Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра теоретических основ информатики

М.М. Аббаси, А.П. Бельтюков

КОМПЬЮТЕРНАЯ АРХИТЕКТУРА

Учебно-методические пособие

Ижевск
2024

УДК 004.2(075.8)
ББК 32.971.3Я73
А137

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент: канд. техн. наук, доцент, зав. каф. «Автоматизированные системы обработки информации и управления» Ижевского государственного технического университета имени М.Т. Калашникова **М.Н. Мокроусов**

Аббаси М. М., Бельтюков А.П.

А137 Компьютерная архитектура : учеб.-метод. пособие. – Ижевск : Удмуртский университет, 2024. – 89 с.

Пособие нацелено на углублённое освоение дисциплины «Архитектура вычислительных систем» и «Теория систем и системный анализ». Оно состоит из семи глав. Каждая глава посвящена базовым механизмам, моделям и компонентам, необходимым для проектирования компьютерной системы.

Учебно-методические пособие предназначено для студентов, обучающихся по направлению «Прикладная информатика». Пособие также может быть использовано студентами других смежных направлений, изучающими принципы управления проектами и разработку нового компьютерного оборудования.

УДК 004.2(075.8)
ББК 32.971.3Я73

ВВЕДЕНИЕ

Компьютеры – это одна из ключевых частей нашей повседневной жизни, начиная от машин, которые мы используем для работы, до смартфонов и умных часов, на которые мы полагаемся. Работа всех компьютеров, независимо от их размера, основана на наборе правил, определяющих то, как программное и аппаратное обеспечение объединяются и взаимодействуют для работы компьютера. Это называется компьютерной архитектурой. В этой работе мы рассмотрим, что такое компьютерная архитектура на самом деле, и поможем студентам понять её различные компоненты и принцип работы. Понимание компьютерной архитектуры для студентов, изучающих компьютерные науки и информационные технологии, очень важно, поскольку это помогает им понять основополагающие принципы языков программирования, облегчает им разработку компьютерных программ и приложений. Аналогичным образом, разработка компьютерных алгоритмов и измерение их эффективности возможны только в том случае, если программист понимает и знает организацию компьютерной архитектуры и то, сколько времени требуется для передачи команды или данных из памяти в другие компоненты системы.

Простыми словами, компьютерная архитектура – это организация компонентов, составляющих компьютерную систему, и значение операций, которые определяют ее функционирование. Она определяет, что отображается в машинном интерфейсе, на который нацелены языки программирования и их компиляторы. Организация компонентов компьютерной системы зависит от набора правил и методов, которые описывают функциональность компьютеров, управление ими и их реализацию. Если быть точным, это не что иное, как правила, по которым система функционирует.

Одна из основных функций компьютерной архитектуры заключается в обеспечении баланса между производительностью, экономичностью, стоимостью и надёжностью компьютерной системы.

Например, архитектура набора команд (ISA) служит связующим звеном между программным обеспечением и аппаратным обеспечением компьютера. Она представляет собой знание программиста о машине.

Компьютерная система может понимать только двоичный язык, в то время как её пользователи понимают программный язык высокого уровня. Таким образом, для взаимодействия между пользователем и компьютером архитектура набора команд играет важную роль в переводе программного языка высокого уровня в двоичный язык. Это двухэтапный процесс, во-первых, программа, написанная программистом на любом программном языке высокого уровня, преобразуется в программный язык ассемблера, который является полу-машинным и полу-языком программирования высокого уровня. Это важно, поскольку некоторые компоненты компьютерной архитектуры работают на программном языке ассемблера, особенно те, которые выполняют декодирование адресов различных устройств ввода-вывода, подключенных к системе, и адресов ячеек памяти. После выполнения всего необходимого декодирования программа преобразуется в машинный язык, также называемый двоичным языком. Недавно было разработано и протестировано несколько различных систем счисления, таких как четырех-, восьмеричная, шестнадцатеричная системы счисления, которые потенциально могут заменить двоичный язык в будущем.

Другая важная роль компьютерной архитектуры заключается в контроле и управлении передачей данных между различными ее компонентами, такими как регистры, внутренняя память, внешняя память и т. д. Были разработаны различные конфигурации компьютерной архитектуры для ускорения перемещения данных, что позволяет увеличить объем обработки данных. В основе базовой архитектуры лежит центральный процессор с основной памятью и системой ввода-вывода по обе стороны от центрального процессора. Существует несколько других доступных вариантов, таких как архитектура, использующая основную память в качестве места в компьютерной системе, откуда исходят инструкции и данные и поступают

в память. Аналогично, в другой организации компьютерной архитектуры общие данные и управление используются для соединения всех устройств, составляющих компьютерную систему.

Использование и функции компьютерной системы определяют дизайн ее архитектуры. В случае, если компьютер предназначен для личного пользования, стоимость и производительность должны быть сбалансированы для разработки архитектуры компьютера, подходящей пользователю для его повседневного использования. В случае, если компьютерная система будет представлять собой базу данных или сетевой сервер, требования и дизайн архитектуры будут совершенно иными. Надёжность системы будет основным фактором, определяющим дизайн компьютерной системы. Ещё одно важное применение компьютерной системы, за которой будущее технологий, – это разработка и внедрение программ искусственного интеллекта. Для программ, реализующих искусственный интеллект, требуется производительность в реальном времени, высокая надёжность, нулевая частота ошибок и т. д. Таким образом, дизайн компьютерной системы будет отличаться.

Одним из основных направлений деятельности исследователей сегодня является разработка моделей и механизмов, которые могут контролировать факторы, влияющие на производительность компьютерных систем и снижающие ее, особенно если эти системы используются в оборонной промышленности, где незначительная проблема в компьютерной архитектуре может привести к катастрофе. Защитные модели включают в себя проектирование компьютерной архитектуры с дополнительными компонентами таким образом, чтобы в случае выхода из строя какого-либо компонента компьютерной системы дополнительный компонент мог автоматически заменить неисправный, и система бесперебойно выполняла бы свои функции.

«Архитектура вычислительных систем» и «Теория систем и системный анализ» являются основными и значимыми дисциплинами для студентов бакалавриата, обучающихся по направлению «Прикладная информатика». Это учебно-методические пособие будет

полезно студентам для понимания и выполнения сложных задач, изложенных в указанных дисциплинах. Эта работа разделена на несколько глав, в которых подробно рассматриваются компоненты и функции компьютерной архитектуры. При подготовке данного пособия основное внимание уделялось тому факту, что студент должен изучить все современные концепции компьютерной архитектуры. После прочтения этой работы студент должен быть способен работать над проектированием как аппаратных, так и программных компонентов компьютерных систем. Студенты должны уметь измерять производительность своих компьютерных программ и проектировать компьютерную архитектуру для своих систем. В качестве примеров работ используются учебные работы обучающихся Удмуртского государственного университета.

ГЛАВА 1. КОМПЬЮТЕРНАЯ АРХИТЕКТУРА И ЕЁ ОСНОВЫ

1.1. Что такое компьютер?

Компьютером называется электронное устройство, работающее в соответствии с заранее заданным набором инструкций. В современную эпоху с помощью компьютера мы можем решать самые разные проблемы. Обычно современный компьютер состоит только из одного обрабатывающего компонента, такого как центральный процессор, и оперативной памяти. Работа центрального процессора состоит из выполнения логических и арифметических операций арифметико-логическим устройством, а выбор операции осуществляется устройством управления. Периферийные устройства используются для обмена информацией с внешним миром.

1.1.1. Механический аналоговый компьютер:

Механический аналоговый компьютер появился в XX веке и широко использовался для вычислений. Во время Второй мировой войны механический аналоговый компьютер использовался только для конкретных военных целей. В это же время были созданы первые электронные цифровые компьютеры. Размером такой компьютер был с большую комнату и потреблял очень много энергии.

1.1.2. Что такое компьютерная архитектура?

Структурный дизайн компьютера относится к атрибутам системы, заметным только программисту. Он состоит из тех компонентов, которые отвечают за бесперебойное выполнение программ и инструкций пользователя.

1.2. Базовый компонент компьютерной архитектуры

Основные компоненты компьютерной архитектуры следующие:

1. Центральное процессорное устройство
2. Основная память
3. Устройство ввода-вывода
4. Компьютерная шина.

1.2.1. Центральное процессорное устройство

Центральное процессорное устройство (CPU) или «центральный процессор». Процессор является наиболее важным компонентом компьютерной архитектуры. Его можно назвать «головой» компьютера. Он обрабатывает данные и вводит их по определённой командам (в определённой последовательности), преобразуя в ценную информацию.

Процессор разделен на два основных механизма:

– Арифметико логическое устройство (ALU);

– Устройство управления (Control Unit – CU).

Арифметико логическое устройство (ALU)

Арифметико-логическое устройство (ALU) выполняет все логические и арифметические операции с данными.

Устройство управления (CU)

Устройство управления (CU) – «Блок управления». Это наиболее важная часть центрального процессора.

Компьютер пропускает все команды пользователя через блок управления одна за другой, управляя операциями, которые надо выполнять.

1.2.2. Основная память

Память – это компьютерный компонент, используемый для хранения данных и команд.

Команды пользователя извлекаются из памяти компьютера и выполняются одна за другой центральным процессором (ЦП). Таким образом команды образуют программу.

1.2.3. Устройство ввода-вывода

Устройства ввода-вывода включают, например, принтер, мышь, клавиатуру, монитор, дисковод. К компьютеру можно подключать разные устройства.

1.2.4. Компьютерная шина

Соединение частей компьютера с помощью специальных устройств – шин также является важной частью компьютерной архитектуры, и используется для передачи данных между функциональными блоками компьютера. В устройстве шины можно различить такие уровни как механический, электрический и логический.

1.3. Зависимость компьютерной системы от её архитектуры

Архитектура системы – это концептуальная структура, которая определяет функции и компьютерные компоненты. Она также контролирует поведение компонентов и взаимосвязь между ними. А также даёт нам алгоритм дальнейшей работы с компьютерной системой, где мы создаём план применения оборудования системы и других компонентов. Компьютерная архитектура описывает развивающиеся и специфичные компоненты системы, которые повышают её интеллект. Это, в свою очередь, даёт нам идеи о том, как мы можем разработать дизайн для пользователей этой системы, а также даёт нам технические методы, которые расширяют понимание системы.

Системная архитектура является языком, который используется для коммуникации между компонентами компьютерной системы.

1.4. Измерение производительности компьютера (на основе архитектуры)

Производительность компьютера означает, насколько быстро компьютер выполняет работу.

Производительность зависит от:

Доступности компьютера, времени отклика, скорости обработки, пропускной способности, размера и веса, теста производительности и его настройки.

1.4.1. Доступность

Доступность состоит в том, что компьютерная система работает в течение всего срока службы, без неисправностей и ухудшения производительности.

1.4.2. Время отклика

Это время между началом и завершением работы. Если время ответа меньше, значит производительность выше.

1.4.3. Скорость обработки

Особой целью выбора компьютерной архитектуры является выполнение ею большого количества программных инструкций. Поэтому увеличение скорости обработки информации также увеличивает производительность компьютерной системы.

1.4.4. Пропускная способность

Это общий объем работы, выполненной за заданное время (например, мегабайт в секунду).

1.4.5. Размер и вес

Это важный фактор для измерения производительности системы. Меньший размер устройства не позволяет установить на него большее количество элементов. Однако с развитием технологий

высокопроизводительные компоненты можно устанавливать на меньшем месте. Однако в этом случае стоимость системы может увеличиться во много раз.

1.4.6. Контрольные показатели:

Это стандартные тесты, которые используются для измерения производительности системы или подсистемы при выполнении чётко определённой задачи.

1.4.7. Настройка производительности:

Настройка производительности используется для повышения производительности операционной системы (ядра) – базовой управляющей программы компьютера.

1.5. Важность компьютерной архитектуры при проектировании системы

Компьютерная архитектура содержит набор команд, организацию или микроархитектуру (внутреннюю реализацию компьютера на уровне регистров и функциональных блоков) и системную архитектуру (организацию компьютера на уровне таких устройств как «кэш» – вспомогательная память – и шины). Архитектура также помогает другим частям вычислений, таким как операционная система (для ввода, вывода, и с технологией памяти) и язык высокого уровня (для реализации указателей, передачи параметров).

Архитектура компьютера является ключевым компонентом компьютерной инженерии. Она касается всех аспектов проектирования и организации центрального процессора и его интеграции в компьютерную систему. Компьютерный дизайнер должен иметь представление о программном обеспечении, чтобы реализовать оптимальную архитектуру. Многие процессоры используются не в персональных компьютерах или мощных компьютерах-серверах, а во встроенных системах (системах, встроенных в другое оборудование).

Слово архитектура и компьютерная архитектура определяют потребности человека, который использует систему, технологию и логически проектирует систему. В настоящее время хорошим примером архитектуры является архитектура фон Неймана, которая используется чаще, чем другие архитектуры. Эта архитектура была предложена математиком Джоном фон Нейманом. Здесь представлен дизайн компьютера с центральным процессором, а центральный процессор состоит из следующих частей:

1. Арифметико-логическое устройство.
2. Регистры.
3. Оперативная память для сохранения данных и кода программ.
4. Компьютерные устройства ввода/вывода.
5. Внешнее запоминающее устройство.

Исследовательский центр IBM предложил слово «компьютерная архитектура», объединив слова «компьютер» и «архитектура». Базовую структуру компьютерной архитектуры вместе с её компонентами и взаимодействием между ними можно увидеть ниже, на рис. 1.

