Оглавление

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**, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **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/** или напишите в издательство по адресу **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**, 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**.

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

Об авторе

Даниэль Куссвюрм более 30 лет занимается разработкой программного обеспечения и информационными технологиями. За свою карьеру он разработал инновационное программное обеспечение для различного медицинского и научного оборудования и приложения для обработки изображений. Во многих из этих проектов он успешно использовал язык ассемблера x86 для значительного повышения быстродействия ресурсоемких алгоритмов и решения уникальных задач программирования. Он получил степень бакалавра электроники и электротехники в Университете Северного Иллинойса, а также степень магистра и доктора информатики в Университете ДеПола.

О техническом редакторе

Пол Коэн пришел в корпорацию Intel на самых первых этапах развития архитектуры x86, начиная с 8086, и ушел из Intel после 26 лет работы в сфере продаж, маркетинга и управления. В настоящее время он является партнером Douglas Technology Group и специализируется на написании технической литературы по заказу Intel и других корпораций. Пол также совместно с Академией молодых предпринимателей (YEA) ведет учебный курс, который превращает учеников средних и старших классов в настоящих, уверенных в себе предпринимателей. Он также является комиссаром по вопросам дорожного движения города Бивертон, штат Орегон, и входит в совет директоров нескольких некоммерческих организаций.

Благодарности

Производство фильма и издание книги в чем-то похожи. Трейлеры к фильму чествуют актеров, играющих главных героев. На обложке книги звучат имена авторов. Актеры и авторы в конечном итоге получают признание публики за свои усилия. Однако невозможно создать фильм или опубликовать книгу без преданности делу, опыта и творческого подхода профессиональной закадровой команды. Эта книга не исключение.

Я хочу поблагодарить за прекрасную работу талантливый редакционный коллектив Apress, особенно Стива Энглина, Марка Пауэрса и Мэтью Муди. Пол Коэн заслуживает отдельной похвалы за его тщательный технический обзор и практические предложения. Корректор Эд Куссвюрм заслуживает аплодисментов и признательности за его упорный труд и конструктивные отзывы. Я беру на себя полную ответственность за любые оставшиеся недостатки.

Я также хочу поблагодарить Нирмала Селвараджа, Дулси Нирмала, Кезию Эндсли, Дханеша Кумара и весь редакционный персонал Apress за их вклад, а моих профессиональных коллег за их одобрение и поддержку. Наконец, я хотел бы поблагодарить родителей Армина (вечная ему память) и Мэри, а также двоюродную сестру Мэри и двоюродных братьев Тома, Эда и Джона, которые вдохновляли меня на написание этой книги.

Вступление

С момента изобретения персонального компьютера (ПК) разработчики программного обеспечения использовали язык ассемблера x86 для создания инновационных решений применительно к широкому спектру алгоритмических задач. В первые дни эры ПК было обычной практикой писать большие фрагменты программного кода или полные приложения с использованием языка ассемблера x86. Учитывая преобладание в XXI веке языков высокого уровня, таких как C++, C#, Java и Python, может быть удивительно узнать, что многие разработчики программного обеспечения все еще используют язык ассемблера для кодирования критически важных в плане быстродействия участков своих программ. И хотя с годами компиляторы заметно улучшились с точки зрения генерации машинного кода, который стал эффективнее по объему и производительности, все еще остаются ситуации, когда разработчику программного обеспечения имеет смысл использовать преимущества программирования на языке ассемблера.

Архитектура современных процессоров x86 *с одним потоком команд и множеством потоков данных* (SIMD, single instruction stream – multiple data stream) является еще одним объяснением постоянного интереса к программированию на языке ассемблера. Процессор с поддержкой SIMD обладает вычислительными ресурсами, которые упрощают параллельные вычисления с использованием нескольких значений данных, что может значительно повысить быстродействие приложений, обеспечивающих высокую скорость отклика в реальном времени. Архитектуры SIMD также хорошо подходят для ресурсоемких проблемных областей, таких как обработка изображений, кодирование аудио и видео, автоматизированное проектирование, компьютерная графика и интеллектуальный анализ данных. К сожалению, многие языки высокого уровня и инструменты разработки по-прежнему не могут полностью или хотя бы частично использовать возможности SIMD современного процессора x86. С другой стороны, язык ассемблера открывает разработчику программного обеспечения полный доступ к ресурсам SIMD процессора.

