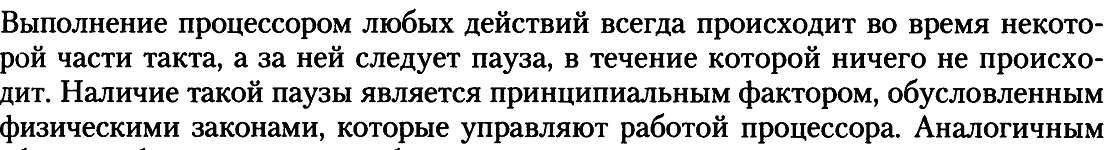
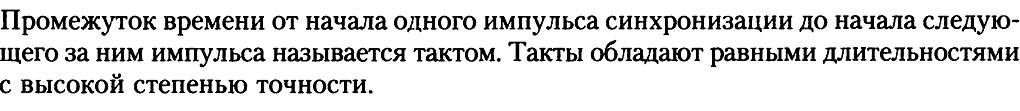
**Тема 3. Логические основы обработки данных**

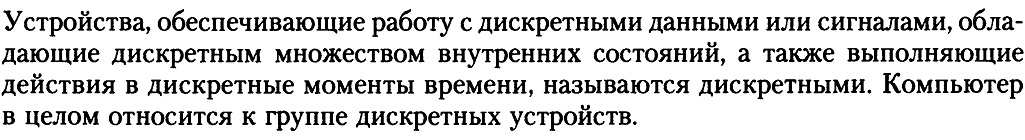
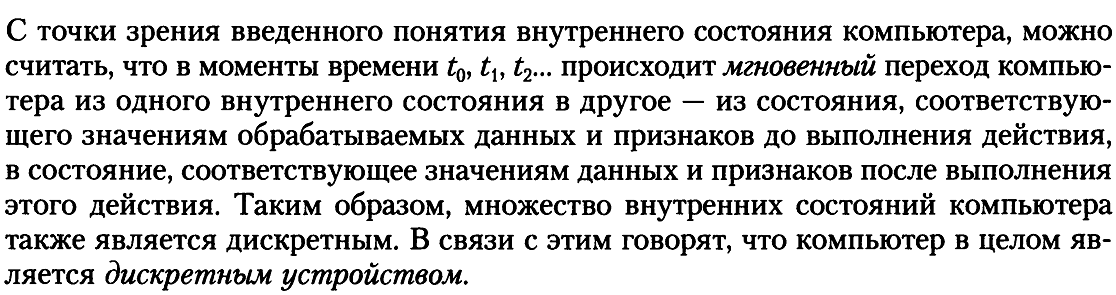
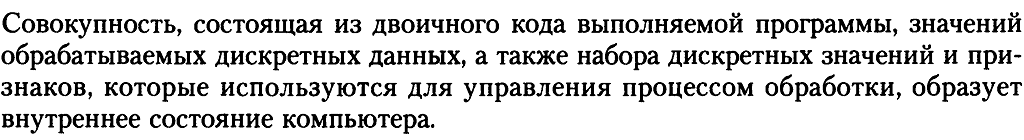
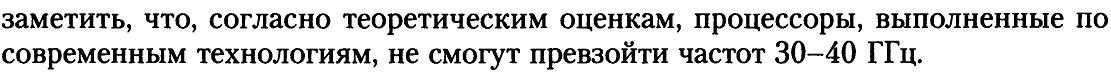
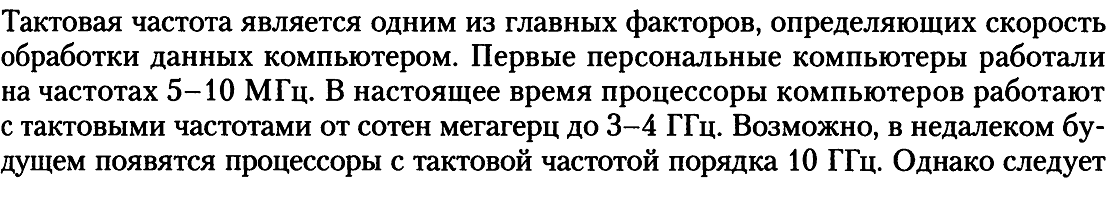
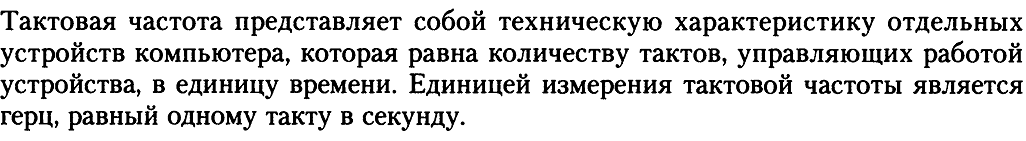
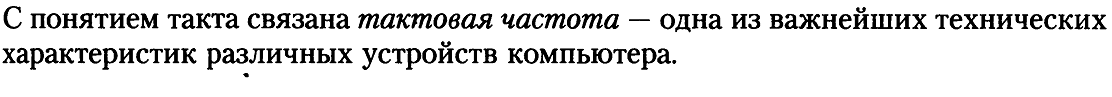
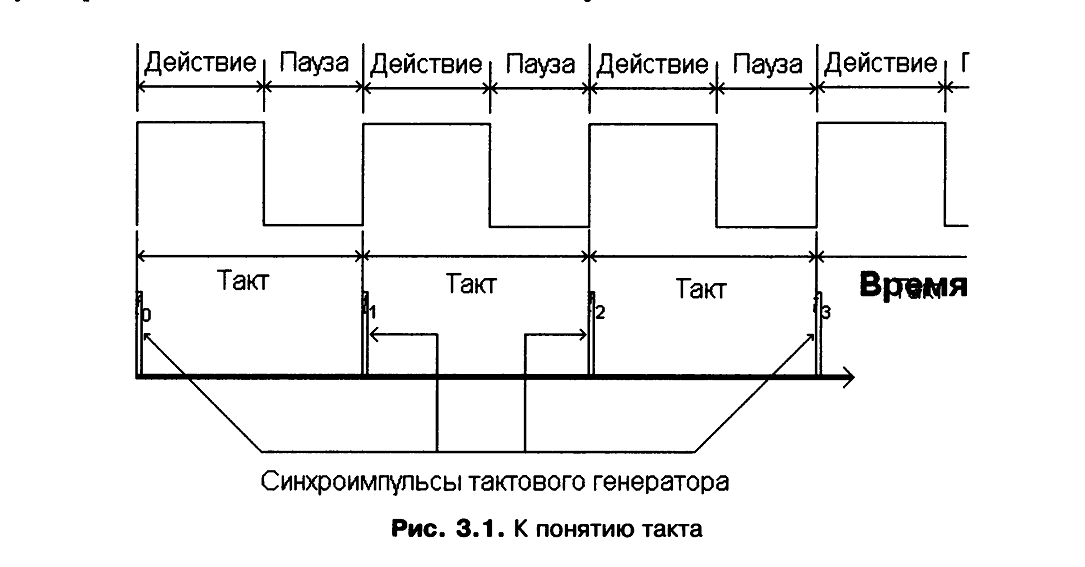
После рассмотрения способов представлени данных в памяти компьютера необходимо выяснить, как осуществляется их обработка. Рассмотри некоторые вопросы, связанные с физическими и логическими основами обработки дискретных данных.

**3.1. Понятие такта**

В выполнении действий над данными участвуют несколько устройств. Эти действия должны быть согласованы и синхронизированы друг с другом. Для синхронизации работы компьютера используется специальное устройство – **тактовый генератор**, который через равные промежутки времени вырабатывает импульсы синхронизации – **синхроимпульсы.**

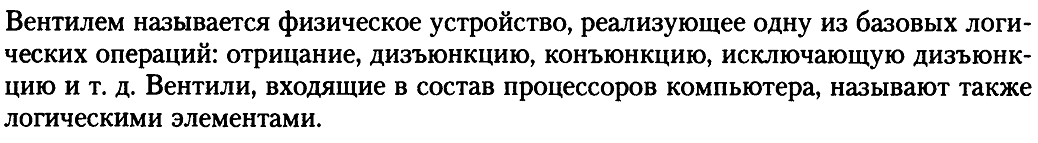


Рассматривая выполнение процессором некоторую последовательность действий, можно считать, что любые действия, происходящие внутри такта, происходят мгновенно, в моменты времени t0, t1, t2, …., соответствующие границам тактов.



**3.2 Вентили и комбинационные схемы**

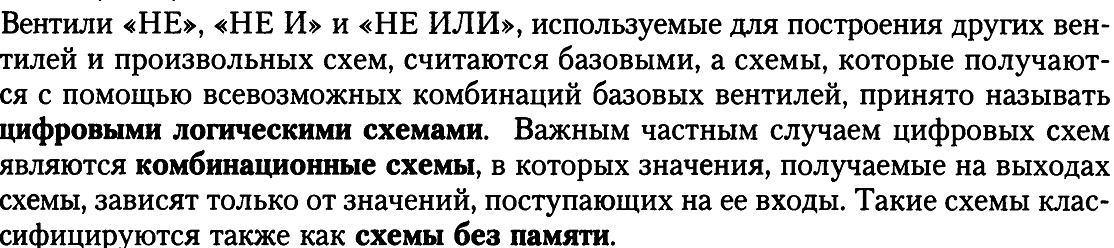
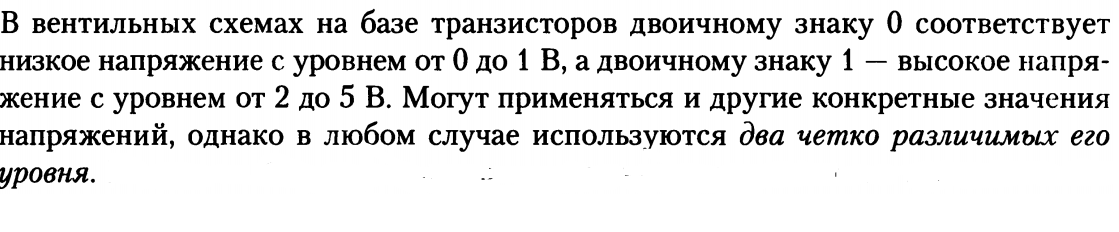
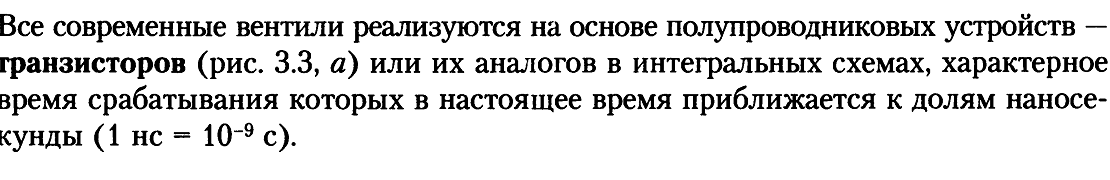
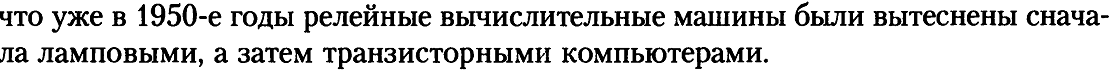
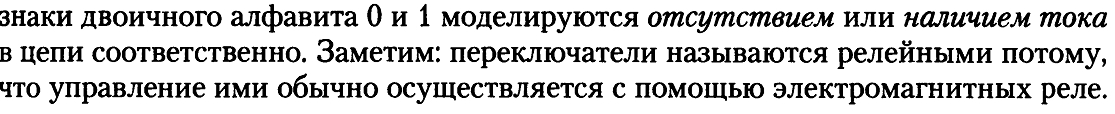
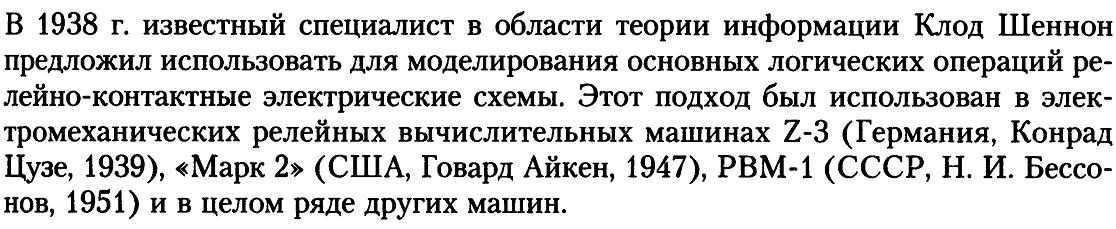
Основными базовыми операциями, которые должен выполнять процессор компьютера, являются логические операции И (отрицание), ИЛИ (дизъюнкция), ИЛИ (конъюнкция), арифметическое сложение, сдвиг кода. Используемые для реализаций этих и других операций устройства принято называть **вентилями** (от немецкого **Ventil** – клапан)



**3.2.1.Основные логические элементы**

|  |  |
| --- | --- |
| Логический элемент И (AND) | Логический элемент ИЛИ (OR) |
| Логический элемент НЕ (NOT)  (инвертор) | Логический элемент И-НЕ (NAND) |
| Логический элемент ИЛИ-НЕ (NOR) | Логический элемент Исключающее ИЛИ (XOR) mod2 |

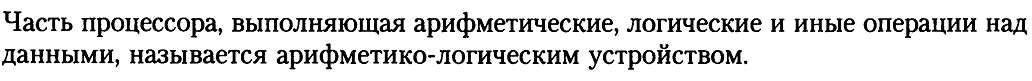
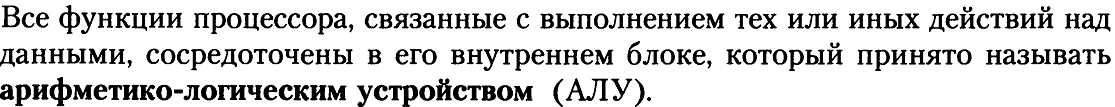
**3.2.2 Реализация вентилей**



Для построения комбинационной схемы необходимо представить логическую функцию в виде логической формулы, провести минимизацию и получить правило соединения логических элементов. Эти вопросы были рассмотрены в курсе ДМиМЛ.

Существуют типовые комбинационные схемы, например, компаратор, дешифратор, шифратор, мультиплексор, демультиплексор и т.д.

**3.2.3. Арифметически-логическое устройство**



**3.3. Схемы памяти на базовых вентилях**

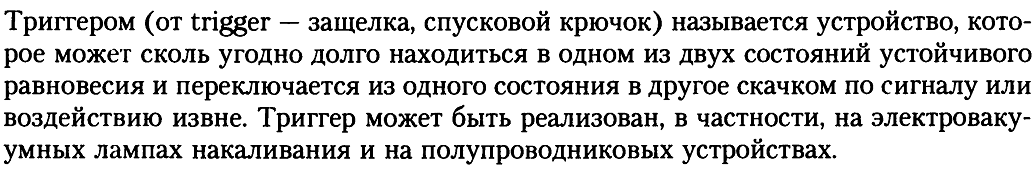
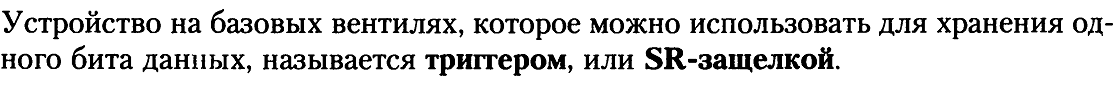
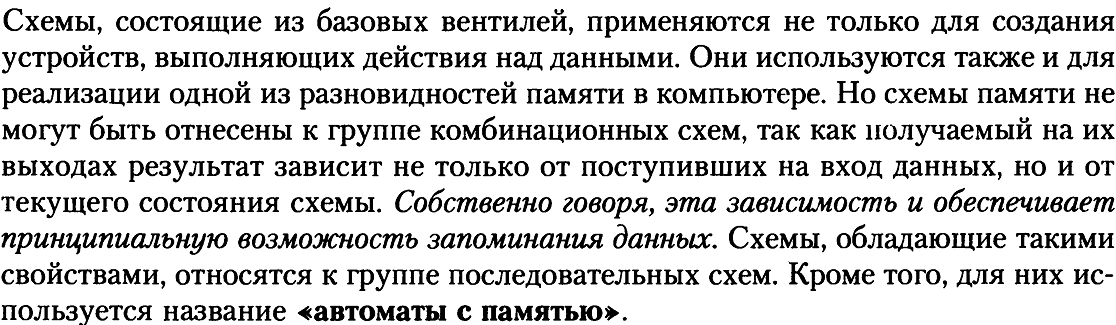
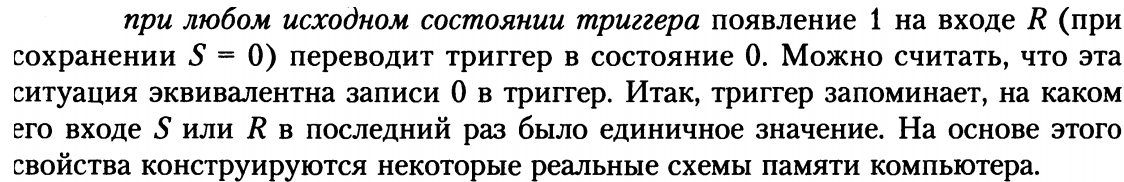
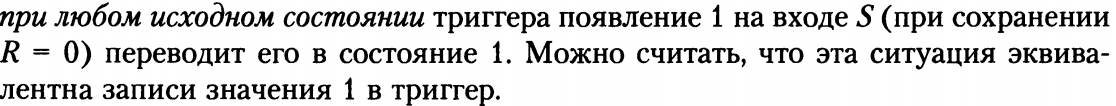
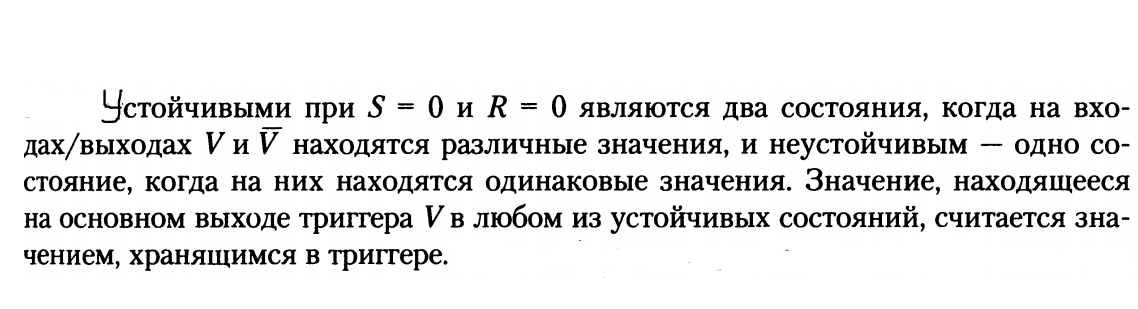
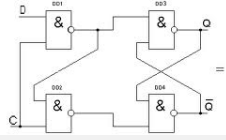
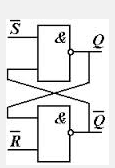
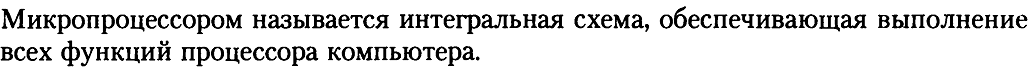
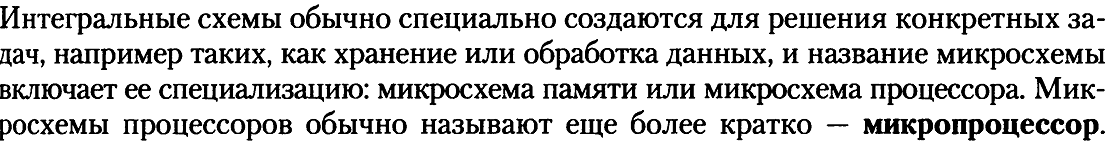
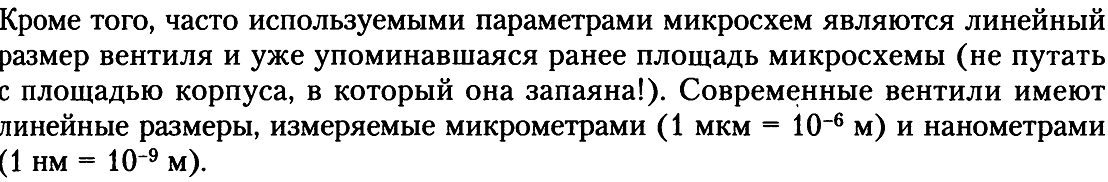
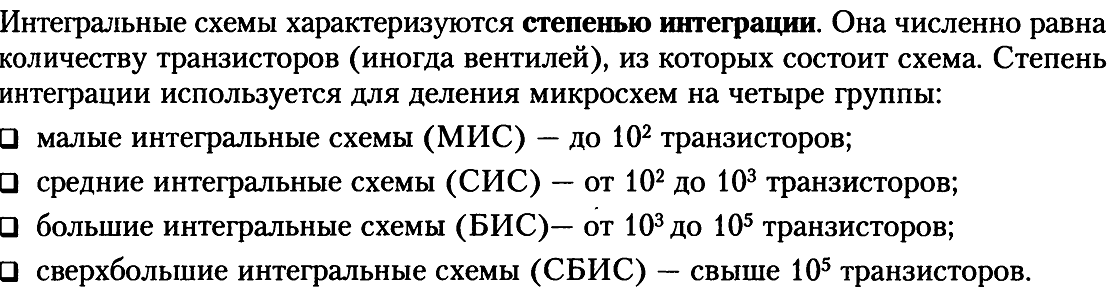
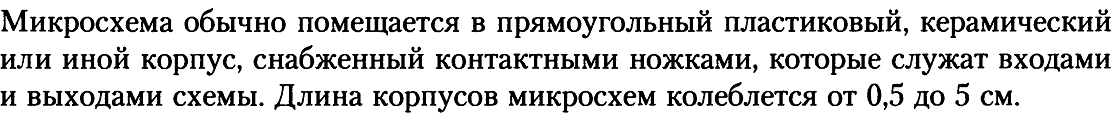
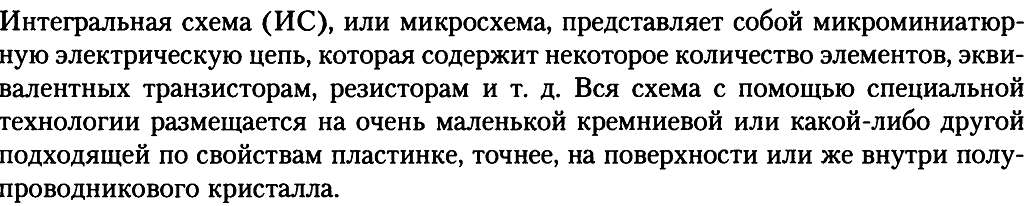


Схема SR-триггера. Схема D-триггера



**3.4. Интегральные схемы**



## Тема 4 Многоуровневая компьютерная организация

Цифровой компьютер – это машина, которая может решать задачи, выполняя данные ей команды. Последовательность команд, описывающих решение определенной задачи, называется программой. Электронные схемы каждого компьютера могут распознавать и выполнять ограниченный набор простых команд. Поэтому все программы перед выполнением должны быть превращены в последовательность таких команд, которые обычно не сложнее чем

* Сложить два числа;
* Проверить, не является ли число нулем;
* Скопировать кусок данных из одной части памяти в другую.

Это примитивные команды и являются тем языком, которым человек общается с компьютером. Такой язык называется машинным. Обычно разработчик старается сделать машинные команды как можно проще, что бы избежать дополнительных сложностей при конструировании компьютера, снизить затраты на электронику. Поэтому большинство машинных языков очень примитивны и их использование трудно и утомительно.

Это привело к тому, что с течением времени появились ряд уровней абстракций, каждая из которых настраивается над абстракцией более низкого уровня. Такой поход называется многоуровневой компьютерной организацией. Такой подход в рассмотрении архитектуры компьютера принят в классической книге «Архитектура компьютера» Эндрью Таненбаума, ведущего специалиста в области разработки компьютеров из МТИ.

**4.1 Современные многоуровневые машины**

Существует разница между тем, что удобно людям и тем, что удобно для компьютера (или что удобно программисту и что удобно разработчику электроники). В литературе это получило название семантического разрыва. Семантический разрыв определяет различие принципов, лежащих в основе языков программирования высокого уровня и тех принципов, которые определяют архитектуру ЭВМ. В этом курсе мы рассмотрим, каким образом можно решить эти проблемы.

#### 4.1.1 Языки, уровни и виртуальные машины

Обозначим через Я1 – язык программирования, удобный для человека. Машинные команды также образуют некоторый язык, понятный компьютеру, который обозначим как Я0. Для того, чтобы перевести программы с языка Я1 на понятный компьютеру язык Я0 можно использовать следующие подходы.

Первый способ – замена каждой команды, написанной на языке Я1 эквивалентным набором команд, написанных на языке Я0. В этом случае компьютер выполняет новую программу, написанную на языке Я0 вместо старой, записанной на языке Я1. Такая технология называется трансляцией.

Второй способ – написание программы на языке Я0, которая берет команды, написанные на языке Я1 в качестве исходных данных, рассматривает каждую команду по очереди и сразу выполняет эквивалентный набор команд языка Я0. Эта технология не требует составления новой программы на языке Я0, Эта технология называется интерпретацией, а программа, осуществляющая интерпретацию называется интерпретатором.

Различия между трансляцией и интерпретацией заключается в том, что при трансляции все программа Я1 переделывается в программу Я0 и программа Я1 может быть отброшена, а в память компьютера загружается программа Я0. При интерпретации каждая команда перекодируется и тут же выполняется.

На практике проще представить себе существование некоторого гипотетического компьютера, или виртуальной машины, для которой машинным языком является удобный для нас язык Я1, чем думать о трансляции и интерпретации. Назовет такую машину М1, а машину, понимающую язык Я0, машиной МО.

Хотелось бы сконструировать машину, для которой машинным языком был бы язык Я1, однако эта машина была бы слишком дорогой.

Чтобы трансляция и интерпретация были бы целесообразны, необходимо, чтобы языки Я0 и Я1 не очень отличались друг от друга. Казалось бы, это требование находится в некотором противоречии с целью создания языка Я1, удобного для человека. Однако это противоречие разрешается следующим очевидным образом.