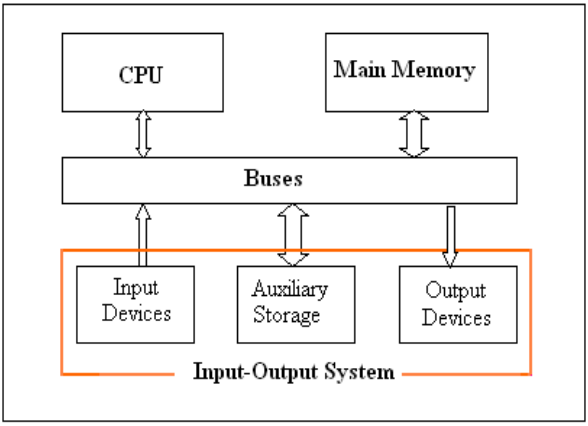

Рис. 1. Основные компоненты компьютерной системы (взято с сайта https://www.teach-ict.com/ по теме «компьютер и его компоненты»)

1.6. Компьютерная архитектура (организация фон Неймана)

В архитектуре фон Неймана данные и инструкции сохраняются в одном блоке. Архитектура фон Неймана обладает следующими свойствами: данные и инструкции смешиваются в одной и той же объединённой памяти. Она также известна как Принстонская архитектура (рис. 2).

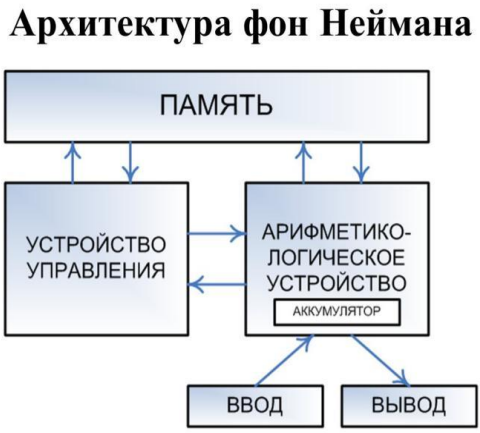

Рис. 2. Основные компоненты в архитектуре Неймана (взято с сайта https://habr.com/ru/articles/316520/ по теме Собственная платформа. Часть 0.1 Теория. Немного о процессорах)

1.7. Эволюция компьютерной архитектуры во времени

Компьютерная архитектура 1950-х - 1960-х годов

В это время, для выполнения арифметической операции на компьютере, впервые был использован термин «компьютерная арифметика».

Компьютерная архитектура 1970-х - середины 1980-х годов

В это время создаётся дизайн набора команд, в частности Архитектура набора команд (ISA – instruction set architecture).

Компьютерная архитектура с 1990-х по 2000-е годы

Создаётся проектирование центрального процессора, системы памяти, системы ввода-вывода, мультипроцессоров, сетей.

Компьютерная архитектура с 2000-х годов и далее

Были созданы архитектуры специального назначения, функционально реконфигурируемые, были предложены специальные решения для архитектур обработки данных с низким энергопотреблением и мобильных устройств, были разработаны системы высокой мощности при низкой стоимости, были разработаны архитектуры для интеллектуальных систем, обеспечивающие производительность в реальном времени.

1.8. Инновации в компьютерных системах

1.8.1. Компьютеры первого поколения (1944–58 гг.)

Электронные компьютеры появились в XX веке после 1930-х гг. Впервые для электронных схем были разработаны вакуумные трубки (радиолампы). Эти компьютеры, основанные на вакуумных трубках, называются компьютерами первого поколения.

ENIAC представляет собой компьютер первого поколения. Он был построен в 1943–1946 годах. Он представлял собой следующее:

- в нем использовались 18000 вакуумных трубок;
- в нем использовались отдельные блоки памяти для программ и данных;
- он выполнял операции сложения, умножения, деления и вычитания;
- он использовал 20 электронных блоков памяти (20 аккумуляторов-накопителей).

1.8.2. Компьютеры второго поколения (1954–68 гг.)

В компьютерах начинают использоваться транзисторы, которые были разработаны для электронных схем в 1947 году. Эти компьютеры на основе транзисторных схем называются компьютерами

второго поколения. Транзисторы значительно уменьшают рассеиваемую мощность и занимаемое место в компьютере. Это также помогло увеличить скорость работы компьютеров по сравнению с системами первого поколения.

Компьютеры IBM1620 и IBM7094 представляют второе поколение. Они были построены в 1954–64 годах. Второе поколение просуществовало до 1968 года. Они отличались наличием следующих компонентов:

- транзисторы в компьютерах;
- дополнительные регистры в процессоре;
- несколько режимов адресации для выборки данных;
- концепция пакетной обработки.

1.8.3. Компьютеры третьего поколения (1964–75 гг.)

В компьютерах этого поколения использовались интегральные схемы (ИС), которые были разработаны для использования в качестве электронных схем в 1960-х годах. Эти компьютеры на базе ИС называются компьютерами третьего поколения. ИС снижает рассеиваемую мощность и значительно сокращает пространство, необходимое для компьютера. Наличие ИС также увеличивает оперативную скорость вычислений по схемам на основе транзисторов. IBM360 представляет собой компьютер третьего поколения и был построен в 1964–65 годах. В нем были представлены следующие компоненты:

- микросхемы, каждая из которых содержит 100–1000 электронных логических элементов;
- 32-разрядный формат команд.
- программирование на ассемблере, а также на языках высокого уровня, например FORTRAN и COBOL.

1.8.4. Компьютеры четвёртого поколения

Микросхемы VLSI (очень крупномасштабная интегральная схема) применялась в компьютерах с 1970-х годов и микросхемы VVLSI разрабатывались для процессоров и памяти с 1990-х годов. Компьютеры на базе VLSI и VVLSIIC называются компьютерами четвёртого поколения. Микропроцессор был изобретён как центральный процессор на одной микросхеме VLSI. Микросхемы основной памяти объёмом 1 МБ плюс адреса памяти были представлены в виде одной микросхемы VLSI. Была изобретена вспомогательная высокоскоростная кэш-память для ускорения работы с памятью для размещения в микропроцессоре. Эти VLSI и VVLSI значительно сократили пространство, требуемое для компьютера, и значительно увеличили скорость вычислений. Компьютеры на базе IBM PC и PENTIUM представляют четвёртое поколение. ПК IBM были представлены в 1980 году, а PENTIUM – с 1993 года. В них использовался один чип процессора VLSI в качестве микропроцессора.

Кэш память

- содержит большое количество регистров по 16 или 32 бита в каждом микропроцессоре;
- содержит большое количество исполняемых кодов операций (команд).

1.8.5. Будущее компьютерной архитектуры

Компьютерная архитектура сегодня определяет успех информационных технологий, поэтому разработка её так важна для компьютерных архитекторов и инженеров. Очень важно, чтобы исследовательские сообщества компьютерных архитекторов изучали тенденции в области технологий и предлагали компьютерную архитектуру, которая может удовлетворить технологические потребности.

Технологии меняются очень быстрыми темпами. Правила или научные законы, известные в прошлом, могут оказаться недостаточными для удовлетворения потребностей современных технологий. Сегодня наблюдаются очень серьёзные изменения в тенденциях в области технологий.

1.9. Технологические драйверы

Тенденции в технологиях меняют весь облик будущего компьютерной архитектуры. Темп изменений технологий, согласно закону Мура, сегодня изменился и замедляется. Теперь стало труднее добиться большей скорости выполнения вычислений. Добавление большего количества транзисторов в чип в прошлом экспоненциально улучшало вычислительную мощность системы, но теперь повышение эффективности системы при добавлении транзисторов снизилось. Поэтому становится очень сложно увеличивать производительность системы параллельно с изменением конструкции компьютерной архитектуры.

1.9.1. Драйверы компьютерных приложений

Компьютерные инженеры должны учитывать новую и меняющуюся природу различных приложений и программ. За последние годы возросла разработка мультимедийных приложений и приложений для вычислений с интенсивным использованием данных. Также мы стали свидетелями значительного роста как мобильных, так и встраиваемых технологий. Все эти факторы заставили компьютерных архитекторов переосмыслить компьютерную архитектуру и её интерфейсы (такие как загрузка, хранение, вычисления), которые оставались неизменными в течение длительного периода времени.

1.9.2. Метрики и цели

Ещё одно важное изменение тенденции, которое усилило обеспокоенность компьютерных архитекторов, это метрики и цели, которые используются для оценки производительности компьютерных систем. В недавнем прошлом вычисления были ориентированы исключительно на производительность, но теперь стало более ясно, что другие показатели также требуют внимания. Помимо производительности, компьютерные системы должны быть надёжными, безопасными и работать с управляемым и гибким бюджетом мощности. Достижение этих целей требует обширного взаимодействия аппаратного и программного обеспечения, и поэтому становится жизненно важным, чтобы компьютерная архитектура была спроектирована с соответствующими механизмами и интерфейсами для облегчения этого сотрудничества.

Нет уверенности в том, какой именно тип компьютерной архитектуры мы увидим в будущем, а также никто не может оценить, какой будет форма компьютерной архитектуры?

Мы не можем представить будущее компьютерной архитектуры, но у нас все же есть некоторые идеи о будущем компьютерной архитектуры.

Задание № 01

Требуется решить задачу из списка задач с чётким и подробным объяснением.

1. Каковы основные компоненты компьютерной архитектуры? Как проектирование компьютерной архитектуры влияет на производительность компьютерной системы?

Чтобы решить задачу, изучите и разберитесь в тексте подзаголовка № 1.2 и подзаголовка № 1.5.

2. Каковы основные изменения в компьютерной архитектуре, начиная с разработки ранних версий компьютеров в начале 1950-х годов и заканчивая современным временем?

Решение задачи приведено в подзаголовке № 1.7 данной главы. Необходимо прочитать и разобраться в разработке компонентов в разное время.

3. В чем заключается основное различие в архитектуре и производительности компьютерных систем ENIAC и VLSI? Какая из них обладает более высокой производительностью, чем другие, и какой компонент обеспечивает разницу в производительности?

Решение этой задачи приведено в подзаголовке № 1.8. Решение требует надлежащего понимания всех компонентов обоих типов компьютерных архитектур, особенно микропроцессора и разрядности программирования.

4. Каким будет дизайн компьютерной архитектуры в будущем? Чем она будет отличаться от компьютерной архитектуры настоящего времени?

Решение этой задачи смотрите в подзаголовке № 1.8.5 данной главы.

ГЛАВА 2. ИНТЕГРАЛЬНЫЕ СХЕМЫ

2.1. Что такое интегральная схема?

Интегральная схема, также известная как чип или микрочип – это миниатюрная электронная схема, в которой различные транзисторы были установлены одновременно с использованием единого физико-химического процесса. Она представляет собой конструкцию небольших размеров из полупроводникового материала, площадью в несколько квадратных миллиметров, на которой электронные схемы обычно изготавливаются методом фотолитографии и которая закреплена в капсулированном пластике или керамике. Оболочка имеет соответствующие металлические проводники для соединения между интегральной схемой и печатной платой.

Интегральные схемы выполняют одну или несколько электронных функций. Эти функции иногда просты, а иногда и более сложны. Сложные функции часто вынуждают включать несколько основных типов электронных компонентов вместе для реализации схемы.

В электронике микросхема – это миниатюрная электронная схема, состоящая в основном из полупроводниковых приборов и пассивных компонентов, которые встроены в поверхность тонкой подложки наполовину из стекловолокна.

Традиционно для изготовления электронных схем использовались вакуумные трубки (радиолампы). Вакуумные трубки были менее надёжны, очень быстро нагревались и занимали много места. Изобретение транзисторов и их использование в электронных схемах является большим достижением. Электронные схемы теперь более надёжны и имеют меньшие размеры.

Интегральная схема также обеспечивает поддержку прикладных микроэлектронных схем. Компоненты микроэлектронных схем соединяются фиксированным образом. Интегральные схемы бывают разных типов в зависимости от процесса их изготовления и цели, для которой они изготавливаются. Наиболее распространёнными

среди них являются монолитные интегральные схемы (технология полупроводниковых блоков), плёночные схемы (тонкопленочная технология, технология толстых плёнок) и гибридные схемы, в которых объединены монолитная интегральная схема и плёночные схемы.

По типу обработки сигналов проводится различие между аналоговыми и цифровыми схемами. В более широком смысле к интегральным схемам также относятся микроэлектронные схемы, которые должны быть интегрированы к другим компонентам компьютера, или интегрированные компоненты и интегрированная оптика.

Наиболее важным параметром для измерения сложности интегральной схемы является степень её интеграции. Она может быть достигнута за счёт усовершенствованного литографического процесса (lithography), точного контроля легирования, ионной имплантации, нейтронно-активационного легирования и сложных методов выращивания кристаллов.

Производство интегральных схем как на этапе проектирования, так и на более позднем этапе фактического производства различается по своей сложности. Это одна из ключевых технологий не только в электронике, но и во всей технологии в целом.

2.2. Интегральные схемы в компьютере

Интегральные схемы встречаются во всех современных электронных устройствах, таких как часы, автомобили, телевизоры, MP3-плееры, мобильные телефоны, компьютеры, медицинское оборудование и т. д.

Интегральная схема – это электронное устройство, состоящее из транзисторов, конденсаторов, резисторов и катушек индуктивности, которые полностью интегрированы в единый полупроводниковый элемент. Современные интегральные схемы, такие как микросхемы памяти и микропроцессоры, могут содержать сотни миллионов компонентов.

Интегральные схемы в зависимости от их использования подразделяются на программируемые интегральные схемы и непрограммируемые интегральные схемы. Программируемая интегральная схема – термин, не знакомый простым пользователям компьютеров. Эта схема обеспечивает гибкость при сборке аппаратных компонентов и обеспечивает передачу сигналов между ними, заставляя их работать вместе как единый компонент.

Компьютер состоит из различных аппаратных и программных компонентов. Эти компоненты управляются центральным процессором компьютера. Центральный процессор работает как мозг компьютера и размещён на специальном устройстве – материнской плате компьютерной системы.

2.2.1. Программируемое логическое устройство

Программируемое логическое устройство – это другое название программируемых интегральных схем. Это логическая микросхема, для которой заранее не зафиксирована функция для выполнения в компьютерной системе. Материнская плата на более поздних этапах выполнения задачи назначает функцию или подзадачу программируемым интегральным схемам. Программируемая интегральная схема отличается от большинства компонентов материнской платы, которым заранее назначены задачи или функции для выполнения. Программируемое логическое устройство обеспечивает логические операции.

