ИНТУИТ

НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Основы информационных технологий

Д. В. Луцив, Р. Н. Мокаев, Н. О. Гаранина, Д. В. Кознов

АРХИТЕКТУРА ЭВМ

Учебное пособие

М о с к в а

Национальный Открытый Университет «ИНТУИТ»

2 0 2 2

УДК 004.2
ББК 32.971.3
А87

Луцив Д. В., Мокаев Р. Н., Гаранина Н. О., Кознов Д. В.
А87 Архитектура ЭВМ: учебное пособие / Д. В. Луцив, Р. Н. Мокаев, Н. О. Гаранина, Д. В. Кознов. — М.: ООО «ИНТУИТ.РУ», 2022. — 152 с., ил. (Серия «Основы информационных технологий»).

ISBN 978-5-9556-0201-1

Предлагаемое учебное пособие посвящено архитектуре ЭВМ, включая историю развития ЭВМ, архитектуру фон Неймана, двоичную систему исчисления и машинную арифметику, устройство печатных плат и интегральных микросхем, основы системотехники, устройство процессоров, введение в операционные системы и параллельные вычисления и пр.

Данное пособие основывается на учебных курсах, которые авторы читали последние 10 лет для бакалавров математико-механического факультета Санкт-Петербургского государственного университета. Пособие предназначается для студентов различных учебных программ, в которых информатика и программная инженерия не являются основными темами, в частности, для студентов кафедры прикладной кибернетики СПбГУ. На практике оказывается, что выпускники самых разных специальностей идут сегодня работать программистами. Следовательно, если в рамках соответствующих учебных программ есть курсы по информатике, то должен быть и такой базовый курс по архитектуре ЭВМ. Данное пособие может быть использовано, правда, с некоторыми купюрами, также студентами бакалавриата и преподавателями учебных программ, непосредственно нацеленных на подготовку программистов.

УДК 004.2
ББК 32.971.3

ISBN 978-5-9556-0201-1

Оглавление

Введение

Курс «Архитектура ЭВМ» посвящен принципам организации вычислительных систем и является одним из базовых при обучении программированию. Обычно в этом курсе рассматривается устройство современных ЭВМ, история развития вычислительной техники и современные тенденции, принципы организации современных процессоров, а также булева алгебра и логические схемы и пр. Данная тематика соответствует курсу CS220 из рекомендаций ACM/IEEE Curricula in Software Engineering. В настоящий момент существует много вариантов курса в виде книг, учебных пособий и электронных курсов. Широко известна книга Таненбаум Э., Остин Т. «Архитектура компьютера». На известном учебном электронном ресурсе Coursera пользуется популярностью курс Computer Architecture Дэвида Ветцлава (David Wentzlaff) из университета Принстон (США). Подобные курсы имеются и на других образовательных online-платформах: например, на платформе Stepic можно найти курс Кирилла Кринкина из ЛЭТИ «Введение в архитектуру ЭВМ. Элементы операционных систем». В целом только в России за последние 30 лет опубликовано более полусотни учебных и методических пособий по архитектуре ЭВМ. Отметим среди них курс Хорошевского В. Г. «Архитектура вычислительных систем» и Воеводина В. В. «Архитектура ЭВМ и численные методы». Курсы по архитектуре ЭВМ продолжают появляться и в настоящее время — см., например, Новожилов О. П. «Архитектура ЭВМ и систем» 2019, ВШЭ; Степина В. В. «Архитектура ЭВМ и вычислительные системы», М., 2017.

Причиной популярности данной тематики является то, что она включает базовые компьютерные знания, которые востребованы различными аудиториями — от будущих профессиональных программистов до учащихся гуманитарных специальностей, которым необходимы общие сведения об устройстве современных компьютеров. Соответственно, курсы по этой теме варьируются от узкоспециализированных, сопровождающихся различными практикумами, до ознакомительных, содержащих лишь обзор отдельных тем. Некоторые из курсов делают акцент на низкоуровневом программировании (Ассемблер, C), некоторые захватывают операционные системы, компьютерные сети и информационные системы, а некоторые делают акцент на численных методах и параллельных вычислениях. Также для курса оказывается важным предварительный опыт и знания студентов — например, знания математической логики и теории алгоритмов.

Предлагаемый курс по архитектуре ЭВМ читался авторами в течение ряда лет на математико-механическом факультете СПбГУ на кафедре прикладной кибернетики. Слушателями курса были прикладные математики, у которых программирование является непрофильной дисциплиной, однако наряду с математикой оно активно преподается. Мы предполагаем, что студенты владеют навыками программирования на языках C/C++, а также прослушали курс по представлению данных и алгоритмам (то есть знают, что такое машина Тьюринга, стек, очередь, массив и т. д.). Основной задачей курса является ознакомить слушателей с основами устройства компьютера — архитектурой фон Неймана, центральным процессором, а также дать обзор основных узлов современного ПК. Мы также сделали упор на процесс исполнения прикладной программы на ЭВМ и постарались осветить все необходимые аспекты этого процесса, включая компиляцию и функции операционной системы. Базовый язык программирования, который мы использовали в этом курсе, — это C/C++, поскольку он широко известен и сочетает в себе как высокоуровневые средства — классы, наследование и т. д., так и низкоуровневые возможности, в частности работу с динамической памятью. Следует отметить, что мы не старались исчерпывающе объяснить все детали исполнения программы иллюстрациями на C/C++, вынеся часть материала в упражнения и надеясь на самостоятельную активность студентов. В качестве примеров мы в течение всего курса рассматриваем архитектуру процессоров Intel x86, иногда сравнивая её с другими. Отметим также, что мы старались снабдить каждую лекцию детальными вопросами про основные понятия и элементы курса. Эти вопросы студенты могут использовать для самоконтроля, а преподаватели — для проверки студентов. В вопросы к лекциям мы также включили ряд упражнений для того, чтобы у студентов была возможность создать более сложные и интересные отношения с материалом курса, нежели простое знание/понимание. Также в рамках курса мы дали краткий обзор математической логики и схемотехники (то есть знание студентами математической логики не предполагается). В целом, мы стремились составить у студентов связное представление о предмете и не делали акцента на каких-либо специализированных знаниях.

Дмитрий Кознов

Лекция 1. История развития и поколения ЭВМ

Арифмометры докомпьютерной эпохи. Первые ЭВМ — ABC, Z3, Colossus, Mark I; четыре поколения ЭВМ; обзор отечественных ЭВМ.

Вычислительные устройства докомпьютерной эпохи

Счётные устройства, способные автоматически выполнять сложные математические вычисления, конструировались математиками и инженерами на протяжении многих столетий. Первые механические вычислители, выполнявшие арифметические действия, были созданы в середине XVII века математиками Блезом Паскалем и Готфридом Лейбницем. В середине XIX века Чарльз Бэббидж в Кембриджском университете создал разностную машину, которая позволяла выполнять фиксированный набор вычислительных алгоритмов. Им же была начата, но не завершена, постройка первой *аналитической* машины, на которой предполагалось программировать при помощи перфокарт.

Известный санкт-петербургский математик Пафнутий Чебышёв в XIX веке также конструировал ряд механических вычислительных устройств, которые позволяли складывать и умножать числа. К концу XIX века подобные устройства стали массово выпускаться и назывались *арифмометрами*. Одним из самых удачных приборов такого рода оказался арифмометр российского механика Вильгодта Однера, позже выпускавшийся в СССР под маркой «Феликс».

Ранние ЭВМ

В 30-х — первой половине 40-х годов XX века в ряде стран начали активно разрабатываться программируемые электронно-механические вычислительные устройства. В 1939 году американскими учёными Джоном Атанасовым и Клиффордом Берри была спроектирована первая ЭВМ ABC (Atanasoff-Berry Computer) — цифровое устройство, которое не имело механических движущихся составных частей. Пальму первенства этой ЭВМ присудил Федеральный районный суд США в 1973 году. В ABC впервые появилась двоичная арифметика и элемент электронных схем под названием *триггеры*. ABC пред-

назначалась для решения систем линейных уравнений и не была программируемой, поскольку не позволяла хранить программу в памяти и, таким образом, требовала для своей работы активного участия человека.

В 1941 году немецкий инженер Конрад Цузе создал ЭВМ под названием Z3. Она, так же как и ABC, работала с двоичными данными и использовала двоичные электронные схемы на основе телефонных *электромеханических реле*. Но, в отличие от ABC, данная ЭВМ была уже программируемой, то есть могла выполнять составленные заранее программы, а не требовала ввода каждой отдельной команды человеком. Для хранения программ Z3 использовала перфорированную ленту, то есть внешний носитель. Однако условные переходы и циклы в машинном языке Z3 отсутствовали — там, где они требовались в программе, было необходимо вмешательство оператора. Данная ЭВМ использовалась для расчётов в области самолётостроения и управления ракетами. Единственный экземпляр Z3 был уничтожен при бомбёжке Берлина в 1945 году.

В 1943 году в Великобритании была создана ЭВМ под названием Colossus. Основным элементом для реализации двоичных электронных схем впервые использовались электронные лампы. В 1944 году была выпущена усовершенствованная версия ЭВМ, названная Colossus Mark II. Эта ЭВМ предназначалась для расшифровки перехваченных немецких зашифрованных радиосообщений, получаемых из радиоэфира, — с помощью этих радиограмм немецкое командование управляло действиями различных войсковых соединений и координировало их. Colossus позволила сократить время расшифровки сообщений с недель до часов. После Второй мировой войны почти все экземпляры данной ЭВМ, а также её чертежи были уничтожены, а секретность с проекта была снята лишь в 2000 году. Примерно в это же время Colossus был восстановлен, и оказалось, что скорость его работы совпадает со скоростью работы компьютера с процессором Pentium 2 — но, разумеется, лишь при решении задач дешифровки данных.

В 1944 году, по заказу Военно-морского флота США, компания IBM создала ЭВМ под названием Mark I для использования в военных расчётах. Кроме того, Mark I применялась в Манхэттенском проекте по созданию в США атомной бомбы. Первая версия компьютера была установлена в Гарвардском университете. Разработкой руководил капитан второго ранга Говард Эйкен. При изготовлении Mark I были использованы двоичные электронные схемы на основе электромеханического реле, ЭВМ весила 4,5 тонны и имела корпус из нержавеющей стали. Данная ЭВМ была программируемой, считывала программу с бумажной перфорированной ленты, но не поддерживала циклов и условных предложений. Циклы реализовывались с помощью замыкания начала и конца перфорированной ленты с командами программы. Mark I впервые реализовала концепцию раздельного хранения программы и данных, и эта идея впоследствии получила название «гарвардской архитектуры». Также следует отметить, что ЭВМ Mark I была первой полностью программируемой ЭВМ, которая работала без участия человека.

Следует отметить, что мощным стимулом для появления и первых шагов вычислительной техники была Вторая мировая война, а также последовавшая за ней гонка вооружений, развитие ядерных и космических программ. Таким образом, возникали новые вычислительные задачи — для расчёта термоядерных моделей, различных задач баллистики, навигации, криптографии, радиолокации, а также логистики, телекоммуникаций и т. д. Эти задачи требовали новых методов решения. При этом ресурсы и бюджеты на создание новых решений для этих задач были фактически неограничены…

Поколения ЭВМ

Историю ЭВМ принято разбивать на так называемые поколения, которые сменялись по мере появления новых технологий производства ЭВМ.

К первому поколению относят ЭВМ, созданные во второй половине 40-х годов XX века. Эти ЭВМ основывались на *электромеханических реле*, *ламповых диодах и триодах*. Данные устройства использовались для создания логических элементов и схем управления первых компьютеров, позволяя выполнять вычислительные процессы. Примерами таких компьютеров являются ENIAC (США), EDVAC (США), Z4 (ФРГ, наследник Z3), МЭСМ (СССР), «Урал» (СССР), «Стрела» (СССР). Однако общепринятые принципы конструирования ЭВМ ещё не были выработаны. Некоторые из перечисленных ЭВМ использовали десятичную арифметику (например, ENIAC), некоторые — двоичную, но с необычным размером машинного слова (43 бита, «Стрела»), и т. д. Эти ЭВМ были весьма ненадёжными и часто не могли бесперебойно отработать в течение одних суток. Большие физические размеры, значительное количество деталей, высокое энергопотребление, сложность обслуживания (прежде всего, громоздкие и дорогостоящие системы охлаждения) — все это было причиной того, что ЭВМ в это время были доступны только крупным научно-исследовательским институтам, военным и государственным учреждениям. Но быстродействие ЭВМ измерялось тысячами операций в секунду, что для человека с арифмометром было уже недостижимо. Становилось всё более очевидным, что дальнейший прогресс науки и техники уже однозначно связывался с использованием и развитием ЭВМ.

Примером ЭВМ первого поколения является ENIAC (Electronic Numerical Integrator and Computer). Данная ЭВМ была фактически первой, которую можно было использовать для решения широкого спектра задач. Она была выпущена в 1945 году в США, имела массу около 30 тонн и потребляла около 200 кВт (современная городская квартира потребляет в среднем менее 1 кВт). ENIAC применялась для артиллерийских и аэродинамических расчётов, а позже была использована при расчётах в области ядерной физики, в частности, при разработке атомного оружия. Среди известных учёных той эпохи, оказавших влияние на развитие вычислительной техники,

следует назвать физика и математика Джона фон Неймана. Участвуя в расчётах в рамках Манхэттенского проекта и основываясь на выявленных недостатках ENIAC, он предложил ряд принципов для конструирования следующей, более совершенной ЭВМ под названием EDVAC. Позже совокупность этих принципов получила название архитектуры фон Неймана. Принципы архитектуры фон Неймана, пусть и в несколько изменённом виде, применяются при конструировании ЭВМ до сих пор.

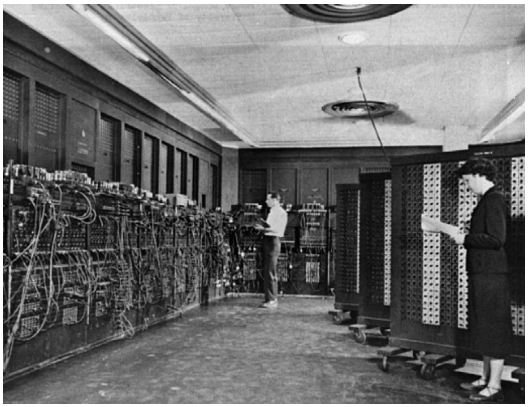


Рис. 1. ЭВМ первого поколения ENIAC

Ко второму поколению относят ЭВМ 1950–1960-х годов. В это время произошёл переход при построении электронных схем от электронных ламп и электромеханических реле к полупроводниковым элементам. Данный переход позволил значительно повысить вычислительную мощность ЭВМ, а также многократно уменьшить их размеры, снизить потребляемую электрическую мощность, а также цену и затраты на обслуживание. В итоге расширился спектр пользователей ЭВМ — их стали приобретать университеты и крупные промышленные предприятия (банки, заводы и т. д.), а также всё больше государственных учреждений. Среди известных ЭВМ второго поколения можно назвать IBM 7090, созданную в 1959 году и предназначенную для научных и инженерных расчётов. Следует также упомянуть DEC PDP-7, созданную в 1964 году; данная ЭВМ была универсальной и по тем временам недорогой. Наконец, следует назвать высокопроизводительные компьютеры CDC-6600 (1964 г.) и БЭСМ-6 (1968 г., СССР).

Рис. 2. ЭВМ второго поколения IBM 7090

К третьему поколению относятся ЭВМ 1960-х-70-х годов. В это время произошёл следующий эволюционный шаг — появились *интегральные* схемы, которые содержали до нескольких тысяч полупроводниковых элементов. Теперь все электронные схемы ЭВМ размещались на наборе интегральных схем. Таким образом принцип модульности сделал ещё один шаг вперёд, что положительно повлияло на технологии производства, позволив, в частности, реализовать взаимозаменяемость отдельных составляющих ЭВМ. В итоге удалось существенно увеличить производительность и надёжность ЭВМ, а также уменьшить размеры и затраты на их эксплуатацию. Примерами ЭВМ этого поколения являются IBM System/360 (IBM S/360), компьютеры семейства ЕС ЭВМ в СССР.

Рис. 3. ЭВМ третьего поколения IBM S/360

К четвёртому поколению относятся ЭВМ, выпускаемые с середины 1970-х годов вплоть до настоящего времени. Главным нововведением в этих ЭВМ стало наличие *микропроцессора* (далее — просто процессора), который:

- является центральным блоком ЭВМ и осуществляет выполнение программ;
- реализован в виде одной компактной интегральной схемы.

Размер этой интегральной схемы составил всего несколько сантиметров, что заметно контрастировало с ЭВМ третьего поколения, процессоры которых размещались на нескольких печатных платах. Цена ЭВМ значительно снизилась и достигла уровня цены автомобиля, а позже — обычной бытовой техники. В результате ЭВМ стали доступны среднему и малому бизнесу, а в 1980-х годах появились и получили широкое распространение персональные компьютеры.

Первыми персональными компьютерами стали ЭВМ компаний Apple и IBM (последние получили название IBM PC). В настоящее время персональные компьютеры производятся многими компаниями — Dell EMC, Lenovo, Huawei и др.

Рис. 4. ЭВМ четвёртого поколения IBM PC (слева) и Apple III (справа)

Закон Мура

В 1965 году инженер Гордон Мур (позднее он основал компанию Intel) выявил и сформулировал эмпирическую закономерность, позже названную законом Мура.

> ***Закон Мура*** *утверждает, что количество полупроводниковых элементов в интегральных схемах каждые 24 месяца удваивается, то есть растёт экспоненциально.*

Это означает, что примерно так же растет и производительность интегральных схем, а значит, и производительность компьютеров в целом.

> ***Следствием закона Мура*** *является утверждение о том, что производительность компьютеров каждые 24 месяца удваивается, то есть растёт экспоненциально.*

С момента своего появления и до настоящего времени закон Мура исправно выполняется. Однако в 1960–1970-х годах и в настоящее время он реализуется разными путями. К настоящему времени физические законы уже не позволяют наращивать производительность электронных схем за счет более «плотного» размещения транзисторов на одной кремниевой пластине. Так происходит потому, что размеры транзисторов становятся сопоставимы с размерами отдельных атомов, а скорость их взаимодействия приближается к скорости света. Делать транзисторы меньше атомов, а скорость их взаимодействия — быстрее скорости света не представляется возможным. Значительно увеличивать размер кремниевой пластины также не получается. Но такое увеличение и не является выходом, поскольку важно требование сохранения небольших размеров, а также дальнейшей миниатюризации — как самих компьютеров, так и вычислительных устройств, встраиваемых в различные механические приборы.

Поэтому в настоящее время конструкторы новых процессоров идут по пути распараллеливания. В частности, создаются многоядерные процессоры: на одной кремниевой пластине размещается до нескольких десятков микропроцессоров (ядер), работающих параллельно, что значительно ускоряет время работы итогового процессора. С другой стороны, такой подход усложняет задачу и конструкторам процессоров, и программистам — организация параллельных вычислений требует существенных интеллектуальных затрат. Однако на данный момент этот подход является основным способом продолжать наращивать вычислительную мощность ЭВМ.

Обзор отечественных ЭВМ

Говоря об отечественных ЭВМ первого поколения, укажем прежде всего на МЭСМ (Малая Электронная Счетная Машина), которая была создана в 1951 году С. А. Лебедевым и была первой ЭВМ общего назначения в СССР и в континентальной Европе. Практически одновременно с МЭСМ была создана ЭВМ М-1, которая обладала схожими с МЭСМ характеристиками, но в ней наряду с лампами широко применялись полупроводниковые диоды. Позже была создана БЭСМ-1 (1955 г.), которая поддерживала арифметику с плавающей запятой, а также БЭСМ-2 (1958 г.), в которой широко применялись полупроводниковые диоды.

ЭВМ второго поколения БЭСМ-6 (1966–1983 гг.) поддерживала локальный параллелизм на базе конвейера, виртуальную память, многозадачность

и защиту программ. Ламповые ЭВМ оборонного назначения М-40 и М-50 (конец 1950-х гг.) были сконструированы для управления комплекса противоракетной обороны (ПРО) СССР. Позже для использования в системах ПРО были сконструированы полностью полупроводниковые ЭВМ Э92б (1961 г.) и 5Э51 (1965 г.). ЭВМ «Сетунь» (1959 г.) и «Сетунь-70» (1970 г.) были основаны на троичной логике. «Сетунь» является единственной в мире серийной троичной ЭВМ. Минск-222 (1966 г.) была первой в мире распределённой вычислительной системой, она была создана на базе ЭВМ второго поколения «Минск-2»/«Минск-22» и обладала высокой гибкостью и настраиваемостью.

Рис. 5. Советские ЭВМ: МЭСМ (первое поколение, слева)
БЭСМ-6 (второе поколение, справа)

Большинство советских ЭВМ третьего и четвёртого поколений (1970–1980-е годы) копировали иностранные, что позволило снизить расходы на их создание и повысить компьютеризацию промышленности в СССР. Однако это негативно сказалось на развитии отечественной вычислительной техники. Поэтому оригинальные отечественные разработки этой эпохи заслуживают особого внимания.

В академии наук СССР в 1970–1990-х годах разрабатывалось семейство высокопроизводительных компьютеров «Эльбрус» (руководители проекта — В. С. Бурцев и Б. А. Бабаян). ЭВМ семейства «Эльбрус» предназначались для обработки больших объемов данных. Эти ЭВМ, конечно же, не были предназначены для небольших организаций или частных лиц: техника такого класса была доступна лишь крупным научным и оборонным центрам. Обладая высокой производительностью, они использовались в ядерных исследовательских центрах, а также, с конца 1980-х годов использовались, в системе противоракетной обороны города Москвы.

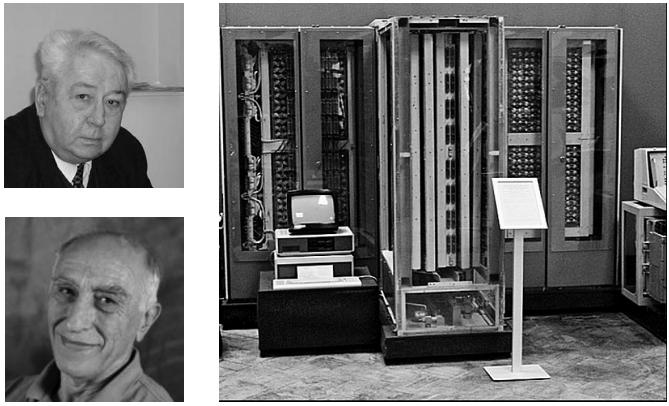


Рис. 6. В. С. Бурцев, Б. А. Бабаян. Советский суперкомпьютер «Эльбрус»

В 80-х годах XX века в Ленинградском государственном университете (ныне — СПбГУ) под руководством А. Н. Терехова была создана ЭВМ четвертого поколения «Самсон». Оригинальная стековая система команд «Самсона» была оптимизирована для эффективного исполнения программ, написанных на языках высокого уровня. По производительности и габаритным размерам «Самсон» был сравним со своими современниками — ранними персональными компьютерами семейства Intel x86. В 1990-е годы «Самсон» был базовой ЭВМ ракетных войск стратегического назначения России.

Рис. 7. А. Н. Терехов. ЭВМ «Самсон»

Примерно в те же годы в вычислительном центре сибирского отделения российской академии наук под руководством В. Е. Котова была создана ЭВМ «Кронос», также имеющая стековую систему команд и ориентированная на исполнение высокоуровневых программ. «Кронос» позиционировалась как универсальная персональная ЭВМ для офисной, инженерной и научной работы и даже предусматривала использование нового в то время графического интерфейса пользователя.

Рис. 8. В. Е. Котов. ЭВМ «Кронос»

Вопросы

1. Приведите примеры докомпьютерных вычислительных устройств.
2. Назовите предпосылки появления первых ЭВМ.
3. Приведите примеры первых специализированных компьютеров.
4. Расскажите о ЭВМ, ABC отмечая то новое, что привнесла эта ЭВМ.
5. Расскажите о ЭВМ Z3, отмечая то новое, что привнесла эта ЭВМ.
6. Расскажите о ЭВМ Colossus, отмечая то новое, что привнесла эта ЭВМ.
7. Расскажите о ЭВМ Mark I, отмечая то новое, что привнесла эта ЭВМ.
8. Что было одной из основных движущих сил развития первых ЭВМ?
9. А что, на ваш взгляд, является движущими силами в настоящее время, и имеется ли разница?
10. Опишите назначение и спектр задач ЭВМ первого поколения.
11. Приведите пример ЭВМ первого поколения.
12. Появление каких технологий привело к созданию ЭВМ второго поколения?
13. Приведите примеры ЭВМ второго поколения.
14. Какие технологии легли в основу ЭВМ третьего поколения?
15. Приведите примеры ЭВМ третьего поколения.
16. Назовите ключевую особенность ЭВМ четвертого поколения.
17. Расскажите о первых компьютерах четвёртого поколения.
18. Назовите компании, выпустившие первые персональные компьютеры.
19. Какие компании выпускают персональные компьютеры сегодня?
20. Сформулируйте закон Мура.
21. За счет чего реализуется закон Мура в настоящее время?
22. Сделайте обзор истории отечественных ЭВМ.
23. Назовите первую советскую ЭВМ общего назначения.
24. Какие вы можете назвать советские ЭВМ первого, второго и третьего поколений?
25. Перечислите отечественные ЭВМ четвёртого поколения.

26. Расскажите про советский суперкомпьютер «Эльбрус».
27. Расскажите про отечественные персональные компьютеры «Самсон» и «Кронос».

Литература

1. *Хорошевский В. Г.* Архитектура вычислительных систем: Учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.
2. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
3. *Терехов А. Н.* УВК «Самсон» — базовая ЭВМ РВСН // Труды SORUCOM-2011. 2011. С. 282–286.

Лекция 2. Архитектура фон Неймана

Принципы архитектуры фон Неймана; системная шина и общая схема типовой ЭВМ, основанной на архитектуре фон Неймана; гарвардская архитектура; комбинирование фон Неймановской и гарвардской архитектур в современных вычислительных системах.

Принципы архитектуры фон Неймана

Традиционно, под архитектурой ЭВМ понимают полную и детальную спецификацию интерфейса пользователя ЭВМ. При этом пользователями считались как люди — в основном программисты, работающие с ЭВМ, — так и технические средства, взаимодействующие с ЭВМ. Это определение было дано Фредериком Брукcом в 1975 году в его знаменитой книге «Мифический человеко-месяц» и стало общепринятым в индустрии. Однако оно, на наш взгляд, выглядит несколько архаично и не вполне вписывается в контекст нашего курса.

> ***Архитектурой ЭВМ*** *будем называть основные принципы организации вычислительных устройств, их основные узлы и интерфейсы, а также технологии и методы, с помощью которых они реализованы*[1].

Большинство современных ЭВМ созданы на основе *архитектуры фон Неймана*, предложенной Джоном фон Нейманом и его коллегами из Университета Принстона. Эта архитектура была создана для ЭВМ под названием EDVAC, разрабатываемого в рамках Манхэттенского проекта (проект по созданию атомной бомбы в США, осуществлялся в конце Второй мировой войны). Отчет, описывающий основные принципы организации этого компьютера, был написан фон Нейманом в 1945 году и, с разрешения американских военных, разослан в основные ведущие мировые университеты. Этот отчёт оказал решающее влияние на развитие индустрии и исследований в компьютерной сфере. Ниже представлены основные принципы архитектуры фон Неймана.

1. Программное управление работой ЭВМ.
2. Принцип хранимой программы.
3. Принцип адресуемости памяти.

[1] Дальнейшее обсуждение термина «Архитектура ЭВМ» можно найти в учебнике В. Г. Хорошевского «Архитектура вычислительных систем».

4. Использование двоичной системы исчисления.
5. Иерархичность запоминающих устройств.
6. Использование условных переходов.

Детально рассмотрим эти принципы.

Программное управление работой ЭВМ. Любое сложное оборудование имеет интерфейс для взаимодействия с оператором, и чем сложнее задачи, для решения которых оборудование предназначено, тем более сложный интерфейс взаимодействия с оператором оно имеет. Посредством этого интерфейса человек управляет оборудованием — принимает решения, которые последнее принять не в состоянии, проводит диагностику, перезапуск и т. д. Например, многие современные самолёты обладают интеллектуальными системами автопилотирования, но, даже умея самостоятельно решать сложные навигационные задачи, эти системы требуют от экипажа задания начальных условий маршрута, отдельных параметров режима полёта и т. д. ЭВМ также является сложным оборудованием, которое помогает конечному пользователю решать его задачи, как правило, уже не лежащие в области программирования и вычислительной техники, как, например, рассматриваемая выше задача управления самолетом. Другим примером являются многочисленные задачи управления бизнес-организацией — учёт различных ресурсов, мониторинг индустриальных процессов и пр. Можно также вспомнить про мобильные устройства, которые решают задачу общения людей, помогают ориентироваться на местности, удобно производить покупки и выполнять платежи и т. д.

Принцип программного управления ЭВМ архитектуры фон Неймана утверждает, что управление вычислительной техникой — мобильными устройствами, ноутбуками, рабочими станциями и серверами и пр. — может осуществляться оператором не напрямую, а с помощью *программ*. В идеале программы вообще освобождают оператора (конечного пользователя) от участия в управлении работой ЭВМ, требуя лишь задания начальных данных. Но даже последние часто считываются программой автоматически, являясь, так сказать, параметрами контекста.

Программа, исполняемая на ЭВМ, состоит из отдельных команд (действий), и каждая команда позволяет осуществить единичный акт преобразования информации над некоторыми данными. Команды, которые могут исполняться конкретной ЭВМ, составляют её *машинный язык*.

Машинный язык ЭВМ должен был обладать выразительной силой: требуется, чтобы с его помощью было возможно и удобно задавать соответствующие алгоритмы. С другой стороны, он должен позволять эффективное выполнение на соответствующей ЭВМ. Компромисс между наглядностью и выразительной силой, с одной стороны, и вычислительной эффективностью, с другой стороны, является одной из главных проблем современных языков программирования (будем считать, что читателю очевидно, что современные языки программирования являются надстройкой над машинными языками ЭВМ, а программы, созданные с их помощью, относительно несложно превращаются в команды ЭВМ).

Один из пионеров вычислительной техники Чарльз Бэббидж пытался реализовать данный принцип ещё в XIX веке, но потерпел неудачу. Технологии XX века позволили успешно создать сложные программируемые устройства. Соответственно, для них стали разрабатываться специальные «понятные» им языки (машинные языки), с помощью которых стало возможно выразить вычислительные алгоритмы.

Принцип хранимой программы. Этот принцип утверждает, что в процессе исполнения программа вместе со своими данными хранится в ЭВМ, а не передается через операторский интерфейс по одной команде. При этом важным является то, что и код, и данные программы находятся в одном и том же месте — в оперативной памяти. В процессе исполнения программы процессор выбирает из оперативной памяти команды и их операнды (то есть данные). При этом имеется возможность сделать команду операндом, и тогда над ней можно выполнять различные операции — таким образом, появляется возможность преобразовывать программы при их выполнении. Также следует отметить, что принцип хранимой программы обеспечивает одинаковое время выборки команд и операндов из памяти, позволяет использовать косвенные системы адресации для работы с памятью произвольных размеров и многое другое.

Наличие оперативной памяти, в которой хранится как программа, как и её данные, была новаторской идеей. Например, компьютер Mark I представлял собой фактически мощный электромеханический калькулятор, снабжённый управляющим устройством. Исполняемая программа непосредственно считывалась с перфоленты, а не загружалась в оперативную память целиком перед запуском, то есть принцип хранимой программы архитектуры фон Неймана нарушался. Затраты на покомандное считывание сильно влияли на скорость исполнения программы. Однако на тот момент (40-х годы XX века) это всех устраивало: тот факт, что вычислительное устройство можно было программировать, а не подавать ему на вход каждую команду вручную, уже был значительным шагом вперёд. Менее знаменитый современник компьютера Mark I, немецкий компьютер Z3, созданный и утерянный во время Второй мировой войны, был устроен аналогичным образом.

Принцип хранимой программы впоследствии позволил создать компиляторы, операционные системы, различные системные программы, которые манипулируют машинным кодом программ как данными, позволяя достичь значительной автоматизации в программном управлении ЭВМ.

Принцип адресуемости памяти. Данный принцип постулирует, что память ЭВМ (в частности, оперативная память) устроена следующим образом. Она представляет собой набор пронумерованных ячеек, каждая из которых имеет, таким образом, адрес доступа — номер. Эти ячейки называются *словами*, у разных ЭВМ размер слова бывает разным (подробно про это будет рассказано в следующих лекциях). Команды программы, а также её данные располагаются в этих словах последовательно. Процессору в любой момент времени доступно любое слово в памяти по номеру (адресу). Таким образом, память является массивом слов.

Уже позднее данный принцип позволил организовывать различные схемы адресации, используя адресные сегменты, косвенную адресацию и т. д. (все это будет подробно рассмотрено в следующих лекциях). Такие схемы, в свою очередь, позволили реализовать гибкие стратегии размещения программ и данных в памяти, а также осуществлять эффективный доступ к ним во время работы процессора.

Использование двоичной системы исчисления. Этот принцип означает, что для представления команд программ и их данных используется двоичная система счисления. Первые компьютеры использовали десятичную систему исчисления, были также варианты использования троичной системы исчисления.

Именно использование двоичной системы позволило наладить индустриальное производство ЭВМ. Дело в том, что, поскольку в двоичной системе имеются только две цифры (0 и 1), для их представления может быть использована любая физическая система с двумя стабильными состояниями. Например, триод и транзистор (открытое или закрытое состояние), триггер с двумя устойчивыми состояниями, импульсная схема (наличие или отсутствие электрического импульса) и т. п. Предпринимались попытки конструирования ЭВМ на основе других систем исчисления (например, троичной и десятичной), и были достигнуты интересные результаты. Но лишь для двоичной системы удалось создать электронные схемы, надёжные и недорогие при массовом производстве.

Этот принцип также не был очевидным. Альтернативой двоичной системе исчисления была в первую очередь традиционная десятичная система. Для неё существовало много средств, например, алгоритмы для арифмометров, широко использовавшихся в 40–50 годы XX века. Арифмометры — это небольшие механические вычислители, позволявшие выполнять 4 главных арифметических действия и помещавшиеся на письменном столе. Были также созданы, опубликованы и широко использовались таблицы значений функций и физических величин с достаточно высокой точностью в форме десятичных дробей и т. д.[2] Таким образом, при переходе на двоичную систему требовалось преобразовать налаженные десятилетиями средства и процедуры расчётов, применяемые во многих областях промышленности и науки.