Создается следующий набор команд Я2, который в большей степени ориентирован на человека. Этому языку соответствует виртуальная машина М2, соответствующие трансляторы или интерпретаторы Я2-Я1 и т.д. Изобретение ряда языков, каждый из которых более удобен для человека, чем предыдущий может продолжаться до тех пор, пока мы не дойдем до подходящего. Каждый последующий язык рассматривает предыдущий как основу и поэтому можно представить компьютер в виде уровней (см. рис. В1).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Уровень n |  | Виртуальная машина Mn  С машинным языком Яn | | 🡪 | Программы на языке Яn либо интерпретируются программой-интерпретатором, работающей на языке более низкого уровня, либо транслируются на машинный язык машины более низкого уровня |
|  |  | :  : | |  |  |
| Уровень 1 |  | Виртуальная машина M1  С машинным языком Я1 | | 🡪 | Программы на языке Я1 либо интерпретируются программой-интерпретатором, работающей на языке Я0, либо транслируются на машинный язык Я0 |
|  |  |  |  |  |  |
| Уровень 0 |  | Виртуальная машина M0  С машинным языком Я0 | | 🡪 | Программы на языке Я0 непосредственно выполняются электронными схемами |

Рис. В1.

Человеку, пишущему программы на языке Яn не обязательно знать об интерпретаторах и трансляторах более низких уровней. Большинство пользователей, использующих машину уровня n интересуются только этим уровнем. Однако для изучения работы компьютера необходимо изучить все уровни.

Деление на уровни является в некотором роде условным. Можно говорить о шести уровнях представления современного компьютера. (рис. В2).

|  |  |  |
| --- | --- | --- |
| Уровень 5 | Язык высокого уровня | |
|  |  | *Трансляция (компилятор)* | |
| Уровень 4 | Уровень языка ассемблера | |
|  |  | *Трансляция (ассемблер)* | |
| Уровень 3 | Уровень операционной системы | |
|  |  | *Трансляция (ассемблер)* | |
| Уровень 2 | Уровень архитектуры команд | |
|  |  | *Интерпретация (микропрограмма) или непосредственное выполнение* | |
| Уровень 1 | Микроархитектурный уровень | |
|  |  | *Аппаратное обеспечение* | |
| Уровень 0 | Цифровой логический уровень | |

На рисунке отсутствует уровень физических устройств, расположенный ниже уровня 0. Это электронные схемы и не являются предметом рассмотрения курса.

На самом нижнем уровне – *цифровом логическом* – объекты состоят из вентилей и хотя вентили являются электронными схемами, их работа может быть описана как цифровые средства. Вентили образуют бит памяти, которые в свою очередь, образуют регистры.

Следующий уровень – *микроархитектурный.* На этом уровне рассматриваются совокупности регистров, которые образуют локальную (или регистровую) память и арифметическо-логическое устройство (АЛУ), предназначенное для выполнения простых операций. Регистры вместе с АЛУ образуют *тракт данных*. Основная операция состоит в следующем: выбирается один или два регистра, АЛУ производит над ними некоторую операцию (например, сложение), результат помещается в один из регистров. В некоторых машинах тракт данных контролируется *микропрограммой*. На других контроль осуществляется аппаратными средствами.

Второй уровень – *уровень архитектуры системы команд.* Руководства по машинному языку, выпускаемые фирмами-производителями содержат информацию именно об этом уровне. Когда там описывается система команд, то описываются команды, выполняемые программой-интерпретатором или аппаратными средствами.

Следующий третий уровень – *уровень операционной системы*-- обычно гибридный. Большинство команд этого уровня имеется также и на уровне архитектуры системы команд (команды, имеющиеся на одном уровне, также могут иметься и на других). Особенности уровня: новые команды, иная организация памяти, способность выполнять две и более программ одновременно и некоторые другие. Команды третьего уровня, идентичные командам второго, выполняются микропрограммой или аппаратными средствами, а не операционной системой. Часть команд (не имеющихся на втором уровне), выполняются операционной системой.

Нижние уровни конструируются не для того, что бы с ними работал обычный программист. Они предназначены для работы интерпретаторов и трансляторов и поддерживаются системными программистами. Уровни с четвертого и выше предназначены для работы прикладных программистов. Следующая особенность: уровни 2 и 3 обычно интерпретируются, а уровни 4 и выше как правило, поддерживаются транслятором. Уровни 1, 2 и 3 – цифровые, т.е. программы, написанные на языках этих уровней, состоят из наборов цифр. Начиная с четвертого уровня, языки содержат слова и сокращения, понятные человеку.

Четвертый уровень представляет собой символьную запись языка более низкого уровня. Программа, которая выполняет трансляцию называется *ассемблером .*

Пятый уровень обычно состоит из языков, разработанных для прикладных программистов. Такие языки называются языками высокого уровня.

Выводы: компьютер проектируется как иерархическая структура уровней, каждый из которых надстраива6ется над предыдущим. Каждый уровень представляет собой абстракцию с различными объектами и операциями.

Набор типов данных, операций и особенностей каждого уровня называется *архитектурой.* Архитектура связана с аспектами, которые видны программисту. Аспекты разработки, технологии и т.д. не являются частью архитектуры. Термины компьютерная архитектура и компьютерная организация в сущности означают одно и то же.

#### 4.1.2 Развитие многоуровневых машин

*Аппаратное обеспечение* состоит из электронных схем, памяти, устройств ввода-вывода, т.е. из осязаемых объектов.

*Программное обеспечение* состоит из алгоритмов и программ. В первых вычислительных машинах разница между программными и аппаратными средствами была очевидной. Со временем эта грань стала размываться. Сейчас можно говорить о том, что аппаратное и программное обеспечения логически эквивалентны. Карен Панетта Ленц говорил: «Аппаратное обеспечение – это всего лишь окаменевшее программное обеспечение». Разделение функций программного и аппаратного обеспечения определяется такими факторами как стоимость, скорость, надежность, а также частота ожидаемых изменений.

4.1.2.1. Изобретение микропрограммирования

У первых компьютеров было только два уровня уровень архитектуры набора команд и цифровой логический уровень.

В 1951 году Морис Уилкс (Кембриджский университет) предложил идею трехуровневого компьютера. Такой компьютер должен иметь встроенный неизменяемый интерпретатор (микропрограмму), функция которого заключалась в выполнении программ посредством интерпретатора. Таким образом, аппаратное обеспечение должно было выполнять только микропрограммы с ограниченным набором команд. Электронные схемы существенно упростились, цена уменьшилась, а надежность возросла. К 70-м годам идея микропрограммирования стала преобладающей.

4.1.2.2 Изобретение операционной системы

Первые операционные системы появились в 60-е годы. Придумана она была для того, чтобы автоматизировать работу оператора. Однако создание операционной системы было первым шагом на пути в развитии новой виртуальной машины. К уровню архитектуры команд добавлялись новые команды и в итоге сформировался новый уровень. Некоторые команды нового уровня были идентичны командам предыдущего, но появились и новые команды, которые полностью отличались. Эти команды тогда назывались *макросами ОС* или *вызовами супервизора*. Сейчас используют термин *системный вызов*. Первые операционные системы были ориентированы для работы в *пакетном режиме*. В начале 60-х годов в МТИ разработали операционную систему, которая позволяла одновременно работать нескольким пользователям. В такой системе ресурсы центрального процессора разделялись между несколькими пользователями и такие системы назывались и сейчас называются системами с *разделением времени*.

4.1.2.3 Перемещение функциональности на уровень микрокоманд

С 1970 г. микропрограммирование стало обычным, производители вводили новые команды путем расширения микропрограммы, т.е. программными методами. Многие новые команды не представляли особой ценности, т.к. эти же действия можно было сделать уже имеющимися средствами. Однако новые команды могли выполнять операции быстрее или предоставляли дополнительные удобства пользователю. Например:

* Ускорение работы с массивами (индексная и косвенная адресация)
* Системы прерывания, которые дают команду процессору, как только закончилась операция ввода или вывода;
* Способность приостановить одну программу и начать другую, используя небольшое количество команд (переключение процесса);
* Специальные команды для обработки изображений, мультимедийных данных.

4.1.2.4 Устранение микропрограммирования

В 60-х – 70-х годах количество микропрограмм увеличивалось, но они работали все медленнее и медленнее, т.к. требовали значительного объема памяти. Пришло понимание того, что с устранением микропрограмм резко сократиться количество команд и компьютеры станут работать быстрее. Таким образом, компьютеры вернулись к тому состоянию, в котором они были до изобретения микропрограммирования.

Вывод: граница между аппаратным и программным обеспечением постоянно перемещается. Так же обстоит дело с уровнями – между ними нет четких границ.

### 4.2 Развитие компьютерной архитектуры

#### 4.2.1. Нулевое поколение – механические компьютеры (1642-1945)

Первую счетную машину создал французский ученый Блез Паскаль в 1642 году. Ему тогда было 19 лет, он создал машину для своего отца, сборщика налогов. Машина могла выполнять только операции сложения и вычитания.

Тридцать лет спустя великий немецкий математик Готфрид Вильгельм Лейбниц (1646-1716) построил счетную машину, которая помимо операций сложения и вычитания могла выполнять операции умножения и деления.

Еще через 150 лет профессор математики Кембриджского университета Чарльз Бэбидж (1792-1871) (изобретатель спидометра) разработал и сконструировал разностную машину, предназначенную для подсчета таблиц чисел морской навигации. Машина могла выполнять только один алгоритм – метод конечных разностей с использованием полиномов. Машина, выполнявшая только один алгоритм, Бэбиджу вскоре наскучила и он начал разрабатывать (и потратил на это очень много средств) аналитическую машину. У аналитической машины было запоминающее устройство, вычислительное устройство, устройство ввода (для перфокарт), устройство вывода (перфоратор и печатающее устройство). Машина могла выполнять разные задачи. Она считывала команды с перфокарт и выполняла их. Поскольку аналитическая машина программировалась на ассемблере, то ей было необходимо программное обеспечение. Первый программист – племянница поэта Байрона Ада Ловлейс. В ее честь назван язык программирования АДА. Аналитическая машина была механической. Идеи Бэбиджа опередили эпоху в том смысле, что технологическая база не позволяла создавать устройства такой сложности с приемлемой надежностью.

В конце 30-х -- начале 40-х годов ХХ века счетные машины были сконструированы в Германии и Америке, в которых были использованы электромагнитные реле. Машина Джона Атанасова была очень развита для своего времени. В ней использовалась бинарная арифметика, информационные емкости, которые периодически обновлялись. К сожалению, эта машина так и не заработала.

Говард Айкен, опираясь на исследования Бэбиджа, решил создать такой же компьютер, но на основе реле. Работа над первым компьютером была закончена в 1944 году. Называлась машина «MARK 1». Затем началась эра электроники.

#### 4.2.2 Первое поколение – электронные лампы (1945-1955)

Стимулом для разработки электронного компьютера стала Вторая мировая война. Машина создавалась для шифровки и дешифровки в Великобритании. Одним из создателей этой машины был Алан Тьюринг.

В Америке Джон Моушли со своим студентом Дж. Преспером Экертом начали конструировать компьютер, предназначенный в первую очередь для составления таблиц для нацеливания тяжелой артиллерии. К моменту завершения разработки война закончилась, машина стала не нужна для военных целей, и разработчикам было разрешено организовать школу, где они рассказывали о своей работе. Эта машина – ENIAC. Патент на ЦВМ они не получили, т.к. приоритет был отдан Атанасову.

В это же время в Институт специальных исследований в Принстоне приехал один из участников проекта ENIAC Джон фон Нейман, чтобы сконструировать свою версию компьютера.



Он предложил размещать программу вместе с данными в оперативной памяти и использовать бинарную арифметику. Основной проект известен теперь как фон-неймановская вычислительная машина. Он был использован в машине EDSAC. Практически все современные компьютеры являются фон-неймановскими машинами. Схема архитектуры этой машины приведена на рисунке В3. Машина не имела операций с плавающей точкой. Нейман полагал, что любой сведущий математик способен держать плавающую точку в голове.

Приблизительно в это же время в МТИ был создан компьютер Whirlwind-1. Особен­нос­ти компьютера: слова неболь­шой длины (16 бит) и работа в РМВ. Он является прототипом мини-компьютера.

В 1953 году IBM создала свой первый компьютер IBM-704.

#### 4.2.3.Второе поколение – транзисторы (1955-1965)

Транзисторы были изобретены в лаборатории Bell Джоном Бардином, Уолтером Браттейном и Уильямом Шокли, за что в 1956 году им была присуждена Нобелевская премия. Первый компьютер на транзисторах был построен в МТИ и назывался ТХ-1, а затем ТХ-2. Практического значения эти компьютеры не имели, но один из разработчиков, Кеннет Ольсен в 1957 году основал фирму DEC и произвели первую серийную машину на транзисторах PDP-1 (1961 г.). Эта была самая быстродействующая машина того времени. Время цикла – 5 микросекунд. Это в два раза меньше, чем у IBM-7090 (транзисторного аналога IBM-709). Стоил PDP-1 $120 000, а IBM – миллионы. Компания DEC продала десятки компьютеров PDP и так возникла компьютерная промышленность. Один из компьютеров был отдан в МТИ, где был создан первый графический дисплей, а студенты написали первую компьютерную игру – «Война миров».

Затем была создана машина PDP-8, которая была 12-разрядной, стоила $16 000, а главное нововведение – одна шина. *Шина* – это набор параллельно соединенных проводов для связи компонентов компьютера. Структура компьютера с общей шиной приведена на рис. В4. Такая структура с тех пор используется во всех компьютерах.



В 964 году компания CDC выпустила машину CDC-6600, которая имела производительность на порядок выше, чем IBM-7090 и ее более дешевый аналог IBM-1401. Высокая производительность обеспечивалась за счет того, что внутри центрального процессора находилась машина с высокой степенью параллелизма. Разработчиком этого компьютера был Сеймур Крей. Он посвятил свою жизнь созданию мощных компьютеров, которые сейчас называются *суперкомпьютерами.* Это компьютеры CDC-6600, CDC-7600, Crey-1.

Разработчики упомянутых выше компьютеров занимались в первую очередь аппаратным обеспечением, стремясь повысить его надежность, быстродействие и снизить стоимость.

Следует отметить еще один проект – Burroughs B50000. Разработчики создавали компьютер с намерением программировать ее на языке Algol 60 (предшественник языка Pascal), сконструировав аппаратное обеспечение так, что бы упростить работу компилятору. Так появилась идея, что программное обеспечение тоже надо учитывать при разработке компьютера.

#### 4.2.4 Третье поколение – интегральные схемы (1965-1980)

В 1958 году была изобретена кремниевая технология (изобретатель – Роберт Нойс). Компьютеры на интегральных схемах были меньшего размера, работали быстрее, стоили дешевле. Наиболее значительные следующие.

К 1964 г. Фирма IBM лидировала на рынке, но выпускаемые ей компьютеры были программно несовместимы. Компания сделала решительный шаг. Она выпустила серию компьютеров на транзисторах System 360, которые были предназначены как для научных, так и для коммерческих расчетов. System 360 содержала много нововведений. Это было семейство компьютеров с одним и тем де ассемблером. Каждая новая модель была больше и мощнее предыдущей. Идея создания семейств компьютеров вскоре стала популярной и в течении нескольких лет большинство компьютерных компаний выпустило целые серии сходных машин.

Еще одно нововведение – *мультипрограммирование*. В памяти располагалось несколько программ и пока одна программа ждала окончания ввода-вывода, другая выполнялась.

Мир микрокомпьютеров сделал также большой шаг вперед вместе с производством компьютеров PDP-11. Во многих отношениях PDP-11 была младшим братом IBM 360 по организации компьютера и наличию в семействе машин разной стоимости и производительности.

#### 4.2.5. Четвертое поколение – сверхбольшие интегральные схемы

Появление СБИС в 80-х годах позволило помещать на одну плату сначала десятки тысяч, а затем и миллионы транзисторов. К 80-м годам цены на компьютеры упали на столько, что приобретать компьютеры смогли не только организации, но и отдельные люди. Началась эра персональных компьютеров. Первые персональные компьютеры продавались в виде комплектов, как правило, на базе Intel 8080. Программное обеспечение пользователь писал сам. Затем появилась операционная система CP/M. Эта ОС помещалась на дискету, включала систему управления файлами и интерпретатор для выполнения пользовательских команд, которые набирались на клавиатуре.

Компания IBM, лидирующая в то время на рынке компьютеров, тоже решила заняться производством персоналок. Для ускорения процесса разработки копания IBM предоставила одному своему сотруднику, Филипу Эстриджу крупную сумму денег на создание персонального компьютера. Компьютер IBM PC появился в 1981 году и стал самым покупаемым в истории.

Но IBM вместо того, чтобы держать проект в секрете или защитить его патентами, опубликовала полные проекты, включая электронные схемы. Многие компания тут же начали делать *клоны*, которые продавали дешевле. Из других компаний, производивших персональные компьютеры на базе своих процессоров, выжить удалось только некоторым, и то только потому, что они работали в узких областях.

Первая версия IBM PC была оснащена операционной системой MS-DOS, которую выпускала крошечная компания Microsoft. Эта компания разработала также собственную ОС Windows, которая работала на базе MS-DOS. Успех процессора 8088 воодушевил Intel на усовершенствования. Модель 386 – первый представитель линейки Pentium.

В середине 80-х годов на CISC-архитектурой на RISC-компьютеры, которые проще и работают быстрее. В 90-х годах появились суперскалярные компьютеры. Первый 64-разрядный компьютер был выпущен в 1992 году (Alpha, DEC), но коммерческий успех был скромным – 64-разрядные компьютеры приобрели популярность только в спустя десятилетие и в качестве персональных серверов.

#### 4.2.6 Пятое поколение – невидимые компьютеры

В 1981 году правительство Японии объявило о намерении выделить национальным компаниям 500 миллионов долларов на разработку компьютеров пятого поколения на основе технологий искусственного интеллекта. Однако проект, в целом, оказался несостоятельным. Причина, скорее всего, заключается в том, что идея несколько опередила технологию.

Пятое поколение компьютеров материализовалось в виде малых по размерам компьютеров – карманных компьютеров и «невидимых компьютеров» -- компьютеров, встраиваемых в бытовую технику, банковские карточки и т.п. Процессоры этого типа предусматривают большие функциональные возможности, широкий спектр применения за умеренную цену. Компьютеры пятого поколения ассоциируются не с некоторой архитектурой, а парадигмой использования. В настоящее время этому явлению применяется термин «всепроникающая компьютеризация».

### *В.3 ТИПЫ КОМПЬЮТЕРОВ*

#### *Технологические аспекты*

Закон технологического прогресса, известный как закон Мура (Гордон Мур – один из основателей Intel) утверждает, что количество транзисторов на одной микросхеме удваивается каждые 18 месяцев. По-существу закон Мура не закон, а эмпирическое наблюдение за развитием технологий, и, по мнению специалистов, продержится еще лет 10, может больше. Однако, существует физический предел, определяемый принципом неопределенности Гейзенберга.

Закон Мура связан с так называемым *эффективным циклом*. Т.е. новые возможности порождают новые потребности, для удовлетворения которых требуются новые возможности и т.д.

Еще один фактор развития компьютерных технологий – первый закон программного обеспечения, названный в честь Натана Мирвольда (главный администратор Microsoft). Закон гласит; программное обеспечение это газ, который полностью заполняет резервуар, в котором находится. Т.е. программное обеспечение продолжает развиваться и требовать все больше ресурсов.

Шутка Г.Мура на тему темпов развития IT-технологий; если бы авиационные технологии развивались такими же темпами, как компьютерные, то самолеты стоили бы $500, облетали Землю за 20 минут на 20 литрах керосина и были бы размером с обувную коробку.

#### *Широкий спектр компьютеров*

Исследователь из лаборатории Bell Ричард Хамминг заметил, что количественное изменение характеристик на порядок ведет к качественному изменению. В компьютерных технологиях количественные характеристики изменились за 30 лет на 6 порядков. Компьютерные технологии развиваются как по пути увеличения мощности так и снижения цены на единицу мощности. Примерная классификация компьютеров приведена в таблице.

|  |  |  |
| --- | --- | --- |
| *Тип* | *Цена $* | *Сфера применения* |
| «Одноразовые компьютеры» | 0,5 | Поздравительные открытки |
| Встроенные компьютеры | 5 | Часы, машины, бытовые приборы |
| Игровые компьютеры | 50 | Домашние компьютерные игры |
| Персональные компьютеры | 500 | Настольные и портативные компьютеры |
| Серверы | 5 тыс. | Сетевые серверы |
| Комплексы рабочих станций | 50 – 500 тыс. | Супер-мини компьютеры |
| Мэйнфреймы | 5 млн. | Пакетная обработка данных в банке |

##### Одноразовые компьютеры

Наибольшим достижение в этой области можно считать появление микросхем RFID (Radio Frequency Identification – радиочастотная идентификация). Эти микросхемы без батареек, содержат приемо-передатчик, который по внешнему запросу выдает код. Это может быть использовано для снятия штрих-кодов, идентифицировать можно до конкретной единицы продукции, с увеличением объема памяти таких микросхем можно записывать и другие данные.

Микросхемы могут быть активными и пассивными, работать на разных частотах (чем выше частота, тем выше скорость передачи данных, но меньше радиус действия).

##### Микроконтроллеры

Микроконтроллеры выполняют функции управления устройствами и организации пользовательских интерфейсов. В отличие от RFID-микросхем, которые выполняют минимальный набор функций, микроконтроллеры представляют собой полноценные вычислительные устройства. Содержат процессор, память, устройства ввода-вывода. В ряде случаев программное обеспечение прошивается в памяти призводителем.

Микроконтроллеры разделяются на универсальные и специализированные.

Особенности микроконтроллеров; низкие цены, работа в реальном масштабе времени, жесткие ограничения на размер и электропотребление.

##### Игровые компьютеры

Это обычные компьютеры, в которых расширенные возможности графических и звуковых контроллеров сочетаются с ограниченным ПО и пониженной расширяемостью. Они оптимизированы на конкретную область применения – выполнение трехмерных игр. Все остальное считается вторичным – отсюда и низкая, по сравнению с ПК, цена.

##### Персональные компьютеры

Некоторые специалисты называют ПК с процессорами Intel, отделяя их от компьютеров, оснащенных высокопроизводительными RISC-микросхемами (Sun UltraSPARC), которые называют рабочими станциями. Но особой разницы между этими компьютерами нет.

К персональным компьютерам близки карманные компьютеры PDA.

##### Серверы

Мощные персональные компьютеры и рабочие станции часто используются в качестве сетевых серверов как в пределах локальной сети, так и в Интернете. Серверы поставляются в однопроцессорной или мультипроцессорной конфигурациях. Некоторые серверы способны обрабатывать до миллиона транзакций в секунду.

С точки зрения архитектуры серверы ничем не отличаются от персональных компьютеров, только работают быстрее, имеют больше дискового пространства, устанавливают более быстрые сетевые соединения. Серверы работают под управлением тех же операционных систем, как правило Windows и UNIX.

##### Комплексы рабочих станций

COW-системы (Clusters Of Workstations) – кластеры рабочих станций состоят из нескольких персональных компьютеров или рабочих станций. Эти компьютеры соединены высокоскоростной сетью и снабжены специальным ПО. Предназначены для решения «больших» задач.

В виде кластеров могут быть организованы веб-серверы. Если частота обращения в страницам сайта исчисляется тысячами в секунду, то дешевле организовать кластер из нескольких сотен (или тысяч) серверов и распределить между ними нагрузку. Такие кластеры называются серверными фермами (server farms).

##### Мэйнфреймы

Это большие компьютеры, размером с комнату. В большинстве случаев эти системы – потомки больших компьютеров IBM 360/370. Обычно они работают не намного быстрее, чем мощные серверы, но у них выше скорость процессов ввода/вывода, обладают большим дисковым пространством. Эти системы дорогие. Многие компании считают, что выгоднее вложить деньги в такую систему, чем переписывать все программное обеспечение под персональные компьютеры. Именно из-за этих систем и возникла проблема 2000 года.

В последнее время под влиянием Интернета наблюдается возрождение мэйнфреймов как полноценной категории компьютеров. Они занимают нишу мощных серверов Интернета, способных обрабатывать огромное количество транзакций в секунду.

До последнего времени существовала еще одна категория вычислительных машин – *суперкомпьютеры.* Это системы с высокопроизводительными процессорами, высокоскоростные диски, сетевые интерфейсы. Сейчас, когда вычислительные возможности, аналогичные тем, которые предлагают суперкомпьютеры реализуются в виде кластеров, эта категория компьютеров постепенно отмирает.

### *В.4 СЕМЕЙСТВА КОМПЬЮТЕРОВ*

#### *Pentium 4*

В 1968 году Роберт Нойс, изобретатель кремниевой интегральной схемы, Гордон Мур и Артур Рок, капиталист из Сан-Франциско, основали корпорацию Intel для производства микросхем. Сначала дела шли не очень хорошо. В 60-х годах калькуляторы были размером с принтер и весили 20 кГ.

Лет В 1969 году японская фирма Buscom обратилась к компании Intel с просьбой выпустьть 12 несерийных микросхем. Инженер Тэд Хофф решил, что можно поместить 4-х битный универсальный процессор на одну миросхему. Так в 1970 году появился первый процессор на одной микросхеме, процессор 4004. В 1972 году Intel выпустила 8-битный процессор 8008.

Новая микросхема вызвала большой интерес и Intel начала разработку новой микросхемы, у которой предел обращения к памяти в 16 Кбайт был преодолен. Так появился 8080, выпущенный в 1974 году. Этот процессор произвел революцию.

1978 год – процессор 8086. Он имел 16-разрядные внутренние регистры, 16-разрядные внутренние и внешние шины данных и мог адресовать до 1 Мб физической памяти. Быстродействие – 0,8 MIPS (миллион оп/с).

Затем появился 8088, с такой же архитектурой, как у 8086. Он имел шину не 16, а 8 бит, работал медленнее, но был дешевле. Когда фирма IBM выбрала 8088 для IBM PC, эта микросхема стала эталоном в производстве персональных компьютеров.

В начале 80-х годов Intel разработала 80286, совместимый с 8086. Он уже мог адресовать до 16 Мб физической памяти, имел специальный защищенный режим работы, который обеспечивал гибкий механизм адресации, управление доступом к памяти, управление привилегиями и т.п. Система команд также была расширена, а быстродействие достигало 2,7 MIPS.

Intel386 (1985) – это уже полностью 32-разрядный микропроцессор с 32-разряднымивнутренними регистрами и шинами данных. Его адресное пространство – 4 Гб. Добавлены страничный механизм и добавлен новый режим работы V86, который обеспечивал совместимость данного микропроцессора с программами, написанными для предыдущих устройств серии. Внутренняя архитектура микропроцессора Intel386 позволила организовать параллельное выполнение нескольких операций (выборка команд, декодирование, исполнение команд) что позволило увеличить быстродействие до 6 MIPS. Приблизительно в это же время была разработана новая операционная система Windows.