2.2.2. Микросхемы на основе Fusion

Микросхемы на основе Fusion («прожигания») представляют собой тип программируемых интегральных схем. Эти компоненты могут быть запрограммированы только один раз. Микросхемы на основе однократного программирования всё ещё обладают определённой гибкостью при проектировании системы, поскольку их можно запрограммировать один раз. Если компьютер сломан, такой чип нельзя извлечь и поместить в другую компьютерную систему для использования в новой роли.

2.2.3. Перепрограммируемые интегральные схемы

Перепрограммируемые интегральные схемы – ещё один тип программируемых интегральных схем. Это микросхема, которая может использоваться для выполнения различных функций. Программируемые микросхемы работают совместно с ячейкой памяти. Именно сочетание двух элементов придает микросхеме гибкость. Ячейка памяти задаёт конкретную программу чипа. Ячейка памяти чипа может быть сброшена, чтобы затем можно было назначить чипу новую задачу или функцию.

2.3. Типы интегральных схем

Существует три основных типа интегральных схем.

Монолитные схемы: они изготавливаются на одном листе кремния с небольшим содержанием германия, арсенида галлия, кремний-германиевых ионов в схеме;

Тонкопленочные гибридные схемы: они очень похожи на монолитные схемы, но их изготовление намного сложнее. Изготовление гибридных схем используется для создания множества аналого-цифровых и цифро-аналоговых преобразователей;

Толстопленочные гибридные схемы: они сильно отличаются от монолитных схем. Они обычно содержат неинкапсулированные монолитные схемы, транзисторы, диоды на диэлектрической подложке, соединенные проводящими дорожками.

Резисторы на интегральных схемах наносятся методом трафаретной печати. Все это заключается в пластиковые или металлические капсулы, в зависимости от требуемого рассеивания тепловой энергии. Как правило, гибридные схемы используются в источниках питания, автомобильных цепях зажигания и т. д.

2.4. Показатель эффективности

2.4.1. Пропускная способность

Это общий объем работы, выполняемой за заданное время, например, (миллион байт) в секунду или, например, для перемещения механизма доступа диска (из одного положения в другое).

2.4.2. Задержка или время отклика

Это время между началом и завершением события, например, миллисекунды для доступа к диску.

2.5. Тенденции в области энергопотребления интегральных схем

2.5.1. Динамическая мощность

Для комплементарных микросхем металл–оксид–полупроводник энергия, используемая при переключении транзистора, называется динамической мощностью.

Величина требуемой мощности на транзистор пропорциональна произведению ёмкостной нагрузки транзистора, квадрата напряжения и частоты переключения между включённым и выключенным состояниями. Математически это можно представить следующим образом:

$$Power_{dynamic} = 1/2 \times Capacitive\ Load \times Voltage^2 \times Frequency\ Switched$$

Энергетический показатель вычисляется по следующей формуле:

$$Energy_{dynamic} = Capacitive\ Load \times Voltage^2$$

2.5.2. Статическая мощность

Как следует из названия, текущая мощность в цепи должна оставаться постоянной. Однако утечка тока из цепи приводит к увеличению потребления и потере энергии в цепи. Математически

статическая мощность представляет собой произведение статического тока на напряжение:

$$Power_{static} = Current_{static} \times Voltage$$

2.6. Тенденции в стоимости производства компьютерных систем

Стоимость производства компьютерной системы и её компонентов со временем снижается даже без существенных улучшений в технологии производства. На стоимость производства компьютерных систем влияют следующие факторы:

- проекты систем с более высокой выживаемостью при тестировании имеют меньшую стоимость по сравнению с проектами с более низкой выживаемостью при тестировании;
- объем времени, затраченный на производство компьютеров, и коммерциализация влияют на цену и себестоимость;
- добавление в систему большего количества компонентов для повышения производительности может повысить производительность, но это увеличит стоимость системы;
- целевой рынок покупателей и их требования к компьютерной системе могут привести к изменению дизайна системы и в конечном итоге повлиять на стоимость системы;
- часто мы пытаемся найти баланс между ценой и производительностью;
- стоимость также зависит от типа и назначения компьютерной системы, например, настольных компьютеров, серверов, встраиваемых систем. Все они имеют разную функциональность и разную стоимость.

2.7. Матрица

Существует специальная печатная плата, используемая для изготовления интегральных схем и известная как матрица. Для разрушения или придания формы матрице с целью создания интегральных схем в промышленности используются машины высокого давления.

2.7.1. Матричное литье

Матричное литье – это производственный процесс изготовления металлической детали путём нагнетания расплавленного металла под высоким давлением в матричную полость. Результатом этого процесса является изготовление изделий с высокой степенью точности и повторяемости. В процессе литья под давлением также получаются мелкие детали, такие как текстурированная поверхность и т. д. Обычно при изготовлении деталей, таких как медь, алюминий и магний, используются в основном цветные металлы. В течение многих лет для удовлетворения определённых потребностей каждого электронного устройства использовалось множество различных сплавов.

2.7.2. Полупроводниковая пластина

Полупроводниковая пластина подобна кремниевой пластине, которая используется для создания интегральной схемы. В своё время было выявлено несколько металлов, которые в настоящее время используются в производстве интегральных схем. Вначале кремний был единственным материалом, используемым в качестве проводника тока в интегральных схемах. Позже было обнаружено, что одного кремния недостаточно. Стали использоваться и другие материалы. Для превращения крошечных участков кремния в превосходные проводники электричества с использованием меди применялись специальные химические процессы. Аналогично для обеспечения сопротивления цепей и уравновешивания протекания тока по ним использовались хорошие изоляторы, такие как стекло и резина.

Для включения или выключения протекания тока по цепи используются создаваемые в схеме транзисторы.

Наиболее распространенным примером интегральных схем, используемых в нашей повседневной жизни, являются телевизоры и радиоприёмники, имеющие печатную плату (PCB). Важным компонентом этих схем является разрядник или переключатель, который работает как электрическая карта с небольшими электронными компонентами (такими как резисторы и конденсаторы) и миниатюрными медными соединениями, соединяющими их вместе.

Простые печатные платы небольших электронных устройств могут быть изготовлены с использованием одного бита, кодируемого направлением тока. Но для сложных схем, таких как компьютер, этого недостаточно. Даже простейшему компьютеру требуется восемь электронных переключателей для хранения одного байта (символа) информации.

Изобретение транзисторов стало важной вехой на пути к производству современных интегральных схем. Три американских физика изобрели транзисторы в 1947 году. Транзисторы были в несколько раз меньше вакуумных ламп и реле, потребляли гораздо меньше энергии и были намного надёжнее. Но все ещё оставалась проблема соединения всех этих транзисторов в сложные схемы. Даже после изобретения транзисторов сложные компьютерные схемы все ещё были не очень надёжными.

Основная идея, лежащая в основе производства интегральных схем, заключалась в создании целостной схемы со всеми её многочисленными компонентами, взаимосвязанными друг с другом, и воссоздании всего этого в микроскопически крошечной форме на поверхности куска кремния. Это была новаторская идея, которая сделала возможным производство всех видов «микроэлектронных» гаджетов, от цифровых часов и карманных калькуляторов до ракет для посадки на Луну и ракет со встроенной спутниковой навигацией.

2.8. Наиболее распространённое применение интегральных схем в электронных устройствах

Наиболее часто интегральные схемы используются в мобильных телефонах и компьютерных системах. Основными компонентами этих электронных устройств являются процессор и память. Процессор выполняет все операции, требуемые от электронного устройства, а память используется для временного или постоянного сохранения результатов операций. Эти компоненты изготавливаются из особо чистых металлов, таких как кремний, который химически легирован для обеспечения различных электрических свойств.

2.8.1. Легирование полупроводников

Традиционно люди считали, что материалы, доступные в природе, можно разделить на две основные категории: те, которые позволяют электричеству проходить через них (проводники), и те, которые этого не делают (изоляторы). Металлы составляют большую часть проводников, в то время как неметаллы, такие как пластик, дерево и стекло, являются изоляторами.

Позже исследователи поняли, что все гораздо сложнее, особенно когда речь заходит об определённых элементах в середине периодической таблицы Менделеева (в группах 14 и 15), в частности о кремнии и германии. Что касается изоляторов, то легко заставить их вести себя как проводники, добавив к ним небольшое количество примесей в процессе, известном как легирование, в то время как заставить проводники вести себя как изоляторы гораздо сложнее.

Процесс изготовления интегральной схемы начинается с одного большого кристалла кремния, имеющего форму длинного толстого стержня, который "нарезается" на тонкие диски (размером примерно с компакт-диск), называемые пластинами. Пластины разделены на множество одинаковых квадратных или прямоугольных областей, каждая из которых будет составлять отдельный кремниевый чип (иногда называемый микрочипом).

Затем на каждом чипе создаются тысячи, миллионы или даже миллиарды компонентов путём легирования различных участков поверхности, чтобы превратить их в кремний n-типа или p-типа (электронной или «дырочной» проводимости). Легирование осуществляется множеством различных процессов. В одном из них, известном как «распыление», ионы легирующего материала выстреливаются в кремниевую пластину подобно пулям из пистолета. Другой процесс, называемый «осаждением из паровой фазы», включает введение легирующего материала в виде газа и его конденсацию таким образом, чтобы атомы примесей образовали тонкую плёнку на поверхности кремниевой пластины.

Производство интегральных схем, содержащих сотни, миллионы или миллиарды компонентов на кремниевом чипе размером с ноготь, является более сложным и включает в себя множество химических процессов. Вот почему интегральные схемы и полупроводники создаются в безупречных лабораторных условиях, называемых “чистыми помещениями”, где воздух тщательно фильтруется, а работники должны входить и выходить через воздушные шлюзы, надевая все виды защитных средств.

2.9. Затраты на изготовление интегральных схем

Стоимость упакованной интегральной схемы, определяется исходя из:

- стоимости матрицы (анг die);
- стоимости тестирования матрицы (Cost of testing die);
- стоимости упаковки интегральной схемы (Cost of Packaging);
- доли конечного выхода матрицы (Final test yield).

Математически стоимость интегральной схемы определяется следующим образом:

$$CostofIC = \frac{Cost\ of\ die + Cost\ of\ testing\ die + Cost\ of\ Packaging}{Final\ test\ yield}$$

Стоимость интегральной схемы рассчитывается путём сложения стоимости матрицы, стоимости тестовой матрицы, стоимости её упаковки, делённой на конечный результат тестирования или выживаемость матрицы во время тестирования. Термины «матрица» и «пластина» объяснены в предыдущем разделе.

Стоимость матрицы (die)

$$Cost\ of\ die = \frac{Cost\ of\ wafer}{Die\ per\ wafer \times Die\ yield}$$

(деление стоимости пластины на произведение числа матриц в пластине на долю выхода годных матриц).

Матрица на пластину (die per wafer)

Количество матриц на пластину составляет

$$Dies\ per\ wafer = \frac{\pi \times (wafer\ diameter/2)^2}{Die\ area} - \frac{\pi \times wafer\ diameter}{\sqrt{2 \times Die\ area}}$$

Первый член – это отношение площади пластины к площади матрицы.

Второе слагаемое компенсирует наличие зазоров в круглой области – прямоугольные матрицы могут находиться вблизи периферии круглых пластин.

Деление длины окружности на диагональ квадратной матрицы приблизительно соответствует количеству матриц на краю пластины.

Пример

Задача: Найдите количество матриц на пластине диаметром 300 мм (30 см) для матриц со стороной 1,5 см.

Решение:

- диаметр пластины = 300 мм = 30 см;
- длина стороны матрицы = L = 1,5 см;
- площадь матрицы = L2 = (1,5 см)2 = 2,25 см^2.

$$Dies\ per\ wafer = \frac{\pi \times (wafer\ diameter/2)^2}{Die\ area} - \frac{\pi \times wafer\ diameter}{\sqrt{2 \times Die\ area}}$$

$$Dies\ per\ wafer = \frac{\pi \times (30cm/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2 \times 2.25}}$$

$$Dies\ per\ wafer = \frac{706.9}{2.25} - \frac{94.2}{2.12} = 270$$

Стоимость интегральной схемы

Выход матрицы

- Выход матрицы – это доля исправных матриц на пластине.
- Дефекты распределяются случайным образом на пластине.
- Выход матрицы обратно пропорционален сложности изготовления.

Стоимость тестирования

Стоимость тестирования определяется тремя составляющими:

- стоимость тестирования в час;
- среднее время тестирования матрицы;
- доля выхода годных матриц.

Математически это выражается следующей формулой:

$$Cost\ of\ Testing = \frac{Cost\ testing\ \ per\ hour \times Average\ Die\ Testing\ Time}{Die\ Yield}$$

- стоимость тестирования составляет от 50 до 500 долларов в час в зависимости от необходимого тестера;
- тестирование матрицы занимает в среднем от 5 до 90 секунд.

Задание № 02

Решения задач, приведённых в списке, требует понимания математической формулы и элементов уравнений для определения количества и стоимости производства различных элементов компьютерной архитектуры.

1. Существует схема, обеспечивающая подачу тока на стабилизатор, подключенный к холодильнику. Постоянный ток, подаваемый на стабилизатор, составляет 20 ампер, а напряжение в два раза меньше тока. Ёмкостная нагрузка стабилизатора составляет 25, а частота переключения составляет пять раз в секунду. Определите статическую и динамическую мощности в цепи через 6 минут.

Для решения задачи требуется знание уравнений и их элементов, приведённых в подзаголовке 2.5, для расчёта статической и динамической мощности.

2. Предположим, что у нас матричное литье диаметром 30 см. Найдите выход матрицы для матрицы со стороной 1,5 см и 1,0 см, предполагая, что плотность дефектов составляет 0,3 на $см^2$, а значение α is равно 4.

Решение этой задачи приведено в подзаголовке 2.9. Решение задачи требует понимания всех элементов уравнений, упомянутых в этом разделе.