Иерархичность запоминающих устройств. Этот принцип утверждает, что ЭВМ должна иметь различные виды памяти, обладающие различным объёмом и быстродействием. Кратко рассмотрим разные виды памяти современных ЭВМ.

Сначала сфокусируемся на долгосрочной и оперативной памяти.

Долгосрочная память предназначается для длительного хранения программ и данных компьютера. Фотографии, фильмы, документы и пр. хранятся именно в долгосрочной памяти. Там же хранятся все программы, которые установлены и могут исполняться на вашем компьютере. Таким образом,

[2] Люди старшего поколения помнят таблицы Владимира Брадиса, в которых были собраны значения тригонометрических функций, логарифмов и прочие величины для использования в математических расчётах с точностью четыре знака после запятой.

содержание долгосрочной памяти остается неизменным после выключения и включения компьютера, чего нельзя сказать о других видах памяти. Эта память имеет большие объемы и относительно низкую скорость доступа. Существуют следующие варианты реализации долгосрочной памяти:

- механические магнитные диски (жесткие диски, которые часто называют «винчестерами»);
- твердотельные накопители (SSD);
- внешние запоминающие устройства — гибкие и оптические диски, твердотельная флэш-память, магнитные ленты.

Оперативная память предназначена для хранения программы и её данных в процессе исполнения. То есть программа загружается из долгосрочной памяти в оперативную, давая возможность процессору выполнить её. Таким образом, процессор не работает с долгосрочной памятью, что сильно уменьшило бы скорость исполнения программы. Соответственно, оперативная память имеет существенно меньший объем, чем долгосрочная, но скорость доступа к ней значительно выше — ведь требуется, чтобы процессор мог получать следующие команды и значения соответствующих переменных очень быстро. В связи с этим для изготовления оперативной памяти используются иные технологии, нежели для изготовления долгосрочной памяти.

Взаимодействие процессора и оперативной памяти осуществляется через *системную шину* — быстрый канал передачи информации. Несмотря на эффективную реализацию, системная шина является принципиальным узким местом архитектуры фон Неймана, замедляющим быстродействие компьютера. На решение этой проблемы нацелено большое количество различных решений на самых разных уровнях организации процесса выполнения программы.

Разделение памяти компьютера на оперативную и долгосрочную было одной из главных инноваций архитектуры фон Неймана. Однако в современных компьютерах иерархия памяти значительно сложнее. Кратко рассмотрим следующие виды памяти:

- регистры процессора;
- кэш-память процессора;
- внешние устройства памяти (дискеты, лазерные диски, флэш-карты);
- память для хранения настроек компьютера.

Процессор имеет набор *регистров*, каждый из которых является небольшой ячейкой памяти (8, 16, 32 или 64 бита). Эти ячейки используются процессором для арифметических операций, а также для выполнения других команд. Команды, оперируя регистрами, выполняются существенно быстрее, нежели в ситуации, когда данные берутся из долгосрочной памяти, поскольку регистры находятся непосредственно внутри процессора. Например, у процессора Intel 8080 было 7 регистров, а у Intel Itanium — 256 регистров. Одним

из факторов, который влияет на количество регистров процессора, является его цена и массовость производства.

Поскольку объем памяти, который содержится в регистрах, крайне незначителен, а проблема оптимизации сообщения процессора и оперативной памяти в процессе исполнения программ чрезвычайно актуальна, используется ещё один вид памяти — *кэш-память*. Это особый вид быстрой памяти, которая находится в процессоре и предназначается для хранения данных, активно используемых процессором при выполнении одной или нескольких программ, позволяя существенно ускорить их выполнение. В многоядерных процессорах эта память устроена иерархично — каждое ядро имеет свою кэш-память, и весь процессор также имеет свою собственную кэш-память. Подробнее об этом будет рассказано в следующих лекциях.

> ***Быстродействие памяти*** *— это скорость выполнения операции чтения данных и записи данных, которую обеспечивают микросхемы, реализующие данную память.*

Высокое быстродействие памяти (регистры, кэш-память, оперативная память) требует принципиально иных технологий аппаратной реализации, чем низкое быстродействие (жесткий диск). При этом быстрая память оказывается дороже (причём в разных смыслах), чем медленная, поэтому она имеет существенно меньший объем. Объём памяти типичного офисного компьютера, который мы рассмотрим в следующих лекциях, составляет от 8 до 16 Gb оперативной памяти и от 500 до 2000 Gb — жесткий диск.

Наконец, скажем несколько слов о *внешних устройствах памяти*, которые широко использовались и используются для различных компьютеров. Речь идёт о дискетах, лазерных дисках и флэш-картах. Они являются внешними, переносными — их легко можно вставить в компьютер, что каждый из нас делал много раз. Наряду с Интернетом они используются для передачи данных между разными компьютерами. Существует несколько поколений таких устройств. Пожалуй, первыми были гибкие диски, или дискеты, используемые для персональных компьютеров IBM PC. Позже появились лазерные диски, правда, они не предназначались для активной передачи данных — записать что-либо на них можно было, в зависимости от их разновидности, однократно или ограниченное количество раз, они использовались главным образом для дистрибутивов программ, а также для фильмов и аудиозаписей. Наконец, флэш-память (так называемые флэшки) широко используются в настоящее время и позволяет переносить как фильмы, фотографии и аудиозаписи, так и программы, а также различные документы и другие виды данных. Флэш-память удобна, поскольку не везде имеется Интернет, а иногда бывает, что его скорость недостаточна. Также не все пользователи используют внешние хранилища наподобие Яндекс-диска или Google-Drive — например, по соображениям безопасности. Флэш-память изготовляется на основе специальных транзисторов, которые запоминают своё состояние (открытое или закрытое) при выключении питания.

Память для настроек компьютера используется для хранения основных параметров, используемых при его запуске. В персональных компьютерах такая память имеет объём в пределах нескольких килобайт и хранит настройки жёстких дисков, состояние адаптера Wi-Fi, настройки загрузки операционной системы с жёсткого диска или по локальной сети и т. д. Отдельный вид памяти для настроек нужен, поскольку столь базовые настройки не могут быть прочитаны с жёсткого диска: ведь часть из них используется при его инициализации. А в некоторых комплектациях компьютеров, загружающихся по локальной сети, жёсткого диска может и вовсе не быть.

Подводя итог, заметим, что наличие разных видов памяти в ЭВМ позволяет находить гибкий компромисс между размерами (ёмкостью), быстродействием, ценой и надежностью запоминающих устройств, поскольку для разных задач требуются различные значения этих характеристик!

Также следует отметить, что технологии производства разных видов памяти существенно различаются.

- Оперативная память на самом нижнем уровне хранит биты данных в виде электрических зарядов конденсаторов.
- Твердотельные диски реализуются на основе флэш-памяти, состоящей из специального вида транзисторов, запоминающих своё состояние.
- Кэш-память и регистры хранят данные с помощью специальных электронных схем — триггеров.
- Механические жёсткие диски используют магнитную память.
- Память для настроек может изготавливаться с применением технологии CMOS (Complementary Metal-Oxide-Semiconductor), которая основана на триггерах и требует постоянного электропитания; она может работать несколько лет от часовой батарейки. *Триггер* является простейшей электронной схемой, которая имеет два устойчивых состояния: высокий или низкий электрический потенциал.

О разных видах памяти мы более подробно расскажем в следующих лекциях.

Наконец, отметим, что рассматриваемый в данном разделе принцип иерархичности запоминающих устройств, как и предыдущие принципы архитектуры фон Неймана, на заре компьютерной эпохи был совершенно неочевидным.

Использование условных переходов

> ***Условный переход*** *— это машинная команда или конструкция языка программирования, которая позволяет направить выполнение программы по разным веткам в зависимости от значения некоторой переменной или результата выполнения некоторого условия.*

В современных языках программирования, а также в машинных языках, существует большое количество различных условных переходов. Например, в языке C имеется конструкция `if/then/else`, которая позволяет программе «ветвиться» по значению некоторого логического условия. Имеется также команда `switch/case`, которая позволяет «ветвить» поток выполнения программы по условиям, имеющим не два значения — истину или ложь как логические условия, — а некоторый спектр значений, тем самым реализуя более чем два ветвления.

Кроме непосредственного использования условные переходы очень важны в циклах. Ведь в циклах постоянно проверяется выполнение некоторого условия и, и в зависимости от его истинности, или запускается следующая итерация тела цикла, или происходит выход из цикла на следующую после цикла инструкцию. Также условные переходы используются неявно в ряде других конструкций языков программирования и машинных языках.

Несмотря на то, что сегодня условные переходы кажутся чем-то само собой разумеющимся, они появились далеко не сразу. Например, уже упомянутые компьютеры Mark I и Z3 их не поддерживали. В том месте программы, где требовался условный переход, компьютер должен был остановиться и передать управление оператору.

Системная шина

На рис. 9 представлена схема типовой ЭВМ, созданной на основе архитектуры фон Неймана. Все узлы ЭВМ разделены на три группы — процессор, оперативная память, периферийные устройства (жесткий диск, графическая карта компьютера, сетевая карта, различные порты — USB, HDMI и т. д.)[3]. Связь процессора с остальными узлами ЭВМ осуществляется с помощью унифицированного канала доступа — системной шины.

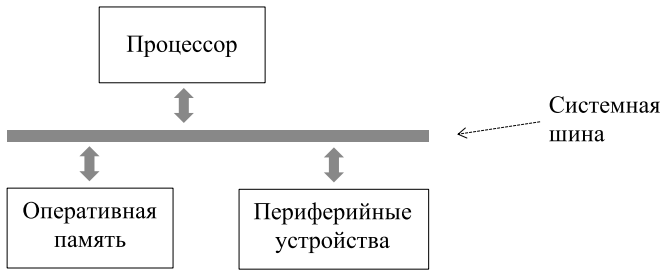


Рис. 9. Общая схема типовой ЭВМ, основанной на архитектуре фон Неймана

[3] Следует отметить, что рис. 9 не вполне точно следует оригинальной архитектуре фон Неймана — там не было единого процессора, а вместо него имелись управляющее и арифметически-логическое устройства, а также в качестве периферийных устройств использовались лишь устройства ввода-вывода. Процессор как единое устройство, которое интегрирует разные блоки, участвующие в исполнении программ, появился лишь в ЭВМ четвёртого поколения.

Дадим определение системной шине.

> ***Системная шина*** *является быстрым каналом передачи данных и предназначается для обеспечения эффективной коммуникации процессора с оперативной памятью и периферийными устройствами ЭВМ.*

У первых компьютеров системная шина состояла из трёх шин специального назначения: шины данных, адресной шины, управляющей шины.

- *Шина данных* предназначена для передачи данных между процессором, памятью и внешними устройствами.
- *Адресная шина* предназначена для передачи адресов ячеек памяти и идентификаторов внешних устройств.
- *Управляющая шина* предназначена для передачи управляющих сигналов между процессором и оперативной памятью. Подобная координация необходима, поскольку выполнение тех или иных действий требует времени и устройства должны «знать», когда им необходимо начать то или иное действие. В качестве примера можно привести координацию процессором считывания данных оперативной памятью с шины данных. Получив по управляющей шине соответствующий сигнал, оперативная «решает», что данные на шине данных готовы к считыванию, и соответственно начинает их чтение.

Отметим, что в современных ЭВМ системная шина организована иначе, но приведённое выше разделение каналов передачи информации активно применяется на разных уровнях организации системной шины.

Гарвардская архитектура

Узким местом архитектуры фон Неймана является системная шина: она интенсивно используется во время исполнения программы, поскольку и код программы, и её данные находятся в одном месте (оперативная память), а исполняются в другом (процессор). Поэтому её пропускная способность во многом определяет скорость исполнения программы. Существуют различные способы решения этой проблемы, например, кэш-память, который будет обсуждаться далее, однако она является принципиальной и полностью её решить не удаётся.

Совместное хранение программы и ее данных в оперативной памяти ЭВМ позволяет существенно удешевить производство ЭВМ и упростить её архитектуру, но не является эффективным по быстродействию. Если реализовать раздельное хранение программы и ее данных, то можно пересылать команды программы и соответствующие данные в процессор параллельно, что значительно ускоряет работу ЭВМ. Такой подход получил название *гарвардской архитектуры*.

> ***Гарвардская архитектура*** *предусматривает раздельное хранение программы и данных в оперативной памяти разного вида, а также возможность параллельного считывания того и другого процессором при помощи отдельных шин.*

Гарвардская архитектура была предложена Говардом Эйкеном в конце 1930-х годов при работе над компьютером Mark I. Кроме того, упоминаемый нами компьютер Z3 также имел гарвардскую архитектуру.

Комбинирование фон Неймановской и гарвардской архитектур

Существенным недостатком гарвардской архитектуры является то, что требуется реализовать два устройства оперативной памяти и две независимых шины: при массовом производстве это значительно повышает стоимость компьютера. Однако в ряде случаев это оказывается целесообразным и идеи гарвардской архитектуры используются на практике. Перечислим несколько типичных случаев, в которых применяется гарвардская архитектура.

- Быстрая кэш-память многих современных процессоров разделяется на память для данных и память для машинного кода. При этом загруженные данные невозможно исполнять (то есть рассматривать их как код), а загруженный код — менять (то есть рассматривать его как данные). Доступ к разным видам кэш-памяти осуществляется через разные внутренние шины процессора, что позволяет повысить его производительность, правда, ценой усложнения и удорожания.
- В простых специализированных вычислительных устройствах для управления лифтами, узлами современных автомобилей, кондиционерами и т. д. машинный код записывается непосредственно при изготовлении, и таким образом устройство может только считывать его при исполнении, но не имеет возможности изменять. Оперативная память этих устройств предназначена исключительно для данных и не используется для хранения машинного кода. Для хранения машинного кода в таких устройствах обычно используется специальная память объёмом до нескольких сотен килобайт, и её интегрируют в блок процессора, предназначенный для чтения машинного кода. Таким образом, в данном случае гарвардская архитектура не усложняет вычислительное устройство, а, напротив, упрощает и удешевляет его.
- Для реализации вспомогательных узлов ЭВМ — контроллеров жёстких дисков, графических адаптеров и пр. — также используются идеи гарвардской архитектуры. Например, контроллер жёсткого диска попадает в предыдущий пункт, поскольку выполняет фиксированный набор несложных функций. Графический адаптер оказывается более сложным устройством, которое может выполнять

различные операции, но при этом исполняемый им машинный код размещается в отдельной памяти, запись в которую осуществляет центральный процессор ЭВМ, а сам графический адаптер лишь считывает этот код для исполнения.

Вопросы

1. Дайте определение архитектуры ЭВМ.
2. Перечислите основные принципы архитектуры фон Неймана.
3. Изложите принцип хранимой программы.
4. Объясните, что означает программное управление работой ЭВМ.
5. Приведите пример оборудования, которое не управляется программно.
6. Расскажите про принцип хранимой программы.
7. Что означает принцип адресуемости памяти?
8. Что такое машинное слово?
9. Расскажите о преимуществах использования двоичной системы исчисления.
10. Что означает принцип иерархичности запоминающих устройств?
11. Что такое быстродействие памяти?
12. В чём отличие долгосрочной и оперативной памяти?
13. Какие виды памяти в современных ЭВМ вы знаете?
14. Расскажите о технологиях реализации памяти различного вида.
15. Дайте определение триггера.
16. Почему настройки компьютера нельзя хранить на винчестере, а требуется специальный вид долгосрочной памяти?
17. Что такое условный переход?
18. Расскажите, как первые компьютеры обходились без условного перехода.
19. Что такое системная шина?
20. Расскажите о структуре системной шины.
21. Расскажите о схеме типовой ЭВМ, созданной на основе архитектуры фон Неймана.
22. Объясните, что является узким местом архитектуры фон Неймана.
23. Назовите преимущества и недостатки гарвардской архитектуры.
24. Назовите примеры первых фон Неймановских и гарвардских компьютеров.
25. Приведите случаи использования гарвардской архитектуры на практике.

Литература

1. *Хорошевский В. Г.* Архитектура вычислительных систем: учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.

2. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
3. *Брукс Ф., Чапель Х.* Мифический человеко-месяц, или Как создаются программные системы. Пер. с англ. СПб.: Символ-Плюс, 2010. 304 с.

Лекция 3. Двоичная система исчисления и машинная арифметика

Двоичная и другие системы исчисления; машинное слово и машинная арифметика; единицы измерения цифровых данных: килобайты, мегабайты, гигабайты, терабайты.

Двоичная и другие системы исчисления

Одним из положений архитектуры фон Неймана является использование двоичной системы исчисления. То есть числа и вся информация в ЭВМ представляются с помощью нулей и единиц. Все операции над данными, в частности арифметические — сложение, вычитание, умножение, деление и пр., — также выполняются в двоичном исчислении. Решающим фактором при выборе двоичной системы оказалось то, что электронная промышленность освоила эффективное массовое производство двоичных цифровых схем.

Двоичное представление числа представляет собой последовательность, состоящую из единиц и нулей, то есть двоичных цифр.

> *Каждая двоичная цифра хранится в одном **бите**. Таким образом, бит является простейшей информационной единицей в современных ЭВМ.*

Рассмотрим, как представляются числа в различных системах исчисления. Сначала разложим число 1929 по десятичному основанию:

$$1929_{10} = 1 * 10^3 + 9 * 10^2 + 2 * 10^1 + 9 * 10^0$$

Теперь разложим это число по двоичному основанию:

$$1929_{10} = 1 * 2^{10} + 1 * 2^9 + 1 * 2^8 + 1 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 11110001001_2$$

В ряде случаев используется шестнадцатеричное представление чисел, то есть разложение по основанию 16. Шестнадцатеричная система исчисления оказывается востребованной, например, при визуализации двоичных данных (в частности, бинарного кода программ) для восприятия человеком.

Хотя, конечно, код программы — не самое понятное, что можно себе вообразить, его шестнадцатеричное представление более читабельно, чем двоичное. В шестнадцатеричной системе исчисления используются десятичные цифры от 0 до 9 и несколько букв для обозначения чисел в диапазоне от 10 до 15: A, B, C, D, E, F. Так, число 1929 в шестнадцатеричном представлении выглядит так:

$$1929_{10} = 7 * 16^2 + 8 * 16^1 + 9 * 16^0 = 789_{16}$$

В общем случае представление натурального числа D в k-ичной системе исчисления выглядит так:

$$D_{10} = \sum_{i=0}^{p-1} k^i * d_i = d_0, \dots, d_{p-1_{\mathrm{k}}},$$

где p — это количество разрядов (цифр) числа, а $d_0, \dots, d_{p-1}$ — его k-ичные цифры, $\forall i\ d_i \in \{0, \dots, k-1\}$. Такое число изображается с помощью десятичных цифр, а также, возможно, с применением дополнительных цифр, если основание системы исчисления больше, чем 10.

Таким образом, в шестнадцатеричной системе исчисления мы имеем 16 цифр, в десятичной — 10, а в двоичной — 2. С помощью этих цифр в каждой системе исчисления составляются числа.

Нетрудно составить алгоритм, который раскладывает произвольное натуральное число в k-ичном исчислении, а также доказать единственность такого представления.

Машинное слово

Двоичное представление числа 1929 состоит из 11 цифр, следовательно, для его хранения требуется 11 бит. Можно сказать, что бит является атомарной ячейкой в памяти ЭВМ. Но адресовать каждый бит нецелесообразно: ЭВМ работает со строками, числами различного вида, а также массивами, меняя сразу по многу бит. Поэтому целесообразно адресовать не биты, а группы битов. В современных компьютерах атомарным множеством бит является байт — набор из 8 бит. Многие команды процессора умеют работать с байтами. Соответственно, данные выравниваются до границы байта. Так, число 1929 в двоичном виде занимает 2 байта и выглядит так: 0000011110001001. Пять лишних бит заполняется нулями.

В современных ЭВМ единицей адресации является не байты, поскольку последние позволяют оперировать слишком маленькими числами, а машинные слова.

> ***Машинное слово*** *— это атомарное количество информации, с которым может оперировать данная ЭВМ. Каждая ЭВМ имеет фиксированный размер машинного слова, например, в современных Intel-архитектурах размер машинного слова составляет 32 или 64 бита, то есть четыре или восемь байт соответственно.*

Размер машинного слова задаёт следующие важные характеристики ЭВМ:

- количество бит, которые процессор может обработать за один такт, что в числе прочих причин также определяет и размер регистров процессора;
- количество бит в шине данных, а следовательно, и
- количество бит, которое процессор может за одну операцию прочитать из оперативной памяти;
- максимальный объём оперативной памяти, которая может быть неадресуема непосредственно процессором.

У первого процессора Intel 4004 размер машинного слова составлял 4 бита. У ЭВМ следующих поколений размер машинного слова составлял 6, 18, 20, 36 или 48 бит. У большинства современных компьютеров (Intel x86 и др.), которые мы упоминаем в данном курсе, размер машинного слова, как мы уже упоминали выше, составляет 32 или 64 бита. Со временем размер машинного слова неуклонно увеличивается, что закономерно, так как это позволяет процессору за одну элементарную команду обрабатывать больший объём данных. В то же время увеличение машинного слова требует более развитых технологий аппаратной реализации.

Машинная арифметика

Речь пойдёт о представлении двоичных целых чисел для эффективной реализации арифметических операций. Выражаясь более точно, мы расскажем, как представляются отрицательные целые числа и как реализуется работа с ними[4].

Прежде всего отметим: математики считают, что ряд целых чисел бесконечен в обе стороны, то есть не существует самого малого и самого большого целого числа. В машинной арифметике бесконечность недопустима и всё конечно, поэтому существует минимальное и максимальное целое число. Причина этого заключается в том, что целое число размещается в машинном

[4] В рамках данного курса мы опустим вопрос машинного представления чисел с плавающей точкой, а также арифметических операций над ними. Желающие могут ознакомиться с этой тематикой в источ-никах, перечисленных в списке литературы к этой лекции.

слове и, соответственно, максимальное и минимальное значения целого числа ограничены размером машинного слова.

Предположим, что размер машинного слова равняется восьми. Тогда мы имеем следующее представление восьмибитовых двоичных чисел:

$$1_{10} = 0000\ 0001_2$$
$$10_{10} = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 0000\ 1010_2$$
$$21_{10} = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 0001\ 0101_2$$

Следует отметить, что оставшиеся незадействованными старшие разряды заполняются нулями и получается, что любое целое число имеет размер восемь бит. Следовательно, мы можем всегда использовать один байт для хранения целых чисел. Но очевидно, что таким образом мы можем оперировать целыми числами, не превышающими самого большого числа, которое можно представить в восьми битах. Вот это число:

$$1111\ 1111_2 = 1 * 2^8 + 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 511_{10}$$

Однако мы пока говорили о неотрицательных целых числах. Для представления отрицательных целых чисел резервируется старший бит — он делается знаковым: если этот бит равен 0, то перед нами неотрицательное число, а если он равен 1, то перед нами отрицательное целое число. Ниже представлены восьмибитовые отрицательные числа.

$$-1_{10} = 1000\ 0001_2$$
$$-10_{10} = 1000\ 1010_2$$
$$-21_{10} = 1001\ 0101_2$$

Понятно, что, занимая старший разряд для обозначения знака числа, мы уменьшаем значение максимально допустимого 8-битового целого числа — теперь оно составляет

$$0111\ 1111_2 = 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 255_{10}$$

Соответственно, минимально допустимое восьмибитовое целое число будет равно –255. Отметим, что представленный способ представления целых чисел называется *прямым представлением*. Его недостатком является необходимость обрабатывать знаковый разряд специальным образом, отличным от того, как обрабатываются остальные разряды восьмибитового целого числа. Для того чтобы решить эту проблему, введём понятие *дополнительного представления* целого числа.

Дополнительное представление неотрицательного числа (то есть числа, старший разряд которого равен 0) тождественно прямому представлению. Если же мы имеем двоичное отрицательное число, то сначала инвертируем все двоичные разряды этого числа (то есть вместо 1 записываем 0, вместо 0 записываем 1). Знаковый разряд мы оставляем неизменным. После этого мы прибавляем к модулю данного числа единицу. Если получаем переполнение в предпоследнем старшем разряде, то единицу переноса мы отбрасываем.

$$-\ 1_{10} = 1000\ 0001_{2\ пр} = 1111\ 1111_{2\ доп}$$
$$-\ 10_{10} = 1000\ 1010_{2\ пр} = 1111\ 0110_{2\ доп}$$
$$-\ 21_{10} = \ 1001\ 0101_{2\ пр} = \ 1110\ 1011_{2\ доп}$$

Теперь мы получаем следующее преимущество при выполнении операции вычитания по сравнению с использованием прямого представления: $A_1 - A_2 = \ A_1 + (-A_2)$. То есть мы заменяем операцию вычитания сложением с отрицательным вычитаемым (из школьного курса математики мы можем помнить, что первый аргумент операции вычитания называется уменьшаемым, а второй — вычитаемым). Ниже представлен пример.

$$5_{10} - 10_{10} = 0000\ 101_{2\ пр} - 000\ 1010_{2\ пр} = 0000\ 0101_{2\ доп} + \ 1111\ 0110_{2\ доп} =$$
$$= \ 1111\ 1011_{2\ доп} \ = \ 1000\ 0101_{2\ пр} \ = \ -\ 5_{10}$$

Единицы измерения цифровых данных

$2^{10} = 1024$ бит принято называть *килобитом* (Kbit), поскольку 1024 очень близко к 1000 и тут срабатывает аналогия с другими единицами измерения — с километром, в котором 1000 метров, килограммом, в котором 1000 грамм, и т. д. Килобиты, точнее, килобиты в секунду (Kbit/s), используются при указании скорости передачи информации — например, для факсимильных аппаратов и модемов[5].

$2^{10} = 1024$ байт принято называть *килобайтом* (Kb). Не так давно в килобайтах измеряли различные виды памяти ЭВМ: оперативную память, постоянное запоминающее устройство, видеопамять и т. д. Сейчас для этих целей применяются более крупные единицы (см. ниже).

$2^{20} = 1\ 048\ 576 \cong \ 10^6$ байт равно примерно одному миллиону байт и составляет один *мегабайт* (Mb).

[5] Для обозначения числа $2^{10} = 1024$ в 1998 г. Международной электротехнической комиссией была стандартизована *бинарная приставка* «киби». Аналогично, числу 2^{20} соответствует «меби-», а не «мега-», 2^{30} — «гиби-», а не «гига-» и т.д. Но десятичные приставки применительно к соответствующим степеням 2 успели «прижиться» раньше появления двоичных, поэтому именно они повсеместно используются до сих пор.

$2^{30} = 1\,073\,741\,824 \cong 10^9$ байт равно примерно одному миллиарду байт и составляет один *гигабайт* (Gb). Несколько гигабайт составляют объём оперативной памяти современного типового персонального компьютера, несколько десятков гигабайт — объем современного флэш-накопителя (флэшки). Такие объёмы появились в обиходе в связи с активным использованием видеоданных и появлением высокоскоростного Интернета. Приведём следующий интересный факт о гигабайтах и флэш-накопителях. Файловая система FAT-32, используемая сегодня на многих небольших флэш-накопителях, не поддерживает работу с файлами размером более 4 гигабайт. Соответственно, возникают проблемы при копировании на такие флэш-накопители файлов с видео в высоком качестве (часто размер таких файлов превышает 4 гигабайта). При необходимости работы с крупными файлами следует использовать другие файловые системы (например, NTFS или exFAT).

$2^{40} = 1\,099\,511\,627\,776 \cong 10^{12}$ байт равно одному триллиону байт и составляет один *терабайт* (Tb). Типичный объём жесткого диска современного настольного компьютера составляет 0,5–2 терабайта.

$2^{50} = 1\,125\,899\,906\,842\,624 \cong 10^{15}$ байт — один *петабайт* (Pb). Нетрудно посчитать, что это составляет 1024 терабайт. Данными, измеряемыми в этих единицах, оперируют уже не отдельные люди, а крупные организации и дата-центры. Петабайт — действительно крупная единица: для примера, один петабайт составляет непрерывная видеозапись хорошего качества длительностью около полутора месяцев. Но в отдельных случаях и такие объёмы оказываются слишком малы. Так, датчики Большого Адронного Коллайдера во время экспериментов могут генерировать до петабайта данных ежесекундно. На жёстких дисках подобное количество данных хранить уже слишком дорого, поэтому для этого часто используются менее удобные в обращении, но гораздо более дешёвые и ёмкие магнитные ленты.

Вопросы

1. Почему двоичная система исчисления является базовой для представления данных в ЭВМ?
2. Чем число отличается от цифры?
3. Составьте алгоритм перевода десятичной записи числа в двоичную.
4. Обобщите этот алгоритм до произвольной k-ичной системы исчисления.
5. Докажите единственность k-ичного представления любого натурального числа.
6. Дайте определения бита.
7. Дайте определение байта.
8. Дайте определение машинного слова. Приведите примеры длины машинного слова для разных процессоров.
9. Какие характеристики ЭВМ определяет машинное слово?
10. Опишите прямое машинное представление целого числа.

11. Каков недостаток прямого представления целого числа?
12. Каким будет максимально возможное значение 16-битового целого числа в случае, если отрицательные числа нас не интересуют?
13. Какими будут максимально и минимально возможные целые 16-битовые числа?
14. Что такое знаковый разряд?
15. Опишите алгоритм построения дополнительного кода для целого числа.
16. Приведите примеры вычитания двух двоичных чисел на основе дополнительного кода.
17. Каким множителям при обозначении количества данных соответствуют приставки кило-, мега-, гига- и тера-?

Литература

1. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.

Лекция 4. Устройство современного настольного компьютера

Принцип модульности и его использование при конструировании технических систем; печатные платы и интегральные схемы; устройство ЭВМ на примере настольного компьютера Dell OptiPlex: системная плата, процессор, оперативная память, контроллеры и порты, жёсткий диск и др.

Принцип модульности

Современные технические системы, например автомобили, самолеты, пароходы, бытовые приборы и, разумеется, компьютеры, состоят из множества отдельных деталей. Но если, например, бытовая техника обычно разбирается только для ремонта и обслуживания, то компьютеры могут также разбираться и с целью модернизации (*Upgrade*). Такие устройства, как графический адаптер, оперативная память, жёсткий диск, процессор, могут подключаться к другим компонентам компьютера при помощи стандартных разъёмов и фиксируются при помощи стандартных механических креплений (защёлок, винтов и т. д.) В итоге покупатель может в момент приобретения компьютера сэкономить часть денежных средств, а потом заменять отдельные детали, повышая производительность и другие характеристики ЭВМ.

Поскольку различные детали компьютера сопрягаются друг с другом стандартными способами, их установка и замена обычно не требуют специальных знаний и доступны продвинутым пользователям. Наличие достаточного опыта позволяет и вовсе собрать компьютер из этих компонент самостоятельно.

В отношении готовых компьютеров возможность модернизации может ограничиваться условиями гарантии производителя. Например, не лишаясь гарантии, компьютеры Apple можно модернизировать лишь в авторизованных сервисных центрах. Для компьютеров, совместимых с IBM PC (большинство современных персональных компьютеров), условия гарантии обычно мягче, однако многие производители также не одобряют самостоятельную замену деталей.

Исторически возможность самостоятельной модернизации стала одним из существенных факторов популярности IBM PC в начале 1980-х годов. Достаточно быстро этой возможностью стали пользоваться не только пользователи и компания IBM, но и сторонние производители оборудования, которые стали самостоятельно выпускать компоненты для IBM PC, совместимые с оригинальными, а следом и компьютеры, совместимые с IBM PC.

Итак, в современном индустриальном производстве вычислительной техники разные модули систем производятся разными компаниями-производителями. Например, компания Intel производит только процессоры, а уже другие компании выпускают итоговые ЭВМ с использованием этих процессоров. Если заглянуть глубже, то детали, из которых состоят микросхемы, в том числе и процессоры, также могут производиться различными специализированными компаниями, а производители микросхем уже закупают и используют эти готовые детали. На сегодняшний день принцип модульности глубоко проник в индустрию, и он важен не столько потому, что пользователь может сам выполнить модернизацию своего компьютера, но также потому, что в разработке целевого изделия могут участвовать различные компании-производители. Кроме того, важно, что одни и те же детали могут использоваться в различных целевых системах. Всё это существенно увеличивает производительность индустрии в целом, поскольку производителям нет нужды делать все детали для своих систем самостоятельно.

Интегральные схемы и печатные платы

Многие компоненты ЭВМ, реализующие сложные логические функции — процессор, запоминающие устройства, различные контроллеры и т. д. — производятся в виде интегральных схем.

> ***Интегральная схема (микросхема, чип, Chip)*** *— это электронная схема, изготовленная на основе полупроводниковых технологий и размещённая на одной кремниевой пластине.*

Будучи единым целым, интегральная схема не допускает обслуживания и ремонта: при выходе из строя она может быть заменена лишь целиком. Но именно «интегральность» позволила наладить эффективный индустриальный серийный выпуск недорогих интегральных схем. Более того, они также имеют небольшие размеры: вспомним, что большой размер первых ЭВМ существенно усложнял их обслуживание и радикально сокращал число их пользователей. На рис. 10 приведен пример интегральной схемы.

Рис. 10. Интегральная схема — микропроцессор Intel Core i5

Ряд компонент современных ЭВМ представляют собой единое целое с точки зрения решаемых функций и связей между собой и состоят из множества микросхем, а также отдельных электронных элементов, таких как транзисторы, конденсаторы и т. д. Для удобства интеграции все эти элементы помещают на отдельные печатные платы.

> ***Печатная плата*** *позволяет интегрировать набор микросхем и одиночных электронных элементов посредством размещения на пластине из диэлектрика, на поверхности которой расположены проводники, позволяющие соединять интегрируемые элементы друг с другом.*

В роли диэлектрика в печатных платах обычно выступают различные виды стеклопластика. Термин «печатная» связан с технологией изготовления плат, о которой пойдёт речь ниже.

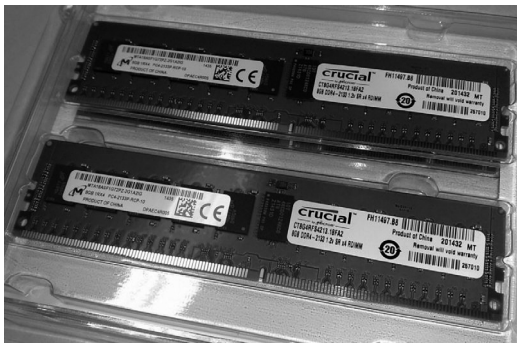


Рис. 11. Две печатные платы с оперативной памятью объёмом по 8 Gb

Важно понимать, что печатная плата выполняет не только связующую роль в плане передачи сигналов по проводникам, но и является важным конструктивным элементом. Именно платы составляют значительную часть современных электронных устройств — компьютеров, роутеров, модемов, телевизоров и смартфонов (в смартфоне плата обычно одна). И именно они обеспечивают модульность компьютеров: большинство компонент компьютера, допускающих замену пользователем, таких как оперативная память (рис. 11) или видеокарта, выпускаются и продаются в виде печатных плат. Более того, печатные платы могут вставляться друг в друга, как, например, оперативную память вставляют в материнскую плату компьютера.

