Учебное пособие

П.Б. Могнонов

ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

П.Б. Могнонов

ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

•

Учебное пособие

Улан-Удэ
Издательство ВСГТУ
2003

УДК. 681.3(075.8)
ББК. 32.972 я 73

Рецензенты:
кафедра системотехники Сибирского государственного технологического университета (зав. кафедрой, д.т.н., профессор *Г.А.Доррер*);
д.т.н., профессор, академик АИНРФ *Ю.Ф.Мухопад*

Могнонов П.Б. Организация микропроцессорных систем: Учебное пособие. - Улан-Удэ: Изд-во ВСГТУ, 2003. – 355 с.

В учебном пособии рассмотрены вопросы организации микропроцессорных систем. Показаны принципы построения основных подсистем: микропроцессора, памяти и ввода-вывода. Подробно освещены вопросы организации высокопроизводительных вычислительных систем на базе перспективных микропроцессоров. Показаны современные тенденции построения микропроцессоров.

Пособие предназначено для студентов, обучающихся по направлению 654600 «Информатика и вычислительная техника».

Печатается по решению редакционного издательского совета Восточно-Сибирского государственного технологического университета

ISBN 5-89230-014-5

ББК 32.972 я 73

Глава 1

ОСНОВЫ ПОСТРОЕНИЯ ЦИФРОВЫХ СИСТЕМ

1.1.Основные понятия и принципы теории систем. ЭВМ как сложная система

Система - это совокупность элементов, объединенных в одно целое для достижения определенной цели. При этом под целью понимается множество результатов, определяемых назначением системы. ЭВМ - это система, предназначенная для автоматизации вычислений на основе алгоритмов.

Чтобы описать систему, необходимо определить функцию и структуру системы. Функция системы - правило получения результатов, определяемых назначением системы. Другими словами, функция системы - это описание процессов, которые имеют место в системе. Функции ЭВМ чаще всего описываются в форме алгоритмов.

Структура системы - фиксированная совокупность элементов и связей между ними. Математической формой изображения структуры системы является граф. Инженерная форма отображения структуры системы является схема. Схема и граф тождественны по своему содержанию и различаются лишь по форме. В схемах для обозначения элементов используются различные геометрические фигуры, разнообразие форм которых облегчает чтение схем.

Система описана, если заданы ее функция и структура. Функция определяет порядок процессов в системе, а структура - состав и взаимосвязь элементов (частей), из которых состоит система.

Системам присуще следующее качество. Свойства совокупности элементов, объединенных в одну систему, не являются простой суммой свойств элементов, а имеют новое качество, отсутствующее в элементах.

Принцип (способ), по которому объединение элементов приводит к появлению новых свойств, отличных от свойств элементов, называется принципом организации. Другими словами, организация системы - это способ построения с целью осуществления определенных функций в системах, состоящих из большого числа элементов.

Организация - понятие более высокого порядка, чем функция и структура. Функция и структура - это конкретизация принципа организации, всего лишь один вариант организации. С другой стороны, различные принципы организации приводят к системам, различающимся своими структурами и функциями, но тождественными по своим свойствам, своему назначению.

Принцип порождения функций, необходимых и достаточных для обеспечения определенных свойств систем, определяется термином функциональная организация. Функциональная организация - это принцип построения абстрактных систем, заданных своими функциями.

Принцип порождения структур, необходимых и достаточных для реализации заданных функций, определяется термином структурная организация. Структурная организация - это принципы перевода абстрактных систем, заданных в виде функций, в материальные системы, состоящие из физически существующих элементов.

Проектирование систем (ЭВМ, цифровых устройств) базируется на определенных принципах организации систем с требуемыми свойствами: автоматически выполнять вычисления, хранить информацию, выполнять арифметические операции и т.д.

Существенной особенностью ЭВМ является его сложность. Сложность - это свойство систем, состоящих из большого числа элементов. Для сложных систем характерно то, что функция, реализуемая системой, не может быть представлена как композиция функций, реализуемых наименьшими элементами системы. Отсюда следует, что для определения системы необходимо использовать несколько форм описания функций и структур - иерархии функций и структур. Это объясняется эффектом организации, порождающим в

совокупностях элементов новые свойства.

Например, в совокупности электронных элементов ЭВМ - свойство выполнять логические операции; в совокупности логических элементов - свойство складывать, или сравнивать числа; в совокупности устройств - свойство реализовывать вычисления на основе алгоритмов.

Иерархический подход к описанию сложных систем предполагает, что на высшем уровне иерархии система рассматривается как один элемент, имеющий входы и выходы для связи с внешними объектами. В этом случае функция системы не может быть задана подробно и представляется как отображение состояний входов на состояния выходов системы. Чтобы раскрыть устройство и порядок функционирования системы, глобальная функция и сама система разбиваются на части - функции и структурные элементы следующего, более низкого уровня иерархии. В свою очередь эти части опять детализируются, и так до тех пор, пока функция и структура системы не будут полностью раскрыты, с необходимой степенью подробности.

Очевидно, что элемент - это, прежде всего, удобное понятие, а не физическое свойство.

Таким образом, любая система содержит в себе иерархию процессов и элементов, логика которой приводит к необходимости использовать иерархию функций и структур описания того, как устроена и работает система. Нижележащий уровень иерархии раскрывает сущность процессов и устройство элементов, относящихся к более высокому уровню.

1.2.Функциональный подход к построению цифровых систем

Устройство и порядок функционирования систем, в том числе и ЭВМ, предопределяются, с одной стороны, назначением системы, а с другой стороны, - элементной базой, «материалом» из которого строится система. Под элементной базой понимается набор различных элементов, по основе которого строится система, и называется структурным базисом. Набору элементов структурного базиса соответствует набор функций, называемый функциональным базисом. Отсюда следует, что система должна проектироваться исходя из её назначения с учетом свойств структурного базиса, существующего на данный момент проектирования, или потенциально возможной.

Пусть назначение системы задаётся в виде функции F. Исходя из содержания и свойств функции F, можно определить структурный базис элементов пригодный для создания системы. Когда структурный базис элементов определен, становится известным функциональный базис $\varphi_{1,\dots},\varphi_N$, реализуемых элементами.

Функции элементов $\varphi_{1,\dots,}\varphi_N$ более просты, чем функция системы F. По этой причине, чтобы выявить структуру S, реализующую функцию F, необходимо путем формальных преобразований построить эквивалентную функции F функцию Ф, представленную в виде композиций функций $\varphi_{1,\dots,}\varphi_N$. Таким образом, функция F будет детализирована до элементарных функций $\varphi_{1,\dots,}\varphi_N$, для реализации каждой из которых имеется определенный структурный элемент.

Местоположение функций $\varphi_{1,\dots,}\varphi_N$ в записи Ф предопределяет места соответствующих элементов в структуре. Поэтому функцию Ф можно использовать в качестве формы для построения структуры S, состоящей из элементов типа $\varphi_{1,\dots,}\varphi_N$ и реализующей заданную функцию Ф = F, т.е. функцию системы.

Если система сложная, указанная процедура повторяется по отношению к каждой из функций $\varphi_{1,\dots,}\varphi_N$, реализуемых элементами структуры S. В результате этих действий будут выявлены структура элементов верхнего уровня и структуры элементов последующих уровней иерархии. Описанный подход к проектированию систем базируется в предположении о доминирующей роли функций в отношении структур. Как следствие доминирующей роли функций, структура системы определенная для верхних уровней

иерархии, играют преобладающую роль в отношении структур низших уровней, т.е. состав элементов и связей в структурах верхних уровней предопределяет состав элементов и связей в структурах низших уровней. Из сказанного следует, что системы должны проектироваться по принципу "сверху – вниз" - от верхнего уровня представления функций и структур к его нижнему.

Подход к проектированию систем, основанный в признании доминирующей роли функций в отношении структур, называется функциональным подходом. В основу проектирования ЭВМ заложен принцип первичности функций в отношении структур, из которого следует, что структура ЭВМ, т.е. конфигурация схем, предопределяется функцией (назначением) ЭВМ.

Для унификации различных структурных решений используются понятия операционного устройства и интерфейса. Эти понятия вводятся в функциональном отношении. Операционное устройство - это преобразователь дискретной информации с произвольной функцией. Интерфейс определяется как алгоритм обмена информации между устройствами. Введение заданных понятий создает основу для построения методики проектирования разнообразных операционных устройств, таких как процессор, процессоров ввода - вывода, контроллеров внешних устройств ЭВМ и т.д. В основу проектирования операционных устройств различного назначения положен принцип функционального микропрограммирования и представления устройства как композиции операционного и управляющих автоматов.

1.3.Принцип программного управления как основной принцип построения ЭВМ

Современные ЭВМ строятся на основополагающем подходе – принципе программного управления. В основе этого принципа лежит представление алгоритма в форме операторной схемы, которая задает правило вычислений как композицию операторов (операций над данными) двух типов: операторов, обеспечивающих преобразование информации, и операторов, анализирующих информацию с целью определения порядка выполнения операторов. Принцип программного управления может быть реализован в ЭВМ многими способами. Один из способов реализации программного управления был предложен в 1945 году Дж. фон Нейманом и с тех пор *НЕЙМАНОВСКИЙ ПРИНЦИП* программного управления используется в качестве основного принципа построения всех современных ЭВМ. Этот принцип состоит в следующем.

Информация кодируется в двоичной форме и разделяется на единицы (элементы) информации, называемые *словами*.

Разнотипные слова различаются по способу использования, но не способами кодирования.

Слова информации размещаются в ячейках памяти машины и идентифицируются номерами ячеек, называемые *адресами слов*.

Алгоритм представляется в форме последовательности управляющих слов, которые определяют наименование операции и слова информации, (называемые операндами) участвующие в операции, и называются *командами*. Алгоритм, представленный в терминах машинных команд, называется *программой*.

Выполнение вычислений, предписанных алгоритмом, сводится к последовательности выполнению команд в порядке, однозначно определяемой программой.

Первый пункт. Представление информации в виде нулей и единиц значительно упрощает конструкцию ЭВМ, а также повышает ее надежность. Совокупность нулей и единиц, используемых для представления отдельных чисел, команд и т.п., рассматриваются как самостоятельные информационные объекты и называются словами. Слово обрабатывается ЭВМ как одно целое – как машинный элемент информации.

Второй пункт. Все слова, представляющие числа, команды и т.п., выглядят в ЭВМ совершенно одинаково и сами по себе неразличимы. Только порядок использования слов в

программе вносит различие в слова. Отсюда следует, что команды программы становятся в такой же степени доступными для обработки, как и числа. А это приводит к интересным возможностям, таким как одни и те же операции можно использовать для обработки слов различной природы.

Третий пункт. Этот пункт указывает на специфику хранения и поиска информации, порождаемую свойствами машинной памяти. Машинная память – это совокупность ячеек, где ячейка выделяется для хранения элемента информации (слова). Идентифицируются ячейки памяти с помощью адресов. Поэтому для обозначения слов (команд и т.п.) в ЭВМ нет никаких средств, кроме адресов, присваиваемых величинам, командам в процессе составления программы вычислений.

Четвертый пункт. Алгоритм представляется в виде упорядоченной последовательности команд следующего вида:

1 … e	1… m	1… m		1… m
КОП	A_1	A_2	…	A_n

(1.1)

КОП, A_1, A_2, …, A_n - поля команды, представляющие соответственно код операции и адреса операндов, участвующих в операции. Команда (1.1) позволяет инициировать 2^e операций, и каждый адрес может принимать 2^m различных значений, обеспечивая ссылку на любую из 2^m величин или команд.

Форма (1.1) характеризует *структуру*, или иначе *формат команды*. Требуемый порядок вычислений предопределяется алгоритмом и описывается последовательностью команд, образующих программу вычислений.

Пятый пункт. Процесс вычислений, выполняемый ЭВМ по заданной программе, состоит в последовательном выполнении команд. Первой выполняется команда, заданная пусковым адресом программы. Адрес следующей команды однозначно определяется в процессе выполнения текущей команды и может быть адресом следующей команды, либо адресом любой другой команды. Процесс вычислений продолжается до тех пор, пока не будет выполнена команда, предписывающая прекращение вычислений.

Важно отметить, что вычисления, производимые машиной, определяются программой. Следовательно, многообразие программ, которые могут быть выполнены ЭВМ, определяет класс функций, которые способна реализовать ЭВМ.

К настоящему времени принцип программного управления, предложенный Дж. фон Нейманом, является ведущим принципом построения современных ЭВМ. Это объясняется тем, что хотя возможности неймановских машин доведены до предела, пока они удовлетворяют потребностям в вычислениях во многих областях применений. С другой стороны, повышение требований к производительности и надежности ЭВМ диктует потребность в пересмотре классического неймановского принципа построения ЭВМ и систем, чтобы приблизить машинные формы представления данных и алгоритмов к естественным языкам.

Так широкое распространение получил принцип построения ЭВМ с развитыми системами интерпретации, разработанный коллективом, возглавляемым В.М. Глушковым.

ЭВМ этого класса обеспечивает восприятие алгоритмов, записанных на языках высокого уровня - в виде знаков, операций, наименование величин и данных, представляемых в естественной форме. Эти указанные возможности реализуются за счет введения в ЭВМ нетрадиционных средств адресации и операций над информацией.

1.4.Обобщенная структура ЭВМ и его информационная модель

Структура (архитектура) ЭВМ может быть реализована различными способами. Детальные структурные схемы отдельных типов ЭВМ могут значительно отличаться друг от друга. Однако, в связи с тем, что микроЭВМ представляют собой определенный тип цифровой вычислительной машины, их структурные схемы довольно близки. Такие ЭВМ,

функционирующие на основе принципа фон Неймана, называются машиной неймановского класса.

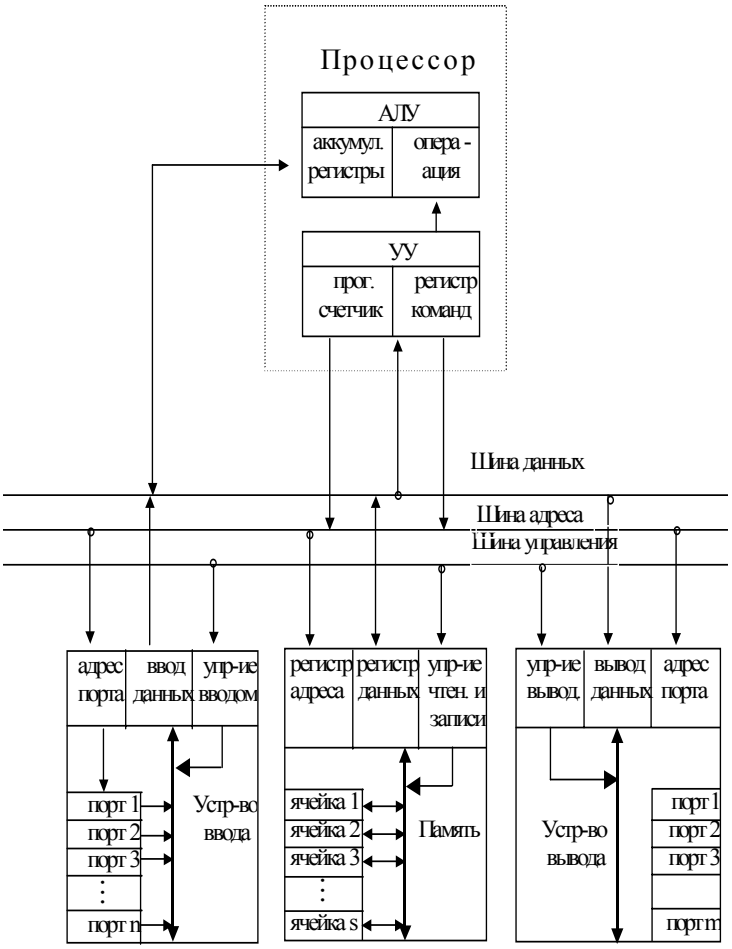


Рис.1.1. Обобщенная структурная схема ЭВМ

В общем случае ЭВМ, обобщенная структурная схема которой показана на рис. 1.1, состоит из центрального процессора, памяти и внешних (периферийных) устройств.

Процессор – центральное устройство ЭВМ. Процессор «воспринимает» программу и на ее основе управляет работой всех устройств ЭВМ, инициируя выполнение действий в памяти и устройствах ввода-вывода.

Таким образом, центральный процессор является блоком обработки данных. Он выбирает команды из памяти, дешифрирует их и выполняет. Он вырабатывает временные сигналы и сигналы управления, передает данные в память и из памяти, а также из устройств ввода-вывода, выполняет арифметические и логические операции и идентифицирует внешние сигналы.

В течение каждого цикла команды ЦП выполняет много управляющих функций:

1. Помещает адрес команды в адресную шину памяти;
2. Получает команду из шины данных и дешифрирует ее;
3. Выбирает адреса и данные, содержащиеся в команде; адреса и данные могут находиться в памяти или в регистрах;
4. Выполняет операцию, определенную в коде команды; операцией может быть арифметическая или логическая функция, передача данных или функция управления;
5. Следит за управляющими сигналами, такими как прерывание, и реагирует соответствующим образом;
6. Генерирует сигналы состояния, управления и времени, которые необходимы для нормальной работы УВВ и памяти.

Согласно принципу программного управления и обобщенной структурной схемы в составе ЭВМ можно выделить следующее множество функциональных блоков:

$$M = \{a, b, c, d, e\},$$

где, a – устройство ввода, b – арифметико-логическое устройство (операционный автомат), c – устройство управления (управляющий автомат), d – запоминающее устройство, e – устройство вывода.

Информационный граф такой ЭВМ представлен на рис.1.2. Узлами приведенного информационного графа являются соответствующие функциональные блоки ЭВМ, а дугами – информационные потоки между соответствующими устройствами. Эта модель позволяет наглядно отобразить весь информационный обмен между устройствами ЭВМ.

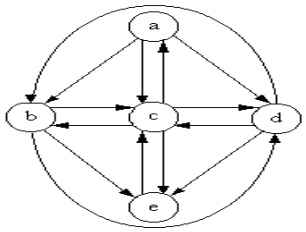


Рис.1.2. Информационный граф

Объединение функциональных блоков в ЭВМ производится с помощью системы шин: шины данных, по которой осуществляется обмен информацией между блоками ЭВМ; шины адреса, используемой для передачи адресов, по которым осуществляется обращение к различным устройствам ЭВМ; шины управления для передачи управляющих сигналов.

Обобщенный алгоритм функционирования ЭВМ представлен на рис. 1.3.

Рис.1.3. Обобщенный алгоритм функционирования ЭВМ

Одно из основных различий между ЭВМ – это организация системных шин, обеспечивающих связь между отдельными блоками ЭВМ. По этому признаку все структуры малых ЭВМ могут быть квалифицированы следующим образом: ЭВМ с многошинной структурой и ЭВМ с общей шиной.

Типичная архитектура ЭВМ с многошинной структурой представлена на рис.1.4.:

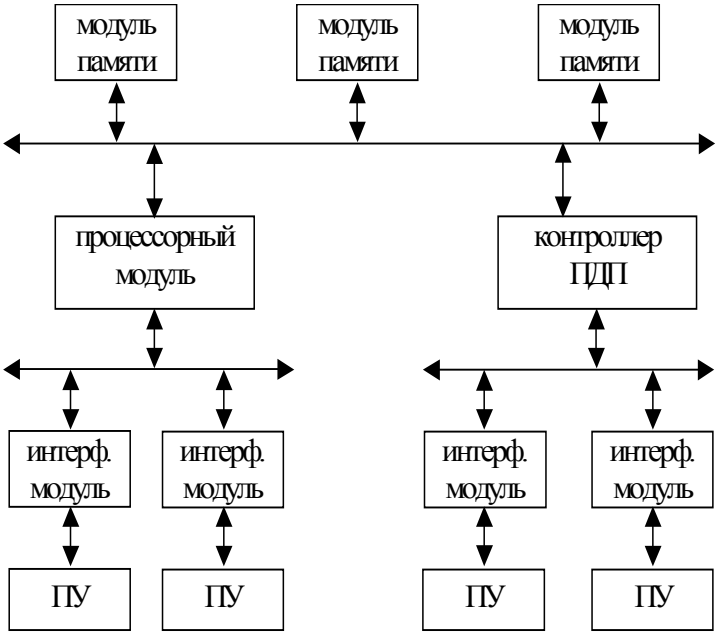


Рис.1.4. ЭВМ с многошинной структурой

Основная особенность ее организации состоит в том, что для каждого способа обмена информацией с ПУ используется отдельная группа шин: отдельные шины для программного режима обмена информации с прерываниями или без прерываний и для ввода-вывода информации в режиме прямого доступа к памяти, которые передают блоки данных с большой скоростью.

Протоколы обмена данными, структура шин и быстродействие при обмене для каждой из групп шин могут, оптимальным образом, адаптированы к ПУ в соответствии с выбранным методом.

Другой популярной структурой шин, которая используется во многих ЭВМ, является структура с общей шиной (рис.1.5.).

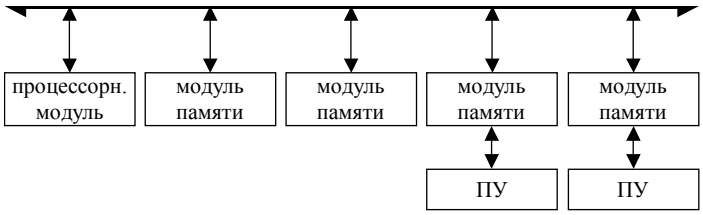


Рис.1.5. ЭВМ с одношинной структурой

В этом случае блоки ЭВМ объединяются посредством одной группы шин, в которую входят подмножества шин данных, адреса и управляющих сигналов.

При такой организации системы шин обмен информацией между процессором, периферийными устройствами и памятью выполняется по единому правилу, отдельные команды ввода-вывода для обращения к ПУ в системе команд отсутствуют. Это позволяет повысить гибкость и эффективность ЭВМ, так как весь набор команд обращения к памяти может использоваться для передачи и обработки содержимого регистров ПУ. Кроме того,

другим важным достоинством является простота структуры шин и минимизация числа связей для обмена информацией между устройствами ЭВМ.

1.5.Классы устройств ЭВМ

Согласно обобщенной структуре ЭВМ и в зависимости от функций устройства ЭВМ можно подразделить на следующие классы операционные, запоминающие и ввода- вывода. Различие функций, реализация которых возлагается на устройства различных классов, порождает множество способов структурной организации этих устройств, т.е. типов элементов, из которых строятся устройства и способов соединения элементов между собой.

Операционные устройства. Операционным называется устройство, предназначенное для выполнения множества операций $F = \{f_1,...,f_G\}$ над операндами, представляемыми множеством слов $D = \{d_1,...,d_H\}$, с целью вычисления слов $R = \{r_1,...,r_Q\}$, определяющих значения результатов, причем в каждый момент времени устройство выполнит единственную операцию $R = f_g(D)$, выделяемую номером (кодом) $g = 1,...,G$.

Под операцией f_g понимается вычисление значения функций в точке $D = D^*$, и любая вычисляемая функция может рассматриваться как операция. Устройство может реализовать как простейшие операции вида $r_q = d_h$ - присваивание к результату r_q значение операнда d_h , так и сколь угодно сложные операции, заданные в форме алгоритмов.

Операционное устройство как структурный элемент изображен на рис.1.6,а.

На вход устройства поступают входные слова $D = \{d_1,...,d_H\}$ и код g операции f_g, которую должно выполнить устройство. Код операции инициирует работу устройства, и через некоторое время на выходе формируется значение результата $R = f_g(D)$.

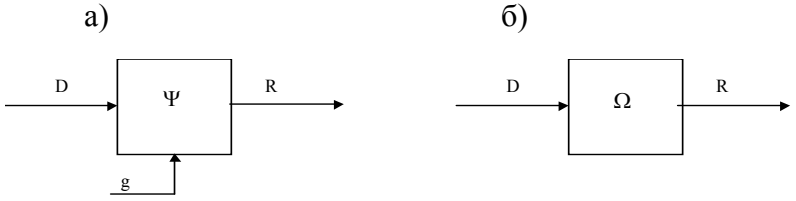


Рис.1.6. Операционное устройство

Операционное устройство можно рассматривать как преобразователь дискретной информации. Функцию преобразователя в этом случае удобно определить следующим образом. Слово g определяющее код операции f_g, можно считать элементом $d_0 = g$ множества входных слов $D = \{d_1,..., d_H\}$. Тогда обозначая через Ω оператор, реализуемый устройством, функция устройства будет изображаться в виде:

$$R = \Omega(D),$$

где

$$\Omega = \begin{cases} f_1, & \text{если } d_0 = 1; \\ \cdot & \\ \cdot & \\ \cdot & \\ f_G, & \text{если } d_0 = G \end{cases}$$

Структура этого устройства представлена на рис.1.6.б, где символ Ω определяет функцию, т.е. тип устройства.

Функция операционного устройства определена, если задана множества входных слов D, выходных слов R и операций F, реализуемых устройством.

Операционные устройства строятся из логических и запоминающих элементов. Чаще всего используются логические элементы универсального базиса, реализующие функцию И-НЕ. В качестве запоминающих элементов используются триггеры.

Основные характеристики операционных устройств это быстродействие и затраты оборудования. Быстродействие устройства определятся средним количеством операций, реализуемых устройством в единицу времени. Быстродействие вычисляется как величина, обратная среднему времени выполнения операции в устройстве:

$$V = 1/U$$

где $U = \sum_{g=1}^{G} p_g \tau_g$; p_g - вероятность выполнения операции $f_g(p_1 + p_2 + ... + p_G = 1)$; τ_g - среднее время выполнения операции f_g.

Затраты оборудования в операционном устройстве оцениваются суммарной стоимостью элементов, составляющих устройство.

К операционным устройствам ЭВМ относятся процессоры, процессоры канала ввода - вывода, устройства управления внешними устройствами и т.п.

Запоминающие устройства (ЗУ). Запоминающим устройством (накопителем) называется устройство, предназначенное для хранения множества элементов информации и снабженное средствами селекции, обеспечивающие запись и (или) чтение заданного элемента информации. Элементами информации могут быть биты, байты, слова и поля переменной длины - записи состоящие из последовательностей битов и байтов. Элемент информации выделяется на множестве элементов с помощью адреса, либо с помощью ключа, либо другими способами. Адрес - число, определяющее местоположение элемента информации в устройстве. Ключ - это признак, идентифицирующий элемент информации.

Основными характеристиками запоминающих устройств являются емкость, быстродействие и стоимость. Емкость ЗУ определяется предельным количеством элементов информации, размещаемых в ЗУ. Емкость исчисляется в битах, байтах и иногда в словах. Быстродействие ЗУ определяется затратами времени на обращение к ЗУ с целью чтения или записи информации. Удельная стоимость - это затраты средств (в рублях) на хранение единицы информации (бита или байта).

Устройства ввода-вывода. Устройством ввода-вывода называется устройство, предназначенное для чтения и (или) записи информации на носитель путем преобразования электрических сигналов в сигналы иной физической природы. Функция устройства ввода-вывода – это преобразование сигналов одной физической природы в сигналы другой природы, т.е. передачи информации из одной среды в другую. На входе устройства ввода имеется носитель информации – клавиатура, переключатели, графические планшеты и т.п. – а на выходе формируется совокупность электрических сигналов, представляющих информацию, последовательно считываемую с носителя. Устройство вывода выполняет обратное преобразование. Например, последовательность электрических сигналов преобразуется в механические перемещения печатающего устройства.

Основные характеристики устройств ввода-вывода – это скорость ввода-вывода и стоимость. Скорость ввода-вывода определяется числом символов информации, вводимых - выводимых за единицу времени. Для некоторых устройств, скорость ввода-вывода задается специфическими значениями: число строк, печатаемых в минуту и т.д.

1.6. Принцип микропрограммного управления

Функциональная и структурная организация операционных устройств базируется на принципе микропрограммного управления, который определяется следующим образом.

Любая операция f_g , g = 1,…,G, реализуемая устройством, рассматривается как сложное действие, которое разделяется на последовательность элементарных действий над словами информации, называемых микрооперациями.

Для управления порядком следования микроопераций используются логические условия, которые в зависимости от значений слов, преобразуемых микрооперациями, принимают значения "истина" или "ложь" ("1" или "0").

Процесс выполнения операций в устройстве описывается в форме алгоритма, представленного в терминах микроопераций и логических условий, и называется микропрограммой. Микропрограмма определяет порядок проверки значений логических условий и следования микроопераций, необходимых для получения требуемых результатов.

Микропрограмма используется как форма представления функции устройства, на основе которой определяется его структура, и порядок функционирования во времени.

Этот принцип определяет процесс функционирования устройства, как процесс реализации микроопераций и проверки логических условий, предопределяемой микропрограммой, и позволяет упорядочить и формализовать построение операционных устройств различного назначения.

В функциональном и структурном отношении операционное устройство разделяется на две части: операционный и управляющий автоматы (рис.1.7).

Операционный автомат ОА служит для хранения информации, выполнения набора микроопераций и вычисления значений логических условий, т.е. операционный автомат, является структурой, организованной для выполнения действий над информацией. Микрооперации, реализуемые операционным автоматом, инициируются множеством *управляющих сигналов* $Y = \{y_1,\ldots,y_N\}$, с каждым из которых отождествляется определенная микрооперация.

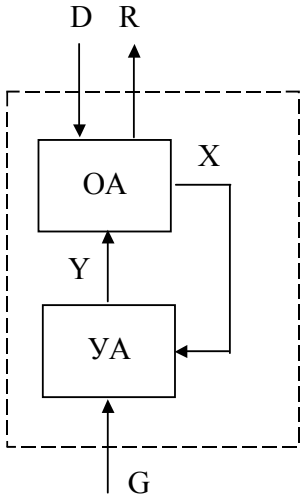


Рис.1.7. Структура операционного устройства

Значения логических условий, вычисляемые в операционном автомате, отображаются множеством *осведомительных сигналов* $X = \{x_1,\ldots,x_L\}$, каждый из которых отождествляется с определенным логическим условием. *Управляющий автомат УА* генерирует последовательность управляющих сигналов, предписанную микропрограммой и соответствующую значениям логических условий. Иначе говоря, управляющий автомат задает порядок выполнения действий в операционном автомате, вытекающий из алгоритма выполнения операций. Наименование операции, которую необходимо выполнить в устройстве, определяется кодом g операции. По отношению к управляющему автомату сигналы $g_1,\ldots,g_h$, посредством которых кодируется наименование операции и осведомительные сигналы $x_1,\ldots,x_L$,формируемые в операционном автомате, играют одинаковую роль: они влияют на порядок выработки управляющих сигналов Y. Поэтому сигналы $g_1,\ldots,g_h$ и $x_1,\ldots,x_L$ относятся к одному классу – осведомительных сигналов, поступающих на вход управляющего автомата.

Таким образом, любое операционное устройство – процессор, канал ввода-вывода, устройство управления внешним устройством – является композицией операционного и управляющего автоматов. Операционный автомат, реализуя действия над словами информации, является исполнительной частью устройства, работой которого управляет управляющий автомат, генерирующий необходимые последовательности управляющих сигналов.

На данном этапе рассмотрения вопроса операционный и управляющий автоматы могут быть определены своими функциями – перечнем выполняемых ими действий, исходя из которых в дальнейшем, будет определена структура автоматов.

Функции *операционного автомата* определяется следующей совокупностью сведений:

Множеством входных слов $D = \{d_1,...,d_H\}$, вводимых в автомат в качестве операндов.

Множеством выходных слов $R = \{r_1,...,r_Q\}$, представляющих результаты операций.

Множеством внутренних слов $S = \{s_1,...,s_N\}$, используемых для представления информации в процессе выполнения операций. В дальнейшем будем предполагать, что входные и выходные слова совпадают с определенными внутренними словами, т.е. $D \subseteq S$ и $R \subseteq S$.

Множеством микроопераций $Y=\{y_m\}$, $m=1,...,M$, реализующих преобразование $S=\varphi_m(S)$ над словами информации, где φ_m - вычислимая функция.

Множеством логических условий $X=\{x_l\}$, $l=1,...,L$, где $x_l = \Psi_l(S)$ и Ψ_l - булева функция.

Таким образом, функция операционного автомата задана, если определены множества D, R, S, Y, X. Заметим, что время не является аргументом функции операционного автомата. Функция устанавливает список действий – микроопераций и логических условий, - которые может выполнять автомат, но никак не определяет порядок следования этих действий во времени. Иначе говоря, функция операционного автомата характеризует средства, которые могут быть использованы для вычислений, но не сам вычислительный процесс. Порядок выполнения действий во времени определяется в форме функций управляющего автомата.

Функция управляющего автомата – это операторная схема алгоритма (микропрограммы), функциональными операторами которой являются символы $y_1,...,y_M$, отождествляемые с микрооперациями, и в качестве логических условий (предикатов) используются булевы переменные $x_1,...,x_L$.Операторная схема алгоритма наиболее часто представляется в виде граф - схемы или логической схемы алгоритма. Каждая из этих форм определяет вычислительный процесс в последовательном аспекте – устанавливает порядок проверки логических условий $x_1,...,x_L$ и порядок следования микроопераций $y_1,...,y_M$.

1.7. Уровни представления цифровых систем

Как отмечалось выше, для описания сложных систем используется иерархический принцип, который основан на применении нескольких уровней описаний, каждому из которых свойственна определенная степень детализации. Этот принцип позволяет уменьшить степень сложности описания и упрощает процесс исследования свойств системы.

На рис.1.8 представлены основные уровни описания структуры и функций цифровых машин. К нижнему, нулевому уровню относятся описания, представляемые электрическими схемами. Электрические схемы определяют структуру, как совокупность взаимосвязанных электрических элементов. Функции, реализуемые схемами, отражают взаимосвязь между электрическими параметрами, характеризующими состояние входов и выходов схемы.