3. Найдите количество матриц на пластине диаметром 250 см для матриц 2,8 см с одной стороны и 6,0 см с другой.

Для решения задачи можно использовать математическую формулу, приведённую в подзаголовке 2.9, и пример из того же раздела, связанный с определением количества матрицы.

ГЛАВА 3. КОНТРОЛЬНЫЕ ПОКАЗАТЕЛИ (БЕНЧМАРК) И НАДЁЖНОСТЬ

3.1. Что такое контрольные показатели (бенчмарк)?

Стандарты для измерения производительности любой системы или её компонентов называются контрольными показателями или бенчмарками. Для измерения производительности системы бенчмарки выполняют анализ определённых параметров системы, таких как время выполнения, скорость передачи данных в памяти и т. д. с параметрами идеальной системы. Результаты анализа обычно представлены по шкале от 1 до 10, где 10 соответствует лучшему состоянию, а 1 – худшему. Существуют определённые приложения, доступные онлайн и оффлайн для измерения контрольных показателей для различных категорий систем, приложений, программ и алгоритмов.

Цель контрольных показателей (бенчмарк)

По мере развития компьютерной архитектуры становилось все труднее сравнивать производительность различных компьютерных систем, просто изучая их технические характеристики. Поэтому были разработаны контрольные показатели (бенчмарк), позволяющие сравнивать различные архитектуры. Например, процессоры Intel i5, как правило, работали на более высокой тактовой частоте, чем Pentium 4.

Контрольные показатели (бенчмарк) предназначены для выполнения определённого типа программ или процессов в двух разных системах или на разных компонентах одной и той же системы с целью измерения их производительности. Контрольные показатели (бенчмарк) особенно важны при проектировании центральных процессоров, предоставляя разработчикам процессоров возможность оценивать и находить компромиссы в микроархитектурных решениях.

Известно, что производители компьютеров настраивают свои системы таким образом, чтобы обеспечить нереально высокую производительность в контрольные показатели (бенчмарк) тестовых испытаниях, которые не воспроизводятся при реальном использовании. Например, некоторые компиляторы могут обнаружить определённую математическую операцию, используемую в хорошо известном тесте с плавающей запятой, и заменить её более быстрой математически эквивалентной операцией.

В настоящее время компании-компиляторы регулярно используют контрольные показатели (бенчмарк) тесты для улучшения не только своих собственных результатов, но и реальной производительности приложений.

Учитывая большое количество доступных контрольные показатели (бенчмарк), производитель обычно может найти по крайней мере один контрольный показатель (бенчмарк), который показывает, что его система будет превосходить другие системы; можно показать, что другие системы превосходят другие тесты. Производители обычно сообщают только о тех критериях (или аспектах критериев), которые показывают их продукцию в наилучшем свете. Также известно, что они искажают значение критериев, чтобы, опять же, представить свою продукцию в наилучшем свете.

В идеале контрольные показатели (бенчмарк) тесты должны заменять реальные приложения только в том случае, если приложение недоступно или слишком сложно, или дорого переносится на определённый процессор или компьютерную систему.

3.1.1. Алгоритмы

В алгоритмах бенчмарк используется для проверки соотношений времени и объёма памяти. Два алгоритма сравниваются друг с другом путем вычисления времени их выполнения. На основе времени их выполнения будет принято решение использовать эти алгоритмы в разных областях применения. Аналогичным образом также измеряется использование памяти во время выполнения.

3.1.2. Компьютерная система

Две компьютерные системы сравниваются друг с другом по производительности, с предоставлением им одинаковых входных данных. Если значения производительности, полученные двумя компьютерами на одном и том же входе, отличаются, это означает, что в системах существует некоторый разрыв в производительности. Эти типы тестов производительности используются в лабораториях для недавно изготовленных систем.

3.1.3. Компьютерное программное обеспечение

Производительность компьютерного программного обеспечения измеряется путём анализа времени его выполнения в конкретной компьютерной системе или в операционной системе. Помимо компьютерных приложений, тесты также используются в приложениях для бизнеса/коммерции и повседневной жизни.

3.1.4. Бизнес/Коммерческие приложения

В бизнес-приложениях инвесторы могут инвестировать в продукты, сравнивая их продажи и прибыль. Этот критерий может быть измерен путём расчёта и сравнения общей стоимости + чистого дохода различных продуктов.

3.2. Типы контрольных показателей

- контрольные показатели для программ реального времени (программ, в которых наиболее важен точный своевременный ответ на вводимые данные) являются первым типом контрольных показателей. У них есть выходные данные и опции, которые пользователь может выбрать во время запуска программы. Например, компиляторы, программное обеспечение для обработки текста, программное обеспечение для разных видов промышленности и т. д.
- контрольные показатели ядра используются для тех частей системы (обычно операционной системы), к которым имеет

доступ только программист. Для обычных пользователей доступ к этим программам запрещён. Программы ядра – это небольшие ключевые фрагменты реальных программ. Например, Livermore Loops и Linpack.

- контрольные показатели toy используются для программ, которые обычно содержат от 10 до 100 строк кода, и это приводит к результату, который уже известен пользователю. Например, головоломка и быстрая сортировка.
- Синтетические контрольные показатели используются для соответствия среднему профилю выполнения. Например, контрольные показатели Whetstone и Dhrys.

В общем, контрольные показатели или бенчмарк – это контрольные показатели, которые измеряют производительность системы при выполнении четко определенной задачи. Одной из наиболее успешных попыток создания стандартизированных наборов контрольных показателей приложений была SPEC (Standard Performance Evaluation Corporation), которая берет свое начало в попытках обеспечить лучшие контрольные показатели для рабочих станций в конце 1980-х годов.

3.3. Соглашение об уровне обслуживания (SLA) и цель уровня обслуживания (SLO)

Соглашение об уровне обслуживания – это официальный контракт между производителем и потребителем продукта. В нем подробно описывается, как производитель должен содействовать и предоставлять услуги потребителю в случае обнаружения неисправности в продукте. Ответ производителя подробно изложен в форме цели уровня обслуживания (SLO), гарантирующей надёжность их услуг. Производитель должен выплатить заказчику штраф, если услуги предоставляются не в соответствии с соглашением. Два общих условия, упоминаемых во всех соглашениях об уровне обслуживания – это выполнение обслуживания и прерывание обслуживания. Выполнение услуги – предоставление услуги потребителю

в соответствии с условиями соглашения. Прерывание обслуживания происходит, когда предоставляемые услуги отличаются от того, что изложено в соглашении об уровне обслуживания.

Некоторые параметры продукта, соблюдаемые производителем перед подписанием соглашения об уровне обслуживания, приведены ниже.

3.3.1. Среднее время до отказа (Mean time to failure):

Среднее время до отказа – это среднее время, в течение которого электронные компоненты могут потребовать ремонта. Это показатель надёжности модуля.

$$\text{Среднее время до отказа} = \frac{\text{общая сумма всех часов работы этих устройств (часов)}}{\text{общее количество используемых устройств}}$$

3.3.2. Среднее время на ремонт (Mean time to recover):

Среднее время ремонта – это среднее время, в течение которого компоненты могут быть отремонтированы после прерывания необходимого обслуживания.

$$\text{Среднее время ремонта} = \frac{\text{общее время ремонта (в часах)}}{\text{общее количество отказов}}$$

3.3.3. Среднее время между отказами (Mean time between failure):

Среднее время между отказами представляет собой сумму среднего времени до отказа и среднего времени восстановления и представляется в виде уравнения:

$$\textit{Среднее время между отказами (MTBF)} = \textit{Среднее время до отказа (MTTF)} + \textit{Среднее время ремонта (MTTR)}$$

Хотя широко используется промежуток времени между отказами, наиболее подходящим временем является промежуток времени до отказа, часто используются оба.

3.3.4. Сбой во времени (Failure in time, число сбоёв на единицу времени):

Это величина, обратная среднему времени до отказа. Это среднее время, в течение которого ожидается, что электронное устройство полностью перестанет работать. По статистике временных отказов видно, что отказ электронного устройства происходит через тысячи часов его работы.

3.3.5. Доступность модуля (Module Availability):

Это показатель выполнения обслуживания в отношении чередований между состояниями выполнения и прерывания. Для систем без резервирования с ремонтом доступность модуля рассчитывается с использованием приведённого ниже уравнения.

$$Module\ Availability = \frac{MTTF}{(MTTF + MTTR)}$$

3.4. Трудности контрольных показателей (бенчмарк)

Анализ контрольных показателей или бенчмарк – это дело непростое и часто требует нескольких повторений, чтобы прийти к предсказуемым и полезным выводам. Интерпретация данных сравнительного анализа также чрезвычайно сложна. Ниже приведён неполный список распространённых трудностей:

- производители, как правило, разрабатывают или настраивают контрольные показатели (бенчмарк) для своих компьютерных систем, прежде чем продавать их клиентам. Например, один из известных компьютерных контрольный показатель (бенчмарк) “Norton SysInfo” (SI) особенно прост в настройке и управлении для получения желаемых результатов. Этот контрольный показатель в основном ориентирован на одновременный расчёт скорости выполнения нескольких операций на одном компьютере, и при интерпретации таких результатов, которыми клиентам трудно манипулировать, следует соблюдать крайнюю осторожность;

- аналогичным образом, некоторые производители манипулируют результатами до такой степени, что их обвиняют в «мошенничестве» при проведении контрольных показателей (бенчмарк). Это означает временное выполнение расчётов работы системы и организацию её компонентов таким образом, чтобы это приводило к более высоким показателям для их систем. Но когда система включается в реальных условиях, производительность системы оказывается намного ниже ожидаемой;
- многие контрольные показатели (бенчмарк) полностью фокусируются на скорости выполнения вычислений, игнорируя другие важные характеристики компьютерной системы, такие как качество предоставляемых системой услуг, её безопасность, доступность, надёжность, целостность выполнения, удобство обслуживания, масштабируемость;
- в целом, контрольные показатели (бенчмарк) должны не только измерять производительность и общую стоимость производственной системы, но также должны быть направлены на выявление факторов, которые могут привести к снижению производительности системы или увеличению стоимости её производства. Иногда даже незначительное отклонение от стандартного пакета в реальных условиях приводит к значительно более высокой цене;
- контрольные показатели, используемые производителем, как правило, игнорируют требования к вычислительным мощностям для разработки, тестирования и аварийного восстановления. Поставщики предпочитают сообщать только о том, что может потребоваться для производственных мощностей, чтобы первоначальная цена приобретения казалась как можно более низкой;
- контрольные показатели (бенчмарк) системы испытывают трудности с адаптацией к широко распределенным серверам, особенно к тем, которые особенно чувствительны к структурам и типологиям сетевого распределения. Появление феномена

грид-вычислений (grid computing) в компьютерных системах усложняет проведение бенчмаркинг, поскольку некоторые рабочие нагрузки "grid computing" делает быстрее, в то время как другие – нет;

- многие контрольные показатели (бенчмарки) зависят от приложения. Это означает, что они дают лучшие результаты при выполнении конкретного приложения в компьютерной системе по сравнению с другими приложениями, запущенными в той же системе. Аналогичным образом, контрольные показатели по-прежнему не способны точно измерить производительность систем, в которых несколько приложений одновременно запущены на одном сервере.

Задание № 03

1. Предположим, что общее время работы системы составляет 10 часов в день. Но есть два сбоя, время простоя которых составляет 1 час и 2 часа. Рассчитайте среднее время между отказами (MTBF) и сбой во времени (FIT)?
2. Предположим, есть торговый автомат, работающий круглосуточно, панель которого перестала работать в 16:30 и была починена специалистом в 20:00. Позже на той же неделе дверцу заклинило, и она не работала целый день. Каков MTTR в данном случае?
3. Предположим, что в больничном учреждении имеется 125 одинаковых роликов конвейерной ленты, которые проработали в общей сложности 60 000 часов в прошлом году. Какова будет MTTF?

Для решения поставленной задачи необходимо понимание математических формул и уравнений, которые подробно описаны в подзаголовке 3.3. Эти термины используются для определения доступности различных компонентов компьютерной системы.

ГЛАВА 4. ЗАКОН АМДАЛА (AMDAHL'S LAW)

4.1. Закон Амдала (Amdahl's Law)

После создания в 1945 году первого электронного цифрового компьютера общего назначения ENIAC развитие компьютерных систем и их технологий пошло быстрыми темпами. Исследователи сосредоточились на разработке многозадачной компьютерной системы с высокой эффективностью. Было предложено несколько концепций и теорий о будущем компьютерных технологий, таких как закон Мура (Moore's law), закон Густафсона (Gastofson law), закон Амдала (Amdahl's law) и т. д. Эти законы в основном были связаны с повышением производительности компьютерных систем. Закон Амдала привлек особое внимание исследователей, поскольку он охватывает аспект параллельных вычислений в будущем.

Закон Амдала гласит, что повышение производительности за счёт добавления некоторого более быстрого компонента выполнения ограничено долей времени, в течение которого может быть использовано более быстрое решение.

Закон Амдала определяет ускорение, которого можно добиться, добавив в систему определенный компонент. Концепцию ускорения можно уточнить с помощью примера. Предположим, что к компьютеру добавлено усовершенствование, которое улучшит производительность системы всякий раз, когда оно используется.

Математическая форма:

$$\text{Ускорение} = \frac{\text{производительность системы для выполнения всей задачи с использованием улучшения}}{\text{производительность системы для выполнения всей задачи без использования улучшения}}$$

Ускорение показывает нам, насколько быстрее система будет выполнять задачу с включённым в неё улучшением по сравнению с исходным компьютером.

Результат применения закона Амдала зависит от двух следующих факторов.

1. Доля вычислительного времени на исходном компьютере, которая позже преобразуется в компьютер новой версии путём добавления новых компонентов, чтобы воспользоваться преимуществами улучшения. Доля улучшения всегда меньше или равна 1.
2. Улучшение, достигнутое благодаря расширенному режиму выполнения – это то, насколько быстрее выполнялась бы задача, если бы расширенный режим использовался для всей программы. Это значение представляет собой время работы исходного режима по сравнению со временем работы расширенного режима. Это значение всегда больше 1.