Устройство настольного компьютера Dell OptiPlex

Далее в качестве примера реальной ЭВМ мы рассмотрим типичный персональный компьютер Dell OptiPlex 7060, предназначенный для выполнения офисных задач. Готовый для работы компьютер (рабочее место) включает

в себя системный блок, а также ряд периферийных устройств, например, монитор, клавиатуру и мышь. Здесь мы рассмотрим только системный блок, который является основным элементом персонального компьютера.

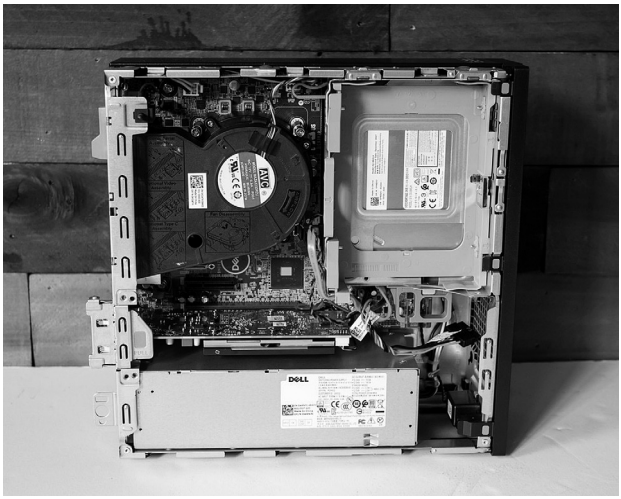


Рис. 12. Системный блок Dell OptiPlex 7060 со снятой крышкой

На рис. 12 представлена фотография системного блока компьютера Dell OptiPlex 7060 со снятой крышкой. Видна часть материнской платы, вентилятор (кулер) для охлаждения материнской платы, отсек для жёстких дисков с одним диском, блок питания, а также дополнительный графический адаптер.

Системный блок Dell OptiPlex 7060 состоит из элементов, перечисленных ниже.

- Материнская (системная) плата (System Board, Mother Board).
- Центральный процессор (Processor).
- Оперативная память (Random Access Memory).
- Постоянное запоминающее устройство (Read-Only Memory, ROM).
- Чипсет (Chipset).
- Графический адаптер (Graphics Adapter).
- Разъёмы расширения (Extension Slots).
- Жесткий диск и (HDD, Hard Disk Drive).
- SATA-контроллеры (Serial ATA Controllers).
- Встроенный звуковой адаптер (Sound Adapter).
- Телекоммуникационные устройства (Network Adapters).
- Контроллеры внешних портов (External Device Port Controllers).
- Память для настроек компьютера.
- Часы и элемент питания (батарейка).
- Оптический привод (Optical Disk Drive).

- Система электропитания (Power Supply System).
- Вентиляторы (кулеры, Fans, Coolers).

Давайте рассмотрим эти элементы подробно.

Материнская плата — это печатная плата, которая является центральным устройством системного блока и всего компьютера, соединяя и координируя все остальные устройства. Последние подключаются к материнской плате при помощи пайки или путём установки в специальные разъёмы.

Центральный процессор является интегральной схемой, которая осуществляет выполнение программ, работающих на данном компьютере. В зависимости от комплектации на материнскую плату данной модели могут быть установлены процессоры Intel Core i3, i5 или i7. Это современные процессоры Intel разных ценовых категорий, обладающие разным количеством *ядер* и объёмом внутренней кэш-памяти. *Ядро (Core)* — эта отдельная часть процессора, обладающая способностью выполнять команды программы; несколько ядер позволяют выполнять одновременно несколько программ. Отметим, что чем выше номер маркировки процессора, тем более мощным является процессор, и, соответственно, тем больше у него возможностей и тем выше его цена.

Оперативная память представляет собой набор печатных плат для хранения информации, непосредственно необходимой во время работы процессора: на этих платах хранятся исполняемые в данные момент программы и их данные (рис. 11). Эти платы вставляются в специальные разъёмы на материнской плате. Меняя или подключая новые платы с оперативной памятью, можно наращивать её общий объём в компьютере. Это бывает важно, поскольку объём оперативной памяти значительно влияет на быстродействие компьютера. Компьютер Dell OptiPlex поддерживает установку от 1 до 4 таких плат с оперативной памятью общей ёмкостью от 4 до 64 Gb. Для офисных задач типичный объём оперативной памяти составляет от 8 до 16 Gb.

Помимо процессора, с оперативной памятью могут также взаимодействовать и узлы компьютера — звуковые адаптеры, различные контроллеры локальной сети Ethernet или Wi-Fi, контроллеры внешних портов. Например, процессор готовит для звукового адаптера очередную порцию цифровой аудиозаписи, помещает её в оперативную память, и, пока звуковой адаптер самостоятельно читает из памяти и воспроизводит запись, процессор может подготовить следующую порцию и заняться другой работой. Эта возможность реализуется посредством технологии DMA (Direct Memory Access).

Постоянное запоминающее устройство — это специальная печатная плата, реализующая постоянную память компьютера, и содержание этой памяти остается постоянным и не перезаписывается во время работы компьютера, при его перезагрузке и включении/выключении. Постоянное запоминающее устройство содержит BIOS и EFI — встроенные программы, предназначенные для запуска операционной системы компьютера после включения или при его перезапуске. В последнее время постоянное запоми-

нающее устройство обычно реализовано в виде флэш-памяти и допускает программное перезаписывание с целью обновления, то есть записи новых версий BIOS и EFI. Однако такие обновления выполняются редко по сравнению с тем, сколь часто выполняется запись данных на жесткий диск или в оперативную память.

Чипсет является набором интегральных схем, размещенных на материнской плате, которые реализуют *системную шину* компьютера, а также некоторые другие функции. Напомним, что системная шина — это общий канал передачи данных, при помощи которого процессор может обмениваться данными с оперативной памятью и периферийными устройствами.

Как уже говорилось в прошлых лекциях, в архитектуре фон Неймана системная шина является узким местом: через неё передаются и данные, и машинной код, и вдобавок выполняется управление и обмен данными с периферийными устройствами. Для повышения производительности системная шина эволюционировала от «набора проводов» до многоуровневого устройства. В настоящее время она устроена существенно иначе и сложнее, чем в первых компьютерах. Как рассказывалось в предыдущих лекциях, в компьютерах первых поколений она делилась на шину данных, адресную и управляющую шины. В большинстве современных компьютеров системная шина состоит из быстрой шины и медленной шины.

- *Быстрая системная шина* связывает процессор, оперативную память и встроенный в центральный процессор графический адаптер; в случае с Intel Core i3, i5 и i7 эта шина, как правило, реализуется внутри кремниевой пластины центрального процессора (для Intel-процессоров используется технология Sandy Bridge). К быстрой шине подключён *коммутатор*, через который подключённые к быстрой шине устройства связываются с медленной шиной. У процессоров Intel Core i3, i5 и i7 этот коммутатор также расположен на кристалле.
- *Медленная системная шина* (распространенный вариант реализации — технология PCI Express) расположена на материнской плате и обеспечивает взаимодействие процессора с другими устройствами, не требующими передачи данных с высокой скоростью. Медленная шина PCI Express не разделяется физически на шины данных, адреса и управления. Вместо этого она передаёт по общему каналу отдельные сообщения (пакеты), содержащие в себе необходимые данные, адреса и сигналы управления. Кроме того, она не соединяет устройства друг с другом, а подключает их к коммутатору. Такое решение удобно для добавления в компьютер разнообразных устройств — контроллеров жёсткого диска, USB-контроллеров, контроллера клавиатуры, адаптеров Wi-Fi и Ethernet и т. д.

Разделение шины на быструю и медленную целесообразно, поскольку скорость взаимодействия с устройством важно сочетать со скоростью его

работы, а последняя у разных устройств оказывается разной. При этом быстрая шина, процессор и оперативная память размещаются на плате по возможности компактно, чтобы сократить расстояния между ними.

Графический адаптер является специальным устройством, предназначенным для обработки изображения и формирования видеосигнала при выводе изображения на монитор компьютера. Сформированный видеосигнал передаётся из графического адаптера на монитор через стандартные разъёмы — HDMI и VGA, которые можно видеть с обратной стороны системного блока. В частности, в эти разъемы можно подключать вместо монитора другие устройства — например, телевизор или проектор.

Графический адаптер может быть интегрирован в центральный процессор или чипсет. Также он может быть выполнен в виде отдельного (discrete) устройства и быть вставленным в материнскую плату через специальный слот расширения. Такой адаптер может вставляться в компьютер, если производительности встроенного адаптера недостаточна — например, в случае необходимости поддержки ресурсоёмких видеоигр. Именно так обстоит дело с компьютером Dell OptiPlex 7060, изображенным на рис. 12: отдельный графический адаптер вставлен в разъём синего цвета на материнской плате, несмотря на то, что в компьютерах семейства Intel x86 Intel Core i3, i5, i7 и выше графический адаптер уже интегрирован в центральный процессор.

Современный графический адаптер имеет собственный процессор, который может быть использован не только по прямому назначению, то есть для обработки видеоизображений, но и для произвольных вычислений. Такая возможность обеспечивается GPGPU-технологиями (General Purpose Graphical Processing Unit), позволяющими программистам использовать графический процессор компьютера для вычислений, которые в обычной ситуации выполняет центральный процессор. Такие технологии оказываются полезными, когда возникают проблемы с производительностью, — например, при моделировании физических процессов, в криптографических задачах, при обучении и использовании нейронных сетей, при решении графовых задач методами линейной алгебры и т. п.

Разъёмы расширения позволяют подключать к материнской плате дополнительные устройства, которыми данный компьютер не комплектуется, — например, устройство видеозахвата или адаптер для оптической локальной сети.

Графический адаптер и контроллеры различных устройств могут быть встроены в материнскую плату, но не обладать достаточной производительностью для решения задач пользователя данного компьютера, а могут и вовсе отсутствовать. В таких случаях более мощные аналоги можно установить в разъёмы расширения. Например, встроенные графические адаптеры хорошо подходят для решения офисных задач, но бывают недостаточно производительны для компьютерных игр, и тогда в системный блок, в слот расширения, вставляется отдельный, вынесенный графический адаптер.

Жесткий диск используется для долгосрочного хранения данных и программ. В отличие от оперативной памяти, быстродействие жёсткого диска существенно ниже, а объём хранимых данных — значительно больше. Программы (приложения), установленные на компьютере, хранятся именно на жестком диске, а перед выполнением загружаются в оперативную память. В зависимости от комплектации данная модель ЭВМ допускает жесткий диск объёмом от 500 Gb до 2 Tb.

Жесткий диск часто называют винчестером. *Винчестер* — это сленговый термин, который появился в 1970-х годах в компании IBM для обозначения жесткого диска IBM 3340. Максимальная конфигурация этого изделия включала два таких жёстких диска объемом по 30 Mb. Два числа 30 вызвали у разработчиков изделия ассоциацию с известной охотничьей винтовкой Winchester, патроны к которой так и назывались — 30–30. Первое число означало калибр в сотых долях дюйма, второе — вес порохового заряда в специальных единицах (гранах). В дальнейшем этот термин стал использоваться для названия всех жестких дисков.

Вместо жёсткого диска на компьютере Dell OptiPlex 7060 может быть установлен твердотельный накопитель (SSD, Solid State Drive) объёмом от 512 Gb до 1 Tb. Твердотельный накопитель хранит данные в виде флэш-памяти большого объёма. Он стоит дороже механического жёсткого диска, но обладает большим быстродействием.

SATA-контроллеры управляют энергонезависимыми накопителями данных — жёсткими дисками, твердотельными накопителями, оптическими приводами. Эти накопители используются для хранения программ и данных — как системных, так и пользовательских, причём, в отличие от оперативной памяти, они не «забывают» информацию, когда компьютер выключен. SATA-контроллеры поддерживают механизм DMA (Direct Memory Access), который позволяет им связываться с оперативной памятью без участия центрального процессора: благодаря этому механизму, пока контроллер загружает данные с накопителя в память или сохраняет данные из памяти на накопителе, процессор может выполнять другую работу.

Встроенный звуковой адаптер является устройством, которое реализует работу компьютера со звуком: поддерживает воспроизведение (проигрывание) звукозаписи в динамики или наушники, а также позволяет выполнять запись аудиоматериала с микрофона. Современные ноутбуки комплектуются встроенными динамиками и микрофоном.

Сетевые адаптеры позволяют подключать компьютер к компьютерным сетям. У Dell OptiPlex 7060 имеются адаптеры для подключения к самым распространённым видам сетей — проводной сети Ethernet и беспроводной сети Wi-Fi.

Контроллеры внешних портов являются устройствами, управляющими внешними портами компьютера, которые обычно выведены с обратной стороны системного блока — один контроллер на один или несколько портов. Dell OptiPlex имеет следующие порты: 10–12 USB-портов (в зависимо-

сти от комплектации), два PS/2-порта, предназначенных для подключения мыши и клавиатуры, а также несколько других видов портов для совместимости со старыми устройствами[6]. К USB-портам могут подключаться любые совместимые с ними внешние устройства — например, принтер или сканер.

Значение термина «порт» в вычислительной технике варьируется в зависимости от контекста. Портом называют и разъём на системном блоке компьютера, и способ сопряжения контроллера внешнего устройства с системной шиной, и точки соединения произвольных устройств, расположенных на некоторой плате (не обязательно материнской), с различными каналами, и, наконец, произвольную «внешнюю розетку» различного электронного оборудования.

Память для настроек компьютера представляет собой особый вид памяти, предназначенный для хранения основных настроек ЭВМ, используемых при её запуске. Речь идёт о состоянии адаптера Wi-Fi, вариантах загрузки операционной системы и т. д. Эти настройки требуется сохранять после выключения компьютера из электрической сети. Для их хранения не используется жесткий диск, поскольку часть этих настроек используется при его инициализации. Более того, в некоторых комплектациях рассматриваемый компьютер может загружаться по локальной сети и хранить свои данные там же, соответственно, жёсткого диска у него может не быть вовсе. В компьютере Dell OptiPlex 7060 память для настроек реализована с помощью технологии CMOS (Complementary Metal-Oxide-Semiconductor), которая требует для своей работы постоянного электропитания, что обеспечивается батарейкой. Часто эту память так и называют — CMOS, хотя для её реализации используются и другие технологии — например, флэш-память.

Часы и источник питания (батарейка). Часы (и календарь) нужны компьютеру для правильной обработки данных (например, у каждого файла записаны время создания и изменения), для работы офисных программ (например, напоминания о встречах), а также для системных утилит (например, периодического запуска антивируса) и пр. Для часов необходим автономный источник питания (батарейка), чтобы не переустанавливать их всякий раз после запуска компьютера.

Оптический привод является дисководом для работы с оптическими дисками, то есть дисками CD (сейчас они используются всё реже) и DVD. Изначально (а это был конец 1970-х годов) технология оптических дисков предназначалась для хранения звукозаписей. Позже эта технология была

[6] Речь идет о последовательных и параллельных портах, которые предназначаются для подключения периферийных устройств, в основном произведённых 20 лет назад и ранее. Последовательный порт использовался для подключения к компьютеру устройств, не требующих передачи больших объемов данных и высокой скорости взаимодействия (мышь, факс-модем), а параллельный порт использовался для подключения более высокоскоростных устройств (сканер, принтер). Аналогичные современные устройства обычно подключаются к компьютеру при помощи USB-портов, а у многих современных ПК последовательных и параллельных портов нет вовсе.

усовершенствована и стала применяться для хранения видеоданных, а потом и произвольных данных.

Оптический диск состоит из нескольких слоёв. Наружные слои, выполненные из пластика, обеспечивают его механическую прочность, а внутренние, изготавливаемые из различных металлов или сплавов, предназначаются для хранения данных. Чтение и запись на оптические диски производятся при помощи лазерного луча. Участки диска, в зависимости от того, записаны ли на них нули или единицы, по-разному отражают свет лазера, что и используется для чтения данных: лазер подсвечивает нужный бит, а фотодатчики по отражённому сигналу определяют его значение. Для записи существуют различные технологии, но все они основаны на принципе изменения оптических свойств внутреннего слоя диска при помощи воздействия более мощным, чем при чтении, лазерным лучом.

Система электропитания преобразует напряжение электрической сети в набор питающих напряжений для ПК, а также имеет набор кабелей питания и разъёмов, подключаемых к разным устройствам.

Вентиляторы предназначены для отвода тепла от узлов компьютера. Современный компьютер выделяет значительное количество тепла в силу следующих причин.

- При прохождении тока через резисторы и полупроводниковые элементы они выделяют тепло.
- КПД механических частей жёсткого диска, а также системы электропитания меньше 1. Жёсткий диск не всю поступающую ему электроэнергию переводит в механическую, и подсистема электропитания компьютера не всю потребляемую электроэнергию преобразует по назначению. «Остаток» энергии эти подсистемы преобразуют в тепло, в результате чего также существенно нагреваются при работе.

При этом все компоненты компьютера достаточно требовательны к температурному режиму: при выходе за пределы рабочих температур меняются характеристики полупроводников, что приводит к отказу электроники, а при сильном перегреве может расплавиться *оловянный припой*, используемый для соединения микросхем с системной платой. Чтобы поддерживать рабочую температуру, в Dell OptiPlex 7060 имеется два кулера: один является общим на весь системный блок, второй обеспечивает охлаждение центрального процессора.

Вопросы

1. В чем заключалось преимущество модульности при появлении и развитии компьютеров IBM PC?
2. В чём заключается значение принципа модульности сегодня?
3. Что такое системная плата?
4. Каковы типичные объёмы оперативной памяти современных офисных ПК?
5. Для чего используется технология DMA?
6. Что такое чипсет?
7. В чём отличие встроенного и отдельного графических адаптеров?
8. Назовите достоинства и недостатки механических жёстких дисков.
9. Назовите достоинства и недостатки твердотельных накопителей.
10. Что такое флэш-память и где она используется в современных компьютерах?
11. Какие данные с и какой целью хранятся в памяти для настроек компьютера?
12. Почему эти данные нельзя хранить на жестком диске?
13. Расскажите про историю термина «винчестер».
14. Каковы типичные объёмы механических и твердотельных накопителей?
15. Назовите различные типы внешних портов.
16. Как называются устройства, обеспечивающие работу внешних портов компьютера?
17. Для каких целей используются быстрая и медленная системные шины?
18. Опишите принцип считывания и записи, используемый оптическими дисками.
19. Зачем нужна система охлаждения компьютера? Какие проблемы она решает?

Литература

1. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
2. OptiPlex 7060 Small Form Factor Service Manual. Dell Inc, 2018. https://topics-cdn.dell.com/pdf/optiplex-7060-desktop_service-manual2_en-us.pdf.

Лекция 5. Интегральные схемы и печатные платы

Полупроводники и полупроводниковые элементы; производство интегральных схем: полупроводниковые свойства кремния, изготовление кремниевых пластин, фотолитография; производство печатных плат: фотолитография, склеивание слоёв.

Полупроводники и полупроводниковые элементы

Напомним о том, что первые ЭВМ конструировались с использованием радиоламп и электромагнитных реле, но после изобретения транзистора в конце 1940-х годов произошла смена поколений, и начиная со второго поколения ЭВМ создавались уже на базе полупроводниковых элементов. Последние обеспечили существенный прогресс вычислительной техники, так как позволили радикально уменьшить размеры компьютеров, также уменьшить затраты электроэнергии и повысить надёжность ЭВМ. Кроме того, полупроводниковые компьютеры оказались существенно дешевле компьютеров предыдущих поколений. Полупроводниковая элементная база используется и в настоящее время.

Обратимся к курсу школьной физики и вспомним, что такое полупроводники.

> ***Полупроводник*** *— это вещество, электрическая проводимость которого существенно зависит от состава примесей, добавленных к нему; также его проводимость может существенно меняться под воздействием температуры, электрических и магнитных полей.*

Полупроводники с различными характеристиками производятся путём добавления примесей в кристаллические вещества (обычно в кремний). Сочетая различные полупроводники, изготовляют транзисторы, диоды, тиристоры и т. д. Полупроводниковые элементы в электронных схемах усиливают и преобразуют электрические сигналы. В частности, способность транзисторов менять свою проводимость в зависимости от характеристик управляющего электрического сигнала позволяет создавать на их основе логические вентили.

Электрическая проводимость вещества обеспечивается электронами на внешней оболочке его атомов. Эта же оболочка в основном опре-

деляет и *валентность* вещества — способность его атомов связываться с атомами других веществ и, для кристаллических веществ, способность формировать кристаллическую решётку. Поэтому электроны, находящиеся на внешней электронной оболочке, называются *валентными*. Металлы легко обмениваются валентными электронами, что позволяет им быть *проводниками*, то есть иметь высокую *электрическую проводимость*. Чистые полупроводники, не содержащие примесей, имеют гораздо менее «подвижные» валентные электроны и, как следствие, низкую электрическую проводимость.

Остановимся подробнее на технологии изготовления полупроводниковых элементов. Основным химическим элементом, используемым при их производстве, является кремний — природный полупроводник. Кремний является не единственный природным полупроводником, но он — один из самых распространенных химических элементов на Земле и составляет более 27% земной коры. Поэтому его легко добывать, а также несложно перерабатывать — чистый технический кремний эффективно и недорого производится из оксида кремния, то есть песка (см. рис. 13).

Как уже обсуждалось выше, собственная электрическая проводимость кристаллического кремния мала, поскольку все валентные электроны атомов кремния задействованы в кристаллической решётке, реализуя связи с четырьмя соседними атомами. Для изменения электрических свойств кремния в него добавляют *легирующие примеси*. Эти примеси позволяют «расшевелить» часть его валентных электронов, увеличить таким образом электрическую проводимость кремния, а также повлиять на характер этой проводимости. В качестве примесей используются фосфор, мышьяк, бор и некоторые другие элементы с подходящей конфигурацией электронных оболочек. В частности, у мышьяка пять валентных электронов, поэтому при добавлении небольшого количества мышьяка в кремний, который четырехвалентен, один из его электронов остаётся свободным. Такие свободные электроны могут перемещаться от одного атома мышьяка к другому, перенося заряд, и в итоге кремний, легированный мышьяком, приобретает свойство *электронной проводимости* или *n-проводимости*. Бор, напротив, имеет три валентных электрона, поэтому кремний с примесью бора содержит свободные места, которые могут занимать электроны близлежащих атомов. В этом случае по кристаллу перемещаются, перенося заряд, уже не электроны, а так называемые положительно заряженные дырки, а кремний приобретает *p-проводимость,* или *дырочную проводимость*. Отметим, что, будучи «отсутствием электрона», дырка является не физическим объектом, а удобной умозрительной абстракцией, фактически же заряд по-прежнему переносит электрон.

В транзисторах, диодах и других полупроводниковых элементах *p*- и *n*-полупроводники соединяются, и вблизи их границы образуется

p-*n*-переход, который может реагировать на электрическое поле, «открывая» или «запирая» движение электрического тока. На основе одного *p*-*n*-перехода изготавливается полупроводниковый *диод*, пропускающий ток лишь в одном направлении — от *p*-полупроводника к *n*-полупроводнику. Если полупроводники чередовать в последовательности *n*-*p*-*n*, то получится *NPN-транзистор*, который проводит ток между крайними *n*-полупроводниками (открывается) только при подаче высокого электрического потенциала на средний p-полупроводник. Его «близнец» — *PNP-транзистор* — работает наоборот: проводит ток между крайними *p*-полупроводниками только при подаче низкого потенциала на *n*-полупроводник.

Следует отметить, что работоспособный транзистор с характеристиками, допускающими его практическое использование, получается лишь при соблюдении множества дополнительных условий — толщины и объёма полупроводников, концентрации легирующих примесей и т. д. Исследования свойств полупроводников проводились физиками в различных странах с 1920-х годов, но биполярный транзистор, аналогичный современным, был создан лишь в 1947 году. В 1956 году создатели биполярного транзистора У. Шокли, У. Браттейн и Д. Бардин были удостоены Нобелевской премии. Исследования полупроводников и их применения в микроэлектронике продолжаются и сейчас. Один из самых известных современных учёных в этой области — наш соотечественник академик Ж. И. Алфёров (1930–2019), в 2000 году также удостоенный Нобелевской премии по физике.

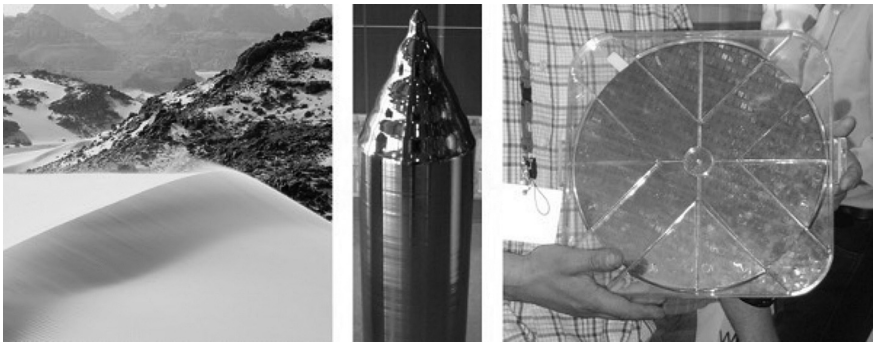


Рис. 13. Оксид кремния — песок в пустыне (слева), монокристалл кремния (по центру) и срез монокристалла с заготовками для микросхем на нём (справа)

Кроме кремния в природе имеются другие природные полупроводники — например, германий. Существуют транзисторы и интегральные схемы на основе германия, но доступность кремния и наличие отработанных технологий производства сделали его предпочтительным базовым материалом.

Технология производства интегральных схем

Хотя отдельные транзисторы, диоды и прочие элементы и сегодня непосредственно монтируются на печатные платы, основой функционал современных компьютеров реализуют уже интегральные схемы. Они группируют на кремниевых пластинах огромное количество транзисторов, диодов и прочих элементов, создавая специализированные устройства — например, процессоры. Интегральные схемы позволили существенно удешевить компьютеры, а также уменьшить их размеры и увеличить производительность.

> ***Монокристалл*** *— это состояние кристаллического вещества, которое характеризуется крайне незначительным количеством нарушений геометрии его кристаллической решётки, то есть почти все атомы решётки расположены в идеальном порядке.*

В природе монокристаллы практически не встречаются. Но для производства микроэлектроники они важны, поскольку размеры создаваемых на базе кремния электронных элементов уже сопоставимы с расстоянием между атомами в кристаллической решётке (около 0,5 нм). Следовательно, отдельные дефекты в кристалле могут привести к появлению неработоспособных полупроводниковых элементов.

Чтобы сделать природный кремний пригодным для использования в изготовлении печатных плат, его очищают от примесей, достигая концентрации кремния, близкой к 100%. Далее создают (выращивают) монокристаллы кремния в виде цилиндров. Эти цилиндры режут на пластины, на которых и размещают интегральные схемы (см. рис. 13). Сейчас изготовляют цилиндры диаметром в 300 миллиметров, но разрабатываются технологии для создания 450 миллиметровых цилиндров. Увеличение диаметра цилиндров потенциально позволяет увеличить площадь интегральной схемы, и, следовательно, разместить на одной пластине больше транзисторов. Это, в свою очередь, позволяет выпускать более высокопроизводительные процессоры. Но увеличение диаметра кремниевых кристаллов сопряжено с большими технологическими трудностями.

Для того чтобы сформировать на кремниевой пластине отдельные электронные элементы, используется специальная высокоточная технология под названием фотолитография.

> ***Фотолитография*** *— это технология нанесения рисунка на различные поверхности посредством следующих действий: сплошное нанесение «красящего» вещества на поверхность, нанесение защитного слоя, фотохимическая стабилизация защитного слоя в необходимых местах, растворение «красящего» вещества на остальных участках поверхности (то есть там, где оно не было стабилизировано).*

При производстве интегральных схем фотолитография выполняется следующим образом. На кремниевые пластины напыляются легирующие примеси, которые, как и в случае с изготовлением отдельных полупроводниковых элементов, создают на поверхности кремния участки с *p*- или *n*-проводимостью. Вначале создаются все участки с *p*-проводимостью. Для этого вся поверхность кристалла на небольшую глубину легируется *p*-примесью. Следом кристалл покрывается *фоторезистом* — веществом, которое приобретает химическую устойчивость к растворителю под действием света. Затем фоторезист «засвечивается» через специальный оптический трафарет, благодаря чему нужные участки кремниевой пластины приобретают химическую устойчивость. Наконец, растворителем смывается незасвеченный фоторезист и примеси на незащищённых участках. В итоге *p*-примесь фиксируется в верхних слоях кремния только на нужных участках будущей интегральной схемы. После синтеза *p*-участков этот процесс повторяется с другим трафаретом, уже для *n*-примеси. В результате разные участки кремниевой пластины легируются разными примесями. Перечисленные выше шаги могут повторяться несколько раз или даже несколько десятков раз. Как и в случае с отдельными полупроводниковыми элементами, на границах этих участков образуются *p*-*n*-переходы, на основе которых создаются миниатюрные транзисторы, диоды и другие элементы. Следует отметить, что в зависимости от технологии, используемой на конкретном производстве, синтез *p*- и *n*-участков на кремниевой пластине может выполняться в другом порядке. Также возможно применение фоторезистов, которые, напротив, теряют устойчивость под действием света.

В современных процессорах используется большое количество полупроводниковых элементов, количество которых исчисляется миллиардами. Разумеется, все они работают не сами по себе, а в связке друг с другом. Подобно тому как в электромеханических приборах для соединения отдельных элементов используются провода, для соединения полупроводниковых элементов интегральной схемы на кремниевую пластину наносятся проводники. Эти проводники являются токопроводящими дорожками из атомов металла с высокой электрической проводимостью, например, из золота.

Нанесение на кремниевую пластину легирующих примесей и проводников производится в несколько слоёв. Эта многослойность необходима, поскольку даже простейшие электрические схемы далеко не всегда можно сделать «плоскими». Например, если сложную схему расположить на плоскости, то различные её соединения будут пересекаться, не образовывая при этом электрического соединения в месте пересечения. Но этого можно достичь, если у схемы появляется третье измерение, то есть появится возможность разместить одно из пересекающихся соединений глубже — и тогда оно перестанет пересекать предыдущее, но общая геометрия схемы при этом сохранится.

На сегодняшний день достигнуты большие успехи в миниатюризации электронных элементов. В первые десятилетия производства интегральных схем именно эта миниатюризация обеспечивала выполнение закона Мура, но сейчас этого уже недостаточно: в дальнейшем уменьшить в 100 раз размер транзистора, который и сейчас состоит из нескольких десятков атомов, уже не удастся. Дальнейшее увеличение производительности ЭВМ (и по-прежнему исполнение закона Мура) осуществляется путём распараллеливания — размещения на кристалле блоков процессора, работающих параллельно.

Технология производства печатных плат

Основными компонентами современных вычислительных устройств являются печатные платы. На платах размещают интегральные схемы (например, процессоры) и отдельные элементы (транзисторы, диоды, резисторы, конденсаторы). Последние обычно нужны в тех случаях, когда, например, элемент слишком велик по размеру для помещения на интегральную схему (например, конденсатор с большой ёмкостью), или когда требуется менять элемент или набор элементов в зависимости от разной версии платы (варьировать содержимое интегральной схемы при серийном выпуске нельзя). Кроме этого, на плате монтируются разъёмы для соединения с другими платами и устройствами, а также вспомогательные механические компоненты — кулеры, радиаторы. Наконец, непосредственно на плате имеется набор токопроводящих дорожек, которые в нужных сочетаниях соединяют электрические выводы перечисленных компонент между собой и с интегральными схемами, что позволяет им обмениваться сигналами друг с другом.

Печатная плата представляет собой лист диэлектрика — обычно используется *стеклопластик*, который является жёстким и прочным материалом из стекловолокна, пропитанного каким-либо полимером. На лист диэлектрика наносятся медные «дорожки», соединяющие выводы размещённых на плате интегральных схем и прочих компонент.

Кратко опишем технологию производства печатных плат.

Для изготовления печатных плат, подобно изготовлению интегральных схем, также используется технология, основанная на фотолитографии. В качестве заготовки для печатной платы используют лист стеклопластика. Его покрывают слоем меди для создания токопроводящих дорожек, соединяющих между собой различные элементы печатной платы — интегральные схемы, элементы, разъёмы и т. д. Для того чтобы сформировать из сплошного слоя меди отдельные дорожки, выполняются следующие действия. На медный слой наносится слой фоторезиста (здесь, конечно, применяется другой состав, чем при изготовлении интегральных схем). На фоторезист проецируется изображение дорожек, после чего засвеченные участки фоторезиста, под которыми скрываются будущие дорожки, становятся химически устойчивыми. Затем выполняется *травление*: плата помещается в растворитель, который растворяет медь, не защищённую фоторезистом. В итоге отсекается

всё ненужное, то есть из всего медного слоя на плате остаются только отделённые друг от друга диэлектриком медные дорожки, а также *монтажные площадки*. Они являются специальными участками платы, к которым в определённом порядке подходят «дорожки», образуя на этих участках набор электрических контактов. На контактах выполняется пайка интегральных схем, разъёмов и прочих компонент. При пайке осуществляется электрическое соединение выводов компонент (так называемых «ножек») с контактами и, следовательно, «дорожками» на плате. Также пайка обеспечивает жёсткое крепление компонент к плате за те же «ножки». В зависимости от особенностей компоненты монтаж выполняется разными способами. Когда требуется высокая механическая прочность или возможность проводить большой ток, применяется *сквозной монтаж*, при котором «ножки» компоненты проходят сквозь плату, а монтажная площадка представляет собой набор контактов с отверстиями для ножек. Сквозной монтаж применяется для крепления на плату разъёмов и крупных мощных устройств — например, реле или трансформаторов. Для мелких компонент — отдельных микросхем и элементов — высокая прочность и возможность проводить большие токи обычно не требуются. В этом случае может применяться более дешёвый и компактный *поверхностный монтаж*, заключающийся в простом припаивании ножек к контактам или непосредственно к «дорожкам».

Электрическая схема печатной платы может, как и в случае с интегральными схемами, быть объемной, то есть потребовать более одного слоя. Для этого с обратной стороны заготовки также наносится медный слой, который используется для создания дорожек, и в итоге схема становится «двухэтажной». Но для сложных печатных плат (например, современных системных плат) двух слоёв бывает недостаточно. В этой ситуации изготовляют несколько печатных плат, а затем склеивают их между собой, получая в результате единую многослойную плату. Электрические соединения между дорожками в различных слоях плат выполняют при помощи сверления сквозных отверстий в плате и заполнения этих отверстий токопроводящим припоем (оловом).

