Windows и macOS все необходимое для работы gedit включено в дистрибутив. – Прим. перев.



Согласитесь, с подсветкой синтаксиса исходный код на ассемблере немного проще читать.

При написании ассемблерной программы на экране открыто два окна – окно `gedit`, содержащее исходный код на ассемблере, и окно с приглашением (промптом) командной строки в каталоге проекта, так что можно с легкостью переключаться между редактированием и управлением файлами проекта (выполнять ассемблирование и запускать программу, заниматься отладкой и т. п.). Разумеется, что в более крупных и сложных проектах такой подход вряд ли можно применять – потребуется интегрированная среда разработки (*integrated development environment* – IDE). Но прямо сейчас работы с простым текстовым редактором и интерфейсом командной строки (CLI – *command line interface*) вполне достаточно. Преимуществом этого процесса становится тот факт, что мы можем сосредоточиться на изучении ассемблера, а не многочисленных функциональных возможностей и особенностей IDE. В последующих главах будут рассматриваться полезные инструментальные средства и утилиты, некоторые из которых обладают графическим пользовательским интерфейсом, прочие же связаны с интерфейсом командной строки. Как бы то ни было, описание и использование IDE не относится к тематике этой книги.

Для любого упражнения из данной книги используется отдельный каталог *project*, содержащий все необходимые для проекта и генерируемые файлы.

Разумеется, в дополнение к текстовому редактору необходимо проверить наличие некоторых других установленных инструментальных средств, таких как `gcc`, `GDB`, `make` и `NASM`. Сначала потребуется `gcc` – используемый по умолчанию в Linux компилятор и редактор связей (линкер).

`gcc` – это сокращение от `GNU Compiler Collection` – стандартного инструментального средства компиляции и редактирования связей в Linux. (Аббревиатура `GNU` является рекурсивной: `GNU is Not Unix`. Использование рекурсивных аббревиатур для имен стало общепринятым в среде разработчиков еще в 1970-е гг., и все началось с программистов `LISP`. Да, эти старые шутки уже не смешны.)

В командной строке введите `gcc -v`. Компилятор `gcc` в ответ выведет несколько сообщений, если он уже установлен. Если комплект `gcc` отсутствует, то необходимо установить его с помощью следующей команды:

```
sudo apt install gcc
```

Далее выполните то же самое для `gdb -v` и `make -v`. Если вы не понимаете смысл этих команд, то, прежде чем продолжать чтение, вам необходимо пополнить знания о Linux.

Также необходимо установить `NASM` и пакет `build-essential`, содержащий ряд инструментальных средств, которые будут использоваться в дальнейшем. В `Ubuntu Desktop 18.04` это делается так:

```
sudo apt install build-essential nasm
```

После установки команда `nasm -v` выведет номер версии, если пакет `NASM` был установлен корректно. После установки всех перечисленных выше программных средств вы полностью готовы к реализации своей первой программы на ассемблере.

Введите исходный код программы, показанной в листинге 1.1, в текстовом редакторе, которым предпочитаете пользоваться, сохраните введенный код в файле с именем *hello.asm*. Как уже было отмечено ранее, для сохранения файлов этого первого проекта используется отдельный каталог. Каждая строка кода будет объяснена немного позже в этой главе. Обратите внимание на следующие характеристики исходного кода на ассемблере («исходный код» – это содержимое файла *hello.asm* с программными инструкциями, которые вы только что ввели):

- в исходном коде можно использовать символы табуляции, пробела и перехода на новую строку, чтобы сделать код более удобным для чтения;
- на каждой строке записывается только одна инструкция;
- текст после точки с запятой является комментарием. Другими словами, описанием, предназначенным для чтения человеком. Компьютеры не обращают никакого внимания на комментарии.

В текстовом редакторе создайте еще один файл, содержащий строки, приведенные в листинге 1.2.

**Листинг 1.2.** *makefile* для *hello.asm*

```
#makefile для hello.asm
hello: hello.o
    gcc -o hello hello.o -no-pie
hello.o: hello.asm
    nasm -f elf64 -g -F dwarf hello.asm -l hello.lst
```

На рис. 1.2 показано, как этот файл выглядит в редакторе *gedit*.



```
1 #makefile for hello.asm
2 hello: hello.o
3 gcc -o hello hello.o -no-pie
4 hello.o: hello.asm
5 nasm -f elf64 -g -F dwarf hello.asm -l hello.lst
```

**Рис. 1.2.** Содержимое файла *makefile* в редакторе *gedit*

Сохраните этот файл с именем *makefile* в том же каталоге, где находится файл *hello.asm*, и закройте окно редактора.

Файл *makefile* будет использоваться командой *make* для автоматизации сборки программы. Сборка (*building*) означает проверку исходного кода на ошибки, добавление всех необходимых сервисов операционной системы и преобразование исходного кода в последовательность инструкций, распознаваемых компьютером. В этой книге будут использоваться простые файлы *makefile*. Если вы хотите узнать больше о файлах *makefile*, то справочное руководство находится здесь:

<https://www.gnu.org/software/make/manual/make.html>.

Учебное руководство можно найти здесь:

<https://www.tutorialspoint.com/makefile/>.

Чтобы понять, что именно делает *makefile*, необходимо читать его снизу вверх. Упрощенное описание: утилита *make* работает с деревом зависимостей. Она определяет, что файл программы *hello* зависит от объектного файла *hello.o*. Затем обнаруживается, что файл *hello.o* зависит от файла исходного кода *hello.asm* и что файл *hello.asm* ни от чего не зависит. Далее *make* сравнивает даты последнего изменения файлов *hello.asm* и *hello.o*, и если дата изменения *hello.asm* более поздняя, то *make* выполняет строку после имени *hello.o*, т. е. *hello.asm*. Затем *make* снова начинает процедуру чтения *makefile* и обнаруживает, что дата изменения файла *hello.o* более поздняя, чем дата изменения файла *hello*. Поэтому выполняется строка после имени *hello*, т. е. *hello.o*.