В 1989 г. выпущен микропроцессор Intel486, который содержал интегрированное устройство вычислений с плавающей точкой (FPU) и внутреннюю кэш-память для данных и команд.

В 1993 году был создан первый микропроцессор семейства P5 – Pentium, характеризующийся значительно более совершенной архитектурой. В этих процессорах устройство целочисленных вычислений имело два практически идентичных блока, в которых различные команды могли выполняться параллельно. Были доработаны модули, отвечающие за выборку и декодирование команд: была предусмотрена возможность прогнозировать действия процессора, путем осуществления предвыборку и предкодирование команд еще до получения результатов предыдущих команд. Переработано FPU с целью повышения его быстродействия. Были внесены некоторые изменения в архитектуру процессов: появилась поддержка объемных страниц, более гибким стал режим V86, введены дополнительные средства внутренней самодиагностики. Быстродействие таких процессоров могло достигать 100 MIPS.

В 1993-1997 Intel продолжал работу по совершенствованию архитектуры своих процессоров. Был выпущен Pentium Pro – микропроцессор архитектуры P6, которая стала основной для более поздних микропроцессоров, вплоть до Pentium III. В этих процессорах: четыре модуля параллельно осуществляли выполнение команд, предвыборка дополнилась возможностями предсказания ветвлений, кэш-память стала двухуровневой, введена расширенная система обработки мультимедийной информации (MMX).

Pentium II/III -- их быстродействие может достигать 1000 MIPS, объем оперативной памяти до 64 Гб, внутренней кэш-памяти – 2 Мб. Система обработки мультимедийной информации была усовершенствована: в микропроцессор Pentium III были введены 70 новых так называемых SIMD-команд.

Помимо основной линейки процессоров Intel разрабатывает специальные микросхемы. Например,

* 1998 г. – Celeron, уступал по производительности Pentium, но и стоил дешевле.
* 1988 г., Xeon -- имеет КЭШ-память большего объема, ускоренная внутренняя шина, усовершенствованные средства поддержки мультипроцессорного режима.

Pentium 4 (2000 )-- система основана на новом конструктивном решении, по традиции обошел все предыдущие модели по производительности. В версии с тактовой частотой в 3,06 ГГц была введена многопоточность. Для повышения скорости обработки звуковых и видеоданных был внедрен дополнительный набор SSE-команд.

В 2003 г. Появилась микросхема Pentium M – мобильный.

Все микросхемы Intel обратно совместимы. Закон Мура применим и к процессорам. Однако возникает проблема теплоотдачи, что, вероятно, внесет коррективы в этот закон.

В ноябре 2004 г. Intel прекратила выпуск Pentium 4 с тактовой частотой 4 ГГц из-за проблем с теплоотдачей. Проблему повышения быстродействия решают за счет размещения на одной микросхеме двух процессоров и увеличить объем КЭШ-памяти – память потребляет меньше энергии.

#### *UltraSPARC III*

В 70-х годах в университетах очень популярна была ОС UNIX, но персональные компьютеры не подходили для этой системы. Энди Бехтольсхайм (аспирант из Стенфорда) разрешил эту проблему, самостоятельно построив рабочую станцию UNIX из стандартных компонентов, имеющихся в продаже и назвал ее SUN (сеть Стенфордского университета).

27-летний индиец Винод Косла предложил Энди Бехтольсхайму организовать компанию по производству рабочих станций SUN. Он нанял также аспиранта Скотта Мак-Нили и программиста Билла Джоя, главного создателя системы UNIX. Они организовали компанию Sun Microsystems. Первый компьютер был оснащен процессором Motorola 68020 и имел большой успех, так же, как и последующие модели. Они были мощнее персональных компьютеров и имели аппаратные и программные средства работы в сети ARPANET (предшественник Internet). В 1987 году компания решила разработать свой собственный процессор, основанном на революционном проекте Калифорнийского университета (RISC II). Этот процессор назывался SPARC (Scalable Processor ARCitecture – наращиваемая архитектура процессора).

В отличие от других компаний, Sun решила не производить процессоры, а предоставила патент нескольким компаниям. Так появились MicroSPARC, HyperSPARC, SuperSPARC, TurboSPARC.

Первый SPARC был 32-разрядным, а в 1995 году была разработана 64-разрядная версия. UltraSPARC с самого начала был предназначен для работы с изображениями, аудио, видео и мультимедиа вообще.

UltraSPARC IV представлят собой двухпроцессорный комплекс, где два процессора UltraSPARC III размещены на одной микросхеме с одним КЭШем.

#### *Микросхема 8051*

Микросхема применяется во встроенных системах, ее история началась в 1976 году. В это время уже сформировался спрос на встраиваемые вычислительные устройства, а использовать процессор 8080 с дополнительными модулями памяти оказывалось дорого.

Модель 8748 (Intel) – микроконтроллер на основе 17 000 транзисторов, состоит из процессора типа 8080, постоянной памяти в 1 Кбайт для размещения программ, оперативной памяти 64 байт, 8-разрядный таймер, 27 шин ввода/вывода. Микросхема имела коммерческий успех.

Следующая модель 8051 на 60 000 транзисторах появилась в 1980 г., где процессор был более быстрый, 4К постоянной и 128 байт оперативной памяти, 32 шины ввода/вывода, последовательный порт и два 16-разрядных таймера. Вскоре вышли другие модификации и сформировалось семейство микроконтроллеров MSC-51.

Функциональность микросхем определялось программным обеспечением, которое прошивалось в постоянной памяти. Для создания ПО использовались модели 8751 и 8752, предусматривающие возможность тестирования ПО и перезаписи памяти, т.о. можно было корректировать и отлаживать ПО. Разработанное и отлаженное ПО передавалось производителю, который прошивал память программ уже без возможности перезаписи.

В настоящее время объем продаж микроконтроллеров на несколько порядков выше объема продаж процессоров Pentium, и только в 2001 году 8-разрядные стали продаваться лучше 4-х разрядных. Микросхемы серии 8051 одна из более распространенных.

Популярность объясняется ценой (10-15 центов в партии), производством микросхем по лицензии Intel занимаются множество фирм, поскольку микросхемы выпускаются давно, то для них имеется программное обеспечение и средства его разработки.

*Актуальность архитектурных исследований. На одном из первых компьютеров мира EDSAC (1949 г., Кембридж) время такта 2 микросекунды (2\*10-6), можно было 2\*n операций выполнить за 18\*n миллисекунд, т.е. 100 операций в секунду. Вычислительный узел суперкомпьютера HP V2600 время такта составляет 1.8 наносекунды (1.8\*10-9 секунд), а пиковая производительность около 77 миллиардов арифметических операций в секунду. Таким образом производительность возросла в семьсот миллионов раз, а тактовая частота около 1000 раз.*

#### *Единицы измерения*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Основные метрические приставки** | | | | **Память** | |
| **Порядок** | **Приставка** | **порядок** | **Приставка** | **Объем** | **Название** |
| 10-3 | милли | 103 | кило | 210 | Кбайт |
| 10-6 | микро | 106 | мега | 220 | Мбайт |
| 10-9 | нано | 109 | гига | 230 | Гбайт |
| 10-12 | пико | 1012 | тера | 240 | Тбайт |
| 10-15 | фемто | 1015 | пета | 103 бит/с | Кбит/с |
| 10-18 | атто | 1018 | экса | 106 бит/с | Мбит/с |
| 10-21 | зепто | 1021 | зета | 109 бит/с | Гбит/с |
| 10-24 | йокто | 1024 | йота | 1012 бит/с | Тбит/с |

## Тема 5. Организация компьютерных систем

Цифровой компьютер состоит из связанных между собой процессора, памяти и устройств ввода-вывода.

### 5.1. Процессоры



Центральный процессор – мозг компьютера. Его задача – выполнять программы, находящиеся в основной памяти. Процессор состоит из нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. АЛУ выполняет арифметические и логические операции. В центральном процессоре находится память для хранения промежуточных результатов и некоторых команд управления – регистровая память.

Самый важный регистр – счетчик команд (указатель команд), который указывает, какую команду следует выполнять дальше. Еще имеется регистр команд, в котором находится команда, выполняемая в данный момент. Это самая быстрая память. Разделяют регистры общего назначения и специальные. Количество регистров и их организация определяется архитектурой компьютера.

#### 5.1.1. Устройство центрального процессора

Внутреннее устройство тракта данных типичного фон-неймановского компьютера приведено на рис. 1.1. *Тракт данных* состоит из регистров, АЛУ и нескольких соединяющих шин.

Выполнение команд

Центральный процессор выполняет каждую команду за несколько шагов:



1. вызывает следующую команду из памяти и переносит ее в регистр команд;



1. меняет положение указателя команд, который теперь указывает на следующую команду;
2. определяет тип вызванной команды;
3. если команда использует слово из памяти, определяет местонахождение этого слова;
4. переносит слово (если это необходимо) в регистр центрального процессора;
5. выполняет команду;
6. переходит к шагу 1, чтобы начать выполнение следующей команды.

Такая последовательность шагов (*выборка – декодирование – исполнение*) является основой работы всех компьютеров.

Описание работы центрального процессора можно представить в виде программы. Такая программа называется программой-интерпретатором.

Сама возможность написать программу, имитирующую работу центрального процессора показывает, что программа не обязательно должна выполняться реальным процессором. Написание программ-интерпретаторов широко используется при разработке компьютерных систем. Интерпретатор разбивает команды на маленькие шажки.

Первые компьютеры содержали небольшое количество команд и эти команды были простыми. Но разработка более сложных компьютеров привела к появлению более сложных команд, т.к. при наличии более сложных команд программы выполняются быстрее, но это потребовало дополнительных затрат именно в аппаратную часть вычислительных систем.

Развитие программного обеспечения и требования к совместимости команд привели к тому, что сложные команды стали выполняться и на дешевых компьютерах.

В конце 50-х годов компания IBM решила, что производство семейства компьютеров, совместимых программно выгодно. Чтобы описать уровень совместимости компания ввела понятие архитектура. Новое семейство компьютеров должно иметь одну общую архитектуру и много разработок, отличающихся по цене и производительности.

Для переноса сложных команд на дешевые компьютеры использовалась интерпретация. Эта технология была предложена Уилксом в 1951 году и позволяла разрабатывать дешевые компьютеры, которые могли выполнять большое количество команд. Так была создана System/360. В дорогих компьютерах сложные команды выполнялись аппаратными методами, а в дешевых – программными. Просты компьютеры с интерпретатором имеют и некоторые другие преимущества:

* возможность фиксировать неправильное выполнение команды и даже восполнять недостатки аппаратного обеспечения;
* возможность добавлять новые команды при минимальных затратах;
* структурированная организация, которая разрабатывать, проверять и добавлять новые команды.

Со временем интерпретаторы стали применяться практически во всех компьютерах. Эта тенденция достигла своего пика в разработке компьютера VAX, у которого было несколько сотен команд и более 200 способов определения операндов в каждой команде. Это привело к появлению множества второстепенных команд. Производительности было уделено мало внимания, что и стало роковым для VAX, а также для производителя DEC. В 1998 году Compaq купила DEC.

#### 5.1.2. RISC и CISC

В 1980 году группа разработчиков университета Беркли (Дэвид Паттерсон и Карло Секвин) начала разработку процессоров без использования интерпретаторов. Для обозначения этого понятия был придуман термин RISC (Reduced Instruction Set Computer). Новые процессоры существенно отличались от коммерческих процессоров того времени. Они имели ограниченный набор команд, а преимущество состояло в том, что выбор и декодирование команд осуществляется быстро.

RISC противопоставляется CISC (Complex Instruction Set Computer). Типичный CISC-компьютер – VAX.

Учитывая преимущества RISC-компьютеров можно было предположить, что они стали бы доминировать над СISC-компьютерами. Однако этого не произошло.

Во-первых, RISC-компьютеры были несовместимы по программному обеспечению с другими моделями.

Во-вторых, идеи RISC воплощались в компьютерах с СISC архитектурой. Например, процессоры Intel начиная с 486, содержат ядро RISC, которое выполняет наиболее простые (а, значит, наиболее распространенные) команды за один такт тракта данных. При таком «гибридном» подходе работа происходит не так быстро, как у RISC, но программная совместимость оправдывает эту потерю в быстродействии.

#### 5.1.3. Принципы разработки современных компьютеров

Существует ряд принципов разработки, которые называются *принципами RISC*, которым по - возможностям стараются следовать производители универсальных компьютеров.

*Все команды непосредственно выполняются аппаратными средствами.* Таким исключается этап интерпретации, что обеспечивает высокую скорость выполнения. Однако сложные и/или редко встречающиеся команды могут быть разбиты на несколько команд и выполняться как микропрограмма.

*Компьютер должен начинать выполнение большого числа команд.* Процессор MIPS-500 способен приступить к выполнению 500 млн ком/с. Этот принцип полагает, что параллелизм может играть главную роль в повышении производительности.

*Команды должны легко декодироваться.* Предел количества вызываемых команд в секунду зависит от процесса декодирования. Декодирование необходимо для того, чтобы определить, какие ресурсы необходимы для выполнения команды. Полезны любые средства для упрощения процедуры декодирования.

*К памяти должны обращаться только команды записи/считывания.*

*Должно быть большое количество регистров.*

#### 5.1.4. Параллелизм на уровне команд

Параллелизм – возможность выполнять две и более операций одновременно. Существует две основные формы параллелизма: параллелизм на уровне команд и на уровне процессоров. В первом случае параллелизм осуществляется в пределах отдельных команд и обеспечивает выполнение большого количества команд в секунду. Во втором случае над одной задачей работают одновременно несколько процессоров.

##### 5.1.4.1. Конвейеры

Главное препятствие высокой скорости выполнения команд является обращение в память. Для разрешения этой проблемы разработчики придумали средство для вызова команд из памяти заранее. Эти команды помещались в набор регистров, который называется буфером выборки с упреждением.

Одним из самых простых и наиболее распространенным способов повышения быстродействия процессоров является конвейеризация процесса вычислений. В любом процессоре машинная команда проходит ряд этапов обработки, например: выборку команды из памяти (ВК), вычисление адреса операнда (ВА), выборку операнда из памяти (ВО), собственно выполнение операции.

В процессоре последовательной ЭВМ для выполнения этих функций используется единственное устройство, поэтому время выполнения команды равно: tk = tвк + tва + tво + tалу.

Для того, чтобы уменьшить время выполнения команды можно для каждой функции ввести собственное оборудование. В таком процессоре любая команда проходит через все устройства, находясь на каждом этапе время Δt, постоянное для всех этапов. Теперь, несмотря на то, что время выполнения операции может даже увеличиться, результаты выполнения команд появляются через интервалы времени Δt = tk/n, где n – число этапов конвейера команд.

Приведенная временная диаграмма строилась в предположении, что в потоке отсутствуют команды условных переходов, все команды имеют одинаковое время прохождения каждого этапа. Наличие команд условных переходов будет вынуждать переход к командам, которые в данный момент отсутствуют в конвейере, что потребует опустошения и и повторного заполнения конвейера. Неодинаковая длина команд будет приводить к приостановке конвейера.

Стандартный способ увеличения быстродействия конвейерного процессора состоит в следующем: в существующем варианте конвейера выбирается устройство с наибольшим временем срабатывания и разделяется на два или более устройств с меньшим временем срабатывания каждое.

На рис. 5.2. показан конвейер из 5 блоков, которые называются *стадиями.*



Рассмотрим конвейеризация устройства выборки команд. Поскольку на каждый полученный в АЛУ результат происходит одна выборка команды из памяти, то время выборки команды не должно превышать времени выполнения операции в АЛУ. Но запоминающие устройства могут иметь цикл обращения в значительно превышающий требуемый цикл конвейера. Выходом из сложившейся ситуации может являться расслоение памяти – разделение ее на множество автономных по функционированию блоков. Число этих блоков N = tmem/Δt, где Tmem – время обращения в память, Δt – цикл конвейера. Обычно N выбирают как целую степень 2.

Некоторые варианты такой памяти изображены на рис.5.3.

Для организации чтения со сдвигом в регистры адреса (РА) блоков памяти 1 … 4 с интервалом Δt подается новый адрес из счетчика адресов команд. С таким же сдвигом по времени на выходе блока памяти будут появляться коман­ды, которые затем по­сту­пают в буфер команд (БК) (память типа очере­ди). Из БК считываются команды, которые ука­зы­ва­ет счетчик команд для БК. При достаточ­ной длине БК там могут, например, целиком раз­мещаться циклы. В этих случаях выборка команд будет происходить толь­ко из БК. В этих случаях выборка команд не будет мешать выборке операндов.



Во втором случае за один цикл памяти в БК заносится несколько команд, операции в БК выполняются как и ранее.

##### 5.1.4.2. Суперскалярные архитектуры

Один конвейер хорошо, а два лучше. Одна из возможных схем процессора с двойным конвейером приведена на рис. 1.4



Сначала конвейеры использовались только в компьютерах RISC. Конвейеры в процессорах Intel появились только с 486, который содержал один конвейер, а Pentium – 2 конвейера из 5-ти стадий. Эти конвейеры были не одинаковы. Главный конвейер (u-конвейер) мог выполнять произвольные команды, а второй (v-конвейер) мог выполнять только простые команды с целыми числами, а также одну простую команду с плавающей точкой.

Имеются сложные правила определения, является ли пара команд совместимой для того, чтобы выполняться параллельно.

Переход к четырем конвейерам возможен в принципе, но требует значительных аппаратных затрат. Вместо этого используют другая идея – один конвейер с большим количеством функциональных блоков, как показано на рис. 5.5. (Pentium II).



В 1987 году для такого подхода был введен термин *суперскалярная архитектура*, хотя впервые такая структура была использована в конце 50-х в компьютере CDC 6600.

#### 5.1.5.Параллелизм на уровне процессов

Конвейерная и суперскалярная архитектура обычно увеличивает скорость работы всего лишь в 5 – 10 раз. Чтобы увеличить скорость в 50, 100 и более раз необходимо разрабатывать компьютеры с несколькими процессорами.

##### 5.1.5.1. Матричные компьютеры

Многие научно-технические задачи предполагают использование массивов и других упорядоченных структур. Для таких структур может быть применен принцип параллельной обработки команд. Т.Е. одно действие выполняется над множеством данных. Существует два основных метода, которые используются для быстрой обработки векторных операций.

*Матричный процессор* состоит из большого числа сходных процессоров, которые выполняют одну и ту же последовательность команд применительно к разным наборам данных. Первым таким процессором был ILLIAC IV, структурная схема которого изображена на рис.1.6. Он содержит решетку 8 х 8 элементов процессор/память. Имеется блок контроля, который рассылал команды, выполняемые всеми элементами одновременно. Первоначально планировалось иметь четыре решетки, но из-за высокой стоимости была реализована только одна. Одна решетка (или сектор) выполняет 50 млн оп/с с п.з.



Для программистов *векторный процессор* очень похож на массивно-параллельный. Но в отличие от первого, все операции сложения выполняются в одном блоке суммирования, который имеет конвейерную структуру.

Оба типа процессора работают с массивами данных. Но у матричного процессора имеется столько же суммирующих устройств, сколько элементов в массиве, то векторный процессор содержит *векторный регистр*, состоящий из набора стандартных регистров. Эти регистры последовательно загружаются из памяти при помощи одной команды.

Векторные процессоры менее эффективны, но значительно дешевле. Векторный процессор может быть добавлен к обычному процессору и те части программы, которые могут быть преобразованы в векторную форму и выполняются векторным блоком, а остальная часть обычным процессором.

В настоящее время матричные процессоры не выпускаются, но принцип, на котором они реализованы используется, по-прежнему актуален. Например, аналогичная идея используется в наборах ММХ- и SSE- командах Pentium 4.

##### 5.1.5.2. Мультипроцессоры

Система нескольких параллельных процессоров, разделяющих общую память, называется *мультипроцессором.* Поскольку каждый процессор может записывать или считывать информацию из любой области памяти, их работа должна быть согласована. Схема одного из способов реализации этой идеи приведена на рис. 5.7. – система с общей памятью. Такие системы производят многие компании.



Увеличение количества процессоров порождает конфликты при использовании общей памяти. Для преодоления этого недостатка была предложена модель, схема которой приведена на рис. 5.8 – система с локальной памятью.



Главное различие между системами с общей и индивидуальной памятью состоит в характере адресной системы. В машине с общей памятью адресное пространство является единым, следовательно, если в программах нескольких процессоров встречается одна и та же переменная x, то эти процессоры будут обращаться в одну и ту же ячейку памяти. Это ведет к следующим последствиям:

1. Наличие общей памяти не требует физического перемещения данных между взаимодействующими программами, которые параллельно выполняются на разных процессорах. Это упрощает программирование и исключает затраты времени на межпроцессорный обмен.
2. Несколько процессоров могут одновременно обращаться к общим данным и это может привести к получению неверных результатов. Чтобы исключить такие ситуации, необходимо ввести систему синхронизации параллельных процессов, что усложняет механизмы ОС.
3. Поскольку при выполнении каждой команды каждым процессором необходимо обращаться в общую память, то требования пропускной способности шины этой памяти высоки, что и ограничивает число процессоров в системах с общей памятью.

В системах с индивидуальной памятью каждый процессор имеет независимое адресное пространство, и наличие одной и той же переменной x в программах разных процессоров приводит к необходимости физического перемещения данных между взаимодействующими программами в разных процессорах. Однако, поскольку основная часть обращений производится каждым процессором в собственную память, то требования к шине ослабляются, и число процессоров в системах с распределенной памятью может достигать сотен.

##### 5.1.5.3. Мультикомпьютеры

Создание больших мультипроцессорных систем представляется сложной задачей, основная проблема которой состоит, чтобы связать все компьютеры с памятью. Чтобы избежать этих проблем было предложено создавать системы, состоящие из большого числа взаимосвязанных компьютеров, у каждого из которых имелась своя собственная память, а общей памяти нет. Такие системы называются *мультикомпьютерами.* Обмен информацией происходит посредством посланий (сообщений). Каждый компьютер связан не обязательно со всеми другими, а только с некоторыми. В качестве топологии используются 2D, 3D, деревья, кольца. Для того, чтобы сообщения доходили до адресата, они могут проходить через один или несколько промежуточных компьютеров. Сейчас запусеаются в работу мультикомпьютеры, содержащие около 10 000 процессоров. Более подробно организацию таких систем рассмотрим в конце курса.

### 5.2. Кэш-память

Процессоры всегда работали быстрее, чем память. Поэтому для повышения производительности системы в целом необходимо искать способы ускорения выборки команд и данных из памяти.

Существуют технологии сочетания малой, но быстрой памяти с большой и меленной и по разумной цене. Маленькая память с высокой скоростью обращения называется кэш-памятью. Сейчас мы рассмотрим принципы работы такой памяти, а более подробно этот вопрос будет обсуждаться позднее.

Основная идея кэш-памяти – в ней находятся слова, которые чаще всего используются. Процессор сначала обращается в кэш. Если там нет требуемого слова, он обращается в память.

Эффективность использования кэш памяти определяется тем, насколько «удачно» она заполнена. Основная идея заполнения кэш-памяти заключается в том, что, как правило, последовательно используемые данные и команда расположены поблизости. Если определенное слово вызывается из памяти, то вместе с ним считываются и записываются в кэш соседние слова.

Разработка кэш-памяти очень важна для процессоров с высокой производительностью. При разработке кэш-памяти принципиальное значение имеет решение следующих вопросов:

* размер кэш-памяти;
* размер строки (cache lines) кэш-памяти (совместно переписываемые слова при обращении к основной памяти);
* устройство памяти (способ определения наличия требуемого слова);
* расположение данных и команд – *смежная память* (данные и команды располагаются совместно или *раздельная* (данные и команды располагаются в разных памятях;
* количество блоков кэш-памяти.

### 5.3. Вспомогательная память

Сколько бы памяти не было, ее всегда будет мало. Иерархическая структура памяти является традиционным решением проблемы хранения большого количества информации. Структура изображена на рис. 1. 10.



По мере продвижения по структуре сверху вниз возрастают три параметра:

* увеличивается время доступа;
* увеличивается объем памяти;
* увеличивается объем информации, получаемых на одну у.е.

### 1.4. Процесс ввода-вывода

Большинство персональных компьютеров имеют структуру, сходную с приведенной на рис. 5.11.



На рисунке все компоненты соединены одной шиной, что соответствует логической структуре компьютера. Однако физически существуют различные шины.

Если контроллер считывает данные или записывает их в память без участия центрального процессора, то говорят, что осуществляется прямой доступ к памяти (Direct Memory Access DMA). Когда передача данных заканчивается, контроллер вызывает *прерывание*, по которому ЦП приостанавливает работу текущей программы и выполняет особые процедуры – *программу обработки прерываний.* Это необходимо, чтобы проверить правильность выполненных операций и сообщить операционной системе о завершении процесса ввода-вывода.

Шина используется не только контроллером, но и ЦП. Возможны ситуации, когда процессор и контроллер одновременно хотят получить доступ к шине – в этом случае особая микросхема (*арбитр шины*), чья очередь первая. Обычно предпочтение отдается устройствам ввода-вывода.

С увеличением производительности процессоров и памяти шина перестала справляться с нагрузкой. Разработка новой шины потребовала бы замены и контроллеров, поэтому IBM была вынуждена сохранить старую шину, которая сейчас называется шиной *ISA (стандартная промышленная шина)*. Поскольку быстродействия шины не хватало, многие компании начали производить компьютеры с несколькими шинами. Сейчас самой популярной является шина *PCI (взаимодействие периферийных компонентов).* Она была разработана компанией Intel, но все патенты были сделаны всеобщим достоянием, чтобы вся компьютерная индустрия могли перенять эту идею.

Типичная конфигурация приведена на рис. 5.12.

В такой конфигурации процессор общается с контроллером памяти по специальной шине с высокой скорость передачи данных. Периферийные устройства с высокой скоростью передачи данных могут подсоединяться к шине PCI. Шина PSI имеет параллельное соединение с шиной ISA, чтобы можно было использовать контроллеры ISA и соответствующие устройства.



## Тема 6. Микроархитектурный уровень

### 6.1. Микроархитектура

Над цифровым логическим уровнем находится микроархитектурный уровень. Его задача – интерпретация уровня команд в управляющие сигналы (команды) для цифровых устройств. Строение микроархитектурного уровня зависит от того, каков уровень архитектуры команд, от стоимости и назначения компьютера.