Разъёмы, интегральные схемы и прочие элементы припаиваются к монтажным площадкам на плате. Для этого также используется оловянный припой. Температура плавления олова составляет всего 231,90 °C и легко достижима при перегреве печатных плат во время работы компьютера. Перегрев электронных компонент и протекание через паяное соединение большого тока могут привести к расплавлению и вытеканию припоя, что влечет нарушение контакта. Один из простых приёмов ремонта печатных плат в такой ситуации — это повторный нагрев при помощи технического фена, в результате чего капельки припоя плавятся и затекают обратно на свои места.

Для предотвращения перегрева интегральных схем используются вентиляторы (кулеры). Они крепятся на печатную плату с помощью винтов или защелок, которые вставляются в специально просверленные на плате крепёжные отверстия.

Пример готовой печатной платы показан на рис. 14.

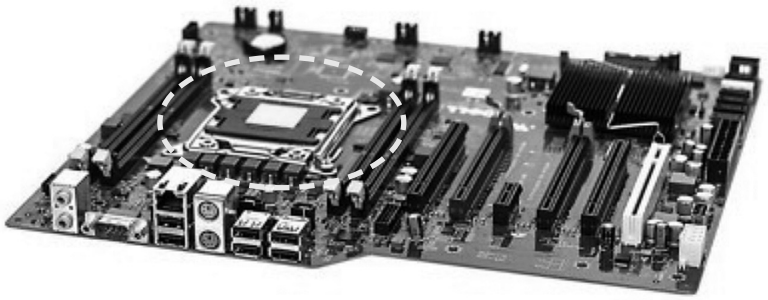


Рис. 14. Печатная плата (материнская плата Dell Precision T3600) с множеством монтированных элементов и разъёмов, в том числе разъёма для установки процессора и креплениями для радиатора с кулером

Вопросы

1. Что такое полупроводник?
2. Что такое дырочная и электронная проводимость, как ими можно управлять?
3. Что такое легирующие примеси?
4. Что такое фоторезист и для чего он используется?
5. Что такое монокристалл?
6. Что такое *p*-*n*-переход, и как он используется в электротехнике?
7. Почему именно кремний используется в электронной промышленности?
8. Как изготавливаются кремниевые подложки для интегральных схем?
9. Дайте определение фотолитографии.
10. Опишите процесс изготовления интегральной схемы по шагам.
11. Что такое печатная плата?
12. Для чего при изготовлении печатных плат используется стеклопластик?
13. Опишите технологию производства печатных плат в целом.
14. Какую функцию на печатных платах выполняют монтажные площадки?
15. Что такое оловянный припой, и для чего он используется?
16. Объясните причину перегрева печатных плат.
17. Расскажите о способе преодоления этой проблемы.
18. Расскажите о способе преодоления последствий перегрева, если он всё-таки произошёл.
19. Какие вы знаете виды монтажа печатных плат, и в каких случаях они используются?

20. Почему оказываются востребованы многослойные печатные платы?
21. Опишите процедуру изготовления многослойной печатной платы.

Литература

1. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.
2. *Свистова Т. В.* Основы микроэлектроники: учеб. пособие [электронный ресурс] — Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017.
3. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.

Лекция 6. Основы схемотехники

Булевы функции; таблицы истинности булевых функций, система базовых булевых функций; вентили.

Булевы функции

Пусть $B = \{0,1\}$. *Тогда n-местной* ***булевой функцией*** *будем называть функцию, действующую из* $B \times \ldots \times B \ldots$ *в* B.

Рассмотрим следующие простейшие булевы функции.

- $A \wedge B$ — «и» (конъюнкция), выдает 1, когда оба аргумента равны 1. Иначе выдает 0.
- $A \vee B$ — «или» (дизъюнкция), выдает 1, когда хотя бы один аргумент равен 1. Иначе выдает 0.
- $\neg A$ — «не» (отрицание), выдает 1, если аргумент равен 0, и 0, если аргумент равен 1.
- $A \to B$ — «импликация», выдаёт 0, когда первый аргумент равен 1, а второй — 0 (то есть из истины не может следовать ложь), а в остальных случаях выдает 1.
- $\mathrm{Xor}(A, B)$ (другой вариант записи — $A \oplus B$) — «исключающее или» (сложение по модулю 2), выдает 1, когда ровно один аргумент равен 1, в остальных случаях выдает 0.

Таблицы истинности булевых функций

Булевы функции, в отличие от функции в математическом анализе, являются дискретными. Более того, количество возможных уникальных сочетаний аргументов, а также соответствующих им значений является небольшим и может быть перечислено явно.

> *Значения булевых функций в соответствии со значениями их аргументов можно представить при помощи* ***таблиц истинности****. Каждая строка такой таблицы соответствует сочетанию аргументов и соответствующему значению функции.*

Таблица 1 представляет таблицы истинности для ряда известных булевых функций.

Табл. 1. Таблицы истинности для некоторых простейших булевых функций

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$\mathbf{Xor(A,B)}$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Система базовых булевых функций

Существует следующая важная теорема.

Теорема 1. Любая булева функция может быть представлена в виде суперпозиции следующих булевых функций — «и», «не» или «и», «или».

Несмотря на то, что хватает двух функций из трех, для представления произвольной булевой функции удобно использовать все эти три функции (в этом, кстати, заключается идея доказательства приведённой выше теоремы).

Например, $A \to B \ = \neg A \vee B, \mathrm{Xor}(A,B) = (A \wedge \neg B) \vee (\neg A \wedge B)$.

> *Набор булевых функций, с помощью которого можно выразить все остальные булевы функции, будем называть* ***системой базовых булевых функций***.

Наборы {«и», «не»} и {«или», «не»} являются системами базовых булевых функций. Следует отметить, что существуют и другие аналогичные системы. Например, каждая из приведенных ниже функций образует базовую систему булевых функций (то есть каждая из этих систем состоит из одной единственной функции).

- $A|B = \neg(A \wedge B)$ — штрих Шефера («и–не»).
- $A \downarrow B = \ \neg(\mathrm{A} \vee B)$ — стрелка Пирса («и–или»).

Нетрудно доказать, что других систем базовых функций, состоящих всего лишь из одной функции, больше нет.

Вентили и логические схемы

Теперь перейдём от булевых функций к внутреннему устройству компьютеров — к вентилям, логическим схемам, функциональным блокам процессора и командам машинного языка.

> *Физическая реализация простейшей булевой функции с помощью полупроводниковых элементов (транзисторов, резисторов, диодов и пр.) называется* ***вентилем*** *(Gate).*

Таким образом, вентили являются элементарными логическими схемами, и более сложные схемы, реализующие различные функциональные блоки процессора, строятся на их основе.

На рис. 15 приведены обозначения основных логических вентилей в соответствии со стандартом ANSI/IEEE91-1984. ANSI (American National Standards Institute) — американский институт, который осуществляет надзор за разработкой и использованием различных компьютерных стандартов. Так вот, в стандарте ANSI/IEEE91-1984 описаны графические изображения как вентилей, так и более сложных логических схем, а также некоторых других схем цифровой электроники. Исторически он базируется на стандартах Министерства обороны США 1950–1960-х годов — вентили ранних ЭВМ создавались на базе ламп и электромеханических реле, но обозначались на схемах аналогичным образом. Сейчас этот стандарт является международным и используется в различных сферах производства.

Теперь определим более сложный элемент под названием логическая схема.

> ***Логическая схема*** *— это функция, которая на вход получает некоторый двоичный вектор, а на выходе выдает преобразованный двоичный вектор.*

Процессор состоит из набора функциональных блоков, таких как арифметико-логическое устройство, устройство чтения машинного кода, набор регистров, кэш-память и т. д. В свою очередь, эти функциональные блоки состоят из множества различных логических схем.

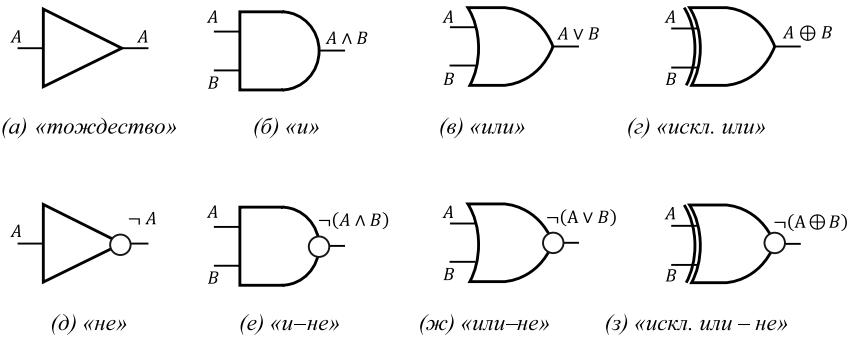
При построении логических схем используют атомарные элементы — вентили, которые являются реализацией простейших булевых функций. Демонстрация этого факта будет проведена в следующей лекции, однако уже теперь мы проведем некоторую подготовку.

Для работы основных функций процессора очень важны логико-арифметические операции. С их помощью реализуются важнейшие составные команды процессора: загрузка/выгрузка в оперативную память, присваивание, условные переходы, циклы и пр. Кроме того, различные математические функции (например, log или sin) обычно вычисляются современными

процессорами при помощи частичного суммирования рядов Тейлора. Такие вычисления не только требуют непосредственно арифметических операций, но также используют циклы, следовательно, и логические операции. В свою очередь, математические функции используются для обработки мультимедиа-данных — изображений, видео, звука. Разумеется, используются они и при выполнении ЭВМ своего «исходного предназначения» — физических и математических расчётов.

Итак, логические операции у нас имеются — это булевы функции, реализуемые вентилями. А то, как обстоит дело с арифметическими операциями, будет показано в следующей лекции, где описывается логическая схема для сложения (то есть сумматор).

Ну и, наконец, относительно небольшой набор вентилей может напомнить нам систему базовых булевых функций — с той разницей, что в такую систему добавлено больше функций, чем содержит минимальная система базовых функций, поскольку требуется строить реальные (не теоретические) логические схемы и нужны удобные строительные блоки, реализующие наиболее часто встречающиеся операции.



(а) «тождество» *(б) «и»* *(в) «или»* *(г) «искл. или»*

(д) «не» *(е) «и–не»* *(ж) «или–не»* *(з) «искл. или – не»*

Рис. 15. Схематическое изображение вентилей в соответствии со стандартом ANSI/IEEE91-1984

Вопросы

1. Дайте определение булевой функции.
2. Что такое таблица истинности? Приведите примеры.
3. Докажите, пользуясь таблицами истинности, что $A \to B = \neg A \vee B$ и $\mathrm{Xor}(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B)$.
4. Что такое система базовых булевых функций?
5. Докажите с помощью таблиц истинности, что любую булеву функцию можно представить как суперпозицию функций из системы {«и», «не»}.

6. Сделайте то же для системы функций {«или», «не»}.
7. Докажите, что стрелка Пирса являются системами базовых булевых функций.
8. Докажите, что штрих Шефера являются системами базовых булевых функций.
9. Докажите, что стрелка Пирса и штрих Шефера являются единственными системами базовых булевых функций, состоящих из одной функции.
10. Что такое вентиль?
11. Приведите примеры вентилей.
12. Что такое логическая схема?
13. Постройте логическую схему «исключающее или» на основе вентилей «и», «или» и «не».
14. Почему для процессоров важны логико-арифметические операции?

Литература

1. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.
2. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
3. IEEE Standard Graphic Symbols for Logic Functions (Including and incorporating IEEE Std 91a-1991, Supplement to IEEE Standard Graphic Symbols for Logic Functions) // IEEE Std 91a-1991 & IEEE Std 91–1984, 1984, С. 1–160.

Лекция 7. Логические схемы на примере двоичного сумматора

Одноразрядный двоичный сумматор; четырехразрядный двоичный сумматор; оптимизация сумматора с целью ускорения переноса для арифметики большой разрядности; арифметико-логическое устройство (блок) процессора.

В этой лекции мы рассмотрим пример важной логической схемы — двоичного сумматора, который выполняет сложение двух чисел, представленных в двоичной системе исчисления.

Одноразрядный двоичный сумматор

Сначала рассмотрим одноразрядный сумматор, который позволяет складывать два бита. Для этого вспомним алгоритм сложения «в столбик», универсальный для систем исчисления с любым основанием. Этот алгоритм предписывает складывать числа, начиная с младших разрядов, причём на входе каждого разряда помимо слагаемых есть ещё и перенос из предыдущего, равный 0 или 1, а на выходе помимо суммы также равный 0 или 1 перенос в следующий разряд.

```
  1              1
 115            101
 116           +001
 ---           ----
 231            110
```

Рис. 16. Сложение 3-разрядных чисел «в столбик» с переносом (перенос выделен красным) в десятичной (слева) и двоичной (справа) системах исчисления

Очевидно, что для двоичной системы (речь об одном разряде) не только перенос, но и слагаемые, и сумма представлены одним битом (одной двоичной цифрой). Поскольку мы вольны интерпретировать значение 0/1 и как целое число и как ложь/истину, то можем рассматривать логические схемы как арифметические функции, которые оперируют числами в двоичной записи (то есть представленными при помощи цифр 0 и 1). При одноразрядном сложении мы руководствуемся следующими правилами:

- s содержит результат суммы двух слагаемых a и b и переноса из предыдущего разряда c по модулю 2: $s = a \oplus b \oplus c$;
- c' содержит значение переноса; перенос возникает при сложении двух одноразрядных двоичных чисел, когда их сумма плюс значение переноса из предыдущего разряда превышает 1: $c' = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$.

Итак, мы имеем функцию $f(a, b, c) = (s, c')$. Таблица истинности для этой функции представлен в табл. 2. Очевидно, что так заданный одноразрядный сумматор является одним шагом для сложения многоразрядных двоичных чисел.

Табл. 2. Таблица истинности для одноразрядного сумматора

a	b	c	s	c'
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Реализация одноразрядного сумматора с помощью вентилей, то есть логическая схема сумматора, представлена на рис. 5. Убедимся, что эта схема действительно делает то, что нужно, то есть соответствует табл. 2.

- Разряд суммы s. Входы a и b схемы поступают на вход вентиля (1) «исключающее или», а его выход и вход c, в свою очередь, — на вход вентиля (2) «исключающее или». Выход же вентиля (2) является одновременно и выходом схемы c. Таким образом, получаем, что $s = (a \oplus b) \oplus c = a \oplus b \oplus c$, то есть на выходе s мы действительно имеем разряд суммы.
- Перенос в следующий разряд $c' = (a \wedge b) \vee \big(c \wedge (a \oplus b)\big)$. Эта формула отличается от ранее предложенной для c', но легко убедиться, что она даёт тот же результат, причём с использованием меньшего количества вентилей. Действительно, $c \wedge (a \oplus b) = c \wedge (a \vee b) \wedge \neg(a \wedge b) = \neg(a \wedge b) \wedge \big((b \wedge c) \vee (c \wedge a)\big)$, следовательно, $(a \wedge b) \vee \big(c \wedge (a \oplus b)\big) = (a \wedge b) \vee \Big(\neg(a \wedge b) \wedge \big((b \wedge c) \vee (c \wedge a)\big)\Big) = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$, то есть мы получаем именно то, что нам и требовалось.

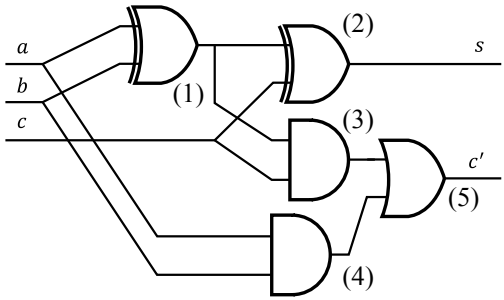


Рис. 17. Логическая схема одноразрядного сумматора

Четырехразрядный двоичный сумматор

Теперь, когда мы умеем складывать одноразрядные числа, перейдём к сложению двоичных многоразрядных чисел. Без потери общности рассмотрим сложение 4-разрядных чисел.

Снова вспомним сложение «в столбик» многоразрядных чисел. Обратим внимание, что оно «сконструировано» из множества одноразрядных сложений с переносом. Если принять во внимание, что система исчисления у нас двоичная и логическая схема сложения одноразрядных чисел у нас уже есть, то можно сконструировать из набора одноразрядных сумматоров многоразрядный. Для удобства обозначим уже построенный нами одноразрядный сумматор так, как показано на рис. 18.

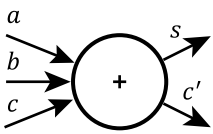


Рис. 18. Условное обозначение одноразрядного сумматора

Четырехразрядный сумматор, складывающий числа, состоящие из цифр $a_0a_1a_2a_3$ и $b_0b_1b_2b_3$ соответственно (младшие разряды мы записали слева), можно представить как функцию следующего вида:

$$f(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_{\to 0}) = (s_0, s_1, s_2, s_3, c_{3\to}).$$

Логическая схема для этой функции показана на рис. 19. Как и положено, каждый из четырёх одноразрядных сумматоров на этой схеме получает два бита исходных слагаемых и бит переноса из предыдущего разряда, а выдаёт бит суммы и бит переноса в следующий разряд.

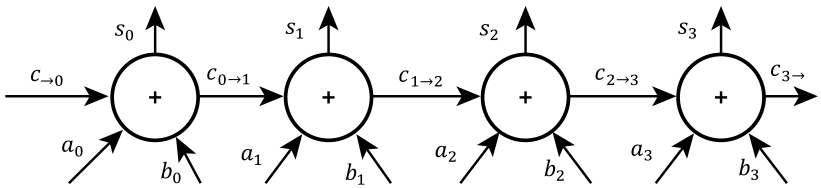


Рис. 19. Логическая схема 4-разрядного сумматора

Внимательные читатели могли заметить вход $c_{\to 0}$, на который подаётся 0, и выход $c_{3\to}$. Они оставлены на схеме не только для поддержания надлежащей степени общности. При помощи них можно объединять 4-разрядные сумматоры в более «длинные», подсоединяя выход переноса предыдущего ко входу переноса следующего так же, как мы соединяли одноразрядные сумматоры, получая 8-разрядные, 12-разрядные, 16-разрядные и так далее. Кроме того, поскольку в современных компьютерах эти вход и выход доступны для программ (при помощи так называемого *флага переноса*), то, пользуясь ими, можно программно реализовать длинную арифметику, то есть запрограммировать функции, которые на компьютере с 64-битной арифметикой смогут выполнять операции над целыми числами произвольной длины.

Оптимизация сумматора с целью ускорения переноса

Современные компьютеры оперируют, как правило, 32- или 64-битными целыми числами. При этом при сложении таких чисел узким местом является перенос: сигнал переноса должен быть *последовательно* обработан во всех разрядах двоичного представления числа начиная с младшего и заканчивая старшим. Последовательное выполнение переноса является причиной существенного замедления работы сумматоров. Для ускорения переноса при проектировании современных сумматоров используется специальный приём распараллеливания, который мы рассмотрим на примере 64-битного сумматора.

64-битный сумматор реализуется как два 32-битных — один для младшей 32-битной секции числа, а другой — для старшей. Младшая секция, представляющая собой обычный 32-битный сумматор, складывает биты $a_0, a_1, \dots, a_{31} + b_0, b_1, \dots, b_{31}$ и выдаёт биты суммы $s_0, s_1 \dots, s_{31}$, а также бит переноса $c_{31\to 32}$. Старшая секция складывает с учётом переноса $c_{31\to 32}$ старшие биты $a_{32}, a_{33}, \dots, a_{63} + b_{32}, b_{33}, \dots, b_{63}$. Но чтобы не ждать переноса из младшей секции, старшая организована следующим образом: она состоит из двух 32-битных сумматоров, получающих на вход одни и те же слагаемые $a_{32}, a_{33}, \dots, a_{63}$ и $b_{32}, b_{33}, \dots, b_{63}$, и при этом один из сумматоров всегда получает бит переноса $c_{31\to 32} = 0$, а другой — $c_{31\to 32} = 1$. Сумматоры старшей

секции «запускаются» одновременно друг с другом и с сумматором младшей и одновременно с ней заканчивают работу. К моменту окончания работы этих трёх 32-битных сумматоров (одного «младшего» и двух «старших») становится известно действительное значение переноса $c_{31\to32}$, и в итоговую сумму попадают результирующие биты лишь одного из «старших» сумматоров — того, который работал в условиях верного предположения значения входящего переноса $c_{31\to32}$.

Описанный подход можно описать словами «Сделай всё, что можешь, как можно скорее, часть из этого пригодится, остальное выбросим». Такая стратегия вычислений называется *спекулятивной*. Спекулятивность усложняет сумматор в нашем примере: она приводит к тому, что фактически результаты 1/3 вычислений (одного из трёх 32-разрядных сумматоров) выполняются впустую. Тем не менее она популярна, так как позволяет ускорить работу сумматора почти вдвое. Иногда сумматоры делят и на бóльшее число секций.

Арифметико-логическое устройство

> *Сумматор, мультипликатор (логическая схема для умножения двоичных чисел), а также модули для выполнения логических операций и некоторые другие логические схемы составляют* ***арифметико-логическое устройство (АЛУ)****, которое является одним из основных блоков процессора.*

О важности арифметических и логических операций мы уже говорили в прошлой лекции. Арифметико-логическое устройство — это функциональный блок процессора, который, по сравнению с обычным сумматором, можно считать своеобразным «швейцарским ножом»: «снаружи» он во многом напоминает обычный сумматор, но функциональность у него гораздо шире. С остальными блоками процессора его связывают входы и выходы, описанные ниже.

1. Как и у обычного сумматора, у АЛУ имеется два входа для первого и второго аргументов бинарных операций (например, сложения или умножения); для унарных операций (изменение знака, логическое отрицание) используется один из них.
2. Как и у обычного сумматора, у АЛУ имеется выход для результата операции.
3. Как и у обычного сумматора, у АЛУ имеется вход и выход переноса.
4. АЛУ также имеет управляющий вход, задающий, какую именно следующую операцию следует выполнить, то есть какой из инструментов швейцарского ножа нужно сейчас использовать.

В зависимости от значения на управляющем входе (4) ко входам и выходам (1–3) подключается та или иная логическая схема, входящая в состав АЛУ:

- сумматор;
- побитовые логические операции;
- побитовый сдвиг;
- мультипликатор и пр.

Сделаем небольшой обзор реализаций упомянутых арифметико-логических операций.

Сложение выполняется при помощи сумматора. Вычитание выполняется также сумматором, поскольку очевидным образом выражается через сложение, что мы обсуждали в предыдущих лекциях.

Побитовые логические операции — это операции, которые выполняются побитово, например, побитовое отрицание двоичного числа выполняет отрицание каждого его бита, побитовая операция «и» двух чисел выполняет «и» для каждой пары соответствующих битов и т. д. Таким образом, для побитового отрицания, получая на вход 32-битовое двоичное число $a_0, a_1, \dots, a_{31}$, выдает следующий результат: $\neg a_0, \neg a_1, \dots, \neg a_{31}$. Побитовое «и» (операция & в языке C), получая на вход два 32-битовых числа $a_0, a_1, \dots, a_{31}$ и $b_0, b_1, \dots, b_{31}$, выдаёт число $a_0 \wedge b_0, a_1 \wedge b_1, \dots, a_{31} \wedge b_{31}$. Аналогично действуют и другие побитовые операции — «или», «исключающее или» и т. д. Поскольку, в отличие от сумматора, все биты обрабатываются независимо, проблем с задержкой переноса не возникает и побитовые операции всегда выполняются за один такт.

Сдвиг двоичного числа — это операция, смещающая разряды (биты) числа от младшего к старшему, дописывая справа нуль и теряя старший бит (левый сдвиг, обозначается “<<” в языке C) или от старшего к младшему, дописывая слева нуль и теряя младший бит (правый сдвиг, “>>” в языке C). Для того чтобы сдвинуть аргумент a на 1 бит влево, АЛУ присоединяет выходные биты к соответствующим входным. Сдвиг числа влево на 1 бит эквивалентен умножению на 2, но выполняется намного быстрее. Аналогично работает сдвиг вправо, и он эквивалентен делению на 2 нацело. Поскольку при сдвигах АЛУ всего лишь соединяет выходы с соответствующими входами, сдвиги, так же как и побитовые операции, выполняются за один такт. Отметим, что современные АЛУ за один такт умеют сдвигать числа и на произвольное количество битов, а также выполнять некоторые другие разновидности сдвигающих биты операций.

Умножение выполняется мультипликатором. Если мы вспомним алгоритм умножения «в столбик», то увидим, что он состоит из сдвигов, сложений и умножений на одноразрядное число. В случае двоичной системы умножение на одноразрядное число реализуется тривиально: поскольку одноразрядное число может быть либо 0, либо 1, то и произведение будет равно или другому множителю, или нулю. Таким образом, мультипликатор также можно изготовить в виде логической схемы. Но она получится весьма

громоздкой: в ней будет уже очень много вентилей и связей между ними. Поэтому в некоторых АЛУ операция умножения выполняется не единой логической схемой, а как последовательность *микроопераций* (про микрооперации мы расскажем в следующих лекциях) с запоминанием промежуточных результатов в специальных внутренних регистрах, используемых только в АЛУ. Чтобы лучше представить себе работу такого АЛУ, попробуйте написать программу на любом языке программирования, которая умножает два числа, пользуясь для этого лишь операциями сдвига и сложения. По аналогии с умножением деление также можно реализовать при помощи логической схемы, но она получится ещё сложнее, поэтому деление также часто реализуется в виде последовательности микроопераций.

Отметим, что программная реализация умножения, которую мы предлагаем выполнить читателям, — отнюдь не просто «учебное» упражнение. АЛУ некоторых простых процессоров, например Intel 8080, позволяло выполнять лишь сложение, вычитание, сдвиг и побитовые операции, а умножение для этих процессоров действительно приходилось реализовывать уже программистам.

Вопросы

1. Что такое арифметико-логическое устройство?
2. Почему велика его роль в устройстве процессора?
3. Что такое одноразрядный сумматор?
4. Постройте таблицу истинности одноразрядного сумматора.
5. Постройте логическую схему одноразрядного сумматора.
6. Проверьте правильность работы одноразрядного сумматора при различных входных значениях *a, b* и *c*.
7. Постройте на основе нескольких 1-разрядных сумматоров один 4-разрядный сумматор.
8. Объясните, что такое многоразрядный сумматор.
9. Для чего используются сумматоры в ЭВМ?
10. Назовите узкое место многоразрядных сумматоров.
11. Опишите способ оптимизации переноса в многоразрядном сумматоре.
12. Что такое спекулятивные вычисления?
13. Что такое АЛУ, и какие операции оно может выполнять?
14. Перечислите входы АЛУ.
15. Перечислите выходы АЛУ.
16. Что такое побитовые логические операции?
17. Расскажите про реализацию побитовых логических операций.
17. Что такое побитовый сдвиг?
19. Расскажите про реализацию побитового сдвига.
20. Расскажите про реализацию в АЛУ умножения.

Литература

1. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
2. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.
3. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.

Лекция 8. Современные процессоры

Основные характеристики процессора: тактовая частота, разрядность, машинный язык и система команд; многоядерные процессоры; краткий обзор современных семейств процессоров — Intel x86, ARM, MIPS, AVR; отечественные процессоры «Эльбрус» и «Байкал».

Как мы уже говорили выше, компьютер состоит из большого количества электронных, а также некоторого количества механических устройств. Большинство из них предназначены либо для хранения и передачи данных (оперативная память, жёсткий диск, сетевой адаптер), либо для выполнения служебных функций (электрическое питание, охлаждение). Главной задачей компьютера является исполнение программ, и её выполняет *центральный процессор*. Имеющиеся в современных компьютерах другие устройства, которые могут быть достаточно интеллектуальными (например, веб-камера или звуковой адаптер), также могут обладать собственными процессорами и самостоятельно преобразовывать данные, однако функциональность этих устройств ограничена и они выполняют лишь узкий набор специальных задач.

> ***Центральный процессор (Central Processing Unit, CPU)*** *является основным модулем компьютера и предназначается для выполнения программ.*

Основными характеристиками процессора являются:

- тактовая частота;
- разрядность;
- машинный язык и система команд.

Тактовая частота процессора

Важно, чтобы процессор и другие узлы компьютера выполняли свои действия синхронно, поскольку иначе их деятельность окажется несогласованной. Так, например, различные узлы компьютера должны «знать», когда именно в регистрах процессора или на выходе логических схем будут готовы корректные данные. Если начать считывать эти данные раньше положенного сорока (например, считывать данные на выходе описанного в предыдущем разделе сумматора, не дождавшись полного прохождения переноса), то они окажутся некорректными и в работе ЭВМ произойдёт сбой.

Обычных часов для измерений времени с этой целью недостаточно, поскольку требуется очень высокая точность, в частности, в связи с тем, что измерять приходится небольшие промежутки времени — доли наносекунд и даже меньше. Наконец, в данном случае не требуется абсолютное время, достаточно лишь относительного — то есть требуется отвечать на вопросы типа «сколько прошло времени с момента возникновения того или иного события?», «сколько времени нужно ждать?» и т. д.

Учет времени в компьютере осуществляется специальным способом — с помощью подсчета количества *тактовых импульсов*, которые были сгенерированы в период от одного события до другого. Тактовые импульсы генерируются через равные промежутки времени *тактовым генератором* — специальной электронной схемой, встроенной в центральный процессор компьютера или его материнскую плату. Таким образом, поток тактовых импульсов образует поток времени внутри компьютера.

> *Количество тактовых импульсов в единицу времени, генерируемых тактовым генератором, называется* ***тактовой частотой*** *процессора. Тактовая частота измеряется в герцах (Hz).1 Hz (1 Гц) соответствует одному событию в секунду — таким образом, 1 Hz =1* c^{-1}.

Тактовая частота является одной из важных характеристик процессора и компьютера в целом: чем она выше, тем больше операций процессор выполняет за единицу времени, тем выше его производительность. В силу важности этой характеристики часто говорят о тактовой частоте всего компьютера, имея в виду тактовую частоту его процессора.

Невозможно достичь значительного увеличения производительности компьютера путём простого увеличения тактовой частоты: для того чтобы электронные схемы работали на высоких частотах, они должны быть соответствующим образом сконструированы. Однако до некоторой степени повысить тактовую частоту готового компьютера возможно — это называют *разгоном (Overclocking)* процессора. Разгон позволяет повысить производительность компьютера, но он требует специальных знаний и опыта для аккуратного подбора допускающих разгон компонент, подстройки параметров электропитания, организации дополнительного охлаждения и т. д. Подобная практика достаточно распространена, но производителями электроники она не рекомендуется, поскольку повышает риск выхода компьютера из строя.

Процессор является самым высопроизводительным модулем ЭВМ, поэтому тактовая частота, используемая для синхронизации различных блоков внутри процессора, является слишком высокой для других модулей компьютера. В частности, эта тактовая частота оказывается высокой для синхронизации взаимодействия процессора с другими модулями. Поэтому в современных ЭВМ, как правило, имеется две тактовых частоты — внутренняя и внешняя.

Внутренняя тактовая частота ЭВМ *используется внутри процессора для синхронизации его блоков и составляет на сегодняшний день 1 GHz и больше.*

Внешняя тактовая частота ЭВМ *используется для синхронизации процессора и прочих устройств компьютера, с которыми он взаимодействует, и составляет на сегодняшний день сотни MHz.*

Для ЭВМ первого поколения внутренняя тактовая частота была ограничена значением100 KHz, и это означает, что они могли совершать до 100 000 операций в секунду. Оперативная память этих компьютеров строилась на базе электромеханических реле, и её быстродействие составляло всего несколько Hz, поэтому и внешняя тактовая частота ЭВМ первого поколения составляла те же самые несколько Hz. Для последующих ЭВМ, обладавших электронной оперативной памятью, внешняя и внутренняя тактовые частоты отличались уже не так значительно. Типичным было отличие в несколько раз, в отдельных случаях частоты могли совпадать. Для ЭВМ второго поколения верхняя граница внутренней тактовой частоты увеличилась до 1 MHz, что соответствовало миллиону операций в секунду, для третьего поколения — 10 MHz и, соответственно, 10 миллионов операций в секунду. Тактовая частота современных процессоров, используемых в ЭВМ четвертого поколения, составляет до 5 GHz. Значения тактовой частоты для ряда известных процессоров приведены в таблице 3.

Табл. 3. Внутренняя тактовая частота ряда известных процессоров

Название процессора	Год выпуска	Тактовая частота
Intel 4004	1971	740 KHz
Motorola 6800	1974	2 MHz
Intel 80186	1982	6 MHz
Intel 80486 DX	1989	20 MHz
Intel 80486 DX4	1994	100 MHz
Pentium 4	2000	1,6 GHz
Intel Xeon Westmere	2010	3,6 GHz
IBM zEC12	2012	5,5 GHz

Отметим, что произвольно поднять тактовую частоту не позволяет ограничение на передачу импульсов скоростью света и невозможностью сделать размеры транзисторов и прочих полупроводниковых элементов меньше, чем размер атома.

Помимо этих ограничений неограниченно увеличивать тактовую частоту не позволяют свойства используемых полупроводников. Кремниевые диоды и транзисторы управляют прохождением электрических сигналов, меняя свою электрическую проводимость. Они делают это очень быстро, но в масштабах современной электроники отнюдь не мгновенно: изменить своё состояние они могут «всего лишь» несколько миллиардов раз в секунду. В последние годы физиками ведутся исследования, посвященные возможности замены традиционных кремниевых транзисторов на транзисторы с применением графена, ожидаемая скорость «срабатывания» которых будет в сотни раз выше. Но пока можно говорить лишь об интересных перспективах ускорения электронных схем, до внедрения новых технологий в индустрию ещё далеко.

Разрядность процессора

Другой важной характеристикой процессора является его разрядность, которая определяет следующие важнейшие характеристики компьютера.

1. Количество бит, которые процессор может обработать за одну операцию над целыми числами. Это могут быть как арифметические операции (например, сложение, вычитание, умножение), так и побитовые логические (например, «и», «или», «не»).
2. Размер арифметических регистров процессора, то есть регистров, предназначенных для хранения целых чисел. Регистры процессора, как наиболее быстрый вид памяти, интенсивно используются для хранения аргументов целочисленных операций, примеры которых мы приводили выше, и их результатов.
3. Количество бит в шине данных и, следовательно, количество бит, которое процессор может за одну операцию (за один такт) прочитать из оперативной памяти или записать в неё.
4. У большинства современных процессоров разрядность определяет максимальный объём оперативной памяти, напрямую адресуемой процессором. Эта величина зависит от размера адресных регистров, предназначенных для хранения адресов данных (в том числе и машинного кода) в оперативной памяти.
5. При программировании на языке C, как правило, разрядность задаёт размеры длинного целого числа (`long int`) и указателя (`void *`).