№	Наимено-вание	Описывае-мые объекты	Структурный базис	Функцио-нальный базис	Единицы информации

0	Уровень электрических схем	Логические и запоминающие элементы	Электронные компоненты	Соотношения теории электрических цепей	—
1	Уровень логических схем	Операционные элементы, микропрограммные автоматы	Логические и запоминающие элементы	Операции булевой алгебры и теории конечных автоматов	Биты
2	Уровень операционных схем	Операционные устройства	Операционные элементы и микропрограммные автоматы	Микрооперации и логические условия	Слова, поля
3	Уровень структурных схем	ЭВМ и комплексы машин	Операционные устройства	Внутренние процедуры	Слова, поля
4	Программный уровень	Операционные системы	Команды и операторы	Операции из системы команд и процедур	Слова, массивы, сегменты

Рис.1.8. Уровни описания структуры и функций цифровых систем

Уровни электрических схем отражают только физические свойства процессов в ЭВМ и используются для описания информационных процессов.

На уровне логических схем описание ЭВМ детализируется до уровня запоминающих элементов и логических операций, выполняемых над битами информации, а объектами описания являются операционные элементы (сумматоры, регистры и т.д.) и микропрограммные автоматы.

На уровне операционных схем объектами описания являются операционные устройства, построение которых базируется на принципе микропрограммного управления. Структурный базис этого уровня определяется множеством операционных элементов и микропрограммных автоматов.

Структура ЭВМ как целого объекта описывается на уровне структурных схем. На этом уровне определяется весь комплекс устройств, составляющих ЭВМ, и описывается порядок взаимодействия устройств в процессе выполнения отдельной операции из системы команд машины.

Самым высоким уровнем представления функций, реализуемой машиной, является уровень программ. Процесс выполнения программ определяет вычислительный процесс, реализуемый в ЭВМ. Выделяя, отдельные операторы, можно говорить о структуре программ, а операции, выполняемые программой над множеством значений, составляют функции программы. Функции, выполняемые операционной системой, определяют операционные возможности вычислительной системы.

Контрольные вопросы

1. Как определяется функция и структура системы?
2. Определите сущность функциональной и структурной организации систем.
3. Определите основные положения принципа программного управления.
4. Укажите структурные компоненты ЭВМ и их основное назначение.
5. Какие классы устройств используются для построения ЭВМ?
6. Опишите алгоритм функционирования ЭВМ.

7. В чем заключается принцип микропрограммного управления?
8. Каким образом определяется операционное устройство?
9. Что такое операционный автомат и как задается его функция?
10. Определите управляющий автомат и опишите его функцию.
11. Укажите основные уровни описания цифровых машин.

Глава 2

ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРА

2.1. Этапы развития микропроцессорной техники

За всю историю развития микропроцессорной техники, ведущие позиции в этой области занимает американская фирма Intel (Integrated Electronics). В 1971 году она разработала и выпустила первый в мире 4-битный микропроцессор 4004. Он содержал 2300 транзисторов, выполненный по технологии 10 мкм, работал на частоте 108 кГц, адресовал 640 байт памяти и имел производительность 0,06 MIPS. Но подлинный успех фирме Intel принес 8-битный микропроцессор 8080, который был выпущен в 1974 году. Он имел частоту 2 мГц, адресовал 64 Кбайта памяти, вмещал 6000 транзисторов благодаря технологии 6 мкм. Процессор требовал трех источников питания (+5, +12, -5 В) и сложной двухтактной синхронизации. Этот микропроцессор получил очень широкое распространение во всем мире. Сейчас в нашей стране его аналог - микропроцессор К580ИК80 применяется во многих бытовых и персональных компьютерах и разнообразных контроллерах.

Следующим этапом стал микропроцессор 8085, выполненный по технологии 3 мкм, размещал на кристалле 6500 транзисторов, имел тактовую частоту 5 мГц и производительность 0,37 MIPS. Архитектура этого микропроцессора соответствовала архитектуре 8080, но в него добавили порт последовательного интерфейса, и перешли на единое питающее напряжение +5 В.

В 1979 году фирма Intel первой выпустила 16-битный микропроцессор 8086, возможности которого были близки к возможностям процессоров миникомпьютеров 70-х годов. Этот микропроцессор работал на частоте 5 мГц, производительность его составляла 0,33 MIPS, но обрабатывал 16-битные операнды. Был выполнен по технологии 3 мкм с 29000 транзисторами на кристалле. Микропроцессор 8086 оказался "прародителем" целого семейства, которое обычно называют семейством 80x86. Аналог этого микропроцессора К1810ВМ86 применяется в персональных компьютерах, выпускаемых в нашей стране.

Желание расширить адресное пространство памяти до 1 Мбайта в процессоре с 16-битными регистрами заставило использовать в микропроцессоре 8086 сегментную организацию памяти, которая в последующих микропроцессорах фирмы Intel было сохранено ради совместимости.

Через год появился МП 8088, архитектурно повторяющий МП 8086 и имеющий 16-битные внутренние регистры, но ее внешняя шина данных составляет 8 бит. Широкой популярности МП 8088 способствовало его применение фирмой IBM в ПК PC и PC/XT. Очень быстро для этих компьютеров был накоплен такой огромный объем программного обеспечения, что в последующих МП фирме Intel пришлось предусматривать специальный режим эмуляции МП 8086. Обычно этот режим называется режимом реального адреса (Real Addres Mode) или R-режим.

В 1981 году появились МП 80186/80188, которые сохраняли базовую архитектуру МП 8086/8088, но содержали на кристалле контроллер прямого доступа к памяти, счетчик/таймер и контроллер прерываний. Кроме того, была расширена система команд. Однако широкого распространения эти МП (как и ПК PCjr на их основе) не получили.

Следующим крупным шагом в разработке новых идей стал МП 80286, появившийся в 1982 году. Разработанный по технологии 1,5 мкм, он содержал 134000 транзисторов. При разработке этого МП были учтены достижения в архитектуре миникомпьютеров и больших компьютеров. МП 80286 может работать в двух режимах: в режиме реального адреса он эмулирует МП 8086, а в защищенном режиме виртуального адреса или P-режиме предоставляет программисту много новых возможностей и средств. Среди них отметим расширение адресного пространства до 16 Мбайт, появление дескрипторов сегментов и дескрипторных таблиц, наличие защиты по четырем уровням привилегий, поддержку организации виртуальной памяти (1 Гбайт) и мультизадачности. МП 80286 применялся в ПК PC/AT и младших моделях PS/2.

В 1986 году появился новый микропроцессор фирмы Intel 80386 (275000 транзисторов, 1,5 мкм). При разработке этого 32-битного МП потребовалось решать две задачи: совместимость и производительность. Первая из них решена путем введения трех режимов работы.

В R-режиме, который действует после включения питания или системного сброса, процессор копирует работу МП 8086 и использует 16-битные регистры; адресное пространство составляет 1 Мбайт памяти.

В P-режиме МП 80386 может выполнять 16-битные программы МП 80286 без каких-либо модификаций. Вместе с тем в этом режиме он может выполнять свои "естественные" 32-битные программы, что обеспечивает повышение производительности системы. В этом режиме реализуются все новые возможности и средства МП 80386, такие как масштабированная индексная адресация памяти, ортогональное использование РОН, новые команды, средства отладки, а также введен режим страничного управления памятью. Адресное пространство памяти в этом режиме составляет 4 Гбайта. Операционная система P-режима может создать задачу, которая работает в режиме виртуального процессора 8086 или V-режим. Прикладная программа, которая выполняется в этом режиме, полагает, что она работает на процессоре 8086. Однако, программе запрещается выполнять некоторые команды, связанные с управлением ввода вывода. Поэтому при нарушении правил защиты генерируется прерывание и управление передается ОС.

Позже, в 1988 году появился МП 80386 SX, который полностью совместим с процессором 80386, но имеет внешнюю шину данных 16 бит и адресное пространство 16 Мбайт. После этого полноразрядный вариант получил официальное название 80386 DX. В 1990 году был выпущен МП 80386 SL в миниатюрном корпусе и с пониженным потреблением энергии, разработанный специально для портативных компьютеров.

В 1989 году фирма Intel выпустила на рынок МП 80486, содержащий в кристалле 1,2 миллиона транзисторов и выполненный по технологии 1 мкм. Два главных отличия от предыдущего МП состоит в том, что математический сопроцессор реализован на одном кристалле с центральным процессором FPU (Floating Point Unit) и имеется внутренняя совмещенная кэш-память команд и данных емкостью 8 КБайт. Кроме того, для повышения производительности в этом CISC-процессоре применено RISC-ядро.

2.2. Архитектура микропроцессора

Микропроцессор (МП) - программно-управляемое устройство, предназначенное для обработки цифровой информации и управления процессом этой обработки, выполненное в виде одной (или нескольких) интегральной схемы с высокой степенью интеграции электронных компонентов.

Микропроцессор характеризуется очень большим числом параметров и качеств. Это такие качества и параметры как: тип корпуса, количество источников питания, требования к синхронизации, мощность рассеивания, температурный диапазон, возможность расширения разрядности, цикл выполнения команд (микрокоманд), уровни сигналов, помехоустойчивость, нагрузочная способность, долговечность и т.д.

МП, как операционное устройство микропроцессорных систем, обеспечивает эффективное автоматическое выполнение операций обработки цифровой информации в соответствии с заданным алгоритмом. Для решения широкого круга задач в различных областях применений МП должен обладать алгоритмически полной системой команд.

Для описания МП, как функционального устройства, необходимо охарактеризовать формат обрабатываемых данных и команд, количество, тип и гибкость команд, методы адресации данных, число внутренних регистров общего назначения, регистров результата, возможности организации и адресации стека, параметры виртуальной памяти и информационную емкость прямо адресуемой памяти.

Для построения микропроцессорных систем большое значение имеют средства построения системы прерываний программ, построения эффективных систем ввода-вывода данных и развитого интерфейса.

2.2.1. Типы и форматы данных

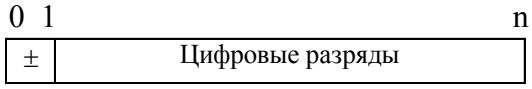
Основными типами данных, которые встречаются при обработке информации в микропроцессорах, являются числа, представленные в двоичной системе счисления, алфавитно-цифровые символы, представленные последовательностями символов, и логические значения.

Числовые значения могут быть двух типов: целое и действительное. Для представления целых значений и выполнения операций над ними используются целые двоичные числа, а для действительных значений – числа с фиксированной или плавающей запятой.

Действительные числа, представленные в форме с плавающей запятой, гарантирует возможность обработки широкого диапазона значений с высокой точностью, но при этом использование арифметики с плавающей запятой приводит к увеличению времени обработки и значительному увеличению количества оборудования. В связи с этим, в современных БИС обработки данных эта задача возлагается на специализированные микропроцессоры, и получили название сопроцессоры обработки чисел с плавающей запятой.

Алгоритмы выполнения операций над целыми числами просты и при этом достигается высокая скорость обработки информации, поэтому этот тип чисел реализуется в микропроцессорах.

Целое двоичное число имеет следующий формат:



Разряд с номером 0 является знаком числа. Знак плюс кодируется цифрой 0, знак минус – цифрой 1. Разряды от 1 до n являются цифровыми. Запятая фиксируется после младшего разряда с номером n. Этот формат обеспечивает представление целых значений в диапазоне от $-(2^n - 1)$ до $+(2^n-1)$.

Для представления отрицательных чисел наибольшее распространение получил не модифицированный дополнительный код вследствие таких его достоинств, как однозначное представление отрицательного и положительного нуля, отсутствие циклического переноса, использование одного разряда для представления знака числа. При этом признаком переполнения, например, при сложении двух чисел в дополнительном коде является наличие переноса только в знаковый разряд из старшего разряда кода или только из знакового разряда кода.

Целое двоичное число без знака X длиной в n-бит может принимать значения в диапазоне $0<X<2^n-1$, где n может равняться 8,12,16 битам. Для представления чисел с многократной точностью используется соответствующее число ячеек памяти, в каждой из которых хранится одно слово. Для выборки таких чисел из памяти необходимо задать адрес

младшего слова, а остальные слова выбираются путем увеличения адреса текущей ячейки памяти на 1.

В некоторых микропроцессорах заложена возможность обработки чисел, представленных в двоично-десятичной системе счисления. Возможность выполнить арифметическую обработку десятичных чисел позволяет избежать потерь при обратном и прямом преобразовании чисел из одной системы счисления в другую и исключить потери в точности представления исходных чисел, которые могут иметь место при переходе от одной системы счисления к другой. Например, десятичное число 0,3 нельзя представить абсолютно точно в двоичной системе счисления. Поэтому для представления десятичных чисел используют специальные двоично-десятичные коды с весами 8-4-2-1. Для хранения много разрядных десятичных чисел используется последовательно запоминание в ячейках памяти цифр десятичного числа. Для сложения и вычитания чисел в десятичной системе счисления в микропроцессоре используется не одна, а две команды: команда двоичного сложения и команда десятичной коррекции. Например, кодирование знакового разряда S производится следующим образом:

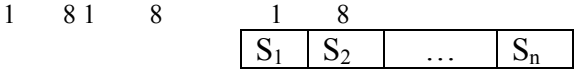
$$S \begin{cases} 0000 - \text{для положительных чисел} \\ 1001 - \text{для отрицательных чисел} \end{cases}$$

Сложим два числа +52 и -43:

```
  0000,0101 0010  (52)
  1001,0101 0111  (-43)
  1001,1010,1001  двоичное сложение
  0110,0110       десятичная коррекция
  0000,0000,1001  (9)
```

Если тетрада содержит число больше 9, то прибавляется число 6.

Значительная доля информации, обрабатываемая микропроцессором, является текстовой, которая представлена строками символов некоторого алфавита. Отдельный символ (буква, цифра, знак и т.д.) кодируется 7 и 8-битными кодами. Строка символов при 8-битном кодировании представляет собой последовательность байт следующего вида:



S – 8-битный двоичный код символа. Таким образом, строка символов представляется полем переменной длины. Обычно длина поля может изменяться от 1 до 256 байт.

Для кодирования символов используются специальные коды, наиболее распространенными из которых являются 7- и 8-битные двоичные коды.

Логические значения являются булевыми переменными принимающие одно из двух значений ‘ложь’ или ‘истина’ и кодируется цифрами 0 и 1 соответственно. Действия над ними можно производить только с помощью логических операций и могут являться элементами наборов – булевых векторов и матриц.

Для записи и индикации адресов и данных применяются удобные и компактные восьмеричная и шестнадцатеричная системы счисления.

2.2.2. Общие сведения о системе команд

Под командой понимают совокупность сведений, необходимых процессору для выполнения определенного действия при реализации программы.

Множество команд, реализуемых в МП, образуют систему команд. Выбор системы команд является сложнейшей и важной задачей проектирования микропроцессора, так как система команд определяет область и эффективность ее применения.

Теоретически ограничения на число команд ЭВМ нет, например, при введении команд из нескольких слов можно выделить больше бит под код операции. Однако, как

показывает практика программирования, при реализации обширной системы команд, программисты обычно начинают использовать только некоторое подмножество.

Чем сложнее команда, тем более вероятно, что ее действия можно интерпретировать группой команд, а это уменьшает эффективность всей системы команд. Но с другой стороны, чем сложнее команда, тем более быстро выполняется программа из-за сокращения обращений к памяти.

Теоретически показано, что минимальная алгоритмически полная система команд процессора состоит из одной или нескольких универсальных команд. Существует два подхода к созданию эффективной системы команд.

Первый подход был заложен в разработку МП с сокращенным набором команд, на основе которых разрабатываются RISC-компьютеры (Reduced Instruction Set Computer). С одной стороны RISC-процессоры, имеющие более простую внутреннюю архитектуру, за счет меньшего числа запрограммированных инструкций (команд), более дешевы в изготовлении, более надежны и отличаются высокой производительностью. С другой стороны, перенесение бремени обработки информации в RISC-процессорах на программное обеспечение приводит к снижению быстродействия за счет увеличения длины программ, особенно за счет существенного усложнения трансляторов с языков высокого уровня.

Второй подход базируется на МП с расширенным набором команд - CISC-процессорах (Complexed Instruction Set Computer). В эти микропроцессоры встраиваются дополнительные аппаратные средства, позволяющие реализовать многие десятки и сотни команд. Такие развитые системы команд дают возможность обеспечить компактную запись алгоритмов и соответственно эффективность программы. С другой стороны, с введением в МП традиционной структурой большего числа регистров и большего объема кэш-памяти (МП 68040, i80486) их быстродействие приближается к быстродействию МП RISC-архитектуры.

Практически во всех современных микропроцессорных системах используются сложные развитые системы команд. Их ядро, состоящее из набора универсальных команд, реализуются аппаратным способом в центральном микропроцессоре. Кроме того, специализированные части набора системы команд реализуются вспомогательными или периферийными микропроцессорами. Эти расширяющие возможности обработки данных в специальных арифметических или логических микропроцессорах позволяют ускорить выполнение определенных команд и тем самым сократить время исполнения программ.

2.2.3. Форматы команд

Важной характеристикой команды является ее формат, определяющий структурные элементы команды, каждый из которых несет определенную функциональную нагрузку. Среди таких элементов команды выделяют следующие:

1. Код операции, являющий ее основным элементом, определяет действие (операцию) самой команды (КОП).
2. Источник данных (операнд). Этот элемент в общем случае содержится в поле адреса. Для одноместных операций (например, сдвиг) требуется один операнд, а для двухместных операций (сложение) - два операнда.
3. Приемник данных, определяющий место назначения результата выполнения операции. Этот элемент команды также содержится в поле адреса.
4. Источник следующей команды (адрес следующей команды).

Очевидно, что команда, которая содержит всю эту информацию (рис.2.1) будет очень длинной.

Код операции	Адрес операнда 1	Адрес операнда 2	Адрес результата	Адрес следующей команды

Рис.2.1. Формат команды

Например, если бы поле кода операции составила 4 бита (можно закодировать только 16 различных операций) и каждый из адресов составлял 12 бит (что позволяет адресовать 4 Кбайт памяти), то общая длина команды составила бы 40 бит. Ясно, что с такой командой трудно было бы работать микропроцессору, который оперирует с 8-ми или 16-битными словами. Таким образом, часть информации, в которой нуждается микропроцессор, должна быть задана неявно и не должна зависеть от особенностей конкретной команды.

Существует множество методов уменьшения форматов команд микропроцессора, среди которых наиболее часто употребляемые следующие:

1. Использование программного счетчика, содержащего адрес команды. МП увеличивает содержимое программного счетчика после каждого обращения к памяти программ и таким образом вызывает из программной памяти следующую команду с очередным, более старшим адресом.
2. Использование адреса одного из источников для записи результата.
3. Использование адресов источника и места назначения информации неявным образом. Эти неявно выраженные адреса могут быть записаны в регистры или в ячейки, адресуемые через регистры.
4. Ограничение адресами регистров, вместо использования полных адресов ячеек памяти.

Введение программного счетчика представляет собой метод выборки последующей команды. С одной стороны, представляющий собой эффективный механизм реализации последовательных структур алгоритма, при этом отпадает необходимость специфицировать адрес следующей команды, что приводит к сокращению формата команды. С другой стороны, возникает необходимость использования дополнительных команд, реализующие переход, переход по условию, пропуск и останов.

В микропроцессорах с ограниченной разрядностью широкое применение получил и второй метод уменьшения формата команды введением специального регистра - аккумулятора в качестве источника и места назначения информации. Типичная команда ADD B, означает, что требуется сложить содержимое регистра B с содержимым аккумулятора и результат поместить в аккумулятор. Такие команды называются одноадресными, форматы которых приведены на рис.2.2, очевидно, что такие команды могут быть короткими.

КОП	Адрес операнда

Рис.2.2. Формат одноадресной команды

Однако, программы, содержащие одноадресные команды, требуют дополнительных команд для разгрузки данных в аккумулятор, а затем записи результатов в память или в регистры общего назначения (РОН).

Использование неявной адресации определяет собой один из самых гибких методов адресации, позволяющих сократить формат команды - это метод косвенной адресации, который находит самое широкое распространение в микропроцессорах. Так, как неявно выраженные адреса могут быть записаны в регистры или в ячейки памяти, адресуемые через эти регистры. Этот метод требует введения в структуру микропроцессора специальных регистров, которые образуют внутреннюю память МП, и называются регистрами общего назначения (РОН).

Значительное уменьшение формата команды достигается при использовании адресов внутренних регистров. При этом используются короткие адреса РОН, вместо адресов ячеек памяти, и потребуются дополнительные команды для загрузки этих регистров и запоминания их содержимого. Отметим, что все эти методы уменьшают формат команд, усложняют процесс программирования.

Классификация команд

Классификация команд по важнейшим признакам представлена на рисунке 2.3.

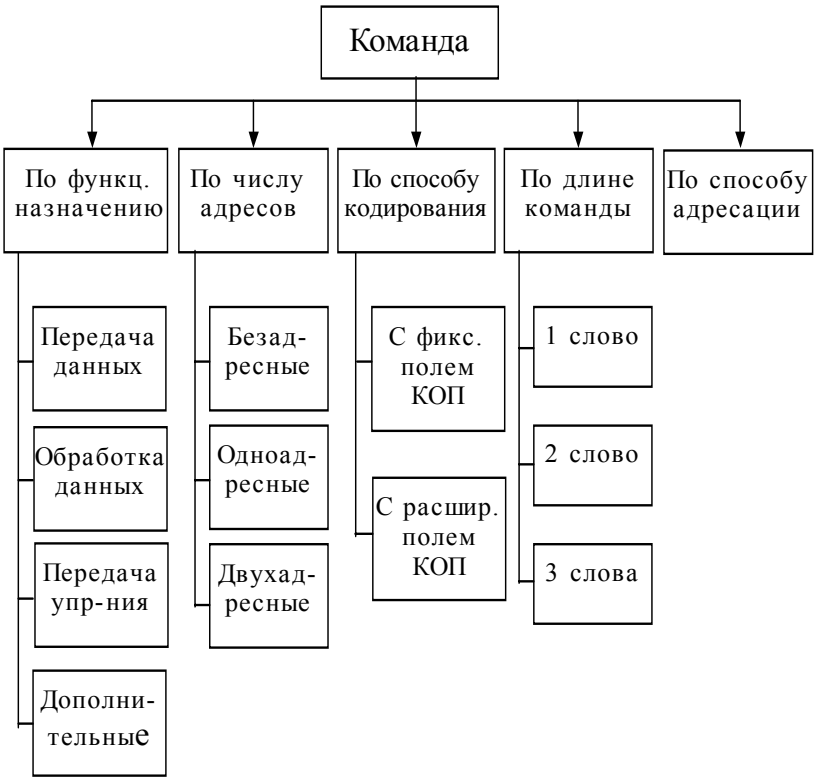


Рис.2.3. Классификация команд

Команды передачи данных пересылают данные из одной части ЭВМ в другую без изменения этих данных и включают в себя следующие команды: передачи кодов внутри процессора, из регистров процессора в память, из памяти в регистры процессора и между регистрами процессора и периферийными устройствами. В процессорах, допускающих различную длину данных, например, байты, слова, двойные слова, для каждой длины имеются отдельные команды передачи данных.

Команды обработки данных выполняют операции преобразования данных. В основном они используются в арифметико-логическом устройстве МП и включают в себя следующие команды: арифметические и логические операции, операции сдвига и сравнения. По числу используемых операндов их можно классифицировать на одно- и двухоперандные операции. Данные двухоперандных команд хранятся в регистрах процессора или в ячейке памяти, а при непосредственной адресации один из операндов указывается в самой команде. Результат формируется в регистре-приемнике или в аккумуляторе. Команды данной группы формируют признаки результатов: перенос из старшего разряда, переполнение результата, межтетрадный перенос и др.

Команды передачи управления используются для изменения естественного порядка следования команд, путем изменения содержимого регистра команд, и организации циклических участков в программе. Они включают в себя следующие команды: безусловного перехода; условного перехода; оперирующие с программами. Дополнительные (специальные) команды изменяют значения признаков МП без изменения данных или порядка, в котором выполняются команды. Это команды останов, отсутствие операции, ожидание и др. Таким образом, эти команды скорее влияют на функционирования, чем на функции обработки данных. Они составляют незначительную часть большинства программ.

Основным структурным элементом команды является код операции (КОП),

определяющее действие команды. При использовании фиксированного числа бит под код операции для кодирования всех m команд необходимо в поле кода операции выделить $\log_2 m$ двоичных разрядов. Но, учитывая ограниченную длину слова, в микропроцессорах, функционально различное назначение команд, источники и приемники результатов операций, а также что не все команды содержат адресную часть для обращения к памяти и периферийным устройствам для кодирования команд широко используются принципы кодирования с переменным числом бит под поле КОП для различных групп команд.

Другой важной характеристикой команды является адресность, определяемая количеством адресов в команде. В микропроцессорах ввиду ограничения длины слова команды подразделяется на безадресные, одноадресные и двухадресные, которые могут быть представлены в виде одного, двух и трех машинных слов.

2.2.4. Режимы адресации

При выполнении программы многие команды требуют обращения к памяти для выборки данных, записи промежуточных и окончательных результатов вычислений. Поэтому механизм адресации в значительной мере определяет способность МП эффективно осуществлять обработку информации. Для микропроцессоров это является особенно важным из-за ограниченной длины их команд, что часто является причиной использования команд в два или три слова. Поэтому для преодоления данного ограничения имеется множество режимов адресации, которые позволяют:

1) определить полный адрес памяти меньшим числом бит, тем самым ,сокращая длину команд;

2) обращаться к ячейкам памяти, адреса которых вычисляются во время обработки, что обеспечивает удобный доступ к данным различной структуры;

3) вычислять адреса данных относительно позиции команды таким образом, что программу можно загружать в любую область памяти без всяких изменений адресов в программе.

Способ определения операнда называется режимом адресации. Все режимы адресации можно разделить на две группы. К первой группе относятся режимы, в которых исполнительный адрес определяется одним значением кода в команде. Режимы адресации второй группы используют содержимое адресной части команды и одного или нескольких регистров для формирования исполнительного адреса.

К первой группе основных режимов адресации относятся прямая, регистровая, непосредственная, регистровая косвенная, автоинкрементная, автодекрементная. Ко второй группе относятся страничная, индексная, относительная адресации.

Прямая адресация. В данном режиме адресации код адреса в команде является исполнительным адресом обращения к памяти. Однако указание полного прямого адреса требует много бит в команде ЭВМ с большим адресным пространством, и для уменьшения его некоторые ЭВМ используют короткую прямую адресацию, обеспечивая доступ к ограниченной части адресного пространства. Из-за своей простоты режим прямой адресации имеется во многих малых ЭВМ. Приведем пример команды, реализующей данный метод. Эта команда и последующие примеры команд взяты из системы команд МП КР580. Команда LDA addr является трехбайтной, загружает содержимое ячейки памяти с адресом addr в аккумулятор. Цикл выполнения этой команды состоит из трех машинных циклов: выборка кода операции, чтение младшего байта адреса и загрузка содержимого ячейки памяти в аккумулятор. Таким образом, для прямой адресации требуются дополнительные циклы обращения к памяти, и не хватает гибкости для обработки массивов данных. Команды, использующие этот метод адресации, обеспечивают доступ только к отдельным ячейкам памяти.

Регистровая адресация. В данном режиме адресации операнды содержатся в одних из регистров МП. Команда MOV B, C является однобайтной. Она передает содержимое

регистра C в регистр B и выполняется за один машинный цикл. Этот метод адресации обеспечивает короткий формат команды и является самым быстрым методом адресации, пригодный для большинства МП. Недостатком этого метода является то, что для загрузки регистров и сохранения его содержимого в памяти требуются дополнительные команды.

Непосредственная адресация. При непосредственной адресации операнд является частью команды и передается в МП из памяти вслед за кодом операции. Команда MOV B, 1Fh является двухбайтной, и число 1F непосредственно загружается в регистр B. Эта команда выполняется за два машинных цикла. Команды с непосредственной адресацией используются для инициализации регистров общего назначения, загрузки косвенных адресов и для введения констант, необходимых при вычислениях. Эти команды не обладают соответствующей гибкостью для обработки массивов данных, в них фиксированы и адрес, и данные.

Регистровая косвенная адресация. Получила широкое распространение в малых ЭВМ и по быстродействию приближается к прямой адресации, так как косвенный адрес извлекается из внутреннего регистра процессора и не требуется дополнительного цикла памяти. В регистровой косвенной адресации регистр или регистровая пара содержит исполнительный адрес операнда. Загрузка регистров осуществляется с использованием команд непосредственной адресации.

Команда MOV M,r (или MOV r,M) является однобайтной, передает содержимое регистра r в ячейку памяти (или наоборот) адресуемой H,L парой, и выполняется за два машинных цикла.

Использование этого режима позволяет вычислять адреса памяти во время выполнения программы, что требуется в процедурах передачи данных, при просмотре элементов массива и т.п.

Автоинкрементная и автодекрементная адресация. Вычисляет дополнительный адрес практически также как и при регистровой косвенной адресации, а затем производится увеличение содержимого регистра. В ЭВМ с побайтной адресацией содержимое регистра увеличивается на 1, а с 2-х-байтной адресацией (слова) на 2. При этом размер операнда определяется КОП.

В автодекрементном режиме адресации адрес операнда формируется вычитанием 1 или 2 из регистра адреса. Отличие от автоинкрементной адресации состоит в том, что вычитание производится до использования содержимого регистра как исполнительного адреса. Этот режим адресации обеспечивает эффективное использование любого регистра в качестве указателя стека.

Страничная адресация. При использовании страничного режима адресации память разбивается на ряд страниц одинаковой длины. Адресация страниц осуществляется или с программного счетчика или с отдельного регистра страниц, а адресация ячеек памяти внутри страниц - адресом, содержащимся в команде. При этом обычно адрес страницы формируется одним из следующих способов:

1) номер страницы располагается в базовой (нулевой) странице;
2) номер страницы данных формируется путем пристыковки (конкатенации) старших разрядов программного счетчика с адресом находящимся в команде, что позволяет получать адрес той же страницы, в которой находится команда;
3) номер страницы берется из регистра страниц, в который программа ранее загрузила номер нужной страницы.

Индексная адресация. Она удобна для обращения к массивам и таблицам. Для образования исполнительного адреса к адресной части команды прибавляется смещение из индексного регистра, называемого индексом. Когда индексный режим используется для доступа к массиву, адрес в команде соответствует базовому адресу массива, а значение индексного регистра - индексу компоненты массива.

Относительная адресация. При использовании относительной адресации исполнительный адрес формируется путем сложения базового адреса с адресным полем команды. В качестве базового адреса используется содержимое программного счетчика.

Использование относительной адресации позволяет строить программы, свободно перемещаемые в памяти за счет того, что в команде указано смещение по отношению к содержимому программного счетчика. Так как содержимое программного счетчика постоянно увеличивается, формируемые исполнительные адреса изменяются для одного и того же смещения, что характерно для динамического способа адресации.

В заключение следует отметить, что единства в вопросах важности различных режимов адресации нет. Различие в их выборе, по-видимому, объясняется тем, что разработчики МП ориентируют их на определенные области применения.

2.3. Классификация микропроцессоров

На рис. 2.4 приведена классификация микропроцессоров по важнейшим признакам.

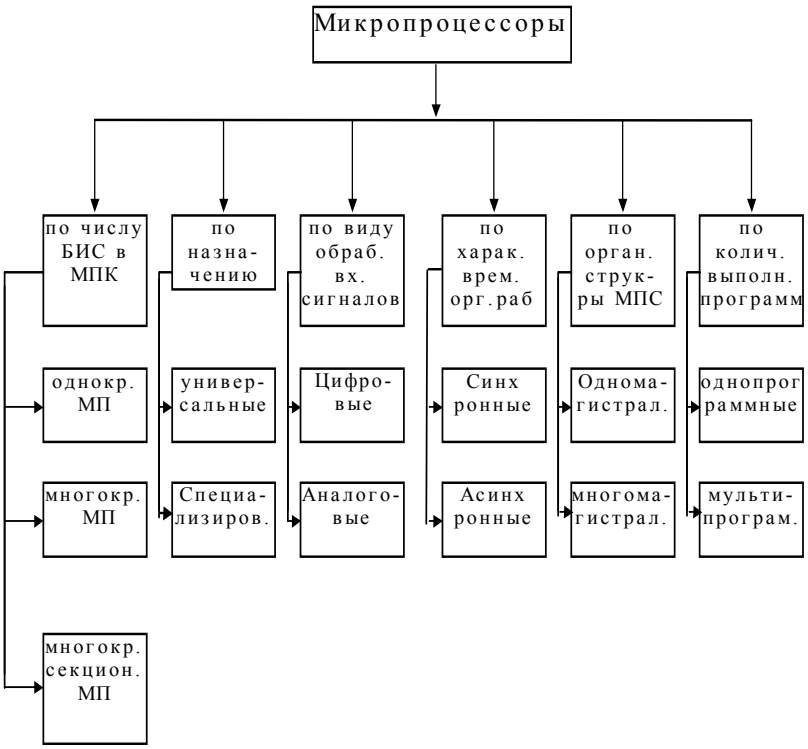


Рис.2.4. Классификация микропроцессоров

Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (ОМП).

По мере увеличения степени интеграции элементов в кристалле и числа выводов корпуса параметры ОМП улучшаются. Однако возможности ОМП ограничены аппаратурными ресурсами кристалла и корпуса.

Многокристальные микропроцессоры получаются в результате разбиения его логической структуры на функционально законченные части, которые реализованы в виде БИС или СБИС.

Функциональное разбиение структуры процессора при создании трех кристального микропроцессора показано на рис.2.5.

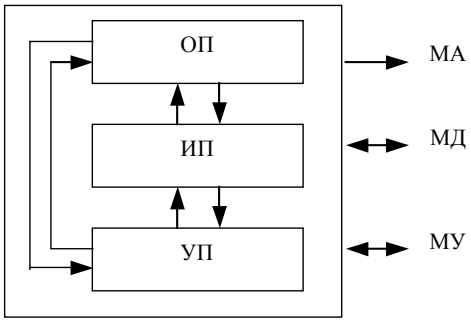


Рис.2.5. Функциональные разбиение структуры процессора при создании трехкристального микропроцессора

Операционный процессор (ОП) служит для обработки данных. Управляющий процессор (УП) выполняет функции выборки, декодирования и вычисления адресов операндов, а также генерирует последовательности микрокоманд.