В настоящее время уровень архитектуры команд часто содержит простые команды, которые выполняются за один цикл (системы RISC). В других системах (например Pentium II) на этом уровне имеются более сложные команды, выполняемые за несколько циклов.

Для того, чтобы выполнить команду небходимо

* Найти операнды в памяти;
* Считать операнды;
* Выполнить операцию;
* Записать полученный результат в память.

Управление уровнем команд со сложными командами отличается от управления уровнем с простыми операциями, т.к. выполнение сложных команд требует определенной последовательностти операций.

Сейчас хотелось бы описать общие принципы разработки микроархитектурного уровня. К сожалению, таких общих принципов не существует и каждая разработка индивидуальна. Поэтому мы будем рассматривать конкретные примеры. Рассмотрм некоторые понятия присущие всем разработкам.

Микроархитектура содержит микропрограмму, располагаемую в ПЗУ. Микропрограмма должна вызывать, декодировать и выполнять команды. Поскольку аппаратное обеспечение состоит из вентилей и регистров, то микропрограммы должна выдавать команды, управляющие этими вентилями и регистрами.

Принято считать, что разработка микроархитектуры – это проблема программирования. Каждая команда интерпретируется как функция, которая реализуется микропрограммой.

Микропрограмма содержит набор переменных, к которым имеют доступ все функции. Эти переменные называются *состоянием* компьютера. Каждая ыункция изменяет некоторые переменные, формируя новое сосотяние компьютера. Например, счетчик команд указывает на местонахождение функции (команды уровня архитектуры команд), которую нужно выполнить следующей.

Каждая команда состоит из нескольких полей, каждое из которых выполняет определенную функцию. Первое поле называется *кодом операции.* Это поле определяет действие, которое не необходимо выполнить. Другие поля указывают на операнды, типы и т.д. Чем сложнее команда, тем больше полей требуется для ее описания.

#### 6.1.1. Тракт данных



*Тракт данных* – это часть центрального процессора, состоящего из АЛУ и его входов и выходов. В общем тракт данных содержит ряд регистров и АЛУ. Регистры и АЛУ связаны между собой шинами. Под действием управляющих сигналов содержимое определенных регистров передается на вход АЛУ, а результат записывается в требуеый регистр. Укрупненная и упрощенная структурная схема тракта данных приведена на рис. 3.1.

Функционирование АЛУ зависит от управляющих сигналов, имеет два входа. Схема сдвига имеет свое управление.

За один цикл АЛУ можно считать и записать один и тот же регистр. Процессы считывания и записи происходят в разных частях цикла. Содержимое одного из регистров выдается на шину С в начале цикла и сохраняются там на протяжении всего цикла. Затем АЛУ выполняет операцию, результат которой через схему сдвига поступает на шину С. После стабилизации сигналов на шине С ее содержимое передается в один или несколько регистров. Одним из нескольких регистров может быть тот, от которого поступил сигнал на шину В. Для реализации этих действий должна быть выполнена синхронизация тракта данных.

#### 6.1.2. Синхронизация тракта данных

Синхронизация тракта данных иллюстрируется рис. 6.2.



Хотя в трактe данных нет запоминающих устройств, для прохождения сигналов требуется некоторое время. Для правильной работы тракта данных требуется жесткая синхронизация и достаточно длительный цикл. Никаких синхронизирующих сигналов на вход тракта данных в течение цикла не поступает. Поэтому, необоснованное сокращение длительности цикла может привести к ошибкам работы тракта данных. АЛУ и схема сдвига работают постоянно, но в течение времени Δw+Δx входные сигналы не действительны, а в течении времени Δw+Δx+Δy недействительны выходные сигналы.

#### 6.1.3. Работа памяти

Работа с памятью может быть осуществлена двумя способами: через порт с пословной адресацией (регистры MAR и MDR) и через порт с байтовой адресацией (MBR – буферный регистр). Регистры MAR и MDR используются для чтения и записи слов, а регистры MBR и РС – для считывания программы на уровне архитектуры команд в виде потока байтов.

#### 6.1.4. Микрокоманды

Для управления трактом данных необходимы управляющие сигналы (29), которые можно разделить на 5 функциональных групп:

* Сигналы для записи данных из шины С в регистры (9);
* Сигналы для разрешения передачи содержимого регистров на шину В и в АЛУ (9);
* Сигналы управления АЛУ и схемой сдвига (8);
* Сигналы управления регистрами MAR\MDR (2);
* Сигналы управления регистрами PC\MBR (1).

Значения этих сигналов определяют операции для одного цикла тракта данных. Если был установлен сигнал обращения в память в k-ом цикле, то данные из памяти могут появиться в регистрах MDR(MBR) только в конце следующего k+1-го цикла, а использовать эти данные можно только k+2 цикле и то только в том случае, если эти данные были в КЭШ-памяти.

#### 6.1.5. Разработка микроархитектурного уровня

При разработке микроархитектурного уровня постоянно приходится искать компромисс между желаниями и возможностями. При разработке центрального процессора очень важную роль играет выбор между высокой скоростью и низкой стоимостью, между сложностью программ и сложностью аппаратного обеспечения.

##### 6.1.5.1. Скорость и стоимость

Существует три основных подхода , которые позволяют увеличить скорость выполнения операций:

1. Соркащение количества циклов, необходимых для выполнения кманды.
2. Упрощение организации машины таким образом, чтобы можо было сделать цикл короче.
3. выполнение нескольких операций одновременно.

Первые два подхода очевидны, но существуют различные способы их реализации. Число циклов, необходимых для выполнения операции азывается *длиной пути*. Длину пути можно уменьшить за счет введения дополнительного оборудования и использования параллельного выплнения команд. Основные подходы:

* + в микропрограмме, реализующей команду, находят циклы, для выполнния которых не требуется работа АЛУ. Такие циклы ставить в конце последовательности микрокоманд.
  + Введение дополнительной шины. Т.е. к АЛУ подвести две шины и на входы подавать сигналы с произвольных регистров. Таким образом исключается цикл передачи одного из операндов в Акк, но усложняется кодировка микрокоманды и аппратура -- *переход к трехшинной архитектуре*.
  + Введение блока выборки команд: процедура выборки следующей команды передается отдельному блоку – *команды из памяти должны вызываться специализированным функциональным блоком.*

##### 6.1.5.2. Конвейерная архитектура

Следующий рассматриваемый вариант усовершенствования архитектуры – ввести в машину больше параллелизма.

Длительность цикла определяется временем, необходимым на прохождение сигнала через тракт данных. В цикле тракта данных есть три основных компонента:

1. Время, необходимое на передачу значений выбраных регистров на входы АЛУ.
2. Время рабты АЛУ и схемы сдвига.
3. Время на передачу полученых значений в регистры и сохранение результатов.

Таким образом один цикл можно разбить на 3 цикла, более которкие. Продвижение микрокоманды осуществляется по трем устройствам. Следующая микрокоманда может быть начата до окончания предыдущей. Проблема: RAW-взаимозависимость (Read After Write – чтение после записи) – последующей операции могут понадобиться данные, которые еще не готовы. В таком случае возникает простой.

Это простейший конвейер с 3 стадиями, количество стадий (степень параллелизма может быть увеличена).

#### 3.1.6. Увеличение производительности.

Усовершенствования компьютеров распадаются на две категории: усовершенствование реализации и усовершенствование архитектуры.

Усовершенствование реализации – такие способы построения нового процессора и памяти, после применения которых система работает быстрее, но архитектура не меняется. Это означает, что старые программы будут работать на новой машине. Это очень нравится потребителям. Например, улучшение производительности от 80386 к 80486, Pentium, Pentium Pro, Pentium II происходило без изменения архитектуры.

Однако возникаем момент, когда старая архитектура себя исчерпала и единственный способ развивать технологии дальше – начать новую разработку. Таким революционным скачком было появление RISC в 80-х годах.

##### 3.1.6.1.КЭШ-память

Одним оз основных вопросов при рзработке компьютеров является построение такой системы памяти, которая могла бы передавать операнды процессору с той же скоростю, с которой она их обрабатывает. Однако, производительность процессора растет значительно быстрее, чем быстродействие памяти и относительно процессора память работает все медленне с каждым десятилетием и эта ситуация все ухудшается.

Одним из способов решения этой проблемы является добавление КЭШ-памяти. Основная технология – введение разделеной КЭШ-памяти – отдельно для операндов и отельно для команд. В такой КЭШ-памяти операции могут начинаться независимо, что увеличивает пропускную способность системы в целом.

Помимо этого между КЭШ-памятью и основной паятью часто помещают КЭШ-паять второго уровня.

Существует два типа локализации адресов:

* *Пространственая локализация*—основана на вероятности того, что в скором времени потребуется обратиться к ячейкам памяти, которые расположены рядом с недавно выбранными ячейками;
* *Временная локализация –*имеет место, когда недавно запрашиваемые ячейки запрашиваются снова. Принцип временной локализации используется в тех случаях, когда решается вопрос о том, какой элемент выкинуть из КЭШ-памяти в случае промаха. Обычно отбрасываются элементы, к которым не было обращений.

Во всех типах КЭШ-памяти используется следующая модель: основная память разделяется на блоки фиксированного размера, которые называются строками КЭШ-памяти. Строка состоит из нескольких последовательных байтов (от4 до 64). При обращении к памяти сначала проверяется наличие требуемой иформации в КЭШ-памяти и в случае промаха из КЭШ-памяти удаляется строка, а на ее место помещается требуемая из основной памяти. Трока больше, чем вызывемый элемент.

##### 6.1.6.2. Прогнозирование ветвлений

Современные компьютеры сильно конвейеризорованы, они могут содержать десять и более стадий. Но конвейеры дают высокое быстродействие только на линейном коде, а в случае наличия ветвления возникает проблема с выбором следующей команды.

Первые конвейеризированные процессоры простаивали до тех пор, пока не становилось известно, куда нужно выполнить переход.

Современные машины содержат средства, позволяющие прогнозировать переходы. Первое предположение заключается в том, что переход будет назад, т.к. если переход находится в конце цикла, то переход назад более вероятен. Если переход предсказан неправильно, то необходимо отменить выполненные команды.

*Динамаческое прогнозирование ветвлений.* Один из самых простых способов – хранить специальную таблицу (в аппаратном обеспечении), в которую центральный процессор записывает условные преходы, когда они встречаются и там их можно искать, если он снова появляется. В такой таблице хранится адрес перехода и бит, который указывает, был ли сделан этот переход. Прогноз состоит в том, что программа пойдет тем же путем, что и в предыдущий раз. Если прогноз не верен, то бит меняется.

*Статическое прогнозирование ветвлений.* Динамическое прогнозирование во время работы программы и это их положительное качество. Однако реализация динамического прогнозирования ветвлений требует специализированного и достаточно дорогого аппаратного обеспечения. Такое прогнозирование осуществляется компилятором.

В некоторых машинах (UltraSPARC) введен дополнительные команды перехода, которые содержат бит, который указывает, совершать переход или нет.

##### 6.1.6.3. Исполнение с изменением последовательности и подмена регистров

Большинство современных компьютеров являются конвейеризированными и суперскалярными. Это означает, что там есть блок выборки команд, который заранее вызывает команды из памяти и передает их в блок декодирования. Блок декодирования передает декодированные команды в соответствующие функциональные блоки для выполнения.

При такой организации возникает следующая проблема: если выполняемой команде нужно значение, получаемое в предыдущей, то она не может начаться до тех пор, пока значение не будет получено. Существуют и другие виды взаимозависимости.

Для устранения этой проблемы используется подход, который заключается в том, что изменяется последовательность выполнения команд таким образом, чтобы зависимые команды не стояли друг за другом.

Возможна ситуация, при которой возникает конфликт при обращении к регистрам: необходимо записать информацию в регистр, а там еще хранятся не востребованные данные. Этого можно избежать, если записывать информацию не в занятый регистр, а куда-то в другое место (можно временно). Для реализации указанных возможностей необходимо вводить дополнительные аппаратные и программные средства.

##### 6.1.6.4. Спекулятивное выполнение

Компьютерные программы можно разбить на *базовые элементы,* каждый из которых представляет собой линейную последовательность команд с точкой входа и точкой выхода в конце. Базовый элемент не содержит никаких управляющих структур. Базовые элементы связываютя между собой операторами управления. Рассмотренный выше прием переупорядочения команд эффективен внутри базового элемента.

Большинство базовых элементов очень короткие и в них недостаточно параллелизма.

Для повышения эффективности нужно сделать так, чтобы переупорядочение команд можно было применить не только в пределах одного базового элемента. Выгоднее всего передвинуть потенциально более медленный базовый элемент передвинуть вперед, чтобы его выполнение началось раньше. Такой операцией может быть операция считывания из памяти, операция с плавающей точкой и т.д. Перемещение операции вверх называется *подъемом.*

В этом случае возникает проблема: понадобится ли эта операция? Выполнение команды до того, как стало известно, понадобится ли она, называется *спекулятивным выполнением.* Чтобы использовать эту технологию, требуется поддержка компилятора, аппаратного обеспечения и некоторое усовершенствование архитектуры.

В связи со спекулятивным выполнением команд возникают некоторые проблемы. Важно, чтобы ни одна из спекулятивных команд не имела окончательного результата, который нельзя отменить. Эта проблема решается путем подмены регистров.

Вторая проблема: что делать, если запрашиваемые спекулятивной командой данные не находятся в КЭШ-памяти? В ряде современных компьютеров в такой ситуации команда не выполняется.

### 6.2. Микроархитектура процессора Pentium 4

#### 6.2.1. Общий обзор системы Pentium 4

Pentium 4 – поддерживает 32-битные операнды и арифметику, 64-битные операции с плавающей точкой, а также 8-ми и 16-ти битные операции, унаследованные от предыдущих моделей. Процессор может адресовать до 64 Гбайт памяти и считывать слова по 64 бита за раз.

Микроархитектура Pentium 4, называемая NetBurst, представляет собой решительный отход от принципов архитектуры, используемой в процессорах Pentium Pro, Pentium II, Pentium III. Упрощенная схема архитектуры приведена на рис 3.2.



*Подсистема памяти* включает объединенный КЭШ второго уровня (L2), логика доступа к внешнему ОЗУ. Объем КЭШа от 256 Кбайт до 1 Мбайт. Длина строки 128 байт. На рисунке не показан блок предварительной выборки, который пытается перенести данные из основной памяти в КЭШ второго уровня до того, как они затребованы. Из КЭШа второго уровня данные могут передаваться в другие блоки КЭШ-памяти на очень высокой скорости, теоретическая пропускная способность – 96 Гбайт/с.

Блок предварительной обработки выбирает команды из L2 и декодирует их в порядке выполнения команд в программе. Каждая команда разбивается на последовательность RISC-операций. Таким образом, команда ISA процессора Pentium 4 преобразуется в последовательность RISC-операций, которые исполняются RISC-ядром микросхемы. Этот механизм позволяет соединить устаревший набор CISC-команд с современным трактом данных.

Декодированные операции отправляются в КЭШ трас (trace chase), в роли которого выступает КЭШ команд первого уровня. Поскольку кэшируются не исходные команды, а декодированные операции, то тем самым исключается повторное декодирование и извлечение команд из КЭШа. В этом одно из отличий архитектуры NetBurst. Здесь же выполняется прогнозирование ветвлений.

В блок планировщика команд (*контроль исполнения с изменением последовательности*) команды передаются в порядке, определяемым программой. Выполняться команды могут в другой последовательности. Возвращаются они в «правильной» последовательности, за это отвечает блок пересортировки.

*Блок исполнения* объединяет специализированные блоки, которые работают параллельно. Данные они получают из регистрового файла и КЭШа первого уровня.

#### 6.2.2. Конвейер NetBurst

Упрощенная схема конвейера Pentium 4 приведена на рис. 3.3.

Блок предварительной обработки получает инструкции из КЭШа второго уровня порциями по 64 бит. Они декодируются в микрооперации и помещаются в КЭШ трасс. Емкость КЭШа трасс составляет 12 000 микроопераций и по производительности сопоставим с традиционным КЭШем первого уровня на 8 или 16 кбайт.

В КЭШе трасс каждые шесть микроопераций объединяются в группу, занимающую одну строку. Микрооперации из одной строки выполняются без нарушения последовательности, хотя они могут быть образованы на из последовательных команд. Практикуется также объединение трассируемых строк.

Если для выполнения команды требуется более четырех микроопераций, то она не декодируется и не помещается в КЭШ трасс. Эта операция помечается специальным маркерами система производит поиск микроопераций в памяти микрокоманд.

Встретившись с командой безусловного перехода, блок декодирования ищет предсказанный объект в *буфере объектов перехода* (Branch Target Buffer – *BTB*) первого уровня и продолжает декодирование соответствующего адреса. В КЭШе BTB L1 сохраняются 4 000 последних переходов. Если необходимая команда отсутствует в таблице, то применяется статическое прогнозирование ветвлений. Для прогнозирования микроопераций перехода применяется *буфер трасс объектов* перехода или *ВТВ трасс.*

Второй компонент конвейера – логика исполнения с изменением последовательности – получает данные из КЭШа трасс емкостью 12000 микроопераций. При поступлении из блока предварительной обработки каждой последующей микрооперации (за цикл их поступает 3) блок распределения и подмены регистрирует ее в таблице, состоящей из 128 записей и *называемой буфером переупорядочивания команд* (ReOrder Buffer, *ROB*). В этом буфере хранятся данные о состоянии микрооперации вплоть до пересортировки ее результатов.



Затем блок распределения и подмены проводит проверку на предмет доступности ресурсов, необходимых для выполнения микрооперации. Если ресурсы свободны, то микрооперация устанавливается в очередь на выполнение. Для микроопераций, выполняемых в памяти и вне памяти устанавливаются разные очереди. Если выполнение микрооперации в данный момент невозможно, она откладывается, однако обработка других микроопераций продолжается. Этот принцип позволяет поддерживать загрузку всех функциональных блоков на максимально высоком уровне. В каждый момент времени могут одновременно обрабатываться 126 команд, причем 48 из них могут загружаться из памяти, а 24 – сохраняться впамяти.

Блок распределения и подмены помещает готовые к выполнению операции помещает готовые к выполнению операции в одну из двух очередей. Четыре планировщика ответственны за извлечение микрокоманд из очередей. Каждый планировщик регламентирует обращение к тем или иным ресурсам:

1. Планировщик 1 – АЛУ1 и блок смещения операций с ПТ.
2. Планировщик 2 – АЛУ2 и блок исполнения операций с ПТ.
3. Планировщик 3 – команды загрузки.
4. Планировщик 4 – команды сохранения.

Поскольку планировщики и АЛУ работают на скорости, в двое превышающую тактовую частоту, то первые два планировщика могут передавать две микрооперации за цикл. С учетом того, что АЛУ также работает с удвоенной частотой, процессор Pentium 4 с тактовой частотой 3 ГГц может выполнять 12 млрд. целочисленных оп/с.

Два целочисленных АЛУ не одинаковы. АЛУ1 выполняет любые арифметические и логические операции, а также операции перехода. АЛУ2 способно выполнять только операции сложения, вычитания, сдвига и циклического сдвига. Не идентичны также и два блока исполнения операций с плавающей точкой. Первый выполняет только сдвиги и SSE-команды, а второй поддерживает арифметические операции с плавающей точкой, а также ММХ и SST команды.

АЛУ и блоки исполнения операций с ПТ получают данные от двух регистровых файлов емкостью по 128 записей. В этих файлах хранятся операнды и результаты. В силу подмены регистров восемь из них содержат регистры, доступные на уровне архитектуры команд (EAX, EBX, ECX и т.д.).

КЭШ данных первого уровня имеет емкость в 8 Кбайт. В отличие от КЭШа трасс, эти данные не декодируются. Длина строки – 64 байт. В случае промаха в КЭШ первого уровня происходит обращение к КЭШу второго уровня. В любой момент времени в состоянии исполнения могут находиться до четырех запросов, направленных из КЭШа первого в КЭШ второго.

Так как операции выполняются с нарушением последовательности, то результаты могут быть помещены в КЭШ первого уровня только после пересортировки. Такую операцию выполняет блок *пересортировки*.

Если команда сохранения прошла пересортировку результатов, но предшествующие команды еще не завершились, КЭШ обновлять нельзя и результат передается в буфер незавершенных команд. Буфер рассчитан на 24 команды сохранения. Если одна из последующих команд загрузки пытается считать сохраненные данные, то эти данные непосредственно из буфера будут направлены к команде. Этот процесс называется *перенаправлением для загрузки* (store-to-load-forwarding).

Таким образом, в Pentium 4 поддерживается старая архитектура команд, которая выполняется на современном RISC – ядре. Это достигается путем деления команд Pentium на микрооперации, их кэширования и прередачи (по три микрооперации за раз) конвейеру, где они выполняются с помощью нескольких АЛУ, которые в оптимальных условиях обрабатывают до 6 микроопераций за цикл. Микрооперации выполняются с отклонением от исходной последовательности, но возвращаются и сохраняются в КЭШе первого уровня в заданном порядке.

### 6.3. Микроархитектура процессора UltraSPARC III Cu

UltraSPARC III Cu – это 64-разрядная машина (по данным и адресам), но в целях совмещения с предыдущими версиями, она воспринимает 32- разрядные операнды, адреса и программы. Шина памяти 128-битная. «Cu» означает, что проводники микросхемы сделаны из меди, а не из алюминия. У меди ниже сопротивление, поэтому при той же ширине проводников можно обеспечить более высокое сопротивление

Вся серия SPARC изначально представляла собой систему RISC. Большинство команд – трехадресные, поэтому они очень хорошо подходят для конвейеризации и нет необходимости разбивать команды CISC на микрооперации RISC (как в Pentium). Однако, в последние годы появляются новые команды обработки мультимедийной информации, для выполнения которых требуются специальные устройства.

#### 6.3.1. Общий обзор системы UltraSPARC III

Структурная схема UltraSPARC III представлена на рис. 3.4. В целом она проще микроархитектуры NetBurst.



КЭШ команд имеет объем в 32 Кбайт, длина строки – 32 байт. Поскольку большинство команд UltraSPARC III занимает 4 байта, то в КЭШе можно хранить 8 000 команд. (Pentium 4 – 12 000).

*Блок вызова команд* подготавливает для выполнения до 4 команд за цикл. В случае промаха кэш -памяти первого уровня количество вызывемых команд уменьшается. При обнаружении условного перехода происходит обращение к *таблице переходов* емкостью 16 000 записей. Повысить надежность прогнозирования ветвлений помогают дополнительные биты, связанные с каждым словом в КЭШе. Подготовленные команды поступают в 16-командный буфер, который сглаживает направленный в конвейеры поток команд.

*Блок исполнения целочисленных операций* состоит из двух АЛУ и короткого конвейера для обработки команд перехода. Кроме того здесь имеются регистры.

*Блок исполнения операций с ПТ* состоит из 32 регистров и трех независимых АЛУ. Этот же блок выполняет графические операции.

Блок загрузки/сохранения управляет командами записи и считывания. Содержащиеся в нем тракты данных обеспечивают соединение с тремя КЭШами. *КЭШ данных* представляет собой традиционную кэш-память первого уровня емкостью 64 Кбайт и длиной строки 32 байта. *КЭШ предвыборки* емкостью 2 Кбайт используется для обеспечения упреждающей выборки. *КЭШ записи* представляет собой небольшой по объему (2 Кбайт) блок КЭШ-памяти, предназначенный для объединения результатов записи, а следовательно, для оптимизации потребления ресурсов Шировой (256-разрядной) шины, ведущей к КЭШу второго уровня.

В микросхеме UltraSPARC III предусмотрена **логика управления доступом к памяти**. Состоит из трех компонентов: *системного интерфейса, контроллера КЭШа второго уровня, контроллера памяти*. Системный интерфейс обеспечивает взаимодействие с памятью по 128-разрядной шине. Теоретически объем памяти может достигать 8 Тбайт, однако пазмер печатной платы, на которой расположен процессор допускает только 16 Гбайт.

Контроллер КЭШа второго уровня сопряжен с объединенным КЭШем второго уровня, расположенный вне микросхемы. Поскольку КЭШ расположен вне микросхемы и иметь объем до 8 Мбайт. Длина строки зависит от объема КЭШа – от 64 байта для 1 Мбайт до 512 байт в КЭШе объемом 8 Мбайт.

Контроллер памяти преобразует 64-разрядные виртуальные адреса в 43-разрядные физические. Организация памяти будет рассмотрена в последующих разделах.

#### 6.3.2. Конвейеризация системы UltraSPARC III Cu

Система UltraSPARC III содержит конвейер с 14 стадиями. Некоторые стадии различны для команд с целыми числами и команд с п/з (рис. 3.5).



*Стадия A* (Address generation – генерация адреса). На этом этапе определяется адрес следующей команды. Так как прогнозировать ветвление за один цикл невозможно, то команда, следующая за командой перехода всегда выполняется.

*Стадия P* (Preliminary fetch – предварительная выборка) вызывает команды из КЭШа команд первого уровня (до четырех за цикл). Для выявления переходов и проверки правильности прогноза используется таблица переходов.

*Стадия F* (Fetch – выборка) завершается выборка команд и их передача в КЭШ команд.

*Стадия B* (Branch target – обнаружение объекта перехода) происходит декодирование выбранных команд. Если среди них обнаружены спрогнозированные переходы, то они передаются на стадию А для непосредственной выборки соответствующих команд.

*Стадия I* (Instruction group formation – группировка команд) – команды сортируются по группам, в зависимости от того, к каким функциональным блокам они обращаются. Имеются различия между АЛУ для выполнения целочисленных операций, а также для АЛУ с ПТ и графических операций. В обоих случаях различаются наборы команд, которые способны выполнять те или иные АЛУ. Сортировка команд производится на ступени I.

*Стадия J* (Instruction stage grouping – извлечение команды из очереди) – команды подготавливаются к отправке в блок выполнения во время следующего цикла. В течение одного цикла на стадию R можно передать до четырех команд.

*Стадия R* (Register –регистр) – производится поиск регистров, необходимых для обработки команд целочисленных операций. Запросы на предоставление регистров для команд с ПТ перенаправляются в соответствующий регистровый файл. Если необходимый регистр оказывается недоступным по причине занятости предыдущей командой, то текущая команда приостанавливается, а все последующие команды блокируются. В отличие от Pentium 4 в UltraSPARC III Cu команды вне исходной последовательности не выполняются.