Процессор с разрядностью 64 бита и тактовой частотой 1 GHz может обработать за секунду столько же данных, сколько процессор с разрядностью 32 бита и тактовой частотой 2 GHz. Практически все современные микропроцессоры, выпускаемые компаниями Intel и AMD, являются 64-разрядными, но для совместимости с предыдущими версиями поддерживают 32-битный режим.

Совместимость процессоров означает что у новых, 64-разрядных процессоров появляется возможность исполнять старые программы, скомпилированные в 32-разрядном режиме для старых 32-разрядных процессоров.

Как и в случае с тактовой частотой, у процессора также есть *внутренняя* и *внешняя* разрядности. Внутренней разрядностью называют разрядность регистров процессора. Внешняя разрядность — это разрядность шин данных и адреса, к которым подключён процессор. У многих современных процессоров эти разрядности совпадают, но в общем случае это не всегда так. Так, например, у Intel 8086 и внутренняя, и внешняя разрядность — 16 бит, а у его более дешёвого аналога Intel 8088 внутренняя разрядность 16 бит, а внешняя — 8. Разделение внутренней и внешней разрядностей позволяет конструкторам ЭВМ использовать процессор, совместимый с Intel 8086 (то есть Intel 8088 исполняет те же программы), но при этом удешевить шину данных и оперативную память, снизив их разрядность (ценой снижения пропускной способности). Ещё одним примером является процессор Intel Pentium I: внутренняя разрядность у него, как и у предыдущих процессоров семейства, составляет 32 бита, а внешняя — 64. С одной стороны, это позволило сохранить обратную совместимость с 32-битными программами, с другой — ускорить обмен данными между оперативной памятью и кэш-памятью (данные в современных процессорах загружаются не напрямую из оперативной памяти в регистры, а вначале попадают в кэш-память, о которой мы расскажем ниже).

Машинный язык и система команд процессора

Современные высокоуровневые языки программирования позволяют представлять и алгоритмы, и данные таким образом, что программист может, работая с программным кодом, продолжать мыслить в терминах исходных задач. Для процессора, который в итоге исполняет написанные программы, такой уровень абстракции недоступен. Процессор может исполнить лишь программу на *машинном языке*.

> ***Машинный язык (Machine Language) процессора*** *является средством представления программы в памяти компьютера при помощи двоичного машинного кода, который «понимается» центральным процессором, то есть может быть им выполнен. Машинный язык состоит из (i) системы команд (Instruction Set) — набора операций, которые может выполнять процессор; (ii) видов данных (различные числа, адреса и пр.), для которых определены эти операции; (iii) способа кодирования команд и данных в двоичном коде.*

Система команд определяет операции, которые может выполнять процессор. Прежде всего, это часто используемые операции:

- сложение/умножение/деление в целочисленной арифметике;
- распространённые математические функции (экспонента, логарифм, тригонометрические функции и др.);
- загрузка/выгрузка данных в процессор;
- операции над массивами данных и т. д.

Операции ориентированы на определённые виды данных, с которыми процессор может работать. Например, это могут быть целые числа разной длины со знаком и без, вещественные числа разной точности, символьные данные, указатели. Наконец, кодирование определяет, каким образом эти операции и данные должны храниться в оперативной памяти, ведь там хранятся в итоге массивы байтов. Например, различные семейства процессоров могут по-разному упорядочивать в памяти байты у многобайтовых целых чисел — от младшего к старшему или от старшего к младшему[7].

Машинные языки разных семейств процессоров существенно отличаются, подобно тому как отличаются языки различных народов. Особенности естественных языков определяются средой обитания и историей их носителей, а машинных — назначением соответствующих компьютеров и преемственностью внутри семейств. Например, процессоры компьютеров семейства System/360 хоть и относятся к архитектуре фон Неймана, однако поддерживают помимо двоичной и десятичную арифметику, что важно для финансовых вычислений. Другой пример — постепенное (на протяжении более 40 лет) расширение поддержки вещественной арифметики и обработки массивов чисел в процессорах семейства Intel x86 по мере роста количества мультимедиа-приложений для персональных компьютеров.

Многоядерные процессоры

Такие фундаментальные ограничения, как размер атома и скорость света, не позволяют произвольно увеличивать тактовую частоту и разрядность процессора. Сейчас увеличение производительности компьютеров в основном осуществляется путём распараллеливания различных операций, выполняемых процессором.

> ***Распараллеливание*** *означает организацию параллельной работы различных блоков процессора, а также параллельное исполнение отдельных команд или групп команд программы.*

[7] Представление чисел, при котором вначале в памяти хранятся байты со старшими разрядами, а затем с младшими, называется Big-endian. 32-битное число $ABCD3210_{16}$ в оперативной памяти компьютера Big-endian (например, семейства System/360) будет представлено четырьмя байтами: AB_{16}, CD_{16}, 32_{16}, 10_{16}. В компьютере Little-endian (например, Intel x86 и большинство других современных семейств) байты будут следовать в обратном порядке: 10_{16}, 32_{16}, CD_{16}, AB_{16}. Термины Little- и Big-endian являются отсылкой к роману Дж. Свифта «Путешествия Гулливера» и были введены в 80-е годы XX века.

Применяются различные техники повышения производительности процессоров при помощи распараллеливания. Одной из известных популярных сейчас техник является создание многоядерных процессоров.

> ***Многоядерный процессор (Multicore CPU)*** *— это процессор, который позволяет одновременно (параллельно) исполнять несколько программ, используя при этом несколько ядер.*

> ***Ядро (CPU Core)*** *является частью процессора, которая может самостоятельно декодировать прочитанные из памяти команды программы и последовательно их выполнять.*

Многоядерный процессор не столько быстрее исполняет обычные (однопоточные) программы, сколько позволяет быстро исполнить несколько программ одновременно. Однако многопоточные программы многоядерный процессор исполняет существенно быстрее, чем одноядерный, предоставляя различным потокам параллельно работающие ядра. Поэтому в тех случаях, когда необходимо повысить производительность однопоточных программ, нужно использовать параллельные алгоритмы и переписывать эти программы в многопоточном стиле.

Популярность многоядерных процессоров обусловлена их выгодными экономическими характеристиками. При производстве один многоядерный процессор стоит дешевле нескольких одноядерных процессоров с таким же суммарным количеством ядер. Причина в том, что ядро процессора — это ещё не весь процессор. Кроме ядер в процессоре имеется ещё набор блоков, осуществляющих обращение к шинам, обрабатывающих различные сигналы, кэширующих данные и т. д. Подобные блоки реализованы в многоядерном процессоре в единственном экземпляре, тогда как ядер, одновременно исполняющих разные последовательности команд, в одном процессоре может быть до нескольких десятков. Отметим, что существенно проще и дешевле выпускать однопроцессорный компьютер, чем многопроцессорный. В конечном итоге реальная потребность увеличения производительности компьютеров ценой разумного удорожания определила нынешнюю популярность именно многоядерных процессоров.

На сегодняшний день почти все процессоры, в том числе мобильные, имеют несколько вычислительных ядер. Например, у процессоров Intel Core i7, распространённых в современных персональных компьютерах, может быть 2, 4, 6 или 8 ядер, а процессор A13 современных смартфонов Apple имеет 6 ядер. Ядра мобильных процессоров могут иметь разные характеристики — например, выделяются «медленные» ядра и «быстрые» ядра. «Медленные» ядра имеют экономичное энергопотребление, но относительно невысокую производительность и процессор использует их постоянно. «Быстрые» ядра обладают высокой производительностью, но потребляют много энергии и используются процессором лишь при необходимости. Например,

упомянутый выше мобильный процессор A13 имеет 6 ядер, из которых 4 являются «медленными» и 2 — «быстрыми».

Краткий обзор некоторых современных семейств процессоров

Рассмотрим следующие современные семейства процессоров:

- Intel x86,
- ARM,
- MIPS,
- AVR,
- «Эльбрус»,
- «Байкал».

Этот набор позволяет получить представление о различных современных процессорах, предназначенных для настольных компьютеров и мобильных устройств, для встраиваемых и телекоммуникационных систем, а также для суперкомпьютеров.

Семейство процессоров Intel x86. Процессоры данного семейства на протяжении сорока лет являются основой для комплектации компьютеров IBM PC и последующих совместимых с ними компьютеров. Данные компьютеры составляют на настоящий момент большинство персональных компьютеров на планете. Также благодаря достаточной вычислительной мощности и богатым функциональным возможностям процессоры этого семейства часто используются при построении многопроцессорных суперкомпьютеров, используемых для высокопроизводительных научных вычислений. Основным производителем этих процессоров традиционно является компания Intel, но помимо этой компании совместимые процессоры выпускаются также такими компаниями, как AMD, Cyrix, VIA Technologies. Поэтому, строго говоря, более корректно обозначать это семейство как x86, потому что в этом случае мы будем говорить о процессорах разных производителей, не только компании Intel. Но в рамках нашего курса нам такая степень общности не требуется.

Семейство процессоров ARM. На рынке мобильных устройств в последнее десятилетие доминируют процессоры семейства ARM. Разработавшая их архитектуру в 1990-х годах компания Arm Holdings сама процессоров не производит, но лицензирует их производство другими компаниями — например, Qualcomm, Mediatek, NVIDIA. Архитектура ARM показала свою пригодность не только для мобильных устройств. В последние годы на базе процессоров ARM также выпускаются ноутбуки (например, Chromebook), настольные компьютеры (например, современные компьютеры Apple на основе ARM-совместимого процессора Apple M1), одноплатные компьютеры Raspberry Pi. Наконец, в Японии создан суперкомпьютер Fugaku, построенный на базе более

чем 150 000 процессоров Fujitsu A64FX архитектуры ARM. Этот суперкомпьютер создан в Центре вычислительных наук Института физико-химических исследований в городе Коба, Япония. Он стал победителем ряда рейтингов суперкомпьютерных вычислительных систем в июне 2020 года. Официально Fugaku был введён в эксплуатацию в марте 2021 года.

Семейство процессоров MIPS было разработано в середине 1980-х годов в Стэнфордском университете. Сейчас их производство лицензирует компания MIPS Technologies, Inc. Как и в случае с ARM, компания, владеющая правами на архитектуру процессоров, сама не занимается их производством. Среди производителей процессоров архитектуры MIPS можно перечислить такие известные компании, как Siemens, Philips, Toshiba, NEC. Сейчас процессоры MIPS используются преимущественно во *встраиваемых компьютерах*, управляющих, например, станками и лифтами, а также для оборудования вычислительных сетей — в роутерах, шлюзах и т. д. Сейчас многие производители вычислительных устройств создают свои процессоры на базе архитектуры MIPS, добавляя в них специализированные функции, например, в случае телекоммуникации — функции для обработки сигналов.

Семейство процессоров AVR. Для разработки простых устройств популярны *однокристальные компьютеры* семейства AVR компании Atmel. Однокристальными они называются потому, что в одной интегральной схеме (то есть на одном кристалле) содержатся все устройства, необходимые для функционирования компьютера. Помимо собственно процессора интегральная схема включает в себя несколько килобайт оперативной памяти и флэш-памяти для хранения программы, тактовый генератор, аналогово-цифровые и цифрово-аналоговые преобразователи для непосредственного соединения с датчиками и схемами управления различными устройствами. *Аналогово-цифровой преобразователь* (АЦП) — это электронная схема, преобразующая значение физической величины (как правило, напряжения) в цифровой вид, то есть в машинное число. Цифрово-аналоговый преобразователь (ЦАП) — электронная схема, выполняющая обратное преобразование. Например, аудиосигнал, выдаваемый звуковым адаптером, — аналоговый, и генерирует его цифрово-аналоговый преобразователь, получающий на вход оцифрованный звуковой сигнал.

> *Однокристальные вычислительные устройства, сочетающие функции процессора и периферийных устройств, называют* ***микроконтроллерами****.*

Этот термин, как и в случае с умещающимися на одном кристалле микропроцессорами, подчёркивает их «самодостаточность»: фактически для работы микроконтроллеру требуется лишь электропитание. Микроконтроллеры используются для управления материнскими платами, жесткими дисками, для управления устройствами бытовой техники (микроволновые печи, холодильники, стиральные машины и пр.). Кроме компании Atmel

микроконтроллеры выпускаются многими другими компаниями — например, Siemens, Microchip, Fujitsu.

Возвращаясь к микроконтроллерам AVR, отметим, что они очень экономичны в потреблении электроэнергии: устройство, состоящее из микроконтроллера AVR и нескольких датчиков, может несколько месяцев работать, питаясь от трёх «пальчиковых» батареек. Существуют микроконтроллеры с ещё меньшим потреблением электроэнергии, которые могут работать несколько лет от батарейки для наручных часов.

Простота, дешевизна, «самодостаточность» и низкое энергопотребление сделали микроконтроллеры AVR популярными среди начинающих инженеров и радиолюбителей. В радио- и робототехнических кружках они часто применяются в образовательных целях.

Наконец, отметим, что микроконтроллеры AVR имеют гарвардскую архитектуру. В случае с однокристальными компьютерами, у которых внутри процессора находится память как для машинного кода, так и для данных (то есть шин нет!), гарвардская архитектура позволяет упростить устройство.

Семейство процессоров «Эльбрус». Современные отечественные процессоры «Эльбрус» (компания-разработчик — АО «МЦСТ») нельзя отнести к тому же семейству, что и процессоры одноименных советских суперкомпьютеров, которые мы рассматривали в первых лекциях. Современные процессоры «Эльбрус» разрабатываются с начала 2000-х годов, являются многоядерными, имеют оригинальную архитектуру и систему команд. По характеристикам процессоры «Эльбрус» сопоставимы с современными процессорами семейства Intel x86. Они позиционируются как процессоры для персональных компьютеров и серверов и предназначены для военных и государственных учреждений в тех случаях, когда использование иностранных процессоров недопустимо по соображениям безопасности. Для компьютеров на базе процессоров «Эльбрус» используется несколько отечественных операционных систем, основанных на ОС Linux. Из-за очень небольшого тиража рыночная цена этих процессоров существенно превосходит цену аналогичных по характеристикам процессоров Intel x86. В случае со специальными применениями этот факт не является критичным, но это закрывает для процессоров «Эльбрус» рыночный вариант использования.

Семейство процессоров «Байкал». Отечественные процессоры «Байкал» (компания-разработчик — АО «Т-Платформы») серийно выпускаются начиная с 2016 года. Эти процессоры входят в уже описанное выше более крупное семейство процессоров MIPS. Процессоры «Байкал» имеют производительность несколько ниже, чем у современных процессоров Intel x86, но вполне приемлемую для создания на их основе серверов и офисных настольных компьютеров. Их важной особенностью является дешевизна — они несколько дешевле, чем процессоры Intel x86. На данный момент офисные компьютеры на основе процессоров «Байкал» массово поставляются в МВД РФ. Также, в отличие от компьютеров на основе процессоров «Эльбрус», их низкая цена позволяет надеяться на хорошие рыночные перспективы.

Вопросы

1. Дайте определение процессору.
2. Что такое тактовый импульс?
3. Что такое внутренняя и внешняя тактовые частоты процессора?
4. Какие типичные значения тактовой частоты характерны для ЭВМ разных поколений?
5. Какие факторы не позволяют неограниченно увеличивать тактовую частоту процессоров?
6. Что такое разрядность процессора?
7. Что такое совместимость процессоров?
8. Какие типичные значения имеет разрядность современных микропроцессоров?
9. Что такое внутренняя и внешняя разрядности процессора?
10. Что такое машинный язык процессора?
11. Что такое система команд процессора?
12. Как машинный язык связан с языками программирования высокого уровня?
13. Что такое ядро процессора?
14. В чём преимущество многоядерных процессоров по сравнению с одноядерными?
15. Расскажите про «быстрые» и «медленные» ядра многоядерных процессоров.
16. Расскажите о семействе процессоров Intel x86.
17. Расскажите о семействе процессоров ARM.
18. Расскажите о семействе процессоров MIPS.
19. Расскажите о семействе процессоров AVR.
20. Расскажите о семействе процессоров «Эльбрус».
21. Расскажите о семействе процессоров «Байкал».

Литература

1. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
2. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.
3. *Хорошевский В. Г.* Архитектура вычислительных систем: учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.
4. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.

Лекция 9. CISC- и RISC-архитектуры процессоров

CISC- и RISC-архитектура: определение, особенности машинного языка, преимущества и ограничения; пример машинного кода программы на C для процессора Intel x86 (CISC-архитектура) и процессора ARM (RISC-архитектура).

Как мы убедились выше, различные архитектуры процессоров могут заметно отличаться, но задача повышения быстродействия разумной ценой актуальна для любых архитектур. В рамках этой лекции мы рассмотрим две распространённые архитектуры процессоров — CISC- и RISC.

Определения, особенности, преимущества и недостатки

Традиционный подход к проектированию процессоров ставил одной из главных задач эффективную аппаратную реализацию часто встречающихся операций и удобство написания программ на языке Ассемблер и машинном языке. Этот подход называется *CISC-архитектурой* (Complicated Instruction Set Computer); данный термин переводится как «компьютер со сложной системой команд».

> ***CISC-архитектура*** *характеризуется большим количеством разнообразных команд, способных выполнять сложные, многошаговые действия.*

При использовании данного подхода система команд и машинный язык обладают свойствами, перечисленными ниже.

- *Выразительность системы команд.* Используется большое количество разнообразных команд — от простейших арифметических до сложных, совмещающих, например, арифметические операции и различные проверки операндов (например, на равенство и неравенство или на состояние отдельных битов в них). Системы команд некоторых процессоров поддерживают также работу с массивами данных — копирование, поэлементное сравнение и т. д.

- *Гибкость при задании аргументов команд.* Одна операция, например сложение, может быть представлена в CISC-процессоре целым семейством команд. Все эти команды позволяют различными способами задавать аргументы сложению, которые, таким образом, могут браться из машинного кода, а также из оперативной памяти — или по явно заданному адресу, или по адресу, заданному в регистре. Подобные возможности позволяют минимизировать вспомогательное копирование данных (например, из памяти в регистр) и помогают эффективно кодировать с помощью небольшого числа команд, а часто вообще с помощью одной единственной команды, такие распространённые операции, как, скажем, сравнение двух переменных или сложение переменной с элементом массива.
- *Экономное кодирование* — возможность создавать машинный код, занимающий минимальный объём памяти. Двоичное представление разных команд в оперативной памяти имеет разную длину и разный формат: наиболее частые команды умещаются в один байт, а на редко используемых не экономят, кодируя их пятью или даже более байтами. Это позволяет в среднем существенно сократить размер машинного кода.
- *Существенно разная длительность выполнения команд.* Простые команды, такие как обнуление регистра, выполняются в CISC-процессорах быстро, в то время как сложные команды, такие как деление, требуют значительного времени.

В качестве примеров процессоров, имеющих CISC-архитектуру, можно привести процессоры знаменитых менфреймов IBM System/360, а также процессоры семейства Intel x86.

Теперь перечислим недостатки CISC-архитектуры.

- Разработка процессоров с выразительной, но обширной и сложной системой команд является трудозатратной задачей, что повышает стоимость разработки процессоров и риск большого количества ошибок.
- Выпускаемые процессоры, будучи сложными, имеют высокие себестоимость и рыночную цену.
- Сложная система команд затрудняет внутрипроцессорные оптимизации, например, автоматическое распараллеливание выполнения машинных команд.

Многие CISC-процессоры реализуют сложные команды с помощью *микрокода* — специализированного машинного языка, позволяющего задавать последовательность действий (микроопераций), выполняемую одной машинной командой процессора. Таким образом, команды CISC-процессоров оказываются настолько сложными, что их приходится описывать при

помощи дополнительного машинного языка, который исполняется специализированным процессором внутри процессора.

RISC-архитектура (Reduced Instruction Set Computer) была нацелена на решение этих проблем.

> ***RISC-архитектура** процессора подразумевает небольшую систему команд, каждая из которых имеет эффективную аппаратную реализацию, а сложные операции, необходимые для прикладных программ, составляются из множества этих простых команд.*

Вот основные свойства процессоров с RISC-архитектурой.

- *Минималистичность системы команд.* Машинные команды RISC-процессора имеют простую функциональность, соответственно, время на их декодирование и исполнение оказывается небольшим, а блоки процессора, отвечающие за декодирование, — простыми.
- *Отсутствие гибкости в задании аргументов.* Обработку и пересылку данных из оперативной памяти в регистры процессора выполняют разные команды, то есть нет арифметических и логических операций, чьи операнды находятся в оперативной памяти и поддерживают различные варианты адресации. Соответственно, арифметические и логические операции реализуются командами, которые работают только с регистрами процессора. Например, нельзя с помощью одной команды сложить два числа, одно из которых находится в регистре, а другое — в памяти, а нужно отдельной командой загрузить значение из памяти в некоторый регистр и только после этого запустить команду сложения. Этот принцип в литературе часто называется *load-store architecture*, что как раз и означает обмен данными процессора с памятью при помощи отдельных команд. Чтобы компенсировать это ограничение, RISC-процессоры обычно имеют большое количество регистров общего назначения. Большая регистровая память позволяет постоянно держать много данных внутри процессора и тем самым минимизировать количество операций обмена данными с оперативной памятью.
- *Унифицированное кодирование.* Двоичное представление разных машинных команд (то есть соответствующий им машинный код) имеет одинаковый формат и фиксированный размер. Для большинства RISC-архитектур одна команда занимает одно машинное слово. Таким образом, имеется простой и «единообразный» машинный язык.
- *Фиксированное время выполнения команд.* Все команды выполняются за одно и то же количество тактов, что упрощает различные оптимизации — в частности, конвейеризацию, о которой мы подробнее расскажем ниже. Однако в RISC-процессорах имеется некоторое количество сложных команд, время исполнения которых может су-

щественно превышать стандартное, — например, умножение и деление. Эти операции сложнее, чем сложение и вычитание, — они требуют выполнения большого количества сдвигов, сложений/вычитаний и различных проверок. Поскольку такие команды выбиваются из стройной концепции RISC, для их реализации применяют специальные приёмы. В частности, команда деления исполняется асинхронно — само деление далее выполняется отдельным специальным блоком процессора, в то время как сам процессор исполняет другие команды. При этом исполняемая программа «знает», через сколько тактов деление завершится, и также знает, где (в каких регистрах) будут расположены частное и остаток от деления.

Ценой потери выразительности и экономности машинного языка процессоры с RISC-архитектурой достигают следующих преимуществ по сравнению с CISC-процессорами:

- упрощение и удешевление процессоров, как при конструировании, так и при производстве;
- обширные возможности для реализации различных оптимизаций, в частности, внутрипроцессорного распараллеливания.

Чтобы наглядно проиллюстрировать разницу между CISC- и RISC-архитектурами, рассмотрим следующий пример. Команда сложения (add) в процессорах семейства Intel x86 (CISC-архитектура) имеет два аргумента — первое и второе слагаемое, а результат исполнения команды помещается на место первого слагаемого. При этом любое из слагаемых может быть или регистром, или числом, хранящимся в оперативной памяти. Все возможные сочетания типов аргументов соответствуют различным командам add. Таким образом, в процессорах Intel x86 имеется три разных команды `add`: регистр/регистр, регистр/память, память/регистр. Команды `add` типа память/память у процессоров Intel x86 отсутствует, поскольку в рамках выполнения арифметических команд предполагается не более одного обращения к памяти. Это обусловлено преемственностью в рамках семейства Intel x86: в первых процессорах семейства (Intel 8086 и Intel 8088) несколько обращений к данным в оперативной памяти существенно замедлили бы реализацию арифметики и потребовали бы усложнения процессора. Следует отметить, что кроме вышеперечисленных команд сложения у процессоров Intel x86 имеется и другие команды `add`, которые предназначены для работы с константами, для использования разных вариантов адресации и пр.

В RISC-процессоре имеется одна команда `add`, и она работает только с регистрами, то есть имеет вид регистр/регистр. Если одно из слагаемых находится в памяти, то оно сначала в явном виде (то есть отдельной командой) должно быть загружено в соответствующий регистр, а уже потом должна быть вызвана команда сложения. Вместо этих двух команд в процессорах

Intel x86 просто вызывается соответствующая команда `add`, которая сначала загружает недостающий операнд из памяти в процессор, а потом выполняет сложение.

Данный простой пример показывает, что машинный язык RISC-процессора гораздо менее дружественный для программиста, программирующего на языке Ассемблер или в машинном коде, чем машинный язык CISC-процессора, позволяя обходиться меньшим количеством команд за счет гибкости задания аргументов. Но, как мы уже сказали выше, простой машинный язык позволяет упростить и *управляющее устройство* процессора — блок, ответственный за управление другими блоками процессора. В дополнение к уже упомянутому выше удешевлению процессора, это упрощение дает конструкторам RISC-процессоров ещё одну косвенную выгоду: возможность сделать управляющее устройство процессора более компактным и, как следствие, сократить время прохождения сигналов внутри него, что позволяет увеличить тактовую частоту процессора.

RISC-архитектура также имеет свои слабые стороны.

- Упомянутые выше оптимизации требуют бóльших, нежели в случае CISC-процессоров, усилий по созданию компилятором с языков высокого уровня оптимального машинного кода. Современные CISC-процессоры справляются с этим «самостоятельно», хотя и ценой значительного усложнения. Подробнее об этом мы расскажем ниже в разделе о конвейеризации и внеочередном исполнении машинных команд.
- Поскольку объем кода для RISC-процессора получается больше, чем для тех же программ, скомпилированных для CISC-архитектур, то увеличивается нагрузка на шины и на оперативную память ЭВМ.

Наиболее известными представителями RISC-архитектуры являются семейства ARM (процессоры этого семейства являются основой многих современных мобильных устройств, а также некоторых персональных компьютеров и даже суперкомпьютеров) и MIPS (наиболее популярны как основа для оборудования вычислительных сетей).

Одним из обстоятельств, повлиявших на популярность RISC-архитектуры (а это был рубеж 1970-х и 1980-х годов), стал тот факт, что значительная часть программ к этому времени разрабатывалась на языках высокого уровня, и, таким образом, с машинным языком уже имели дело компиляторы, а не программисты. Следовательно, необходимость удобного для человека машинного языка (одно из основных преимуществ CISC-архитектуры) стало неактуальным. В то же время RISC-процессоры хотя и потеснили CISC, но не вытеснили их окончательно. Большинство новых процессоров, разрабатываемых в последние годы, относятся к уже существующим семействам, а большинство этих семейств имеют RISC-архитектуру. Тем не менее для популярных CISC-семейств процессоров, например Intel x86 и IBM System/360

(и последующих), разработано такое количество ПО, что отказаться от этих процессоров в пользу других, пусть и более совершенных, уже невозможно по экономическим соображениям. Отметим также, что разработчики современных CISC-процессоров не отказываются от использования отдельных идей RISC. Так, в некоторых процессорах семейства Intel x86 (например, Intel 80486) машинные команды транслируются в последовательности микроопераций, которые уже исполняет «скрытый от программиста» внутренний RISC-процессор.

Пример

Чтобы наглядно проиллюстрировать особенности CISC- и RISC-архитектур, приведём пример. В качестве CISC-процессора рассмотрим процессор семейства Intel x86, а в качестве RISC-процессора возьмём ARM. Рассмотрим следующую несложную программу на C и покажем, какой машинный код создаётся компилятором для этой программы для CISC- и RISC-процессоров.

```
int a, b;
bool res;
a = a + b;
b = 3 * b;
if (a > b)
  res = true;
else
  res = false;
```

Для компиляции этой программы под платформу Intel x86 мы использовали компилятор Microsoft Visual C/C++ 19, для ARM — компилятор GNU C Compiler 8. Читатели для тех же целей могут воспользоваться сервисом https://godbolt.org/, который позволяет, не инсталлируя эти компиляторы на свой компьютер, транслировать программы на языках C/C++ под разные архитектуры, и наглядно показывает получившийся код на языке Ассемблер, а также в виде машинных команд.

Результаты трансляции для обеих архитектур представлены в табл. 4. Для каждого оператора программы на языке C даны команды машинной программы Intel x86 и ARM. Для каждой команды приводится её адрес в оперативной памяти, машинный код (и то, и другое — в шестнадцатеричной записи), а также ассемблерное представление этой команды (для большей наглядности).

Ниже представлены дополнительные пояснения и комментарии к табл. 4.

- Программа для процессора Intel x86 состоит из 11 команд, а для ARM — из 18.

- Арифметические регистры в Intel x86 именуются символами, например, `eax`, `ecx`, а в ARM они обычно нумеруются, например: `r0, r1,…` . Это связано с тем, что традиционно у RISC-процессоров арифметических регистров больше, чем у CISC, поскольку первым предпочтительно как можно больше операндов заранее разместить в регистрах с тем, чтобы во время исполнения основных команд не обращаться к оперативной памяти.
- Код команд процессора Intel x86 имеет разную длину — от одного до четырех байтов вместе с операндами, а код команд ARM занимает в памяти по четыре байта. В итоге программа для Intel x86 занимает 34 байта, а для ARM — 72. Эта закономерность сохраняется и для больших программ: машинный код для процессоров ARM обычно в 1,5–2,5 раза объёмнее, чем для Intel x86.
- Для Intel x86 компилятор использует команду умножения целых чисел со знаком (`imul`), а для ARM умножение заменяется сдвигом и сложением (`b * 3 → b * 2 + b → b << 1 + b`) — как уже обсуждалось выше, операция умножения является ресурсоёмкой, поэтому при компиляции программы под RISC-процессор везде, где имеется возможность избежать использования команды машинного умножения, компилятор делает это.
- Для Intel x86 в командах безусловного (`jmp`) и условного (`jle`) перехода параметр, указывающий, насколько следует «прыгнуть» относительно адреса текущей команды (так называемое *смещение*), задаётся в байтах (например, `jmp .+4` означает: «перепрыгни» через четыре байта машинного кода). У ARM все команды, представленные в виде машинного кода, имеют одинаковую длину — по четыре байта. Поэтому команды перехода (`b` и `ble`) оперируют не адресом, а сразу номером команды, соответственно и смещение задаётся не в байтах, а в количестве команд, причём не считая следующей (например, `jmp .+1` означает: «перепрыгни» через две команды).

Отметим также, что мы намеренно запретили использованным компиляторам оптимизировать программу. При включённых оптимизациях современные компиляторы выполняют много преобразований, которые в нашем случае существенно усложнили бы результирующий код, сделав его непригодным для нашей иллюстративной задачи. Читатели могут в качестве упражнения самостоятельно задать параметры оптимизации и посмотреть на то, что получится в итоге.

Табл. 4. Результаты компиляции программы на С в машинный код для процессоров Intel x86 и ARM

Операторы программы на С	Машинный код для CISC-процессора Intel x86			
	Адр.	Маш. код	Ассемблер	Комментарий
Int a, b; bool res;				
a = a + b;	00	8b 45 f4	mov eax, DWORD PTR [ebp-0xc]	Загрузка переменной a в регистр EAX (память→рег)
	03	03 45 f8	add eax, DWORD PTR [ebp-0x8]	Сложение регистра EAX с переменной b и сохранение суммы в регистре EAX (память+рег→рег)
	06	89 45 f4	mov DWORD PTR [ebp-0xc], eax	Выгрузка из регистра EAX переменной a (рег→память)
b = 3 * b;	09	6d 4d f8 03	imul ecx, DWORD PTR [ebp-0x8], 0x3	Загрузка переменной b в регистр ECX (память→рег) и умножение его значения на 3
	0d	89 4d f8	mov DWORD PTR [ebp-0x8], ecx	Выгрузка переменной b из регистра ECX (рег→память)
if (a>b)	10	8b 55 f4	mov edx, DWORD PTR [ebp-0xc]	Загрузка переменной a в регистр EDX (память→рег)
	13	3d 55 f8	cmp edx, DWORD PTR [ebp-0x8]	Сравнение регистра EDX и переменной b (a>b)
	16	7e 06	jle .+6	Условный переход на ветвь else (на 6 байт вперёд). если оказалось, что переменная a ≤ переменной b
res = true;	18	c6 45 ff 01	mov BYTE PTR [ebp-0x1], 0x1	Запись константы в переменную в памяти (res = true)
else	1c	eb 04	jmp .+4	Безусловный переход на конец программы (на 4 байта вперёд)
res = false;	1e	c6 45 ff 00	mov BYTE PTR [ebp-0x1],0x0	Запись константы в переменную в памяти (res = false)

Операторы программы на С	Машинный код для RISC-процессора ARM			
	Адр.	Маш. код	Ассемблер	Комментарий
`Int a, b;` `bool res;`				
`a = a + b;`	00	e5 1b 20 0c	ldr r2, [fp, #-12]	Загрузка переменной a в регистр r2 (память→рег)
	04	e5 1b 30 10	ldr r3, [fp, #-16]	Загрузка переменной b в регистр r3 (память→рег)
	08	e0 82 30 03	add r3, r2, r3	Сложение результатов r2 и r3, запись результата в r3
	0c	e5 0b 30 0c	str r3, [fp, #-12]	Выгрузка из регистра r3 переменной a (рег→память)
`b = 3 * b;`	10	e5 1b 20 10	ldr r2, [fp, #-16]	Загрузка переменной b в регистр r2 (память→рег)
	14	e1 a0 30 02	mov r3, r2	Копирование значения регистра r2 в регистр r3
	18	e1 a0 30 83	lsl r3, r3, #1	Сдвиг регистра r3 на 1 бит влево (смысл: r3 *= 2)
	1c	e0 83 30 02	add r3, r3, r2	Прибавление значения регистра r2 к регистру r3
	20	e5 0b 30 10	str r3, [fp, #-16]	Выгрузка переменной b из регистра r3 (рег→память)
`if ( a>b )`	24	e5 1b 20 0c	ldr r2, [fp, #-12]	Загрузка переменной a в регистр r2 (память→рег)
	28	e5 1b 30 10	ldr r3, [fp, #-16]	Загрузка переменной b в регистр r3 (память→рег)
	2c	e1 52 00 03	cmp r2, r3	Сравнение регистров r2 и r3 (a>b)
	30	da 00 00 02	ble #+2	Условный переход через 3 команды, если оказалось, что переменная a ≤ переменной b
`res = true;`	34	e3 a0 30 01	mov r3, #1	Загрузка в регистр r3 константы 1 (true)
	38	e5 4b 30 05	strb r3, [fp, #-5]	Выгрузка переменной res из регистра r3 (рег→память)
`else`	3c	ea 00 00 01	b #+1	Безусловный переход на конец программы (через 2 команды)
`res = false;`	40	e3 a0 30 00	mov r3, #0	Загрузка в регистр r3 константы 0 (false)
	44	e5 4b 30 05	strb r3, [fp, #-5]	Выгрузка переменной res из регистра r3 (рег→память)

Вопросы

1. Дайте определение CISC-архитектуры.
2. Перечислите свойства машинных языков CISC-процессоров.
3. Назовите преимущества и недостатки CISC-архитектуры.
4. Дайте определение RISC-архитектуры.
5. Перечислите свойства машинных языков RISC-процессоров.
6. Назовите преимущества и недостатки RISC-архитектуры.
7. Назовите обстоятельства, повлиявшие на популярность RISC-архитектур.
8. Почему в настоящее время недружественный машинный язык RISC-процессоров не является существенным недостатком?
9. Приведите пример сложной команды, расскажите, как она реализована.
10. Что такое микрокод, и для чего он используется?
11. Приведите пример сложной машинной команды CISC-процессора.
12. Перечислите варианты видов аргументов для двухаргументных машинных команд.
13. Продемонстрируйте разницу между CISC- и RISC-процессорами на примере операции сложения.
14. Какие программы имеют больше машинных команд и почему — программы для CISC- или RISC-процессоров?
15. Какие программы имеют больший объем машинного кода — для CISC- или RISC-процессоров?
16. Какие программы выполняются быстрее — для CISC- или RISC-процессоров?
17. Оттранслируйте фрагмент программы на языке C для CISC-процессора, прокомментируйте результаты трансляции.
18. Оттранслируйте фрагмент программы на языке C для RISC-процессора, прокомментируйте результаты трансляции.