Автономность работы и большое быстродействие БИС УП позволяет выбирать команды из памяти с большой скоростью, чем скорость их выполнения на ОП. При этом в УП образуется очередь еще невыполненных команд, а также заранее подготавливаются те данные, которые потребуются в следующих циклах работы ОП. Такая опережающая выборка команд экономит время ОП на ожидание операндов, необходимых для команд программы.

Интерфейсный процессор (ИП) позволяет подключить память и ПУ к микропроцессору. По существу является сложным контроллером для устройств ввода-вывода информации. БИС ИП выполняет также функции канала прямого доступа к памяти.

Выбираемые из памяти команды распознаются и выполняются каждой частью МП автономно, и поэтому может быть обеспечен режим одновременной работы всех БИС МП, т.е. конвейерный поточный режим исполнения последовательности команд программы (выполнение последовательности с небольшим временным сдвигом). Такой режим значительно повышает производительность МП.

Многокристальные секционные микропроцессоры получаются, когда функционально законченные части логической структуры микропроцессора (горизонтальное разбиение) разбиваются на секции малой разрядности, и эти секции реализованы в виде БИС (вертикальное разбиение) как показано на рис.2.6.

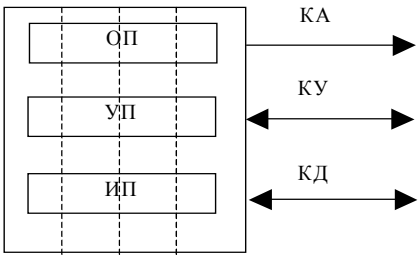


Рис.2.6. Реализация МП в виде комплекта секционных БИС

Многокристальные секционные микропроцессоры имеют разрядность от 2-4 до 8-16 бит и позволяют создавать разнообразные высокопроизводительные процессоры ЭВМ. Ему присущи все свойства многокристальных МП.

Универсальные микропроцессоры. Это такие микропроцессоры, в системе команд которых заложена алгоритмическая универсальность.

Это означает, что выполняемый машиной состав команд позволяет получить преобразование информации в соответствии с любым заданным алгоритмом. Такие МП используются для решения широкого круга разнообразных задач. При этом эффективная производительность МП слабо зависит от проблемной специфики решаемых задач.

Специализированные микропроцессоры предназначены для решения определенного класса задач, а иногда только для решения одной конкретной задачи. В системе команд таких МП заложен ограниченный круг команд. Это позволяет обеспечить простоту управления, компактность аппаратных средств, низкую стоимость и малую мощность потребления. Среди них можно выделить три наиболее распространенных класса устройств.

МП для обработки данных предназначены для обслуживания баз данных, используемых в различных областях деятельности.

Математические МП предназначенные для повышения производительности при выполнении арифметических операций за счет, например, матричных методов обработки.

В составе специализированных МП выделяется особый класс устройств, которые называются микроконтроллерами. Микроконтроллеры ориентированы на выполнение сложных последовательностей логических операций, предназначенные для задач управления.

Аналоговые микропроцессоры. Сами микропроцессоры - цифровые устройства, однако, они могут иметь встроенные аналого-цифровые преобразователи (АЦП) и цифро-аналоговые преобразователи (ЦАП). Поэтому входные аналоговые сигналы передаются в МП через АЦП, преобразуются в цифровую форму, обрабатываются и после обратного преобразования в аналоговую форму ЦАП поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой функциональные аналоговые преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование, декодирование сигналов в реальном масштабе времени и т.д.).

Отличительной чертой аналоговых МП является способность к переработке большого объема числовых данных, т.е. к выполнению операций сложения и умножения с большой скоростью, при необходимости отказа от операций прерываний и переходов.

Синхронные микропроцессоры - МП, в которых начало и конец выполнения операций задаются устройством управления.

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции.

В **одномагистральных ЭВМ** все устройства имеют одинаковый интерфейс и подключен к единой информационной магистрали (магистралям данных, адресов и управления).

В **многомагистральных микроЭВМ** устройства группами подключаются к своей информационной магистрали. Это позволяет передавать одновременно информацию по нескольким магистралям. Такая организация усложняет их конструкцию, однако, увеличивает производительность.

В **однопрограммных микропроцессорах** выполняется только одна программа. Переход к выполнению другой программы происходит после завершения текущей программы.

В **мультипрограммных микропроцессорах** выполняется несколько программ (обычно несколько десятков). Организация мультипрограммной работы микропроцессорных управляющих систем позволяет осуществлять контроль за состоянием и управлением большого числа источников или приемников информации.

2.4. Структурная организация микропроцессоров

Микропроцессор – это устройство, предназначенное для обработки данных. Он вырабатывает временные сигналы и сигналы управления, передает данные в память и из памяти и устройств ввода-вывода, выполняет арифметические и логические операции и

идентифицирует внешние сигналы. На рисунке 2.7 показана структурная организация типового микропроцессора.

По функциональному назначению микропроцессор относится к классу операционных устройств, и, согласно модели Глушкова, в его структуре можно выделить две части: управляющий блок (устройство управления) и операционный блок (см. рис.2.7).

Процесс функционирования МП разбивается на командные циклы. В течение каждого цикла команды МП выполняет ряд управляющих функций:

1) помещает адрес команды в адресную шину памяти;

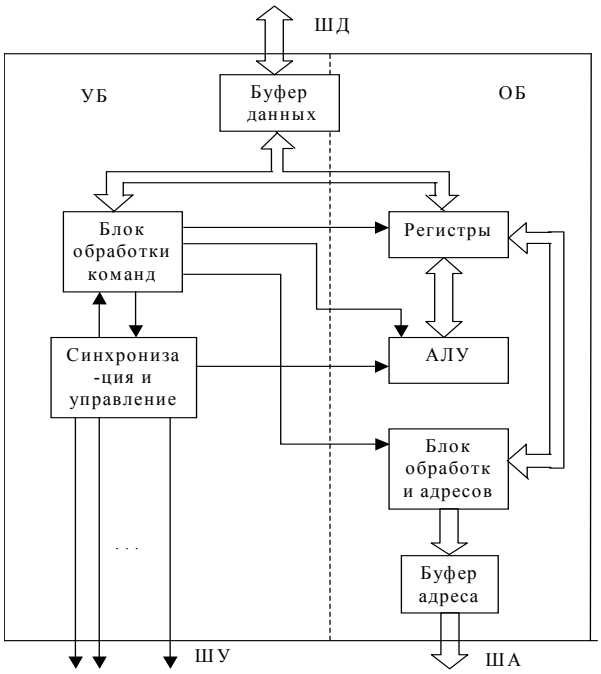


Рис.2.7. Структурная организация типового МП

2) получает команду из шины ввода данных и дешифрирует ее;

3) выбирает адреса и данные, содержащиеся в команде. Адреса и данные могут находиться в памяти или в регистрах;

4) выполняет операцию, определенную в коде команды. Операцией может быть арифметическая или логическая функция, передача данных или функция управления;

5) сохраняет результат выполнения операции в памяти или внутренних регистрах;

6) следит за управляющими сигналами или сигналами, такими как прерывание, и реагирует соответствующим образом;

7) генерирует сигналы состояния, управления и времени, которые необходимы для нормальной работы устройств ввода-вывода и памяти.

2.4.1. Операционные блоки микропроцессора

На рис.2.8. представлена структура операционной части МП, на которой показан типовой набор регистров.

Основу операционной части МП образуют рабочие регистры. Регистры представляют собой сверхоперативное ЗУ небольшой емкости. Регистры состоят из триггеров и адресуются подобно ячейкам памяти. Данные в регистрах могут храниться до тех пор, пока шина или некоторый блок не будут готовы принять их, или пока они не потребуются по программе.

Регистры, содержимое которых не изменяется под воздействием программы, позволяют сохранять данные для последующего использования. С помощью внутренних шин регистры связаны друг с другом, а с другими блоками системы связь осуществляется под управлением программы. Ограничение числа регистров микропроцессора связано со

сложностью их изготовления на кристалле и большим числом внутренних соединений. С другой стороны, наличие большого числа внутри процессорных регистров приводит к расширению возможностей дешифрирования и адресации команд и данных и уменьшению числа операций обращения к памяти и формата команд.

Регистры могут выполнять различные функции. В некоторых МП программист может присваивать регистрам разнообразные частные функции, однако, в большинстве МП имеются несколько основных регистров: счетчик команд, регистр команд, регистр адреса памяти, указатель стека, аккумулятор, регистры общего назначения (РОН), регистр кода условий.

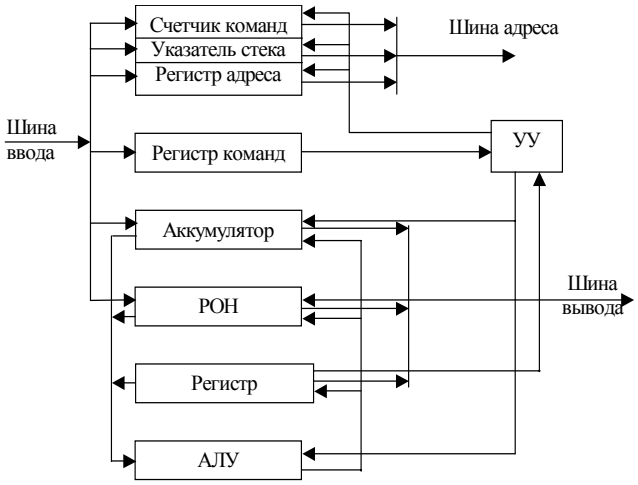


Рис.2.8. Структура операционной части МП

Счетчик команд (СК) содержит адрес ячейки памяти в которой хранится очередная команда. Начальный адрес СК загружается с пульта управления ЭВМ или СК после начальной установки переходит в нулевое состояние, и выполнение программы начинается с нулевой ячейки программной памяти. В эту ячейку можно поместить команду безусловной передачи управления любому адресу программной памяти. Цикл выполнения команды начинается с того, что МП посылает содержимое счетчика команд в шину адреса, по которому извлекается из памяти первое слово команды, содержащего код операции. Далее в СК путем увеличения его содержимого формируется адрес следующего слова выбираемой команды или адрес следующей команды. Данная процедура характерна для линейной последовательности естественного порядка команд. При выполнении команд условного и безусловного переходов, обращения к подпрограммам адрес следующей команды в СК формируется путем загрузки в него адресной части вышеуказанных команд. Код адреса из СК передается в программную память через однонаправленную шину адреса.

Регистр команд (РК) принимает выбранную из памяти программ команду (один из важных элементов команды – код операции) и хранит ее в течение цикла выполнения команды. Содержимое РК является исходной информацией для блока обработки команд. Загрузка РК осуществляется по двунаправленной шине данных.

Регистр адреса памяти содержит адрес данных. Через этот регистр происходит обращение к данным, расположенным в памяти.

Указатель стека (УС) хранит адрес последней занятой ячейки в области стековой памяти. Под стековой памятью понимают оперативную память, в которую запись и выборка слова производится по принципу: последний записанный элемент выбирается из памяти первым. Адрес ячейки стековой памяти, по которому выполняется запись или чтение слова, называется вершиной стека. Начальный адрес стека устанавливается в указателе стека программным путем. Некоторые МП используют внутреннюю стековую память команд для запоминания только адресов команд для возврата из подпрограмм. В этом случае, из-за ограниченной емкости такой памяти в ней нельзя запоминать содержимое регистров

прерываемой программы, слово-состояние программы и другую информацию при переключении программ. Поэтому подавляющее большинство МП используют внешнюю стековую память, емкость которой для 16-битного указателя стека может составлять до 16 Кслов. Адресация стековой памяти также осуществляется через шину адреса МП.

Таким образом, счетчик команд, регистр адреса данных и указатель стека являются основными регистрами блока обработки адреса, участвующие в формировании адресов к ячейкам системной памяти.

Аккумулятор (А) используется для хранения одного из операндов и результата операции, т.е. является источником и приемником информации. Команды ввода и вывода данных обычно осуществляют обмен кодов с периферийными устройствами также через аккумулятор. Некоторые МП содержат несколько аккумуляторов.

Регистры общего назначения (РОН) нашли широкое использование в большинстве МП и выполняют различные функции. РОН используются для внутреннего хранения исходных операндов при выполнении бинарных логических и арифметических операций, промежуточных результатов вычислений, иногда в РОН хранятся индексы, если в МП отсутствуют специальные индексные регистры. Это позволяет повысить быстродействие ЭВМ за счет сокращения пересылок кодов между МП и памятью. Регистры общего назначения являются программно-доступными, и обращение к ним осуществляется посредством команд передачи данных. Число РОН для большинства МП составляет 8 – 32, то для их адресации достаточно в поле адреса команд 3 –5 битов соответственно, это приводит к сокращению формата команд, использующих регистровый метод адресации.

Регистр кода условий или регистр состояния содержит набор одноразрядных признаков, которые отображают состояние МП или несколько внешних входов или выходов. Эти признаки – основа для работы микропроцессора, осуществляющего принятие решений на их базе. Различные МП имеют различное число и назначение признаков. Наиболее распространенными являются следующие признаки:

- **перенос (CF)** – 1, если при выполнении операции сложения (вычитания) в старшем разряде слова образуется признак переноса (заёма). Этот признак может сохранять свое значение или может принимать участие в переносе от одного слова к другому в арифметических операциях повышенной точности или при сдвиге операнда;
- **вспомогательный перенос (AF)** - 1, если в результате операции возник признак переноса из младшей тетрады в старшую тетраду, что используется для выполнения арифметических операций с двоично-кодированными десятичными числами;
- **переполнение (OF)** – 1, если операция выдает дополнительный код с переполнением. Этот признак сигнализирует о потере старшего бита результата сложения или вычитания, иначе он сообщает о превышении результата арифметической операции разрядности МП;
- **знак (SF) –** 1, если старший значащий бит результата операции был 1. Этот признак используется в арифметических операциях и при использовании дополнительного кода соответствует знаку числа;
- **четность (PF) –** 1, если в результате операции число единиц в слове было нечетным (нечетный паритет) или четным (четный паритет). Этот признак предназначен для контроля правильности передач данных;
- **нуль (ZF) –** 1, если результат операции равен нулю. Это часто используется в управлении циклом и в процессе поиска некоторого адресуемого числа.

Арифметико-логическое устройство

Арифметико-логическое устройство (АЛУ) предназначено для арифметической и логической обработки данных, выполнения операций сдвига, формирования признаков результатов операций.

В микропроцессорах наибольшее распространение получили АЛУ, где основным типом данных являются целые числа, характеризующиеся простотой реализации и высокой

скоростью обработки информации. Такие АЛУ представляют собой быстродействующие блоки целочисленной обработки информации. Специализированные БИСы используются для обработки данных, представленных в виде чисел с плавающей запятой.

АЛУ строят на основе двоичного сумматора и вспомогательных логических схем. Типовая схема АЛУ показана на рис.2.9.

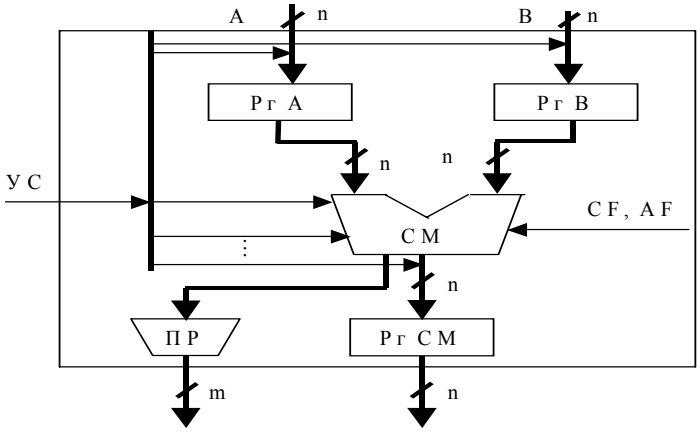


Рис.2.9. Арифметико-логическое устройство

АЛУ состоит из двоичного двухвходового комбинационного сумматора (СМ) разрядностью n, определяющего длину слова МП. Двух входных регистров РгА и РгВ, осуществляющих прием данных с входных магистралей под управлением микроопераций приема слов. Причем, если МП имеет одну внутреннюю магистраль, то второй операнд будет находиться в одном из внутренних регистров МП, и схема АЛУ будет содержать один из регистров А или В. Выходного регистра РгСМ для сохранения результата операции. Роль РгСМ в большинстве МП выполняет аккумулятор. Комбинационной схемы ПР, формирующего m признаков, описанных в этом параграфе. Кроме того, АЛУ имеет входы признаков переноса CF, AF для выполнения операций умножения и деления, а также функциональные входы УС.

Функциональные входы представляют собой множество управляющих слов (микроопераций), вырабатываемые устройством управления, и определяют, какую операцию выполняет АЛУ.

Типичные операции: сложение, вычитание, логическое И, логическое ИЛИ, логическое ИСКЛЮЧАЮЩЕЕ ИЛИ, логическое НЕ, инкрементирование (увеличение на 1), декрементирование (уменьшение на 1), сдвиговые операции, обнуление (результат равен нулю). Функциональные входы также определяют режимы работы шины.

При помощи специальных дополнительных схем могут решаться другие арифметические задачи, такие как умножение и деление, нахождение синуса и косинуса, определение логарифмов или экспонент. Эти специальные схемы способны обеспечивать высокое быстродействие, но арифметические блоки, которые могут непосредственно решать специализированные задачи, стоят гораздо дороже, чем стандартные целочисленные блоки АЛУ.

Таким образом, АЛУ может решать любую из разнообразных задач в течение одного цикла под управлением сигналов на функциональных входах.

2.4.2. Устройство управления микропроцессора

Выполнение любой команды в МП можно разбить на два цикла: цикла выборки команды и цикла выполнения команд.

В цикле выборки команды устройство управления (УУ) производит чтение кода операции и его декодирование. В цикле выполнения команды в соответствие с типом

реализуемой команды осуществляется определение адресов операндов, участвующих в операции, непосредственно выполнение команды и сохранение результатов операции.

Любая команда, выполняемая операционным блоком, описывается некоторой микропрограммой и реализуется за определенное количество тактов, в каждом из которых выполняется одна или несколько микроопераций. При выполнении микропрограммы на соответствующие управляющие шины операционного блока подается определенным образом распределенная во времени последовательность управляющих функциональных сигналов (микроопераций). Порядок выполнения микроопераций может изменяться в зависимости от признаков операции, вырабатываемых в АЛУ и являющихся входными сигналами УУ.

Таким образом, устройство управления формирует распределенную во времени и пространстве последовательность внешних и внутренних управляющих сигналов (УС), обеспечивающих выборку и выполнение команды.

Одной из важнейших характеристик УУ является возможность изменения последовательности управляющих слов (микроопераций). По этому критерию УУ подразделяются на УУ с «жесткой» логикой, или специализированные УУ, и на универсальные или микропрограммные УУ. На рис.2.10 представлена структура УУ с «жесткой» логикой.

Входной информацией для УУ является содержимое регистра команд, определяющее тип выполняемой команды (КОП), и признаки операций, вырабатываемые АЛУ.

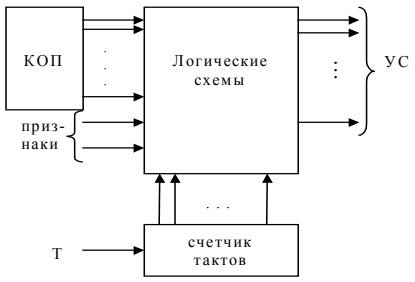


Рис.2.10. Устройство управления с «жесткой» логикой

Выходная информация представляет собой совокупность управляющих сигналов, вырабатываемых УУ в соответствии с заданной микропрограммой. Интервал времени, отводимый на выполнение микрооперации, называется рабочим тактом или тактом процессора. Длительность такта устанавливается по самой продолжительной микрооперации. Синхронизация УУ осуществляется с помощью счетчика тактов, управляемого внешним генератором тактовых импульсов Т.

В состав УУ входят запоминающие и комбинационные схемы, выполняющие функции запоминания текущего состояния, определяющие совокупность УС, и формирования следующего состояния в соответствии с входными признаками. Микропрограмма в таком УУ задается взаимосвязями между элементами логических схем, основу которых составляют счетчики, регистры, дешифраторы. Изменение микропрограммы в УУ приводит к заданию новых взаимосвязей между элементами, что равносильно проектированию новой логической схемы УУ. Таким образом, основным свойством УУ с «жесткой» логикой является фиксированный набор системы команд, который задан на этапе проектирования. УУ с «жесткой» логикой используются в специализированных и однокристальных МП.

Структурная схема микропрограммного УУ (МУУ) – устройство управления с хранимой в памяти логикой показана на рис.2.11.

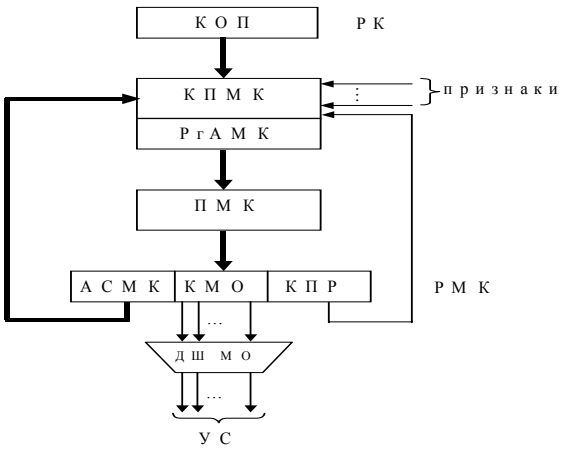


Рис.2.11. Микропрограммное устройство управления

В состав устройства входят контроллер последовательности микрокоманд (КПМК), регистр адреса микрокоманд (РАМК), память микрокоманд, регистр микрокоманды (РМК) и дешифратор микроопераций (ДШ МО).

Основным назначением КПМК является реализация управляющих структур, встречающихся в микропрограммах: линейной последовательности, структуры вида «если P, то X, иначе Y» и структуры вида «пока P, делай X». При этом контроллер реализует следующие функции:

- производит дешифрацию кода операции команды (КОП) для обращения к первой микрокоманде микропрограммы, интерпретирующей данную команду;
- формирует адреса следующих микрокоманд по указанным выше трем управляющим структурам;
- сохраняет признаки переходов, поступающие из операционного блока и формируемые при выполнении микрокоманд условного перехода;
- осуществляет управление прерываниями на микропрограммном уровне.

Память микрокоманд предназначена для хранения микрокоманд, ее емкость и разрядность однозначно определяются набором реализуемых микропрограмм. Путем изменения набора микропрограмм можно гибко менять систему команд микропроцессора и тем самым ориентировать его функциональную направленность.

Регистр микрокоманды предназначен для хранения микрокоманды при выполнении текущего микрокомандного цикла. Микрокоманда содержит три основных поля: код микрооперации КМО, адрес следующей микрокоманды АСМК, поле кода признаков КПР, в котором указывается, какой признак разветвления в микропрограмме необходимо анализировать КПМК.

Дешифратор микроопераций служит для декодирования кода микрооперации и формирования управляющих сигналов, инициирующие выполнение соответствующих микроопераций в операционном блоке.

Микропрограммное устройство управления функционирует следующим образом. КОП с регистра команд поступает на вход КПМК. КПМК дешифрирует его и на выходе регистра адреса микрокоманд контроллера формируется адрес первой микрокоманды выполняемой микропрограммы. Микрокоманда, подлежащая реализации в текущем микрокомандном цикле, считывается из памяти на регистр микрокоманд. ДШМО декодирует код микрооперации и формирует управляющие слова, которые инициируют выполнение микрокоманды в операционном блоке. АСМК может указываться в микрокоманде явным образом или формироваться естественным путем, как это имеет место при выборке команд. После выполнения выбранной микрокоманды микрокомандный цикл повторяется.

Основными вопросами при проектировании микропрограммного УУ, которые приходится решать с целью достижения оптимальных параметров УУ, являются:

1) определение совместимости во времени циклов выборки и выполнения микрокоманд;

2) анализ способов формирования следующего адреса микрокоманды;

3) выбор способа кодирования микрокоманд;

4) выбор типа синхронизации при формировании микрооперации.

Выборка и выполнение микрокоманд

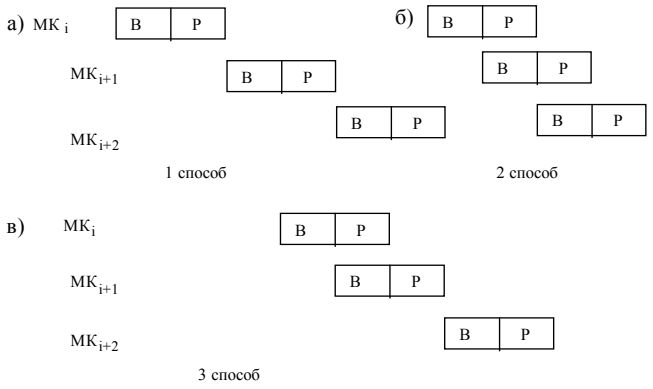
Выделяют три способа совместимости выборки и выполнения микрокоманд:

- последовательный;
- параллельный;
- последовательно-параллельный.

При последовательном способе (рис.2.12,а) выборка следующей микрокоманды $МК_{i+1}$ не инициируется до завершения предыдущей микрокоманды $МК_i$. Достоинством данного подхода является простота организации микрокомандного цикла.

При параллельном способе (рис.2.12,б) имеет место совмещение этапов выборки $МК_{i+1}$ и выполнения $МК_i$. В этом случае при одинаковом времени на этапах выборки и выполнения микрокоманд достигается фактическое сокращение микрокомандного цикла в два раза.

Однако при выполнении микрокоманд условной передачи управления адрес следующей микрокоманды зависит от результата выполнения предыдущей, и в этом случае используют последовательно-параллельный способ выборки и выполнения МК (рис.2.12,в).



i+ Рис.2.12.Выборка и реализация команды: В – выборка; i

Р – реализация

Здесь выборка микрокоманды условного перехода $МК_{i+2}$ начинается только после завершения выполнения предыдущей команды $МК_{i+1}$. Иногда с целью сокращения времени реализации микрокомандного цикла при условном переходе предсказывают направление перехода, и при удачном предсказании получают выигрыш в быстродействии, иначе цикл выборки повторяется.

Способы формирования адреса следующей микрокоманды

Способ адресации микрокоманд определяет правило формирования адреса следующей микрокоманды. В ЭВМ используется два основных способа адресации: принудительный и естественный методы.

Принудительная адресация сводится к указанию в каждой микрокоманде адреса следующей микрокоманды,

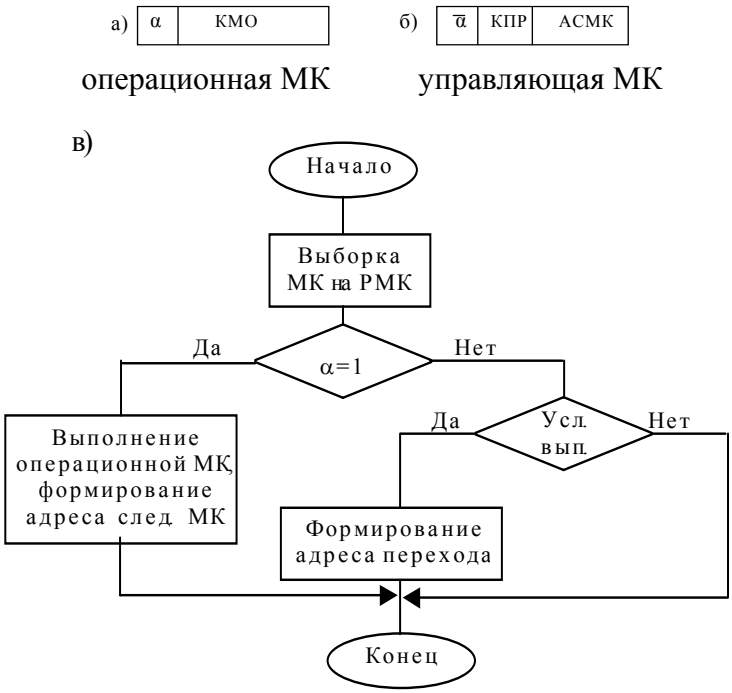


Рис.2.13. Форматы микрокоманды: а) – операционная МК; б) – управляющая МК; в) – микрокомандный цикл

а при естественной адресации - адрес следующей микрокоманды образуется путем приращения адреса предыдущей, как это имеет место при формировании следующей команды.

Это позволяет за счет исключения поля адреса из операционных микрокоманд уменьшить разрядность памяти. На рис.2.13 а), б) приведены форматы операционной и управляющей микрокоманды, а на рис.2.13 в) приведен микрокомандный цикл при естественной адресации. Признак α определяет тип микрокоманды: $\alpha = 1$ - операционная микрокоманды; а при $\alpha = 0$ - управляющая.

Кодирование микрокоманд

Основные способы кодирования микрокоманд.

1) горизонтальное кодирование

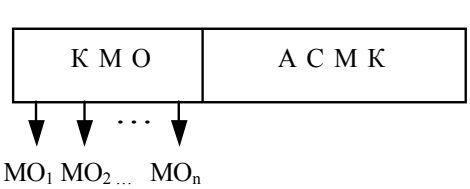


Рис.2.14. Горизонтальное кодирование

Этот способ (рис.2.14) является простейшим вариантом кодирования, при котором каждый разряд поля кода микрооперации однозначно определяет управляющий сигнал для выполнения микрооперации.

Из-за большого набора микроопераций (10-100) горизонтальное кодирование может потребовать большой разрядности памяти микрокоманд. С другой стороны, из-за ограничений на совместимость микроопераций в большинстве микрокоманд лишь небольшое число разрядов в поле КМО будет содержать 1, в основном же, микрокоманда

будет состоять из 0. Это приводит к неэффективному использованию памяти микропрограмм.

Достоинством горизонтального кодирования является возможность параллельной работы нескольких операционных блоков, что позволяет существенно повысить быстродействие и приводит к высокой степени загрузки оборудования, а также простота формирования УС.

2) вертикальное кодирование

При вертикальном кодировании микрооперация определяется не состоянием одного из разрядов микрокоманды, а двоичным кодом, содержащимся в

1 m

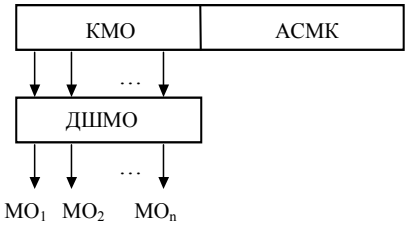


Рис.2.15. Вертикальное кодирование

операционной части микрокоманды (рис.2.15). Количество разрядов операционной части микрокоманды определяется как:

$$m=]\log_2 n[.$$

Отсюда видно, что основным достоинством является небольшая длина микрокоманды, что приводит к сокращению емкости ПМК. Однако, в этом случае требуются сложные дешифраторы на большое количество микроопераций, и увеличиваются временные затраты на дешифрацию, а главное – каждой микрокоманде указывается лишь одна микрооперация, что приводит к увеличению длины микропрограмм по сравнению с горизонтальным кодированием.

3) смешанное кодирование

Развитием способов кодирования микрокоманд с целью устранения основных недостатков, присущих горизонтальному и вертикальному способам, является горизонтально-вертикальное, или смешанное, кодирование микрокоманд (рис.2.16).

При смешанном кодировании множество микроопераций V разбивается на k подмножеств (или полей):

$$V = \bigcup_{i=1}^{k} V_i .$$

Подмножества V_i кодируются горизонтальным, а микрооперации внутри каждого из этих подмножеств вертикальным способами. В этом случае каждому подмножеству V_i выделяется отдельное поле в операционной части микрокоманды. Длина n операционной части микрокоманды равна:

$$n = \sum_{i=1}^{k}]\log_2 m_i[,$$

где m_i - число микроопераций в поле i ($i = \overline{1,k}$).

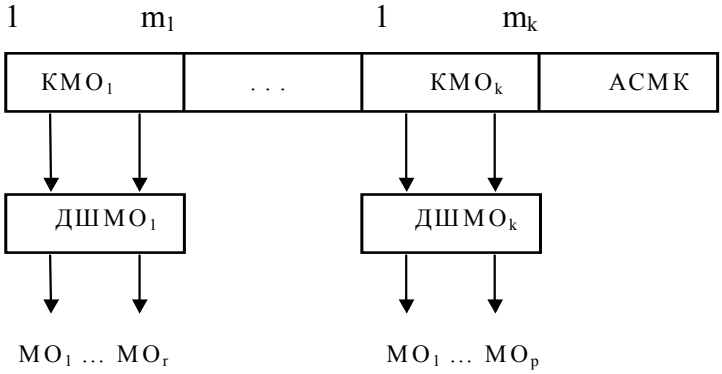


Рис.2.16. Смешанное кодирование

Дешифраторы ДШМО$_i$ ($i = \overline{1,k}$), дешифрирующие код микроопераций отдельных полей, вырабатывают управляющие слова, реализуемые в течение одного микрокомандного цикла. Таким образом, применение вертикального метода кодирования позволяет оптимизировать емкость ПМК, а горизонтального способа – сохранить принципы параллельной работы операционных блоков микропроцессора. Данный способ кодирования находит широкое применение в МПУУ.

4) косвенное кодирование

С целью дальнейшего сокращения разрядности микрокоманды используется косвенный метод кодирования микрокоманд. Косвенное кодирование характеризуется наличием дополнительных полей, содержимое которых меняет смысл основных полей микрокоманды (рис.2.17).