*Стадия E* (Execution – выполнение) предназначена для выполнения целочисленных команд. Время выполнения большинства целочисленных операций – один цикл. Сразу по завершении операции обновляется регистровый файл. Некоторые сложные целочисленные операции передаются в специальный блок. Команды загрузки/сохранения на этом этапе начинаются, но не заканчиваются. На стадии Е также обрабатываются команды условного перехода и определяется их направление. В случае неверного прогноза сигнал направляется на ступень А и конвейер освобождается.

*Стадия С* (Cache – кэш) – завершается доступ к КЭШ-памяти первого уровня. Здесь же определяются результаты команд, предусматривающих чтение данных из памяти.

*Стадия М* (Miss – промах) – производится обработка слов, запрошенных, но не найденных в КЭШ-памяти первого уровня. Здесь же выполняются операции знакового расширения и выравнивания байтов, четвертинок и половинок слов, найденных в КЭШе первого уровня. Эля операций загрузки с ПТ, у которых в КЭШе предвыборки имеет место попадание, на этом этапе удается получить результаты. По соображениям синхронизации КЭШ предвыборки при обработке целочисленных данных не задействуется.

По соображениям синхронизации КЭШ предвыборки при обработке целочисленных данных не задействуется.

*Стадия W* (Write – запись) – результаты записываются в регистровый файл рабочего регистра.

*Стадия X* (eXtend – продленное выполнение) – завершается большинство команд с ПТ и графических команд. Перед формальной пересортировкой результатов, происходящих на ступени D результаты этих команд предоставляются последующим командам путем перенаправления для загрузки.

*Стадия T* (Tape – перехват) – перехватываются исключения, связанные с целочисленными командами и командами с ПТ. Если возникло исключение или прерывание, то все ранее запущенные команды должны быть завершены, а запуск последующих отменен.

*Стадия D* – состояние целочисленных регистров и регистров с ПТ фиксируется в соответствующих архитектурных регистровых файлах. При возникновении исключения или прерывания видимыми становятся именно эти значения, а не содержимое рабочих регистров. Кроме того, на этой ступени результаты всех завершенных команд сохранения записываются в КЭШ записи (вместо КЭШа первого уровня). В конечном итоге строки этого КЭШа переписываются в КЭШ второго уровня, минуя КЭШ первого уровня.(содержимое КЭШей не пересекается). Эта схема упрощает сборку мультипроцессорных систем на базе UltraSPARC).

Приведенная схема значительно упрощена работы конвейера значительно упрощена.

### 6.4. Микроархитектура процессора 8051

#### 6.4.1. Общий обзор системы 8051

Диаграмма микроархитектуры 8051 представлена на рис. 6.7.



Центральное положение на микросхеме занимает основная шина. *Регистр АСС* представляет собой основной арифметический регистр, через него проходят почти все арифметические команды.

*Регистр В* применяется для операций умножения и деления, а также используется как временный регистр. *Регистр SP* является указателем стека. В *регистре IR* содержатся команды, выполняемые в данный момент.

*Регистры ТМР1 и ТМР2* – защелки, в которые помещаются операнды. Результаты работы *АЛУ* записываются в любой регистр, который доступен через основную шину. Коды состояний записываются в регистр PSW (Program Status Word – слово состояния программы).

В 8051 предусмотрены независимые модули памяти для размещения данных и кода. Емкость ОЗУ для данных составляет 128 (8051) или 256 (8052) байт. *Регистр RAM ADDR* адресует эту память. Емкость памяти кода может составлять 64 Кбайт (при условии расположения вне микросхемы), поэтому разрядность *регистра ROM ADDR* составляет 16 разрядов. *Регистр DPRT* (Double PointTeR –двойной указатель) – 16-разрядный временный регистр, предназначенный для управления и сборки 16-разрядных адресов. *Регистр РС* представляет собой 16-разрядный счетчик команд. Регистр PC incrementer – специальный аппаратный модуль, выполняющий роль псевдорегистра. Ни к PC, ни к PC incrementer нельзя обратиться через основную шину. *BUFFER* – еще один 16-разрядный регистр временный регистр. Каждый 16-разрядный регистр 8051состоит из двух 8-разрядныых регистров, с каждым из которых можно выполнять разные операции.

Таймеры 16-разрядные, позволяют управлять выполнением приложений в масштабе реального времани.

Предусмотрены четыре 8-разрядных *порта*, которые могут управлять 32 внешними кнопками, индикаторами, датчиками и т.п.

Процессор относится к синхронным процессорам, большинство операций завершается за один цикл. Каждый цикл делится на 6 частей, каждая часть называется **состоянием**. В первом состоянии следующая команда вызывается из ПЗУ и отправляется в регистр IR. Во втором состоянии проводится декодирование команды, а значение РС увеличивается на единицу. В третьем состоянии подготавливаются операнды. В четвертом состоянии один из операндов по основной шине передается и размещается в ТМР1. В этом же состоянии возможно копирование содержимого ACC в регистр ТМР2, после чего два операнда готовы к выполнению. В пятом состоянии происходит выполнение команды. В шестом состоянии происходит передача результата операции в регистры Одновременно в регистре ROM ADDR производится подготовка к вызову следующей команды.

### 6.5. Сравнение Pentium, UltraSPARC и 8051

Машина Pentium 4 содержит старый набор команд CISC. UltraSPARC –система RISC. 8051 – простой 8-разрядный процессор, применяемый как встроенный компьютер.

Все три машины имеют сходные функциональные блоки. Все функциональные блоки принимают микрооперации, которые содержат код операции, два входных и один выходной регистр. Все они могут выполнять микрооперации за один такт. Все они конвейеризированы и применяют прогнозирование ветвления. Все содержат раздельную кэш-память для команд и данных.

Главное различие между Pentium 4, UltraSPARC – переход от набора команд к функциональному блоку. Pentium разбивает команды CISC для перехода к трехрегистровому формату и делает из больших команд маленькие микрооперации.

UltraSPARC ничего не надо делать, поскольку ее первоначальные команды уже представляют собой маленькие удобные микрооперации. Вот поэтому большинство новых архитектур команд – архитектуры типа RISC (если нет причин использовать другое).

Микросхема 8051 очень проста. ЕЕ архитектура больше напоминает RISC-архитектуру, поскольку команды выполняются за один цикл без разбиения на составные части. Но ни конвейеризаци, ни кэширование на предусмотрены.

## Тема 7. Уровень архитектуры команд

Исторически уровень архитектуры команд развивался раньше других и сначала был единственным. В наши дни этот уровень называют «архитектурой» машины. Уровень архитектуры команд имеет особое значение: он является связующим звеном между аппаратным и программным обеспечением.

Все разработчики считают, что нужно транслировать программы, написанные на различных языках высокого уровня, в общую промежуточную форму – на уровень архитектуры команд – и соответственно конструировать аппаратное обеспечение, которое может непосредственно выполнять программы этого уровня (уровня архитектуры команд). Уровень архитектуры команд связывает компиляторы и аппаратное обеспечение, это язык, понятный и компиляторам и аппаратному обеспечению (рис. 7.1).



Когда появляется новая машина, первые вопросы, который задает покупатель: «Совместима ли машина с предыдущими версиями?», «Могу ли я установить мою старую операционную систему?» и «Будут ли работать старые программы?». Покупатели редко рвутся выбросить старое ПО и заняться разработкой нового с нуля.

Этот факт заставляет разработчиков сохранять один и тот же уровень команд в разных моделях и обеспечивать совместимость ПО (по крайней мере снизу вверх). Разработчики должны обеспечить совместимость на уровне команд, но они могут делать что угодно с аппаратным обеспечением. Таким образом возникает задача построения лучших машин но совместимых с предыдущими версиями.

Какую архитектуру команд можно считать хорошей?

1. Хорошая архитектура должна определять набор команд, которые можно эффективно реализовать в современной и будущей технике, что приводит к рентабельным разработкам на несколько поколений.
2. Хорошая архитектура команд должна обеспечивать ясную цель для оттранслированной программы. Поскольку уровень архитектуры команд является промежуточным между программной и аппаратной частью, то он должен быть удобен как для разработчиков аппаратного обеспечения, так и для составителей программного обеспечения.

### 7.1 Общий обзор уровня архитектуры команд

#### 7.1.1. Свойства уровня команд

Уровень команд – это то, каким представляется компьютер программисту машинного языка. Программа уровня архитектуры команд – это то, что выдает компилятор. Чтобы произвести программу уровня команд, составитель компилятора должен знать, какая модель памяти используется в компьютере, какие регистры, типы данных и команды имеются в наличии и т.д.

В связи с этим определением такие вопросы как программируется ли микроархитектура или нет, конвейеризирован компьютер или нет, является ли компьютер суперскалярным не относится к уровню архитектуры команд, однако знания об этом могут повысить эффективность разрабатываемых компиляторов.

Для одних архитектур уровень команд определяется формальным документом, который выпускается промышленным консорциумом, для других – нет. Например, для системы V9 SPARC и JVM имеются официальные определения. Цель такого официального документа – дать другим производителям выпускать машины данного типа, чтобы эти машины могли выполнять одни и те же программы и получать одни т те же результаты. В документе говорится, какая модель памяти используется, какие имеются регистры, какие действия выполняют команды, но не описывается микроархитектура.

В таких документах присутствуют нормативные разделы, в которых излагаются требования, и информационные разделы, призванные помочь разработчикам, но не являющиеся формальной частью.

Для уровня архитектуры набора команд процессора Pentium 4 такого документа не нет, поскольку Intel не хочет, что бы другие производители выпускали микросхемы Pentium 4. Компания Intel доже обращалась в суд, что бы запретить производство своих микросхем другим производителем, но проиграла процесс.

Другое важное качество уровня архитектуры команд состоит в том, что в большинстве машин поддерживается по крайней мере два режима:

* В *привилегированном режиме* запускается операционная система. Этот режим позволяет выполнять все команды.
* *Пользовательский режим* предназначен для запуска прикладных программ. Он не позволяет запускать некоторые потенциально опасные программы.

#### 7.1.2. Модели памяти

Во всех компьютерах память разделена на ячейки, которые имеют последовательные адреса. В настоящее время наиболее распространены ячейки в 8 разрядов (байт). Причиной применения байта – ASCII-символ, который занимает 7 разрядов и бит четности. Если в будущем будет доминировать кодировка UNICODE, то ячейки, возможно, станут 16-разрядными (24 лучше, чем 23).

Байты обычно группируются в слова по 4 или 8 байт. Многие архитектуры требуют, чтобы слова были выровнены. При этом память работает более эффективно.

Большинство машин имеет единое адресное пространство. В некоторых машинах содержатся отдельные адресные пространства для команд и данных. Такая система гораздо сложнее, чем единое адресное пространство, но зато она имеет два преимущества:

* Появляется возможность иметь 232 байтов для программы и 232 байтов для данных при размере регистра адреса в 32 разряда;
* Исключается сама возможность интерпретировать код как данные и наоборот.

Один из аспектов модели памяти – семантика памяти. Естественно ожидать, что если в программе идет запись по некоторому адресу, а затем считывание из этого же адреса, то мы получим толь что сохраненное значение. Но в некоторых машинах микрокоманды переупорядочиваются. Возникает опасность, что память не будет действовать так, как ожидалось (сопроцессор). Ситуация усложняется в случае с мультипроцессором, когда каждый процессор посылает свой запрос в память. Системные разработчики могут применять различные подходы к решению этих задач. В некоторых случаях запросы в память делают упорядоченными, в других – возлагают на разработчиков компиляторов (или даже программистов) заботу о правильном функционировании памяти. Все эти проблемы затрудняют работу разработчиков ПО, хотя надо отметить, что существуют модели памяти, в которых аппаратное обеспечение автоматически блокирует определенные операции с памятью связанные с зависимостью записи/считывания.

#### 7.1.3. Регистры

Во всех компьютерах имеется несколько регистров, которые видны на уровне архитектуры команд. Они нужны для хранения промежуточных результатов, для контроля выполнения программы и для других целей.

Регистры уровня команд можно разделить на две категории: специальные регистры и регистры общего назначения. Специальные регистры включают счетчик команд, указатель стека, а также другие регистры с особой функцией. Регистры общего назначения содержат ключевые локальные переменные, и промежуточные результаты. Их основная функция – обеспечить быстрый доступ к часто используемым данным (избегать обращений в память). Машины RISC с высокоскоростным процессором обычно содержат минимум 32 регистра общего назначения, и их количество постоянно растет.

В некоторых машинах все регистры взаимозаменяемы, в некоторых регистры общего назначения могут быть специализированы. Например, в Pentium 4 регистр EDX может использоваться в качестве регистра общего назначения, но который получает половину произведения и содержит половину делимого при делении.

Кроме регистров, доступных на уровне команд, всегда существует довольно большое количество регистров, доступных только в привилегированном режиме. Эти регистры контролируют различные блоки кэш-памяти, основную память, устройства ввода/вывода и другие элементы аппаратного обеспечения.

Существует регистр, который представляет собой «гибрид», доступный и в пользовательском и в привилегированном режимах. Это регистр PSW (Program State Word – слово состояния программы, который еще называют *флаговым.* Флаговый регистр содержит различные биты, необходимые центральному процессору. Самые важные из них – *коды условий*. Они устанавливаются в каждом цикле АЛУ и отражают состояние результата предыдущей операции:

* N -- результат отрицательный (Negative);
* Z -- результат равен нулю (Zero);
* V -- переполнение (oFerflow)
* C – перенос самого левого бита(Carry Out);
* F – перенос бита 3 (Auxiliary carry – служебный перенос);
* P – результат четный (Parity).

Флаговый регистр хранит не только коды условий, его содержимое в каждой машине может быть разным. Флаговый регистр обычно читается в пользовательском режиме, но некоторые поля могут записываться только в привилегированном режиме.

### 7.2. Общий обзор уровня команд машины Pentium 4

Процессор Pentium 4 развивался на протяжении многих лет. Основная архитектура команд обеспечивает выполнение программ, написанных для 8086 и 8088, а в машине даже содержатся элементы 8-разрядного процессора 8080. На процессор 8080, в свою очередь сильно повлияли требования совместимости с процессором 8008, который был основан на процессоре 4004 (4-битная схема).

Pentium 4 имеет 3 операционных режима, в двух из которых он работает как 8086.

*Real Address Mode* – режим реальной адресации (или просто реальный режим), полностью совместим с 8086. В этом режиме возможна адресация до 1 Мб физической памяти.

Существенным дополнением является *Virtual 8086 Mode* – режим виртуального процессора 8086. Этот режим является особым состоянием задачи защищенного режима, в котором процессор функционирует как 8086. На одном процессоре в таком режиме могут параллельно выполняться несколько задач с изолированными друг от друга ресурсами. При этом использование физического адресного пространства памяти управляется механизмами сегментации и трансляции страниц. Попытки выполнения недопустимых команд, выхода за пределы отведенного пространства памяти контролируются системой защиты. Когда пользователь WINDOWS начинает работу с MS-DOS, программа, которая действует под DOS, запускается в виртуальном режиме, чтобы программа WINDOWS не могла вмешаться.

*Protected Virtual Address Mode* – защищенный режим виртуальной адресации (или просто защищенный режим). В этом режиме процессор позволяет адресовать до 4 Гб физической памяти. В этом режиме доступны 4 уровня привилегий.

#### 7.2.1. Регистры

В Pentium имеется 31 регистр; они подразделяются на 16 регистров прикладного программиста (пользовательские регистры) и 15 регистров системного программиста (системные регистры).

Пользовательские регистры процессора изображены на рис. 7.2.

Регистры общего назначения по сути являются расширением 16-ти разрядных регистров, например EAX есть расширение AX, причем доступ к старым 16-ти разрядным и 8-ми разрядным частям регистра типа (AH) сохраняется. Роль регистров общего назначения сохранилась из 8086 и подробно на них останавливаться не будем.

В общем, каждый сегментный регистр определяет сегмент (т.е. блок смежных ячеек) памяти. Если в процессоре 8086 содержимое сегментного регистра прямо задает физический базовый адрес сегмента и максимальный размер всех сегментов составляет 64 Кбайт, то в процессорах 286 и старше содержимое сегментного регистра определяет сегмент косвенно, через так называемую дескрипторную таблицу (descriptor table). С помощью дескрипторной таблицы для каждого сегмента задаются базовый адрес, размер (limit) и право доступа (access rights), показывающие, какие программы и в каких операциях могут обращаться к конкретному сегменту. Производить прямые обращения по физическим адресам памяти программист не может.

Записываемые в сегментный регистр слова (16 бит) называют селекторами (selector), поскольку оно выбирает или «селектирует» один из сегментов из множества возможных.

Указатель команды EIP\IP предназначен для адресации команд внутри текущего сегмента кода.

Регистр флажков EFLAGS\FLAGS содержит флажки, которые подразделяются на восемь флажков состояния и шесть управления. Формат регистра приведен на рис. 7.3 (флажки управления помечены \*, а флажки состояния – ^).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 | 15 | 7 0 | *Аккумулятор*  *База*  *Счетчик (для сдвигов)*  *Данные*  *(в/вв – содержится адрес порта)*  *Указатель стека*  *Указатель базы*  *Индекс источника*  *Индекс получателя* |  |
| EAX AX | | |  |
|  | AH | AL |  |
| EBX BX | | |
|  | BH | BL |
| ECX CX | | |
|  | CH | CL |
| EDX DX | | | регистры общего назначе-ния |
|  | DH | DL |
| 31 | 15 | 7 0 |
| ESP | | |
|  | SP | |
| EBP | | |
|  | BP | |
| ESI | | |
|  | SI | |
| EDI | | |
|  | DP | |
|  | 15 | 7 0 |  |  |
|  | CS | | *Сегмент кода*  *Сегмент стека*  *Сегмент данных*  *Доп. сегмент данных*  *Доп. сегмент*  *Доп. сегмент* | сегмент-ные регистры |
|  | SS | |
|  | DS | |
|  | ES | |
|  | FS | |
|  | GS | |
| 31 | 15 | 7 0 |  |  |
| EIP | | |  | указатель команды |
|  | IP | |
| 31 | 15 | 7 0 |  |  |
| EFLAGS | | |  | регистр флажков |
|  | FLAGS | |

Рис. 7.2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ID | VIP | VIF | AC | VM | RF |
|  |  |  |  |  |  |  |  |  |  | \* | \* | \* | \* | \* | \* |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | NT | IOPL | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |
|  | ^ | ^ | ^ | \* | \* | \* | ^ | ^ |  | ^ |  | ^ |  | ^ |

Рис. 7.3

Биты 31-22, 15, 5, 3, 1 зарезервированы. При записи в эти биты должны устанавливаться в предварительно прочитанное состояние.

##### Флаги состояния:

* *CF (Carry Flag) –* флажок переноса (бит **0**). Этот бит устанавливается в **единицу**, если арифметическая операция вызвала перенос или заем из старшего бита результата. Программно доступен.
* *PF (Parity Flag) –* флажок четности (бит 2). Флажок устанавливается в единицу, если младшие восемь бит результата содержит четное число единичных бит.
* *AF (Auxiliary carry Flag) –* флажок вспомогательного переноса (бит 4). Устанавливается в единицу, если арифметическая операция вызвала перенос или заем из младшей тетрады. Главное применение флажка связано с десятичной арифметикой. Программно не доступен.
* *ZF (Zero Flag) –* флажок нуля (бит 6). Устанавливается в единицу, если результат операции равен нулю.
* *SF (Sing Flag) –* флажок знака (бит 7). Большинство команд устанавливает этот флажок в тоже состояние, что и старший бит результата.
* *OF (Overflow Flag) –* флажок переполнения (бит 11). Устанавливается в единицу, если в арифметических операциях со знаковыми операндами результат превышает допустимый диапазон.
* *IOPL( Input / Output Privilege Level) –* уровень привилегий ввода\вывода (биты 12 и 13). Это поле содержит два бита и показывает минимальный уровень привилегий для данной задачи. Более подробно будет рассмотрено в теме о системе защиты.
* *NT (Nested Flag) –* флажок вложенной задачи (бит 14). Более подробно будет рассматриваться при изучении многозадачности. Используется для для контроля за прерванными задачами и при вызове процедур.

##### Флаги управления.

* *TF (Trap Flag или Trace Flag) –* флажок покомандной работы (бит **8**). Если он установлен в 1, то после выполнения каждой команды генерируется внутреннее прерывание. Используется для отладки программ.
* *IF (Interrupt Flag) –* флажок прерывания (бит **9**). Если он установлен в единицу, то внешние прерывания разрешены по входу INTR. Флажок не влияет на прерывания по входу NMI.
* *DF (Direction Flag) –* флажок направления (бит **10**). Задает направление обработки цепочек. Доступен программисту.
* *RF (Resume Flag) –* флажок возобновления (бит **16**). Если флажок установлен в 1 можно маскировать некоторые особые случаи в отладке программы.
* *VM (Virtual Mode) –* флажок виртуального режима (бит **17**).
* *AC (Alignment Check) –* флажок контроля выравнивания (бит **18**). Если от установлен в 1, то разрешен контроль выравнивания. Рассматривать не будем.
* V*IF (Virtual Interrupt Flag)* – флаг виртуального прерывания (бит 19, Pentium ...). Используется в специальном расширенном режиме обработки прерываний, является виртуальным подобием флага IF. Применяется совместно с флагом VIP и позволяет обеспечивать нормальное функционирование устаревшего ПО.
* *VIP ((Virtual Interrupt Pending) –* ожидание виртуального прерывания (бит **20** Pentium ..).
* *ID (Id Flag) –* флаг доступности команды идентификации CPUID (бит 21, Pentium ...). Если программа или процедура способна установить или сбросить этот флаг, значит команда CPUID поддерживается.

Системные регистры GDTR, LDTR, IDTRпредназначены для хранения базовых адресов таблиц дескрипторов. Их еще называют р*егистрами управления памятью.* Они определяют местонахождение структур данных, управляющих сегментацией памяти. Регистр задачи **TR** применяется для слежения за тем, какая задача выполняется процессором в данный момент.

Для доступа к системным регистрам используются команды:

*LGDT, SGDT, LLDT, SLDT, LIDT, SIDT.*

*Регистр глобальной дескрипторной таблицы* GDTR имеет длину 48 бит. Содержит 32 битный (для 286 –24-битный) базовый адрес и 16-битный предел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 47 | 15 | 0 |  | |
| Линейный базовый адрес | Предел | | GDTR |  |

Рис. 7.4

*Регистр локальной дескрипторной таблицы LDTR* Видимая часть (которую можно программно читать и изменять) 10-байтного регистра содержит 16-разрядный селектор дескриптора LDT. Сам дескриптор LDT (базовый адрес, предел, атрибуты) автоматически загружается в скрытую часть LDTR из глобальной таблицы дескрипторов. Формат LDTR:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 15 |  |  | | | | |
| Селектор | |  |  | База | Предел | Атрибуты |  |
|  | |  | программно недоступно, загружается автоматически | | | | |

Рис. 7.4

*Регистр таблицы дескрипторов прерываний* содержит 32 битный (для 286 –24-битный) базовый адрес и 16-битный предел. Формат регистра:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 47 | 15 | 0 |  | |
| Линейный базовый адрес | Предел | | IDTR |  |

Рис. 7.5

*Регистр задачи TR.* Видимая часть регистра содержит 16-битный селектор дескриптора сегмента состояния задачи (TSS). Сам дескриптор TSS загружается автоматически в скрытую часть TR из глобальной таблицы дескрипторов. Формат регистра задачи совпадает с форматом регистра локальной дескрипторной таблицы.

Управляющие регистры впервые были использованы в 386 микропроцессоре. Ранее (286) было так называемое слово состояния машины, которое для сохранения совместимости полностью помещено в управляющий регистр CR0. Практически все режимы и специальные возможности микропроцессора устанавливаются определенными флагами в управляющих регистрах.

Доступ к управляющим регистрам возможен с помощью команд

*MOV CR, SMSW, LMSW, CLTS (очистить флаг переключени якоманд).*

*Регистр CR0*. Первые 16 бит ­– слово состояния машины. Биты NE, WP, AM, GW, CD были введены в регистр начиная с Intel486.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 19 | 18 | 17 | 16 | 15 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PG | CD | GW | \*......\* | | AM | \* | WP | \* ... \* | | NE | ET | TS | EM | MP | PE |
|  | | | | | | | | MSW (Слово состояния машины I286) | | | | | | | |

Рис. 7.6

**PE (разрешение защиты, бит 0)** – предназначен для переключения процессора (реальный/защищенный режим). PE=0 –режим реальной адресации, PE=1 –защищенный режим.

**MP (слежение за сопроцессором, бит 1)** – Если сопроцессор не установлен, то этот бит должен быть равен 0. Для процессоров со встроенным сопроцессором бит должен быть установлен в 1.

**EM (эмуляция сопроцессора, бит 2) –** предназначен для генерации особой ситуации для всех команд сопроцессора. При отсутствии сопроцессора флаг должен быть установлен. Для выполнения команд MMX – сброшен.

**TS (задача переключена, бит 3) –**устанавливается при каждом переключении задачи. Вызывает ситуацию исключения при выполнении команд сопроцессора, ММХ/3Dnow!, SIMD. Предназначен для исключения выполнения этих команд над данными, оставшимися от других задач. Команда CLTS очищает этот бит.

**ET (тип сопроцессора, Intel 386, 486)** – при ET =1 используется 32-битный протокол, а при ET=0 16 битный. В процессорах Pentium считается зарезервированным.

**NE (ошибка сопроцессора, бит 5, Intel486)** – в зависимости от того, как установлен этот бит происходит обработка ошибок сопроцессора: либо как внешнее прерывание, либо как внутреннее.

Механизмы реакции различных моделей процессоров и сопроцессоров на возникновение исключительных прерываний могут изменяться для различных моделей. (СМ. тех. описание)

**WP (защита записи, бит 16, Intel 486 ...)** – защищает страницы пользовательского уровня (определенные только для чтения) от записи программой–супервизором, работающей на более высоком уровне привилегий. Если WP=0, то супервизор может осуществлять запись в пользовательские страницы, защищенные от записи.

**AM (маска выравнивания, бит 18, Intel486 ...)** – если бит сброшен – контроль запрещен, а если установлен (совместно с флагом AC), то генерируется особая ситуация контроля выравнивания.

**NW (запрет сквозной записи, бит 29, Intel486 ...) –** предназначен для управления встроенной кэш-памятью.

**CD (разрешение работы кэш-памяти, бит 30, Intel486 ...) –** если СВ =0, то работа внутренней кэш-памяти разрешена, в противном случае – запрещена.

**PG (включение страничного механизма, бит 31, Intel386...)** – если PG=1, то страничный механизм включен, иначе –выключен. Используется только в защищенном режиме. Если этот бит установить в реальном режиме, то будет генерироваться ошибка общей защиты.