Литература

1. *Гуров В. В.* Архитектура микропроцессоров. М.: ИНТУИТ.РУ, Бином. 2010. 272 с.
2. *Хорошевский В. Г.* Архитектура вычислительных систем: учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.
3. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
4. *Patterson D. A., Ditzel D. R.* The case for the reduced instruction set computer // ACM SIGARCH Computer Architecture News. 1980. Vol. 8. № 6. P. 25–33.
5. Intel 80386 Programmer's Reference Manual. Intel Corporation, 1986.

Лекция 10. Конвейеризация

Определение конвейера; конфликты при конвейерном исполнении машинного кода; особенности обработки конфликтов в CICS- и RISC-архитектурах; пример генерации машинного кода для CISC-процессора ARM и RISC-процессора MIPS для демонстрации приёмов обработки потенциальных конфликтов конвейеризации на уровне компиляции; параллельное исполнение команд на примере процессоров Intel x86.

Определение конвейера

Команды процессора часто задействуют значительное количество блоков процессора и исполняются за несколько тактов. Например, рассмотрим команду сложения числа из регистра с числом из памяти `add eax, DWORD PTR [ebp-0x8]` в процессоре Intel 80386. Эта команда выполняется следующим образом.

1. В начале выполняется чтение числа из регистра `eax` в арифметико-логическое устройство (АЛУ).
2. Значение адресного регистра `ebp` и константа `-0x8` подаются на вход адресному сумматору (заметим, что здесь речь не о сумматоре, входящем в состав АЛУ, а об отдельном адресном сумматоре, предназначенном для арифметических действий над адресами; различных специализированных сумматоров в процессоре может быть достаточно много).
3. Вычисляется адрес в памяти для второго операнда.
4. Выполняется чтение по этому адресу числа из памяти во временный регистр, соединённый с АЛУ.
5. Выполняется сложение двух чисел.
6. Результат записывается в регистр `eax`.

Таким образом, эта команда выполняется за 6 тактов (следует отметить, что этот пример существенно упрощён для наглядности). При этом если не использовать дополнительных оптимизаций, то на каждом такте задействуется лишь часть блоков процессора, а остальные простаивают. Например, при сложении двух значений, которые находятся в регистрах процессора, блок работы с памятью бездействует.

> ***Конвейер*** *— это технология, предназначенная для параллельного выполнения команд программы различными блоками процессора. Она позволяет загрузить блоки процессора при выполнении команд оптимально, без простоев*[8].

Приведём пример конвейера. Рассмотрим упрощённый процессор, у которого каждая команда исполняется в три этапа, точнее, за три такта:

- чтение аргументов команды из памяти или регистра (ЧТН);
- исполнение команды (ИСП);
- запись результата в память или регистр (ЗАП).

На рис. 20 изображен процесс выполнения этим процессором последовательности команд Команда 1, Команда 2, Команда 3, Команда 4, Команда 5 с использованием конвейера. Рассмотрим четвёртый такт конвейера, который выделен на рисунке. Команда 1 уже полностью выполнена, Команда 2 находится на этапе записи данных (ЗАП), Команда 3 — на этапе исполнения (ИСП), Команда 4 — на этапе чтения данных (ЧТН), а к выполнению Команды 5 процессор ещё не приступал. Из-за того, что на каждом такте процессор выполняет с помощью конвейера сразу несколько частей (этапов) команд, все пять команд выполняются за 7 тактов, а не за 15, как было бы при последовательном исполнении этой же цепочки команд. Это оказывается возможным ввиду того, что процессор параллельно выполняет различные этапы команд.

Время →

↓ Программа	**Команды**	**Такты**						
		1-й	**2-й**	**3-й**	**4-й**	**5-й**	**6-й**	**7-й**
	Команда 1	ЧТН	ИСП	ЗАП				
	Команда 2		ЧТН	ИСП	**ЗАП**			
	Команда 3			ЧТН	**ИСП**	ЗАП		
	Команда 4				**ЧТН**	ИСП	ЗАП	
	Команда 5					ЧТН	ИСП	ЗАП

Рис. 20. Исполнение машинных команд с использованием конвейеризации

[8] Конвейеры появились в 1950-х годах, термин «конвейер» (Pipeline) ввёл конструктор советских ЭВМ С. А. Лебедев.

Конфликты при конвейерном исполнении машинных команд

Результаты выполнения команд используются следующими командами программы. Посмотрим, как это простое соображение согласуется с конвейером.

Приведённый выше пример показывает, что Команда 3 не сможет использовать результаты Команды 2, поскольку свои входные данные она читает раньше, чем Команда 2 записывает свои результаты. В том случае, когда Команде 3 действительно нужны результаты Команды 2, Команда 2 и Команда 3 называются зависимыми.

> *Ситуации, возникающие при конвейеризованном исполнении команд, которые препятствуют корректному выполнению очередной команды, называются* ***конфликтами****.*

Ниже перечислены виды конфликтов.

1. *Конфликт по данным* между зависимыми машинными командами заключается в том, что на конвейере одновременно находятся на разных стадиях выполнения команды, которые могут быть корректно исполнены лишь строго последовательно.
2. *Конфликт по ресурсам* возникает в ситуации, когда двум командам на конвейере одновременно нужен доступ к какому-либо блоку процессора, с которым в один момент времени может работать только одна команда.
3. *Конфликт по управлению* заключается в том, что следующая команда на конвейере является условным переходом, но условие для него ещё не вычислено предыдущей командой и непонятно, какую ветку условного оператора следует загружать на конвейер.

Поскольку конфликты по данным являются распространённой на практике проблемой, приведём ряд примеров таких конфликтов.

- Следующая команда на конвейере должна читать данные на выходе предыдущей, но предыдущая ещё не закончила их обработку; при этом нарушается зависимость «чтение после записи».
- Следующая команда записывает данные, но они используются (читаются) предыдущей командой, то есть при этом нарушается зависимость «запись после чтения».
- Следующая команда записывает (в регистры процессора или в оперативную память) свои результаты раньше предыдущей, из-за чего предыдущая потом может перезаписать эти результаты: при этом нарушается зависимость «запись после записи».

Сделаем замечание относительно третьего вида конфликтов (по управлению). Такие конфликты дополнительно осложняются тем обстоятельством, что заранее неясно, в какое место программы произойдёт условный переход, и поэтому непонятно, из какой его ветки загружать на конвейер следующие команды. В связи с этим во многих процессорах есть *система предсказания условных переходов*, которая собирает статистику по переходам и выбирает наиболее вероятный вариант. В случае ошибки предсказания конвейер очищается от частично обработанных команд, что вызывает задержки.

Для того чтобы программа могла работать корректно с применением конвейера, конфликтные ситуации требуется обрабатывать, то есть обнаруживать и разрешать. Обнаружение конфликтов оказывается непростой задачей, которую мы не будем здесь рассматривать. Разрешение конфликтов может выполняться различными способами. Самым простым способом является *«торможение» конвейера (Pipeline stall)* с целью задержки выполнения следующей команды до момента исчерпания конфликта. Но, во-первых, конфликтную ситуацию требуется обнаружить, что непросто, а во-вторых, задержки уменьшают преимущества конвейера.

Ряд подходов к обработке конфликтов представлен ниже.

1. Статическое переупорядочивание машинных команд при компиляции программ с языков высокого уровня в машинный код.
2. «Разнесение» конфликтующих команд при компиляции на безопасное расстояние друг от друга с помощью вставки необходимого количества специальной команды NOP (No Operation). Команда NOP ничего не делает, но замедляет работу программы на один такт, в некоторых процессорах — и на большее число тактов.
3. Динамическая обработка конфликтов во время исполнения программы — идентификация и разрешение конфликтов выполняется в момент выполнения программы. При этом процессор задерживает выполнение зависимых команд (как, например, Intel 80486), а также самостоятельно переупорядочивает команды, чтобы исключить конфликты с минимизацией потери времени (так действуют процессор Intel Pentium и последующие процессоры семейств Intel x86).

Особенности обработки конфликтов в CISC- и RISC-архитектурах

Рассмотрев общие подходы к обработке конфликтов на конвейерах, остановимся теперь на особенностях этой процедуры в CISC- и RISC-процессорах. Свойства машинного языка оказывают определяющее влияние на то, каким образом можно организовать конвейерное исполнение машинного кода. Как следствие, различия между CISC- и RISC-процессорами приводят к использованию различных подходов к обработке конфликтов.

CISC-процессоры делают акцент на динамической обработке конфликтов, поскольку в CISC-процессорах команды имеют различную сложность и время работы, задействуют разные блоки процессора и обращаются к операндам различного вида. Кроме того, в некоторых семействах динамическая обработка конфликтов является единственным возможным решением, поскольку механизм конвейеризации в них появился не сразу и требовалась совместимость с предыдущими версиями процессоров, то есть механизм конвейеризации не является глубоко интегрированным в процессор.

Рассмотрим пример. Первым конвейеризованным процессором семейства Intel x86 был процессор Intel 80486. Для того чтобы у него была возможность корректно и по возможности быстро исполнять машинный код, предназначенный для предыдущих версий неконвейеризованных процессоров, ничего не оставалось, кроме как «научиться» разрешать конфликты динамически. Ни на какие «подсказки» компилятора он при этом рассчитывать не мог, ведь существующий машинный код для предыдущих процессоров семейства Intel x86 про конвейеризацию ничего не знал.

Реализация динамической обработки конфликтов в CISC-процессорах требует значительного усложнения конвейера. Для сравнения можно представить себе, как усложнился бы конвейер на машиностроительном заводе, если бы на нём приходилось одновременно собирать бульдозеры, комбайны и танки, вдобавок разных моделей. Отметим, что иногда компиляторы и низкоуровневые программисты всё же помогают CISC-процессорам, создавая максимально «безконфликтный» машинный код. Это положительно сказывается на скорости работы программ на старых процессорах, которые, как, например, Intel 80486, умеют при обнаружении конфликтов лишь задерживать выполнение команд, но не переупорядочивать их.

Машинный код RISC-процессоров, с точки зрения способа хранения команд в оперативной памяти, оказывается проще, чем у CISC-процессоров. Команды RISC-процессоров выполняются за одинаковое число тактов, а более простой машинный язык упрощает обнаружение и обработку конфликтов. Некоторые современные RISC-процессоры (например, процессоры семейства ARM), также как и CISC-процессоры, могут обнаруживать конфликты и переупорядочивать исполнение ко-манд. Другие же RISC-процессоры вовсе не обрабатывают конфликты, например, ранние версии процессоров семейства MIPS — даже само название этого семейства расшифровывается как Microprocessor without Interlocked Pipelined Stages, то есть микропроцессор без блокировок на конвейере. Такие процессоры требуют, чтобы подготовленный для них машинный код не содержал потенциальных конфликтов конвейеризации, и эту «бесконфликтность» обеспечивает компилятор.

Параллельное исполнение команд

Как уже многократно указывалось, повышение производительности современных процессоров осуществляется в значительной степени за счёт

распараллеливания. Одним из популярных способов распараллеливания является конвейеризация. Но хотя конвейеризация и позволяет значительно ускорить исполнение программ, максимального эффекта она достигает лишь в идеальных ситуациях, когда конфликты не возникают или когда удаётся переупорядочить команды таким образом, чтобы эти конфликты исчерпать. Для дальнейшего повышения производительности применяются (иногда в сочетании с конвейеризацией) более сложные решения.

В качестве первой техники параллельного исполнения команд процессором рассмотрим *суперскалярность* — возможность одновременно выполнять несколько машинных команд за счёт наличия в процессоре нескольких однотипных функциональных блоков (арифметико-логических устройств, математических сопроцессоров и т. д.) В семействе Intel x86 первым процессором, где была реализована суперскалярность, был процессор Intel Pentium (1993 г.). Этот процессор содержал два арифметико-логических устройства, которые позволяли исполнять одновременно две соседние команды, если они не зависели друг от друга. При этом независимые команды одновременно обрабатывались двумя разными конвейерами. Для этого компиляторы стремились генерировать машинный код, соседние команды которого не зависели бы друг от друга.

Более продвинутой техникой параллельного исполнения машинных команд является *внеочередное исполнение (Out of Order Execution)*. Эта техника позволяет исполнять команды программы не в порядке следования, а в порядке готовности к выполнению. При использовании внеочередного исполнения некоторое множество команд программы исполняется тогда, когда для всех этих команд готовы нужные им входные данные. Первой ЭВМ, в которой был реализован механизм внеочередного исполнения команд, был суперкомпьютер CDC 6600 компании Cray Research, созданный в 1963 году.

Приведём следующий пример. Пусть в программе подряд идут следующие команды:

```
(1) a = b / c
(2) d = b + a
(3) x = y * z
```

Деление двух чисел (Команда 1) является сложной командой, которая выполняется существенно дольше, чем, например, сложение. Вспомните алгоритм деления «в столбик»: в нём последовательно выполняется много операций и проверок. Процессор выполняет деление практически так же, но только в двоичной системе. В то же время от результатов Команды 1 зависит выполнение Команды 2, а выполнение Команды 3 не зависит. Поэтому можно приступить к выполнению Команд 1 и 3 одновременно, а Команду 2 начать выполнять после вычисления значения `a`.

Рассмотрим общий сценарий внеочередного исполнения команд в современных процессорах на примере распространённого известного процессора

Intel Core i7. После выборки процессором очередной команды для исполнения эта команда делится на *микрооперации* — простые действия, такие как «сложить два числа», «записать значение в регистр», «выдать адрес ячейки на шину адреса» и подобные им. Микрооперации помещаются в специальный *буфер переупорядочивания (Reorder Buffer)*, в котором они упорядочиваются так, чтобы, с одной стороны, не нарушить корректность программы, а с другой стороны, оптимально загрузить блоки процессора. Причём в буфере находятся и переупорядочиваются одновременно микрооперации нескольких идущих подряд команд. Чтобы получить больше свободы для переупорядочивания команд, процессор выполняет переименование регистров: для разных микроопераций, работающих с одними и теми же регистрами, назначаются новые регистры. Получившиеся микрооперации выполняются шестью блоками процессора — тремя АЛУ и тремя блоками доступа к памяти. При обнаружении условного перехода одновременно начинают обрабатываться обе ветви `then` и `else`, то есть используется *спекулятивное* исполнение программы. Но после фактического вычисления условия фиксируются только результаты команд из выбранной в переходе ветки.

Компания Intel реализовала внеочередное исполнение команд задолго до запуска в производство современных многоядерных процессоров, в рамках процессоров Intel Pentium Pro в начале 2000-х годов. Было замечено, что блоки процессора выполняли микрооперации эффективно, но часто простаивали, ожидая результатов выборки машинных команд из оперативной памяти. Для того чтобы оптимально загрузить эти блоки, была создана технология Hyper-Threading. Она выбирала машинные команды из оперативной памяти в двух параллельных потоках. Одноядерный процессор с применением технологии Hyper-Threading фактически начинает работать как двухъядерный. Технология Hyper-Threading была запатентована сотрудником компании Sun Microsystems в 1994 году, но впервые была реализована лишь в 2002 году в компании Intel.

Вопросы

1. Дайте определение конвейера.
2. Дайте определение конфликта на конвейере.
3. Перечислите виды конфликтов конвейера.
4. Что такое зависимые команды?
5. Что такое конфликт по ресурсам?
6. Что такое конфликт по управлению?
7. Перечислите способы преодоления конфликтов.
8. Что такое статическое переупорядочивание команд при решении конфликтов?
9. Расскажите об использовании команды NOP для разрешения конфликтов.
10. Что такое динамическое разрешение конфликтов?

11. Какой подход используется при разрешении конфликтов на конвейерах CICS-процессоров и почему?
12. Какой подход используется при разрешении конфликтов на конвейерах RICS-процессоров и почему?
13. Зачем нужна система предсказания условных переходов?
14. Как CISC-процессоры разрешают конфликты конвейера?
15. Каковы особенности разрешения конфликтов в RISC-процессорах?
16. Почему конвейеризация в RISC-процессорах реализована проще, чем в CISC?
17. Что такое суперскалярность?
18. Что такое внеочередное исполнение машинных команд?
19. Расскажите про технологию Hyper-Threading.

Литература

1. *Гуров В. В.* Архитектура микропроцессоров. М.: ИНТУИТ.РУ, Бином. 2010. 272 с.
2. *Хорошевский В. Г.* Архитектура вычислительных систем: учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.
3. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
4. *Patterson D. A., Ditzel D. R.* The case for the reduced instruction set computer // ACM SIGARCH Computer Architecture News. 1980. Vol. 8. № 6. P. 25–33.

Лекция 11. Кэш-память

Назначение и способы организации кэш-памяти; стратегии замещения; принцип локальности; особенности организации кэш-памяти в многоядерных процессорах на примере Intel Core i7.

Назначение и способы организации кэш-памяти

Как следует из принципов архитектуры фон Неймана, память ЭВМ неоднородна и в ней имеется иерархия запоминающих устройств. Одним из элементов этой иерархии в современных компьютерах является кэш-память процессора.

> ***Кэш-память процессора (Cache)*** *— это особый вид быстрой памяти, которая находится в процессоре и предназначается для хранения данных, активно используемых при выполнении данной программы в текущий момент времени, что позволяет существенно ускорить работу и снизить нагрузку на оперативную память и системную шину.*

Размер кэш-памяти существенно меньше и самой оперативной памяти, и той её части, которая выделяется для работы одной программы. Поскольку эта память находится внутри процессора, то ограничения на размеры кремниевой пластины, на которой находится процессор, и различные другие факторы не позволяют сделать эту память достаточно большой. Поэтому по мере выполнения программы данные в кэш-памяти постоянно обновляются (эта процедура называется замещением и будет рассмотрена ниже).

Фактически современные процессоры не работают напрямую с данными в оперативной памяти, но предварительно загружают их в кэш-память. При этом исполняемая процессором программа не контролирует работу с кэш-памятью и не знает о том, помещены ли туда необходимые данные туда, то есть для неё кэш-память является прозрачной. Cache в переводе с английского означает «тайник», что как раз отражает прозрачность этого вида памяти для прикладной программы.

Обмен данными между оперативной памятью и кэш-памятью осуществляется блоками фиксированного размера. Такой блок будем называть *строкой кэш-памяти*. Размер одной строки и общий объём являются основными характеристиками кэш-памяти, которые отличаются для разных моделей процессоров. Например, в процессорах Intel Core i7 размер кэш-памяти ва-

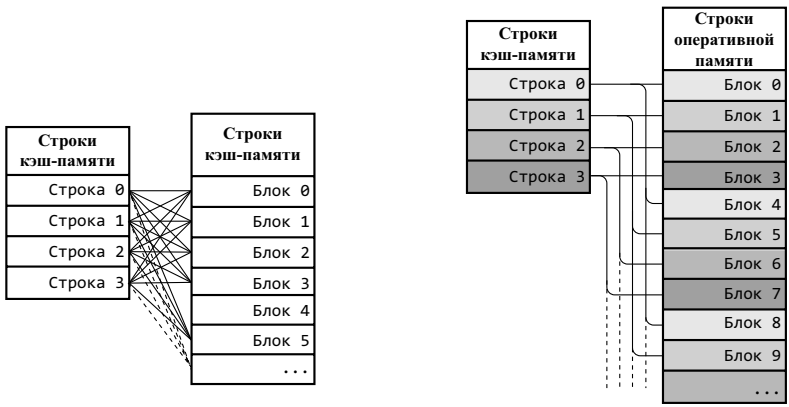
рьируется от 4 до 12 Mb, а размер строки кэш-памяти составляет 64 байта. При одинаковом общем размере кэш-память, состоящая из большего количества строк, эффективнее, но сложнее в реализации.

Современные процессоры имеют многоуровневую кэш-память. Можно сказать, в этом случае принцип иерархических запоминающих устройств фон Неймана реализуется следующим образом: уровни с меньшими номерами кэшируют наиболее часто используемые данные в уровнях с бóльшими номерами, и при этом каждый следующий уровень имеет больший размер и меньшее быстродействие, чем предыдущий.

Чтобы для исполняемой программы кэш-память была прозрачной, используется *механизм отображения кэш-памяти*. Этот механизм поддерживает соответствие между данными в кэш-памяти и данными в оперативной памяти. Последнее важно, так как в программном коде, в операндах команд, которые работают не с регистрами, а с оперативной памятью (например, команда сложения вида регистр/память), указаны адреса в оперативной памяти, а не в кэш-памяти, и обращения по этим адресам нужно заменить на обращения в кэш-память. Эту подмену и реализует механизм отображения.

Механизм отображения может быть ассоциативным, прямым или гибридным.

- *Ассоциативный механизм отображения* обеспечивает возможность связать каждую строку кэш-памяти с любым блоком оперативной памяти (см. рис. 21а). Это наиболее гибкий механизм, но его аппаратная реализация является дорогостоящей и потребляет много энергии. Это происходит из-за того, что при обращении процессора к оперативной памяти по определённому адресу все строки кэш-памяти должны опрашиваться на предмет того, не содержат ли они копию соответствующего блока оперативной памяти. Также при выделении новой или освобождении занятой строки кэш-памяти процессор должен поддерживать перечень свободных строк, который нужен для того, чтобы при необходимости выделить очередную строку и быстро найти свободную, а не проверять все имеющиеся строки.
- *Прямой механизм отображения* позволяет связать с каждым блоком памяти лишь одну из строк кэш-памяти (см. рис. 21б). Прямое отображение существенно дешевле в реализации, чем ассоциативное, но оно одновременно и менее эффективно.
- Во многих современных процессорах, в том числе в процессорах семейства Intel x86, используется *механизм гибридного отображения*: на один и тот же блок оперативной памяти может отображаться не единственная (как в прямом отображении) и не произвольная (как в ассоциативном) строка кэш-памяти, а несколько строк, часто — 4 или 8, в зависимости от модели процессора. Такой механизм значительно более гибок, чем прямое отображение, но при этом не столь сложен в реализации, как механизм ассоциативного отображения.



(а) ассоциативное отображение *(б) прямое отображение*

Рис. 21. Ассоциативное и прямое отображения на примере кэш-памяти из четырёх строк

Стратегии замещения и принцип локальности

Поскольку объём кэш-памяти процессора существенно меньше объёма оперативной памяти, уже после непродолжительной работы программы кэш-память оказывается полностью занятой. При необходимости обращения к блоку оперативной памяти, который не был помещён в кэш-память, он перемещается туда. Но для этого нужно выбрать строку в кэш-памяти, в которую нужно пометить данный блок. Для этого процессор выполняет *операцию замещения*, состоящую из следующих шагов:

- выбор подходящей строки кэш-памяти для замещения;
- сохранение в оперативной памяти актуальных данных из этой строки;
- копирование в рассматриваемую строку соответствующего блока оперативной памяти.

Способ, которым в кэш-памяти выбирается строка для замещения, называется *стратегией замещения*. При замещении обычно используется *принцип Белади*: следует заместить строку, содержащую данные, которые не понадобятся процессору дольше всего. Но однозначно выбрать такую строку нельзя, поэтому используются различные эвристические подходы. Самым распространенным является стратегия *LRU (Least Recently Used)*, которая фиксирует время последнего обращения к каждой строке кэш-памяти и замещает ту из них, к которой процессор дольше всего не обращался. Поясним, как эта стратегия может быть реализована в зависимости от используемого механизма отображения.

- Механизм ассоциативного отображения позволяет реализовать стратегию LRU непосредственно в том виде, в котором она описана выше.
- Для механизма прямого отображения выбор строки для замещения является тривиальным, так как с каждым блоком оперативной памяти может быть связана лишь одна строка кэш-памяти, которую и следует заместить.
- Для механизма гибридного отображения стратегию LRU можно реализовать, выбирая из тех строк, которые могут быть связаны с данным блоком оперативной памяти.

Несмотря на то, что программа на C/C++, Java, Python и т. д. не может управлять кэш-памятью процессора, при разработке программ иногда её надо учитывать. Например, при работе с данными, превышающими размер кэш-памяти, следует соблюдать принцип локальности.

> ***Принцип локальности*** *утверждает, что данные, которые обрабатываются в одном фрагменте программы, должны быть расположены в оперативной памяти рядом.*

Приведём пример программы на языке C для процессора Intel Core i7 (как было указано выше, кэш-память этого процессора составляет от 4 до 12 Mb в зависимости от его модели).

```
int my_array[10000][10000];

...

for(int i = 0; i < 10000; ++i)
  for (int j = 0; j < 10000; ++j)
    my_array[i][j] *= 2;
```

В этом фрагменте объявлен двумерный массив (матрица) целых чисел `my_array` размером 10000×10000. Целое число (int) в современных компиляторах языка C представляется четырьмя байтами. Соответственно, для массива `my_array` требуется $4 * 10000 * 10000$ байт $\cong$ 400 мегабайт. Очевидно, что этот массив не помещается в кэш-память процессора целиком. Следовательно, по мере того как программа будет работать с этим массивом, в кэш-память процессора будут подгружаться из оперативной памяти все новые и новые фрагменты данного массива. Далее двумерные массивы языка C хранятся в оперативной памяти построчно: `my array[0][0],... my_array [0][9999], my_array [1][0], ... my_array [1][9999], ...`. Отметим, что именно в этом порядке элементы массива обрабатываются в цикле `for` в программе, представленной выше: то есть, когда в кэш-память из оперативной памяти загружается строка, программа полностью её обрабатывает и больше не обращается к этим данным. Затем в кэш-память загружается следующая

строка и так далее. Очевидно, что в этой программе принцип локальности соблюдается.

Изменим эту программу так: заменим оператор `my_array [i][j] *= 2` на `my_array [j][i] *= 2`. Теперь последовательно обрабатываемые элементы массива расположены в оперативной памяти далеко друг от друга, и программа вынуждена постоянно «прыгать» по массиву, что влечет многократную загрузку/выгрузку одних и тех же строк кэш-памяти. В такой программе принцип локальности не соблюдается, и она будет работать медленнее, чем предыдущая.

Кэш-память многоядерных процессоров

Реализация кэш-памяти многоядерных процессоров, да и в целом вычислительных устройств с аппаратной поддержкой многозадачности, сопряжена с рядом трудностей. Одна из наиболее значительных трудностей вызвана необходимостью обеспечить *когерентность кэш-памяти*.

> ***Когерентность*** *— свойство многоядерного процессора, подразумевающее согласованность данных в кэш-памяти его разных ядер при исполнении программ.*

Рассмотрим пример сбоя, который может произойти при использовании кэш-памяти, не обладающей свойством когерентности. Допустим, что в программе на языке C есть строка `int x, y`. В оперативной памяти эти переменные размещаются рядом и попадают в один блок, который передается в кэш-память и потом, через некоторое время, «возвращается» в оперативную память из кэш-памяти. Далее предположим, что у нас имеется двухъядерный процессор и каждое из ядер меняет одну из этих переменных — ядро 1 меняет переменную `x`, а ядро 2 меняет переменную `y` (это может происходить, например, в разных потоках нашей программы). Таким образом, процессор выполняет действия, перечисленные ниже.

1. Ядро 1 читает значение переменной `x`, записывая в свою кэш-память соответствующий блок оперативной памяти.
2. Ядро 2 читает значение переменной `y`, записывая в свою кэш-память тот же блок оперативной памяти (напоминаем, что обе переменные находятся рядом, а кэш-память и оперативная память не могут обмениваться фрагментами памяти, соответствующими только одной переменной, и захватывают лишнее; в этом-то и проблема!).
3. Ядро 1 записывает новое значение в переменную `x` в своей кэш-памяти.
4. Ядро 2 записывает новое значение в переменную `y` в своей кэш-памяти.

5. В оперативную память записывается строка кэш-памяти ядра 1; данная строка содержит как измененное значение переменной x, так и прежнее значение переменной y.
6. В оперативную память записывается строка кэш-памяти ядра 2; данная строка содержит как старое значение переменной x, так и измененное значение переменной y.

Очевидно, что на шаге 6 значение переменной x окажется прежним, которое было до того, как ядро 1 изменило эту переменную. Таким образом, результаты действий ядра 1 по изменению переменной x окажутся потерянными, что является ошибкой.

Для того чтобы избежать таких проблем, сохранив при этом эффективность работы кэш-памяти, в многоядерных процессорах приходится применять сложные технические решения.

В качестве примера рассмотрим процессор Intel Core i7 (рис. 22). Этот процессор имеет трёхуровневую кэш-память. Внутри каждого ядра находится кэш-память первого уровня (L1); она является самой быстрой. Для повышения производительности кэш-память уровня L1 организована по принципу гарвардской архитектуры: машинный код и данные хранятся в ней раздельно. Кэш-память второго уровня (L2) имеет больший объём и меньшее быстродействие, в ней код и данные хранятся вместе. У каждого процессорного ядра имеется своя кэш-память второго уровня, и другие ядра с ней работать не могут. Наконец, кэш-память третьего уровня (L3) является общей для всех ядер, и она оказывается самой большой, но и самой медленной. Посредством кэш-памяти уровня L3 процессор обращается к системной шине для доступа к оперативной памяти.

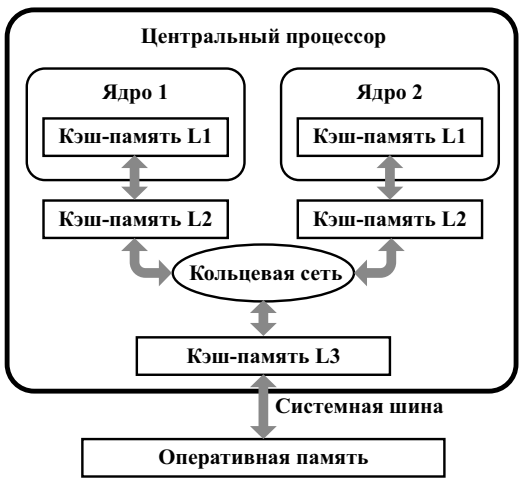


Рис. 22. Организация кэш-памяти в двухъядерном процессоре Intel Core i7

Для обеспечения когерентности внутри процессора Intel Core i7 используется внутренняя шина специальной конструкции, которая называется *кольцевой сетью*. К ней последовательно подключены несколько кэшей разных ядер. Получив запрос на предоставление данных, каждый из них решает, обработать ли этот запрос или передать его дальше. Когда ядро 1 на рис. 22 обращается к блоку памяти, копии которого нет в его внутренней кэш-памяти L1, этот запрос обрабатывает его кэш L2. Если требуемые данные нашлись в кэш-памяти L2 ядра 1, то она возвращает их ядру 1. Если же в кэш-памяти L2 ядра 1 не нашлось нужных данных, то выполняется запрос кэш-памяти L2 следующего ядра — в данном случае ядра 2. Если в кэш-памяти L2 ядра 1 требуемые данные нашлись, то она обрабатывает запрос и возвращает данные в кэш-память L2 ядра 0. Если ни в одном из кэшей L2 ни нашлось требуемых данных, запрос возвращается необработанным ядру 1, и кольцевая сеть повторно направляет его уже в кэш-память L3 (который в случае промаха обращается уже к оперативной памяти). Как только какое-то из ядер вносит изменения в кэш-данные, кольцевая сеть отправляет остальным ядрам сообщение о том, что копии этих данных в их кэш-памяти потеряли актуальность и в случае последующих обращений их надо будет запросить у изменившего их ядра. В рассмотренном выше примере с кэшированием двух переменных `x` и `y` из разных потоков кольцевая сеть внесет следующие изменения в действия процессора (в этом примере мы описываем, как это происходит в процессоре Intel Core i7).

- На шаге 3 ядро 1 по кольцевой сети сообщает, что все копии блока памяти, за исключением той, которая находится в кэш-памяти ядра 1, становятся неактуальными.
- На шаге 4, прежде чем менять значение переменной `y`, ядро 2 запрашивает актуальные данные строки кэш-памяти у ядра 1, затем меняет значение переменной `y`. А после изменения сообщает ядру 1, что актуальная копия теперь у него.
- Шагов 5 и 6 не будет, поскольку ядра процессора Intel Core i7 не взаимодействуют с оперативной памятью самостоятельно. Вместо этого в какой-то момент актуальная копия блока памяти попадёт из кэш-памяти ядра 2 в общую кэш-память L3, а ещё позже из L3 — в оперативную память.

Вопросы

1. Дайте определение кэш-памяти процессора.
2. Назовите основные характеристики кэш-памяти.
3. Что такое механизм отображения кэш-памяти?
4. Зачем нужна иерархическая кэш-память?
5. Что такое механизм прямого отображения кэш-памяти?
6. Что такое механизм ассоциативного отображения кэш-памяти?

7. Что такое механизм смешанного отображения кэш-памяти?
8. Опишите процедуру выполнения операции замещения.
9. Что такое стратегия замещения?
10. Расскажите про LRU-стратегию.
11. Что такое принцип Белади?
12. Расскажите, в каких отношениях находится прикладная программа с кэш-памятью.
13. Что такое принцип локальности?
14. Приведите пример соблюдения локальности при программировании на языке C.
15. Приведите пример несоблюдения принципа локальности при программировании на языке C.
16. Проверьте на практике, что программа, описанная в лекции и не соблюдающая принцип локальности, будет работать медленнее, чем та, в которой этот принцип соблюдается.
17. Что такое когерентность кэш-памяти?
18. Расскажите, как с помощью кольцевой сети реализуется когерентность кэш-памяти ядер в процессоре Intel Core i7.