Двухбитное дополнительное поле КМО$_1$ кодирует одну из четырех групп микроопераций, а шестибитное основное поле КМО$_2$ определяет реализацию в данной

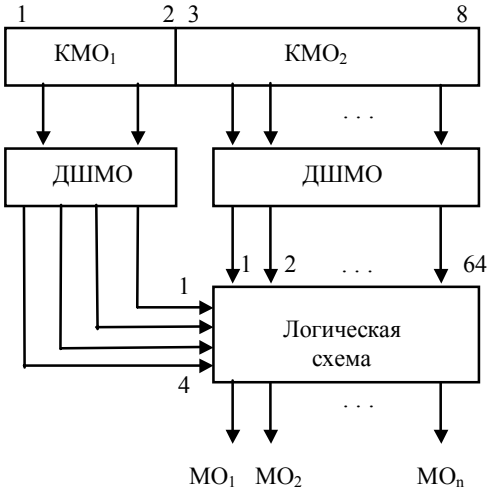


Рис.2.17. Косвенное кодирование

группе микрооперации. Таким образом, интерпретация полей, формирующих УС, зависит от бит дополнительных полей.

Косвенное кодирование сокращает объем памяти микропрограмм, так как позволяет уменьшить длину микрокоманды. Однако, оно требует введения дополнительных дешифраторов, логических схем, что приводит к временным затратам из-за коммутации полей микрокоманды.

5) двухуровневое кодирование

В современных микропроцессорах широко используется двухуровневое кодирование микрокоманд. Например, в МП68000 фирмы Motorola первый уровень микрокоманд использует вертикальное кодирование МК, выбирающие широкие горизонтальные МК второго уровня, называемые нанокомандами. Основой использования данного принципа кодирования является то, что в случае горизонтального кодирования часто для реализации микропрограмм используется небольшая часть параллельно выполняемых МО из максимально возможного их числа. Поэтому за счет хранения данных комбинаций в отдельной памяти нанокоманд (ПНК) небольшой емкости и разрядности, требуемой для горизонтального кодирования МК, можно достичь значительного сокращения общей емкости памяти с максимальным параллелизмом при выполнении микроопераций.

На рис.2.18 представлена структурная схема УУ с двухуровневым кодированием микрокоманд, в котором на первом уровне используется вертикальное кодирование, а на втором – горизонтальное. Выполнение микрокоманды осуществляется следующим образом.

Первая микрокоманда с вертикальным кодированием извлекается из ПМК, поле кода микрооперации, которой является адресом нанокоманды. Нанокоманда, считанная из памяти, и определяет множество микроопераций, реализуемых в текущем микрокомандном цикле.

Например, память для хранения микропрограмм объемом 4К 32-битных микрокоманд при числе нанокоманд 256 может быть реализовано как совокупность памяти

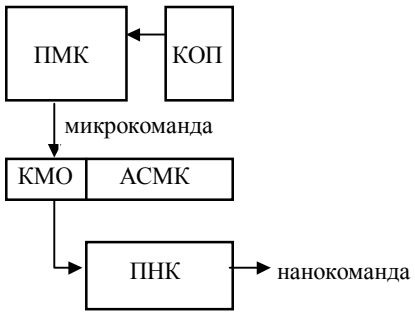


Рис.2.18. УУ с двухуровневым кодированием микрокоманд

микрокоманд емкостью 4К 8-битных слов и памяти нанокоманд емкостью 256 32-битных слов. Это позволяет сократить память в три раза. Обычно такие структуры УУ реализуются на программируемых логических матрицах.

Синхронизация микрокоманд

В основе синхронизации микрокоманды лежит число тактирующих сигналов, необходимых для ее реализации. С этой точки зрения выделяют однотактные микрокоманды и многотактные, для реализации которых требуется последовательность тактирующих сигналов.

В однотактных микрокомандах все микрооперации выполняются одновременно в течение одного такта, и называется микрокомандным циклом.

В многотактных микрокомандах такт (микрокомандный цикл) разбивается на части, называемые микротактами, и указанные в микрокоманде микрооперации выполняются в различные микротакты. В этом случае приходится учитывать временные зависимости между отдельными микрооперациями. Однако, становится возможным включать в микрокоманду взаимно исключающие микрооперации, разводя их по разным тактам.

Многотактная синхронизация позволяет минимизировать число микрокоманд в памяти, упрощает параллельную выборку и выполнение микрокоманд, также упрощает связи между источниками и приемниками информации при выполнении микрокоманд.

а) б)

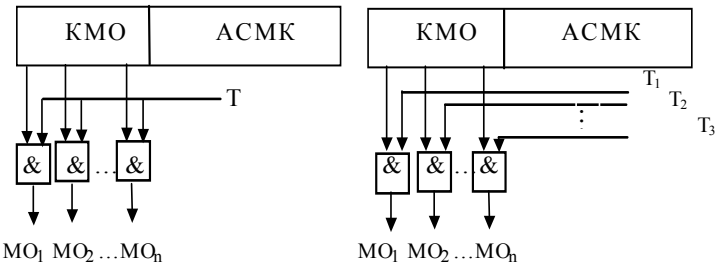


Рис.2.19. Синхронизация: а) однотактная б) многотактная

Недостатком многотактной синхронизации является то, что она требует большого объема оборудования по обеспечении синхронизации отдельных фаз сложной микрокоманды. Достоинством однотактной синхронизации является простота его технической реализации.

2.5. Организация интерфейса в микропроцессорных системах

ЭВМ осуществляет обмен информацией с внешним миром посредством периферийных устройств (ПУ), таких как телетайпы, клавиатура, дисплеи, магнитные диски, ленты, АЦП, ЦАП и др. К числу ПУ могут относиться более простые цифровые устройства: триггеры, регистры и т.п.

Все ПУ требуют определенного набора управляющих сигналов, протокола обмена, способа обмена с МП и вида используемого кода. Поэтому шины обмена информацией подключаются не непосредственно к ПУ, а через интерфейс, структура и принцип работы которого в сильной мере определяется совместимостью сопрягаемых компонентов.

Для подключения ПУ к МП используют специальные электронные схемы, называемыми интерфейсными модулями. Сложность интерфейса определяется:

1) типом ПУ;
2) их числом;
3) расстоянием между МП и ПУ;
4) физической природой;
5) их архитектурой.

В ряде случаев компоненты интерфейса разбиваются на две группы: одну относят к интерфейсу МП, а другую - к ПУ и называют контроллером. Вообще термины "интерфейс" и "контроллер" строго не определены и их иногда используют как синонимы.

Кроме аппаратных средств для организации интерфейса необходимо разрабатывать некоторое программное обеспечение, которое включает в себя программы идентификации типа информации (данные, управляющие символы и т.п.), программы преобразования форматов, программы-драйверы для управления обменом информации, программы обработки запросов прерываний и др.

Под интерфейсом будем понимать совокупность программных и аппаратных средств, с помощью которых компоненты системы объединяются таким образом, чтобы она могла решать определенную задачу.

Интерфейс представляет собой совокупность линий и шин, сигналов, электронных схем, связывающие данные устройства, и алгоритмы (протоколы) процедур, обеспечивающие обмен информации между устройствами системы.

Сложность интерфейса в значительной мере определяется степенью совместимости МП и ПУ, т. е. сложностью необходимых преобразований. Под совместимостью будем

понимать возможность объединения отдельных компонентов в единую операционную сеть посредством программных и аппаратурных средств.

Совместимость определяется четырьмя основными признаками:

- быстродействием ЭВМ и ПУ;
- кодами, используемыми для обмена;
- архитектурой процессора;
- электрическими параметрами.

Рассмотрим эти признаки совместимости. Различное быстродействие МП и ПУ требует контроля над состоянием признаков готовности (флажков) ПУ к обмену и введения буферных регистров для временного хранения передаваемых кодов с целью согласования работы МП и ПУ по быстродействию.

Для передачи данных используются стандартные коды, содержащие алфавитно-цифровые, специальные, управляющие символы, которые должны интерпретироваться компонентами системы одинаковым образом с точки зрения семантики. Например, если необходимо некоторое перекодирование, то эта функция выполняется программными или аппаратными средствами интерфейса.

Архитектура процессора определяет протокол обмена или метод синхронизации МП и ПУ. Передача данных, кодов, признаков состояния, управляющих сигналов должны подчиняться этому протоколу обмену.

Электрические характеристики процессора должны быть совместимы с характеристиками логических схем интерфейса, которые в свою очередь должны быть согласованы с ПУ. Если объединяемые компоненты не соответствуют друг другу по одному или нескольким признакам, то они не могут быть объединены без интерфейсных модулей. Интерфейсный модуль - это устройство, позволяющее реализовать полную совместимость между сопрягаемыми устройствами.

Обмен информацией между ЦП, памятью и ПУ осуществляется по системе шин. Состав, назначения, правила использования, функциональное и физическое разделение при передаче данных, адресов, управляющих сигналов различны для разных ЭВМ.

Однако есть общие закономерности в организации интерфейса, процедуре обмена между двумя устройствами, одно из которых является источником, а другое приемником информации.

В общем случае, для обмена информацией необходимо осуществить следующие основные действия:

- адресацию устройства-приемника;
- анализ состояния приемника на готовность для обеспечения совместимости по быстродействию;
- выполнение требуемых управляющих функций (действий) в приемнике (начальный сброс, перемотку, старт, остановка и др.);
- передачу и буферирование данных, если необходимо;
- окончание процесса обмена путем засылки в процессор сигнала прерывания.

Несмотря на многообразие типов МП, можно выделить две основные организации интерфейса между МП, памятью и ПУ:

- двухшинную организацию, или интерфейс с изолированной системой шин. Название «двухшинная» отражает тот факт, что с функциональной точки зрения есть два способа передачи кода, процессор - память и процессор – периферийное устройство, обращение к которым осуществляется отдельными группами команд;
- одношинную организацию интерфейса, или интерфейс с общей шиной. При этой организации передача кодов осуществляется одним способом процессор – внешняя подсистема, обращение к которой осуществляется одной группой команд. В этом случае

обращение к памяти и ПУ будет осуществляться одинаковым способом и одними и теми же командами.

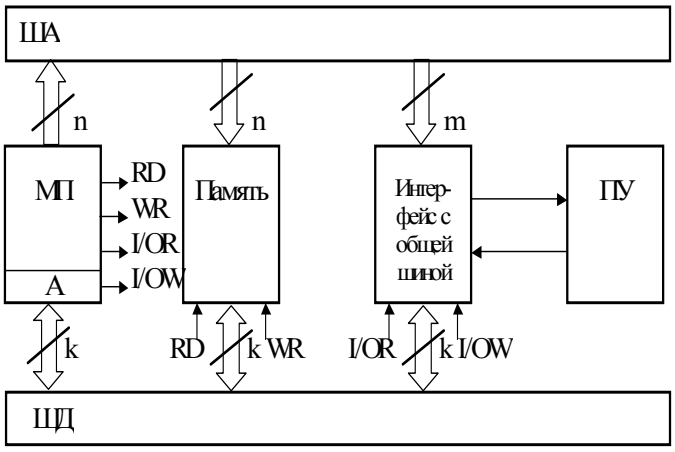
Интерфейс с изолированной шиной

На рис.2.20,а показана организация данного интерфейса. Особенностью данного интерфейса является раздельная адресация памяти и периферийных устройств при обмене информацией. Это осуществляется путем использования отдельных групп команд для обмена информацией с ПУ и памятью. В качестве примера рассмотрим организацию обмена в МПС на базе КР580. Для обмена с ПУ используются двухбайтовые команды ввода IN port и вывода OUT port, формат которых приведен на рис.2.20,б.

Число выполняемых тактов у этих команд равно 10. Число циклов равно 3 (4+3+3) Данные команды позволяют адресовать 256 портов ввода и 256 портов вывода. Управление обменом выполняется под действием управляющих сигналов ввода I/OR и I/OW, которые формируются системным контроллером при выполнении этих команд.

МП передает адрес порта по 8 младшим и 8 старшим линиям шины адреса. Передача осуществляется между аккумулятором А и буферным регистром интерфейсного модуля ПУ.

а)



б)

Рис. 2.20. Интерфейс с изолированной шиной

Обмен информацией с памятью выполняется путем использования команд передачи данных. Команда MOV r,M (число тактов 7, число циклов 2, число байт 1) выполняет передачу из ячейки памяти, адресуемой HL парой, в аккумулятор или в любой РОН, кодируемый в поле r. А команда MOV M,r производит обратные действия. Максимальная емкость адресуемой памяти при такой организации обмена для 16-битной шины адреса составляет 64Кбайт.

Недостатками при организации интерфейса с изолированной шиной являются:

1. 1.Обмен с ПУ осуществляется только через аккумулятор и приводит к удлинению программ обработки слов от ПУ. Рассмотрим пример ввода кода с ПУ с адресом, равным 001, на регистр В.

Метка	Код	Операнд	Комментарий
	MOV	C,A	;Временное запоминание содержимого аккумулятора в регистре С
	IN	001H	;Ввод кода из ПУ в аккумулятор
	MOV	B,A	;Передача содержимого аккумулятора в регистр В
	MOV	A,C	;Восстановление аккумулятора

Этот пример показывает, что для ввода кода из ПУ, если занят аккумулятор, требуется выполнение трех команд вместо одной.

2. Число подключаемых периферийных устройств ограничено. Это ограничение определяется размерами поля адреса в командах ввода-вывода и в нашем случае, составляет 256 подключаемых ПУ ввода и вывода.

Рис.2.21

3. Для обработки содержимого буферного регистра ПУ, например, для анализа готовности к обмену, его код необходимо передать в МП.

Интерфейс с общей шиной

При данной организации интерфейса часть общего адресного пространства отводится для периферийных устройств, регистры которого адресуются, как и ячейки памяти (рис.2.21). Обращение к ПУ осуществляется посредством набора команд, используемых для передачи данных в память.

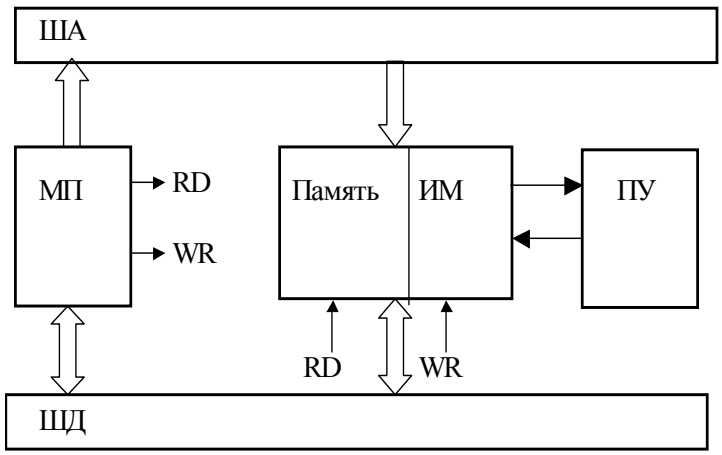


Рис. 2.21. Интерфейс с общей шиной

При этом команда ввода-вывода не используется, а в некоторых МП вовсе отсутствует. Под ПУ можно выделить любое адресное пространство, но при этом, если нет аппаратного разделения этих адресных пространств, необходимо разделить их программно.

Достоинства этого интерфейса:

1) расширение набора команд для обращения к ПУ, что позволяет сократить объем программ и повысить быстродействие;

2) значительное увеличение числа подключаемых к МП периферийных устройств;

3) возможность внепроцессорного обмена информацией между ПУ, если в системе команд имеются команды передачи между ячейками памяти;

4) возможность обмена информацией не только с аккумулятором, но и с любым регистром МП.

Недостатки данной организации интерфейса:

1) сокращение объема памяти;

2) усложнение дешифрирующих схем (при аппаратном разделении адресного пространства).

Контрольные вопросы

1. Определите основные архитектурные элементы микропроцессора.
2. Перечислите основные типы и формы представления данных в микропроцессорах.
3. Какие существуют способы уменьшения формата команд?
4. Дайте классификацию системы команд по функциональному назначению и кратко охарактеризуйте их.
5. Объясните различие каждого поля в командах передачи управления.
6. Что такое режим адресации и с какой целью он вводится?
7. Укажите достоинства регистрово-косвенной адресации.
8. В чем заключаются достоинства режимов индексной и относительной адресации?
9. Дайте краткую классификацию микропроцессоров.
10. Определите основные черты специализированных микропроцессоров и приведите примеры их использования.
11. Перечислите основные элементы операционной части микропроцессора с указанием их функций.
12. Укажите основные циклы выполнения команд микропроцессора и дайте им краткую характеристику?
13. Определите основную роль счетчика команд микропроцессора?
14. С какой целью вводится указатель стека и определите его основное назначение?
15. Для чего служат регистры общего назначения и дайте им краткую характеристику?
16. В чем заключается назначение АЛУ? Укажите его состав.
17. Какие наиболее распространенные признаки вырабатываются АЛУ? Укажите их назначение.
18. Укажите основные способы реализации устройства управления микропроцессора.
19. Опишите функционирование микропрограммного устройства управления.
20. Укажите основное назначение контроллера последовательности микрокоманд.
21. Содержимое, какого регистра микропроцессора является входной информацией для устройства управления?
22. Укажите основные способы формирования адреса следующей микрокоманды.
23. Какие основные способы кодирования микрокоманд используются для построения устройств управления?
24. Укажите преимущества горизонтального кодирования микрокоманд перед вертикальным?
25. Определите преимущества и недостатки косвенного кодирования микрокоманд?
26. В чем заключаются преимущества двухуровневого кодирования микрокоманд и дайте им оценку?
27. Укажите достоинства и недостатки однотактной и многотактной синхронизации микрокоманд.
28. Определите основное назначение интерфейса и его функции.
29. Определите назначение интерфейсного модуля и его сложность.
30. Какими признаками определяется совместимость интерфейса.
31. Какие способы организации интерфейса используются в микропроцессорных системах?
32. Определите достоинства и недостатки интерфейса с изолированной шиной.
33. Определите достоинства и недостатки интерфейса с общей шиной.

Глава 3

ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

Вводом-выводом (ВВ) называются передачи данных между ядром ЭВМ, включающим в себя процессор и основную память, и периферийными устройствами. Архитектура ввода-вывода (режимы работы, форматы команд, особенности прерываний, скорость обмена и др.) непосредственно влияет на эффективность всей микропроцессорной системы.

За время эволюции ЭВМ подсистема ВВ претерпела наибольшие изменения благодаря расширению сферы применения ЭВМ и появлению новых ПУ. Особенно важную роль средства ВВ играют в управляющих ЭВМ. Разработка аппаратных и программных средств ВВ является наиболее сложным этапом проектирования новых систем на базе микропроцессора вследствие разнообразия подключаемых ПУ.

Несмотря на разнообразие ПУ, разработано несколько стандартных способов подключения их к микропроцессору и программирования ввода-вывода. В микропроцессорных системах применяются три режима ввода-вывода:

- программный ввод-вывод (программно-управляемый);
- ввод-вывод по прерываниям (форсированный ввод-вывод);
- прямой доступ к памяти (ПДП).

Программный ввод-вывод характеризуется тем, что инициирование и управление вводом-выводом осуществляется процессором, а ПУ играет пассивную роль, сообщая микропроцессору свое состояние, т.е. готовность к операциям ввода-вывода.

Ввод-вывод по прерываниям инициируется не процессором, а периферийным устройством генерирующий специальный сигнал прерывания. Процессор реагирует на этот сигнал готовности ПУ к передаче данных и переходит на подпрограмму обслуживания устройства, вызвавшего это прерывание. Действия, выполняемые этой подпрограммой, определяются пользователем, а непосредственно операциями ввода-вывода управляет процессор.

В режиме прямого доступа к памяти (ПДП) действия микропроцессора приостанавливаются, он отключается от системной шины и совершенно не участвует в операциях ввода-вывода. Все операции по обмену данными производятся под управлением ПУ.

3.1. Программный ввод-вывод

Данный режим характеризуется тем, что все действия по вводу-выводу реализуются командами прикладной программы. Инициатива проведения операций ввода-вывода в этом способе исходит от микропроцессора. Для большинства ПУ до выполнения операций ввода-вывода необходимо убедиться в их готовности, после чего непосредственно осуществляется обмен данными. Таким образом, ввод-вывод является асинхронным. Для определенной категории ПУ (например, индикаторы на светодиодах) отсутствует необходимость проверки готовности к операциям ввода-вывода, в этом случае в соответствующих местах программы используются команды ввода IN или вывода OUT. Такая передача данных называется синхронным или безусловным вводом –выводом.

Общее состояние ПУ в этом режиме ввода–вывода характеризуется флагом готовности READY, называемым также флажком готовности/занятости (READY/BUSY). Иногда состояния готовности и занятости идентифицируются отдельными флажками READY и BUSY, входящими в слово состояния устройства.

Интерпретация флажка READY в устройствах ввода и устройствах вывода различна. Для устройств ввода обычно принимается следующая интерпретация флажка READY и выполняемых с ним операциям:

- состояние READY=0 означает отсутствие входных данных в регистре ввода;
- состояние READY=1 определяет наличие входных данных в регистре ввода;
- подпрограмма инициализации сбрасывает флажок READY;
- при загрузке входных данных ПУ в регистр ввода флажок READY автоматически устанавливается;
- при вводе данных (считывании) микропроцессором флажок READY сбрасывается.

Для устройств вывода соответствующие состояния и действия имеют другой смысл:

- состояние READY=0 означает недоступность регистра вывода для приема данных от микропроцессора;
- состояние READY=1 сигнализирует о доступности регистра вывода;
- подпрограмма инициализации устанавливает флажок READY;
- при выводе (записи) данных микропроцессором флажок READY сбрасывается;
- при восприятии выходных данных из регистра вывода непосредственно в ПУ флажок READY автоматически устанавливается.

Микропроцессор проверяет флажок готовности с помощью одной или нескольких команд. Если флажок установлен, инициируется собственно ввод или вывод одного или нескольких слов данных. Когда же флажок сброшен, микропроцессор выполняет цикл из 2-3 команд с повторной проверкой флажка READY до тех пор, пока устройство не будет готово к операциям ввода-вывода. Данный цикл называется циклом ожидания и реализуется в различных микропроцессорах по-разному.

Основной недостаток программного ввода-вывода связан с непроизводительными потерями времени микропроцессора в циклах ожидания. Для некоторых ПУ основное время микропроцессора приходиться на бесполезные циклы ожидания, в которых он не выполняет какой-либо полезной работы. К достоинству программного ВВ относят простоту его реализации, не требующих дополнительных аппаратных средств.

3.2. Ввод-вывод с прямым доступом к памяти

Обмен данными микропроцессора с медленнодействующими ПУ обычно организуется по прерываниям или по программному опросу. Однако при передаче между основной и внешней памятью микро-ЭВМ больших блоков данных (128-2048 байт) производительность процессора в этих режимах является недостаточным.

Например, в НГМД с одинарной плотностью записи скорость передачи после локализации требуемых данных составляет 31,25Кбайт/сек (интервал между байтами 32мкс), а при двойной плотности записи скорость передачи достигает 62,5Кбайт/сек (интервал 16мкс). Вместе с тем цикл обращения полупроводниковой оперативной памяти составляет всего 0,1-2мкс, в то же время подпрограмма ввода одного байта длится 50мкс. Отсюда видно, что скорость передачи данных в режиме программного ввода-вывода ограничивается только процессором. Поэтому для передачи данных между устройствами внешней памяти и ОЗУ разработан специальный метод передачи данных без участия процессора, получившего название прямого доступа к памяти. Аппаратные средства реализации канала ПДП называются контроллером прямого доступа к памяти (КПДП). Первые КПДП выполнялись на микросхемах малой и средней степени интеграции, а сейчас в большинстве микропроцессорных семействах выпускаются однокристальные контроллеры ПДП.

В идеальном случае режим ПДП совершенно не должен влиять на действия процессора, но для этого потребуется сложный и дорогой тракт в основную память ЭВМ. Поэтому в большинстве ЭВМ используется временное разделение (мультиплексирование) общей системной шины между процессором и КПДП.

При инициализации режима ПДП, системной шиной ″распоряжается″ КПДП, который управляет передачей данных между основной и внешней памятью, действия процессора в этот момент приостанавливаются и он отключается от системной шины. Электрическое

отключение достигается переводом буферов на шинах данных, адреса и некоторых линий управления памятью и вводом-выводом в состояние высокого выходного сопротивления.

Разработано две разновидности ПДП: режим без пропусков тактов микропроцессора и режим с пропуском тактов микропроцессора.

В первом режиме реализации прямой доступ осуществляется без участия процессора (параллельно микропроцессору). Для этого используются те интервалы машинных циклов, в течение которых микропроцессор не обращается к основной памяти (например, такими интервалами в микропроцессоре КР580 являются такты T_4,T_5). Процессор (или дополнительная схема) идентифицирует эти интервалы для КПДП специальным сигналом, означающим доступность системной шины. Производительность процессора в этом режиме не уменьшается, но для каждого типа процессора потребуется свой контроллер ПДП. С другой стороны, сами передачи будут носить нерегулярный характер ввиду отсутствия у некоторых команд этих интервалов, что приведет к уменьшению скорости передачи данных в режиме ПДП.

Во втором способе реализации КПДП полностью ″захватывает″ системную шину на время передачи, при этом процессор отключается от системной шины и переходит в режим холостого хода. Таким образом, передачи ПДП осуществляются путем пропуска тактов процессора в выполняемой программе. При выполнении передач ПДП содержимое внутренних регистров процессора не модифицируются, поэтому его не нужно запоминать в памяти, а затем восстанавливать, как при обработке прерываний. Выполнение программы осуществляется сразу после окончания ПДП. Тем не менее, в условиях интенсивных передач ПДП эффективная производительность процессора уменьшается.

Аппаратная реализация каналов ПДП определятся особенностями ЭВМ и устройств внешней памяти, но можно сформулировать общие принципы работы каналов ПДП.

Общие принципы организации ПДП

Режим ПДП является самым скоростным способом обмена между основной и внешней памятью. Режим ПДП реализуется с помощью специальных аппаратных средств – котроллеров ПДП без использования программного обеспечения. Для осуществления режима ПДП контроллер должен выполнить ряд последовательных операций для передачи данных в этом режиме, называемых также циклами ПДП:

- принять запрос на ПДП от ПУ (DREQ);
- сформировать запрос процессору для перехода в режим ПДП (HRQ);
- принять сигнал (HLDA), подтверждающий переход процессора в режим ПДП (ШД, ША, ШУ в z-состояние);
- сформировать сигнал (DACK), сообщающий ПУ о начале выполнения циклов ПДП;
- сформировать на ША адрес ячейки памяти, предназначенной для обмена;
- выработать сигналы чтение из памяти, запись в ПУ (MR, IOW) и чтение из ПУ, запись в память (IOR, MW), обеспечивающие управление обменом ;
- по окончании ПДП либо повторить цикл ПДП, изменив адрес, либо прекратить ПДП, сняв запросы ПДП.

На рис.3.1 показана структурная схема микропроцессорной системы с контроллером ПДП.

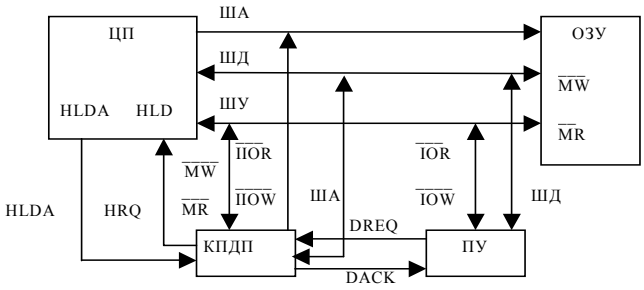


Рис.3.1. Структурная схема МПС с контроллером ПДП

Циклы ПДП выполняются с последовательно расположенными ячейками памяти, поэтому КПДП должен иметь счетчик адреса ОЗУ. Число циклов ПДП определяется специальным счетчиком. Управление обменом осуществляется специальной логической схемой, формирующей в зависимости от типа обмена пары управляющих сигналов: MR, IOW (циклы чтения) и IOR, MW (циклы записи).

Из изложенного следует, что контроллер ПДП по запросу (DREQ) от устройства внешней памяти ПУ должен взять на себя управление системными шинами и выполнять совмещенные циклы чтения или записи до тех пор, пока содержимое счетчика циклов ПДП не будет равно нулю. После этого, устройство внешней памяти снимает запрос ПДП (DREQ), что приводит к снятию соответствующего запроса в процессор (HRQ), и он возобновляет приостановленную программу.

3.3. Ввод-вывод в режиме прерывания

При организации режима прерывания в микропроцессорных системах непроизводительные потери времени микропроцессора в циклах ожидания резко сокращаются. Это позволяет строить гибкие системы прерывания с подключением большого количества устройств, работающих с различными скоростями.

Рассмотрим принципы организации ввода-вывода в режиме прерывания. При выполнении текущей программы могут возникать события, как внешние, так и внутренние по отношению к микропроцессору, которые требуют немедленной реакции с его стороны. Эта реакция состоит в том, что при возникновении подобных событий микропроцессор должен прервать обработку текущей программы и перейти к выполнению другой программы однозначно связанной с данным событием. По завершению этой программы микропроцессор должен вернуться к выполнению прерванной программы. Этот процесс называется прерыванием программ.

Каждое событие, требующее прерывание, должно сопровождаться сигналом, оповещающим микропроцессор о его возникновении. Этот сигнал называется запросом прерывания (INT). Программы, затребованные запросами прерывания, называются прерывающими программами, а программы, выполнявшиеся до появления запросов прерывания, будем называть прерываемыми программами.

Запросы на прерывание могут возникать как внутри микропроцессора, так и от ПУ. К первым относятся, например, запросы при возникновении таких событий, как переполнение разрядной сетки, попытка деления на 0, сигнал от систем автоматического контроля и т.д. Запросы прерывания возникают по мере необходимости в обмене информации.

Каждое ПУ может посылать в микропроцессор сигнал запроса прерывания INT, когда оно готово к операциям ввода-вывода, который требует немедленной реакции микропроцессора. Сигнал INT появляется в произвольные моменты времени, асинхронно по отношению к действиям микропроцессора, и управлять его появлением программа не может. Поэтому, заранее неизвестно, в какой точке программы, и какие ПУ инициируют прерывания.

Следовательно, непосредственное использование команд ввода-вывода представляется невозможным, а для этого необходимо микропроцессору придать дополнительные аппаратные и программные средства, совокупность которых получила название системы прерывания.

Основное назначение системы прерывания - это автоматическое прерывание программ с целью увеличения его скоростью переключения.

Основные функции системы прерывания:

1) приостановка и сохранение состояния прерываемой программы;
2) идентификация прерывающего устройства и организация перехода к прерывающей программе.
3) восстановление состояния прерванной программы и возврат к ней;
4) программное изменение приоритетов запросов.

Программа обслуживания прерывания (прерывающая программа) непосредственно выполняется в микропроцессоре и разрабатывается системным программистом для каждого конкретного ПУ.

Подпрограмме обслуживания потребуются внутренние регистры микропроцессора: аккумулятор, программный счетчик, некоторые РОН, содержимое которых могут быть модифицированы. С другой стороны прерываемая программа должна быть возобновлена с точки прерывания, т.е. должны восстановиться программный счетчик и все внутренние регистры микропроцессора. Кроме как увеличения времени выполнения, факт обслуживания прерывания не должен влиять на прерываемую программу.

Для сохранения содержимого внутренних регистров микропроцессора удобно использовать в качестве такого временного хранилища стек.

Практически в каждом микропроцессоре реализована особая структура системы прерываний, а программируемые БИС управления прерываниями еще более увеличивают число разновидностей этой структуры.

Однако общая последовательность реакции различных микропроцессоров на сигнал прерывания примерно одинакова и содержит следующие действия:

- ПУ генерирует сигнал прерывания, который подается на вход прерывания микропроцессора (INT); на этой линии по схеме ИЛИ объединяются запросы всех ПУ, работающих в режиме прерывания;
- микропроцессор завершает текущую команду и, если прерывания разрешены (не замаскированы), формируют сигнал INTA (INT ACK) подтверждения прерывания; до получения этого сигнала ПУ сохраняет активный уровень сигнала INT;
- осуществляется запоминание содержимого программного счетчика и некоторых РОН в стеке;
- микропроцессор идентифицирует прерывающее ПУ для перехода к соответствующей подпрограмме обслуживания;
- выполняется короткая (30-50 байт) подпрограмма обслуживания прерывания, в которой запрограммированы действия по передаче данных, модификации указателей, проверке окончания операций ВВ и др.;
- восстанавливается состояние прерванной программы, для чего запомненное содержимое регистров извлекаются из стека;
- возобновляется выполнение прерываемой программы; это действие инициируется командой возврата из прерывания RTI, являющейся последней командой подпрограммы обслуживания прерывания.

Процессы запоминания содержимого внутренних регистров после восприятия сигнала прерывания называются контекстным переключением микропроцессора.

Скорость контекстного переключения оказывает заметное влияние на производительность ЭВМ, особенно в условиях интенсивных прерываний. Поэтому во многих микропроцессорах предусматриваются средства ускорения контекстного переключения. Например, команды, которые загружают в стек и извлекают из стека

содержимое группы регистров. Из общей последовательности событий при прерываниях видно, что они похожи на действия при вызове подпрограммы. Однако вызов подпрограммы запрограммирован и полностью предсказуем, а переход к обслуживанию прерывания инициируется внешним сигналом, момент появления которого предсказать невозможно. Тем не менее, реакцию микропроцессора на сигнал прерывания, позволяет считать прерывание аппаратным вызовом подпрограммы.

3.3.1. Характеристики систем прерываний

Для оценки эффективности систем прерывания рассмотрим следующие характеристики.

Если управление сохранением состояния и возвратом возложено на саму прерывающую программу, то она состоит из трех частей: подготовительной и заключительной, обеспечивающих переключение программ, и собственно прерывающей программы, выполняющую затребованную запросом передачу информации.

На рис.3.2 приведена временная диаграмма процесса прерывания.