*Регистр CR1*.– зарезервирован.

*Регистр CR2*. если включен механизм страничной адресации и генерируется особая ситуация (страничная ошибка), то в этом регистре будет полный 32-разрядный адрес, поступление которого в блок страничной адресации вызвало ошибку.

*Регистр CR3*. При включенном страничном механизме (CR4.PAE=0) регистр CR3 (20 старших бит) содержат 20 старших бит физического адреса каталога страниц. Младшие 12 принимаются равными нулю. Начиная с процессора 486 используются еще два бита этого регистра (PWT и PCD), управляющие кэшированием. В процессорах (Pentium ...), которые поддерживают расширение физического адреса (CR4.PAE=1), в режиме расширенного адреса уже 27 разрядов содержат адрес таблицы указателей на каталоги страниц. Младшие 5 разрядов полагают равными 0.

**PWT (сквозная запись страниц, бит 3, Intel486 ...)** – используется для управления кэшированием страниц.

**PCD (запрещение кэширование страниц, бит 4, Intel486 ...)** – используется для управления кэшированием текущего каталога страниц.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 |  | | 3 | 4 | |  |  |  |
|  | Физ. адрес каталога страниц |  |  | PSD | PWT |  | | CR3 |  |

Рис. 7.7

*Регистр CR4.* Обеспечивает включение/выключение самых разнообразных режимов и дополнительных возможностей процессора. Впервые был использован в процессоре Pentium. Различные модификации процессора обеспечивают различные наборы возможностей. Поэтому перед программированием любых битов этого регистра следует с помощью команды CPUID проверить наличие возможностей для конкретной модификации процессора.

**WME (расширение виртуального 8086 режима, бит 0, Pentium ...)**

**PVI (виртуальные прерывания защищенного режима, бит 1, Pentium ...)**

**TSD (ограничение маркера времени бит 2, Pentium ...)**

**DE (расширение отладки бит 3, Pentium ...)**

**PSE (расширение размера страниц, бит 4, Pentium ...)** Если бит установлен в 1, то разрешает использование страниц расширенного размера (4 или 2 Мб) в зависимости от текущей разрядности физического адреса. В противном случае размер страниц ограничен 4 Кб.

**PAE (расширение физического адреса, бит 5, Pentium Pro ...) –**если бит установлен в 1, то разрешено использование 36-битной физической адресации.

**MCE (расширение контроля машины, бит 6, Pentium ...)**

**PGE (разрешение глобальных страниц, бит 7, Pentium Pro...)**

**PCE (разрешение счетчика производительности, бит 8, Pentium Pro ...) –** предназначен для разрешения мониторинга производительности в пользовательских программах.

**OSFXSR (использование команд быстрого сохранения/восстановления, бит 9, Pentium II...)** используется для быстрого восстановления/сохранения состояния FPU/MMX/SIMD.

**OSXMMEXCPT** **(разрешение особой ситуации SIMD, бит 10, Pentium III...) –** управляет реакцией процессора на SIMD-исключения.

##### Отладочные регистры

Регистры отладки DR0 –DR3 содержат линейные адреса четырех точек останова. Регистры DR4, DR5 – зарезервированы.

DR6 содержит информацию о том, какая из точек останова вызвала особую ситуацию отладки.

DR7 устанавливает режимы срабатывания точек отладки; служат целям отладки.

Регистры TR6, TR7 для работы с кэш–памятью.

##### Регистры FPU

Первые процессоры архитектуры х86 не поддерживали напрямую вычисления с плавающей точкой. Фирмой Intel выпускались отдельные устройства (микросхемы), именуемые сопроцессором (8087, 80287, 80387), реализующие такие вычисления. Сейчас FPU встраивается в сам процессор. Тем не менее, поскольку в х86 сопроцессор представляет собой отдельное устройство, необходимо представлять специфику его работы.

Сопроцессор подключается к системной шине параллельно с центральным (основным) процессором и может работать только с ним. Он не имеет своей индивидуальной программы и не может осуществлять выборку команд. Команды сопроцессора находятся в общем потоке команд. Если выбранная команда является командой центрального процессора, он выполняет ее обычным образом, а сопроцессор не привлекается. Когда выбирается команда сопроцессора, действие центрального процессора зависит от специфики конкретной команды. Если команда не связана с обращением к памяти, ЦП ее игнорирует, и переходит к следующей команде. Но если команда требует обращения к памяти, ЦП вычисляет адрес операнда и обращается к памяти, но адрес и данные «перехватывает» сопроцессор. После этого сопроцессор реализует конкретные действия по выполнению команды. Они могут производиться параллельно с дальнейшими действиями центрального процессора. Этот параллелизм требует некоторой синхронизации действий ЦП и сопроцессора.

*Синхронизация по командам.* Когда ЦП выбирает для выполнения команду сопроцессора, последний может быть занят выполнением своей предыдущей команды. Следовательно, перед каждой командой сопроцессора должна находиться специальная команда ЦП, которая только проверяет состояние сопроцессора и, если он занят, переводит ЦП в состояние ожидания (эти команды имеют мнемоники WAIT, FWAIT). Обычно АССЕМБЛЕР вставляет эти команды автоматически).

*Синхронизация данных.* Если выполняемая сопроцессором команда записывает операнд в ячейку памяти перед последующей командой центрального процессора, которая обращается к той же ячейке, также необходима команда проверки состояния сопроцессора. АССЕМБЛЕР учесть такие ситуации не может, поэтому ответственность за их использование ложится на программиста.

FPU x86 опирается на стековую организацию, особенности и преимущества которой уже обсуждались. Программная модель FPU представлена на рис. 7.8.

Основу программной модели составляет регистровый стек из 8 80-разрядных регистров R0–R7, называемых также численными или арифметическими регистрами. В них хранятся числа в так называемом расширенном вещественном формате. В любой момент трехразрядное поле TOP (top of stack) в слове состояния SW определяет регистр, являющейся вершиной стека и обозначаемый ST(0) или ST. Пример приведен на рис. 4.9. Здесь вершина стека R3 (ST(0)), регистр R4 –это ST(1), регистр R2 в самом низу стека и обозначается ST(7). Команды в FPU рассчитаны на такую относительную адресацию.

Операция загрузки в стек декремент поля TOP и загружает данные в новую вершину, извлечение из стека действует наоборот.

Особенностями этого регистрового стека является следующее: стек имеет кольцевую структуру; контроль использования стека должен осуществлять сам программист; в командах допускается явное и неявное обращение к регистрам стека с модификацией и без модификации поля TOP.

ТЭГИ

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 79 |  | | 63 | | 0 | 1 | 0 |
| R0 | S | E | | M | |  |  |  |
| R1 |  |  | |  | |
| R2 |  |  | |  | |
| R3 |  |  | |  | |
| R4 |  |  | |  | |
| R5 |  |  | |  | |
| R6 |  |  | |  | |
| R7 |  |  | |  | |
|  | | | | | |  | |
|  | 15 | 0 |  | | |
| CW | регистр управления | |  | | |
| SW | регистр состояния | |
| TW | Слово ТЭГов | |  | | |
|  | 47 | 0 | | |  |
| IP | указатель команды | | | |
| DP | указатель данных | | | |

Рис. 7.8

С каждым регистром стека ассоциируется двухбитный тег (признак) совокупность которых образует слово тегов TW. Тег регистра R0 находится в младших битах этого слова, а тег регистра R7 – в старших. Тег в общем виде показывает, что содержится в соот­ветствующем регистре:



* 00 – допустимое ненулевое число;
* 01 – нуль;
* 10 – специальное значение ( о них позже);
* 11 – пустой регистр.

При своей инициализации FPU записывает 11 во все теги регистров, отмечая регистры как пустые. Попытка обращения к такому регистру вызывает особый случай (прерывание). Регистры CW, SW будут рассмотрены позже; регистры IP и DP используются в процедурах при обработке особых случаев.

Устройство FPU имеет два программно–доступных 16-разрядных регистра, содержимое которых определяет режим его работы и текущее состояние. Регистры представлены на рис. 4.10.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CW | x | x | x | x | R | C | P | C | x | x | PM | UM | OM | ZM | DM | IM |
| SW | B | C3 | T | O | P | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |

Рис.7.10

*Слово управления CW.*Определяет для FPU варианты обработки численных данных. Шесть младших разрядов представляют собой индивидуальные маски особых случаев, т.е. необычных и ошибочных ситуаций, обнаруживаемых устройством FPU при выполнении программы. К битам маски относятся:

* PM – маска точности (истинный результат операции не представим в формате получателя;
* UM – маска анти переполнения;
* OM – маска переполнения:
* ZM – маска деления на ноль;
* DM – маска денормализованного операнда;
* ID – маска недействительной функции.

Если любой из этих бит установлен в состояние 1, то возникновение соответствующего особого случая не будет вызывать прерывания процессора; в противном случае – будет.

Поясним понятие недействительной операции — сюда могут относиться корень квадратный из отрицательного числа и т.д.

Управление точностью. Поле PC (2 байта) определяет точность вычислений 24 бита (PC = 00), 53 бита (10) и 64 бита (11). По умолчанию используется 64 бита.

Управление округлением. Устройство FPU имеет четыре режима округления, выбираемые полем RC слова управления:

00 – округление к ближнему;

01 – округление вниз (к -∞);

10 – округление вверх (к +∞);

11 – усечение (к 0).

По умолчанию принимается округление к ближнему.

*Слово состояния SW.* Занимает 16 бит и отражает общее состояние устройства FPU. В слове состояния SW младшие 6 бит отведены для регистрации особых случаев; назначение этих бит аналогично слову CW. При обнаружении любого из незамаскированных особых случаев устройство FPU устанавливает в состояние 1 соответствующий флажок и вызывает прерывание ЦП.

Бит SF нарушения стека предназначен для того, чтобы регистрировать ошибки в работе регистрового стека. Устройство FPU устанавливает его в состояние 1, если команда вызывает переполнение стека (особый случай недействительной операции).

Биты C0–C3 кода условия похожи на арифметические флажки центрального процессора в регистре EFLAGS. Они регистрируют особенности результатов команд сравнения. Эти биты, в основном, применяются для условных переходов. Команда FSTSW AX передает слово состояния в регистр AX. Затем командой SAHF модно переместить биты C0 –C3 и биты флажков.

Трехадресное поле вершины стека TOP показывает, какой из внутренних регистров данных является текущей вершиной стека.

Бит B занятости служит для совместимости с 8087 (совпадает с ES).

#### 7.2.2. Технология MMX

**Технология MMX ориентирована на приложения мультимедиа, 2D/3D-графику и коммуникации.** Это расширение базовой архитектуры появилось только после второго поколения процессоров Pentium. Основная идея ММХ заключается в одновременной обработке нескольких элементов данных за одну инструкцию —технология SIMD (Single Instruction–Multiple Data). Расширение ММХ использует новые типы упакованных целочисленных данных:

* упакованные байты (Packed Byte) –восемь байт;
* упакованные слова (Packed word)–четыре слова;
* упакованные двойные слова (Packed doubleword)–два двойных слова;
* учетверенное слово (Quadword)–одно слово.

Эти типы данных специальным образом могут обрабатываться в регистрах ММХ0–ММХ7, представляющие собой младшие биты стека 80–битных регистров FPU. Эти регистры, также как и регистры FPU не могут быть использованы для адресации памяти. Совпадение регистров ММХ и FPU накладывает ограничения на чередование кодов ММХ и FPU. Частое чредование кодов ММХ и FPU снижает производительность за счет необходимости сохранения и восстановления весьма объемного контекста FPU – и забота об этом лежит на программисте.

Еще одна особенность технологии ММХ – поддержка *арифметики с насыщением* (при переполнении фиксируется максимально/минимально допустимое число). Граничные значения определяются типом данным.

В систему команд введено дополнительно 57 инструкций для одновременной обработки нескольких единиц данных. Одновременно обрабатываемое 64-битное слово может содержать как одну единицу обработки, так и 8 однобайтных, 4 двухбайтных или 2 четырехбайтных операнда.

Новые инструкции включают следующие группы:

* арифметические (+,- в различных режимах, умножение, комбинацию умножения и сложения);
* сравнение элементов данных;
* преобразование форматив;
* логические, выполняемые на 64-разрядными кодами;
* сдвиги (арифметические и логические)
* пересылки данных между регистрами ММХ и целочисленными регистрами и памятью;
* очистка ММХ – установка признаков пустых регистров в слове ТЕГов.

Регистры ММХ в отличие от регистров FPU адресуются физически, а не относительно указателя стека ТОР. Любая инструкция ММХ обнуляет поле ТОР регистра состояния FPU. В слове ТЕГов свободному регистру соответствует комбинация 11, а все остальные комбинации соответствуют занятому регистру. После каждой операции ММХ биты ТЕГов используемого регистра обнуляются. Неиспользуемые в ММХ биты (79-64) регистров FPU заполняются единицами, поэтому ошибочное использование данных ММХ инструкциями FPU приведет к исключению. Команда EMMS предназначена для предназначена для очистки контекста ММХ т.е. помечает все регистры как пустые.

Наличие поддержки ММХ определяется по биту 23 регистра EDX после вызова инструкции CPUID(1).

Технология 3DNow! введена фирмой AMD в процессорах K6–2, расширяет возможности ММХ. Она позволяет оперировать с новым типом данных – парой упакованных чисел в формате с плавающей точкой. Эти числа занимают по двойному слову в 64-разрядных регистрах ММХ. Процессор К6­–2 имеет два исполнительных блока, которые способны одновременно выполнять операции с плавающей точкой над своими регистрами. Каждая такая операция занимает всего два такта, операции полностью конвейеризируются. Таким образом, с конвейеров процессора за каждый такт могут сходить четыре результата операций с плавающей точкой. В систему команд добавлена 21 новая инструкция ММХ, большая часть которых предназначена для обработки упакованных чисел с плавающей точкой. Имеется также новая целочисленная ММХ инструкция для усреднения 8 пар 8-битных чисел. Имеется команда быстрого переключения FPU-MMX. Технология 3Dnow! программно совместима с прежними процессорами и операционными системами, поскольку регистры ММХ отображаются на регистры FPU и для них работают механизмы сохранения контекста в многозадачных ОС.

Поддержка 3Dnow! определяется по биту 31 регистра EDX после вызова CPUID(8000\_0001h).

#### 7.2.3. Регистры SIMD

В процессорах, начиная с Pentium III, возможно использование специального архитектурного расширения, предназначенного для потоковой обработки данных. Это расширение включает дополнительный набор команд, дополнительные режимы генерации ошибок и исключений, дополнительные внутренние регистры и форматы данных, воспринимаемые процессором. Данное расширение – усовершенствование команд мультимедийной обработки ММХ.

Доступ ко всем SIMD–регистрам возможен только с помощью новых SIMD–команд. Перед использованием данного расширения ПО должно с помощью команды CPUID проверить, поддерживается ли оно конкретной моделью микропроцессора.

*Регистры общего назначения SIMD* Восемь регистров общего назначения (ХММ0 …ХММ7) вводятся в процессор Pentium III … для использования командами потоковой обработки. В отличие от ММХ-регистров, эти регистры не являются отображаемыми – это новые физические регистры, доступ к которым осуществляется с помощью новых команд. Поэтому при совместном использовании команд SIMD, FPU, MMX не возникает никаких проблем. Регистры ХММ0 … ХММ7 могут использоваться только для хранения данных, но не для косвенной адресации.

### 7.3. Общий обзор архитектуры уровня команд системы UltraSPARC III

Архитектура SPARC была впервые введена в 1987 г. и была первой архитектурой промышленного назначения типа RISC. Изначально она была 32-разрядной, но UltraSPARC III – это 64-разрядная машина.

Структура памяти проста: память представляет собой линейный массив из 264 байтов. Память настолько велика, что в настоящее время она не реализуема. Современные реализации имеют размер адресного пространства 244 (UltraSPARC III), но в будущем память будет увеличиваться.

Одна из проблем разработки архитектуры заключается в том, что архитектура команд ограничивает размер адресуемой памяти.

Архитектура команд SPARC достаточно проста, хотя организация регистров усложнена для повышения эффективности вызова процедур.

В системе UltraSPARC имеется две группы регистров: 32 64-битных регистров общего назначения и 32 регистра с плавающей точкой. Варианты названий регистров приведены в таблице 7.1.

Таблица 7.1

|  |  |  |
| --- | --- | --- |
| Регистр | **Вариант названия** | Функция |
| R0 | GO | Связан с 0. Все, что сохраняется в этом регистре, просто игнорируется |
| R1- R7 | G1—G7 | Содержит глобальные переменные |
| R8 – R13 | O0 – O5 | Содержит параметры вызываемой процедуры |
| R14 | SP | Указатель стека |
| R15 | O7 | Временный регистр |
| R16 – R23 | L0 – L7 | Содержит локальные переменные для текущей процедуры |
| R24 – R29 | I0 – I5 | Содержит входные параметры |
| R30 | FP | Указатель на основу текущего стекового фрейма |
| R31 | I7 | Адрес возврата для текущей процедуры |

В действительности процессор UltraSPARC имеет более 32 регистров общего назначения, но видимы для программиста только 32 в любой момент времени. Эта особенность, называемая «регистровыми окнами», предназначена для повышения эффективности вызова процедур. Идея состоит в том, что имеется несколько наборов регистров, точно также, как существует несколько фреймов в стеке. Ровно 32 регистра видны в текущий момент; регистр CWP (Current Window Pointer – указатель текущего окна) следит за тем, какой набор регистров используется в данный момент.

Команда вызова процедуры скрывает старый набор регистров и предоставляет новый набор, который может использовать вызванная процедура. Однако некоторые регистры из старого набора переносятся в новый. Система эффективна, если нет многократных вложений.

В системе UltraSPARC III также есть 32 разряда с плавающей точкой, которые могут содержать либо 32-битные (одинарная точность), либо 64-битные (двойная точность) значения. Предусмотрена возможность использования пары регистров для поддержания 128-битных значений.

Архитектура UltraSPARC III – это архитектура загрузки/хранения. Это значит, что единственные операции, которые обращаются в память – это операции записи и чтения, служащие для передачи данных между регистрами и памятью. Все операнды для команд арифметических действий должны браться из регистров или предоставляться самой командой, а все результаты должны сохраняться в регистрах.

### 7.4. Общий обзор микросхемы 8051

Микросхема 8051 умеет работать только в одном режиме, никаких аппаратных средств защиты не предусмотрено. На микросхеме никогда не выполняется больше одной программы. Модель памяти поддерживает два адресных пространства по 64 Кбайт для программ и данных. Эти пространства разделены, что позволяет реализовать пространства программ в ПЗУ, а пространство данных – в ОЗУ.

Допускается несколько вариантов реализации памяти. В простейшем варианте предусматривается 4 Кбайт ПЗУ для программ и 128 байт ОЗУ для данных. Вся память размещается в одном корпусе с микросхемой.

В модели 8051 емкость обоих модулей увеличена вдвое: 8 Кбайт ПЗУ и 256 байт ОЗУ. В случае использования этой модели программа записывается в ПЗУ на этапе производства и пользователю недоступна.

Систему 8052 можно оснастить внешним ПЗУ для размещения программ емкостью 64 Кбайт и внешним ОЗУ такой же емкости. Кроме того, допускается установка единого для программ и данных внешнего модуля ОЗУ такой же емкости.

Поддерживается также промежуточная модель, в которой 4 Кбайт ПЗУ и 128 байт ОЗУ размещаются в одном корпусе с микросхемой, а остальная память – вне микросхемы.

Рассмотрим механизм работы с регистрами. Большинство программ 8051 составляются с расчетом на 8 8-ми разрядными регистрами. Существует 4 набора регистров, хотя в каждый конкретный момент времени активен только один из них. Активный регистр определяется значением поля в слове состояния программы (регистр PSW). Наличие нескольких наборов позволяет ускорить обработку прерываний, что существенно для систем реального времени.

Вторая особенность регистров заключается в том, что все они представлены в адресном пространстве памяти. Байты 0-7 соответствуют первому набору регистров, 8-15 – второму и т.д. до 31 байта (рис 7.11).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Память для программ** | 4095 |  | | | | | | | | | | |
|  |
|  |
|  | C | A | | ХХ | RS | | | O | XX | P | **PSW** |
| 0 |  | | | | | | | | | | |
|  |  | IE | XX | | E2 | ES | | E1 | X1 | E0 | X0 | **IE** |
| **Сверхоперативная память** | 127 |  | | | | | | | | | | |
| XX | | XX | E2 | ES | | E1 | X1 | E0 | X0 | **IP** |
| **Память с побитовой адресацией** | 48 |  | | | | | | | | | | |
| O1 | | R1 | O0 | R0 | | E1 | T1 | E0 | T0 | **TCON** |
| **4 банка регистров** | 32 |  | | | | | | | | | | |
| 0 | Таймер 1 | | | | | Таймер 2 | | | | | **TMOD** |

Рис. 7.11.

Непосредственно над 4 банками регистров, по адресам 32-47 расположены 16 байт памяти с побитовой адресацией (0-127) В 8051 набор команд установки, сброса, логических операций для работы с этими битами. Специальные регистры:

* PSW : бит переноса, бит служебного переноса, биты, определяющие набор регистров, бит переполнения, бит четности.
* IE : позволяет включать/отключать прерывания вместе или по отдельности. Если бит IE =0, все прерывания отключены. Если IE=1, то модно включать/отключать прерывания, для которых установлены индивидуальные биты. Биты Е0, Е1, Е2 – для таймеров, ES – последовательные каналы, Х0, Х1 – прерывания от внешних источников.
* IP : приоритет прерываний низкий или высокий. 1 – высокий приоритет.
* TCON: регулирует главные таймеры. Биты О1 и О2 устанавливаются аппаратно в случае переполнения таймеров. Остальные – управление.
* TMOD: определяет разрядность таймера (8, 13, 16 бит), переключает из режима реального времени в счетчик тактов и т.п.

Все упомянутые регистры, а также другие (сумматор, порты ввода/вывода) расположены в диапазоне памяти 128-255. Обращение к ним осуществляется также, как к регистрам R0-R7. Тогда, в процессорах с увеличенным объемом памяти возможно неоднозначное толкование адресов. В таких моделях для обращения к регистрам используется прямая адресация, а для обращения к ОЗУ – косвенная.

### 7.5. Типы данных

С точки зрения архитектуры ключевым вопросом является вопрос о том, осуществляется ли аппаратная поддержка конкретного типа данных. Под аппаратной поддержкой подразумевается, что одна или несколько команд ожидают данные в определенном формате и пользователь не может брать другой формат.

#### 7.5.1. Типы данных процессора Pentium 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Тип** | **1 бит** | **8 бит.** | **16 бит.** | **32 бита** | **64 бита** | **128 бита** |
| Бит |  |  |  |  |  |  |
| Целые числа со знаком |  | \* | \* | \* |  |  |
| Целые числа без знака |  | \* | \* | \* |  |  |
| Двоично-десятичные целые числа |  | \* |  |  |  |  |
| Числа с плавающей точкой |  |  |  | \* | \* |  |

Pentium 4 также может манипулировать 8-ми разрядными символами ASCII: существуют специальные команды для копирования и поиска цепочки символов. Эти команды используются и для цепочек, длина которых известна заранее, и для цепочек, в конце которых стоит специальный маркер. Эти операции часто используются в библиотеках операций над строками.

Операнды не обязательно должны быть выровнены в памяти, но если адреса слов кратны 4 (выравнивание по словам), то производительность повышается.

**7.5.1.1. Типы данных FPU**

Устройство FPU определяет численные данные в 7-ми внешних форматах, образуя при этом три класса:

* двоичные числа – целые числа (16, 32, 64 разряда);
* упакованные десятичные целые числа;
* двоичные вещественные числа – 32, 64, 80 разрядов.

*Целые числа* представляются в дополнительном коде и могут занимать 16 бит (целое слово), 32 бита (короткое целое), 64 бита (длинное целое). Эти форматы существуют только в памяти, внутри FPU они автоматически преобразуются в 86-разрядный расширяемый вещественный формат.

*Упакованные десятичные целые.* Они представляются в прямом коде и упакованном формате, т.е. каждый байт содержит две десятичные цифры. Старший бит старшего разряда байта отведен для знака числа, остальные биты этого байта игнорируются. Длина этого формата –10 байт (80 бит). Опять же этот формат существует только в памяти.

*Вещественные числа.* Имеется три формата с плавающей точкой (одинарной точности ОТ, двойной точности ДТ, расширенной точности РТ). Значащие числа находятся в поле мантиссы, поле порядка показывает фактическое положение десятичной точки в разрядах мантиссы, а бит знака определяет знак числа (рис.4.11).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 7 0 | 23 | | 0 | |  |  | |  |  |  |
| 32 р. | ОТ | S | E | M | | | |  | M=24 | E=8 | E=+127; -126 | Cм=127 |
|  |  |  | 10 | 0 | 52 | | | 0 |  |  |  |  |
| 64 р. | ДТ | S | E | | M | | | | M=53 | E=11 | E=+1023;-1022 | Cм=1023 |
|  |  |  | 14 | | 0 | 63 |  | | 0 |  |  |  |  |
| 80 р. | РТ | S | E | | | 1 | M | | | М=64 | Е=15 | Е=+16383;-16382 | См=16383 |

Рис. 4.11

Порядок E задается в смещенной форме, он равен истинному порядку, увеличенному на значение смещения.

E = истинный порядок + смещение.

Значение числа с плавающей точкой равно:

(-1)S•2E-смещение•(F0).(F1)(F2) …(Fn),

где n = 23, 52, 63.

Устройство FPU обычно поддерживает представление мантиссы в нормализованной форме, т.е. старший бит равен 1. Следовательно, за исключением числа нуль мантисса состоит из целой части и дроби в следующем виде: 1.(F1)(F2)(F3) … (Fn), где Fi = 0 или 1.

В форматах ОТ и ДТ бит F0 при передаче чисел и хранении их в памяти не фигурирует. Это т.н. скрытый и неявный бит, который содержит 1. Следовательно, в этих форматах невозможно представить ненормализованные числа (за некоторым исключением). Кроме того, скрытый бит не позволяет представить в этих форматах нуль, и он должен кодироваться как специальное значение.

Числа в формате РТ имеют явный бит F0. Напомним, что внутри FPU числа хранятся только в этом формате. Перевод в другие форматы из них осуществляется автоматически.