В самой последней строке файла *makefile* NASM используется как ассемблер (программа ассемблирования, т. е. конечного этапа сборки). За ключом *-f* следует формат вывода, в данном случае *elf64*, означающий Executable and Linkable Format for 64-bit (выполняемый и связываемый формат для 64-битовой системы). Ключ *-g* означает, что необходимо включить отладочную информацию в специальном формате отладки, определенном после ключа *-f*. Здесь используется отладочный формат *dwarf*. Похоже, что программисты, разработавшие этот формат, являются большими поклонниками книг «Хоббит» и «Властелин колец» Дж. Р. Р. Толкиена, возможно, именно поэтому они решили, что DWARF (гном) должен стать великолепным дополнением к ELF (эльфу), на всякий случай, если вам это интересно. Если говорить более серьезно, то DWARF – это сокращение от Debug With Arbitrary Record Format<sup>2</sup> (отладка с использованием произвольного формата записей).

STABS – это еще один отладочный формат, не имеющий ничего общего с кровавыми битвами или ярким эльфийским светом (*stab* – ранение, нанесение ран; режущий глаза, ослепительный свет) из романов Толкиена, это название происходит от **S**ymbol **T**able **S**trings (строки таблицы символов). Здесь мы не будем использовать формат STABS, чтобы вы не запутались окончательно.

Ключ *-l* сообщает NASM о необходимости генерации файла листинга *.lst*. Файлы *.lst* будут использоваться для исследования результатов ассемблирования. NASM создает объектный файл с расширением *.o*. В дальнейшем этот объектный файл будет использоваться редактором связей (линкером).

---

**Примечание.** Часто случается так, что NASM выдает несколько непонятных сообщений и отказывается сгенерировать объектный файл. Иногда NASM начинает выдавать такие «жалобы» настолько часто, что может поставить программиста почти на грань безумия. В таких случаях чрезвычайно важно сохранять хладнокровие, выпить очередную чашечку кофе и еще раз внимательно просмотреть исходный код, потому что именно вы сделали что-то неправильно. Ассемблируя свою программу раз за разом, вы будете находить ошибки все быстрее и быстрее.

---

Когда вы наконец убедите NASM принять созданный объектный файл, он будет сразу же обработан редактором связей (линкером). Редактор связей рас-

---

<sup>2</sup> «Википедия» (англ.) дает другое толкование аббревиатуры DWARF – Debug With Attributed Record Format – отладка с использованием формата записей с атрибутами. Смысл несколько иной, но суть дела не меняется. – *Прим. перев.*

сматривает предложенный объектный код и выполняет поиск в системе других необходимых файлов, обычно системных сервисов или прочих объектных файлов. Эти файлы объединяются со сгенерированным объектным кодом, и редактор связей создает выполняемый файл. Разумеется, редактор связей выдаст все возможные сообщения об отсутствующих компонентах и т. п. Если это произошло, выпейте еще одну чашечку кофе и проверьте свой исходный код и содержимое *makefile*.

В рассматриваемом здесь примере используется функциональность GCC (ниже воспроизводятся соответствующие строки из *makefile*):

```
hello: hello.o
    gcc -o hello hello.o -no-pie
```

Последние версии компилятора и линкера GCC по умолчанию генерируют выполняемый код, не зависящий от положения в памяти, или перемещаемый код (position independent executable – PIE). Это делается для защиты от хакеров, исследующих, как память используется программой, что в итоге позволяет им воздействовать на выполнение программы. В рассматриваемом здесь примере не создается перемещаемый выполняемый код, потому что такая программа слишком сложна для анализа (усложнение делается преднамеренно из соображений обеспечения безопасности). Поэтому в *makefile* добавлен ключ `-no-pie`.

В *makefile* можно добавлять комментарии, начинающиеся с символа «решетка» `#`:

```
#makefile for hello.asm
```

Здесь используется GCC для упрощения доступа к функциям стандартной библиотеки C из ассемблерного кода. Иногда мы будем пользоваться функциями языка C, чтобы сделать примеры ассемблерного кода более простыми. Но вы должны знать, что в Linux существует и другой широко известный GNU линкер `ld`.

Если несколько предыдущих абзацев оказались для вас непонятными, не волнуйтесь, выпейте еще чашечку кофе и продолжайте чтение – это всего лишь вспомогательная информация, которая не очень важна на текущем этапе. Просто помните, что *makefile* – ваш друг, который выполняет за вас огромную работу, а единственное, о чем следует беспокоиться сейчас, – не делать собственных ошибок.

В командной строке перейдите в каталог, где сохранены файлы *hello.asm* и *makefile*. Выполните команду `make` для ассемблирования и сборки программы, затем запустите созданную программу, набрав `./hello` в командной строке. Если появилось сообщение `hello, world` перед очередным промптом командной строки, то все работает правильно. Если сообщение не выведено, то в исходном коде была допущена опечатка или какая-то другая ошибка, поэтому необходимо еще раз проверить содержимое файлов *hello.asm* и *makefile*. Наполните очередную чашечку кофе и приступайте к отладке.

На рис. 1.3 показан пример вывода ожидаемого сообщения на экран.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

[e-Univers.ru](http://e-Univers.ru)