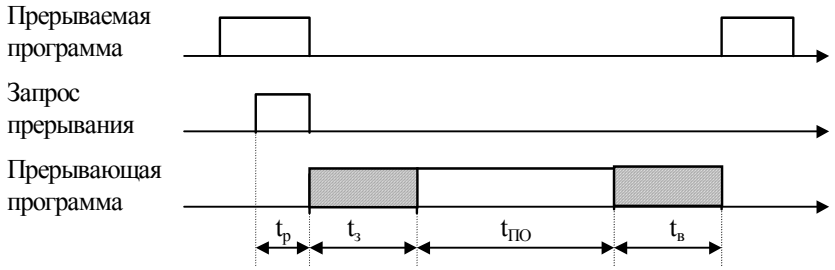


Рис.3.2 Временная диаграмма процесса прерывания

$t_р$- время реакции;

$t_з$- сохранение состояния прерванной программы;

$t_в$- время восстановления состояния прерванной программы;

$t_{ПО}$- время выполнения прерывающей программы.

Время реакции $t_р$ - время между появлением сигнала прерывания и началом выполнения прерывающей программы (включая время выполнения цикла прерывания микропроцессора).

Время обслуживания t_0 есть сумма времени, затраченной на сохранение состояния прерванной программы, и времени на возврат к ней:

$$t_о = t_з + t_в.$$

Глубина прерывания - максимальное число программ, которые могут прерывать друг друга. Глубина равна К, если допускается последовательное прерывание К программ

Насыщение системы прерывания возникает в том случае, если время реакции настолько велико, что запрос оказывается не обслуженным до момента прихода нового запроса от того же источника. В этом случае предыдущий запрос прерывания от данного источника будет системой утрачен. Это явление называется насыщением системы прерывания. Быстродействие микропроцессора, логические возможности системы прерывания и количество источников прерывания должны быть согласованы таким образом, чтобы насыщение было невозможным.

Эффективность прерывания - это отношение времени, необходимого для выполнения прерывающей программы, к общему времени, необходимому для выполнения прерывания:

$$Э=\frac{t_{ПО}}{t_{р}+t_{з}+t_{ПО}+t_{в}}.$$

3.3.2. Виды прерываний микропроцессора

Существуют два основных вида прерывания: программное прерывание и аппаратное прерывание.

Программное прерывание реализуется введением специальных команд прерывания в систему команд процессора. Введение таких специальных команд позволяет создавать гибкие и мощные программные средства (например, операционные системы).

Аппаратные прерывания могут инициироваться как операционными блоками микропроцессора, так и устройствами внешними по отношению к нему. Аппаратные прерывания подразделяются на маскируемые прерывания и немаскируемые прерывания.

Маскируемые прерывания реализуется только при условии разрешения прерывания. Процессор реагирует на запросы маскируемых прерываний по линии INT, если установлен внутренний триггер разрешения прерываний INTE, называемой также маской. На рис.3.3 приведена функциональная внутренняя схема прерываний.

Состояние триггера разрешения прерывания INTE идентифицируется выходным сигналом разрешения прерывания с такой же мнемоникой INTE. Если INTE=0, прерывания запрещены (замаскированы) и процессор не реагирует на сигнал INT=1. С помощью команд разрешения EI и запрещения DI прерываний можно программно управлять состоянием триггера INTE, и пользователь может защитить от прерываний критические сегменты прикладной программы.

При восприятии прерывания триггер прерывания IFF переводиться в нулевое состояние, что приводит к запрещению инкремента программного счетчика и генерированию сигнала подтверждения прерывания INTA.

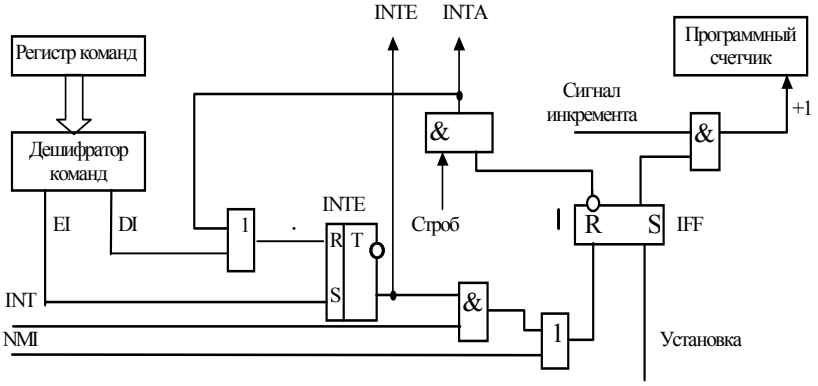


Рис.3.3. Внутренняя схема прерывания

При этом сбрасывается триггер INTE и в дальнейшем разрешить прерывание можно только командой EI. Для программного управления ПУ в их регистрах управления и состояния предусмотрен специальный бит INTEN разрешения прерывания (маска). Иногда биты масок всех устройств объединяются в специальный регистр. Наконец, в некоторых микропроцессорах бит маскирования прерывания входит в слово состояния процессора PSW.

Немаскируемые прерывания реализуют режим безусловного прерывания. Микропроцессор воспринимает запросы прерывания независимо от того, в каком состоянии находится триггер разрешения прерывания. Эти прерывания должны отражать ситуации, в которых желательно немедленное прерывание (например, ошибка аппаратуры контроля, уменьшение напряжения сети до критического уровня и т.д.).

3.3.3. Идентификация прерывающего устройства системы прерывания

Когда в микропроцессорных системах имеется несколько ПУ, работающих в режиме прерываний, сигналы их запросов на обслуживание прерываний объединяются по схеме ИЛИ и подаются на вход INT микропроцессора. В этом случае возникает задача идентификации ПУ, т.е. однозначного перехода к определенной подпрограмме обслуживания. Решение этой задачи возлагается на систему прерывания. Разработано несколько способов решения этой проблемы, различающихся скоростью реакции микропроцессора и объемом дополнительных аппаратных средств. При реализации любого способа необходимо назначить устройствам определенные приоритеты.

Программный полинг

Простейшее решение проблемы идентификации, почти не требующих дополнительных аппаратных средств, заключается в программном опросе (полинге) флажков готовности (сигналов прерывания) ПУ.

На рис.3.4 приведен типичный вариант программного полинга. Первым проверяется флажок готовности ПУ1 с наибольшим приоритетом. Если оно не запрашивало обслуживание, опрашивается следующее ПУ и т.д.

Когда встречается первое устройство готовое к операциям ВВ, управление передается подпрограмме обслуживания этого устройства. При завершении обслуживания в полинге может быть запрограммировано одно из следующих действий:

1. Управление возвращается в основную программу без проверки готовности остальных ПУ. Здесь гарантируется обязательная проверка в каждом цикле полинга ПУ с высоким приоритетом, так как обслуживание их блокирует обслуживание устройств с меньшим приоритетом.

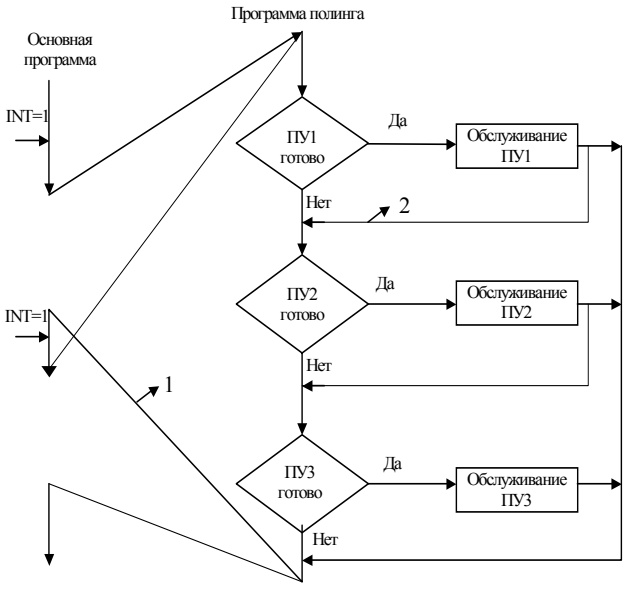


Рис.3.4. Программный полинг

2. Управление возвращается к программе полинга, т.е. в точку проверки прерывания следующего ПУ (на рис.3.4 это показано цифрой). Этот способ гарантирует проверку в каждом цикле полинга всех устройств.

Для ускорения полинга сигналы прерываний всех устройств подключаются к специальному регистру. Приоритет ПУ задается последовательностью опроса флажков

готовности, при обнаружении первого установленного бита осуществляется переход к соответствующей подпрограмме обслуживания.

Недостаток программного полинга заключается в увеличении времени реакции на запросы устройств, так как в этом случае необходима проверка всех устройств, даже тех которые не требуют обслуживания. При увеличении числа ПУ быстро увеличивается время реакции на запрос. Основное достоинство программного полинга заключается в простоте его реализации, почти не требующей дополнительных аппаратных средств.

Аппаратный полинг

В случае аппаратного полинга (дейзи - цепочка, приоритетная цепочка, гирляндное или каскадное включение устройств) микропроцессор и все ПУ соединяются таким образом, что микропроцессор может осуществить автоматический запрос с целью идентификации прерывающего устройства, как показано на рис.3.5.

Когда микропроцессор реагирует на запрос прерывания, он формирует сигнал подтверждения прерывания INTA, на линии, которая последовательно проходит через все устройства. При прохождении сигнала по цепочке проверяется состояние флажков готовности ПУ. Если ПУ не формирует сигнал прерывания, сигнал INTA проходит в следующее ПУ, пока не встретит активное ПУ. Это ПУ блокирует дальнейшее распространение сигнала INTA по цепочке.

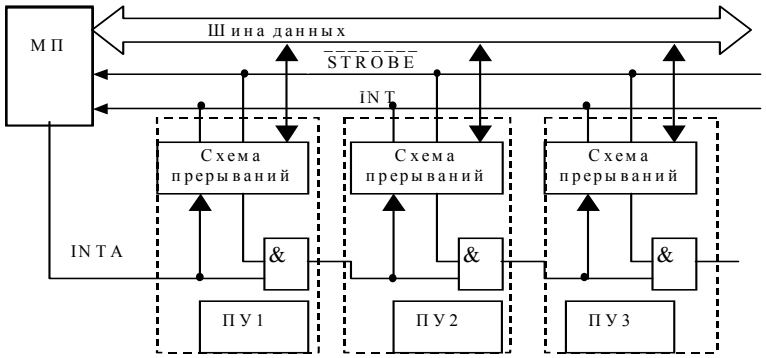


Рис.3.5.Аппаратный полинг

Приоритет ПУ задается их физической близостью к микропроцессору по линии INTA. Затем активное ПУ передает по шине данных свой адрес (вектор прерывания), сопровождая его импульсом STROBE. Этот адрес имеет однозначное соответствие с начальным адресом подпрограммы обслуживания прерывания данного устройства.

Время реакции микропроцессора на запрос прерывания определяется временем распространения сигнала INTA в цепочке и намного превышает времени реакции относительно программного полинга, но он требует дополнительные аппаратные средства для определения приоритета, а также для формирования адреса подпрограммы обслуживания.

Адрес (вектор прерывания), возвращаемый прерывающим устройством, обычно встраивается в интерфейсную плату, и его с помощью перемычек и переключателей может изменять пользователь. Приоритет ПУ определяется размещением его интерфейсной платы в разъеме, занимающее фиксированное положение в схеме.

Векторные приоритетные прерывания

Одним из распространенных способов организации системы прерывания является векторное приоритетное прерывание. Этот способ реализуется с помощью дополнительных аппаратных средств в виде интерфейсных БИС и называются контроллерами прерываний.

Система прерывания при такой организации имеет несколько уровней прерываний (линий запросов прерываний) к которым могут подключаться различные устройства, причем каждый уровень имеет свой вектор прерывания.

Для реализации векторного приоритетного прерывания необходимо распределить приоритеты между уровнями и обеспечить механизм взаимодействия подпрограмм обслуживания прерывания соответствующих уровней.

Распределение приоритетов между уровнями могут реализовываться различными способами.

Одним из этих способов является векторное прерывание с фиксированным приоритетом. Каждому уровню (входу) запросов прерывания присваивается фиксированный приоритет в порядке возрастания. Запрос с более высоким приоритетом прерывает обслуживание прерывания с меньшим приоритетом.

Вторым способом является векторное прерывание с циклическим приоритетом. Каждому входу, как и в предыдущем случае, присваивается фиксированный приоритет. После запроса прерывания и выполнения соответствующей подпрограммы обслуживания приоритеты изменяются в круговом порядке таким образом, что последний обслуженный вход будет иметь низший приоритет.

Этот способ характерен для таких применений, в которых ПУ имеют одинаковый приоритет и ни одному из них нельзя отдать предпочтения.

Реализация векторного прерывания с адресуемым приоритетом аналогична второму способу, но допускает программное определение уровня запросов прерывания, которому назначается низший приоритет.

Запрещение прерывания на время обслуживания любого ПУ может привести к потере запросов прерываний высокоприоритетных устройств, появляющихся при обслуживании устройств с меньшими приоритетами. Для исключения такой ситуации возникает необходимость использования механизма вложения прерываний, который позволит обеспечить взаимодействие подпрограмм обслуживания соответствующих уровней между собой.

Прерывание подпрограмм обслуживания прерываний называется вложенным прерыванием. На рис.3.6. показан процесс вложения прерываний.

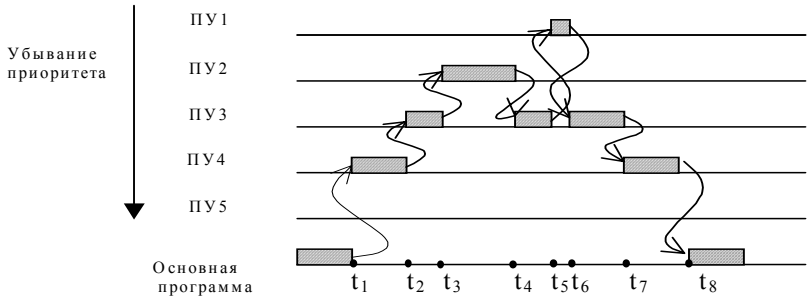


Рис.3.6. Вложение прерываний

В момент t_1 запрашивает обслуживание устройство $ПУ_4$ и микропроцессор переходит на его подпрограмму обслуживания. В свою очередь, эта подпрограмма прерывается в момент времени t_2 запросом $ПУ_3$ с более высоким приоритетом. Подпрограмма обслуживания $ПУ_3$ в момент времени t_3 прерывается запросом $ПУ_2$ с еще более высоким приоритетом, и по завершении обслуживания $ПУ_2$ в момент времени t_4 управление возвращается к продолжению обслуживания $ПУ_3$. В интервале t_5-t_6 аналогичным образом обслуживается запрос $ПУ_1$ с максимальным приоритетом, после чего управление последовательно возвращается к прерванным подпрограммам обслуживания $ПУ_3$ и $ПУ_4$. Наконец, в момент t_8 возобновляется выполнение основной программы. Для того, чтобы

процессор реагировал на запросы прерываний в начале каждой подпрограммы обслуживания их необходимо разрешать командой EI.

Для организации вложенных прерываний в каждой подпрограмме обслуживания прерываний необходимо выполнить следующие действия:

- разрешить прерывание;
- временно запомнить приоритет прерванной программы;
- загрузить в схему приоритетных прерываний новый текущий приоритет;
- собственно обслужить прерывание;
- восстановить прежний приоритет;
- возобновить прерванную программу с помощью команды RTI.

Приоритетное прерывание программ

При определении приоритета прерывания программ различают два его значения: приоритет между запросами прерывания и приоритет между прерывающими программами.

Приоритет между запросами прерываний устанавливает очередность восприятия запросов, поступивших одновременно от различных уровней. Такой приоритет может быть реализован в виде аппаратного полинга или способами, которые указаны выше при организации векторного приоритетного прерывания.

Приоритет между прерывающими программами устанавливает старшинство в выполнении прерывающих программ разных уровней. Он определяет фактический порядок, в котором программы будут использоваться. Обычно этот приоритет реализуется как программно-управляемый на основе маски прерывания.

Маска прерывания представляет собой двоичный код, разряды которого поставлены в соответствие уровням или источникам прерывания. Маска загружается командой программы в регистр маски. Состояние "1" в данном разряде регистра маски разрешает, а "0" запрещает (маскирует) прерывание текущей программы от запросов соответствующего уровня или источника прерывания. Порядок расположения разрядов в регистре маски и нумерация уровней не имеют значения. Для каждой прерывающей программы может быть установлена своя маска. И, наконец, программа, изменяя биты в регистре маски, может устанавливать произвольные приоритетные соотношения между уровнями с любыми номерами без коммутации линий, по которым поступают запросы прерывания.

Таким образом, используя различные виды и способы организации прерываний, можно строить гибкие в то же время сложные системы многоуровневых прерываний, которые позволят подключать значительное число периферийных устройств с различным быстродействием.

Контрольные вопросы

1. Назовите основные отличительные черты программного ввода-вывода.
2. За счет чего повышается скорость передачи данных в режиме прямого доступа к памяти?
3. Почему уменьшается производительность процессора в режиме ПДП?
4. В каких случаях используют ввод-вывод в режиме прерывания ?
5. Укажите основные действия, производимые микропроцессором при прерываниях?
6. Что такое контекстное переключение микропроцессора ?
7. Назовите основные характеристики системы прерывания ?
8. Укажите основные виды прерываний ?
9. Что определяет вектор прерывания и как он формируется ?
10. Укажите основные способы идентификации прерывающего устройства ?
11. Укажите достоинства и недостатки программного и аппаратного полинга ?

12. Определите основные факторы влияющие на организацию векторного приоритетного прерывания ?

13. Какие существуют способы реализации векторного приоритетного прерывания ?

14. При каких случаях используют механизм вложения программ ?

15. Какие действия осуществляются при реализации механизма вложения прерываний?

16. Укажите основное достоинство приоритетного прерывания программ ?

17. Что такое маска прерывания и с какой целью он вводится ?

18. С какой целью вводят многоуровневые прерывания ?

Глава 4

ОДНОКРИСТАЛЬНЫЕ МИКРОЭВМ СЕМЕЙСТВА МК48

4.1. Семейство МК48

Семейство МК48 включает ряд моделей ОМЭВМ, функциональный состав и технические характеристики которых отражают как различие в идеологическом подходе к применению ОМЭВМ, так и прогресс технологии СБИС. Все модели, входящие в семейство МК48, являются полностью совместимыми по системе команд, назначению и разводке выводов, совокупности основных функциональных устройств из базового набора семейства.

Первое поколение отечественных ОМЭВМ семейства МК48 - БИС КМ1816ВЕ48 и КР1816ВЕ35 являются функционально-конструктивными аналогами БИС соответственно 8748 и 8035 фирмы Intel выполнены по n-канальной МОП - технологии, что обусловило следующие ограничения: уровень интеграции до 18 тыс. транзисторов на кристалле, частота следования тактовых сигналов - 6,0 МГц, объем внутренней памяти ОЗУ - 64 байта, ППЗУ - 1 Кбайт и минимальное время цикла - 2,5 мкс.

Второе поколение - БИС КР1816ВЕ49, КР1816ВЕ39 (аналоги БИС 8049 и 8039 фирмы Intel) выполнено по n-канальной МОП - технологии с пропорциональным масштабированием, что позволило повысить уровень интеграции до 36 тыс. транзисторов на кристалле, частоту следования тактовых сигналов до 11,0 МГц, увеличить объем ОЗУ до 128 байт, ПЗУ - до 2Кбайт и снизить минимальное время цикла до 1,36мкс.

Третье поколение семейства МК48 - БИС ОМЭВМ серии К1830: КР1830ВЕ48, КР1830ВЕ35 (аналоги БИС 80С48, 80С35 фирмы Intel) выполнено по КМОП - технологии, что позволило на порядок уменьшить ток потребления по сравнению с БИС КМ1816ВЕ48, КР1816ВЕ35, сохраняя значения остальных параметров.

ОМЭВМ КМ1816ВЕ48, КР1816ВЕ35, КР1830ВЕ48 и КР1830ВЕ35 полностью идентичны в части структурной реализации. При этом, в БИС КМ1816ВЕ48 программная память размещается во внутреннем ППЗУ с ультрафиолетовым стиранием, а в БИС КР1830ВЕ48 - во внутреннем ПЗУ масочного типа. Таким образом, оперативность программирования ППЗУ позволяет использовать ОМЭВМ КМ1816ВЕ48 при создании контроллеров единичных экземпляров или мелкосерийных изделий. Потребители БИС КР1830ВЕ48 лишены такой возможности, так как программирование ПЗУ осуществляется в процессе изготовления БИС по данным "прошивки" заказа потребителя.

В микросхемах КР1816ВЕ35 и КР1830ВЕ35 в отличие от БИС КМ1816ВЕ48, КР1830ВЕ48 память программ реализуется только за счет подключения внешней памяти любого типа (ОЗУ, ППЗУ, ПЗУ) общим объемом до 4 Кбайт. Эта особенность позволяет использовать их в качестве отладочного варианта, когда память программ реализуется в ОЗУ, что позволяет легко модифицировать отлаживаемые программы. ОМЭВМ КР1816ВЕ49 и КР1816ВЕ39 имеют одну и ту же структуру, одинаковые схемотехнические решения и технические характеристики, за исключением памяти программ. ОМЭВМ КР1816ВЕ49 имеет внутреннюю память программ объемом 2Кбайт, выполненную в виде масочного ПЗУ,

а модель КР1816ВЕ39 используется только с внешней памятью программ. Реализация программной памяти КР1816ВЕ49 в виде ПЗУ обусловливает целесообразность применения этих ОМЭВМ только для изделий средне- и крупносерийного производства, что обеспечивает в этом случае низкую стоимость ОМЭВМ. В качестве отладочной модели, а также при разработке единичных экземпляров изделий целесообразно использовать ОМЭВМ КР1816ВЕ39 с внешней программной памятью.

В общем виде, основные отличительные особенности ОМЭВМ семейства МК48 представлены в таблице 4.1.

Таблица 4.1

Микросхемы	Аналог	Объем внутр. памяти прог-рамм, байт	Тип памяти программ	Объ-ем памя-ти дан-ных, байт	f_{max}, мГц	Ток пот-ребления, мА
КР1816ВЕ35	8035	Нет	внеш.	64	6,0	135,0
КР1816ВЕ48	8748	1К	УФППЗУ	64	6,0	135,0
КР1816ВЕ39	8039	Нет	внеш.	128	11,0	110,0
КР1816ВЕ49	8049	2К	ПЗУ	128	11,0	110,0
КР1830ВЕ35	80С35	Нет	внеш.	64	6,0	8,0
КР1830ВЕ48	80С48	1К	ПЗУ	64	6,0	8,0

В ОМЭВМ предусмотрена возможность расширения памяти программ до 4Кбайт, памяти данных до 384 байт и увеличения числа линий ввода-вывода за счет подключения внешних кристаллов памяти программ, ОЗУ и БИС интерфейсов.

Ориентация ОМЭВМ на преимущественное применение в системах управления отразилась на структуре и функциональных характеристиках отдельных устройств. Так, например, процессорное устройство по эффективности вычислительных операций и способов адресации уступает 8-разрядному микропроцессору КР580ВМ80А, однако, оно реализует ряд логических операций над отдельными разрядами аккумулятора и портов ввода-вывода, что повышает его эффективность при выполнении алгоритмов управления. Наличие трех 8-разрядных портов ввода-вывода Р0, Р1, Р2 в ОМЭВМ решает проблему расширения как памяти программ, так и памяти данных, а также обеспечивает возможность обмена информацией с периферией.

С точки зрения архитектурных особенностей регистры общего назначения (РОН), косвенная адресация, программные прерывания и асинхронный ввод-вывод, страничная адресация памяти не представляют принципиально новых архитектурных решений. Конструктивной особенностью ОМЭВМ является принцип сведения к минимуму количества разбросанных по кристаллу регистров, реализация которого повлекла за собой образование 8-уровневого стека и регистров общего назначения на общем адресном пространстве ОЗУ как сверхоперативной памяти.

Программист, работающий с ОМЭВМ семейства МК48, располагает двумя программно переключаемыми банками регистров общего назначения, каждый из которых содержит по 8 регистров и расположен на общем адресном пространстве ОЗУ. Нулевой банк РОН0 занимает адреса 0 ... 7, первый банк РОН1 занимает адреса 24 ... 31. Регистры активного в данный момент банка РОН прямо адресуются большим количеством инструкций. Кроме того, все ячейки ОЗУ, включая стек, адресуются косвенно с использованием нулевого или первого регистров банка РОН.

Ячейки ОЗУ с адресами 8 ... 23 могут использоваться в качестве стека с глубиной 8, каждому уровню которого соответствуют два байта. При обращении к подпрограмме один байт представляет младшие разряды адреса возврата, второй - содержит четыре старших

разряда адреса возврата, а остальные полбайта сохраняют четыре старших разряда слова состояния программы.

При объеме ОЗУ 64*8 разрядов, пользователь, например у КР1816ВЕ48, имеет два 8-разрядных регистровых банка с прямой адресацией регистров. Также он имеет 32*8 - разрядную резидентную память данных с косвенной адресацией, если используется вся глубина стека, или 56*8-разрядную память данных с косвенной адресацией, если используется только один регистровый банк и не используется стек.

Доступ к внешней памяти данных осуществляется с помощью команд MOVX A,@R и MOVX @R,A, которые передают данные между аккумулятором и внешней памятью данных с обращением по адресу, содержащемуся в регистрах - указателях R0 и R1 ОЗУ. Таким образом, может быть адресовано 256 ячеек в дополнение к резидентной памяти ОЗУ. Для работы с внешней памятью данных используется порт P0 ОМЭВМ.

Мощность ввода-вывода ОМЭВМ семейства МК48 может быть оценена следующим образом:

- число входов - 3 * 8 + 3 = 27;
- число двунаправленных выводов - 3 * 8 = 24;
- число выводов с возможностью фиксации данных на них - 3 * 8 = 24;
- число выводов, управляющих вводом-выводом - 4.

Порты P1, P2 обладают сходными возможностями в части фиксации. Данные статически присутствует на выводах порта, и могут быть изменены только с новой выдачей.

При работе с внешней памятью программ функциональная нагрузка на порт P2 увеличивается, так как четыре младших разряда порта используются для расширения адреса обращения к памяти программ. Аналогично загруженность порта возрастает при работе с дополнительными портами ввода-вывода, которые могут быть подключены к ОМЭВМ посредством четырех младших разрядов порта P2. Порт P1 и старшие разряды порта P2 работают как статический порт независимо от режима использования ОМЭВМ.

При работе с внешней памятью программ и устройствами ввода-вывода 8-разрядная шина данных (порт P0) представляет собой двунаправленный порт с синхронным стробированием. Порт P0 может действовать в трех различных режимах: как статический порт ввода-вывода в автономном режиме; как двунаправленный порт-расширитель адреса/данных в любом режиме и как выход младших разрядов адреса/инструкции/констант при использовании внешней памяти программ.

При помощи команд ввода-вывода адресуются все порты ОМЭВМ, а также четыре дополнительных порта ввода-вывода, которые можно реализовать на дополнительных микросхемах. Предусмотрена возможность выполнения логических операций И и ИЛИ над содержимым дополнительных портов и младшими четырьмя разрядами аккумулятора. Кроме трех 8-разрядных портов ввода-вывода на кристалле имеется три вывода специального назначения, которые могут программно проверяться и один из которых можно использовать для организации прерываний.

Входы тестирования и прерывания позволяют программам разветвляться без необходимости загрузки данных через порт в аккумулятор.

При возникновении прерывания ОМЭВМ переходит на выполнение программы обработки прерывания, начальный адрес которого зафиксирован в памяти программ, как в автономном режиме, так и в режиме с внешней памятью. Адрес возврата и слово состояния программы (частично) заносятся во внутренний стек и, следовательно, восстанавливаются после окончания выполнения подпрограммы. Команда RETR разрешает дальнейшие прерывания. Если прерывание запрещено, линия прерывания опрашивается как входная линия для обнаружения запроса. Процессор не вырабатывает специального сигнала "Подтверждение прерывания", однако, для этого можно использовать команды установки разрядов на линиях ввода-вывода, то есть выдачу такого сигнала можно организовать программно.

Характерной особенностью архитектуры ОМЭВМ является наличие на кристалле

таймера, предназначенного для прерывания по его переполнению. Таймер можно использовать также в качестве счетчика внешних событий по одной из линий ввода с переходом к подпрограмме обслуживания прерывания по заданному числу событий.

В дополнение к такому значительному количеству линий ввода-вывода ОМЭВМ обладает мощным набором команд, ориентированных на внешний обмен.

Внутренняя синхронизация, с делением внешней частоты на три, вырабатывает частоту процессора. Тактовые сигналы процессора с делением частоты на пять поступает на счетчик машинных циклов.

4.2. Структурная организация ОМЭВМ

Структурная организация ОМЭВМ показана на рис.4.1. Организацию ОМЭВМ далее будем рассматривать на примере КР1816ВЕ48. Основу структуры микроконтроллера образует внутренняя двунаправленная разделяемая 8-битная шина данных. Она объединяет АЛУ, устройство управления и синхронизации, память, порты ввода-вывода и счетчик-таймер.

ОМЭВМ конструктивно выполнен в корпусе БИС с 40 внешними выводами. Все выводы электрически совместимы с элементами ТТЛ, входы представляют собой единичную нагрузку, а выходы могут быть нагружены одной ТТЛ-нагрузкой.

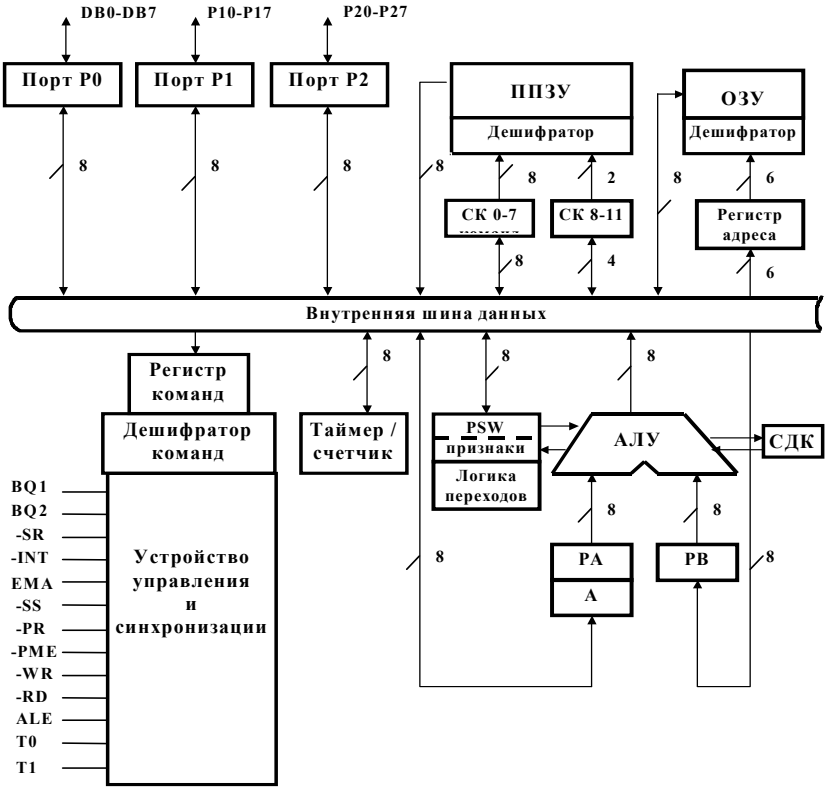


Рис.4.1. Структурная схема ОМЭВМ

Функциональное назначение сигналов ОМЭВМ показано в таблице 4.2. В первой колонке представлены номера выводов микросхемы БИС, во второй – символические имена выводов аналога 8748, в третьей –символические имена КР1816ВЕ48, а в четвертой дается краткое пояснение назначения выводов.

Таблица 4.2

Номер вывода	Обозначение (Intel)	Обозначение (МК48)	Назначение
1	T0	T0	Тестируемый вход T0 используется при выполнении команд условного перехода JT0 и JNT0; после выполнения команды ENT0 CLK используется как выход тактовых сигналов; в КМ1816ВЕ48 - как вывод при программировании и проверке УФППЗУ.
2	XTAL1	BQ1	Выводы для подключения кварцевого резонатора или LC – цепи
3	XTAL2	BQ2	
4	$\overline{\text{RESET}}$	$\overline{SR}$	Сброс
5	$\overline{\text{SS}}$	$\overline{\text{SS}}$	Пошаговое выполнение команд
6	$\overline{\text{INT}}$	$\overline{\text{INT}}$	Прерывание
7	EA	EMA	Переключение на режим работы с внешней памятью.
			Разрешение:
8	$\overline{RD}$	$\overline{RD}$	Чтения внешней памяти данных
9	$\overline{\text{PSEN}}$	$\overline{PME}$	Чтения внешней памяти программ
10	$\overline{WR}$	$\overline{WR}$	Записи во внешнюю память данных
11	ALE	ALE	Фиксации адреса
12...19	D0...D7 (BUS)	DB0...DB7	Шина данных (порт P0)
20	V_{SS}	GND	Общий вывод
21...24 35 38	P20 ... P27	P20...P27	Порт P2
25	$\overline{\text{PROG}}$	$\overline{\text{PR}}$	Выход строба для расширителя ввода-вывода. В микро-схеме КМ1816ВЕ48 используется при программировании УФППЗУ
26	V_{DD}	U_{dd}	Напряжение питания РПЗУ для КМ1816ВЕ48. Напряжение резервного питания ОЗУ для КР1816ВЕ49/ВЕ39. Вход установки режима микропотребления для КР1830ВЕ48/ВЕ35
27...34	P10...P17	P10...P17	Порт P1
39	T1	T1	Тестируемый вход 1 используется при выполнении команд условного перехода JT1 и JNT1 и как вход счетчика внешних событий после выполнения команд STRT CNT
40	V_{CC}	U_{CC}	Напряжение питания.

4.2.1. Процессор

Основу процессора ОМЭВМ составляет 8-разрядное арифметико-логическое устройство (АЛУ), позволяющее выполнять арифметические, логические операции и операции сдвига над данными, представленными в двоичном коде, а также обрабатывать данные, представленные в двоично-десятичном коде.

В состав АЛУ входят собственно арифметико-логическое устройство, аккумулятор, регистр аккумулятора, регистр временного хранения, схема десятичной коррекции аккумулятора.