*В качестве примера рассмотрим представление числа –247.375 в вещественных форматах FPU.*

*-247.375 =>-11110111.011*

*Определяем истинный порядок. Он равен –7.*

*Смещенный порядок:*

* *для обычной точности: 7+127=134 🡺 1000 0110*
* *для двойной точности: 7+1023=1030 🡺100 0000 0110*
* *для расширенной точности: 7+16383=16390 🡺 100 0000 0000 0110.*

*Значения чисел:*

* *ОТ: 1 1000 0110 1110 1110 0110 00 ….00*
* *ДТ: 1 100 0000 0110 1110 1110 0110 00 ….00*
* *РТ:1 110 0000 0000 0110 1111 0111 0011 00…00.*

*Специальные значения.* Форматы чисел допускают представление специальных объектов и значений. Для их представления зарезервированы значения смещенного порядка 000 … 0 и 111 … 1.

*Денормализованные вещественные числа* – это число, которое меньше минимального нормализированного числа для каждого вещественного формата. Такие числа имеют минимальный смещенный порядок 000 … 0 и ненулевую мантиссу, бит (F0) явный или неявный считается нулевым. Введены потому, что иногда промежуточные результаты вычислений могут принимать очень малые значения.

*Истинный нуль.* Нуль кодируется нулевым смещенным порядком и мантиссой. Нуль является знаковым (положительные или отрицательные нули).

*Бесконечность.* Поддерживается знаковое представление бесконечности. Это значение кодируется со смещенным порядком из всех единиц и мантиссой 1.000… . Знаки бесконечностей учитываются и сравнение возможно. Бесконечности интерпретируются так:

.

*Нечисла.* Обозначаются NaN (Not-a-Number). Оно имеет смещенный порядок из всех единиц, любой знак и любую мантиссу, за исключением 1.000… 0. В свою очередь могут делиться на типы. Возникают иногда как результат недействительных операций. Могут использоваться для отладки программ.

*Неподдерживаемые форматы.* Формат РТ имеет ряд двоичных наборов, не попадающих в рассмотренные классы. Они представляют устаревшие форматы, использовавшиеся в сопроцессоре 8087.

#### 7.5.2. Типы данных машины UltraSPARC III

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Тип** | **1 бит** | **8 бит.** | **16 бит.** | **32 бита** | **64 бита** | **128 бита** |
| Бит |  |  |  |  |  |  |
| Целые числа со знаком |  | \* | \* | \* | \* |  |
| Целые числа без знака |  | \* | \* | \* | \* |  |
| Двоично-десятичные целые числа |  |  |  |  |  |  |
| Числа с плавающей точкой |  |  |  | \* | \* | \* |

Все операнды должны быть выровнены в памяти.

UltraSPARC представляет собой регистровую структуру, и почти все команды оперируют 64-разрядными регистрами. Символьные и строковые типы данных специальными командами аппаратного обеспечения не поддерживаются.

#### 7.5.3. Типы данных 8051

Количество типов данных строго ограничено. Поддерживается тип данных, не используемый в арифметических операциях – бит. Для работы с битами имеются специальные операции.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Тип** | **1 бит** | **8 бит.** | **16 бит.** | **32 бита** | **64 бита** | **128 бита** |
| Бит | \* |  |  |  |  |  |
| Целые числа со знаком |  | \* |  |  |  |  |
| Целые числа без знака |  |  |  |  |  |  |
| Двоично-десятичные целые числа |  |  |  |  |  |  |
| Числа с плавающей точкой |  |  |  |  |  |  |

### 7.6. Форматы команд

Команда – представляет собой код, определяющий операцию вычислительной машины и данные, участвующие в операции. Команда содержит также в явной или неявной форме информацию об адресе, по которому помещается результат операции и адрес следующей команды.

Код команды можно представить состоящим из нескольких частей или полей. Команда в общем случае состоит из операционной и адресной частей.

код операции

операционная адресная

часть часть

Рис. 4.12

Эти части, в свою очередь, могут состоять из нескольких полей.

Операционная часть содержит код операции (КОП), который задает вид выполняемой операции. Адресная часть команды содержит информацию об адресах операндов и результата операции, а в некоторых случаях информацию об адресе следующей команды.

Структура команды определяется составом, назначением и расположением полей в команде. Форматом команды называют ее структуру с разметкой номеров разрядов (бит), определяющих границы отдельных полей команды, или с указанием числа бит в определенных полях.

Рассмотрим некоторые классические структуры команд.

Чтобы команда содержала в явном виде всю необходимую информацию о задаваемой операции, она должна содержать код операции и четыре поля адреса для указания ячеек памяти, содержащих два операнда, ячейки, в которую помещается результат операции и адрес следующей команды. Такой порядок выполнения называется принудительным, а приведенный формат команды является труднореализуемым и неэффективным и в настоящее время не применяется.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| КОП | А1 | А2 | А3 | А4 |

Рис. 7.13

Очевидно, что в большинстве машин и большинстве случаев за командой, хранящейся по адресу *K* и занимающей *L* ячеек, будет выполняться команда, с адресом *K+L* . Такой порядок выработки команд называется естественным, и он нарушается только специальными командами. Тогда учитывать адрес следующей команды в явном виде нет необходимости и можно использовать трехадресную команду (рис. 4.14).

Операция, описываемая трехадресной командой, символически представляется в виде:

ОП[A3] := ОП[A1] \*ОП[A2].

Можно условиться, что результат операции всегда помещается на место одного из операндов. Тогда получаем двухадресную команду, задающую операцию (рис. 7.14):

ОП[A1] := ОП[A1] \*ОП[A2].

В одноадресной команде подразумеваемые адреса имеют уже и результат операции, и один из операндов (рис. 4.15.).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| КОП | А1 | А2 | А3 |  | КОП | А1 | А2 |  | КОП | А1 |

Рис. 7.13. Рис. 7.14. Рис. 7.15.

Подразумеваемым операндом является в данном случае внутренний регистр процессора, называемый регистром результата или аккумулятором. Результат такой операции также записывается в аккумулятор:

Акк := Акк \* ОП[А1].

Наконец, в некоторых случаях возможно использование безадресных команд, когда подразумеваются адреса обоих операндов и результата. Например, при использовании стековой памяти.

С точки зрения программиста наиболее естественны и удобны трехадресные команды. Однако, такие команды длины. Кроме того, часто в качестве операндов используются результаты предыдущих команд. В этом случае выполняемая операция принимает уже вид двух– или одноадресной команды и трехадресный формат использовать неэффективно.

Обычно в ЭВМ используется несколько форматов различной длины. Рассмотренные структуры команд достаточно схематичны. В действительности адресные поля команд большей частью содержат не сами адреса, а только информацию об этих адресах. Способы адресации рассматриваются в следующем разделе.

#### 7.6.1. Критерии разработки для форматов команд

При разработке формата команд необходимо учитывать ряд факторов. Эффективность конкретной архитектуры команд зависит от технологии, которая применялась при разработке компьютера. Например, если доступ к памяти осуществляется быстро, то можно использовать стековую архитектуру, а если доступ к памяти медленный, то желательно иметь много регистров.

Если речь идет об одинаковых машинах, то лучше иметь короткие команды, чем длинные. Такие программы занимают в памяти меньше места. Но минимизация размера команд может усложнить их декодирование. Следовательно, достижение минимальной длины команды должно уравновешиваться временем, затрачиваемым на декодирование и выполнение команд.

Следующая причина минимизации длины команд связана с увеличением скорости работы процессора. Значительный рост скорости работы процессора не соответствует относительно низкой пропускной способности памяти. Трудности, связанные с пропускной способностью памяти имеют отношение не только к основной памяти, но и ко всем видам кэш-памяти.

Если пропускная способность кэш-памяти составляет t бит/с, а средняя длина команды r битов, то кэш-память способна передать самое большое (верхний предел) t/r команд/с. Очевидно, что скорость, с которой могут выполняться команды (скорость работы процессора), может ограничиваться длиной команд, чем короче команда, тем быстрее. А поскольку современные компьютеры выполняют несколько команд за цикл, то вызов нескольких команд за цикл становится обязательным.

Второй аспект разработки – достаточно большой объем пространства в формате команд для выражения всех требуемых команд.

 ( *M* – число команд).

История доказывает, что обязательно нужно оставлять большее количество свободных кодов операций для будущих дополнений к наборам команд.

Третий критерий связан с числом битов в адресном поле. Для адресации *S* ячеек памяти адресная часть одного операнда должна иметь число разрядов



Для адресации памяти требуется более длинные адреса. Одна крайность – это организация памяти, при которой адресуется каждый бит (Burroughs B1700). Другая крайность – это память, состоящая из очень длинных слов (например, серия CDC Cyber содержала 60-битные слова).

Кроме того, для упрощения аппаратуры и увеличения производительности ЭВМ желательно, чтобы длина формата команды была согласована с длиной машинных слов.

Современные компьютерные системы пришли к компромиссу, который в некотором смысле объединяет в себе худшие качества обоих вариантов. Он требует, чтобы адреса были у каждого байта, но при обращении к памяти всегда считывается одно, два, а иногда и четыре слова сразу. В результате считывания одного байта в машине UltraSPARC II сразу вызывается минимум 16 байтов, а иногда и строка кэш-памяти в 64 байта.

#### 7.6.2. Расширение кода операций

Наряду с обычными компромиссами между числом разрядов, отводимых под код операции и адресную часть при фиксированной длине команд, рассмотрим прием, который называется *расширением кода операций*. Суть приема заключается в том, чтобы увеличивать разрядность кода за счет изменения количества адресов в команде. Рассмотрим 16-битную команду:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 16 бит | | | |  |
| *4-битный код операции* | 0000  0001  ….  1110 | Xxxx  Xxxx  ….  xxxx | Yyyy  Yyyy  ….  yyyy | Zzzz  Zzzz  ….  Zzzz | *15 трехадресных команд* |
| *8-битный код операции* | 1111  1111  ….  1111 | 0000  0001  ….  1101 | Yyyy  Yyyy  ….  yyyy | Zzzz  Zzzz  ….  zzzz | *14 трехадресных команд* |
| *12-битный код операции* | 1111  1111  ….  1111 | 1110  1110  ….  1111 | 0000  0001  ….  1110 | Zzzz  Zzzz  ….  zzzz | *31 трехадресная команда* |
| *16-битный код операции* | 1111  1111  ….  1111 | 1111  1111  ….  1111 | 1111  1111  ….  1111 | 0000  0001  ….  1111 | *15 трехадресных команд* |

Другой вариант расширения количества команд – использование команд различной длины. Предлагается делать часто используемые команды короткими, а редко используемые – длинными. Однако в этом случае возникает проблема с декодированием и выравниванием (например, Intel).

#### 7.6.3. Форматы команд процессора Pentium 4

Общий формат команды процессора приведен на рис. 7.16.

Команда содержит следующие поля:

* префиксы команды (необязательны),которые могут следовать в произвольном порядке;
* одного или двух байт (главного) кода операции;
* спецификатора адреса, представленного битами *mod r/m* и *sib*;
* смещения в команде (*displacement*) – если необходимо;
* поле непосредственного значения (если такое значение присутствует).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| mod | | reg/коп | | | r/m | | |  | ss | | index | | | base | | |

*Префиксы*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| повтор | размер адреса | размер операнда | замена сегмента |  | код операции | mod r/m | sib | смещение (в команде) | непосред. операнд |
| 0 или 1 | 0 или 1 | 0 или 1 | 0 или 1 | 1 или 2 | 0 или 1 | 0 или 1 | 0,1,2 или 4 | 0,1,2 или 4 |
| число байт | | | | число байт | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Байт 0 –код операции | | | | | | | |
| код операции | | | | | | d | w |
|  |  |  |  |  |  |  |  |

Рис. 7.16

(Base – база, MODe – режим, Register/Memory – регистр/память, Scale – масштаб, Index – индекс).

Из всех полей команды обязательными являются только один или два байта кода операции.

*Префикс* – это байт со специальным кодированием, который модифицирует операцию одной находящейся за ним команды. Префиксы сгруппированы следующим образом:

* префиксы команды (*REP\REPE\REPZ\REPNE*, применяется в строковых командах, для автоматической обработки всех элементов цепочки и блокировки шины (LOCK) – допускается перед некоторыми командами с обращением к памяти для запрета другим процессорам в мультипроцессорных системах);
* замены сегмента (*Segment override*), явно определяет сегментный регистр для конкретной команды вместо сегментного регистра, принимаемого по умолчанию;
* размера операнда (*OperandSize*), переключает 16– и 32–битные операнды;
* размера адреса (*AddressSize*), коммутирует формирование 16– и 32–битных адресов;

По сравнению с 16-ти разрядными процессорами х86, начиная с I80386 система команд расширена в двух направлениях: во-первых, все команды, рассчитанные на 16-разрядные операнды работают и с 32–разрядными; во-вторых, доступны 32-разрядные режимы адресации. Такое расширение реализовано введением бита D (*Default*) в дескриптор сегмента кода и двух префиксов. Когда бит D установлен в 0, команды текущей программы работают с 16-разрядными данными и адресами, если D установлен в 1, то они работают с 32-разрядными данными и адресами.

Префиксы размера и адреса операнда заставляют процессор изменять для команды размеры операнда или адреса.

В большинстве команд вслед за байтом кода операции, который указывает местонахождение операнда в памяти, следует второй байт, который сообщает всю информацию об операнде. Эти 8 бит распределены в 2-битном поле mod и в двух 3-битных регистровых полях reg и r/m.

Рассмотрим поле *reg*. Это поле определяет один из регистров общего назначения. Кодирование этого поля зависит от того, имеется ли в КОП поле W (определяющее размер операнда), а при его наличии от типа операнда. Это поле может находится в байте КОП и в байте mod r\m.

Режим адресации определяют один или два байта адресации, которые находятся после байта адресации. Первым является байт *mod r/m*, вторым байт *sib* (масштаб, индекс, база). Байт *sib* может присутствовать только в командах с 32-разрядной адресацией, когда байт *mod r/m* содержит *r/m* = 100 и значение в поле *mod*≠11.

Таблица 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| reg | поля W нет | | Поле W есть | | | |
| 16 р | 32 р | 16 разрядов | | 32 разряда | |
| W=0 | W=1 | W=0 | W=1 |
| 000 | AX | EAX | AL | AX | AL | EAX |
| 001 | CX | ECX | CL | CX | CL | ECX |
| 010 | DX | EDX | DL | DX | DL | EDX |
| 011 | BX | EBX | BL | BX | BL | EBX |
| 100 | SP | ESP | AH | SP | AH | ESP |
| 101 | BP | EBP | CH | BP | CH | EBP |
| 110 | SI | ESI | DH | SI | DH | ESI |
| 111 | DI | EDI | BH | DI | BH | EDI |

#### 7.6.4. Форматы команд FPU

*Особенности задания команд.* Команды бинарных операций допускают несколько форм. В случае пустого поля операнда (безадресная команда) операция выполняется с двумя верхними элементами стека ST(0) и ST(1). Поcле выполнения операции осуществляется инкремент указателя стека и результат помещается в новую вершину стека, заменяя исходное содержание ST(1). Это классическая операция для машин со стековой адресацией.

Когда в бинарной операции определен один операнд, она выполняется с привлечением указанного в команде регистра (ячейки памяти) и содержимого вершины стека. Результат загружается в старую вершину стека и указатель стека не изменяется.

Если же в бинарной команде указаны два операнда, ими является содержимое двух регистров стека, причем одним из них будет ST(0), а вторым ST(i). Возможны три случая:

* источникам является ST(0), а получателем ST(i);
* источник — ST(i), получатель — ST(0);
* источникам является ST(0), получателем ST(i) и производится инкремент указателя стека, т.е. “источник” пропадает.

Отметим, что в несимметричных операциях деления и вычитания обычно получатель делится на источник (из получателя вычитается источник). FPU допускает и обратные (Reverse) формы таких команд.

Многие команды допускают несколько форм задания операндов, которые при этом принято записывать через слэш «/», например

FADD //src/dst,src .

Эта запись подразумевает три возможные формы команды: без операндов, с одним операндом, с двумя операндами.

src – Source — источник;

dst – Destionation – получатель.

В мнемониках команд FPU приняты следующие соглашения:

* первая буква команды всегда F (только команды FPU начинаются с F);
* вторая буква I обозначает операцию с целым двоичным числом из памяти, буква B – операцию с десятичным числом из памяти, без этих букв – вещественное число;
* предпоследняя или последняя буква R указывает обратную операцию (для вычитания или деления);
* последняя буква P идентифицирует команду, заключительным действием которой является извлечение из стека.

Рассмотрим машинный формат команд.

Для команд FPU существует пять различных форматов кодирования (рис. 4.17), однако в любом формате старшие пять битов двухбайтного кода команды FPU всегда содержит специальное значение 11011, по которому процессор распознает команды FPU.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Команда | | | | | | | | | | | | | | | | | Доп. поле | |
| Первый байт | | | | | | | | | Второй байт | | | | | | | |  | |
| 15 | 14 | 13 | 12 | 11 | 10 | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |  |
| 1 | 1 | 0 | 1 | 1 | OP1 | | | 1 | Mod | | 1 | OP2 | | R/m | | | sib | Disp |
| 1 | 1 | 0 | 1 | 1 | MF | | | OP1 | Mod | | OP2 | | | R/m | | | sib | Disp |
| 1 | 1 | 0 | 1 | 1 | d | p | | OP1 | 1 | 1 | OP2 | | R | ST(i) | | |  |  |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | | 1 | | 1 | 1 | 1 | OP | | | |  |  |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | | 1 | | 1 | 1 | 1 | OP | | | |  |  |

Рис. 7.17

Поля имеют следующее назначение и кодирование:

* MF – формат ячейки памяти. (00 – 32-битное вещественное, 01- 32-битное целое, 10 – 64-битное вещественное, 11 – 64-битное целое);
* P– извлечение из стека (0 – не извлекать; 1 – извлекать);
* D- назначение (0- ST(0), 1 – ST(1));
* R– направление операции;
* ST(i) – элемент регистрового стека;
* OP1, OP2 – первая и вторая части кода операции.
* Поля mod, r/m, sib кодируются также, как для базовых команд процессора.

В АССЕМБЛЕРЕ резервирование памяти осуществляется с помощью директив: DD – двойное слово; DQ – учетверенное слово; DT – определить 10 байт.

#### 7.6.5. Форматы команд процессора UltraSPARC III

Команды 32-битные, выровненные в памяти. Каждая команда определяет только одно действие. Типичная команда указывает два регистра, из которых берутся операнды и выходной регистр. Вместо одного регистра допускается использование константы со знаком. При выполнении команды считывания два регистра (или один регистр и одна константа) складываются вместе и определяют адрес памяти, по которому производится считывание; результат записывается в указанный регистр.

Изначально SPARC имела ограниченное число форматов (рис. 7.18). Со временем добавлялись дополнительные форматы и сейчас их 31.

Первые два бита команды помогают определить формат команды и сообщают программному обеспечению, где найти оставшуюся часть кода операции, если она есть.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 5 | | 6 | | 5 | 1 | 8 | 5 |  | |
| 1a |  | Вых. Рг. | | КОП | | Вх. Рг1 | 0  х | Операция с п/з | Вх. Рг.2 | *3 регистра* | |
| 1b |  | Вых. Рг. | | КОП | | Вх. Рг1 | 1 | Непосредственная константа | | Непосредственный операнд | |
|  | | | | | | | | | | | |
|  | 2 | 5 | | 3 | 22 | | | | | |  |
| 2 |  | Вых. Рг. | | КОП | Непосредственная константа | | | | | | SETHI |
|  | | | | | | | | | | | |
|  | 2 | 1 | 4 | 3 | 22 | | | | |  | |
| 3 |  | A | условие | КОП | Смещение относительно счетчика команд | | | | | *BRANCH (команда перехода)* | |
|  | | | | | | | | | | | |
|  | 2 | 30 | | | | | | | |  | |
| 4 |  | Смещение относительно счетчика команд | | | | | | | | *CALL (команда вызова процедур)* | |

Рис. 7.18

В формате 1а оба источника расположены в регистрах. В формате 1b один операнд – константа. Бит 13 (0 или 1) определяет один из этих форматов.

Поскольку все команды 32-битные, то включить в команду 32-битную константу невозможно. Команда SETHI устанавливает 22 бита, оставляя пространство для другой команды, чтобы установить оставшиеся 10 битов. Это единственная команда, использующая данный формат.

Для непрогнозируемых условных переходов используется формат 3, в котором поле условие определяет, какое условие надо проверить. Бит А нужен для того, чтобы избегать пустых операций при определенных условиях. Прогнозируемые переходы используют тот же формат, но только с 19-битным смещением.

Последний формат используется для вызова процедур. Требуемый адрес – это целевой адрес, разделенный на четыре.

#### 7.6.6. Форматы команд 8051

В 8051 предусмотрены шесть форматов команд (рис. 7.19). Размер команды может быть 1,2 или 3 байта.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 8 | | 8 | 8 | |  |
| 1 | КОП | |  | | | |
|  |  | | | | | |
| 2 | КОП | Рг |  | | | |
|  | | | | | | |
| 3 | КОП | | операнд |  | | |
|  | | | | | | |
| 4 | КОП | | 11-разрядный адрес | | |  |
|  | | | | | | |
| 5 | КОП | | 16-разрядный адрес | | |  |
|  | | | | | | |
| 6 | КОП | | Операнд1 | | Операнд2 |  |

Рис. 7.19

Первый вариант предусматривает только наличие кода операции.

Второй вариант также состоит из одного байта: 5 – КОП, 3 – номер регистра.

В третьем формате используется 2 байта: КОП и операнд, размером в 1 байт.

Четвертый и пятый форматы используются для команд перехода. 11-разрядный адрес используется в случае отсутствия внешней памяти. Если присутствует внешняя память и ее объем составляет более 4 Кбайт, используются 16-разрядные адреса.

Шестой формат использует два операнда. Может использоваться для переноса константы по некоторому адресу.

### 7.7. Адресация

Как было сказано ранее, для адресации *S* ячеек памяти адресная часть одного операнда должна иметь число разрядов



Это требование находится в противоречии с желанием иметь малую разрядность команды и возможностью использовать большое адресное пространство. Существуют два основных подхода к решению этой проблемы.

1. Использовать регистры общего назначения. Это существенно повышает быстродействие, но имеет эффект толь в том случае, если операнд используется многократно. Кроме того, существует ограничение на количество регистров общего назначения.
2. Второй подход подразумевает определение одного или нескольких операндов неявным образом. Это означает использование не трехадресной команды, а двух- одно- и безадресных команд.

Следует различать понятия адресный код в команде **Ак** и исполнительный адрес **Аи.** Адресный код –это информация об адресе команды, содержащаяся в команде. Исполнительный адрес – это номер ячейки памяти, к которой производится обращение. В современных ЭВМ адресный код, как правило, не совпадает с исполнительным адресом. В свою очередь исполнительный адрес может не совпадать с физическим из-за таких особенностей ЭВМ как сегментация и виртуальная память.

Рассмотрим способы адресации, используемые в современных ЭВМ.

#### 7.7.1. Способы адресации

##### 7.7.1.1.Подразумеваемый операнд.

В команде не содержится в явном виде указаний об адресе операнда. Операнд подразумевается и фактически задается кодом операции команды. Вообще, данный метод используется нечасто, однако в архитектуре х86 имеется достаточное количество таких команд: INC, DEC.

##### 7.7.1.2.Подразумеваемый адрес.

В команде не содержится явных указаний об адресе участвующих в операции операндов, или адреса, по которому помещается результат операции, но этот адрес подразумевается. Например, команда может содержать адреса обоих операндов, участвующих в операции, а результат помещается по адресу одного из операндов. Сюда также можно отнести операции с использованием подразумеваемого регистра.

##### 7.7.1.2. Непосредственная адресация.

В команде содержится не адрес операнда, а непосредственно операнд. Непосредственная адресация удобна при использовании различного рода констант.

##### 7.7.1.3. Прямая адресация.

Исполнительный адрес совпадает с адресной частью кода команды. Этот способ адресации был общепринятым в первых вычислительных машинах и продолжает применяться в настоящее время в комбинации с другими способами.

##### 7.7.1.4.Регистровая адресация.

В качестве операнда используется содержимое регистров процессора (ячейки сверхоперативной памяти). Например, если таких регистров 16, то для адреса регистра достаточно 4-х битов. Регистровая адресация наряду с сокращением длины адресов операндов позволяет увеличивать скорость выполнения операций.

##### 7.7.1.5. Косвенная адресация.

Адресный код команды указывает адрес ячейки памяти, в которой находится адрес операнда или команды. Т.е. происходит «адресация адреса». Если в качестве адресная часть команды – регистр, то такая адресация называется *регистровой косвенной адресацией*, а используемых регистр – *указателем*.

Косвенная адресация начала широко использоваться в свое время в малых ЭВМ и микропроцессорах, имеющих короткое машинное слово, для преодоления короткого формата команды.

Особенно часто применяется совместное использование регистровой и косвенной адресаций.



Рис. 7.20

На рис. 4.20 описаны следующие поля:

КОП – код операции;

Рг – номер регистра;

УА – указатель косвенной адресации.

На рис. 7.20 показан механизм пересылки числа 4527 из регистра 5 в ячейку памяти 1765. Операнд 4527 указывается в регистровой прямой адресации, а для задания адреса 1765 используется регистровая косвенная адресация.

##### 7.7.1.6. Относительная адресация или базирование.

Исполнительный адрес определяется суммой адресного кода команды Ак и некоторого числа Аб, называемого базовым адресом:

Аи = Аб + Ак.

Для хранения базовых адресов в машине могут быть использованы специальные регистры или ячейки памяти (базовые регистры). В команде выделяется поле B для указания номера базового регистра.

Относительная адреса­ция позволяет при меньшей длине адресного кода команды обеспечить доступ к любой ячейке памяти. Для этого число разрядов в базовом регистре выбирается таким, чтобы можно было адресовать любую ячейку ОП, а адресный код Ак самой команды используют для представления сравнительно короткого «смещения» (обозначим D). Смещение D определяет положение опе­ран­да относительно начала массива, задаваемого базовым адресом Аб. Схема формирования исполнительного адреса приведена на рис. 7.21.

Большей частью исполнительный адрес при базировании образуется с помощью сумматора согласно выражению:



Аи = (В) + D,

где (В) – содержимое регистра с номером В.

Иногда применяют формирование исполнительного адреса методом совмещения. В этом случае базовый адрес содержит старшие, а смещение – младшие разряды исполнительного адреса, которые объединяются в регистре адреса путем их составления.

Однако при сов­ме­ще­­нии базовый адрес может задавать не любые ячейки памяти, а только те, ад­ре­са которых содержат нули в младших разрядах, со­от­вет­ствующих сме­ще­нию.