Простое ускорение времени выполнения системы при добавлении нового улучшения можно рассчитать, используя приведённое ниже уравнение.

$$\text{Ускорение} = \frac{\text{Время выполнения старое}}{\text{Время выполнения новое}}$$

Закон Амдала – это принцип

Максимальное потенциальное улучшение производительности системы ограничено той частью системы, которая не может быть улучшена. Другими словами, повышение производительности системы в целом ограничено её узкими местами.

Этот закон часто используется для прогнозирования потенциального повышения производительности системы при добавлении большего количества процессоров или повышении быстродействия отдельных процессоров.

4.2. Важность закона Амдала

Закон Амдала используется для определения максимального предполагаемого улучшения системы в целом, когда улучшается только одна часть системы. Он часто используется в параллельных

вычислениях для определения теоретического максимального ускорения при использовании нескольких процессоров. Закон назван в честь компьютерного архитектора Джина Амдала и был представлен на весенней совместной компьютерной конференции AFIPS в 1967 году. Джин Амдал, главный архитектор первой серии мэйнфреймов IBM и создатель Amdahl company и других компаний, обнаружил, что существуют довольно жесткие ограничения на то, насколько можно ускорить выполнение данной распараллеленной работы. Эти объяснения были обобщены в законе Амдала.

В современных компьютерных системах скорость выполнения или производительность системы зависит не только от производительности процессора, но и от архитектуры системы. Она также зависит от последовательности выполнения инструкций и от того, как блок управления должен вести себя в случае конфликта в распределении ресурсов между различными компонентами системы. Ниже приводятся термины, помогающие понять влияние архитектуры компьютера и организации его компонентов на производительность и время выполнения системы.

4.2.1. Качество команд (Instruction Count (IC))

За счёт добавления более мощных команд архитектура компьютера может уменьшить количество команд. Это приводит к увеличению либо тактового времени, либо такта на команду (CPI), либо и того, и другого.

4.2.2. Циклы для каждой команды (CPI)

Цикл на команду определяет аспект производительности процессора в компьютерной архитектуре. Для программы это среднее количество тактовых циклов на команду, которые можно увидеть ниже, на рис. 3.

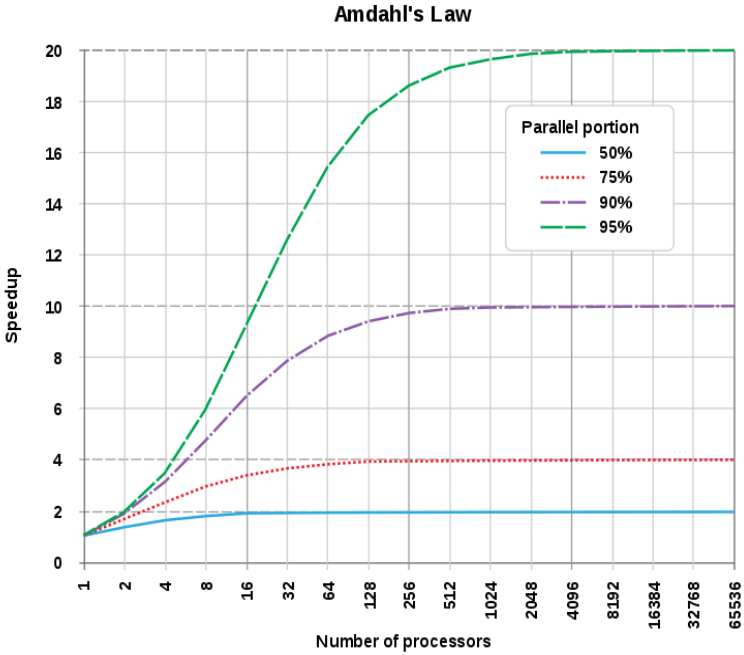

Рис. 3. Зависимость между количеством процессоров и ускорением, основанная на законе Амдала (взято из https://en.wikipedia.org/wiki/Amdahl%27s_law по теме закон Амдала).

4.2.3. Время работы центрального процессора (CPU)

Количество времени, в течение которого мы использовали центральный процессор для выполнения команд компьютерной программы.

$$CPU\ TIME = USER\ TIME + SYSTEM\ TIME$$

ПРОЦЕССОРНОЕ ВРЕМЯ = ПРОЦЕССОРНОЕ ВРЕМЯ ПОЛЬЗОВАТЕЛЯ + СИСТЕМНОЕ ПРОЦЕССОРНОЕ ВРЕМЯ

Математически время центрального процессора равно времени, затрачиваемому в компьютерной системе на ввод данных, и времени, затрачиваемому компьютерной системой на обработку входных данных и генерирование некоторых результатов на основе входных данных.

4.2.4. Время тактового цикла

По сути, все компьютеры сконструированы с использованием тактовых импульсов, работающих с постоянной частотой. Эти дискретные временные события называются тиками, тактовыми тиканьями, периодами тактирования, тактовыми часами или тактовыми циклами.

Мы можем определить тактовый цикл как “Время между двумя соседними паузами генератора”, а тактовую частоту можно определить, как количество этих импульсов в секунду. Тактовая частота может измеряться в мегагерцах (МГц). Некоторые процессоры могут выполнять только одну команду за цикл, но продвинутые процессоры имеют возможность выполнять более одной команды за цикл.

4.2.5. Уравнение производительности процессора:

Время центрального процессора (CPU)

= Общее количество тактовых циклов × время тактовых циклов.

или

Время центрального процессора (CPU) =Общее количество тактовых циклов / тактовая частота.

Общее время центрального процессора равно общему количеству тактовых циклов и времени такта.

4.3. Увеличение и ускорение фракции:

Компьютерные архитекторы часто используют термин «ускорение», чтобы объяснить, как меняется производительность архитектуры по мере внесения изменений в архитектуру. Ускорение – это соотношение времени выполнения до и после внесения изменений.

$$\textit{Ускорение} = \frac{\text{время выполнения перед улучшением}}{\text{время выполнения после улучшения}}$$

Пример: Если выполнение программы в старой версии архитектуры занимает 30 секунд, а в новой версии – 15 секунд, то общее время

Ускорение = 30 секунд/15 секунд = 2,0.

Время выполнения Новое =

$$\text{Время выполнения Старое} \times \left[\text{Неиспользованная доля} + \frac{\text{Неиспользованная доля}}{\text{используемое ускорение}}\right]$$

В приведённом выше уравнении неиспользуемая доля – это доля времени, в течение которого разработка не используется, используемая доля – это доля времени, в течение которого используется улучшение, а используемое ускорение – это ускорение, которое происходит при использовании улучшения.

Примечание: если используемая и неиспользуемая доли вычисляются с использованием времени выполнения до того, как модификация станет полезной, это приведёт к неправильному результату.

4.3.1. Преимущества закона Амдала

- Закон позволяет количественно оценить максимальное потенциальное ускорение, которое может быть достигнуто за счёт параллельного разделения программы, что может помочь

при принятии решений о проектировании аппаратных и программных компонентов компьютерной системы.

- Это также помогает идентифицировать части программы, которые нелегко разделить параллельно. Таким образом, для оптимизации программы эти части программного кода требуют особых усилий и внимания со стороны программиста.
- Обеспечивает основу для понимания компромиссов между параллельным разделением и другими формами оптимизации, такими как оптимизация кода, ресурсов, алгоритмические улучшения и управление памятью.

4.3.2. Недостатки закона Амдала

- Согласно закону Амдала, всегда существует некоторая часть программы, которая не может быть параллельно разделена, и эта часть кода представлена в математическом выводе закона Амдала. В действительности, если мы предположим, что часть программы, которая не может быть разделена, исправлена, мы можем оптимизировать код или заменить эту часть кода кодом, который может быть параллельно разделен. Это делает закон Амдала менее точным.
- Согласно закону Амдала, все процессоры имеют одинаковые характеристики производительности, что может быть не так на практике. Например, в гетерогенной вычислительной среде некоторые процессоры могут быть быстрее других, что может повлиять на потенциальное ускорение, которого можно достичь.
- В нем не учитываются другие факторы, которые могут повлиять на производительность параллельных программ, такие как накладные расходы на связь и балансировку нагрузки. Эти факторы могут повлиять на фактическое ускорение, достигаемое на практике, которое может быть ниже, чем предсказывается законом Амдала.

Закон Амдала и будущее компьютерной архитектуры

В последние годы появление многоядерных процессоров изменило ландшафт параллельных вычислений. Традиционные одноядерные процессоры больше не используются в компьютерных системах. Теперь даже каждый телефон оснащен несколькими ядрами, каждое из которых способно выполнять задачи независимо. Это изменение сделало оптимизацию параллельного разделения еще более важной.

Чтобы в полной мере использовать многоядерные процессоры, разработчики программного обеспечения должны адаптироваться, разделяя параллельно свои приложения. Это требует не только глубокого понимания концепций параллелизма, но и использования передовых библиотек и платформ параллельного программирования.

Одним из фундаментальных принципов, который подчеркивает сущность параллельного разделения, является закон Амдала. Это служит путеводной звездой, напоминающей нам о том, что ускорение работы программы за счет параллельного разделения неразрывно связано с последовательной частью кода.

В последние годы использование многоядерных процессоров возросло. Тенденция к разработке процессоров с несколькими ядрами обусловлена стремлением к повышению производительности параллельных программ при меньшем потреблении энергии. В настоящее время анализ показателей ускорения и энергопотребления играет ключевую роль для приложений, выполняемых в многоядерных системах. По этой причине важно анализировать производительность на основе новых характеристик современных процессоров, таких как технология Intel turbo boost. Эта технология позволяет увеличить частоту многоядерных процессоров Intel.

Закон Амдала был в основном предложен для компьютеров с одноядерными процессорами. Современные компьютеры больше не основаны на одноядерных процессорах. На самом деле они используют преимущества многоядерных процессоров, распределяя одну и ту же работу между разными процессорами одновременно,

чтобы повысить эффективность компьютерной системы. Чтобы закон Амдала оставался актуальным в современных технологиях, исследователи работают над расширением закона Амдала, которое может работать с многоядерными процессорами.

Задание № 04

1. Предположим, мы рассматриваем возможность добавления компонента или усовершенствования, которое будет работать в 10 раз быстрее, чем исходная машина, но будет использоваться только в 40 % случаев. Каково общее ускорение, достигнутое за счёт внедрения усовершенствования?

2. Выполнение задач в компьютерной системе занимает 2 минуты. Сколько времени потребуется для выполнения тех же задач, если мы добавим новый компонент, который может ускорить выполнение задачи на 50 %, в 4 раза по сравнению с первоначальным?

Для ответа на первое и второе задание необходимо понимание математической модели закона Амдала, описанной в подзаголовки 4.2 и 4.3 вместе с примером.

3. Можем ли мы по-прежнему применять концепции закона Амадала к процессорам современных компьютерных систем. Подкрепите аргументы примерами.

Ответ на эту задачу кроется в преимуществах и недостатках закона Амдала, упомянутых в последнем разделе подзаголовка.

4. Почему закон Амдала получил большую популярность и признание по сравнению с другими законами, предложенными в то же время, такими как закон Мура, закон Густафсона?

Объяснение в подзаголовках 4.1 и 4.2 может помочь сформулировать правильный ответ на это задание.

ГЛАВА 5. АРХИТЕКТУРА НАБОРА КОМАНД

5.1. Архитектура набора команд (ISA)

Архитектура набора команд является важной частью компьютерной архитектуры, связанной с программированием, содержащей собственный тип данных, инструкции, режимы адресации (например, от регистра к регистру), память, прерывания (вызываемые процессором), обработку исключений и периферийный ввод/вывод. Типичный цикл работы процессора (рис. 4):

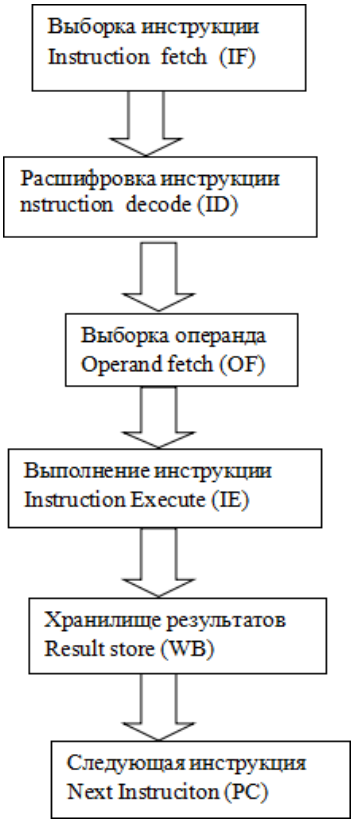

Рис. 4. Различные фазы выполнения команды
в конвейерном процессоре

5.2. Классы архитектуры набора команд

Существуют различные классы архитектуры наборов команд, перечисленные ниже.

5.2.1. Архитектура «память к памяти»

В этой архитектуре команда во время её выполнения можно напрямую обращаться к памяти. Основным недостатком этого подхода является то, что для него требуется больший размер временной памяти, а поскольку доступ к данным осуществляется из памяти во время выполнения команды, требуется больше времени для доступа к этой памяти.

5.2.2. Архитектура «регистр к регистру»

В этой архитектуре регистры используются для временного хранения данных и результатов операции после выполнения инструкции.

Эта архитектура быстрее, чем архитектура «память к памяти», поскольку регистры расположены близко к арифметико-логическому блоку и блоку управления компьютерной системой. Однако это увеличивает стоимость системы, поскольку регистры являются дорогостоящим компонентом.

5.2.3. Архитектура «аккумулятора»

Аккумулятор – это специальный регистр, который содержит адрес следующей команды, находящейся в очереди готовности и будет выполнена центральным процессором следующей.

5.2.4. Архитектура «стека»

В стековой архитектуре переменная и результаты после выполнения инструкции хранятся в стеке. Это не регистр, но он действует как регистр с механизмом first in last out (FILO).