Аккумулятор (А) представляет собой 8-разрядный регистр, предназначенный для записи и хранения данных, подаваемых с внутренней шины. Результат выполнения операции АЛУ всегда заносится через шину в аккумулятор. Выход аккумулятора связан с входом регистра аккумулятора (РА) - 8-разрядного регистра, предназначенного для записи и хранения одного из операндов, над которыми производятся операции в АЛУ. Сигналы с выхода РА непосредственно подаются на вход первого операнда АЛУ.

Регистр временного хранения (РВ) представляет собой 8-разрядный регистр и предназначен для записи и хранения второго операнда при выполнении операций в АЛУ. Вход РВ связан шиной данных и с выходом ПЗУ констант. Сигналы с выхода РВ непосредственно подаются на вход регистра второго операнда АЛУ.

Схема десятичной коррекции (СДК) предназначена для обработки данных, представленных в двоично-десятичном коде. В состав схемы СДК входят узлы анализа содержимого полубайтов (4 разряда старших или младших) аккумулятора, триггер основного и дополнительного переносов и схема формирования корректирующей поправки. Схема десятичной коррекции содержит также ПЗУ констант 00, 06, 60, 66 (шестнадцатеричный код), которые в зависимости от значения младшего и старшего полубайтов и основного и дополнительного переносов подаются через регистр РВ на вход АЛУ.

В состав процессора входят также счетчик команд, дешифратор и регистр команд, регистр состояния программы и схема условных переходов.

Счетчик команд (СК) предназначен для формирования текущего адреса местонахождения команды в памяти программ. Счетчик команд содержит 12 разрядов. Содержимое СК увеличивается после выбора каждого байта команды на единицу. Содержимое СК может изменяться скачкообразно при выполнении команд передачи управления, CALL, RET, RETR и при реализации прерываний. Старший разряд СК изменяется только программно (команды SEL MB0, SEL MB1). Счетчик разбит на две части: счетчик младших разрядов (биты 0 ... 7) и счетчик старших разрядов (биты 8 ... 11). Биты 0 ...7 передаются при адресации через внутреннюю шину на вход дешифратора адреса ППЗУ, а биты 8 ... 9 - непосредственно со счетчика на дешифратор адреса. При использовании внешней программной памяти биты 0 ... 7 СК поступают через порт Р0 (выводы DB0 ... DB7), а биты 8 ... 11 - через порт Р2 (выводы Р20 ... Р23).

Дешифратор команд представляет собой программируемую логическую матрицу, на вход которой поступает код команды с регистра команд, в котором осуществляется запись и хранение кода команды. С выхода дешифратора команд снимаются управляющие сигналы, осуществляющие выполнение этой команды.

Регистр состояния программы (PSW) предназначен для хранения данных о состоянии микроЭВМ. Назначение разрядов PSW (формат PSW показан на рис.4.2) следующее:

- разряды 0...2-разряды указателя стека (SP=S0...S2);
- разряд 3 не используется (при чтении всегда "1");
- разряд 4-разряд, указывающий используемый банк рабочих регистров общего назначения (BS);
- разряд 5-флаг пользователя (F0), используется по команде условного перехода;
- разряд 6-разряд дополнительного переноса (АС), используется по команде десятичной коррекции;
- разряд 7-перенос (С), указывающий на переполнение аккумулятора после предыдущей операции.

7	6	5	4	3	2	1	0
C	AC	F0	BS	1	S2	S1	S0

Рис.4.2. Формат слова состояния процессора (PSW)

Регистр PSW может программно проверяться, модифицироваться полностью или поразрядно. При прерываниях по входу INT и по флагу таймера-счетчика содержимое четырех разрядов (4 ... 7) автоматически заносится в стек, а при возврате из программы обработки прерывания содержимое этих разрядов восстанавливается. После загрузки в стек содержимое указателя стека инкрементируется, а перед извлечением из стека декрементируется. При переполнении стека указатель стека переходит из состояния 7 в состояние 0.

Схема условных переходов предназначена для формирования сигналов управления ветвлением программы при выполнении команд условных переходов. Условия перехода определяются:

- содержимым аккумулятора (возможна проверка на "0" или не "0");
- содержимым бита аккумулятора (проверка BIT на "1");
- состоянием флага переноса C (проверка на "0" и на "1");
- состояниями флагов пользователя F0 и F1 (проверка на "1"). Сигнал системного сброса SR сбрасывает флаги F0 и F1;
- триггером таймера-счетчика TF (проверка на "1");
- сигналами на входах ОМЭВМ T0 и T1 (проверка на "0" и на "1");
- сигналом на входе ОМЭВМ INT (проверка на "0").

4.2.2. Организация памяти программ

Память программ предназначена для хранения и считывания команд, которые поступают в процессор и управляют процессом обработки информации. Общий объем адресуемой памяти программ ОМЭВМ составляет 4Кбайт, при этом, память разделена на две части: резидентная программная память объемом 1024 байт и внешняя программная память, составляющая в сумме с резидентной памятью 4Кбайт. Отличия ОМЭВМ внутри семейства МК48, связанные с объемом памяти, приведены в таблице 4.1.

Если адрес выборки команды выходит за пределы резидентной памяти, то автоматически инициализируется внешняя память. Все выборки из внутренней памяти не сопровождаются никакими внешними сигналами, генерируемыми ОМЭВМ, кроме сигнала ALE, который вырабатывается независимо от режима использования ОМЭВМ и является идентификатором машинного цикла. При обращении к внешней памяти программ содержимое 12-разрядного счетчика команд выводится на 8-разрядную шину данных (порт P0) и четыре младших разряда порта P2. Сигнал ALE задним фронтом фиксирует выставленный адрес. Сигнал стробирует выборку байта из внешней памяти программ. Байт из внешней памяти программ принимается в ОМЭВМ через шину данных (порт P0). Память, расположенная на кристалле микросхемы, занимает адресное пространство от 0000H до 03FFH. Карта распределения памяти программ представлена на рис. 4.3.

Рис.4.3. Карта распределения памяти программ

Все поле адресов от 000H до 0FFFH разбито на два банка - банк 0 с адресами от 000H и до 07FFH и банк 1 с адресами от 0800H и до 0FFFH. Счетчик команд ОМЭВМ содержит 12 бит, но инкрементируются в процессе счета только младшие 11 бит. При последовательном счете счетчик команд из состояния 7FFH перейдет в состояние 000H. Переключение программной памяти с одного банка на другой осуществляется по командам SEL MB0 и SEL MB1 и непосредственно связано со старшим разрядом программного счетчика. Устанавливается этот разряд по первой команде JMP или CALL, следующей за командой SEL MB0 или SEL MB1. Память программ делится не только на два банка емкостью 2Кбайт, но и на страницы емкостью по 256 байт в каждой. В командах условного перехода задается 8-битный адрес передачи управления в пределах текущей страницы.

Для доступа к памяти программ как к таблицам данных служат команды обращения к текущей странице памяти программ MOVP, и к третьей странице - MOVP3. Эти команды позволяют считывать байт из программной памяти в аккумулятор (См. табл.3).

В памяти программ имеется три ячейки специального назначения: адрес 0 - адрес, по которому выполняется первая выборка инструкции по сигналу сброса SR; адрес 03 - начальный адрес подпрограммы, вызываемой по сигналу прерывания ОМЭВМ при условии, что прерывание разрешено; адрес 07 - начальный адрес подпрограммы, вызываемой по переполнению таймера-счетчика при условии, что прерывание разрешено.

4.2.3. Организация памяти данных

Память данных служит для записи, хранения и считывания данных, получаемых в процессе обработки информации.

Память данных, содержащий 64 ячейки ОЗУ, разбит на два банка регистров общего назначения (РОН) с адресами от 00H до 07H - банк РОН 0 и с адресами от 18H до 1FH - банк РОН 1.

Карта распределения памяти данных изображена на рис.4.4. Переключение банков осуществляется программным путем с помощью команд SEL RB0, SEL RB1. Ячейки ОЗУ с адресами от 20H до 3FH используются только как ОЗУ данных. Восьмиуровневый 16-разрядный стек с адресами от 08H до 17H адресуется указателем стека из PSW.

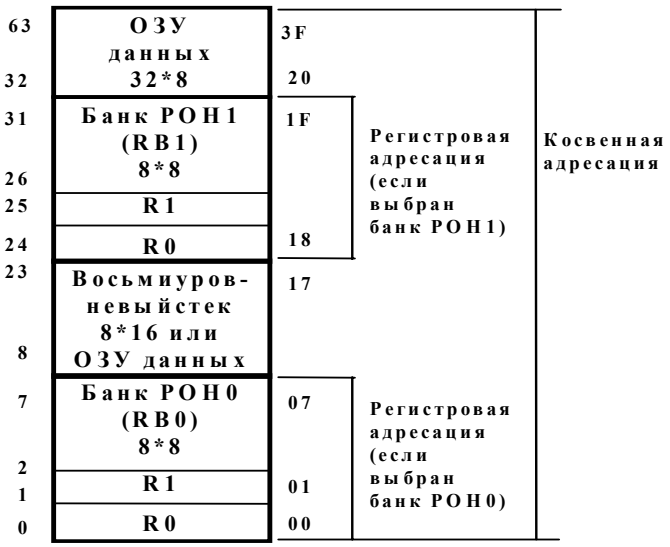


Рис.4.4. Карта распределения памяти данных

Организация стека показана на рис.4.5 (адрес ОЗУ приведен в десятичном коде). Кроме того, с использованием косвенной адресации ячейки стека могут адресоваться как ОЗУ данных. ОМЭВМ семейства МК48 не имеют специальных команд загрузки байта в стек или его извлечения из стека.

Для записи и выборки данных из ОЗУ применяются три вида адресации: регистровая, непосредственная и косвенная. Независимо от типа адресации три младших разряда кода команды указывают один из восьми регистров РОН R0 ... R7 (REG) с учетом принадлежности к ранее выбранному банку регистров. При использовании команд с регистровой адресацией указанный регистр является источником или приемником данных, а при использовании команд с косвенной адресацией указанный регистр содержит адрес данных (в качестве регистров косвенного адреса используются только R0 и R1). С помощью косвенной адресации можно адресоваться к любой ячейке памяти данных. Программист по своему усмотрению может заносить данные для хранения в любые ячейки-регистры банков РОН, стек, а также имеет доступ к любой из ячеек ОЗУ посредством косвенной адресации.

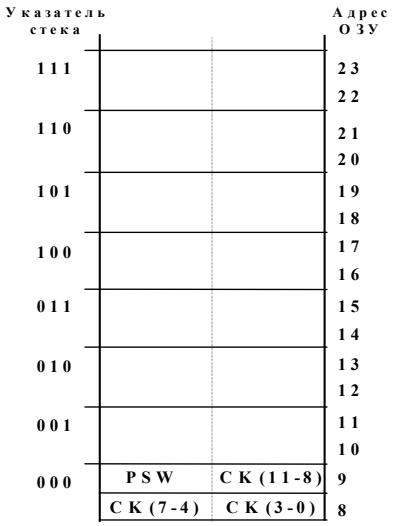


Рис.4.5. Организация стека

В ОМЭВМ предусмотрена возможность расширения внутренней памяти данных до 320 путем подключения микросхем ОЗУ. Обращение к внешней памяти данных осуществляется с помощью команд MOVX @R,A; MOVX A,@R. Обмен информацией с внешним ОЗУ стробируется сигналами $\overline{WR}$ и $\overline{RD}$ и производится через шину данных (порт P0) ОМЭВМ.

4.2.4. Каналы ввода-вывода

Каналы ввода-вывода служат для организации обмена информацией ОМЭВМ с внешними устройствами. В ОМЭВМ имеется 27 линий ввода-вывода, 24 из которых объединены в три 8-разрядных порта P0, P1, P2. Порты P1, P2 в режиме вывода обладают возможностью фиксации данных в так называемых триггерах-защелках. Эти данные статически присутствуют на выводах порта и могут быть изменены только новой выдачей по команде OUTL. Каждая выдача сопровождается занесением данных в защелку порта. В состоянии ввода входная информация не изменяет состояния защелок. При использовании портов P1 и P2 в качестве входов необходимо до подачи входной информации линии портов установить в состояние высокого уровня, выдав на порт байт единиц. В это состояние выводы портов устанавливаются также после подачи сигнала $\overline{SR}$. Возможна произвольная смешанная настройка линий портов P1 и P2, когда одни линии порта работают на ввод, а другие - на вывод. Для настройки линии на режим ввода необходимо в триггер-защелку этой линии записать "1". Вводимые данные должны присутствовать на линиях порта до тех пор, пока не будут программно прочитаны. Электрическая схема одной из линий портов P1 и P2 показана на рис.4.6.

Устройство B1 предназначено для передачи содержимого защелки (D-триггер) на внутреннюю шину данных для дальнейшей модификации по командам ORL PR,#DATA или ANL PR,#DATA.

Устройство B2 обеспечивает передачу входной информации порта на внутреннюю шину данных при выполнении команд, осуществляющих ввод информации с выводов порта.

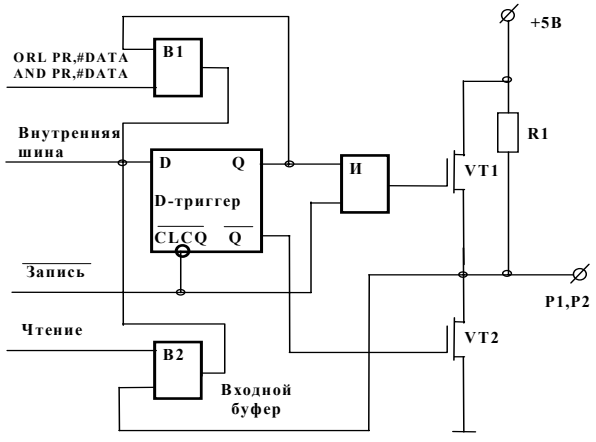


Рис.4.6. Схема одной из линий портов P1 и P2

Устройство И обеспечивает включение транзистора VT1 на время $t_{CУ}$ /6 при изменении содержимого защелки (D-триггер) с "0" на "1" для формирования фронта нарастания сигнала на выводах порта. После выключения транзистора VT1 уровень логической единицы поддерживается на выходе порта с помощью резистора R1. Сопротивление открытого транзистора VT1 составляет приблизительно 5кОм, сопротивление резистора R1 - около 50кОм. Время t_{cy} определяется по следующей формуле:

$$t_{CУ} = 15/f_{BQ1},$$

где f_{BQ1} - частота тактовых сигналов ОМЭВМ, мГц.

Для расширения адреса обращения к внешней памяти программ, а также для увеличения числа линий ввода-вывода дополнительно используются четыре младших разряда порта P2. Стробирование данных при работе с дополнительным портом осуществляется сигналом PR. Входные данные для резидентного порта должны быть поданы в состояние машинного цикла, когда осуществляется считывание, то есть между соседними сигналами ALE при выполнении команды ввода IN.

Кроме операций ввода-вывода информации, предусмотрена возможность выполнения логических операций И, ИЛИ непосредственно на портах P1 и P2 с помощью команд ANL PR,#DATA и ORL PR,#DATA.

Порт P0 - это 8-разрядный двунаправленный порт с тремя состояниями, который может использоваться в качестве статического порта ввода-вывода или двунаправленного порта адреса/данных с тремя состояниями при работе с внешней памятью.

Если порт P0 используется как статический порт, то вывод через него выполняется по команде OUTL BUS,A, а ввод - по команде INS A, BUS. Вывод сопровождается сигналом $\overline{WR}$, а ввод - сигналом $\overline{RD}$. При этом выводимые данные фиксируются в триггерах-защелках и статически выставляются на выводах порта.

В отличие от портов P1 и P2, порт P0 допускает только байтовый обмен, когда по всем линиям порта производится либо ввод, либо вывод.

При работе с внешней памятью программ через порт P0 в режиме мультиплексирования сначала выдается младший байт адреса команды, а затем синхронно с сигналом PME вводится из памяти байт команды.

При работе с внешней памятью данных через порт P0 в режиме мультиплексирования выдается адрес данных, а затем выполняется обмен байтом данных: ввод синхронно с сигналом $\overline{RD}$ либо вывод синхронно с сигналом $\overline{WR}$. Для работы с внешней памятью данных служат команды MOVX.

В режиме работы с внешней памятью, если не используется команда OUTL BUS,A, порт P0 при отсутствии передач находится в высокоимпедансном состоянии.

Команды MOVX и OUTL BUS,A могут применяться поочередно, но при этом статическая информация, выставленная на порте P0 по команде OUTL, будет разрушена последующим выполнением команды MOVX, а порт P0 перейдет в высокоимпедансное состояние.

Если порт P0 активизируется командами MOVX, то в отсутствии передач порт по своим выходам находится в высокоимпедансном состоянии.

На рис.4.7 показана электрическая схема одной из линий порта P0. Устройство B1 предназначено для передачи содержимого защелки (D-триггер) на внутреннюю шину данных для дальнейшей модификации по командам ORL BUS,#DATA или ANL BUS,#DATA. По этим командам выполняются логические операции ИЛИ, И непосредственно на порте P0. Сигналы $\overline{WR}$ или $\overline{RD}$ при выполнении этих команд не формируются.

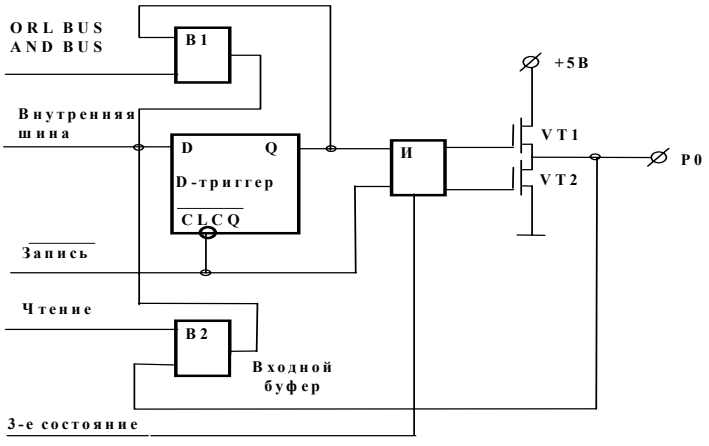


Рис.4.7. Схема одной из линий порта P0

Устройство В2 обеспечивает передачу входной информации порта на внутреннюю шину данных при выполнении команд, осуществляющих ввод информации с выводов порта.

Устройство И предназначено для управления транзисторами VT1 и VT2, обеспечивающими выдачу и прием информации. Сигнал "3-е состояние" инициирует схему И на выключение транзисторов VT1, VT2, обеспечивая тем самым высокоимпедансное состояние на выводах порта P0, необходимое при приеме информации.

Адрес и данные на P0 выдаются через устройство И при отсутствии сигнала "3-е состояние". На рис.4.7 не показан тракт выдачи адреса на порт P0. Адрес поступает на устройство И, минуя D-триггер. Таким образом, выдача адреса не разрушает информацию, записанную по команде OUTL BUS, A.

Команда OUTL BUS,A открывает порт P0, переводя его в режим выходного статического порта. Если после выполнения команды OUTL необходимо подать на выводы порта P0 информацию с внешнего устройства с целью ввода ее в ОМЭВМ, то предварительно необходимо выполнить команду MOVX для перевода порта в высокоимпедансное состояние.

Команды ANL BUS,#DATA и ORL BUS,#DATA, работая с информацией, полученной с выходов защелок (рис.4.7), не открывают порт P0 на выдачу, если он находится в высокоимпедансном состоянии. Поэтому использование этих команд имеет смысл только в том случае, если порт P0 находится в режиме выходного статического порта.

Три линии ввода-вывода (T0, T1, INT) опрашиваются при выполнении команд условного перехода для реализации ветвления программы. Линия ввода-вывода T0 переключается на выход для выдачи тактовых сигналов с частотой в три раза меньшей частоты задающего генератора по команде ENT0 CLK. Линия T1 является входом счетчика внешних событий для таймера-счетчика, если счет разрешен командой STRT CNT. Линия INT используется для внешнего аппаратного прерывания, если прерывание разрешено командой ENI.

4.2.5. Таймер-счетчик

Таймер-счетчик предназначен для подсчета внешних событий и измерения временных интервалов без участия процессора ОМЭВМ. Структурная схема включения таймера/счетчика изображена на рис.4.8.

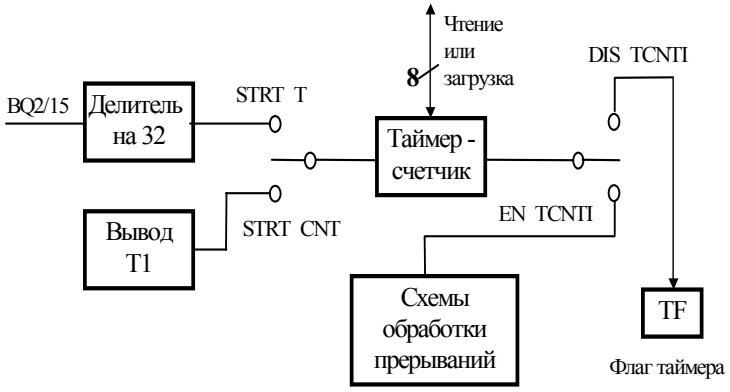


Рис.4.8. Структурная схема включения таймера/счетчика

В состав таймера-счетчика входят делитель частоты 1:32, суммирующий счетчик и триггер флага переполнения.

8-разрядный двоичный таймер-счетчик устанавливается в начальное состояния с помощью команды MOV T, A, по которой содержимое A передается в таймер- счетчик. Счетчик останавливается (но не сбрасывается) по сигналу $\overline{SR}$ или по команде STOP TCNT и остается в этом состоянии, пока не поступит команда STRT T - запуск таймера или команда STRT CNT - запуск счетчика событий.

Содержимое таймера-счетчика может быть прочитано командой MOV A,T, как в то время, когда таймер-счетчик остановлен, так и во время счета. При этом момент чтения не может попасть на межразрядный перенос, вызвав тем самым неопределенность считанного отсчета, так как инкрементирование таймера-счетчика и чтение его содержимого в аккумулятор выполняются в разных состояниях машинного цикла ОМЭВМ.

Момент перехода счетчика из состояния максимального значения FFH в состояние 00H свидетельствует о переполнении таймера-счетчика (T - C) и фиксируется в триггере флага TF и триггере переполнения T, в результате вырабатывается внутренний запрос на аппаратное прерывание. Этот запрос так же, как и внешний запрос прерывания, запоминается в триггере текущего прерывания. Прерывания по переполнению таймера-счетчика считаются внутренними и могут быть разрешены или запрещены соответственно командами EN TCNTI и DIS TCNTI независимо от внешних прерываний. Если внутренние прерывания разрешены, переполнение счетчика вызовет переход к подпрограмме обслуживания прерывания от таймера-счетчика, адрес-вектор которой соответствует ячейке 07 памяти программ. Если запросы на прерывание от таймера-счетчика и внешнего источника приходят одновременно, произойдет переход к подпрограмме обслуживания внешнего прерывания, адрес-вектор которой соответствует ячейке 03 памяти программ.

Поскольку запрос на прерывание от таймера-счетчика зафиксирован, он будет обрабатываться сразу после возвращения из подпрограммы обслуживания внешних прерываний. Запрос на прерывание от таймера сбрасывается командой DIS TCNTI. Однако, эта команда не действует на триггер флага TF таймера-счетчика, который может быть опрошен по команде условного перехода JTF.

После перехода счетчика из состояния FFH в состояние 00H счетчик будет продолжать счет, если не выполнена команда STOP TCNT.

По команде STRT CNT начинается счет событий (событие - переход сигнала на внешнем выводе T1 от высокого к низкому уровню) путем инкрементирования значения текущего состояния таймера-счетчика. Подсчет событий прекращается или по команде STOP TCNT, или при подаче сигнала $\overline{SR}$.

При выполнении команды STRT CNT разрешается передача состояния с входа T1 на вход счетчика. С этого момента каждый переход сигнала на выводе T1 от высокого к низкому уровню вызывает приращение счетчика. Минимально допустимый период следования сигналов на входе T1 составляет 3tcy; минимально допустимая длительность сигналов низкого и высокого уровней составляет соответственно tcy и tcy/5; tcy - длительность машинного цикла, рассчитываемая по формуле: $tcy = 15/f_{BQ1}$, где f_{BQ1} - частота генератора тактовых сигналов ОМЭВМ, мГц.

В результате выполнения команды STRT T внутренние синхросигналы поступают на вход счетчика и запускают его как таймер. Синхросигналы образуются путем деления на 32 частоты основных синхронизирующих импульсов машинного цикла ALE. Делитель сбрасывается во время выполнения команды STRT T. Инкрементирование счетчика осуществляется с частотой внутренней синхронизации. Путем предварительной установки счетчика и фиксации переполнения можно получить задержки до 256*32 периода сигнала ALE. Задержки большей длительности могут быть получены путем накопления переполнений счетчика под управлением программы.

Для временной задержки длительностью менее 32 периодов следования сигнала ALE на вход T1 можно подавать синхроимпульсы нужной частоты и работать со счетчиком в режиме подсчета событий.

4.2.6. Система прерываний

Процессор ОМЭВМ имеет одноуровневую систему прерываний по адресу-вектору, который воспринимает запросы на прерывания от внешнего источника или от внутреннего таймера-счетчика. Запрос внешнего прерывания поступает на вывод INT. Несколько источников прерываний можно объединить по схеме ИЛИ. Прерывания могут быть избирательно разрешены или запрещены с помощью команд ENI и DISI. По команде ENI внешние прерывания разрешаются и воспринимаются при подаче сигнала низкого уровня на вход INT. При поступлении команды запрещения внешних прерываний DIS I или после подачи сигнала $\overline{SR}$ ОМЭВМ не реагирует на поступление низкого уровня на входе INT.

Разрешение прерываний по переполнению таймера-счетчика обеспечивается выполнением команды EN TCNTI. Переполнение таймера-счетчика инициализирует последовательность обработки прерывания. Запрещение прерывания по переполнению таймера-счетчика событий происходит по команде DIS TCNTI.

Внешнее прерывание обладает более высоким приоритетом, то есть, если запросы на прерывание от внешнего источника и от таймера-счетчика возникают одновременно, внешнее прерывание обслуживается в первую очередь. При поступлении запроса на прерывание процессор (после завершения всех циклов текущей команды) передает управление по адресу 03 при внешнем прерывании, либо по адресу 07 при прерывании по переполнению таймера-счетчика. При этом, как и при выполнении команды CALL, обеспечивается загрузка в стек текущего значения 12-разрядного счетчика команд и четырех старших разрядов слова состояния процессора. Механизм прерывания от таймера- счетчика можно использовать как источник внешнего прерывания. Для этого необходимо загрузить в счетчик с помощью программы значение FFH и установить режим счета внешних событий. В этом случае переход сигнала на входе T1 ОМЭВМ из состояния "1" в состояние "0" вызовет прерывание с обращением по адресу ячейки 07 памяти программ.

Ячейка 03 памяти программ соответствует начальному адресу области памяти программ, где хранится программа обработки внешнего прерывания. Обычно по адресу 03 находится команда безусловного перехода на подпрограмму обслуживания. Завершает процедуру обслуживания прерывания выполнение команды RETR.

Поскольку в данной ОМЭВМ реализована одноуровневая система прерываний, то обслуживание вновь поступающих прерываний будет происходить после обработки текущего прерывания. Обработка очередного прерывания может начаться только после завершения второго машинного цикла команды RETR. Сигнал INT запроса на прерывание, которое обслуживается в данный момент, должен быть снят до исполнения команды RETR, в противном случае процессор начнет повторное обслуживание данного запроса прерывания.

Подпрограмму обработки любого прерывания нельзя прервать до команды RETR.

Если для хранения программы требуется объем памяти, превышающий 2Кбайт, то при разработке программ обслуживания прерываний пользователю необходимо учитывать особенности управления памятью программ, присущие данной ОМЭВМ. Процессор ОМЭВМ может прямо адресовать только 2048 байт памяти программ. Для расширения объема адресуемой памяти до 4696 байт предусмотрен механизм переключения банков памяти программ с использованием команд SEL MB0, SEL MB1: (0 ... 2047) байт - банк 0; (2048 ... 4096) байт - банк 1. Однако, его нельзя использовать при обработке прерываний, так как при выполнении перехода или вызова подпрограммы в одиннадцать младших разрядов счетчика команд (СК) (A0 ... A10) производится загрузка адреса из команды, 12-й разряд (A11) загружается из триггера переключения банка памяти DBF, указывающего номер банка памяти программ.

В ОМЭВМ при обслуживании прерываний старший разряд (A11) СК, независимо от состояния триггера DBF, аппаратно устанавливается в "0" всякий раз, когда происходит переход к подпрограмме обработки прерываний. Поэтому все программы обработки прерываний и все подпрограммы, вызываемые ими, должны располагаться в пределах нулевого банка памяти программ (ячейки 0 ... 2047). До выхода из подпрограммы обслуживания прерываний (то есть до исполнения команды RETR) никакая команда не может установить в "1" старший разряд СК. Выполнение команд SEL MB0 и SEL MB1 во время обработки прерываний не рекомендуется, так как эти команды, изменяя состояние триггера DBF, не изменяют значения старшего разряда СК до тех пор, пока происходит обработка текущего прерывания.

4.2.7. Устройство управления и синхронизации

Устройство предназначено для выработки сигналов, обеспечивающих управление выполнением команд, и реализовано на кристалле ОМЭВМ, за исключением источника опорной частоты, в качестве которого можно использовать кварцевый резонатор, LC-цепь или внешний источник синхроимпульсов. Устройство управления и синхронизации состоит из генератора, формирователей тактовых сигналов и формирователей сигналов состояний и режимов работы.

Встроенный генератор представляет собой схему с последовательным резонансом и высоким коэффициентом усиления, обеспечивающую работу в диапазоне 1 ... 6 МГц. Внешний вывод BQ1 является входом каскада усиления, а BQ2-выходом. Кварцевый резонатор или LC-цепочка присоединяется между выводами BQ1 и BQ2 и обеспечивают обратную связь и фазовый сдвиг, необходимые для генерации. Для получения временных соотношений с высокой степенью точности необходимо использовать кварцевые резонаторы. Если не требуется высокое быстродействие и высокая точность задания частоты, можно использовать LC-цепочку. Внешние синхросигналы подаются на входы BQ1 и BQ2, как показано на рис.4.9.

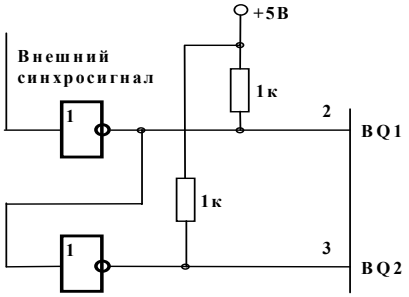


Рис.4.9. Схема включения при подаче внешних синхросигналов

Частота генератора f_{BQ1} делится на три в счетчике состояния для получения частоты процессора (CLK), которая определяет временные соотношения в ОМЭВМ. Сигналы CLK могут быть выведены на внешний вывод T0 по команде ENT0 CLK. Вывод сигналов на внешний вывод T0 блокируется сигналом $\overline{SR}$.

Частота процессора CLK делится на пять в счетчике циклов для получения частоты, определяющего машинный цикл и состоящего из пяти состояний машины S1...S5. Полученный в результате такого деления опорной частоты синхросигнал называется ALE и используется при работе ОМЭВМ с внешней памятью. Этот сигнал выдается в каждом машинном цикле на выводе ALE независимо от того, выполняется или нет в данном машинном цикле обращение к внешней памяти.

Общий сброс осуществляется благодаря наличию встроенного триггера Шмитта и соединенного с источником питания +5В резистора. Они в сочетании с внешним конденсатором емкостью 1мкФ, подключенным к выводу SR, как показано на рис.4.10, обеспечивают при включении ОМЭВМ импульс системного сброса низкого уровня

достаточной длительности для гарантированной установки в исходное состояние всех цепей. Если импульс сброса подается извне, то он должен быть активным не менее 50мс после того, как источник питания установится в номинальное значение.

Подача активного импульса системного сброса производит инициализацию системы, которая осуществляет следующие действия:

- сбрасывает счетчик команд и указатель стека;
- устанавливает порт P0 в высокоимпедансное состояние (при EMA=0), а порты P1 и P2 - в режим ввода;
- выбирает банк RB0 и банк MB0;
- запрещает прерывания по входу INT и переполнению таймера;
- останавливает таймер - счетчик и выдачу синхросигнала на вывод T0;
- сбрасывает триггер флага таймера - счетчика TF и флаги пользователя F0 и F1. □

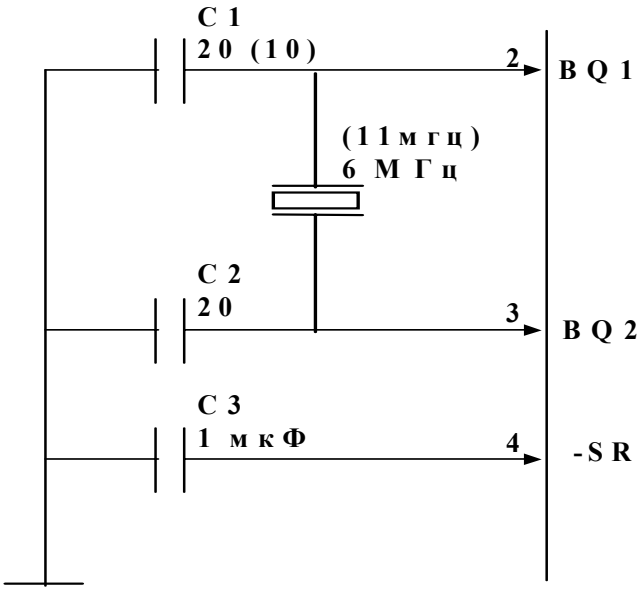


Рис.4.10. Схема формирования сигнала системного сброса

Если EMA=1 при действии активного сигнала системного сброса, то порт P0 (BUS) находится в неопределенном состоянии. В этом случае нельзя считать, что порт P0 находится в высокоимпедансном состоянии.