Литература

1. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
2. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
3. *Bahn H., Noh S. H.* Characterization of Web reference behavior revisited: Evidence for Dichotomized Cache management // International Conference on Information Networking 2003. Jeju, South Korea: Springer-Verlag. P. 1018–1027.

Лекция 12. Адресное пространство и виртуальная память

Физический адрес, прямая адресация, адресное пространство программы. Сегментная адресация: сегменты, их назначение, виртуальный адрес, адресное преобразование; виртуальная память: страничная адресация, страницы, страничное адресное преобразование и их использование при реализации виртуальной памяти; замещение страниц физической памяти, выделение страниц по мере необходимости.

Физический адрес и прямая адресация

При старте программы операционная система создаёт для нее специальный процесс. Можно сказать, что процесс является экземпляром программы, точнее — запущенным на компьютере экземпляром. Как уже упоминалось выше, одновременно на компьютере может быть запущено несколько экземпляров одной и той же программы, с чем вы наверняка сталкивались — например, работая в Windows, легко запустить несколько экземпляров текстового редактора «Блокнот». Далее мы иногда будем использовать термины «процесс» и «программа» как синонимы, считая, что из контекста понятно, о чём идёт речь.

В оперативную память созданного процесса копируется код соответствующей программы, а также выделяется память для данных программы. Процессор взаимодействует с этой памятью посредством адресов. Система адресации у современных компьютеров устроена сложно, и мы начнём её изучать с понятия физического адреса.

> ***Физический адрес (Physical Address)*** *— это номер ячейки в оперативной памяти. Зная этот номер, процессор непосредственно обращается в нужное место оперативной памяти за необходимыми данными.*

Когда мы говорим об адресуемой ячейке в оперативной памяти, то мы имеем в виду машинное слово, которое, как рассказывалось в предыдущих лекциях, является минимально адресуемым фрагментом памяти.

Многие ЭВМ прошлых поколений, а также простые современные вычислительные устройства позволяют своим программам обращаться к оперативной памяти напрямую, по физическому адресу, минуя всевозможные

адресные преобразования, о которых пойдёт речь в этой лекции. Такая организация работы с памятью называется *прямой адресацией*.

> ***Прямая адресация*** *— это механизм использования в машинном коде программ реальных физических адресов в оперативной памяти без применения адресных преобразований.*

Хотя данная схема на первый взгляд выглядит естественной, она не обладает достаточной гибкостью — ведь в действительности не всё так просто! Поэтому в современных персональных компьютерах, смартфонах, планшетах и других сложных вычислительных устройствах прямая адресация не применяется, поэтому нам есть о чём поговорить в рамках данной лекции.

Дадим теперь определения адресного пространства программы. Казалось бы, это — вся оперативная память, которая выделяется процессу. Однако это не так, поскольку программа может оперировать с оперативной памятью большего объёма, чем выделено её процессу. Для этого используется механизм виртуальной памяти, о котором речь пойдёт чуть ниже. А тут мы дадим корректное определение адресного пространства программы.

> ***Адресное пространство (Address Space)*** *— это множество адресов, допустимое для использования программой.*

В частности, значения указателей в программах на языке C, работающих на компьютерах семейства процессоров Intel x86, являются элементами адресного пространства.

Сегментная адресация

Память, которая выделяется процессу при старте операционной системой, является набором сегментов.

> ***Сегмент*** *— это фрагмент оперативной памяти, выделяемый операционной системой процессу.*

Размер и конкретное расположение сегментов процесса операционная система выбирает, исходя из следующих параметров:

- размер машинного кода программы;
- размер её данных;
- оптимизация распределения памяти данного процесса с точки зрения других запущенных процессов, а также, возможно, процессов, которые могут быть запущены в ближайшем будущем.

Рассмотрим классическую (упрощённую) схему памяти процесса, когда эта память состоит из сегмента кода, сегмента данных и сегмента стека. Адреса внутри сегментов формируются как смещения относительно начала сегмента. Код программы знает о сегментах и оперирует этими смещениями. Например, в операции сложения вида регистр/память адрес в памяти для второго операнда указывается в виде смещения от начала сегмента данных (напомним, что именно там хранятся все переменные программы). Таким образом, реальный физический адрес, по которому должен обратиться процессор, вычисляется уже внутри процессора, а последний, в свою очередь, знает физические адреса начала каждого сегмента. Итак, мы подошли к понятию виртуального адреса и адресного преобразования.

> *Адрес, с которым непосредственно работает программа, называется* ***виртуальным адресом (Virtual Address)****.*

> ***Адресное преобразование (Address Conversion)*** *— это механизм вычисления физического адреса по виртуальному.*

Отметим, что сегментная организация памяти процесса является не единственным источником виртуальных адресов.

Рассмотрим теперь подробнее сегменты процесса.

Сегмент кода хранит в оперативной памяти машинный код программы.

Сегмент данных хранит глобальные переменные программы (так называемые *статические данные*), а также используется для *кучи* (Heap) в том случае, если в программе используются *динамические данные*. Размер всего сегмента данных известен во время запуска программы, но память для кучи выделяется в нём по мере исполнения программы, поскольку на момент запуска непонятно, сколько программе понадобится динамической памяти — это зависит от конкретного сценария исполнения программы, поскольку динамическая память запрашивается с уровня прикладной программы и её использование организует сам программист — автор программы. Соответственно, куча является «динамическим пузырём» нефиксированного размера, «раздувающимся» по мере запросов программы.

В программах на языке C для того, чтобы запросить динамическую память, программист использует функцию `malloc`, а в C++ — оператор `new`. Поскольку динамическая память выделяется в «ручном» режиме, то и освобождаться она должна также «вручную», с помощью специальных функций (функция `free` в C, оператор `delete` в C++). Механизм динамической памяти требует от программиста внимательности и организованности, но взамен предоставляет гибкость и возможность создавать более эффективные программы. Однако при использовании этого механизма программист может допустить (и допускает!) сложно обнаруживаемые ошибки. Это стало причиной того, что в современных промышленных языках, таких как Java, C#, Python, использование динамической памяти доступно только через вы-

деление объектов, а не массивов байтов, как в C, а освобождение — посредством автоматической сборки «мусора», освобождающей память объектов, на которые больше нет ссылок из программы.

Сегмент стека используется процессом для организации вызовов процедур (методов) программы. Он реализует структуру данных под названием *стек*.

> *Набор небольших однотипных фрагментов памяти, объединённых в список, в котором новый элемент всегда добавляется только в конец списка и удаляться из этого списка может также с конца, называется* ***стеком****.*

В тот момент, когда в программе выполняется вызов очередной процедуры, в сегмент стека помещаются следующие данные:

- фактические параметры этой процедуры;
- *адрес возврата* из этой процедуры, то есть адрес инструкции в программе, следующей за вызовом процедуры;
- локальные переменные процедуры и дополнительная служебная информация.

Совокупность перечисленных данных образует так называемый *кадр стека (Stack frame)*. Очевидно, что после завершения процедуры и продолжения выполнения программы в том контексте, откуда процедура была вызвана, ни ее локальные данные, ни параметры больше не нужны. Поэтому кадр завершённой процедуры удаляется со стека, освобождая место для других данных. Если процедура вызывается рекурсивно, то её кадры с различными значениями параметров, данных, адреса возврата размещаются на стеке столько раз, сколько раз она была вызвана. Если программа зацикливается на рекурсивном вызове, то она помещает на стек всё больше и больше данных, заполняя всю допустимую для стека память, и после этого аварийно завершается с ошибкой «переполнение стека» (Stack Overflow). Эта ошибка настолько известна и распространена, что в честь неё названа самая популярная платформа вопросов-ответов по программированию Stack Overflow (https://stackoverflow.com/).

Сделаем следующую ремарку. Программа может быть многопоточной, то есть внутри одного процесса может работать несколько параллельных потоков. На физическом уровне эти потоки обычно размещаются на разных ядрах многоядерного процессора. При этом разные потоки внутри одного процесса имеют разные сегменты стека, поскольку стек используется для поддержки вызовов и работы процедур, а в разных потоках одновременно могут быть запущены разные процедуры. Но сегменты кода и сегменты данных у потоков одной программы являются общими, поскольку в сегменте кода хранится весь машинный код программы и все потоки работают (могут работать) со всеми данными программы.

Виртуальная память и страничная адресация

Ранние ЭВМ использовались преимущественно для научных расчётов. Они имели небольшую оперативную память, но при этом требовалось, чтобы они обрабатывали данные, объем которых существенно превышал размер оперативной памяти, а также умели исполнять программы, объем кода которых превосходил объём оперативной памяти. Для этого использовался специальный приём, позволяющий загружать фрагменты данных и фрагменты машинного кода в оперативную память порциями, по мере необходимости. Это сильно усложняло код прикладных программ — программисту приходилось самостоятельно реализовывать эту функциональность. Позже создатели ЭВМ второго и третьего поколений выработали общие подходы к решению этой задачи и вынесли её решение в операционную систему. Так появилась виртуальная память.

> ***Виртуальная память (Virtual Memory)*** *— это механизм работы программы/процесса с адресным пространством, превосходящим по размеру оперативную память ЭВМ или тот её фрагмент, который выделен данной программе/процессу.*

При этом программа/процесс может работать с фрагментами разнородной физической памяти (включая жесткий диск), но адресное преобразование скрывает эту неоднородность, представляя память для программы как единый, сплошной фрагмент и обеспечивая тем самым непрерывное адресное пространство.

Адресное пространство, доступное процессу, делится на *виртуальные страницы*. Для процессоров семейства Intel x86, работающих в 32-разрядном режиме, размер одной страницы составляет 4 Kb. Эти же процессоры, работающие в 64-разрядном режиме, поддерживают настраиваемый размер страниц от 4 Kb до 1 Mb. При этом оперативная память делится на *физические страницы*, размер которых совпадает с размером виртуальных страниц.

Операционная система выполняет *страничное адресное преобразование* для того, чтобы связывать каждую виртуальную страницу с реальной памятью:

- либо с физической страницей в оперативной памяти;
- либо с областью подкачки на жёстком диске (в случае Windows это файл, в Unix-подобных ОС — раздел жёсткого диска).

Возможно, что виртуальная страница не связана ни с каким реальным (физическим) фрагментом памяти. Это означает, что эта страница не инициализирована, то есть к ней ещё не было обращений из программы. Также страница может перейти в это состояние, если программе больше не нужны находящиеся в ней данные и она явно «вернула» эту страницу операционной системе.

Соответствие между страницами виртуальной памяти и физической памятью хранится в специальной структуре данных, которая называется *таблицей страниц* и располагается в оперативной памяти. Таблица страниц для всего адресного пространства получилась бы недопустимо большой, если хранить её в виде одного массива. Например, легко посчитать, что для адресного пространства 32-битного процессора Intel 80386 таблица страниц заняла бы до 4 Mb, в то время как общий объём оперативной памяти компьютеров на базе этого процессора обычно был равен 2 или 4 Mb. Поэтому в 32-разрядном режиме Intel x86 таблица страниц хранится в виде двухуровневой древовидной структуры, а в 64-разрядном режиме Intel x86_64 — в виде четырёхуровневой. Эти структуры заполняются постепенно, по мере использования программой виртуальных страниц.

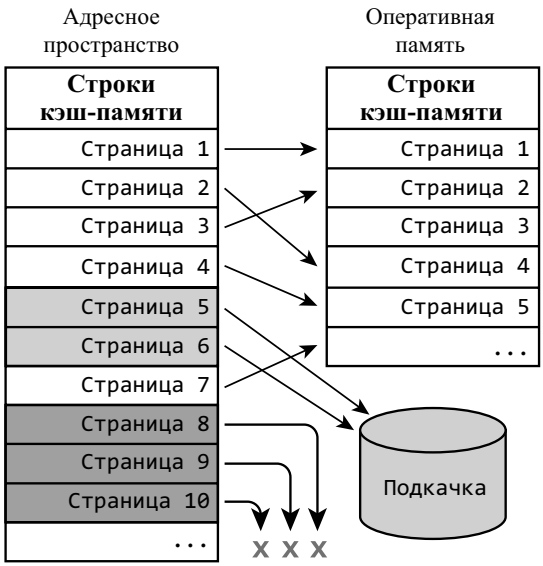


Рис. 23. Страничная адресация и виртуальная память

На рис. 23 представлен механизм страничной виртуальной памяти. Отметим, что операционная система может для каждого процесса также создать отдельную таблицу страниц, исключив, таким образом, оперативную память других процессов из его адресного пространства. Это позволяет усилить механизм изоляции процессов.

В семействе процессоров Intel x86, начиная с Intel 80386, сегментная и страничная адресация могут использоваться одновременно. Вначале процессор делает сегментное преобразование, затем над получившимся адресом он выполняет страничное преобразование, результатом которого уже является физический адрес.

Производители операционных систем для современных процессоров (например, Intel x86_64, ARM v7) постепенно отказываются от использования сегментной адресации, довольствуясь возможностями страничной. Применение одновременно сегментного и страничного адресных преобразований, как в Intel 80386, избыточно: виртуальная память строится только на основе страничного адресного преобразования. Процессоры семейства Intel x86 в 64-битном режиме не используют сегменты, поддерживая сегментную адресацию лишь для совместимости со старыми программами. Многие архитектуры, отличные от Intel x86, сегментную адресацию не поддерживают вовсе.

Выделение и замещение виртуальных страниц

До тех пор, пока программа не обратилась к памяти, виртуальные страницы её процесса не проинициализированы. При первом обращении программы к очередной странице процессор приостанавливает работу программы и передаёт управление операционной системе. Последняя выделяет для виртуальной страницы физическую страницу в оперативной памяти и обновляет таблицу страниц. Если свободных физических страниц нет, то операционная система выгружает данные из некоторой занятой физической страницы в область подкачки на жестком диске и выделяет процессу освободившуюся страницу.

Для выбора выгружаемой страницы используются *стратегии вытеснения*, аналогичные стратегиям вытеснения для кэш-памяти. Вытесненная виртуальная страница помечается как выгруженная, и при последующем обращении к ней процессор также передаёт управление операционной системе, чтобы она могла вновь выделить для этой страницы физическую страницу. Такой порядок работы виртуальной памяти позволяет процессу использовать большое количество памяти, но реальную физическую память получать лишь по мере обращения к фрагментам этой памяти. При запуске процесс может запросить у операционной системы много памяти, но пока он не начал с ней работать, физическая память не будет расходоваться.

Читатель может легко убедиться в том, что современные операционные системы выделяют память описанным образом, написав следующую простую программу на C. Данная программа динамически, с помощью функции `malloc`, запрашивает большой объем памяти (например, 1 Gb), но обращается к этой памяти не сразу, а по прошествии некоторого времени. Если проследить за запущенной программой (в случае Windows для этого достаточно диспетчера процессов), то станет очевидно, что реальная физическая память выделяется не в момент вызова `malloc`, а по мере обращения к памяти из программы.

Вопросы

1. Что такое физический адрес?
2. Что такое прямая адресация?
3. Что такое адресное пространство?

4. Для чего служит сегментная организация памяти процесса?
5. Что такое виртуальный адрес?
6. Кто выделяет оперативную память процессу?
7. Какие виды сегментов процесса вы знаете?
8. Что такое статическая память (статические данные) программы?
9. Что такое динамическая память (динамические данные) программы?
10. В чём разница при работе с динамической памятью в языке C и Java?
11. Опишите особенности сегментной организации оперативной памяти процесса в случае многопоточной программы.
12. Для чего используется сегмент стека?
13. Что находится в кадре стека?
14. Опишите алгоритм работы сегментного адресного преобразования.
15. Каковы исторические предпосылки возникновения виртуальной памяти?
16. Что такое виртуальная память?
17. Опишите алгоритм работы страничного адресного преобразования.
18. В каких состояниях может находиться страница адресного пространства?
19. Опишите механизм работы с виртуальной страницей, строго обозначив различные роли в этом механизме (то есть что делает процессор, а что — операционная система).
20. Каким образом сегментная и страничная адресаций позволяют изолировать процессы?
21. Реализуйте программу на языке C, пользуясь рекомендациями в конце данной лекции, чтобы увидеть действие стратегии вытеснения.

Литература

1. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
2. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
3. *Таненбаум Э., Вудхалл А.* Операционные системы. Разработка и реализация. 3-е издание. СПб.: Питер, 2015.
4. *Silberschatz A., Gagne G., Galvin P. B.* Operating System Concepts, 10th edition. Wiley, 2018. 951 p.

Лекция 13. Введение в операционные системы

Назначение операционной системы; основные функции операционных систем: управление устройствами, изоляция процессов, планировщик задач, виртуальная память, разграничение прав доступа пользователей, файловая система, сетевая поддержка, интерфейс пользователя, управление питанием.

Назначение операционной системы

Современный компьютер включает в себя множество различных устройств — процессор, память, жесткий диск и пр. К компьютеру также могут подключаться внешние (периферийные) устройства, например, принтер или сканер. Компьютер должен уметь управлять этими устройствами для того, чтобы прикладные программы, которые на неё работают, получили возможность пользоваться сервисами этих устройств — например, чтобы можно было из MS Word печатать документы на принтере.

Управление внешними и внутренними устройствами осуществляют специальные служебные программы. Эти программы достаточно сложны, поскольку взаимодействуют с устройствами через низкоуровневые интерфейсы. Более того, для каждого устройства существует большой ряд аналогов — например, количество моделей принтеров от различных производителей измеряется сотнями. И каждая модель имеет свои собственные особенности, отражаемые в её программном интерфейсе. Однако важно, чтобы прикладное программное обеспечение не зависело от того, какое конкретное устройство подключено или установлено на данном компьютере, — то есть функция печати в Microsoft Word должна одинаково работать независимо от вида принтера, подключённого к данному компьютеру. Управление устройствами и сокрытие от прикладных программ особенностей различных моделей этих устройств выполняет *операционная система*.

> ***Операционная система (ОС, Operating System)*** *— это служебное ПО, которое позволяет прикладными программам, работающим на компьютере, эффективно использовать как внутренние устройства компьютера — центральный процессор, оперативную память, жесткий диск, средства сетевой поддержки и пр., так и внешние — например, монитор, клавиатуру, принтер.*

Идея независимости прикладного программного обеспечения от оборудования компьютера появилась давно, как минимум в начале 1950-х годов. Однако понятие операционной системы вошло в широкий обиход в эпоху ЭВМ второго поколения, на рубеже 1950-х и 1960-х годов. Начиная с этого времени операционные системы прошли долгий путь развития. К настоящему моменту создано большое количество разнообразных операционных систем. Можно упомянуть различные версии Windows, используемые преимущественно на офисных компьютерах и ноутбуках, ОС Linux, которая эффективно используется на серверах и в телекоммуникациях, и ОС Android, работающую на мобильных телефонах и планшетах.

Основные функции операционных систем

Перейдём теперь к детальному рассмотрению основных функций операционных систем. Для начала перечислим их.

- Управление устройствами (Device Management).
- Многозадачность (Multitasking).
- Управление виртуальной памятью (Virtual Memory Management).
- Разграничение прав доступа пользователей (Access Rights).
- Файловая система (File System).
- Сетевая поддержка (Network support).
- Интерфейс пользователя (User Interface).
- Управление питанием (Power Management).

Управление устройствами. Исторически эта функция является основной для операционной системы.

> *Различные устройства компьютера взаимодействуют с прикладным ПО, работающим на этом компьютере, при помощи специальных программ — **драйверов (Drivers)**; последние являются частью операционной системы.*

Операционная система включает в себя большое количество драйверов для наиболее популярных моделей устройств, однако не может покрыть всех возможных вариантов. Вот почему после покупки, например, нового принтера, вам может понадобиться найти в Интернете и установить на вашем компьютере соответствующий драйвер.

Многозадачность. На современном компьютере одномоментно может исполняться более одного приложения. Поддержка этой возможности называется многозадачностью. Для обеспечения многозадачности необходимо, чтобы процессор компьютера выполнял программы в соответствии с некоторым расписанием — сначала одну программу, потом другую, потом третью и т. д., возвращаясь через некоторое время к первой, и так по кругу. Это

подход, предназначенный изначально для одноядерных процессоров, называется *псевдопараллелизмом*, поскольку он подразумевает не параллельную работу нескольких программ на одном процессоре, а выполнение программ последовательно, но с частым переключением. Псевдопараллелизм используется и для многоядерных процессоров, поскольку в них одновременно работает больше программ, чем имеется ядер у процессора.

Существует много стратегий реализации псевдопараллелизма. В операционной системе Windows запущенные в данный момент на компьютере программы (процессы) можно увидеть в Task Manager. Для того чтобы минимизировать последствия влияния ошибок в одной программе на другие программы, запущенные на том же компьютере, ОС запускает каждый процесс в изолированном окружении. Это означает, что запущенные процессы не имеют доступа к данным друг друга. Один из основных механизмов такой изоляции — режим адресации, при котором процесс может адресовать лишь свои данные, и по одинаковым адресам, обозначенным в соответствующих программах, у разных процессов находятся разные данные, у каждого — свои.

Управление виртуальной памятью. Запущенные на компьютере программы могут потребовать больше оперативной памяти, чем имеется в наличии или чем ОС может выделить для данной программы. В такой ситуации ОС использует виртуальную память, организуя для программы с использованием жесткого диска память бóльшего объема. Эта иллюзия для прикладной программы достоверна — последняя не вникает в детали распределения памяти, просто используя всю ту память, которую ей выделяет ОС, хотя, безусловно, активное превышение реальных, физических лимитов (то есть интенсивное использование жесткого диска через механизм виртуальной памяти) сказывается на производительности программы.

Файловая система. Жёсткий диск компьютера является сложным устройством, но фактически он хранит лишь массив байтов. Однако непосредственная работа с массивом байт крайне неудобна как для конечного пользователя, так и для прикладных программ. В связи с этим ОС поддерживают абстракцию файла.

> ***Файл (File)*** *— это фрагмент памяти на жёстком диске, который имеет имя, тип, дату создания и прочую служебную информацию. Файл является элементом хранения информации на жёстком диске. Большинство подсистем операционной системы, прикладные программы и пользователи имеют дело именно с файлами.*

Разные виды файлов — текстовые, аудио/видео, docx и т. д. — операционная система и прикладные программы обрабатывают по-разному.

Для операционной системы особенно важными являются исполняемые файлы. Формат исполняемых файлов в разных операционных системах разный. Для ОС Windows основными исполняемыми файлами являются:

- exe-файлы (имеют расширение exe) — самостоятельные программы на машинном языке, которые Windows загружает с жёсткого диска в оперативную память и передаёт на исполнение процессору;
- dll-файлы (имеют расширение dll) подобно exe-файлам содержат код на машинном языке и точно также исполняются, но не являются самостоятельными программами: для реализации модульности программ, позволяющей компоновать код в разные исполняемые файлы (для эффективного выполнения, а также, во многом, для удобств разработки).

Файловая система поддерживает функции сохранения файлов на жестком диске и доступа к ним — как для программ, так и для непосредственных пользователей компьютера. Она же поддерживает иерархию папок. Для операционной системы Windows общеупотребимыми файловыми системами являются FAT и NTFS.

- Исторически FAT (File Allocation Table) была первой файловой системой для IBM PC и предназначалась для работы с гибкими и жёсткими дисками. В настоящее время эта файловая система вытесняется более современными файловыми, но для совместимости со старыми ОС она до сих пор часто используется на съёмных накопителях — «флэшках» и внешних жёстких дисках. Можно столкнуться со следующим ограничением FAT: эта файловая система не позволяет работать с файлами, имеющими размер больше, чем 4 Gb, а также она не поддерживает разграничение прав доступа. Основанная на FAT более современная файловая система exFAT позволяет работать с файлами размером более 4 Gb.
- NTFS (New Technology File System) появилась в Windows NT и используется сейчас в операционных системах семейства Windows в качестве основной. NTFS поддерживает разграничение прав доступа к файлам и папкам, а также сжатие и шифрование файлов. Файловая система NTFS может также восстанавливаться после сбоев. Например, если из папки был удалён файл, но из-за внезапного отключения электропитания освободившееся на жестком диске место не было помечено как свободное, то при возобновлении питания невозможно найти этот файл или использовать соответствующее место. NTFS умеет корректно решать эту и подобные ей проблемы.

Разграничение прав доступа пользователей. В случае, когда одним компьютером могут пользоваться разные люди, требуется обеспечить разграничение доступа к ресурсам и данным этого компьютера — файлам и каталогам, теми или иными приложениями (программами), установленными на компьютере, принтерами и т. д. Часть этих возможностей может быть доступна всем пользователям данного компьютера, но некоторой частью может пользоваться только администратор компьютера и те пользователи, которых он выберет. Эта

задача является актуальной и для локальных сетей, администрирование которых также осуществляется средствами операционной системы.

Для решения задачи разграничения прав доступа каждому пользователю компьютера создаётся *учётная запись (Account)*. Помимо учётных записей для «настоящих» пользователей (людей), в ОС часто имеется несколько служебных учётных записей, «от лица» которых запускаются служебные программы — средства резервного копирования данных, веб-сервера, сервера баз данных и т. д. Защита этих данных обеспечивается механизмом разграничения прав доступа: обычный пользователь не имеет прав для изменения этих данных, поэтому он не может их умышленно или случайно изменить. Например, обычный пользователь Windows не может непосредственно через Проводник записывать и удалять файлы в системных папках, но может делать это только через инсталляцию или деинсталляцию новых приложений, то есть посредством специальных пакетных процедур. В момент работы таких процедур как раз и активизируются служебные учетные записи.

Сетевая поддержка обеспечивает возможность подключения компьютера к локальным и глобальным сетям связи (сегодня это в основном Internet, для мобильных ОС — мобильные сети). Сетевая работа ведётся с использованием сложной многоуровневой системы сетевых протоколов, реализующих передачу пакетов и отслеживание их целостности, обработку различных сетевых ошибок и т. д. Программная реализация этих протоколов строится на основе модели *сетевого стека (Network Stack)* и имеет многоуровневую архитектуру. Подразумевается, что со стороны сети аналогичные стеки реализуются ассамблеями различных сетевых устройств — роутеров, файерволов, концентраторов и пр.

Интерфейс пользователя — это подсистема ОС, предназначенная для взаимодействия с конечным пользователем, которая предоставляет ему возможность использовать различные ресурсы компьютера. Настольные и мобильные ОС (Windows, Mac OS X, настольные версии Linux, iOS, Android) предоставляют пользователям графический интерфейс, используя метафору Рабочего стола: компьютер имитирует на экране различные «офисные» предметы — папки, картотеки, документы и т. д. Значительное внимание в последние десятилетия уделяется работе с компьютером людей с ограниченными возможностями. Так, например, многие современные настольные ОС поддерживают голосовой ввод и озвучку текста на экране. Для серверных ОС традиционным интерфейсом пользователя является *текстовая консоль* — подсистема ОС, позволяющая вводить с клавиатуры данные и команды и отображать результаты на мониторе в текстовом виде[9]. Консоль

[9] Исторически термин «консоль» обозначал устройство, объединяющее функции ввода и вывода данных и управляющих команд компьютера — например, комплект из клавиатуры и монитора. До конца 1970-х гг. устройства ввода и вывода зачастую выпускались в общем корпусе, что и послужило причиной возникновения термина. До широкого распространения мониторов в качестве консолей использовались телетайпы (клавиатура и принтер, которые могли подключаться к линиям связи).

обеспечивает диалог пользователя с командной оболочкой ОС, работающей на сервере, позволяя управлять СУБД, веб-серверами и т. д. Также консоль позволяет запускать и интерактивные программы — например, текстовые редакторы для редактирования системных настроек.

Система управления питанием особенно актуальна для мобильных ОС, поскольку ноутбуки, смартфоны и планшеты работают на аккумуляторах. В этих устройствах важно обеспечить оптимальный расход питания, чтобы компьютер мог работать максимально долго без подсоединения к электрической сети. Также важна поддержка контроля цикла зарядки/разрядки аккумуляторов для продления их срока службы.

Вопросы

1. Объясните назначение операционной системы.
2. Дайте определение понятию операционной системы.
3. Перечислите основные функции операционной системы.
4. Что такое драйвер устройства, и зачем он нужен?
5. Что такое псевдопараллелизм?
6. Что такое процесс, и как процесс соотносится с программой?
7. Зачем требуется разграничение прав доступа пользователей?
8. Что такое системная учётная запись?
9. Что такое файл?
10. Какие функции выполняет файловая система?
11. Перечислите известные вам файловые системы.
12. Расскажите о свойствах файловой системы NTFS.

Литература

1. *Таненбаум Э., Бос Х.* Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
2. *Цикритзис Д., Бернстайн Ф.* Операционные системы. Пер. с англ. М.: Издательство «Мир», 1977. 336 с.
3. *Silberschatz A., Gagne G., Galvin P. B.* Operating System Concepts, 10th ed. Wiley, 2018. 951 p.
4. *Stallings W.* Operating systems: internals and design principles. Pearson Education Limited, 2018.
5. *Walden D., Van Vleck T.* Compatible Time-Sharing System (1961–1973): Fiftieth Annversary Commemorative Overview. IEEE Computer Society, 2011.

Лекция 14. Обзор известных операционных систем

Операционные системы семейства мейнфреймов IBM; Unix — переносимая ОС; Linux — свободно распространяемая (Open Source) ОС; DOS и Windows — ОС для компьютеров семейства IBM PC; Android — самая популярная мобильная ОС.

Современные компьютеры активно развиваются, соответственно, меняются и операционные системы, так как они непосредственно связаны с аппаратурой и должны её эффективно обсуживать. При этом и компьютерные архитектуры, и соответствующие им операционные системы образуют семейства. Например, ОС Windows предназначаются для компьютеров Intel x86.

Кроме того, ОС для новых компьютеров зачастую создаются на основе уже существующих: например, широко распространенная ОС для мобильных устройств Android была создана на основе Linux.

В этой лекции мы рассмотрим ряд известных семейств операционных систем, частью из прежних времён и ставших уже историей, частью современных и активно развивающихся:

- семейство операционных систем для менфреймов компании IBM;
- семейство переносимых операционных систем Unix;
- семейство свободно распространяемых операционных систем Linux;
- семейство операционных систем DOS для компьютеров IBM PC;
- семейство операционных систем Windows для процессоров семейства Intel x86;
- семейство мобильных операционных систем Android.

Операционные системы семейства мэйнфреймов IBM

Мейнфрейм (Mainframe) — это высокопроизводительный сервер, ориентированный на обработку большого количества данных и запросов, то есть для удовлетворения потребностей крупного бизнеса. Мейнфрейм не следует путать с суперкомпьютером, который ориентирован на высокопроизводительные пакетные вычисления (инженерные и научные задачи). История

мэйнфреймов началась в конце 1960-х годов с выпуска компанией IBM известного компьютера System/360. Мейнфреймы выпускались и другими производителями — например, Burroughs и General Electric. Следует отметить, что мейнфреймы используются и выпускаются до сих пор.

Мэйнфреймы компании IBM 1960—1980-х годов работали с большим количеством терминалов (рабочих мест): их число достигало нескольких десятков тысяч. Основными покупателями таких компьютеров были различные учреждения — от университетов и исследовательских институтов до крупных компаний и банков. Отличительной особенностью этих компьютеров была высокая гибкость при комплектации: например, покупатель, приобретая такой компьютер с небольшим количеством оперативной памяти и небыстрым процессором, мог рассчитывать со временем, когда его потребности возрастут, приобрести более мощную модель, которая будет совместима с уже купленной. Совместимость подразумевала, что на новом компьютере будут работать те же программы, что и на старом, и это в значительной степени гарантировалось операционной системой. Подобная гибкость обеспечила семейству этих компьютеров долгую жизнь: разные модели компьютеров, выпускаемые на протяжении более чем 40 лет, могли исполнять программы, написанные в 1960-х годах.

Операционные системы этого семейства, начиная с OS/360, позволяли прикладному обеспечению абстрагироваться от специфики отдельных аппаратных узлов — для программ было неважно, читают ли они данные с перфокарт в 1960-х годах или с жесткого диска в 2000-х. Операционные системы поддерживали также файловые системы, реализовывали многозадачность и виртуальную память, поддерживали работу виртуальных машин. Последнее позволяло заменить одним мощным компьютером несколько компьютеров меньшей мощности, что было удобно и выгодно для крупных вычислительных центров.

Интересно отметить, что программы, созданные для мейнфреймов IBM, пережили эти компьютеры и продолжают успешно функционировать до сих пор во многих крупных учреждениях США и Европы. За годы эксплуатации и сопровождения эти программы «вросли» в бизнес-процессы соответствующих учреждений, и последние крайне неохотно соглашаются на замену их новым ПО. Фактически на рынке разработки ПО победила следующая модель: операционные системы и прочее программное окружение, в рамках которых работают эти программы, эмулируются для возможности исполнения на современных серверах. Противопоставить этому подходу можно движение по автоматизированному реинжинирингу устаревшего ПО (то есть переписывание старых программ на новых языках программирования), которое активно развивалось в конце 1990-х — начале 2000-х годов, но в итоге так и не смогло завоевать рынок.

Переносимые ОС семейства Unix

Другим знаменитым семейством операционных систем является Unix. Исходная версия Unix была разработана в конце 1960-х годов в компании

AT&T. В дальнейшем было создано огромное число различных версий Unix, а также значительное число Unix-подобных ОС. Ниже перечислены новые возможности ОС Unix, обеспечившие ее популярность.

- Иерархическая файловая система с деревом каталогов. У прочих ОС той эпохи данные на жестком диске хранились в формате, напоминавшем таблицы баз данных, что обеспечивало высокую производительность доступа к данным, но не обладало гибкостью — такой формат был фиксирован для отдельной ОС, а файлы, которыми оперировал Unix, имели разные форматы.
- Наличие набора удобных и простых системных программ для реализации базовых функций, часто используемых в прикладных программах: сортировки данных, поиска и преобразования текста, сжатия данных и т. д.
- Удобная интерактивная командная оболочка, которая позволяла комбинировать программы в цепочки, например, отправляя вывод одной программы на вход другой или замещая ввод с консоли вводом из файла и т. д. Такая командная оболочка позволяла создавать ансамбли из взаимодействующих программ, разработанных на разных языках, не вдаваясь в детали их реализации. Аналогичные возможности хоть и предоставлялись ранее (например, в OS/360), но в Unix удалось реализовать их гораздо удобнее для конечного пользователя.