5.3. Выравнивание и рассогласование памяти

Выравнивание – это расположение данных в памяти, которое решает проблему доступа к данным из памяти. Сначала мы должны концептуализировать основную память как непрерывный блок последовательных ячеек памяти. Каждая ячейка содержит определённое количество битов.

К данным из памяти можно получить доступ из их местоположения, используя их адрес в памяти. Этот адрес является наименьшей единицей памяти компьютера.

Пример: ниже, на рис. 5, представлена память размером 32 бита (4 байта).

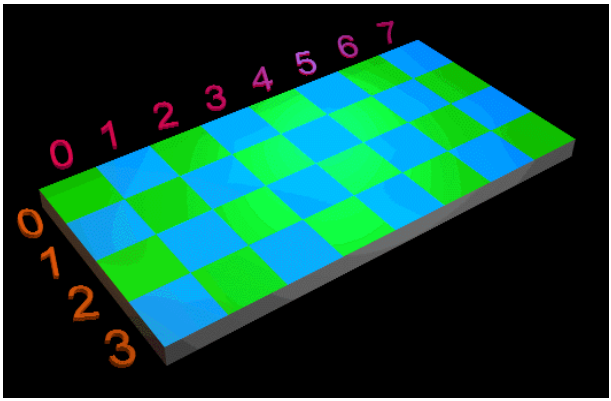

Рис. 5. Память размером 32 бита (4 байта)

Из приведённого выше рисунка видно, что:

- каждая строка представляет собой ячейку с фиксированным размером в восемь бит, помеченную от нуля до семи;
- 4 равные строки по 8 бит занимают 32 бита памяти. Смещение памяти происходит, когда данные передаются между двумя компьютерами, имеющими различную архитектуру памяти и размер слова;
- смещение памяти затрудняет доступ к памяти, что приводит к медленному выполнению программы.

- некоторые компьютерные системы или машины допускают смещение доступа к памяти, в то время как другие не допускают;
- процессор не может справиться с рассогласованием памяти, поскольку это чисто компьютерная проблема аппаратного обеспечения и архитектуры.

Выравнивание и рассогласование при передаче данных между системами, имеющими различную архитектуру

Очень часто передача данных происходила между двумя компьютерными системами с разной компьютерной архитектурой, объемом памяти и расположением блоков памяти для хранения данных. Несоответствие показателей памяти, которое происходило между системами с разной архитектурой, можно увидеть в таблице ниже.

Таблица 1

Различные типы памяти в компьютерных системах

Адрес объекта	Память выравнивания	Смещение памяти
Байт (Byte)	1,2,3,4,5,6,7	Никогда
Half word	0,2,4,6	1,3,5,7
Word	0,4	1,2,3,5,6,7
Double Word	0	1,2,3,4,5,6,7

Из приведённой выше таблицы видно, что некоторые данные передаются из системы с архитектурой "Halfword" в систему с байтовой архитектурой, тогда ячейки памяти 1,3,5 и 7 в байтовой архитектуре останутся пустыми и смещёнными. Это связано с тем, что данные, поступающие из half word, будут пытаться корректироваться по аналогичным адресам памяти 0, 2, 4 и 6, как это было в системной архитектуре Halfword. Такая же проблема существует и для других архитектур. Возможное смещение памяти для каждой архитектуры указано выше в таблице.

5.4. Режим адресации памяти

Режим адресации – это способ выделения операционной системой ячеек памяти или блоков для данных и инструкций.

Типы режимов адресации

Существует четыре распространённых типа режимов адресации.

a) Режим адресации регистра (Register addressing mode).

В режиме адресации регистра регистр является источником данных. В инструкции явно указывается имя регистра, из которого данные должны быть считаны или записаны после выполнения операции.

Пример: ADD R3, R4, R1 (данные в регистрах R4 и R1 суммируются и затем сохраняются в регистре R3).

b) Режим прямой адресации (Direct addressing mode).

В режиме адресации этого типа указывается адрес памяти, с которого данные должны быть считаны или записаны.

Пример: MOV B, 45H (Команда переместит контекст из ячейки памяти 45H в аккумулятор регистра B).

c) Режим непрямой адресации регистра (Indirect addressing mode).

Как следует из названия, данные хранятся в регистре косвенно. Инструкция содержит адрес регистра, из которого данные должны быть считаны.

Пример: MOV B, @ R0 (Эта инструкция переместит данные в накопитель из регистра, адрес которого хранится в R0).

d) Режим моментальный адресации (immediate addressing mode).

В этом типе режима адресации данные, которые будут использоваться для выполнения инструкции, упоминаются или даются в инструкции.

5.5. Что такое архитектуры CISC и RISC?

CISC

Complex instruction set computer (CISC) – это архитектура, которая использовалась для программирования операционных систем ранних компьютеров. Вначале единственным доступным языком компьютерного программирования был язык ассемблера. Аппаратные компоненты компьютеров программировались с использованием машинных кодов и машинных инструкций.

Разработчики центрального процессора пытались создать инструкции, которые можно было бы использовать для выполнения множества задач. С развитием языков более высокого уровня компьютерные архитекторы начали разрабатывать инструкции операционной системы и ядра для непосредственной реализации разнообразных механизмов языков более высокого уровня в операционной системе.

В то время дизайн аппаратного обеспечения был более зрелым, чем дизайн компилятора. Разработчики компьютеров также склонны реализовывать часть функциональности в аппаратном обеспечении, а не только в компиляторе с принудительным использованием памяти. Медленный доступ к памяти подтолкнул разработчиков к созданию инструкций, которые используют меньший объем доступа к памяти.

RISC

Reduced Instruction Set Computer (RISC) – это тип архитектуры микропроцессора, в котором используется высоко оптимизированный набор инструкций, а не дополнительный специализированный набор инструкций, часто используемый в других типах архитектур.

Основное различие между дизайном RISC и CISC заключается в количестве и сложности инструкций. Конструкции CISC включают сложные наборы команд, которые непосредственно поддерживают операции и структуры данных, используемые языками более высокого уровня.

Для выполнения нескольких инструкций одновременно используется механизм pipelining (конвейерная обработка в компьютерной архитектуре).

Pipelining (Конвейерная обработка в компьютерной архитектуре)

Это метод, который позволяет выполнять несколько инструкций одновременно. Инструкции разделены на вложенные инструкции для их параллельного выполнения. Каждая инструкция выполняется в разные фазы, и эти фазы разных инструкций выполняются параллельно.

5.6. Преимущества и недостатки CISC и RISC

CISC

Преимущества:

- во время своей первоначальной разработки CISC-машины использовали доступные технологии для оптимизации производительности компьютера;
- микропрограммирование (для реализации команд CISC) легко реализуется на языке ассемблера и обходится дешевле, чем жёсткая проводка в блоке управления;
- простота микрокодирования новых инструкций позволила разработчикам сделать машины CISC полностью совместимыми: новый компьютер мог запускать те же программы, что и более ранние компьютеры, поскольку новый компьютер содержал бы дополнительный набор инструкций более ранней версии компьютера.

Недостатки:

- вначале в каждой новой версии компьютеров существовало огромное разделение или разница между функциями и компонентами. Наборы команд и аппаратные чипы становятся все более сложными с каждым поколением компьютеров;
- CISC позволял хранить в памяти инструкции, которые могли быть практически любой длины. Это означает, что для выполнения разных инструкций потребуется разное количество тактового времени. Это замедляет общую производительность машины.

- Несколько специфических инструкций использовались нерегулярно. Согласно статистике, почти 18 % существующих инструкций в архитектуре CISC не использовались в классическом программировании.
- Инструкции CISC обычно устанавливают коды условий для каждой инструкции. Установка кодов условий не только требует времени, но и программисты должны помнить о необходимости изучения битов кода условия до того, как последующая команда изменит их.

RISC

Преимущества:

Скорость выполнения инструкции

Базовый и простой набор команд позволяет создавать конвейерный суперкомпьютер. Процессорам RISC часто удаётся выполнять команды в 2–4 раза быстрее, чем процессорам CISC, использующим аналогичную полупроводниковую технологию и те же тактовые частоты.

Более простое оборудование

Конструкция набора команд RISC-процессора очень проста. Он занимает меньше места на чипе и предоставляет дополнительные функции, такие как управление памятью и арифметические операции с плавающей запятой. Микросхемы меньшего размера позволяют производителю полупроводников размещать больше деталей на одной кремниевой пластине, что снижает стоимость каждого чипа.

Более короткий цикл проектирования

Процессоры RISC проще, чем процессоры CISC. Они могут быть спроектированы быстрее и могут использовать преимущества новейших доступных технологий.

Недостатки:

Переход от стратегии проектирования CISC к стратегии проектирования RISC не обходится без проблем. Основная проблема заключается в переносе кода с архитектуры CISC на архитектуру RISC и обеспечении совместимости обеих архитектур друг с другом.

Задание № 05

1. Какие вопросы следует учитывать при проектировании архитектуры наборов команд?
2. Каковы различные этапы выполнения команды компьютером? Почему важна последовательность выполнения на каждом этапе?
3. Какая архитектура лучше – CISC или RISC? Объясните с аргументами.
4. Какова схема распределения битов по объёму памяти, используемая в современных компьютерных системах? Как информация из системы, разработанной на основе старой архитектуры и схемы памяти, хранится в современной компьютерной системе?
5. Какова цель архитектуры наборов команд?

Чтобы ответить на эти вопросы, необходимо разобраться в каждом подзаголовке и разделе данной главы. Все темы расположены в таком порядке, который поможет разобраться во всей информации об архитектуре набора команд.

ГЛАВА 6. MIPS

6.1. Микропроцессоры MIPS. История создания

В 1981 г. команда Стендфордского университета, возглавляемая Джоном Хенесси начала работу над проектом, который впоследствии привёл к появлению первого MIPS-процессора.

Базовая концепция заключалась в значительном повышении производительности за счёт существенного упрощения архитектуры процессора, в основу которой была положена идея конвейеризации. При этом была решена проблема блокировок или вынужденных остановок конвейера, называемых interlocks, которая считалась главным препятствием распространению идеи конвейерного вычисления. Именно это свойство и дало название архитектуре MIPS (Microprocessor without Interlocked Pipeline Stages).

Такая идеология потребовала исключить много полезных инструкций, требующих нескольких тактов на выполнение, однако общая производительность системы существенно увеличилась за счёт повышения рабочей частоты процессора.

6.2. Принципы проектирования MIPS

- Инструкции меньшего размера работают быстрее. В архитектуре MIPS используется всего 32 регистра для выполнения различных арифметических и логических операций.
- Хороший дизайн может привести к хорошим компромиссам между ценой и производительностью.
- MIPS использует тот же механизм для аналогичных типов команд, что позволяет им быстро выполняться.
- Поскольку команды просты и выполняются в обычном порядке, это помогает избежать проблем, связанных с блокировкой между различными инструкциями.

6.3. Дизайн и характеристики архитектуры MIPS

- Классификация архитектуры наборов команд в MIPS основана на регистрах, памяти, стеке, аккумуляторе.
- Используются явные операнды 0, 1, 2 или 3.
- Обычно используемые режимы адресации памяти для операндов – регистровый, непрямой, косвенный.
- Обычно используются следующие типы и размеры операндов: Byte, int, float, double, string, vector. Обычно используются для выполнения инструкций для таких операций, как сложение, вычитание, умножение, перемещение, сравнение.

6.4. Регистры MIPS и их использование

В архитектуре MIPS используется 32 регистра. Каждый регистр имеет определённый номер и имя. Регистры выполняют определённую функцию. Одни используются для хранения значений операндов, другие используются для подсчёта количества раз, когда выполняется цикл инструкций, а третьи имеют различную функциональность, как указано ниже в таблице.

Таблица 2

Различные регистры архитектуры MIPS и их функциональные возможности

Регистрационный номер	Регистрационное имя	Использование регистра
O	Zero	Всегда содержит ноль
1	$at	Зарезервировано для ассемблера
2–3	$v0-v1	Вычисление выражения и процедура, возвращающая значение
4–7	$ao-a3	Аргументы
8–15	$t0-t7	Это временные регистры

16–23	$s0-s7	Регистры для сохранения значений
24–25	$t8-t9	Временные регистры для сохранения дополнительных значений
26–27	$k0-k1	Зарезервировано для ядра операционной системы
28	$gp	Глобальный указатель
29	$sp	Указатель стека
30	$fp	Указатель фрейма
31	$ra	Обратный адрес

6.5. Архитектура MIPS Язык ассемблера и машинный язык

Язык ассемблера MIPS для всех команд записан в 32-разрядных регистрах. Двоичный размер всех команд MIPS равен 32 битам. Он фиксирован и одинаково для каждого типа команд.

Наиболее распространёнными типами команд, реализованными в архитектуре MIPS, являются:

1. Арифметическая команда.
2. Команда перемещения данных между памятью и регистрами.
3. Команда, выполнение которой зависит от некоторого условия.
4. Безусловная команда, такая как команда перехода. Она всегда будет выполняться в свой ход.

Арифметическая команда

Сложение двух чисел

c=x + y на языке C

ADD $1,$2,$3 в архитектуре MIPS это означает добавить $2 и $ 3 и сохранить в $1.

Команда передачи данных

Для передачи слова из памяти в регистр используется команда lw (load word instruction).

lw $2, 100 ($3) в архитектуре MIPS это означает переход к регистру R2, значению, которое существует по адресу, найденному путём добавления смещения 100 памяти со значением регистра 3, память[100 + R3].

Условная команда перехода

В языке ассемблера, MIPS используются команды условного ветвления (conditional branch instruction).

Для использования **branch if equal** (команда перехода при условии равенства) ключевого слова, используется BEQ.

Для использования branch if not equal (команда перехода при условии неравенства) ключевого слова, используется BNE.

Безусловная команда перехода

Для безусловных переходов используются JR (jump) – команды для перехода к регистру или ячейке памяти.