4.3. Система команд

Использование ОМЭВМ в качестве специализированного вычислителя, встраиваемого в контур управления объектом или процессом, определило особенности реализации системы команд. Эти особенности ориентируют систему команд на повышение эффективности алгоритмов управления. Помимо традиционных основных операций в ОМЭВМ введены специализированные команды ввода-вывода с использованием маскирования, позволяющие манипулировать отдельными битами; команды, осуществляющие операции с битами; сканирование таблиц с передачей управления.

ОМЭВМ оперирует с командами четырех типов, которые показаны на рис.4.11. Как видно из рисунка 4.11 все команды имеют формат в один или два байта и выполняются за один или два машинных цикла. В системе команд ОМЭВМ реализованы три способа адресации: регистровая (тип 1), непосредственная (тип 2) и косвенная (тип 1) адресации. Формат команды третьего типа определяют команды передачи управления и вызова подпрограмм по длинному адресу, а четвертый тип – команды передачи управления по условию с коротким адресом. Все команды с непосредственным операндом, команды ввода-

вывода, команды управления, вызова подпрограмм и некоторые команды пересылок выполняются за два машинных цикла, а остальные – за один.

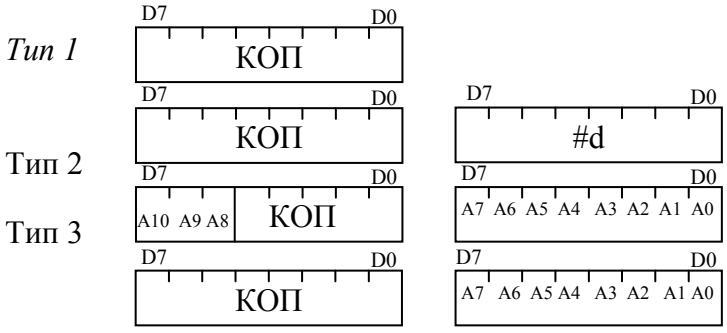


Рис. 4.11 Форматы команд

По функциональному признаку команды можно разбить на шесть групп: команды пересылок и обмена данными; команды арифметических и логических операций и операций над признаками; команды ввода-вывода данных; команды передачи управления и вызова подпрограмм; команды управления таймером; команды управления режимом работы ОМЭВМ.

Команды пересылок и обмена данными

Команды приведены в таблице 4.3. Все команды пересылки данных выполняются с использованием аккумулятора. Данные могут пересылаться между аккумулятором и рабочими регистрами (R0-R8) каждого банка РОН с регистровой адресацией. Например: MOV A, R4- передать содержимое регистра в аккумулятор ((R4) → (A)).

Обмен с ячейками внутреннего ОЗУ осуществляется с использованием косвенных адресов, находящихся в регистрах R0 или R1, выбранного банка РОН. Так команда MOV A,@R0 осуществляет загрузку в аккумулятор содержимого ячейки внутренней памяти, адрес которой задан в регистре R0 (((R0)) →(A)). Кроме того, регистры R0 и R1 являются регистрами косвенных адресов, предназначенные для работы с внешней памятью.

Команда MOVX A,@R1 производит загрузку в аккумулятор содержимого ячейки внешней памяти, адрес которой задан в регистре R1 (((R1)) →(A)).

Таблица 4.3

Мнемоника	Код Двоичный	Время в циклах	Содержание
MOV A,#DATA	00100011 dddddddd	2	DATA→ (A)
MOV A, PSW	11000111	1	PSW→ (A)
MOV A, REG	11111REG	1	(REG)→ (A)
MOV A, @R	1111000R	1	((R))→ (A)
MOV A, T	01000010	1	(T)→ (A)
MOV PSW, A	11010111	1	(A)→ (PSW)
MOV REG, A	10101REG	1	(A)→ (REG)
MOV REG, #DATA	10111REG dddddddd	2	DATA→ (REG)
MOV @R, A	1010000R	1	(A)→ ((R))
MOV T, A	01100010	1	(A)→ (T)
MOVP A,@A	10100011	2	(A)→ (CK_{0-7})

			((CK)) → A
MOVP3 A,@A	11100011	2	(A)→ (CK_{0-7}) 0011→(CK_{8-11}) ((CK)) → A
MOVX A, @R	1000000R	2	((R))→ (A)
MOVX @R,A	1001000R	2	(A)→ ((R))
XCH A,REG	00101REG	1	(A) ↔(REG)
XCH A,@R	0010000R	1	(A) ↔((R))
XCHD A,@R	0011000R	1	(A_{0-3}) ↔ (REG_{0-3})
SWAP A	01000111	1	(A_{0-3})↔(A_{4-7})

Команды арифметических и логических операций

Эта группа команд выполняет операции сложения, инкремента и декремента, сдвигов, конъюнкции и дизъюнкции над непосредственным операндом, содержимым аккумулятора, регистров и ячеек памяти. В ОМЭВМ отсутствует команда вычитания, что приводит к необходимости выполнения ряда операций: получение дополнительного кода, суммирование и вычисление дополнительного кода результата. Сложные операции выполняются по подпрограммам.

Арифметические команды формируют признаки результата операций. Из множества признаков результата операции программно-доступными являются только четыре признака: перенос (C), вспомогательный перенос (AC), признаки пользователя F0 и F1. Признак F0, кроме того, доступен через слово состояния процессора (см. рис 4.2). В таблице 4.4 приведен перечень команд арифметических и логических операций, а также операций над признаками.

Таблица 4.4

Мнемоника	Код двоичный	Время в циклах	Содержание
ADD A,REG	01101REG	1	(A)+(REG)→(A)
ADD A,@R	0110000R	1	(A)+((R))→(A)
ADD A,#DATA	00000011 dddddddd	2	(A)+DATA→(A)
ADDC A,REG	01111REG	1	(A)+(REG)+ +(C)→(A)
ADDC A,@R	0111000R	1	(A)+((R))+(C) →(A)
ADDC A,#DATA	00010011 dddddddd	2	(A)+DATA+(C) →(A)
DA A	01010111	1	Десятичная коррекция A
DEC A	00000111	1	(A)-1→(A)
DEC REG	11001REG	1	(REG)-1→(A)
INC A	00010111	1	(A)+1→(A)
INC REG	00011REG	1	(REG)+1→(A)
INC @R	0001000R	2	((R))+1→((R))
RL A	11100111	1	(A_n)→(A_{n+1}) (A_7)→(A_0)

RLC A	11110111	1	$(A_n)\rightarrow(A_{n+1})$ $(C)\rightarrow(A_0)$ $(A_7)\rightarrow(C)$
RR A	01110111	1	$(A_{n+1})\rightarrow(A_n)$ $(A_0)\rightarrow(A_7)$
RRC A	01100111	1	$(A_{n+1})\rightarrow(A_n)$ $(C)\rightarrow(A_7)$ $(A_0)\rightarrow(C)$
ANL A, REG	01011REG	1	$(A)\wedge(REG)\rightarrow$ $\rightarrow(A)$
ANL A, @R	0101000R	1	$(A)\wedge((R))\rightarrow$ $\rightarrow(A)$
ANL A, #DATA	01010011 dddddddd	2	$(A)\wedge DATA\rightarrow$ $\rightarrow(A)$
ORL A, REG	01001REG	1	$(A)\vee(REG)\rightarrow$ $\rightarrow(A)$
ORL A, @R	0100000R	1	$(A)\vee((R))\rightarrow$ $\rightarrow(A)$
ORL A, #DATA	01000011 dddddddd	2	$(A)\vee DATA\rightarrow$ $\rightarrow(A)$
XRL A,REG	11011REG	1	$(A)\forall(REG)\rightarrow$ $\rightarrow(A)$ $\forall$- исключающее ИЛИ
XRL A,@R	1101000R	2	$(A)\forall((R))\rightarrow$ $\rightarrow(A)$
XRL A,#DATA	11010011 dddddddd	2	$(A)\forall DATA\rightarrow$ $\rightarrow(A)$
CLR #A	00100111	1	$0\rightarrow(A)$
CLR C	10010111	1	$0\rightarrow(C)$
CLR F0	10000101	1	$0\rightarrow(F0)$
CLR F1	10100101	1	$0\rightarrow(F1)$
CPL A	00110111	1	$HE(A)\rightarrow(A)$
CPL C	10100111	1	$HE(C)\rightarrow(C)$
CPL F0	10010101	1	$HE(F0)\rightarrow(F0)$
CPL F1	10110101	1	$HE(F1)\rightarrow(F1)$

Команды ввода-вывода данных

Три встроенных порта (P1, P2, BUS) могут быть рассмотрены в случае необходимости для обеспечения связи ОМЭВМ с объектом управления, имеющим большое число датчиков и исполнительных механизмов. В этом случае, если расширение осуществляется через младшую тетраду (порт-расширитель) порта P2, то дополнительные порты имеют адреса 4-7 (порты P4, P5, P6, P7). В случае, если в качестве порта расширителя используется порт BUS, то все дополнительные порты входят в адресное пространство памяти данных и адресуются командами MOVX через регистры косвенного адреса R0 и R1.

Идентификатор PR определяет порты P1 и P2 двумя битами 01 и 10 соответственно. Идентификатор PD определяет дополнительные порты P4, P5, P6, P7, которые кодируются

тоже двумя битами 00, 01, 10, 11 соответственно. Команды ввода-вывода данных приведены в таблице 4.5.

Таблица 4.5

Мнемоника	Код Двоичный	Время в циклах	Содержание
IN A,PR	000010PR	2	(PR)→(A)
INS A,BUS	00001000	2	(BUS)→(A)
OUTL BUS,A	00000010	2	(A)→(BUS)
OUTL PR,A	001110PR	2	(A)→(PR)
ANL PR,#DATA	100110PR dddddddd	2	(PR)∧DATA→ (PR)
ANL BUS,#DATA	10011000 dddddddd	2	(BUS)∧DATA →(BUS)
ORL PR,#DATA	100010PR dddddddd	2	(PR)∨DATA→ (PR)
ORL BUS,#DATA	10001000 dddddddd	2	(BUS)∨DATA →(BUS)
MOVD A,PD	000011PD	2	$0\rightarrow(A_{4\text{-}7})$ $(PD)\rightarrow(A_{0\text{-}3})$
MOVD PD,A	001111PD	2	$(A_{0\text{-}3})\rightarrow(PD)$
ANLD PD,A	100111PD	2	$(PD)\wedge(A_{0\text{-}3})\rightarrow$ $\rightarrow(PD)$
ORLD PD,A	100011PD	2	$(PD)\vee(A_{0\text{-}3})\rightarrow$ $\rightarrow(PD)$

Команды передачи управления и вызова подпрограмм

Выполнение команд передачи управления и работы с подпрограммами в ОМЭВМ имеет свои особенности и зависит от распределения программ и данных по блокам и страницам памяти, особенно в случае использования внешней памяти. Каждый банк памяти программ разбивается на 8 страниц по 256 байтов. Команды JMP и CALL относятся к третьему типу (см. рис.4.11), где разряды A_8-A_{10} первого байта этих команд определяют номер страницы, а второй байт – адрес ячейки внутри страницы. Тогда кодирование команд JMP и CALL (16-ричный код) в зависимости от номера текущей страницы памяти программ приведено в таблице 4.6.

Таблица 4.6

Страница	JMP	CALL	Страница	JMP	CALL
0	04	14	4	84	94
1	24	34	5	A4	B4
2	44	54	6	C4	D4
3	64	74	7	E4	F4

Граница блоков памяти 2 Кбайт может быть пересечена в любом направлении только по команде перехода или вызова подпрограмм вслед за командой выбора блока памяти. Переключения блока памяти не происходит, пока не выполниться переход. До тех пор, пока

не будет выбран другой блок памяти, все переходы осуществляются только в пределах выбранного блока памяти. Например, если требуемая подпрограмма располагается в другом блоке памяти, доступ к ней осуществляется парой команд: выбор блока памяти и вызов подпрограммы. После выполнения подпрограммы для передачи управления в исходный блок памяти необходимо выполнить команду выбора исходного блока памяти. В том случае, если в подпрограмме нет этой команды, возврат в исходный блок будет для нее выполнен правильно, однако любая очередная команда перехода в исходном блоке передает управление в другой блок. Это особенность является источником большинства ошибок при программировании.

Команда передачи управления JMPP@A позволяет выполнить переход к ячейке памяти в зависимости от содержимого аккумулятора. Эта команда может эффективно использоваться для передачи управления одной из процедур в зависимости от кода символа, вводимого в аккумулятор с клавиатуры, также эта команда может реализовать механизм множественных ветвлений по программе в от результатов предыдущих вычислений.

Команда JBBIT ADDR передает управление по заданному адресу ADDR, если заданный бит аккумулятора имеет значение 1. Шестнадцатеричные коды восьми возможных команд JBBIT в зависимости от номера тестируемого бита аккумулятора выглядят следующим образом:

JB7	JB6	JB5	JB4	JB3	JB2	JB1	JB0
F2	D2	B2	92	72	52	32	12

Для организации программных циклов используется команда DJNZ REG, ADDR (декремент регистра и переход, если не нуль), которая позволяет использовать любой из восьми рабочих регистров в качестве счетчика и осуществить переход в пределах текущей страницы памяти программ.

Перечень команд передачи управления и вызова подпрограмм приведен в таблице 4.7.

Таблица 4.7

Мнемоника	Код двоичный	Время в циклах	Содержание
DJNZ REG,ADDR	11101REG aaaaaaaaa	2	(REG)-1→ →(REG), если REG≠0, то ADDR→(CK), иначе (CK)+2→(CK)
JBBIT ADDR	BIT10010 aaaaaaaa	2	BIT=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JFO ADDR	10110110 aaaaaaaa	2	F0=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JF1 ADDR	01110110 aaaaaaaa	2	F1=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JMP ADDR	aaa00100 aaaaaaaa	2	(A_{8-10})→(CK_{8-10}) (A_{0-7})→(CK_{0-7}) DBF→(CK_{11})
JMPP @A	10110011	2	((A))→(CK_{0-7})
JC ADDR	11110110 aaaaaaaa	2	C=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JNC ADDR	11100110 aaaaaaaa	2	C=0, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JTO ADDR	00110110 aaaaaaaa	2	TO=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)

JNTO ADDR	00100110 aaaaaaaa	2	TO=0, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JT1 ADDR	01010110 aaaaaaaa	2	T1=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JNT1 ADDR	01000110 aaaaaaaa	2	T1=0, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JZ ADDR	11000110 aaaaaaaa	2	(A)=0, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JNZ ADDR	10010110 aaaaaaaa	2	(A)≠0, то ADDR→(CK_{0-7}), Иначе (CK)+2→(CK)
JTF ADDR	00010110 aaaaaaaa	2	TF=1, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
JNI ADDR	10000110 aaaaaaaa	2	INT=0, то ADDR→(CK_{0-7}), иначе (CK)+2→(CK)
CALL ADDR	aaa10100 aaaaaaaa	2	(CK),(PSW_{4-7})→ →(SP) (SP)+1→(SP) A_{8-10}→(CK_{8-10}) (A_{0-7})→(CK_{0-7}) DBF→(CK_{11})
RET	10000011	2	(SP)-1→(SP) ((SP))→(CK)
RETR	10010011	2	(SP)-1→(SP) ((SP))→(CK) ((SP))→(PSW_{4-7})

Команды управления таймером

Таймер может быть (в зависимости от команды) использовать как счётчик тактов или событий от внутреннего или внешнего источника сигналов. В таблице 4.8, помимо команд обмена между таймером и аккумулятором MOV A, T и MOV T, A, приведены основные операции, производимые над таймером/счетчиком.

Таблица 4.8

Мнемоника	Код Двоичный	Время в циклах	Содержание
STRT T	01010101	1	Запуск таймера (T)
STRT CNT	01000101	1	Запуск счетчика событий (C)
STOP TCNT	01100101	1	Остановка T/C
EN TCNTI	00100101	1	Разрешение прерывание от T
DIS TCNTI	00110101	1	Запрет прерывания от T

Команды управления режимом работы

Команды управления режимом работы ОМЭВМ разрешают или запрещают обслуживание прерываний и определяют рабочий банк регистров и блок памяти. Специальной командой ENTO CLK на вывод TO разрешается передача синхронизирующего сигнала с частотой $f_{BQ1}/3$. Выдача этого сигнала может быть отключена только сигналом общего сброса $\overline{SR}$.

Синхросигнал на выходе ТО используется или для инициализации системы или для общей синхронизации внешних устройств, согласованных с ОМЭВМ по частоте работы. Перечень этих команд приведен в таблице 4.9.

Таблица 4.9

Мнемоника	Код двоичный	Время в циклах	Содержание
ENI	00000101	1	Разрешение прерывания от входа INI
DISI	00010101	1	Запрет прерывание от входа INI
SEL RB0	11000101	1	Выбор банка регистров 0
SEL RB1	11010101	1	Выбор банка регистров 1
SEL MB0	11100101	1	Выбор блока памяти 0
SEL MB1	11110101	1	Выбор блока памяти 1
ENTO CLK	01110101	1	Выдача $f_{BQ1}/3$ на вход ТО
NOP	00000000	1	Нет операции

4.4. Режимы работы

Микросхемы семейства МК48 могут работать в следующих режимах, с помощью которых осуществляется управление работой ОМЭВМ, контроль и отладка программ. Это проверка внутренней памяти программ, программирование внутреннего ППЗУ, работа с внутренней памятью программ и внутренней памятью данных, работа с внешней памятью программ и внешней памятью данных, пошаговое выполнение команд, режим уменьшенного энергопотребления только для КР1816ВЕ39, КР1816ВЕ49 и режим микропотребления для КР1830ВЕ35, КР1830ВЕ48.

Режимы работы устанавливаются комбинацией входных и выходных сигналов.

Режим проверки внутренней памяти программ

Режим проверки внутренней памяти программ используется при контроле правильности информации, занесенной в память в процессе ее программирования или изготовления, а также при контроле "чистоты" памяти после ее стирания. Под "чистотой" памяти понимается нахождение всех ячеек памяти после стирания в состоянии низкого порогового напряжения, которое обеспечивает на выводах DB0...DB7 состояние низкого уровня. Для ОМЭВМ процесс проверки внутренней памяти программ включает в себя следующие операции: инициализация (сброс) микросхемы; задание режима проверки внутренней памяти программ; задание адреса; фиксация адреса; чтение данных.

Выводы микросхем ЕМА, Т0, $\overline{SR}$, Р20...Р22 в данном режиме работы ОМЭВМ являются входами. Шина DB0 ... DB7 сначала работает как входной порт, на который подаются младшие адреса выбираемой ячейки памяти, а затем переходит в режим выходного порта, на котором выставляется записанная в выбранной ячейке памяти информация. Работой шины DB0 ... DB7 управляет сигнал Т0: при Т0="0" шина DB0 ... DB7 работает как входной порт, а при Т0="1" переводится в режим выходного порта. При ЕМА=”1” после перехода сигнала Т0 из высокого в низкий логический уровень шина DB0 ... DB7 не позднее чем через 200нс переходит в высокоимпедансное состояние.

Подаваемые на одноименные выводы микросхемы сигналы выполняют следующие функции: низкий уровень сигнала $\overline{SR}$ при ЕМА= “1” обеспечивает инициализацию микросхемы; ЕМА - при подаче напряжения высокого уровня активизирует режим обращения к внутренней памяти программ; Т0 - обеспечивает режим контроля при

логической единице; $\overline{SR}$ - при переходе из низкого логического уровня в высокий фиксирует выбранный адрес при логическом нуле на выводе T0; DB0 ... DB7 - обеспечивают подачу адреса A0 ... A7 и выдачу данных; P20 - P22 - обеспечивают подачу адреса A8, A9 (A10 для КР1816ВЕ49) .

Для работы ОМЭВМ в режиме проверки внутренней памяти программ требуется внутренняя синхронизация, т.е. к выводам ОМЭВМ BQ1, BQ2 должны быть подключены внешние элементы внутреннего генератора или поданы внешние синхроимпульсы.

Выводы ОМЭВМ ЕМА, $\overline{SR}$, P20, P21 являются входами; вывод ALE - выход. Шина DB0 ... DB7 может работать в режиме входного порта, на который подают восемь младших разрядов адреса A0 ... A7 выбираемой ячейки ПЗУ, или в режиме выходного порта, с которого считывают байт информации, записанной в выбранной ячейке ПЗУ. Управление при этом производится сигналом $\overline{SR}$: если ЕМА= “1”, то при $\overline{SR}$ ="0" шина DB0 ... DB7 переходит в высокоимпедансное состояние и работает как входной порт, а при $\overline{SR}$ ="1" шина DB0 ... DB7 с некоторой задержкой, переводится в режим выходного порта. На входы ОМЭВМ P20, P21 подают два старших разряда адреса A8, A9 выбираемой ячейки ПЗУ.

Если нарастающим фронтом сигнала ALE стробируется низкий уровень сигнала $\overline{SR}$, то по спаду ALE шина DB0 ... DB7 уже гарантировано находится в высоко-импедансном состоянии и допустима подача адреса. Адрес с шины DB0 ... DB7 лучше всего убирать одновременно с переходом сигнала $\overline{SR}$ с низкого на высокий логический уровень.

При организации режима контроля памяти следует иметь в виду, что: по выводам DB0 ... DB7 осуществляется подача сигналов адреса и выдача данных для контроля. Поэтому при переходе к режиму контроля необходимо обеспечить высокоимпедансное состояние на выводах DB0 ... DB7, которое исключает попадание на открытые выводы схемы напряжения адресных сигналов, поступающих от источника адресных сигналов. При проверке "чистоты" памяти выходное напряжение на выводах DB0 ... DB7 должно соответствовать напряжению низкого уровня.

Режим работы с внутренней памятью программ

Режим работы с внутренней памятью программ устанавливается при подключении вывода ЕМА к выводу GND. Выполнение программы начинается с команды по адресу 00 после сброса. При работе с внутренней памятью программ никакие внешние управляющие сигналы, за исключением ALE, ОМЭВМ не формируются. Если микросхема имеет внутреннюю память программ и вывод ЕМА подключен к выводу GND, то обращения к этой памяти выполняются автоматически до тех пор, пока адрес обращения не превышает максимальный адрес внутренней памяти программ микросхемы. Если адрес обращения превышает максимальный адрес внутренней памяти программ, то ОМЭВМ автоматически переходит в режим работы с внешней памятью программ. С целью увеличения производительности ОМЭВМ предусмотрено совмещение выполнения внутренних операций в цикле. Например, выполнение выбранной из памяти команды и подготовка следующего адреса команды производятся одновременно. Для синхронизации внешних устройств ввода-вывода можно использовать сигнал ALE, выдаваемый ОМЭВМ в каждом машинном цикле.

Режим работы с внешней памятью

Наряду с наличием возможности работы ОМЭВМ с внешней памятью программ в качестве дополнения к внутренней памяти, имеется режим работы исключительно с внешней памятью программ, который обеспечивается путем подачи напряжения от источника +5 В на вывод ЕМА. Для микросхем КР1816ВЕ35, КР1816ВЕ39, КР1830ВЕ35, не имеющих внутренней памяти программ, этот режим является единственно возможным, и для

этих микросхем вывод EMA всегда должен быть подключен к +5В. Для микросхем КМ1816ВЕ48, КР1816ВЕ49, КР1830ВЕ48 в этом режиме внутренняя память программ отключается и, начиная с нулевого адреса, все обращения выполняются к внешней памяти программ.

Временная диаграмма работы ОМЭВМ с внешней памятью программ показана на рис.4.12. Заштрихованные участки временной диаграммы указывают интервалы, в которых шина DB находится в высокоимпедансном состоянии.

Содержимое двенадцати разрядов счетчика команд выводится в качестве адреса обращения к внешнему ЗУ команд на шину данных DB0 ... DB7 (разряды 0 ... 7 счетчика команд) и на выводы P20 ... P23 порта P2 (разряды 8 ... 11 счетчика команд), адрес внешнего ЗУ фиксируется по окончании сигнала ALE; шина данных переходит в режим ввода и процессор принимает 8-разрядное слово команды, выборка команды из внешнего ЗУ фиксируется по окончании сигнала PME. После окончания времени действия адреса на выводах P2 (0 - 3) выставляется информация, находящаяся в соответствующих защелках порта P2, которая будет находиться на выводах порта P2 до следующего обращения к памяти программ (до следующей выдачи адреса).

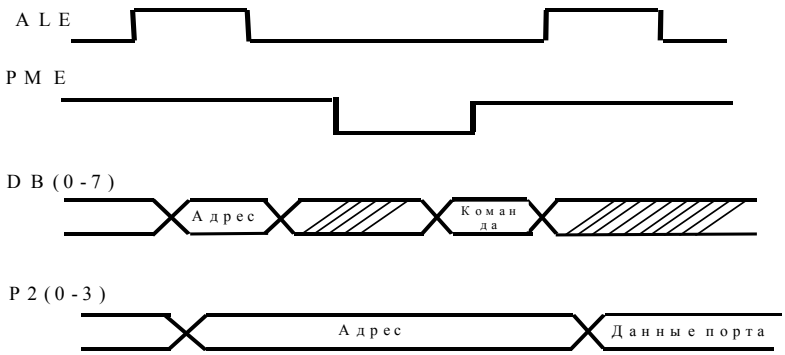


Рис.4.12. Временные диаграммы работы микросхем с внешней памятью программ

В ОМЭВМ используется механизм переключения банков памяти программ по 2Кбайт каждый. Выбор банка определяется содержимым старшего разряда счетчика команд, который загружается содержимым триггера переключения банка памяти DBF каждый раз при выполнении команды перехода JMP или CALL при вызове подпрограммы. При инкрементировании содержимого счетчика команд в старший разряд переноса нет.

Триггер DBF может устанавливаться в "1" (выбор банка 1 - ячейки с адресами 2048 ... 4097) с помощью команды SEL MB1 и сбрасывается в "0" (выбора банка 0 - ячейки с адресом 0 ... 2047) по команде SEL MB0 или по сигналу общего сброса $\overline{SR}$.

Выбор банка 0 по команде SEL MB0 или по сигналу $\overline{SR}$ позволяет обращаться к ячейкам 0 ... 2047 исключительно внешней памяти программ, если вход EMA подключен к источнику +5В.

Команда SEL MB должна выполняться в программе перед тем, когда возникает необходимость переключения банков памяти программ. Собственно переключение осуществляется при выполнении очередной команды перехода или вызова подпрограмм. Поскольку при выполнении команды CALL все двенадцать разрядов СК, включая старший, запоминаются в стеке, пользователь может переходить к подпрограмме, расположенной за пределами текущего банка памяти программ. После окончания подпрограммы номер банка памяти, из которого она была вызвана, будет восстановлен командой возврата из подпрограммы. Следует помнить, что состояние триггера DBF не изменяется командой возврата.

Выборка из внешней памяти программ команд, расположенных по адресам, меньшим адреса 1024, осуществляется при подключении вывода EMA к источнику +5В. ОМЭВМ Р1816ВЕ35, КР1830ВЕ35 и КР1816ВЕ39 не содержат на кристалле память программ и всегда работают в режиме обращения к внешней памяти.

На рис.4.13 изображена схема подключения памяти программ к ОМЭВМ с использованием трех стандартных ППЗУ объемом 1К * 8 байт.

В соответствии с представленной схемой восемь младших бит адреса фиксируются по заднему фронту сигнала ALE в регистре-защелке RG. Выходные сигналы регистра RG и адрес на выводах ОМЭВМ Р20, Р21 образуют 10-разрядный адрес, подаваемый на адресные входы трех микросхем ППЗУ. Два старших разряда адреса с выходов ОМЭВМ Р22, Р23 подаются на адресные входы дешифратора DC.

Низкий уровень сигнала $\overline{PME}$ разрешает работу дешифратора, сигналами с выходов Q0 - Q2 которого производится разрешение работы одной из ППЗУ.

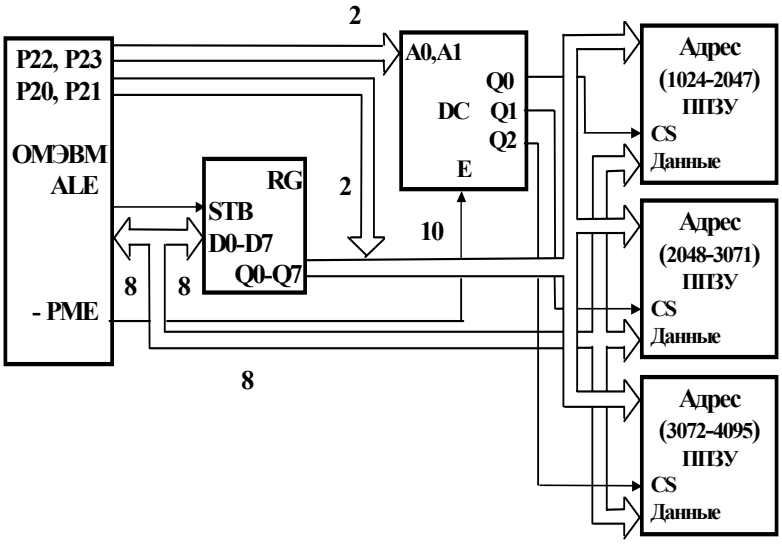


Рис.4.13. Схема подключения памяти программ к ОМЭВМ

С микросхемы ППЗУ, определенного сигналом с дешифратора DC, считывается байт в ОМЭВМ. По окончании сигнала $\overline{PME}$ низкого уровня запрещается работа дешифратора DC, и шины данных всех БИС ППЗУ переходят в высокоимпедансное состояние. Схема готова к следующему циклу чтения.

Использование регистра RG необходимо для развязки шин адреса и данных, так как порт Р0 (DB) по одним и тем же физическим линиям в режиме временного мультиплексирования сначала выдает байт адреса, а затем принимает байт команды (данных) из памяти.

Память данных можно расширять за пределы встроенной памяти с помощью двунаправленной шины данных DB, по которой передаются все адреса и данные. При этом через регистры косвенного адреса R0 и R1 возможен доступ к внешней памяти данных до 256 байт.

Временная диаграмма работы ОМЭВМ с внешней памятью данных показана из рис.4.14. Цикл чтение-запись происходит следующим образом:

1. Содержимое регистров R0 или R1 выводится на шину данных;
2. Стробирующий сигнал фиксации адреса ALE указывает, что адрес выводится на шину данных. Задний фронт сигнала ALE используется для фиксации адреса внешней памяти данных;

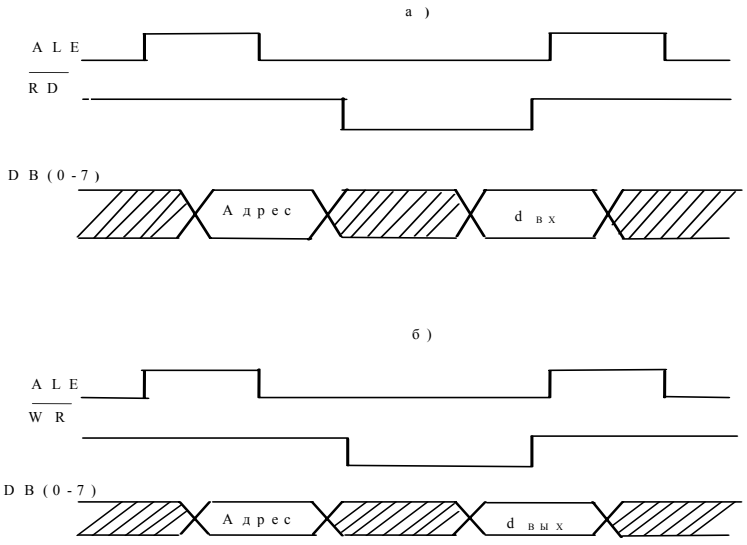


Рис.4.14. Временные диаграммы работы с внешней памятью данных: а - чтение; б - запись

Выходной строб управления считыванием $\overline{RD}$ или записью $\overline{WR}$ на соответствующих выводах ОМЭВМ показывает, какой тип доступа к памяти имеет место. Выводимые данные должны быть зафиксированы в момент прохождения заднего фронта $\overline{WR}$, а вводимые в момент прохождения заднего фронта $\overline{RD}$.

Входные и выходные данные обозначены на диаграммах на рис.4.14 соответственно dвх и dвых. Заштрихованные участки временной диаграммы указывают интервалы, в которых шина DB находится в высокоимпедансном состоянии.

Доступ к внешней памяти данных осуществляется специальными командами пересылки MOVX A,@R и MOVX @R,A, с помощью которых происходит передача 8-разрядных данных между аккумулятором и ячейкой внешней памяти, адресуемой содержимым одного из двух регистров указателей R0 или R1.

На рис.4.15 приведена схема подключения внешней памяти данных использованием двух ИС статического запоминающего устройства произвольной выборки (ЗУПВ) с организацией 256*4.

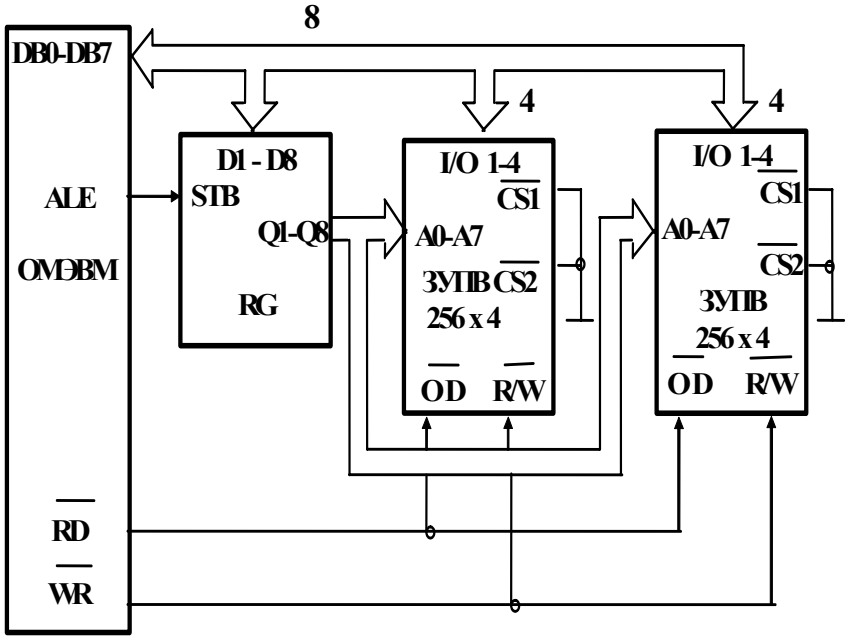


Рис.4.15. Схема подключения внешней памяти

Регистр RG используется для фиксации адреса. Каждая 4-разрядная половина шины данных DB непосредственно подключается к двунаправленной 4-разрядной шине данных ввода-вывода внешнего ЗУПВ.