О чем эта книга

Эта книга рассказывает о программировании на 64-битном (x86-64) языке ассемблера x86. Содержание и структура книги призваны помочь вам быстро освоить программирование на языке ассемблера x86-64 и вычислительные ресурсы векторных расширений набора команд Advanced Vector Extensions (AVX)[1]. Она также содержит множество примеров исходного кода, которые способствуют ускоренному изучению и пониманию основных конструкций языка ассемблера x86-64 и концепций программирования SIMD. После прочтения этой книги вы сможете кодировать быстродействующие функции и алгоритмы с помощью языка ассемблера x86-64 и наборов инструкций AVX, AVX2 и AVX-512.

Прежде чем продолжить, я должен обратить ваше внимание, что *в этой книге не рассматривается программирование на языке ассемблера x86-32*. В нем также не обсуждаются устаревшие технологии x86, такие как модуль с плавающей запятой x87, MMX и Streaming SIMD Extensions. Если вы заинтересованы в изучении этих тем, прочтите первое издание книги (в переводе не издавалось – *прим. перев.*). Эта книга не объясняет архитектурные особенности x86 или привилегированные команды процессора, которые используются в операционных системах. Однако вам все равно придется досконально изучить материал, представленный в данной книге, чтобы разрабатывать код на языке ассемблера x86 для использования в операционной системе.

Хотя теоретически возможно написать целую прикладную программу, используя только язык ассемблера, высокие требования к разработке современного программного обеспечения делают такой подход непрактичным и нецелесообразным. Вместо этого в данной книге основное внимание уделяется кодированию функций языка ассемблера x86-64, вызываемых из C++. Каждый пример исходного кода был создан с использованием Microsoft Visual Studio C++ и Microsoft Macro Assembler (MASM).

Целевая аудитория

Целевая аудитория этой книги – разработчики программного обеспечения, в том числе:

- разработчики, которые создают прикладные программы для платформ на базе Windows и хотят научиться писать алгоритмы и функции высокого быстродействия с использованием языка ассемблера x86-64;
- разработчики, которые создают прикладные программы для сред, отличных от Windows, и хотят изучить программирование на языке ассемблера x86-64;

[1] Расширение системы команд x86 для микропроцессоров Intel и AMD, предложенное Intel в марте 2008. – *Прим. перев.*

- разработчики, которые хотят научиться создавать вычислительные функции SIMD с использованием наборов команд AVX, AVX2 и AVX-512;
- разработчики и студенты, изучающие информатику, которые хотят добиться глубокого понимания платформы x86-64 и ее архитектуры SIMD.

Основная аудитория данной книги – это разработчики программного обеспечения на базе Windows, поскольку примеры исходного кода были разработаны с использованием Visual Studio C++ и MASM. Разработчики, ориентированные на платформы, отличные от Windows, также могут извлечь пользу из этой книги, поскольку большая часть информативного контента излагается независимо от какой-либо конкретной операционной системы. Предполагается, что читатели данной книги уже имеют опыт программирования на языках высокого уровня и базовые знания C++. Знакомство с программированием в Visual Studio или Windows PowerShell не требуется.

Обзор содержания

Основная цель этой книги – помочь вам изучить программирование на 64-битном языке ассемблера x86, а также набор команд AVX, AVX2 и AVX-512. Главы и содержание книги построены таким образом, чтобы достичь этой цели. Ознакомьтесь с кратким обзором того, что вы найдете в этой книге.

В главе 1 рассматривается основная архитектура платформы x86-64. Она включает в себя описание основных типов данных платформы, внутренней архитектуры, наборов регистров, операндов команд и режимов адресации памяти. В этой главе также описывается основной набор команд x86-64. В главах 2 и 3 объясняются основы программирования на языке ассемблера x86-64 с использованием базового набора команд и общих программных конструкций, включая массивы и структуры. Примеры исходного кода, представленные в этих (и последующих) главах, оформлены как рабочие программы, которые в процессе обучения вы можете запускать, изменять или иным образом экспериментировать с кодом.