6.6. Машинный язык

Машинный язык – это язык низкого уровня, в котором команды записываются символами 0 и 1. Всякий раз, когда команда написана на языке высокого уровня, она преобразуется в язык ассемблера. Затем из языка ассемблера в понятную машине форму, называемую машинным языком**.**

Для команд MIPS машинный язык всех команд состоит из 32 бит. Для обозначения машинного языка используются разные форматы. В основном это три формата использования машинного языка в архитектуре MIPS:

1. формат I;
2. формат R;
3. формат J.

6.7. MIPS Типы команд

MIPS: Microprocessor without Interlocked Pipeline Stages

Каждый формат используется для определённого типа инструкций. Например, формат R используется для применения арифметической операции к значениям, расположенным в регистрах. Формат I используется для непосредственных значений и в большинстве случаев для перемещения, загрузки или сохранения данных из ячеек памяти. Формат J используется для команд перехода. Размер различных форматов инструкций и распределение битов в них объясняется ниже.

6.7.1. Формат I команд

Команда I-типа содержит 4 поля: 16-разрядное непосредственное поле (непосредственное значение или адрес), два 5-разрядных адреса регистра, адрес регистра назначения (rt), адрес регистра источника (rs) и 6-разрядный код операции (opcode). Код операции представляет операцию, которая должна быть выполнена командой, он уникален для каждого типа операции. Команды I-типа в основном используются для команд условного перехода, таких как переход на равные условия, переход на не равные условия и т. д.

6	5	5	16
Operation	Rs	Rt	Immediate

6.7.2. Формат J команд

26-разрядный адрес места назначения перехода (target) (от 0 до 25) и 6-разрядный код операции. Команды формата J используются для команд безусловного перехода (таких как: команды перехода, команды переключения, вызовы процедур и т. д.).

6	26
Operation	Target

6.7.3. Формат R команд

Команда R-типа содержит 6 полей: 6-битный код функции (funct), 5-битную величину сдвига (shamt), три 5-битных адреса регистра (rd, rt, rs) и 6-битный код операции (opcode), который всегда равен нулю. Команды R-типа используются для арифметических и логических операций.

6	5	5	5	5	6
Operation	Rs	Rt	Rd	shift	Function

Operation Operation code (Код операции).
Rs Resource register specifier (Спецификатор регистра ресурсов).
Rt Resource /destination register specifier (Спецификатор регистра ресурса /назначения).
Immediate immediate,branch,or address displacement (ветвление или смещение адреса).
Target jump target address (целевой адрес перехода).
rd destination register specifier (спецификатор регистра назначения).
Shift shift amount (величина сдвига).
Function ALU/ shift function specifier (Спецификатор функции ALU/ shift).

Пример

Программа на языке С для рядов Фибоначчи. Ее преобразование в MIPS-ассемблер, а затем в машинный язык.

С Программ:

```
Intfib(void)
{
        Int n=8;
        Int f1=1,f2=2;
        While(n!=0)
```

```
        {
            f1=f1+f2;
            f2=f1-f2;
            n=n-1;
        }
return f1;
}
```

Язык MIPS ассемблера:

```
fib:addi $3,$0,8;
addi $4,$0,1;
addi $5,$0,2;
loop:beq $3,$0,end
add $4,$4,$5;
sub $5,$4,$5;
subi $3,$3,1;
jloop;
end;
SW $4,255($0)
```

Двоичный файл MIPS программы:

Binary of loop: beq $3,$0,end J->Format

6bits	5bits	5bits	16 bits
Op	Rs	rt	Offset
000100	00011	00000	000000000000101

Binary of add $4,$4,$5 R->Format

Op	Rs	Rt	rd	shant	Funct
00000	00100	00101	00000	00000	100000

Instruction binary encoding I->format

	Op	Rs	rt	Offset
addi $4,$0,1	001000	00000	00100	000000000000001
addi $5,$0,2	001000	00000	00101	0000000000000010

Instruction sub $5,$4,$5 R->Format

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Op	Rs	Rt	rd	shant	funct
000000	00100	00101	00101	00000	100010

Задание № 06

1. Напишите программу на компьютерном языке C/C++, чтобы узнать, является ли число чётным или нечётным? Преобразуйте его в программу MIPS и создайте программу на бинарном языке для компьютерной системы.

2. Напишите программу, чтобы извлечь квадратный корень из числа. Преобразуйте его в программу MIPS и создайте программу на бинарном языке для компьютерной системы.

   Для решения задач 1 и 2 студент должен обладать знаниями в области программирования на C/C++ и понимать язык ассемблера. Пример, приведённый в подзаголовке 6.7, облегчает написание программного кода для архитектуры MIPS.

3. Почему для понимания компьютера всегда требуется высокоуровневый программный код в двоичном формате и какую роль в этом преобразовании играет язык ассемблера?

4. Чем архитектура MIPS известна и почему используется до сих пор? В чем преимущества использования архитектуры MIPS?

Для решения задач 3 и 4 требуется понимание теоретической информации об архитектуре MIPS, упомянутой в подзаголовках 6.3, 6.4 и 6.5.

ГЛАВА 7. КОНВЕЙЕРНАЯ ОБРАБОТКА КОМАНД, ЕЁ ПРОБЛЕМЫ И БУДУЩЕЕ

7.1. Что такое конвейерная обработка команд (pipelinning of instructions)?

- Это метод реализации, при котором несколько инструкций выполняются параллельно во время их выполнения.
- Процесс выполнения инструкции разделен на разные фазы.
- Каждая фаза работает параллельно другим фазам во время выполнения инструкции.
- Цель конвейерной обработки – ускорить работу центрального процессора при выполнении инструкций.
- Важными терминами, используемыми при конвейерной обработке, являются время цикла процесса и время тактового цикла.
- Время цикла процесса – это время, необходимое процессу для перехода от одной фазы конвейера к другой.
- В компьютерном процессоре время цикла считается равным одному такту за цикл.
- Целью проектировщика конвейера является поддержание баланса между каждой фазой конвейера.

Конвейерная обработка сокращает среднее время выполнения каждой инструкции. Доступны различные версии конвейерной обработки с различным количеством фаз. Наиболее распространенная и широко используемая в процессоре конвеерная обработка состоит из пяти этапов или фаз выполнения инструкции. Этими фазами являются выборка команд, декодирование команд, выборка операндов, выполнение команд и обратная запись.

7.2. Характеристики конвейерной обработки

Характеристики:

Конвейер использует преимущества параллелизма, который существует между операциями, необходимыми для выполнения инструкции. Этапы фаз конвейерной обработки соединяются друг

с другом для выполнения инструкции. Инструкция поступает в конвейер из одной фазы, переходит к другим фазам и завершается на последней фазе. Простейшая формула, используемая для понимания эффективности конвейера при выполнении инструкции, такова:

$$\text{время, требуемое для выполнения инструкции} = \frac{\text{время, требуемое для выполнения инструкции на машине с не конвейерным управлением}}{\text{количество фаз конвейерной машины}}$$

7.3. Компоненты/фазы конвейерного процессора

Этими пятью этапами конвейерного процессора являются:

- Выборка инструкций (Instruction Fetch)
- Декодирование инструкций и выборка регистра (Instruction Decode and fetching of register)
- Выборка операнда (Operand Fetch)
- Выполнение инструкции (Instruction Execute)
- Обратная запись (Write back)

Выборка инструкций (Instruction Fetch):

На этом этапе команда, которая должна быть выполнена следующей, извлекается из памяти. Адрес памяти следующей команды всегда находится в программном счётчике (Program Counter). Когда операция выборки завершена, у PC есть адрес памяти следующей команды, которая будет извлечена в следующем цикле.

Декодирование инструкций (Instruction Decode/fetching of register):

В следующем цикле команда, полученная на предыдущем этапе, декодируется процессором. Здесь анализируются тип и формат команды. Команда может быть арифметической командой, командой передачи данных, логической командой или каким-либо другим типом команды.

Выборка операнда (Operand Fetch):

Здесь, на этом этапе, операнды в инструкции, идентифицированные во время декодирования инструкции, извлекаются из памяти или из регистров для выполнения над ними операции, записанной в инструкции.

Выполнение инструкции (Instruction Execute):

Если команда является арифметической, то арифметическая операция выполняется над значением операнда, полученным на предыдущем этапе. Аналогично, если это логическая команда, то выполняется логическая операция и т. д.

Обратная запись (Write Back Cycle):

После выполнения инструкции результаты операций сохраняются в регистре или в памяти в зависимости от того, что указано в инструкции.

Инструкции и их выполнение в конвейере можно увидеть на таблице ниже. Этапы конвейерной обработки представлены их сокращениями.

Таблица 3

Параллельное выполнение различных команд в конвейерных процессорах

Инструкция #	1	2	3	4	5	6	7	8	9
Инструкция I	**IF**	**ID**	**OF**	**IE**	**WB**				
Инструкция i+1		**IF**	**ID**	**OF**	**IE**	**WB**			
Инструкция i+2			**IF**	**ID**	**OF**	**IE**	**WB**		
Инструкция i+3				**IF**	**ID**	**OF**	**IE**	**WB**	
Инструкция i+4					**IF**	**ID**	**OF**	**IE**	**WB**

Из приведённой выше таблицы видно, что существует пять инструкций, которые должны быть выполнены с использованием пятифазного конвейерного механизма. Каждой фазе требуется 1 такт для выполнения своей функции. Все инструкции выполняются параллельно. Когда процессор начинает выполнять инструкции, первая инструкция переходит в первую фазу конвейерной обработки в течение первого такта. Как упоминалось ранее, первой фазой конвейерной обработки инструкций является выборка команд. В течение 2-го тактового цикла первая инструкция переходит ко второй фазе конвейерной обработки инструкций. Вторая фаза конвейера называется декодированием команд. В то же время, когда первая команда переходит во вторую фазу выполнения, другая команда входит в пустую первую фазу конвейера, которая является выборкой команд.

Это означает, что в течение 2-го такта параллельно выполняются две операции. Одна команда декодируется, в то время как другая извлекается процессором. То же самое продолжается с другими командами. Из приведённого выше рисунка можно наблюдать, что в течение 5-го такта процессор выполняет максимум, поскольку он выполнил 5 различных команд одновременно. При использовании этого механизма производительность процессора увеличилась. Увеличение производительности процессора для выполнения всех команд можно рассчитать по формуле, приведённой ниже:

Уменьшение времени выполнения инструкций

$$= \frac{\text{время выполнения инструкций с конвейерным процессором}}{\text{время выполнения инструкций без конвейерного процессора}} \times 100$$

$$\text{Уменьшение времени выполнения инструкций} = \frac{5 \times 5}{9} \times \boldsymbol{100}$$

$$\boldsymbol{= 227\%}$$

7.4. Проблемы конвейерных процессоров:

Проблемы при конвейерной обработке процессоров возникают, когда результаты выполнения инструкций зависят друг от друга или, когда инструкции используют некоторые общие операнды.

Не конвейерный процессор завершает выполнение каждой инструкции перед началом выполнения другой инструкции. Это означает, что зависимости между инструкциями обычно не влияют на время выполнения программы на не конвейерном процессоре.

На конвейерном процессоре вычисление времени выполнения программы является более сложным, поскольку зависимости между инструкциями влияют на время выполнения программы.

7.5. Типы проблем с конвейером команд:

Существует три распространённых типа проблем с конвейером команд или конфликтов, возникающих при выполнении инструкций:

1. Конфликты данных.
2. Конфликты структурные.
3. Конфликты управления.

7.5.1. Конфликты данных:

Конфликт данных возникает, когда разные инструкции, которые совместно используют ресурс данных во время их выполнения, изменяют данные на разных этапах конвейерного процессора. Конфликт данных может привести к состоянию гонки (англ. race condition), известной терминологии, используемой, когда возникает конфликт ресурсов между двумя инструкциями. Это рассмотрено ниже на рис. 6.

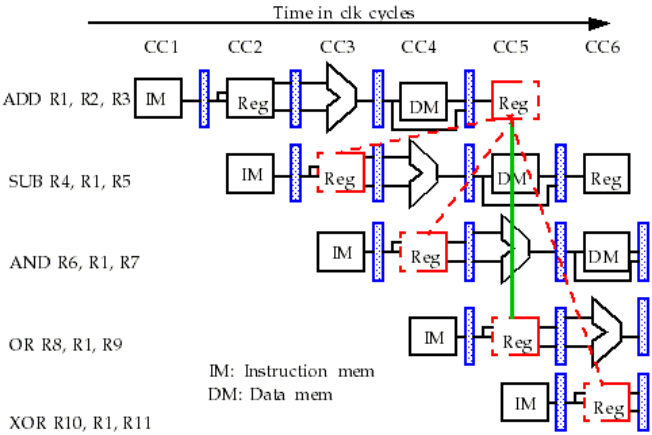

Рис. 6. Конфликт данных между различными командами, выполняющимися на конвейерном процессоре (взято из https://ece-research.unm.edu/jimp/611/slides/chap3_3.html по теме конфликты конвейерной обработки)

В этой версии конвейера, продемонстрированной выше, фазы имеют разные названия, но механизм выполнения инструкции тот же. Для выполнения второй, третьей, четвёртой и пятой инструкций требуется значение регистра R1. Фактическое значение регистра R1 может быть доступно только после полного выполнения первой команды в течение 5-го такта.

При совместном использовании данных или операнда могут возникнуть три различные проблемы.

1. **Чтение после записи (Read after write (RAW))**
2. **Запись после чтения (Write after read(WAR))**
3. **Запись после записи (Write after write(WAW))**

Мы можем объяснить это, рассмотрев две инструкции I1 и I2 таким образом, что I1 появляется перед I2 во время выполнения программы.

Чтение после записи (RAW):

Проблема чтения после записи данных относится к проблеме, когда инструкция ссылается на результат, который не был известен или вычислен. Это происходит, когда одна инструкция выполняется после другой инструкции, но предыдущая инструкция не была полностью обработана по конвейеру (в примере ниже инструкция I2 пытается прочитать исходный код до того, как инструкция I1 выполнит в него запись).