Относительная адреса­ция обеспечивает так на­зы­ваемую *перемещаемость* программ, т.е. возможность перемещения программ в памяти без изменений внутри самой программы.

##### 7.7.1.7. Индексная адресация.

Характерной особенностью вычислительных процессов, происходящих в ЭВМ, является цикличность, при которой повторяются одни и те же процедуры, но над различными операндами (как правило, элементами массивов, расположенными упорядочено в памяти). Поскольку операнды, обрабатываемые при построении цикла, имеют разные адреса, можно для каждого повторения составить свою последовательность команд, отличающуюся только адресными частями.

Такая программа, состоящая из групп команд, отличающихся только адресной частью, является, очевидно, слишком длинной, а написание ее – слишком трудоемким.

Программирование вычислительных циклов существенно упроститься, если после каждого выполнения цикла обеспечить автоматическое изменение в соответствующих командах их адресных частей согласно расположению в ОП обрабатываемых операндов. Такой процесс называется *модификацией команд*, точнее адресных частей команд. Модификация команд основана на возможности выполнения над кодами команд или их частями арифметических и логических операций. Идея модификации команд была предложена фон Нейманом.

В первых вычислительных машинах действительно использовалась модификация команд. В современных ЭВМ используется механизм *индексации.* Это понятие включает в себя специальный способ кодирования команд, командные и аппаратные средства задания и выполнения модификации команд и управления вычислительными циклами. Упомянутые средства называются *индексной арифметикой*. По своей сути индексация является дальнейшим развитием базирования.

Для выполнения индексации в архитектуру процессора вводятся так называемые *индексные регистры*. В формате команды вводится поле X для указания индексного регистра. Исполнительный адрес при индексации формируется путем сложения адресного кода команды (смещения) с содержимым индексного регистра (индексом). Во многих архитектурах содержимое индексного регистра дополнительно умножается на размер операнда. При наличии базирования добавляется еще и базовый адрес.

##### 7.7.1.8.Стековая адресация.

Стековая память реализует безадресное задание операндов. *Стек* представляет собой группу последовательно пронумерованных регистров *(аппаратный стек)* или ячеек памяти, снабженных указателем стека (обычно регистром SP), в котором автоматически при записи и считывании указывается номер (адрес) последней занятой ячейки стека *(вершины стека)*. При выполнении операции записи в стек слово помещается в следующую ячейку стека, а при считывании из стека последнее поступившее в него слово. Таким образом, в стеке реализуется дисциплина обслуживания «последний пришел – первый ушел» (LIFO).

Указанное правило при обращении к стеку реализуется автоматически, и поэтому при операциях со стеком возможно безадресное задание операндов.

Механизм стековой адресации может быть проиллюстрирован рисунком (рис. 7.22).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SP | 97 |  | 96 | |  | 97 |  | | 98 |
|  |  | |  |  |  | |  |
|  |  |  | |  |  |  | |  |
|  |  | 214 | |  |  |  | |  |
| 97 | 321 | 321 | |  | 535 |  | |  |
| 98 | 175 | 175 | |  | 175 |  | | 175 |
| 99 | 253 | 253 | |  | 253 |  | | 253 |
| 100 | 100 |  | 100 | |  | 100 |  | | 100 |
|  |  |  | Заслать в стек из РГ3 | |  |  |  | загрузить из стека в РГ4 | | |
|  |  | 214 |  | Сложить | |  | 535 | |  |

Рис. 7.22

При выполнении команды передачи в стек слова указатель стека увеличивается на 1, а затем слово помещается в ячейку стека, указываемую SP. При загрузке из стека сначала извлекается слово, а затем указатель уменьшается на 1.

При соответствующем расположении операндов в стеку можно вычислять выражения полностью безадресными командами, указывающими только тип операции. Такая команда извлекает из стека в соответствии с кодом операции один или два операнда, выполняет предписанную операцию и заносит результат в стек.

Вычисления с использованием стековой памяти удобно описывать и программировать с помощью Польской инверсной (бесскобочной) записи арифметических выражений. Эта запись производится по следующему правилу: читаем арифметическое выражение слева на право и последовательно, друг за другом выписываем встречающиеся операнды. Как только окажется, что все операнды некоторой операции выписаны, записываем знак этой операции и продолжаем выписывать операнды. Если операция имеет операндом результат некоторой предыдущей операции и знак последней выписан, то считаем этот операнд выписанным.

Например, выражение



в бесскобочной записи имеет вид:



Выражение в польской инверсной записи не содержит скобок, но порядок действий определен однозначно. Программа вычисления такого выражения может иметь следующий, достаточно условный вид:

*push k*

*push l*

*add*

*push m*

*sub*

*push p*

*push s*

*sub*

*mul*

Безадресные команды на основе стековой адресации предельно сокращают форму команд, экономят память и способствуют повышению производительность ЭВМ.

Однако при такой структуре команд возникают сложности с построением команд передачи управления и работы с периферийными устройствами.

В современной архитектуре процессоров стек и стековая адресация применяется при организации переходов к подпрограммам и системах прерывания. Элементы стековой адресации используются в процессорах х86 при работе с данными с плавающей точкой.

##### 7.7.1.9. Способы адресации команд перехода

Командам перехода (а также командам вызова процедур) также нужны особые способы адресации для определения целевого адреса. Способы, существующие для адресации данных могут быть также использованы для вычисления исполнительных адресов команд перехода. Один из возможных вариантов – прямая адресация, когда целевой адрес просто полностью включается в команду.

Другие способы адресации также имеют смысл. Косвенная и регистровая адресация позволяют программа вычислить целевой адрес, помещать его в регистр, а затем переходить по этому адресу. Такой способ дает максимальную гибкость, поскольку целевой адрес вычисляется во время выполнения программы. Но такой подход предоставляет огромные возможности для появления ошибок, которые практически невозможно найти.

Индексная адресация, при которой известно смещение от регистра, также может быть использована.

Может быть также использована относительная адресация по счетчику команд. В этом случае для получения целевого адреса смещение (со знаком), находящееся в самой команде прибавляется к программному счетчику.

#### 7.7.2. Способы адресации процессора Pentium 4

Способы адресации процессора Pentium II нерегулярны и зависят от того, в каком формате находятся команды: 16- или 32-битном.

Режим адресации определяют один или два байта адресации, которые находятся после байта адресации. Первым является байт *mod r/m*, вторым байт *sib* (масштаб, индекс, база). Байт *sib* может присутствовать только в командах с 32-разрядной адресацией, когда байт *mod r/m* содержит *r/m* = 100 и значение в поле *mod*≠11.

Для адресации теперь можно использовать любой регистр общего назначения, а индекс разрешается масштабировать (умножать) на 1, 2, 4 или 8. Ниже приводятся все способы 32–разрядной адресации.

Таблица 7.3

|  |  |  |  |
| --- | --- | --- | --- |
| Непосредственная | mov | eax, | 12345678h |
| Регистровая | mov | eax, | ecx |
| Прямая (абсолютная) | mov | eax, | [3456789h] |
| Регистровая косвенная | mov | eax, | [ecx] |
| Базовая (индексная) со смещением | mov | eax, | [ecx]+1200h |
| Базовая индексная со смещением | mov | eax, | [ecx][edx]+40h |
| Индексная с масштабированием и смещением | mov | eax, | [eci\*4]+400h |
| Базовая индексация с масштабированием | mov | eax, | [edx][ecx\*8] |
| Базовая индексация с масштабированием и смещением | mov | eax, | [ebx][edi\*2]+20h |

Проблема заключается в том, что не все способы адресации применимы ко всем командам и не все регистры могут использоваться при всех способах адресации. Это существенно усложняет работу компилятора.

Схема формирования адреса в 32-разрядной адресации приведена на рис 7.24



Рис. 4.24.

32–разрядные режимы адресации кодируются с помощью двух байт mod r/m и sib.

Байт *mod* управляет способами адресации. Один из операндов определяется по комбинации полей mod и r/m. Второй операнд всегда является регистром и определяется по значению поля reg. Формирование адреса памяти в 32-битной адресации (r/m ≠100).

Таблица 7.4

M[x] – слово в памяти с адресом х.

D -- смещение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r/m | Адрес памяти второго операнда | | | |
| mod = 00 | mod = 01 | mod = 10 | Mod = 11 |
| 000 | М[EAX] | M[EAX+d8] | M[EAX+d32] | EAX или AL |
| 001 | M[ECX] | M[ECX+d8] | M[ECX+d32] | ECX или CL |
| 010 | M[EDX] | M[EDX+d8] | M[EDX+d32] | EDX или DL |
| 011 | M[EBX] | M[EBX+d8] | M[EBX+d32] | EBX или BL |
| 100 | имеется sib | имеется sib | имеется sib | ESP или AH |
| 101 | Прямая адр. | M[EBP+d8] | M[EBP+d32] | EBP или CH |
| 110 | M[ESI] | M[ESI+d8] | M[ESI+d32] | ESI или DH |
| 111 | M[EDI] | M[EDI+d8] | M[EDI+d32] | EDI или BH |

Колонки 01 и 10 включают способы адресации, при которых значение регистра прибавляется к 8-битному или 32-битному смещению, которое следует за командой.

Иногда вслед за байтом mod следует дополнительный байт sib. Байт sib определяет масштабный коэффициент и два регистра.

Поле SS указывает масштабный коэффициент индекса, поле index определяет любой регистр, кроме ESP, который служит индексным регистром, а поле base – определяет базовый регистр. Формирование адреса памяти в 32-битной адресации (r/m =100, имеется sib) приведено в табл. 4.5.

Таблица 7.5

|  |  |  |  |
| --- | --- | --- | --- |
| поле base | Адрес памяти второго операнда | | |
| mod = 00 | mod = 01 | mod = 10 |
| 000 | M[EAX+ss\*ind] | M[EAX+ss\*ind+d8] | M[EAX+ss\*ind+d32] |
| 001 | M[ECX+ss\*ind] | M[ECX+ss\*ind+d8] | M[ECX+ss\*ind+d32] |
| 010 | M[EDX+ss\*ind] | M[EDX+ss\*ind+d8] | M[EDX+ss\*ind+d32] |
| 011 | M[EBX+ss\*ind] | M[EBX+ss\*ind+d8] | M[EBX+ss\*ind+d32] |
| 100 | M[ESP+ss\*ind] | M[ESP+ss\*ind+d8] | M[ESP+ss\*ind+d32] |
| 101 | M[d32+ss\*ind] | M[D32+ss\*ind+d8] | M[d32+ss\*ind+d32] |
| 110 | M[ESI+ss\*ind] | M[ESI+ss\*ind+d8] | M[ESI+ss\*ind+d32] |
| 111 | M[EDI+ss\*ind] | M[EDI+ss\*ind+d8] | M[EDI+ss\*ind+d32] |

Таблица кодирования полей index и ss.

|  |  |  |  |
| --- | --- | --- | --- |
| index | индексный регистр | ss | Множитель |
| 000 | EAX | 00 | \*1 |
| 001 | ECX | 01 | \*2 |
| 010 | EDX | 10 | \*4 |
| 011 | EBX | 11 | \*8 |
| 100 | нет |  |  |
| 101 | EBP |  |  |
| 110 | ESI |  |  |
| 111 | EDI |  |  |

#### 7.7.3. Способы адресации процессора UltraSPARC III

В архитектуре команд процессора UltraSPARC III все команды используют непосредственную и регистровую адресацию за исключением тех команд, которые непосредственно обращаются к памяти. При регистровом способе адресации 5 битов просто сообщают, какой регистр нужно использовать. При непосредственной адресации данные обеспечивает 13-битная константа со знаком. Для арифметических, логических и подобных команд никакие другие способы адресации не используются.

К памяти обращаются команды трех типов: считывания (Load), записи (Store) и одна команда синхронизации мультипроцессора. Для команд записи и считывания существует два способа обращения к памяти.

Первый. Вычисляется сумма двух регистров, а затем через полученное значение производится косвенная адресация.

Второй способ представляет собой обычное индексирование с 13-битным смещением со знаком.

#### 7.7.4. Способы адресации процессора 8051

Предусмотрено пять режимов адресации.

Первый – неявная адресация. Первый операнд находится в сумматоре, второй в памяти или регистрах. Использование сумматора определяется по коду операции.

Второй способ – регистровая адресация. Регистры могут быть как входными, так и выходными.

Третий режим – прямая адресация, адрес операнд указан в команде.

Четвертый режим – косвенная регистровая адресация, подразумевающая размещение в регистре указателя на операнд. Поскольку разрядность обычных регистров составляет 8 бит, то операнды с такой адресацией должны занимать нижние 256 байт памяти.

В пятом режиме непосредственной адресации операнд является частью команды.

Предусмотрен специализированный режим адресации, применяемый к некоторым командам для доступа в память. К микросхеме можно подключить до 64 Кбайт внешней памяти для программ и столько же для данных. За взаимодействие с внешней памятью программ отвечают команды LJMP и LCALL. Для взаимодействия с внешней памятью данных используется 16-разрядный регистр DPTR, в котором размещаются 16-разрядные адреса памяти. Таким образом используется двойной указатель.

#### 7.7.4. Сравнение способов адресации

|  |  |  |  |
| --- | --- | --- | --- |
| **Способ адресации** | **Pentium II** | **UltraSPARC II** | **JVM** |
| Неявная адресация |  |  | \* |
| Непосредственная | \* | \* | \* |
| Прямая | \* |  | \* |
| Регистровая | \* | \* | \* |
| Косвенная регистровая | \* | \* | \* |
| Индексная | \* | \* |  |
| Относительная индексная |  | \* |  |
| Стековая |  |  |  |

На практике для эффективной архитектуры команд совсем не обязательно использовать большое число способов адресации. Поскольку практически весь код, написанный на этом уровне порождается компилятором, то способов адресации должно быть мало, они д.б. четкими и ясными.

Поэтому самые простые архитектуры используют небольшое число способов адресации, а на каждый используемый способ накладываются жесткие ограничения. Обычно вполне достаточно непосредственной, прямой, регистровой и косвенной адресации.

При изучении нового компьютера нужно изучать все команды и способы адресации не только для того, что бы знать, какие из них имеются в наличии, но и для того, чтобы понять, почему был сделан такой выбор и как это можно использовать.

### 7.8. Типы команд

Команды можно грубо поделить на несколько групп, которые могут повторяться от машины к машине, хотя и различаются в деталях. Кроме того, в каждом компьютере имеется несколько необычных команд, которые появились или из соображений совместимости, или по недоразумению или по какой-нибудь другой причине.

#### 7.8.1. Команды перемещения данных

Как правило, в машинах с фиксированной длинной слова единицей перемещаемых данных является слово. Однако, существуют архитектуры команд дают возможность копировать отрезки данных меньше слова, а также группу слов. Некоторые машины с изменяемой длиной слова содержат команды, которые определяют только адреса источника и получателя, а не количество данных. Копирование продолжается до тех пор, пока не появится специальное поле в конце данных.

#### 7.8.2. Бинарные операции

Бинарные операции берут два операнда и получают результат. Все архитектуры команд содержат операции с фиксированной точкой. Большинство компьютеров сегодня поддерживают операции с плавающей точкой. Большинство машин содержит по крайней мере 2 варианта таких чисел: более короткие для скорости и более длинные для получения высокой точности вычислений.

#### 7.8.3. Унарные операции

Унарные операции используют один операнд и производят один результат. Команды таких операций могут быть короче.

#### 7.8.4. Команда вызова процедур

Особенность выполнения команды вызова процедур заключается в том, что программа должна вернуться к соответствующему оператору (как правило, следующему за процедурой). Следовательно, адрес возврата должен либо передаваться процедуре, либо сохраняться таким образом, чтобы была возможность определить его местонахождение по окончанию процедуры.

Адрес возврата может быть размещен в одном из трех мест: в памяти, в регистре или в стеке.

Самое худшее решение – размещение в фиксированной ячейке памяти. В таком случае, если процедура вызывает другую процедуру, то адрес теряется.

Более удачное решение – сохранять адрес возврата в первом слове процедуры. Недостаток такой схемы – процедура не может вызвать сама себя, т.к. первый адрес возврата будет уничтожен вторым вызовом.

Еще более удачное решение – помещать адрес возврата в регистр. Но если процедура рекурсивна, то каждый раз необходимо помещать адрес в новое место, а количество регистров ограничено.

Самое лучшее решение – поместить адрес возврата в стек. Когда процедура завершена, она выталкивает адрес возврата.

### 7.9. Сравнение наборов команд

Наборы команд сравниваемых архитектур существенно отличаются друг от друга.

Pentium 4 – классическая двухадресная 32-битная машина CISC. Эта машина с долгой историей и она содержит много команд, которые обращаются к памяти.

UltraSPARC II – это современная трехадресная 64-битная машина RISC с архитектурой загрузки/сохранения, всего двумя способами адресации и компактным набором команд.

8051 – небольшой встроенный процессор, устанавливаемый на единственную микросхему..

Говорят, что в основу разработки компьютера Pentium 4 лежали три основных фактора:

1. Обратная совместимость.
2. Обратная совместимость.
3. Обратная совместимость.

Сейчас бы никто не начал разработку машины с такой нерегулярной системой команд, с таким маленьким количеством абсолютно разных регистров. По этой причине очень сложно писать компиляторы. Из-за недостатка регистров компиляторам постоянно приходится сохранять переменные в памяти, затем загружать их, что очень невыгодно даже при наличии трех уровней кэш-памяти. Для обеспечения высокой производительности компьютера было найдено много оригинальных технических решений.

Современная разработка уровня команд представлена в процессоре UltraSPARC III. Он содержит полную 64-битную архитектуру команд. Процессор содержит много регистров и имеет набор команд, в которых преобладают трехрегистровые операции, а также небольшая группа команд записи/считывания. Все команды одного размера, хотя число форматов вышло из-под контроля. Большинство новых разработок очень похожи на UltraSPARC III, но содержат меньшее число форматов команд.

В микросхеме 8051 реализован простой стандартный набор команд. Отличительная особенность – четыре набора регистров для ускоренной обработки прерываний, возможность доступа к регистрам в пространстве памяти и мощные команды побитовой обработки. Основное преимущество – низкая стоимость.

Ядро современного компьютера представляет собой сильно конвейеризированное трехрегистровое устройство загрузки/сохранения типа RISC. UltraSPARC II декларирует эту структуру. Pentium II скрывает систему RISC, перенимает старую архитектуру команд и разбивает команды CISC на микрооперации RISC. Машина PicoJava также использует ядро архитектуры RISC, но для этого комбинируется несколько команд для получения одной операции RISC.

### 7.10. Поток управления

Поток управления – последовательность, в которой команды выполняются динамически, т.е. во время работы программы. При отсутствии переходов и вызовов процедур команды вызываются из последовательных ячеек памяти. Вызов процедуры влечет за собой изменение поток управления, выполняемая в данный момент процедура останавливается, и начинается выполнение вызванной процедуры. Сопрограммы связаны с процедурами и вызывают схожие изменения в потоке управления. Сопрограммы используются при решении задач моделирования, для моделирования параллельных процессов. Ловушки и прерывания также меняют поток управления и возникают при определенных ситуациях.

#### 7.10.1. Последовательный поток управления и переходы

В большинстве команды выполняются последовательно и состояние счетчика команд представляет собой возрастающую функцию.

Если программа содержит переход, то соответствие между порядком расположения команд и порядком их выполнения нарушается. В результате последовательность выполнения команд из самой программы уже не видна. Это привело к появлению статьи Дейкстры под названием «Оператор goto следует считать вредным», в которой предлагалось избегать использовать этот оператор. Эта статья дала толчок революции в программировании. Одним из нововведений было устранение операторов goto более структурированными формами потока управления, например, while. Конечно, эти программы компилируются в программы второго уровня, которые могут содержать команды переходов, поскольку без них не обойтись.

#### 7.10.2. Процедуры

Самым важным способом структурирования программ является процедура. Процедура с одной стороны вызывает изменение порядка следования операций, но с другой стороны процедуру можно интерпретировать как определение новой команды на более высоком уровне.

Особый интерес представляет *рекурсивная процедура*. Это такая процедура, которая вызывает сама себя либо непосредственно, либо через цепочку других процедур.

#### 7.10.3. Сопрограммы

В обычной последовательности вызовов существует четкое различие между вызывающей процедурой и вызываемой процедурой.

Рассмотрим две процедуры А и В. Процедура А вызывает процедуру В. По окончанию процедуры В происходит возврат к процедуре А, и она продолжает выполняться дальше, с команды, следующей за командой вызова процедуры В. При повторном вызове процедуры В она будет выполняться с самого начала, а не с места возврата в процедуру А (рис. 4.25).

Иногда нужно иметь две процедуры А и В, каждая из которых вызывает другую в качестве процедуры так, как это показано на рис. 4.26.



При возврате из В к А процедура В совершает переход к тому оператору, за которым последовал вызов процедуры В. Когда процедура А передает управление процедуре В, она возвращается не к самому началу процедуры в, а к тому месту, на котором произошел предыдущий вызов. Две процедуры, работающие таким образом, называются *сопрограммами.*

Сопрограммы обычно используются для того, чтобы производить параллельную обработку данных на одном процессоре. Каждая сопрограмма работает как бы параллельно с другими сопрограммами, как будто у нее есть собственный процессор. Такой подход упрощает программирование некоторых приложений, например, при решении задач моделирования. Такой подход также полезен для проверки программного обеспечения, которое потом будет работать на мультипроцессоре.

#### 7.10.4. Ловушки

*Ловушка (trap)* – это особый тип вызова процедуры, который происходит при определенном условии. Обычно это очень важное, но редко встречающееся условие (например, переполнение). При возникновении такого условия поток управления переходит в какую-то фиксированную ячейку памяти. В этой ячейке памяти находится команда перехода к специальной процедуре – *обработчику системных прерываний*, которая выполняет какое-то определенное действие.

Важно то, что этот вид прерываний вызывается каким-то исключительным условием, вызванным самой программой и обнаруженным аппаратными средствами или микропрограммой. Конечно, обнаружить такую ситуацию можно и программными средствами, но при этом потребуется достаточно времени для постоянного программного контроля такой ситуации, а ловушка экономит время.

#### 7.10.5. Прерывания

Прерывания – это изменения в потоке управления, вызванные не самой программой, а чем-либо другим и обычно связано с процессом ввода-вывода. Как и ловушка, прерывание останавливает работу программы и передает управление программе обработки прерываний, которая выполняет некоторое действие. После окончания этого действия программа обработки прерываний передает управление прерванной программе. Она должна заново начать прерванный процесс в том же самом состоянии, в котором она находилась, когда произошло прерывание. Это значит, что прежнее состояние всех регистров должно быть восстановлено.

Различие между ловушкой и прерыванием следующее: ловушки синхронны с программой, а прерывания асинхронны.

Для иллюстрации работы прерываний, рассмотрим пример: компьютер должен вывести на терминал строку символов. Программное обеспечение сначала собирает в буфер все символы, инициирует глобальную переменную prt, указывающую на начало буфера и устанавливает вторую глобальную переменную count, которая равна числу выводимых символов. Затем ПО проверяет, готов ли терминал и, если терминал готов, выводит первый символ. Начав процедуру вывода, центральный процессор освобождается и может запустить другую программу.

Через некоторое время символ появляется на экране. Теперь может начаться прерывание. Основные шаги в упрощенной форме следующие.

*Действия аппаратного обеспечения:*

1. Контроллер устройства устанавливает линию прерывания на системной шине.
2. Когда центральный процессор готов к обработке прерывания, он устанавливает символ подтверждения прерывания на шине.
3. Когда контроллер устройства узнает, что сигнал прерывания был подтвержден, он помещает небольшое целое число на информационные линии, что бы «представиться» (то есть показать, что это за устройство). Это число – номер прерывания.
4. Центральный процессор удаляет номер прерывания с шины и временно его сохраняет.
5. Центральный процессор помещает в стек счетчик команд и слово состояния программы.
6. Затем центральный процессор определяет местонахождение нового счетчика команд, используя номер прерывания в качестве индекса. Например, если размер счетчика команд составляет 4 байта, тогда номер прерывания n соответствует адресу 4n. Новый счетчик команд указывает на начало программы обслуживания прерываний для устройства, его вызвавшего. Иногда помимо этого загружается или изменяется слово состояния программы.

*Действия программного обеспечения:*

1. Программа обработки прерываний сохраняет все нужные ей регистры таким образом, чтобы их можно было восстановить позднее. Их можно сохранить в стеке или в системной таблице.
2. каждый номер прерывания разделяется всеми устройствами данного типа, поэтому в данный момент времени еще не известно, какое устройство вызвало прерывание. Номер устройства можно считать из какого-нибудь регистра.
3. Теперь можно считывать любую другую информацию о прерывании, например, коды состояния.
4. Если происходит ошибка ввода-вывода, ее нужно обработать здесь.
5. Глобальные переменные prt и count обновляются. Первая увеличивается на1 для того, чтобы показать следующий байт, а вторая уменьшается на 1, чтобы указать, что осталось ввести на один байт меньше. Тот символ, на который указывает в данный момент prt, копируется в выходной буферный регистр.
6. В случае необходимости выдается специальный код, который сообщает устройству или контроллеру прерывания, что прерывание обработано.
7. Восстанавливаются все сохраненные регистры.
8. Выполнение команды RETURN FROM INTERRUPT (выход из прерывания): возвращение центрального процессора в то состояние, в котором он находился до прерывания. После этого компьютер продолжает работать с того места, в котором ее приостановил.

С прерываниями связано понятие *прозрачности*. Когда происходит прерывание, производятся какие-либо действия и запускаются какие-то программы, но когда все закончено, компьютер должен вернуться точно в то же состояние, в котором был до прерывания. Программа обработки прерываний, обладающая этим свойством, называется прозрачной.

Если компьютер имеет только одно устройство ввода-вывода, тогда прерывания работают точно так, как было описано. Однако большой компьютер может содержать несколько устройств ввода-вывода, и возможна ситуация, что во время работы программы обработки прерываний другое устройство захочет произвести свое прерывание.

Для разрешения этой ситуации существуют два подхода.

Первый подход – для всех программ обработки прерываний в первую очередь (даже до сохранения регистров) предотвратить последующие прерывания.

Такой подход имеет недостаток, который заключается в том, что существуют устройства, которые не могут ждать. Если компьютер имеет подобные устройства, то необходимо приписать каждому устройству приоритет. Центральный процессор тоже должен иметь приоритет, который определяется по одному из полей слова состояния программы. Если устройство с некоторым приоритетом вызывает прерывание, то и программа обработки прерываний должна обладать таким же приоритетом.