Глава 4 посвящена архитектурным ресурсам AVX, включая наборы регистров, типы данных и набор команд. В главе 5 объясняется, как использовать набор команд AVX для выполнения скалярных арифметических операций с плавающей запятой, со значениями как с одинарной, так и с двойной точностью. В главах 6 и 7 рассказано про программирование AVX SIMD с использованием упакованных операндов, как целочисленных, так и с плавающей запятой.

Глава 8 знакомит с AVX2 и исследует его расширенные возможности, включая широковещательную передачу, сбор и перестановку данных. Здесь также объясняются операции *слитного умножения-сложения* (FMA, fused multiply-add). Главы 9 и 10 содержат примеры исходного кода, которые иллюстрируют различные вычислительные алгоритмы, использующие AVX2 с упакованными операндами. Глава 11 включает примеры исходного кода, демонстрирующие программирование FMA. В этой главе также рассмотрены примеры, иллюстрирующие недавние расширения платформы x86 с использованием регистров общего назначения.

В главе 12 детально рассмотрен набор команд AVX-512. В этой главе описаны наборы регистров и типы данных AVX-512. В ней также рассмотрены основные усовершенствования AVX-512, включая условное выполнение и слияние, встроенные широковещательные операции и округление на уровне команд.

В главах 13 и 14 содержится множество примеров исходного кода, демонстрирующих, как использовать эти расширенные функции.

В главе 15 представлен обзор современного многоядерного процессора x86 и лежащей в его основе микроархитектуры. В этой главе также описаны конкретные стратегии и методы программирования, которые можно использовать для повышения быстродействия кода на языке ассемблера x86. В главе 16 приведен обзор нескольких примеров исходного кода, иллюстрирующего передовые методы программирования на языке ассемблера x86, включая обнаружение функций процессора, ускоренный доступ к памяти и многопоточные вычисления.

В приложении A рассказано, как выполнить примеры исходного кода с помощью Visual Studio и MASM. Оно также содержит список ссылок и ресурсов, к которым вы можете обратиться для получения дополнительной информации о программировании на языке ассемблера x86.

Исходный код примеров

Примеры исходного кода этой книги доступны на веб-сайте Apress по адресу https://www.apress.com/us/book/9781484240625. Для каждой главы есть ZIP-архив, содержащий файлы исходного кода C++ и ассемблера, а также файлы проекта Visual Studio. Процедура установки не требуется. Вы можете просто извлечь содержимое ZIP-архива главы в папку по вашему выбору.

Внимание! Единственное назначение исходного кода – показать примеры программирования, которые напрямую связаны с темами, обсуждаемыми в этой книге. В этих примерах уделяется минимальное внимание важным проблемам разработки программного обеспечения, таким как надежная обработка ошибок, риски безопасности, вычислительная стабильность, ошибки округления или плохо согласованные функции. Вы принимаете на себя всю ответственность за решение этих проблем, если решите использовать исходный код каких-либо примеров в своих собственных программах.

Примеры исходного кода были созданы с помощью Visual Studio Professional 2017 (версия 15.7.1) на ПК с 64-разрядной Windows 10 Pro. На веб-сайте Visual Studio (https://visualstudio.microsoft.com) содержится дополнительная информация об этом и других выпусках Visual Studio. Технические сведения об установке, настройке и разработке приложений Visual Studio доступны по адресу https://docs.microsoft.com/en-us/visualstudio/?view=vs-2017.

Рекомендуемой аппаратной платформой для запуска примеров исходного кода является ПК на базе x86 с 64-разрядной ОС Windows 10 и процессором, поддерживающим AVX. Для запуска примеров исходного кода, в которых используются эти наборы команд, требуется процессор, совместимый с AVX2 или AVX-512. Вы можете использовать одну из свободно доступных утилит, перечисленных в приложении, чтобы определить, какие расширения набора команд x86-AVX поддерживает процессор вашего компьютера.