Пример:

Инструкция 1. R2 <-R1+R3

Инструкция 2. R4<-R2+R3

Значение первой инструкций будет сохранено в регистре R2, а вторая инструкция будет использовать это значение для вычисления результата для регистра R4. Во время выполнения конвейерного механизма, когда вторая инструкция извлекает операнд, результат выполнения первой инструкций сохранен не будет. Будет зависимость от данных. Таким образом, существует зависимость от данных с инструкцией I2, поскольку она зависит от завершения инструкции I1.

Запись после прочтения (WAR):

Инструкция I2 пытается записать значение в регистр R5 до того, как оно будет прочитано инструкцией I1.WAR представляет проблему с одновременным выполнением инструкций.

Пример:

Инструкция 1. R4 <-R1+R5

Инструкция 2. R5<-R1+R2

Запись за записью (WAW):

Инструкция I2 пытается записать значение регистра до того, как оно будет записано командой I1. В среде параллельного выполнения команд может происходить запись за записью.

Пример:

Инструкция 1. R2 <-R4+R7

Инструкция 2. R3<-R1+R3

7.5.2. Структурные проблемы при конвейерной обработке команды

Структурная проблема возникает, когда часть аппаратных ресурсов процессора требуется двум или более инструкциям одновременно.

Пример:

Когда к одному блоку памяти осуществляется доступ как на этапе выборки (IF), когда инструкция извлекается из памяти, так и на этапе запоминания, когда данные записываются или считываются (WB) из памяти, как описано ниже на рис. 7.

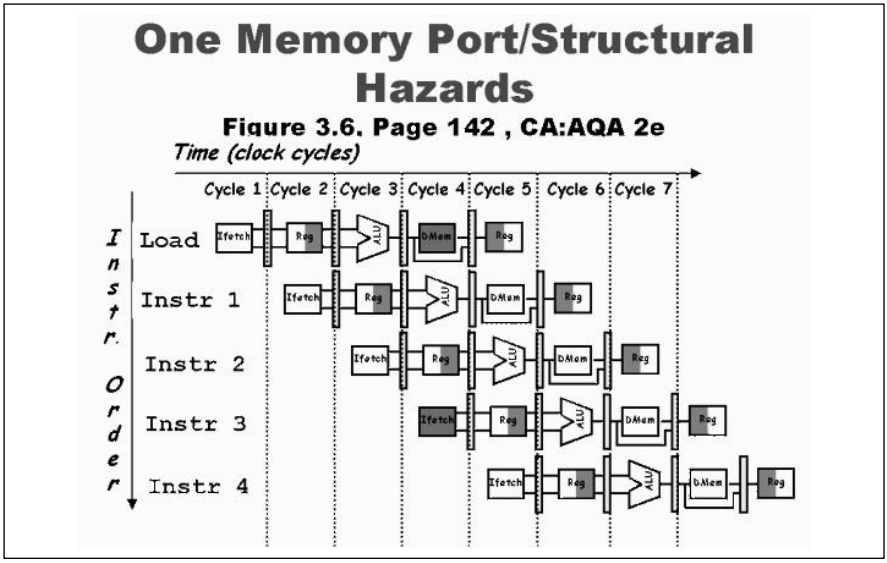

Рис. 7. Структурный конфликт между командами, выполняемыми параллельно на конвейерном процессоре (загружено с https://www.slideserve.com/zhen/cmpe-421-parallel-computer-architecture).

7.5.3. Проблемы управления при конвейерной обработке инструкции

Проблемы с управлением возникают из-за ветвлений в программном коде. В большинстве процессоров с микроархитектурой процессор заранее не знает результат ветвления в коде, когда процессору уже пора вставлять новую инструкцию в конвейер. Это можно лучше понять с помощью приведённого примера ниже на рис. 8.

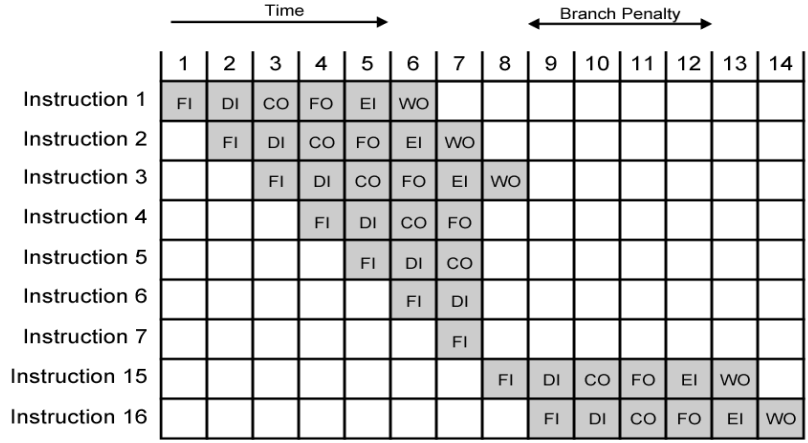

Рис. 8. Механизм, позволяющий избежать конфликт ресурсов в конвейерном процессоре
(загружено с https://www.slideserve.com/ zhen/ cmpe-421-parallel-computer-architecture)

Из приведённого выше примера можно заметить, что существует несколько команд, которые выполняются с использованием механизма конвейерной обработки. Команда выполняется нормально одна за другой до команды номер семь. Во время декодирования команды номер семь происходит ветвление, которое заставило процессор нарушить нормальное выполнение команды. Процессор начинает выполнять инструкции под номером 15, как указано в чётном переходе инструкций.

7.6. Способ избежать проблемы, связанные с конфликтами ресурсов в конвейерных процессорах

Проблемы конфликтов ресурсов в конвейерных процессорах могут быть преодолены путём выполнения динамического планирования инструкций. При динамическом планировании конвейерный процессор извлекает инструкцию и удерживает её, если в инструкции используются какие-либо данные или ресурс, которые совместно используются с другими инструкциями. Конвейерный процессор не будет извлекать никаких других инструкций до тех пор, пока инструкция с общими ресурсами не будет полностью выполнена.

В современных конвейерных процессорах процессор повторно распределяет инструкции, чтобы избежать конфликтов ресурсов и временной задержки, возникающей между выполнениями инструкций.

Динамическое планирование имеет несколько преимуществ:

1. Позволяет коду, который был скомпилирован на одном конвейерном процессоре, запускаться на конвейерном процессоре другого типа.
2. Позволяет обрабатывать некоторые случаи, когда зависимость известна во время компиляции.
3. Позволяет процессору принимать непредсказуемые паузы/задержки, такие как промахи в кэше, выполняя другой код в ожидании устранения промаха.

7.7. Будущее конвейерных процессоров (с точки зрения повышения производительности)

Компьютерная система обладает большей производительностью, чего можно достичь, используя преимущества совершенствования технологий. Организационные усовершенствования центрального процессора позволяют быстрее повышать его производительность.

Пример:

- Использование нескольких регистров, а не одного стека.
- Использование кэш-памяти.
- Конвейерная обработка инструкций.

Задание № 07

1. Предположим, что с помощью конвейера необходимо выполнить шесть команд. Конвейер состоит из 6 этапов. Время выполнения команды в рамках каждого этапа составляет 4 миллисекунды. Определите время выполнения шести команд с конвейером и без конвейера. Также нарисуйте матрицы конвейера.

Решить поставленную задачу поможет подзаголовок 7.3 с графической реализацией выполнения инструкций.

2. Конфликт данных, возникающий при выполнении инструкций в конвейере, имеет разные типы. Какой из них возникает чаще всего и как его можно разрешить.

Объяснение конфликта данных при реализации конвейерной обработки, упомянутое в подзаголовке 7.5.1. Их необходимо понять для решения задачи.

3. Может ли увеличение количества этапов конвейерного выполнения инструкций решить проблемы и конфликты ресурсов, возникающие при их параллельном выполнении? Подкрепите свой ответ аргументами.

В последних подзаголовках 7.6 и 7.7 содержится информация, касающаяся решения задачи 3.

ЗАКЛЮЧЕНИЕ И ВЫВОДЫ

Компьютеры являются частью нашей повседневной жизни. Понимание их архитектуры и работы помогает использовать их более эффективно. Для студентов, изучающих информатику и информационные технологии, архитектура компьютера является, пожалуй, самым фундаментальным предметом в информатике. Без компьютеров область компьютерных наук не существует.

Многие из повседневных действий студентов, будь то просмотр веб-страниц, отправка электронной почты, написание документов, связаны с компьютерной архитектурой или компьютерами. В этом учебно-методическом пособии отражено, как проектируются, создаются и функционируют машины. Знание того, что находится внутри и как это работает, поможет проектировать, разрабатывать и внедрять приложения лучше, быстрее, дешевле, эффективнее и проще в использовании.

В данном учебно-методическом пособии студенты познакомились с современным компьютерным оборудованием, включая внутреннюю работу микропроцессоров и способы программирования многоядерных компьютеров, а также кластера виртуальных/физических машин, которые питают центры обработки данных. Студенты также узнали об истории компьютерных систем, развитии компьютера, идее, лежащей в основе его разработки, факторах, которые со временем помогли в развитии компьютерной системы.

В работе также рассматриваются основные принципы технологии, от которых зависит компьютерная система и её производительность, такие как закон Амдала, закон Мура и т. д. Эти законы предсказывают изменения в технологии и производительности систем в наше время. Это даёт студентам знания о стоимостной и ценовой оценке системы и позволяет им оценить компьютерную систему на основе принципа «цена-затраты» перед фактической покупкой.

Темы раскрывают внутреннюю работу микропроцессоров, включая арифметику с целыми числами и плавающей запятой, концепции конвейерной обработки, многопоточность аппаратного обеспечения, использование параллельного разделения на уровне команд, параллельного разделения на уровне данных и параллельного разделения на уровне потоков.

Это учебно-методические пособие представляет собой полное и обобщённое исследование компьютерной архитектуры, которое недоступно в других связанных исследовательских работах или методологиях по компьютерной архитектуре [1-9]. Эта методология основана на обширных исследованиях и отражает современные концепции компьютерной архитектуры и её компонентов. В нем обсуждаются современные вызовы и угрозы производительности компьютерной системы, а также предлагаются пути их решения путём реорганизации компонентов компьютерной архитектуры и планирования времени между их выполнением, таких как память, регистры, стек, арифметико-логический блок и т. д.

В работе также обсуждался механизм конвейерной обработки инструкций для их параллельного выполнения. Конвейерная обработка – это современная концепция в области технологий, которая используется для повышения эффективности и быстродействия компьютерной системы. В настоящее время исследователи считают конвейерную обработку важной частью компьютерной архитектуры и используют ее для передачи данных и информации между различными компонентами компьютерной системы. Область конвейерной обработки еще не полностью изучена. Выполнение инструкций с помощью конвейерной обработки сопряжено с рядом проблем, которые обсуждались в этом исследовании ранее. Эта работа также предоставляет студентам возможность провести исследование и предложить новое решение по устранению проблем конвейеризации при параллельном выполнении инструкции. В исследовании обсуждаются некоторые распространенные методы, которые часто используются для устранения этих проблем.

Это учебно-методические пособие содержит достаточно информации о будущем компьютерной архитектуры и компьютерных систем. В нем объясняется, что параллельная обработка задач станет основным направлением будущих систем, что повысит их производительность. Параллельная обработка, как правило, применяется в широком спектре приложений, требующих больших объёмов вычислений. Основной целью параллельных вычислений является увеличение вычислительной мощности, доступной для основных приложений. Как правило, в этой инфраструктуре на сервере присутствует набор процессоров или отдельные серверы подключены друг к другу для решения вычислительной задачи.

СПИСОК ЛИТЕРАТУРЫ

1. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P. Droz, “RAMP Blue: A Message-Passing Manycore System In FPGAs,” *Proceedings of International Conference on Field Programmable Logic and Applications*, Amsterdam, The Netherlands, August 2007.
2. N. Njoroge, et al., "Building and Using the ATLAS Transactional Memory System," in Proceedings of the Workshop on Architecture Research using FPGA Platforms, held at HPCA-12. 2006.
3. E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, *and* K. Mai, “A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs,” *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, Monterey, California, 2008.
4. J. D. Davis, Z. Tan, F. Yu, and L. Zhang, “Designing an Efficient Hardware Implication Accelerator for SAT Solving,” SAT 2008 , H. K. Büning and X. Zhao (eds.),LNCS,vol. 4996, pp. 48–62.
5. R. Dimond,M. J. Flynn, O. Mencer, and O. Pell, “Max ware: Acceleration in HPC,” in Proceedings of HotChips 20, August, 2008.
6. J. H. Ahn, W. J. Dally, et al., “Evaluating the Imagine Stream Architecture,” Proceedings of the 31st Annual International Symposium on Computer Architecture, Munich, Germany, June 2004.
7. E. Waingold, et. al., “Baring It All to Software: Raw Machines.” *IEEE Compter*, 30(8), pages80-93, September 1997, MIT/LCS Technical Report TR-709, March 1997.
8. Agarwal, A., et al., "The MIT Alewife machine: architecture and performance," in Proceedings of the ISCA-22. 1995, ACM Press.
9. Abbasi M.M. Computer Architecture: A Detailed Study: Software and Hardware aspects of Computer Architecture October 2015 Publisher: LAP LAMBERT Academic Publishing ISBN: 978-3659790546.

ОГЛАВЛЕНИЕ

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

Учебное издание

Аббаси Мохсин Маншад
Бельтюков Анатолий Петрович

КОМПЬЮТЕРНАЯ АРХИТЕКТУРА

Учебно-методическое пособие

Авторская редакция
Компьютерная верстка: Т.В. Опарина

Подписано в печать 06.05.2024. Формат 60x84 1/16.
Усл. печ. л. 5,4. Уч. изд. л. 4,68.
Тираж 23 экз. Заказ № 882.

Издательский центр «Удмуртский университет»
426034, г. Ижевск, ул. Ломоносова, 4Б, каб. 021
Тел. + 7 (3412) 916-364, E-mail: editorial@udsu.ru

Типография Издательского центра «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, 1, корп. 2.
Тел. 68-57-18