Остановимся детально на последнем свойстве — переносимости. Как и другие ОС того времени, Unix была реализована на языке Ассемблер. Но к середине 1970-х годов она была переписана на языке C. Использование языка C позволило быстро создавать новые версии Unix для различных компьютерных архитектур. При очередном переносе требовалось реализовать новый C-компилятор, что не является сложной задачей ввиду простоты данного языка. Также требовалось переписать небольшую часть специализированного кода Unix, который отвечал за работу с конкретным семейством процессоров. Компания AT&T открыла исходные коды Unix для научно-исследовательских учреждений, университетов и государственных учреждений. В результате было создано много версий этой Unix для разных компьютерных архитектур.

Основные подсистемы ОС Unix перечислены ниже.

- Ядро, которое реализовывало основные функции и включало в себя минимальный базовый набор драйверов аппаратуры.
- Набор утилит — программ, которые выполняли сервисные функции, например, поиск файлов и сжатие данных.
- Набор программных библиотек и компиляторов, например, компилятор языка C и его стандартные библиотеки.

В заключении отметим, что именно переносимость (использование языка C для реализации и открытость исходных кодов) способствовала широкому распространению Unix. Однако несмотря на то, что компания AT&T обеспечила открытые лицензии для университетов и госучреждений, ОС Unix не была свободно распространяемой и нюансы её использования становились предметом судебных разбирательств.

Семейство свободно распространяемых операционных систем Linux

В 1980-х годах Ричардом Столлманом был запущен проект GNU для создания свободной Unix-подобной ОС. В результате были реализованы стандартные для ОС Unix утилиты, а также создано семейство компиляторов GCC (GNU Compiler Collection). Разрабатываемое в рамках этого проекта ядро ОС GNU Hurd оказалось менее удачным, чем реализованное независимо Линусом Торвальдсом ядро Linux. В итоге популярность приобрёл «тандем», состоящий из ядра Linux и набора утилит GNU, за которым закрепилось именно название ядра — Linux.

Первая версия Linux появилась в 1991 году. Она была совместима с Unix (программы для Unix работают под её управлением) и распространяется под свободной лицензией GPL. Почти за 30 лет своего существования ОС Linux стала очень популярной для серверов, а также для встраиваемого и сетевого оборудования. Перечислим причины такой популярности.

- Возможность бесплатно использовать ОС Linux — лицензия GPL, под которой распространяется Linux, предоставляет существенно больше свободы, чем лицензия Unix.
- Возможность самостоятельно комплектовать ОС, то есть создавать различные *дистрибутивы*, освобождённые от функциональности, ненужной в данном контексте; кроме того, Linux можно дорабатывать для собственных нужд, что также предусмотрено его архитектурой.
- Linux имеет скромные требования к оборудованию — производительности процессора, объёму оперативной памяти и жестких дисков.

Для серверов использование ОС, обладающей такими свойствами, является существенным преимуществом. Ведь обычно серверу не нужны ни мощный драйвер графического адаптера, ни «настольные» программы наподобие веб-браузера. Зато ему могут понадобиться специфические системы хранения данных и, соответственно, особые драйверы для них. Последние, в случае Linux, можно встроить прямо в ядро ОС, что повышает производительность. Возможность создавать свои дистрибутивы помогает оптимизировать компьютеры под управлением Linux для различных задач — научных вычислений, специфичных веб-серверов, СУБД и т. д. Скромные системные требования позволяют эффективно размещать Linux на виртуаль-

ных машинах, а современный серверный компьютер может одновременно управлять несколькими виртуальными машинами.

Для сетевого и встраиваемого оборудования ситуация в целом такая же, как и для серверов, но ещё более выраженная. Рассмотрим простой сетевой роутер — «коробочку над дверью», к которой подключено несколько Ethernet-кабелей и которая не соединена ни с монитором, ни с клавиатурой, ни с мышью. Всё, что требуется от роутера, — пересылать данные по сети и обеспечивать возможность дистанционной настройки. Установленная на него ОС Linux, в которой оставлено лишь все необходимое, может занимать на роутере всего несколько Mb. В итоге роутер можно сделать недорогим и обеспечить оптимальное использование его скромных ресурсов «по прямому назначению».

Linux распространяется под лицензией GPL (GNU Public License). Скажем несколько слов об этой лицензии. GPL — это специальный вид лицензий, предназначенный для открытого ПО (Open Source Software), в рамках которой авторы могут свободно использовать исходный код, имеющий эту лицензию, но должны также открыть и свой собственный код. Помимо ОС Linux под лицензией GPL распространяются многие популярные программные продукты, такие как графический редактор GIMP, СУБД MySQL, набор компиляторов GCC (GNU Compiler Collection), система управления версиями Git, аудиоредактор Audacity. Все перечисленные продукты могут работать под управлением различных ОС: Windows, Linux, других Unix-подобных.

Linux является бесплатным и основывается на Open Source идеологии, закреплённой лицензией GPL. Linux развивается большим сообществом разработчиков. Примечательно, что заметную часть работы по этим открытым проектам сейчас берут на себя крупные IT-компании, такие как Microsoft и IBM, поскольку многие предлагаемые ими коммерческие решения также основываются на Linux и проекте GNU.

Создание на основе Linux корпоративных IT-решений и применение Linux для управления специализированным оборудованием (навигационным оборудованием, системами связи и т. д.), а также использование ОС Linux для нужд военно-промышленного комплекса влекут за собой повышенные требования к надёжности. Одним из способов обеспечения таких требований является *верификация ПО* — комплекс мер по анализу ПО на предмет соответствия спецификациям и отсутствия ошибок. Верификация требует решения сложных наукоёмких задач, вокруг которых сосредоточено внимание многих исследовательских коллективов в разных странах.

Весомый вклад в обеспечение качества Linux внесли российские учёные — сотрудники Института системного программирования им. В. П. Иванникова РАН (ИСП РАН, г. Москва). При поддержке консорциума The Linux Foundation и Министерства науки и высшего образования РФ на базе ИСП РАН в 2009 году был создан центр Linux Driver Verification, в рамках которого в ядре и драйверах ОС Linux были обнаружены и исправлены сотни ошибок.

Семейство ОС DOS

В начале 1980-х годов появились персональные компьютеры семейства IBM PC. Для них была создана операционная система DOS (Disk Operating System) — дисковая операционная система, основной задачей которой была поддержка работы с дисками — гибкими (дискетами) и жесткими (винчестерами). Эти диски пришли на смену перфокартам, магнитным лентам и прочим прежним средствам хранения и ввода данных. Программная работа с дисками, будучи трудоемкой и в то же время типовой задачей, была вынесена на уровень ОС. Помимо этого, операционные системы семейства DOS предоставляли прикладным программам сервисы по выделению памяти, а также информацию о системных событиях. Примерами таких событий могли быть вставка дискеты в дисковод, нажатие пользователем кнопки мыши и т. д.

Операционная система DOS была очень простой: она не поддерживала многозадачности, не разграничивала доступ пользователей к данным, не препятствовала доступу программ к системным данным. Но вместе с тем она была недорогой в разработке и поддержке, нетребовательна к аппаратным и системным ресурсам — например, могла работать на компьютерах с оперативной памятью меньше чем 1 Mb. Таким образом, эта операционная система хорошо подходила для первых IBM PC, которые и сами позиционировались как недорогие персональные компьютеры. Первые версии DOS были приобретены, а затем совместно разрабатывались, компаниями IBM и Microsoft. В дальнейшем разработку этих ОС продолжила компания Microsoft, выпустив в период с 1981 по 1995 год 7 версий MS DOS.

Семейство ОС Windows

Рассмотрим теперь семейство операционных систем Windows. По мере роста популярности IBM PC и других аналогичных компьютеров актуальной стала задача создания в ОС многофункционального и унифицированного пользовательского интерфейса, а также интерфейса для прикладных программ.

В 1985 году компания Microsoft выпустила первую версию Windows — графическую оболочку для IBM PC, запускаемую над MS DOS. В первой половине 1980-х годов графический интерфейс пользователя был прогрессивной инновацией. В начале 1980-х годов у компьютеров Apple уже были дружественные пользователю графические оболочки, позаимствовавшие метафору Рабочего стола у разработок Xerox Research, а Microsoft заняла эту нишу на привычном для себя рынке ПО для IBM PC.

Система Windows, помимо функций MS DOS, предоставляла конечному пользователю компьютера удобный графический интерфейс на основе метафоры офисного стола, а программисту — обширный набор системных функций, реализованных в виде хорошо спроектированного программного

интерфейса. Также для Windows был разработан большой набор библиотек, облегчавших разработку ПО с «богатым» оконным интерфейсом.

Несмотря на активное развитие, Windows довольно долго оставалась лишь оболочкой над операционной системой DOS. Но в 1993 году компания Microsoft выпустила Windows NT, которая была уже полноценной операционной системой и не требовала для своей работы DOS. Windows NT поддерживала многозадачность, разграничение доступа пользователей, изоляцию процессов, файловую систему с возможностью разграничения доступа пользователей к данным и другие возможности полноценных операционных систем. Современная операционная система Windows 10 и версии Windows, предназначенные для работы на серверах, являются наследниками Windows NT. В настоящее время операционные системы семейства Windows являются самыми популярными ОС для настольных компьютеров: они установлены более чем на 75% настольных компьютеров во всём мире.

Семейство мобильных ОС Android

Требования к операционным системам для мобильных устройств существенно отличаются от «настольных» и серверных.

- Мобильным устройствам требуется урезанный функционал классических ОС: им не нужна поддержка множества пользователей, не требуются классические системные утилиты наподобие Unix-утилит, не нужна поддержка многочисленных подключенных внешних устройств.
- Мобильные ОС должны быть оптимизированы для выполнения специфических коммуникационных задач, в частности, с учётом неустойчивой сотовой связи.
- Им требуется лишь упрощённая реализация офисных приложений.
- Мобильные ОС должны эффективно поддерживать развлекательные приложения — игры, аудио/видео, фотокамера, шопинг.
- Требуется оптимизация энергопотребления для работы в автономном режиме; при этом современные мобильные устройства зачастую имеют процессоры, лишь незначительно уступающие по мощности настольным.

Первая версия ОС семейства Android была создана усилиями компаний Google, Sony, DELL, Motorola и некоторыми другими во второй половине 2000-х годов на основе Linux. Первые мобильные устройства с Android поступили в продажу в 2008 году. Последняя на момент создания данного курса лекций версия Android 12 вышла осенью 2021 года.

ОС Android максимально эффективно реализует представленные выше требования к мобильным ОС, добавляя к ним функции, перечисленные ниже.

- Изоляция приложений друг от друга: важные данные приложения хранят в отдельных хранилищах, недоступных другим приложениям.
- Шифрование локальных данных пользователя.
- Централизованный сетевой магазин приложений.
- Специальная поддержка фотокамеры для обеспечения достаточно высокого качества фото- и видеосъемки в отсутствие полноценного объектива и большой светочувствительной матрицы.
- Специализированный звуковой драйвер, который обрабатывает отдельно сигналы с нескольких микрофонов для эффективного подавления окружающего шума при разговоре.

Изначально приложения для ОС Android разрабатывались преимущественно на языке Java (с возможным подключением библиотек на C/C++), а основным инструментом для разработки была среда Eclipse. С 2014 года Google предлагает использовать для разработки среду Android Studio, созданную на базе известной среды разработки IntelliJ IDEA компании JetBrains. В 2017 году основным языком разработки для Android-приложений был объявлен язык Kotlin, также разработанный компанией JetBrains. Данная компания была основана в Санкт-Петербурге, и до сих пор Петербургское отделение остаётся базовым отделением компании JetBrains.

Отметим, что многие перечисленные свойства ОС Android также характерны и для других современных мобильных ОС — появившейся несколько раньше iOS и несколько позже — Windows Phone.

Вопросы

1. Перечислите особенности ОС семейства мейнфреймов IBM.
2. Перечислите особенности ОС семейства Unix.
3. Какие особенности ОС Unix послужили причиной её широкого распространения?
4. Опишите структуру Unix.
5. Назовите предпосылки появления ОС Linux.
6. Расскажите про лицензию GPL.
7. Для каких задач используется операционная система Linux?
8. Расскажите историю появления и трансформации операционной системы DOS.
9. Перечислите особенности ОС семейства DOS.
10. Для каких архитектур создавались ОС семейства DOS?
11. Перечислите нововведения, предложенные ОС Windows.
12. Какая версия ОС Windows перестала быть оболочкой над DOS?
13. Расскажите об особых требованиях к ОС для мобильных устройств.
14. Перечислите функции ОС Android.

15. Какая платформа (язык и среда разработки) является основной для программирования под Android исторически? А в настоящее время?
16. Какие кроме Android вы можете назвать другие мобильные ОС?

Литература

1. *Таненбаум Э., Бос Х.* Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
2. *Цикритзис Д., Бернстайн Ф.* Операционные системы. Пер. с англ. М.: Издательство «Мир», 1977. 336 с.
3. *Silberschatz A., Gagne G., Galvin P. B.* Operating System Concepts, 10th ed. Wiley, 2018. 951 p.
4. *Stallings W.* Operating systems: internals and design principles. Pearson Education Limited, 2018.
5. *Walden D., Van Vleck T.* Compatible Time-Sharing System (1961–1973): Fiftieth Anniversary Commemorative Overview. IEEE Computer Society, 2011.
6. *Брукс Ф., Чапель Х.* Мифический человеко-месяц или Как создаются программные системы. Пер. с англ. СПб.: Символ-Плюс, 2010. 304 с.
7. *Робачевский А. М., Немнюгин С. А., Стесик О. Л.* Операционная система UNIX, 2-е изд. СПб.: БХВ-Петербург, 2010. 656 с.

Лекция 15. Высокопроизводительные системы

Примеры вычислительно сложных задач; системы с общей памятью, системы с изолированной памятью; многопоточность и распределённые вычисления; облачные вычисления с использованием виртуальных серверов Amazon EC2 и Microsoft Azure.

Вычислительно сложные задачи

Большинство людей сейчас привыкли пользоваться компьютерами для выполнения производственных и личных канцелярских задач, а также для развлечения. Однако на протяжении всей истории вычислительной техники для решения сложных вычислительных задач были востребованы и продолжают оставаться необходимыми высокопроизводительные ЭВМ. С развитием вычислительных возможностей ЭВМ, а также прогрессом науки и индустрии расширялся и класс этих задач. Перечислим некоторые из таких задач, актуальные на данный момент.

- Моделирование физических процессов в гидро- и аэродинамике, ядерной физике, механике, что позволяет обойтись без дорогостоящих и опасных экспериментов.
- Анализ экспериментальных данных в астрономии, метеорологии, гидро- и аэродинамике.
- Задачи биоинформатики, в частности геномики, которые требуют исполнения алгоритмов на очень больших объёмах входных данных (речь идёт о сотнях гигабайт); и эти алгоритмы имеют существенную вычислительную сложность.
- Решение криптографических задач, которые требуют перебора большого количества случайных решений, — например, майнинг криптовалюты.

В начале курса рассказывалось про закон Мура, согласно которому производительность ЭВМ со временем растёт экспоненциально. Также указывалось, что из-за физических ограничений дальнейшее наращивание производительности процессоров невозможно выполнить за счет более «плотного» размещения транзисторов на одной кремниевой пластине. Напомним, что так происходит потому, что размеры транзисторов становятся сопоставимы с размерами отдельных атомов, а скорость их взаимодействия ограничена

скоростью света. Дальнейшим перспективным направлением увеличения производительности является распараллеливание.

> ***Распараллеливание*** *— это организация программ и средств их выполнения в виде набора параллельных взаимодействующих активностей.*

Важно отметить, что сама программа может быть как реализована с использованием различных конструкций параллельного программирования (например, с помощью средств многопоточности), так и выполняться на различных параллельных архитектурах, например, в облачной инфраструктуре с использованием различного количества вычислительных узлов.

Отметим также, что часто под распараллеливанием понимают добавление параллелизма для уже реализованного алгоритма/программы, исходно реализованных последовательно. Таким образом, термин «распараллеливание» отражает тот факт, что создавать параллельные программы непросто и это требует специальных усилий.

Системы с общей памятью

В этом разделе мы рассмотрим типичные подходы к созданию параллельных вычислительных систем. Вначале остановимся на системах с общей памятью.

> ***Система с общей памятью*** *— это вычислительная система, в которой параллельно выполняемые вычислительные процессы могут одновременно работать с одними и теми же данными, считывая и изменяя их.*

Рассмотрим следующие способы организации таких систем:

- многоядерные процессоры;
- многопроцессорные компьютеры.

Многоядерные процессоры. Мы уже рассматривали многоядерные процессоры выше, здесь же сделаем акцент на использование многоядерных процессоров для высокопроизводительных вычислений.

Основная проблема, с которой сталкиваются инженеры при проектировании многоядерных процессоров, заключается в обеспечении согласованной работы ядер. Последние одновременно работают с общими данными в оперативной памяти, что является источником ошибок. При этом приходится искать компромисс между двумя следующими решениями.

- Использование ядрами общего канала обмена данными с памятью и общей кэш-памятью (простое в реализации, но неэффективное решение).

- Использование отдельных каналов для работы с памятью и отдельной кэш-памяти для каждого ядра, что приближает многоядерный процессор к многопроцессорным системам (сложное в реализации, но эффективное решение).

Пример устройства кэш-памяти многоядерного процессора Intel Core i7, который мы приводили выше, как раз иллюстрирует сложности, возникающие при поиске описанного компромисса.

Многопроцессорные компьютеры. В таких компьютерах параллелизм реализуется через набор процессоров, которые, подобно ядрам многоядерных процессоров, работают параллельно. Такой подход позволяет в известной степени преодолеть технологические ограничения по количеству ядер, которые удаётся разместить на одной кремниевой пластине процессора и ещё более увеличить производительность системы.

В многопроцессорных компьютерах узким местом являются шины, используемые для доступа процессоров к общей оперативной памяти. Также при проектировании шин многопроцессорных компьютеров возникают сложные задачи синхронизации разных процессоров и обеспечения когерентности (согласованности) их кэш-памяти. Важно понимать, что многопроцессорные компьютеры имеют весьма сложное устройство, решая те же проблемы, что и многоядерные процессоры, но в большем масштабе. И решать их приходится уже на уровне системной шины, а иногда — на уровне особого устройства оперативной памяти.

Многопроцессорные компьютеры появились в 1960-х годах. Одними из первых были компьютеры Burroughs, а также некоторые модели System/360. Многие процессоры семейства Intel x86 начиная с Pentium (1993 г.) можно использовать для создания многопроцессорного вычислительного комплекса. В настоящее время ситуация с многопроцессорными компьютерами такова.

- На рынке имеются двухпроцессорные рабочие стации, которые используются для высокопроизводительных задач без привлечения кластеров и суперкомпьютеров — например, для научных вычислений или видеомонтажа.
- При необходимости обеспечения высокопроизводительных серверов (например, в дата-центрах) используются 2–4-процессорные компьютеры (например, на базе процессора Intel Xeon). Это бывает необходимо для масштабирования веб-сервисов и баз данных, для расширения возможностей виртуализации и т. д.
- Специализированные многопроцессорные вычислительные системы, на базе которых создаются вычислительные кластеры и суперкомпьютеры.

Многие современные многопроцессорные рабочие станции и серверы относятся к классу архитектур *NUMA (Non-Uniform Memory Access, неоднородный доступ к памяти)*. В этих архитектурах у каждого процессора имеется собственная оперативная память, соответственно, к ней процессор имеет быстрый доступ, а доступ к оперативной памяти другого процессора осуществляется через специальную шину. Операционная система при этом позволяет отобразить адресное пространство процесса, выполняющегося на каком-либо процессоре, на память другого процессора. Как и в случае с виртуальной памятью, это делается прозрачно для прикладных программ, но неизменно сказывается на производительности, что следует учитывать при написании программ для таких архитектур.

> *Следует отметить, что в настоящее время в персональных компьютерах массово используется не многопроцессорность, а многоядерность, что во многом обусловлено рыночной целесообразностью.*

Системы с изолированной памятью

Теперь рассмотрим другой способ организации параллельных вычислительных систем — системы с изолированной памятью.

> ***Системы с изолированной памятью*** *— это такие вычислительные системы, в которых параллельные активности работают лишь с собственными данными, а взаимодействие между ними реализуется посредством обмена сообщениями.*

Высокопроизводительные вычислительные системы с изолированной памятью организуются в виде вычислительных кластеров. На аппаратном уровне, в рамках вычислительных кластеров и суперкомпьютеров, распараллеливание реализуется объединением компьютеров в единую систему с помощью высокоскоростной локальной сети. Как и в случае с шинами в многопроцессорных компьютерах, в вычислительных кластерах узким местом является синхронизация и взаимодействие вычислительных узлов.

Суперкомпьютеры — это крупные вычислительные кластеры. Исторически суперкомпьютером называли любую ЭВМ (не обязательно кластер) с очень высокой производительностью. В качестве примера можно привести суперкомпьютер CDC 6600 компании Cray Research, созданный в 1963 году. Иногда в суперкомпьютер для повышения производительности добавляются специализированные вычислительные средства — например, для быстрого умножения больших матриц или для эффективной параллельной обработки массивов с целочисленными данными. Примером современного суперкомпьютера является вычислительный кластер суперкомпьютерного центра Санкт-Петербургского политехнического университета им. Петра Великого (https://research.spbstu.ru/skc/). На текущий момент этот суперкомпьютер

имеет в своём составе 668 двухпроцессорных узлов с 14-ядерными процессорами. Он занимает несколько крупных помещений и потребляет электроэнергии до 640 кВт (для сравнения напомним, что современная городская квартира потребляет в среднем менее 1 кВт).

Технологии реализации высокопроизводительных вычислений

Многопоточность

> *На программном уровне параллельные системы с общей памятью могут быть реализованы с помощью **многопоточности**, то есть использования в программе параллельных потоков (threads), работающих в рамках одного процесса и совместно использующих данные программы и прочие доступные ресурсы.*

При этом разные потоки программы операционная система естественным образом распределяет по параллельно работающим ядрам процессора. Многопроцессорные компьютеры также позволяют использовать многопоточное программирование.

Средства реализации многопоточных приложений встроены в основные языки программирования, в частности, в язык C++. В последнем имеется специальный класс `thread` (для его использования нужно подключить библиотеку `thread.h`), и когда программист создаёт объект этого класса и инициализирует его некоторой функцией F, то стартует новый поток, в котором и выполняется функция F. Библиотека `thread.h` предоставляет также средства для синхронизации различных потоков при работе с общими ресурсами — памятью, библиотечными функциями и пр.

Многопоточные программы позволяют существенно повысить производительность приложений, но чреваты трудно обнаружимыми ошибками. Самой распространенной ошибкой многопоточного программирования является гонка по данным (Data Race). Эта ошибка возникает в том случае, когда два потока без специальной синхронизации (т. е. в произвольном порядке) обращаются к одному и тому же участку памяти, и одно из этих обращений является записью данных. Таким образом оказываются возможным сценарии, когда чтению из одного потока предшествует записи из другого в одну и ту же переменную и наоборот. При этом программист, когда писал программу, не предполагал наличия первого вышеупомянутого сценария. Наличие такого недетерминизма и означает отсутствие синхронизации потоков. Оно приводит к ошибочным результатам при работе приложения — например, программа не производит зачисления денег клиента на его банковский счёт, а должна была… Сложность ситуации связана с тем, что эффекты от произошедшей гонки могут обнаруживаться не сразу, а по прошествии некоторого времени. Существует много подходов к по-

иску гонок — на сегодняшний день для индустрии эта тематика является чрезвычайно востребованной.

Для упрощения разработки многопоточных программ для многопроцессорных систем имеются специальные языки программирования. Один из известных современных примеров таких языков — появившийся в 1997 году OpenMP. Это язык, который можно встраивать в другие языки программирования (чаще всего — в C и Fortran) для распараллеливания программного кода, в основном выполняющего математические вычисления.

Фактически OpenMP является встроенным предметно-ориентированным языком программирования. В настоящее время тема предметно-ориентированных языков программирования является очень популярной, поскольку такие языки позволяют достичь многочисленных выгод и создаются не только для специфических областей, но также и для отдельных больших программных проектов и линеек продуктов. Имеются также специальные средства для создания инфраструктуры для таких языков — продукт JetBrains под названием MPS, открытая технология Xtext из мира Eclipse и др. Поверхностно-встроенные языки — это такие предметно-ориентированные языки, которые можно использовать совместно с обычными языками.

Распределённые вычисления. На программном уровне распараллеливание может осуществляться при помощи *распределённых вычислений*: прикладная программа реализует использование и синхронизацию компьютеров вычислительного кластера, управляя тем, какие операции и над какими данными будет выполнять каждый из них, а также то, как они этими данными будут обмениваться. Распределённые вычисления позволяют, например, одновременно обрабатывать на разных компьютерах разные порции данных или, организовав из компьютеров кластера вычислительный конвейер, распределить на них разные стадии обработки данных.

Одной из популярных технологий организации распределённых вычислений является библиотека Message Passing Interface (MPI), предназначенная для организации распределённой работы программ на вычислительных кластерах. MPI позволяет параллельно запустить несколько экземпляров программы (10, 100, 500) на различных процессорах кластера и синхронизировать работу этих экземпляров с общими данными. Функциональность библиотеки MPI описана в одноименном стандарте. Существует значительное число различных реализаций стандарта MPI — например, MPICH, OpenMPI и др.

Облачные вычисления

Одно и то же программное приложение может быть размещено и исполняемо на различном количестве вычислительных узлов. Это бывает нужно для параллельных приложений с целью повысить их пропускную способность (например, для клиент-серверных приложений или приложений Интернета вещей). С другой стороны, вычислительные ресурсы стоят дорого и их увеличение для данного приложения должно быть оправданным.

Кроме этого, необходимо отметить, что вычислительный кластер (любой, который даже не является суперкомпьютером) оказывается дорогим в приобретении и обслуживании. Отдельным исследователям, небольшим лабораториям и организациям обычно выгоднее арендовать вычислительные мощности по мере необходимости, а не приобретать такое оборудование. В последнее десятилетие для решения подобных задач часто применяются *сервисы облачных вычислений*.

Сервис облачных вычислений позволяет выделять вычислительные ресурсы (виртуальные машины) в количестве, необходимом пользователю в данный момент, и настраивать сетевые соединения между ними. В итоге пользователь на время получает доступ к кластеру нужной мощности и конфигурации. Широко известны такие облачные сервисы, как Amazon EC2 или Microsoft Azure.

Технологии облачных вычислений позволяют клиентам получить доступ к серверам в требуемом количестве. Это количество можно по желанию быстро варьировать, но тем не менее, как и в случае с реальными серверами, эти виртуальные серверы необходимо настраивать и сопровождать. Это является сложной задачей системного администрирования, несмотря на наличие большого количества специализированных программных утилит для синхронного управления парком серверов.

Организация параллельных облачных вычислений упрощается при использовании вычислительных сервисов класса *Функция как услуга (Function as Service)*. Наиболее известными представителями этого класса являются платформы Amazon AWS Lambda, Microsoft Azure Functions и Google Cloud Functions. Они предоставляют пользователю программный интерфейс, позволяющий передать сервисам программный код и входные данные. Пользователь реализует функции на одном из поддерживаемых языков программирования (поддерживаются популярные высокоуровневые языки программирования, такие как Java, C#, Python и ряд других) и передаёт ссылки на эти функции в библиотеку, которая, в свою очередь, передаёт код этих функций на серверы Amazon, Microsoft или Google и инициирует их удалённое выполнение. Сервисы решают самостоятельно, каким образом распределить вычисления по реальным серверам. Такой подход избавляет от необходимости вручную настраивать серверы, но лишает пользователя гибкости в настройке параллельных вычислений.

Вопросы

1. Приведите примеры современных вычислительно сложных задач.
2. Сформулируйте закон Мура.
3. Что такое распараллеливание?
4. Что такое параллельные вычислительные системы с общей памятью?
5. Назовите виды параллельных вычислительных систем с общей памятью.

6. Какие средства программирования применяются для систем с общей памятью?
7. Почему стали массово популярны многоядерные процессоры, а не многопроцессорные компьютеры?
8. Что такое параллельные вычислительные системы с изолированной памятью?
9. Какие средства программирования применяются для систем с изолированной памятью?
10. Какие возможности открывает многопоточное программирование?
11. Назовите самую трудно обнаруживаемую и «дорогую» ошибку многопоточного программирования.
12. Что такое распределённые вычисления?
13. Расскажите про облачные вычисления.
14. Что такое вычислительные сервисы класса «Функция как услуга»?
15. Какие известные платформы для этих сервисов вы можете назвать?

Литература

1. MPI: A Message-Passing Interface StandardVersion 3.1. Message Passing Interface Forum. June, 2015. 868 p.
2. *Бурова И. Г., Демьянович Ю. К.* Алгоритмы параллельных вычислений и программирование. СПбГУ, 2007. 207 с.
3. *Немнюгин С. А.* Параллельное программирование для многопроцессорных вычислительных систем. СПб.: БХВ-Петербург, 2002. 400 с.
4. *Малявко А., Менжулин С.* Суперкомпьютеры и системы. Построение вычислительных кластеров. М.: ЛитРес, 2019. 96 с.
5. *Таненбаум Э., Бос Х.* Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
6. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
7. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
8. *Roberts M.* Serverless Architectures. 2016. https://martinfowler.com/articles/serverless.html.

Список литературы к курсу

1. *Брукс Ф., Чапель Х.* Мифический человеко-месяц, или Как создаются программные системы. Пер. с англ. СПб.: Символ-Плюс, 2010. 304 с.
2. *Бурова И. Г., Демьянович Ю. К.* Алгоритмы параллельных вычислений и программирование. СПбГУ, 2007. 207 с.
3. *Гуров В. В.* Архитектура микропроцессоров. М.: ИНТУИТ.РУ, Бином. 2010. 272 с.
4. *Малявко А., Менжулин С.* Суперкомпьютеры и системы. Построение вычислительных кластеров. ЛитРес, 2019. 96 с.
5. *Немнюгин С. А.* Параллельное программирование для многопроцессорных вычислительных систем. СПб.: БХВ-Петербург, 2002. 400 с.
6. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: учебник для вузов. 2-е изд. СПб.: Питер, 2011. 688 с.
7. *Парфенов В. В.* Проектирование и реализация программного обеспечения встроенных систем с использованием объектно-базированного подхода: дисс. на соискание ученой степени канд. физико-математических наук. СПб.: СПбГУ, 1995. 115 с.
8. *Робачевский А. М., Немнюгин С. А., Стесик О. Л.* Операционная система UNIX, 2 изд. СПБ.: БХВ-Петербург, 2010. 656 с.
9. *Свистова Т. В.* Основы микроэлектроники: учеб. пособие [электронный ресурс] — Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017.
10. *Таненбаум Э., Бос Х.* Современные операционные системы. 4-е издание. СПб.: Питер, 2015. 1120 с.
11. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 816 с.
12. *Терехов А. Н.* УВК «Самсон» — базовая ЭВМ РВСН // Труды конференции SORUCOM-2011. 2011. С. 282–286.
13. *Харрис Д. М., Харрис С. Л.* Цифровая схемотехника и архитектура компьютера. Пер. с англ. Imagination Technologies. М.: ДМК Пресс, 2018. 792 с.
14. *Хорошевский В. Г.* Архитектура вычислительных систем: учеб. пособие. 2-е изд., перераб. и доп. М.: Изд-во МГТУ им. Н. Э. Баумана, 2008. 520 с.

15. *Цикритзис Д., Бернстайн Ф.* Операционные системы. Пер. с англ. М.: Издательство «Мир», 1977. 336 с.
16. *Bahn H., Noh, S. H.* Characterization of Web reference behavior revisited: Evidence for Dichotomized Cache management // International Conference on Information Networking 2003. Jeju, South Korea: Springer-Verlag. P. 1018–1027.
17. *Courtois P. J., Heymans F., Parnas D. L.* Concurrent Control with «Readers» and «Writers» // Communications of the ACM, № 14 (10), 1971.
18. *Dagum L., Menon R.* OpenMP: an industry standard API for shared-memory programming // IEEE computational science and engineering. 1998. Vol 5. № 1. P. 46–55.
19. *Dijkstra E. W.* Hierarchical ordering of sequential processes // Acta Informatica. Vol. 1. 1971. P. 115–138.
20. *Gropp W. et al.* Using MPI: portable parallel programming with the messagepassing interface. MIT press, 1999.
21. Intel 80386 Programmer's Reference Manual. Intel Corporation, 1986.
22. IEEE Standard Graphic Symbols for Logic Functions (Including and incorporating IEEE Std 91a-1991, Supplement to IEEE Standard Graphic Symbols for Logic Functions) // IEEE Std 91a-1991 & IEEE Std 91–1984, 1984, C. 1–160.
23. MPI: A Message-Passing Interface Standard. Version 3.1. Message Passing Interface Forum. June, 2015. 868 p.
24. *Netzer R., Miller B.* What Are Race Conditions? Some Issues and Formalizations // ACM Letters On Programming Languages and Systems, № 1 (1), 1992. P. 74–88.
25. OptiPlex 7060 Small Form Factor Service Manual. Dell Inc, 2018. https://topics-cdn.dell.com/pdf/optiplex-7060-desktop_service-manual2_en-us.pdf.
26. *Patterson D. A., Ditzel D. R.* The case for the reduced instruction set computer // ACM SIGARCH Computer Architecture News, 8 (6), 1980. P. 25–33.
27. *Roberts M.* Serverless Architectures. 2016. https://martinfowler.com/articles/serverless.html.
28. *Silberschatz A., Gagne G., Galvin P. B.* Operating System Concepts, 10th edition. Wiley, 2018. 951 p.
29. *Stallings W.* Operating systems: internals and design principles. Pearson Education Limited, 2018.
30. *Walden D., Van Vleck T.* Compatible Time-Sharing System (1961–1973): Fiftieth Anniversary Commemorative Overview. IEEE Computer Society, 2011.

Учебное издание

Луцив Дмитрий Вадимович,
Мокаев Руслан Назирович,
Гаранина Наталия Олеговна,
Кознов Дмитрий Владимирович

АРХИТЕКТУРА ЭВМ

Учебное пособие

Литературный редактор Ю. Никулина
Корректор Ю. Никулина
Компьютерная верстка А. Савин
Подписано в печать 10.01.2022. Формат 60x90 1/16.
Гарнитура «Таймс». Бумага офсетная. Печать офсетная.
Усл. печ. л. 9,5. Тираж 1000 экз.
ООО «ИНТУИТ.ру»
Национальный Открытый Университет «ИНТУИТ»
Москва, Электрический пер., 8, стр. 3
Телефон: +7 (499) 253-9312, 253-9313, факс: +7 (499) 253-9310
E-mail: info@intuit.ru, http://www.intuit.ru