Глава 1

Архитектура ядра x86-64

В этой главе архитектура ядра x86-64 рассматривается с точки зрения прикладной программы. Она открывается кратким историческим обзором платформы x86, формирующим основу для понимания последующего материала. Затем следует обзор основных типов данных – числовых и SIMD. Далее рассмотрена архитектура ядра x86-64, включая описания наборов регистров процессора, флагов состояния, операндов команд и режимов адресации памяти. Глава завершается обзором набора команд ядра x86-64.

В отличие от языков высокого уровня, таких как C и C++, программирование на языке ассемблера требует от разработчика глубокого понимания конкретных архитектурных особенностей целевого процессора. Темы, обсуждаемые в данной главе, соответствуют этому требованию и заложат основу для понимания примеров кода, которые вы встретите далее в данной книге. В этой главе также представлен базовый материал, необходимый для понимания расширенных возможностей SIMD x86-64.

1.1. Исторический обзор

Прежде чем исследовать технические детали архитектуры ядра x86-64, полезно понять, как эта архитектура развивалась за минувшие годы. Следующий краткий обзор посвящен заслуживающим внимания усовершенствованиям процессоров и наборов команд, которые повлияли на то, как разработчики программного обеспечения используют язык ассемблера x86. Читатели, которым интересна более полная история происхождения x86, должны обратиться к ресурсам, перечисленным в приложении.

Платформа процессора x86-64 является расширением исходной платформы x86-32. Первым воплощением платформы x86-32 в кремнии был микропроцессор Intel 80386, представленный в 1985 году. Модель 80386 расширила архитектуру 16-битной 80286 за счет добавления 32-битных регистров и типов данных, режимов плоской модели памяти, 4 ГБ логического адресного пространства и страничной организации памяти. Процессор 80486 превзошел производительность 80386 за счет реализации кешей памяти на кристалле и оптимизированных команд. В отличие от процессора 80386, который нуждался во внешнем сопроцессоре 80387 (FPU, floating-point unit) для вычислений с плавающей запятой, большинство версий процессора 80486 также включали интегрированный модуль x87 FPU.

Расширение платформы x86-32 продолжилось выпуском в 1993 году первого процессора марки Pentium, представителя микроархитектуры P5. Улучшения производительности включали конвейер выполнения сдвоенных команд, 64-битную внешнюю шину данных и отдельные кеши памяти на кристалле как для кода, так и для данных. Более поздние версии (1997 г.) микроархитектуры P5 получили новый вычислительный ресурс, так называемую технологию MMX, которая поддерживает операции SIMD над упакованными целыми числами с использованием регистров разрядностью 64 бита. *Упакованное целое число* (packed integer) – это совокупность нескольких целочисленных значений, которые обрабатываются одновременно.

Микроархитектура P6, впервые использованная в Pentium Pro (1995 г.), а затем в Pentium II (1997 г.), расширила платформу x86-32 за счет *трехстороннего суперскалярного конвейера*. Это означает, что процессор может (в среднем) декодировать, отправлять и выполнять три отдельные команды в течение каждого тактового цикла. Другие усовершенствования P6 включали выполнение команд вне очереди, улучшенные алгоритмы предсказания ветвлений и упреждающего исполнения команд. Pentium III, также основанный на микроархитектуре P6, был выпущен в 1999 году и получил поддержку новой технологии SIMD под названием Streaming SIMD extension (SSE). Эта технология добавляет к платформе x86-32 восемь 128-битных регистров и команды, которые реализуют упакованную арифметику с плавающей запятой одинарной точности.

В 2000 году Intel представила новую микроархитектуру Netburst с технологией SSE2, которая расширила возможности SSE с плавающей запятой за счет обработки упакованных значений двойной точности. SSE2 также включала дополнительные команды, позволяющие использовать 128-битные регистры SSE для упакованных целочисленных вычислений и скалярных операций с плавающей запятой. Линейка процессоров, основанных на архитектуре Netburst, включала несколько вариантов Pentium 4. В 2004 году микроархитектура Netburst была модернизирована и теперь включает в себя SSE3 и технологию Hyper-Threading. SSE3 содержит новые команды операций с упакованными целыми числами и числами с плавающей запятой для платформы x86, в то время как технология Hyper-Threading распараллеливает конвейеры команд внешнего интерфейса процессора для повышения производительности. Процессоры с поддержкой SSE3 входят в 90-нм (и менее) версии линейки продуктов Pentium 4 и Xeon.

В 2006 году Intel запустила в массовое производство новую микроархитектуру под названием Core. Микроархитектура Core стала следствием глубокой модернизации многих интерфейсных конвейеров и исполнительных модулей Netburst с целью повышения производительности и снижения энергопотребления. Она также содержит ряд улучшений SIMD, включая SSSE3 и SSE4.1. Эти расширения добавили на платформу новые команды для работы с упакованными целыми числами и числами с плавающей запятой, но не добавили новых регистров или типов данных. К процессорам на основе микроархитектуры Core относятся процессоры серий Core 2 Duo, Core 2 Quad и Xeon 3000/5000.

Микроархитектура под названием Nehalem последовала за Core в конце 2008 года. Эта микроархитектура вернула платформе x86 гиперпоточность,

которая была исключена из микроархитектуры Core. Микроархитектура Nehalem также поддерживает SSE4.2. Это последнее усовершенствование x86-SSE добавило в набор команд x86-SSE несколько команд ускорителя для определенных приложений. SSE4.2 также содержит новые команды, упрощающие обработку текстовых строк с использованием 128-битных регистров x86-SSE. К процессорам на основе микроархитектуры Nehalem относятся процессоры Core i3, i5 и i7 первого поколения. Она также включает процессоры серий Xeon 3000, 5000 и 7000.

В 2011 году Intel запустила новую микроархитектуру под названием Sandy Bridge. В микроархитектуре Sandy Bridge появилась новая технология SIMD x86 под названием Advanced Vector Extensions (AVX). AVX поддерживает операции над упакованными числами с плавающей запятой (как одинарной, так и двойной точности) с использованием регистров шириной 256 бит. AVX также поддерживает новый синтаксис команд с тремя операндами, повышающий эффективность кода за счет уменьшения количества передач данных из регистра в регистр, которые должна выполнять функция программы. Процессоры на базе микроархитектуры Sandy Bridge включают в себя процессоры Core i3, i5 и i7 второго и третьего поколений, а также процессоры серии Xeon V2.

В 2013 году Intel представила свою микроархитектуру Haswell. Haswell поддерживает набор команд AVX2, который расширяет набор AVX командами поддержки операций с упакованными целыми числами с использованием регистров шириной 256 бит. AVX2 также поддерживает расширенные возможности передачи данных с помощью команд широковещательной передачи, сбора и перестановки. (Команды широковещательной передачи реплицируют одно значение в несколько мест; команды сбора данных загружают несколько элементов из несмежных ячеек памяти; команды перестановки переупорядочивают элементы упакованного операнда.) Другой особенностью микроархитектуры Haswell является включение в нее операции *слитного умножения-сложения* (FMA, fused multiply-add). FMA позволяет программным алгоритмам выполнять вычисления умножения-сложения (или скалярного произведения) с использованием одной операции округления с плавающей запятой, что помогает улучшить как быстродействие, так и точность. Микроархитектура Haswell также включает в себя несколько новых команд для работы с регистрами общего назначения. К процессорам на основе микроархитектуры Haswell относятся процессоры Core i3, i5 и i7 четвертого поколения. AVX2 поддерживают и более поздние поколения процессоров семейства Core, а также процессоры серий Xeon V3, V4 и V5.

Доработки платформы x86 не ограничивались усовершенствованиями SIMD. В 2003 году AMD представила свой процессор Opteron, который расширил исполняющее ядро x86 с 32 до 64 бит. Intel последовала примеру AMD в 2004 году, добавив практически такие же 64-разрядные расширения к своим процессорам, начиная с определенных версий Pentium 4. Все процессоры Intel на базе микроархитектур Core, Nehalem, Sandy Bridge, Haswell и Skylake поддерживают исполняющее ядро x86-64.

Процессоры AMD также претерпели изменения за прошедшие годы. В 2003 году AMD представила серию процессоров на базе своей микроархитектуры

K8. Оригинальные версии K8 включали поддержку MMX, SSE и SSE2, в то время как в более поздних версиях появилась поддержка SSE3. В 2007 году была запущена микроархитектура K10, которая включала расширение SIMD под названием SSE4a. Расширение SSE4a содержит несколько команд по сдвигу маски и хранению потоковых данных, которые не поддерживают процессоры Intel. Вслед за K10 в 2011 году AMD представила новую микроархитектуру Bulldozer. Микроархитектура Bulldozer включает поддержку SSSE3, SSE4.1, SSE4.2, SSE4a и AVX. Она также поддерживает FMA4 – версию слитного умножения-сложения с четырьмя операндами. Процессоры Intel не поддерживают команды FMA4. Обновление 2012 года для микроархитектуры Bulldozer под названием Piledriver включает поддержку как FMA4, так и трехоперандной версии FMA, которую некоторые утилиты для обнаружения функций ЦП и сторонние документы именуют FMA3. Самая последняя микроархитектура AMD, представленная в 2017 году, называется Zen (в 2018 году была представлена микроархитектура Zen 2 – *прим. перев.*). Эта микроархитектура включает усовершенствования набора команд AVX2 и используется в процессорах серии Ryzen.

Высокопроизводительные процессоры для настольных и серверных систем на основе микроархитектуры Intel Skylake-X, также впервые поступившие на рынок в 2017 году, включают новое расширение SIMD под названием AVX-512. Эта усовершенствованная архитектура поддерживает сжатые целые числа и операции с плавающей запятой с использованием регистров шириной 512 бит. AVX-512 также поддерживает дополнения архитектуры, которые упрощают условное слияние данных на уровне команд, управление округлением с плавающей запятой и широковещательные операции. Ожидается, что в ближайшие несколько лет и AMD, и Intel включат AVX-512 в свои основные процессоры для настольных ПК и ноутбуков.

1.2. Типы данных

Программы, написанные на ассемблере x86, могут использовать самые разные типы данных. Большинство типов данных программ происходят из небольшого набора основных типов данных, которые присущи платформе x86. Эти основные типы данных позволяют процессору выполнять числовые и логические операции с использованием целых чисел со знаком и без знака, значений с плавающей запятой одинарной (32-битные) и двойной (64-битные) точности, текстовых строк и значений SIMD. В этом разделе вы узнаете об основных типах данных, а также о нескольких различных типах данных, поддерживаемых x86.

1.2.1. Основные типы данных

Основной тип данных – это элементарная единица данных, которая обрабатывается процессором во время выполнения программы. Платформа x86 поддерживает основные типы данных с разрядностью от 8 бит (1 байт) до 128 бит (16 байт). В табл. 1.1 перечислены эти типы и обычные способы их использования.

Таблица 1.1. Основные типы данных

Тип данных	Длина (биты)	Типичное применение
Байт (byte)	8	Символы, небольшие целые числа
Слово (word)	16	Символы, целые числа
Двойное слово (doubleword)	32	Числа – целые и с плавающей запятой (одинарной точности)
Четверное слово (quadword)	64	Числа – целые и с плавающей запятой (двойной точности)
Двойное четверное слово (double quadword)	128	Упакованные целые числа, упакованные числа с плавающей запятой

Неудивительно, что размер основных типов данных зависит от целых степеней двойки. Разряды основного типа данных нумеруются справа налево, начиная с нуля и до значения `размер-1`, обозначающих младший и старший разряды числа соответственно. Основные типы данных размером более одного байта хранятся в последовательных ячейках памяти, начиная с младшего байта по наименьшему адресу памяти. Этот тип расположения байтов в памяти называется *прямым порядком байтов* (little endian). На рис. 1.1 показаны схемы нумерации битов и порядок байтов, которые применяются с основными типами данных.

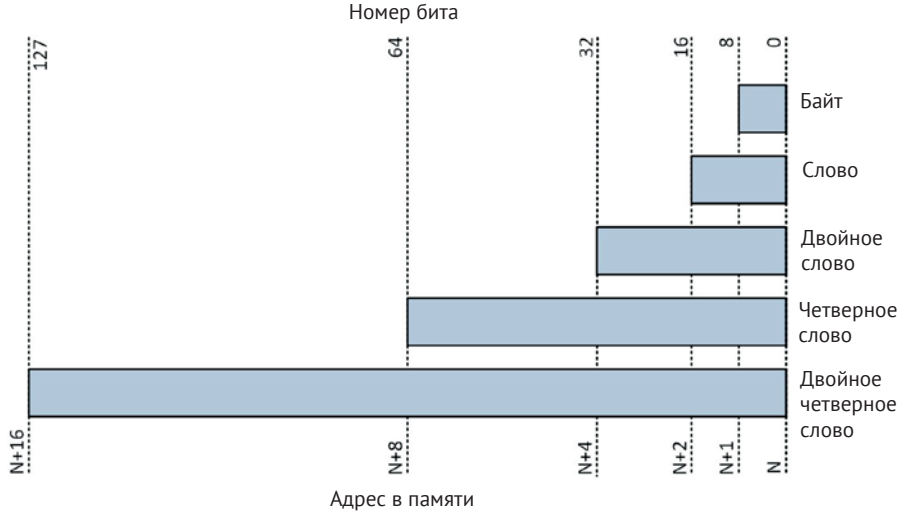

Рис. 1.1. Нумерация битов и порядок байтов для основных типов данных

Правильно выровненный основной тип данных – это тот, адрес которого без остатка делится на его размер в байтах. Например, двойное слово правильно выровнено, когда оно хранится в ячейке памяти с адресом, который делится на четыре без остатка. Точно так же четверные слова правильно выровнены по адресам, делящимся на восемь. Если это специально не оговорено требованиями

операционной системы, процессор x86 не требует правильного выравнивания многобайтовых типов данных в памяти. Однако стандартной практикой является правильное выравнивание всех многобайтовых значений, когда это возможно, чтобы избежать потенциальных потерь быстродействия, которые могут возникнуть, если процессору требуется доступ к смещенным в памяти данным.

1.2.2. Числовые типы данных

Числовой тип данных – это элементарное скалярное значение, такое как целое число или число с плавающей запятой. Все числовые типы данных, распознаваемые ЦП, представлены с использованием одного из основных типов данных, рассмотренных в предыдущем разделе. Таблица 1.2 содержит список числовых типов данных x86 вместе с соответствующими типами C/C++. Эта таблица также включает типы фиксированного размера, которые определены в заголовочном файле C++ `<cstdint>` (см. http://www.cplusplus.com/reference/cstdint/ для получения дополнительной информации об этом заголовочном файле). Набор команд x86-64 поддерживает арифметические и логические операции с использованием 8-, 16-, 32- и 64-битных целых чисел, как со знаком, так и без знака. Он также поддерживает арифметические вычисления и операции манипулирования данными с использованием значений с плавающей запятой одинарной и двойной точности.

Таблица 1.2. Числовые типы данных x86

Тип	Длина (биты)	Тип C/C++	`<cstdint>`
Целое со знаком	8	`char`	`int8_t`
	16	`short`	`int16_t`
	32	`int, long`	`int32_t`
	64	`long long`	`int64_t`
Беззнаковое целое	8	`unsigned char`	`uint8_t`
	16	`unsigned short`	`uint16_t`
	32	`unsigned int, unsigned long`	`uint32_t`
	64	`unsigned long long`	`uint64_t`
С плавающей запятой	32	`float`	Не применимо
	64	`double`	Не применимо

1.2.3. Типы данных SIMD

Тип данных SIMD – это последовательный набор байтов, используемый процессором для выполнения операций или вычислений, в которых участвуют несколько значений. Тип данных SIMD можно рассматривать как *объект-контейнер*, который содержит несколько экземпляров одного и того же основного типа данных (например, байты, слова, двойные слова или четверные слова). Как и в случае основных типов данных, разряды данных SIMD пронумерованы