Выходной сигнал ОМЭВМ $\overline{WR}$ управляет входом R/W ЗУПВ, а выходные драйверы шины данных ЗУПВ управляются по входу OD сигналом $\overline{RD}$. Если не требуется использования дополнительного ЗУПВ, вход CS (выбор кристалла) ЗУПВ присоединяется к общему выводу GND.

На рис.4.16 показан возможный вариант расширения памяти данных с использованием ЗУПВ с организацией 256*8. Предполагается, что используемая микросхема ЗУПВ имеет внутреннюю защелку адреса (регистр RG на рис.4.15.).

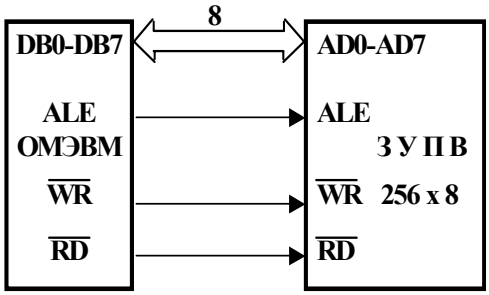


Рис.4.16. Расширение памяти данных с использованием одной микросхемы ЗУПВ

Расширение канала ввода-вывода

Шина данных ОМЭВМ совместима с 8-разрядной двунаправленной шиной микропроцессора КР580ВМ80А, что обеспечивает возможность подключения к ОМЭВМ периферийных устройств в виде микросхем серии КР580 которые можно использовать для реализации ряда дополнительных специальных функций, а также для увеличения числа каналов ввода-вывода и их типов.

На рис.4.17, 4.18 показано соединение ОМЭВМ к стандартному периферийному устройству серии КР580 - ИС КР580ВВ55А. Интегральная схема КР580ВВ55А обеспечивает три 8-разрядных программируемых канала ввода-вывода (PA, PB, PC).

Для связи ОМЭВМ с ИС КР530ВВ55А в последней используются 8-разрядная двунаправленная шина данных D0 - D7, входы $\overline{RD}$ и $\overline{WR}$ для управления чтением-записью, вход "выбор кристалла" (-CS) для активизации логики управления чтением-записью и адресные входы A0, A1 для выбора различных внутренних регистров.

Выводы ОМЭВМ DB, $\overline{RD}$, $\overline{WR}$ соединяются с соответствующими выводами ИС КР58СВВ55А. При реализации этой схемы выбирается способ, с помощью которого должны адресоваться внутренние регистры ИС КР580ВВ55А.

Если регистры адресуются как внешняя память данных с использованием команды MOVX, соответствующее число разрядов адреса (в данном случае 2) должно фиксироваться на шине с помощью сигнала ALE так же, как и при использовании внешней памяти данных (рис.4.17).

Если к шине данных ОМЭВМ подсоединяется только одно устройство ИС КР580ВВ55А, вход ИС CS заземляется.

Если используется несколько ИС КР580ВВ55А, должны фиксироваться дополнительные разряды адреса, которые используются для выбора конкретной микросхемы.

Возможно, применение другого способа адресации, при котором исключается необходимость во внешних схемах фиксации адреса и дешифраторах выбора кристалла благодаря использованию выходных линий портов ОМЭВМ непосредственно в качестве линий адресации и выбора микросхемы. При этом способе требуется вывод на выходной

порт командой OUTL адресной информации перед выполнением команды MOVX (рис.4.18).

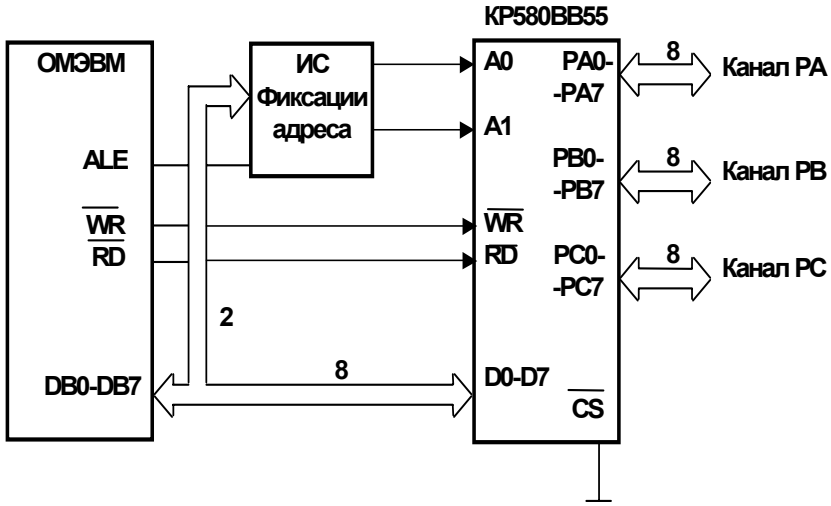


Рис.4.17. Подключение ОМЭВМ к БИС КР580ВВ55А
Вариант 1

Кроме этого, ОМЭВМ позволяет увеличить число каналов ввода-вывода за счет использования команд MOVD A, PD; MOVD PD, A; ANLD PD, A; ORLD PD, A. При этом обмен информацией осуществляется через порт Р2 (Р20 ... Р23).

Для реализации данного режима существует специальная микросхема - расширитель портов.

Схема соединения ОМЭВМ с расширителем портов - ИС КР580ВР43 показана на рис.4.19.

Расширитель портов КР580ВР43 содержит четыре 4-битных двунаправленных порта Р4, Р5, Р6, Р7, которые являются расширением резидентной системы ввода/вывода ОМЭВМ. Обращения к портам Р4 – Р7 производятся по командам MOVD, ANLD и ORLD, выполняемым совместно ОМЭВМ и расширителем. Каждая из указанных команд реализует обмен между ОМЭВМ и КР580ВР43.

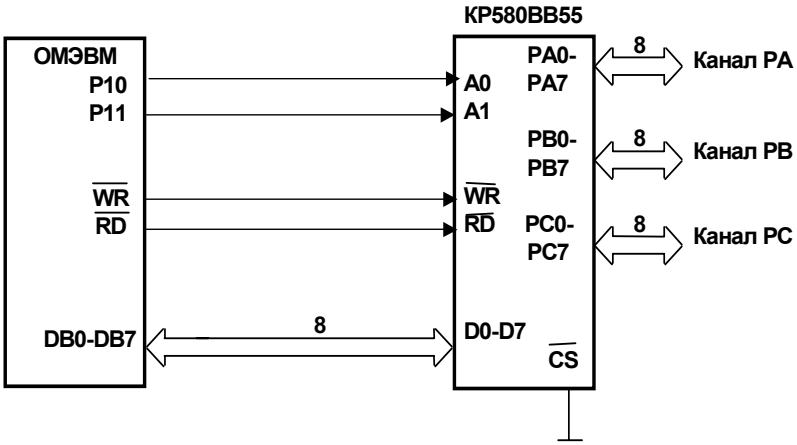


Рис.4.18. Подключение ОМЭВМ к БИС КР580ВВ55А
Вариант 2

Обмен производится по линиям Р20 ... Р23 и синхронизируется выходным сигналом ОМЭВМ –PR.. Каждый обмен состоит из двух пересылок 4-разрядных полубайтов, первый из которых содержит код управляющего слова и адрес порта Р4 - Р7, а второй - 4 бита

данных (рис.4.20). Переход сигнала -PR из состояния высокого уровня в состояние низкого уровня указывает, что на выводах P20 ... P23 находятся код управляющего слова и адрес порта, а переход сигнала - PR из состояния низкого уровня в состояние высокого уровня указывает на то, что на выводах P20 ... P23 находятся данные.
Изменить состояние сигналов на портах P4 - P7, работающих в режиме вывода, можно следующими способами: командой MOVD PD,A загрузить в требуемый порт новое значение; командой ORLD PD,A произвести логическое сложение порта с аккумулятором с загрузкой результата в порт; командой ANLD PD, A произвести логическое умножение порта и аккумулятора с загрузкой результата в порт. Логические операции выполняет КР580ВР43, а не ОМЭВМ.

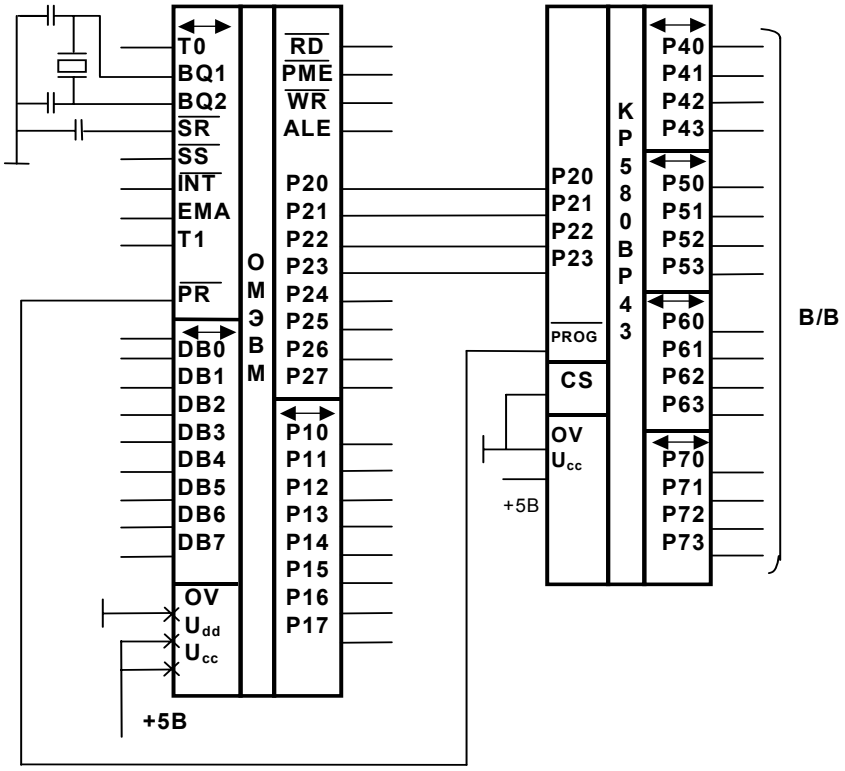
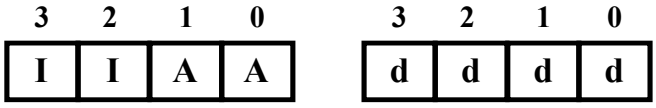


Рис.4.19. Подключение расширителя портов ИС КР580ВР43 к ОМЭВМ



Код управляющего слова

II	AA
00 Чтение	00 Порт 4
01 Запись	01 Порт 5
10 И	10 Порт 6
11 ИЛИ	11 Порт 7

Рис.4.20. Формат пересылок при обменах ОМЭВМ с КР580ВР43

Для настройки порта расширителя на ввод данных необходимо выполнить для него команду ввода MOVD A,PD. Если до выполнения этой команды порт находился в режиме вывода, то результат первого ввода является неопределенным, а все последующие команды ввода будут давать правильный результат. При выполнении операции ввода порт

настраивается на ввод, и его выходы переключаются в высокоимпедансное состояние.

Контрольные вопросы

1. Укажите, к какому классу микропроцессоров можно отнести ОМЭВМ и почему?
2. Перечислите состав и назначение основных функциональных блоков ОМЭВМ.
3. Объясните функциональное назначение управляющих сигналов ОМЭВМ.
4. Определите формат слова состояния процессора, и укажите, для каких целей он служит.
5. В чем заключается особенность организации памяти ОМЭВМ?
6. Какие типовые ошибки программирования возникают при переключении одного блока памяти на другой?
7. Объясните, каким образом реализуется стековая память в ОМЭВМ.
8. Для каких целей служат регистры общего назначения и каким образом происходит переключение одного блока РОН на другой?
9. Каким способом происходит обращение к внешней памяти данных?
10. В чем заключаются особенности организации каналов ввода-вывода ОМЭВМ?
11. Укажите основные функции порта P0.
12. В чем заключается различие между портами P1 и P2?
13. Какие существуют особенности портов P1 и P2 при вводе данных?
14. Укажите способы расширения каналов ввода-вывода.
15. Какими средствами осуществляется обращение к расширенным каналам ввода-вывода?
16. Укажите основное назначение таймера-счетчика. Каким образом происходит переключение его с одного режима на другой?
17. Какой частотой тактируется таймер-счетчик, и какую максимальную задержку можно получить аппаратным способом?
18. Как организована система прерывания в ОМЭВМ?
19. Как определяется тактовая частота процессора и на какое число состояний разбивается машинный цикл ОМЭВМ?
20. Укажите, на какие группы по функциональному признаку разбивается система команд ОМЭВМ?
21. Какие типы команд образуют систему команд ОМЭВМ? Укажите их форматы.
22. Какие методы адресации заложены в системе команд ОМЭВМ?
23. Укажите средства реализации программных циклов.
24. Определите формат команды управления и объясните назначение их полей.
25. Какими средствами можно организовать сканирование таблиц с передачей управления?

Глава 5

ОСНОВЫ ПОСТРОЕНИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ НА БАЗЕ ПЕРСПЕКТИВНЫХ МИКРОПРОЦЕССОРОВ

Развитие и совершенствование МП позволяет все в большей степени решать множество проблем, стоящих пред разработчиками и пользователями средств вычислительной техники. Наиболее важными среди этих проблем являются создание и эксплуатация надежных, высокопроизводительных вычислительных систем с минимальными затратами. Микропроцессорные средства представляют собой техническую базу для реализации множества оригинальных и эффективных теоретических архитектурных концепций.

В основу этих теоретических архитектурных концепций лежит принцип параллельной обработки информации. Различные виды распараллеливания обработки на внутри кристальном уровне позволяют значительно увеличить производительность МП. Дальнейшим шагом повышения производительности вычислительных систем является распараллеливание вычислительных процессов на системном уровне. Распараллеливание на этом уровне ведет к росту производительности системы в целом и повышению надежности за счет применения интегральных схем высокой степени интеграции, которые предполагают различные виды избыточности.

Под перспективными микропроцессорами подразумеваются 32-разрядные и выше микропроцессоры и системы ЭВМ, интегральные процессоры цифровой обработки сигналов, специализированные МП для решения задач искусственного интеллекта и микропроцессоры, которые могут быть использованы для построения параллельных вычислительных систем.

5.1. Основные направления ускорения вычислений

Одной из основных задач повышения производительности вычислительных систем является достижение высокой скорости выполнения программ. Эта цель соответствует как требованиям пользователей, заинтересованных в наиболее быстром получении результатов вычислений, так и тому обстоятельству, что быстродействие определяет общее количество вычислительной работы, которую способна выполнить система за данный отрезок времени.

Область применения методов достижения высокого быстродействия охватывает все уровни создания систем. Рассмотрим эти методы с точки зрения уровней описания цифровых систем, приведенной в первой главе.

На уровне электрических схем уменьшение времени задержки логического элемента связано с технологией изготовления сверхбольших ИС (СБИС). Это прямой путь к увеличению скорости, поскольку, если бы, например, удалось все задержки в машине сократить в k раз, то это привело бы к увеличению быстродействия в такое же число раз.

Изготовление современных 32-разрядных и выше МП стало возможным лишь благодаря появлению СБИС и достижению выхода годных кристаллов, обуславливающего приемлемость их цен. По мере совершенствования технологических процессов выход работоспособных МП будет увеличиваться, а их цены должны снижаться. Основным фактором, определяющим возможность увеличения числа транзисторов на кристалле СБИС, являются минимальные топологические размеры элементов, называемые также проектными нормами. Например, в большинстве 32-разрядных МП используются двухмикронная проектная норма, тогда как в МП предыдущего поколения эта норма составляла 4 мкМ. Такое различие определяет четырехкратное повышение плотности компоновки транзисторов. По мере уменьшения проектных норм увеличивается тактовая частота работы МП. Так, например, если в МП NS 32032 (32-разрядный МП фирмы National Semiconductor) первоначальные проектные нормы составляли 3,5 мкМ, а максимальная тактовая частота 6 мГц, то после увеличения плотности компоновки кристалла удалось повысить эту частоту до

15 мГц. Чтобы достичь проектных норм меньше 2 мкМ, необходимы перспективные методы литографии. Применение ультрафиолетовой, электронно-лучевой и рентгеновской литографии позволило внедрить в практику проектные нормы меньше, чем 1 мкМ. В настоящее время проектные нормы МП пятого и шестого поколений составляют 0,35 и 0,25 мкМ с плотностью упаковки 36,5 млн. транзисторов (Pentium Pro) с тактовой частотой 200 мГц и выше. Однако этот путь имеет ряд ограничений:

1. Для определенного уровня технологии обеспечивается определенный уровень быстродействия элементной базы и как только он оказывается достигнутым, дальнейшее увеличение быстродействие сопровождается огромными расходами вплоть до достижения того порога, за которым уже нет технологий, обеспечивающих большее быстродействие;
2. Более быстродействующие элементы обычно имеют меньшую плотность монтажа, что, в свою очередь, требуют более длинных соединительных линий и, следовательно, приводят к увеличению задержек и уменьшению выигрыша в производительности;
3. Более быстродействующие элементы рассеивают больше тепла, поэтому требуются специальные меры по теплоотводу, что еще больше снижает плотность. монтажа и, следовательно, быстродействие.

На уровне логических схем повышение быстродействия достигается уменьшением числа логических уровней при реализации комбинационных схем. На данном этапе конструкторская задача состоит в создании схем с малым числом логических уровней, которые удовлетворяли бы ограничениям по числу вентилей и их коэффициентам соединений по входу и выходу.

Операционный уровень охватывает способы реализации основных операций, таких как сложение, умножение и деление. Для того чтобы увеличить скорость выполнения этих операций, необходимо использовать алгоритмы, которые приводили бы к быстродействующим комбинационным схемам и требовали бы небольшого числа циклов. Это алгоритмы опережающего просмотра при операциях сложения, сложения с сохраняемым переносом и записи при матричном умножении, реализации деления в виде цепочки операций умножения и т.д. Далее быстродействие вычислительных систем может быть повышено за счет реализации аппаратными или аппаратно-программными средствами встроенных сложных команд, соответствующих тем или иным функциям. К таким функциям относятся, например, корень квадратный, сложение векторов, умножение матриц, быстрое преобразование Фурье и т.д.

На структурном уровне повышение быстродействия можно достигнуть за счет сокращения временных затрат при обращениях к памяти. Это достигается, во-первых, в расширении путей доступа за счет разбиения памяти на модули, обращение к которым осуществляется одновременно, во-вторых, в применении дополнительной сверхбыстродействующей памяти (КЭШ - памяти) и, наконец, в увеличении числа внутренних регистров в процессоре. Другим способом повышения быстродействия на этом уровне является уменьшение длительности исполнения одной команды за счет временного перекрытия различных ее фаз. Этот подход требует дополнительного оборудования и зависит от формата команды, поскольку именно им определяется наличие независимых фаз.

Наконец, программный уровень определяется структурой алгоритма, на основе которого работает система. На этом уровне основной подход базируется на принципе параллельной обработки информации. Этот подход отличается от того, который был реализован в обычной фон-неймановской машине, где команды исполняются строго последовательно одна за другой. Принцип параллельной обработки приводит к различным вариантам архитектуры в зависимости от способа, по которому осуществляется задание очередности следования команд и управление их исполнением.

Эти подходы касаются аппаратуры, структурной организации и архитектуры систем. Совершенствование этих направлений осуществляется с целью обеспечения необходимого ускорения вычислений на программно-аппаратном уровне. На этом уровне существуют два подхода увеличения скорости вычислений. Первый подход предполагает использование

специальных языков программирования, который предоставляет средства для явного описания параллелизма. Второй подход использует методы выявления параллелизма в последовательных программах. Кроме того, алгоритм должен обладать внутренним параллелизмом, адекватным конкретной аппаратуре.

5.2. Основные принципы организации высокопроизводительных вычислительных систем

Основным организационным принципом, ведущим к повышению быстродействия вычислительных систем, является параллельная обработка, при котором в любой момент времени выполняются одновременно более одной базовой операции. Эффективность таких систем тесно связана со структурами вычислительных алгоритмов.

Алгоритм – это описание вычислительного процесса в виде последовательности более простых вычислительных конструкций. Описание алгоритма определяется спецификациями этих конструкций и отношениями следования между ними. При этом алгоритмические структуры различаются между собой содержанием простых вычислительных конструкций и типом отношения следования. Различают два типа отношений следования – предшествования и безразличия. Вычислительная конструкция i предшествует конструкции j, если результат конструкции i является исходным значением для конструкции j или, рекуррентно, любому ему предшествующему. Иными словами, конструкция j не может быть выполнена раньше или одновременно с конструкцией i из-за отсутствия входных данных. Конструкции i и j находятся в отношении безразличия, если исходные данные для каждого из них могут быть получены вне зависимости от результатов выполнения другого. Это отношение предусматривает возможность одновременного выполнения этих конструкций, определяющих соответствующие операции. По степени сложности вычислительные конструкции могут быть объектами, представляющие как простые арифметические операторы, так и сложные (функции и процедуры). Другими словами, сложные вычислительные конструкции, образующие один уровень, могут быть описаны посредством алгоритмов, составленных из вычислительных конструкций следующего более низкого уровня.

Этот процесс характеризует вложенность алгоритма. Следовательно, алгоритм строится на принципе иерархии вычислительных конструкций, основанный на свойстве вложенности. Вложенность определяет глубину (детализацию) распараллеливания алгоритма и является одной из его важнейших свойств.

Степень вложенности алгоритма может изменяться в широком диапазоне – от мелкой детализации, где вычислительные конструкции являются простыми, до крупной, где эти конструкции достаточно сложны, т.е. реализуются с помощью другого алгоритма.

Другим аспектом представления алгоритма являются типы отношений следования, которые непосредственным образом определяют архитектуру вычислительных систем.

5.2.1. Классификация высокопроизводительных вычислительных систем

В общем случае классификацию высокопроизводительных систем можно проводить по различным признакам. Большую популярность получила классификация по архитектуре, предложенная Флинном, которая базируется на взаимосвязи команд и обрабатываемых данных. В соответствии с ней все вычислительные средства делятся на четыре класса: ОКОД, ОКМД, МКОД, МКМД.

ОКОД – одиночный поток команд, одиночный поток данных. Архитектура такой машины представлена на рис.5.1, где УУ – устройство управления, ПЭ – процессорный элемент.

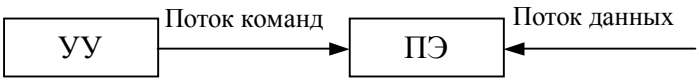


Рис.5.1. Архитектура ОКОД

Алгоритмическая структура этого класса машин показана на рис5.2.

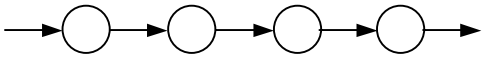


Рис.5.2. Последовательная структура алгоритма

Все вычислительные конструкции, составляющие алгоритм, находятся между собой в отношении предшествования, определяя последовательный характер алгоритма. ОКОД представляют собой обычные последовательные ЭВМ, построенные по фон-неймановскому принципу, в которой имеет место одна последовательность команд, и каждая команда арифметической обработки инициирует выполнение одной арифметической операции.

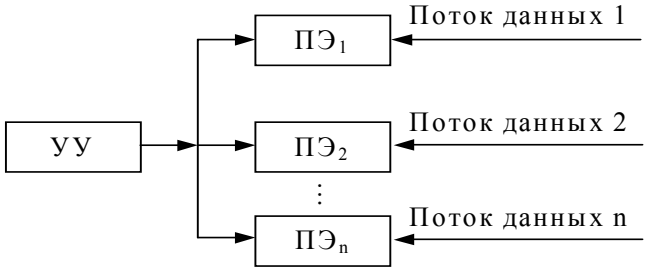


Рис.5.3. Архитектура ОКМД

Для повышения скорости обработки могут быть использованы такие средства, как конвейеризация на уровне команд и кэш-память.

ОКМД – одиночный поток команд, множественный поток данных. Архитектура данной машины показана на рис.5.3.

Для этой архитектуры характерна последовательно-групповая алгоритмическая структура (рис.5.4), основанная на параллелизме объектов.

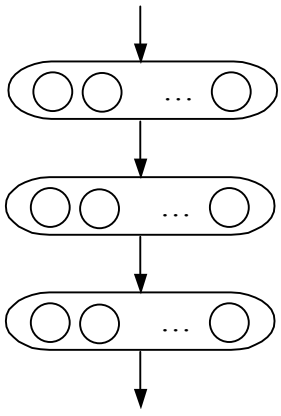


Рис.5.4. Последовательно-групповая структура алгоритма

Вычислительные конструкции, образующие алгоритм, объединены в группы, где конструкции внутри группы находятся в отношении безразличия и могут выполняться

параллельно, а сами группы – в отношении предшествования. К этому классу относятся матричные и векторные ЭВМ.

Матричные ЭВМ состоят из множества процессорных элементов, работающих под управлением одного устройства управления. Важная особенность УУ заключается в том, что на каждом такте оно предписывает исполнение одной и той же команды на всех процессорных элементах над различными элементами данных. Таким образом, все одновременно обрабатываемые элементы данных подвергаются одинаковому преобразованию. Процесс суммирования элементов двух массивов A(n)+B(n) на таких машинах проиллюстрирован на рис.5.5.

Если в матричных ЭВМ элементарная вычислительная конструкция выполняется на разных процессорах, то в векторных системах эта конструкция реализуется конвейерным процессором. В этом и заключается главное различие этих вычислительных систем.

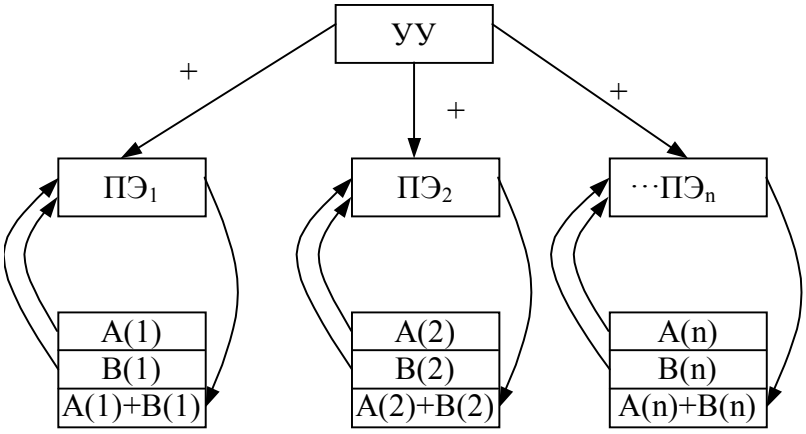


Рис.5.5. Сложение двух массивов на матричных ЭВМ

МКОД – множественный поток команд, одиночный поток данных. Взаимосвязь между командами и данными для данной архитектуры приведена на рис.5.6.

В этой архитектуре имеется множество процессоров, обрабатывающих один поток данных под управлением различных команд. Некоторые специалисты считают этот класс ЭВМ вакантным, однако, большинство из них относят к нему конвейерные системы.

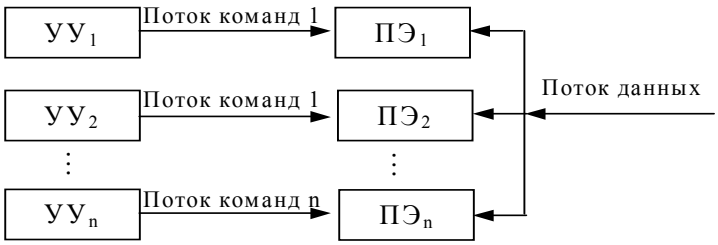


Рис.5.6. Архитектура МКОД

Конвейерные машины основаны на разделение всего алгоритма на последовательные элементарные вычислительные конструкции. Эти конструкции, в свою очередь, разделяются на последовательность некоторых более специализированных этапов, где организуется передача данных от одного этапа к последующему. Такие машины состоят из элементарных процессоров $ПЭ_i$, определяющие соответствующий этап и работающие параллельно. Данные вводятся в конвейер по шагам обработки. На каждом шаге одна операция выполняется над каждым элементом данных, в то время как различные шаги по обработке данных

выполняются последовательно. Как только конвейер будет заполнен, то после каждого шага выдается новый результат вне зависимости от числа шагов в этой цепочке.

Скорость конвейера определяется скоростью самого медленного процессора $ПЭ_i$, и поэтому для таких машин существенным является разбиение элементарных вычислительных конструкций на специализированные этапы, так как самый медленный процессор определяет тактовую частоту конвейера.

Рассмотрим простой пример сложения двух действительных чисел 13,4+7,66. При сложении этих чисел можно выделить, по крайней мере, три этапа. На первом этапе делается выравнивание порядков - т.е. сдвиг второго числа таким образом, чтобы запятые оказались на одном уровне. На втором этапе суммируются дробные части (мантиссы), при этом может образовываться единица переноса. На третьем этапе складываются целые части чисел с прибавлением единицы переноса, если она появилась. Если для каждой операции выделить отдельный процессор, то суммирование может быть изображено так, как показано на рис.5.7.

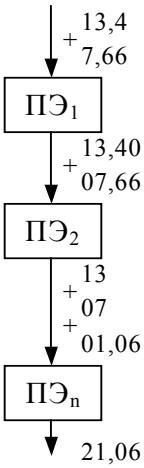


Рис.5.7. Разбиение операции сложения по процессорным элементам

Если на последовательной машине процесс сумирования будет занимать один такт времени, то на конвейере каждый из этапов занимает $1/3$ такта. При сложении одной пары чисел никакого выигрыша во времени не наблюдается, так как $1/3 + 1/3 + 1/3 = 1$.

Теперь предположим, что нам необходимо просуммировать элементы двух массивов и получить суммы A(k)+B(k), при k=8. На последовательной машине каждое сложение требует одного такта, и поэтому все суммы будут получены за 8 тактов. На конвейере через один такт на выходе процессора $ПЭ_3$ появится результат первой суммы A(1)+B(1), а результаты последующих сложений будут выдаваться через каждую треть такта. Их остается семь, поэтому общее время для суммирования восьми чисел $1 + 7\frac{1}{3} = 3\frac{1}{3}$ такта.

По сравнению с последовательной обработкой имеется выигрыш во времени более чем в два раза. В общем случае, если длина конвейера N, каждый этап требует $1/N$ такта и обрабатывается массив длины k, то время обработки будет равно 1+(k-1)/N. Таким образом, для больших массивов конвейерный процессор считает почти в N раз быстрее, чем обычный.

Преимуществом машин класса МКОД являются регулярные связи между элементами конвейера и простота программирования; однако, такие машины плохо работают там, где алгоритм обработки существенно зависит от данных.

МКМД – множественный поток команд, множественный поток данных (рис.5.8)

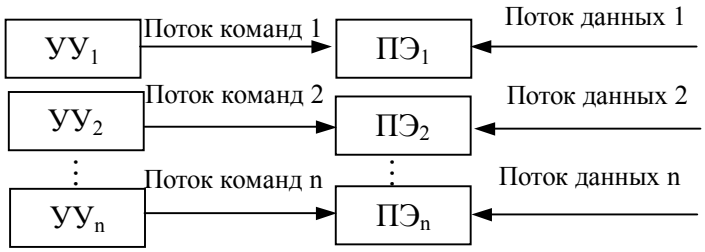


Рис.5.8. Архитектура МКМД

Алгоритмические структуры двух типов, показанные на рис.5.9, определяет данный класс машин, для которых характерен параллелизм вычислительных конструкций (команд).

Первый тип использует алгоритм, структура которых представляет собой совокупность слабосвязанных потоков команд. В этом случае программа распадается на некоторые последовательные процессы, между которыми существует

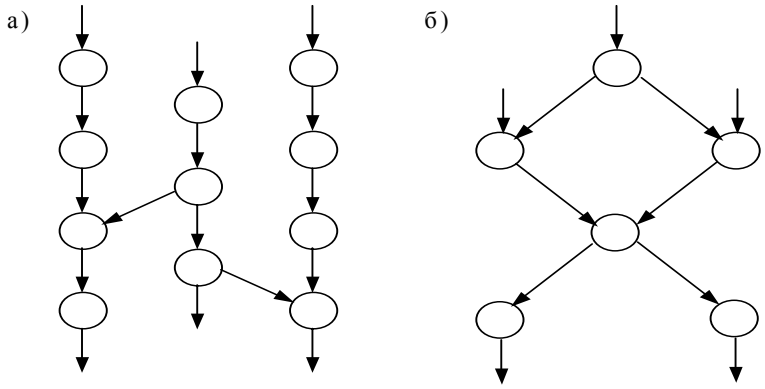


Рис.5.9. Алгоритмические структуры:
а) совокупность слабосвязанных потоков;
б) параллельная структура общего вида.

относительно небольшое число взаимосвязей. Каждый процесс может выполняться на отдельном процессоре, который при необходимости осуществляет взаимодействие с другими процессорами.

Вычислительные системы, отвечающие особенностям данной алгоритмической структуры, относятся к классическим мультипроцессорным системам. Основным вопросом мультипроцессорных систем является механизм синхронизации и взаимосвязи между процессами. Все многообразие таких механизмов можно разбить на два класса: взаимодействующие через общую память и работающие по принципу обмена сообщений.

Каждый процессор $Пр_i$ имеет свое устройство управления и локальную память и осуществляет доступ

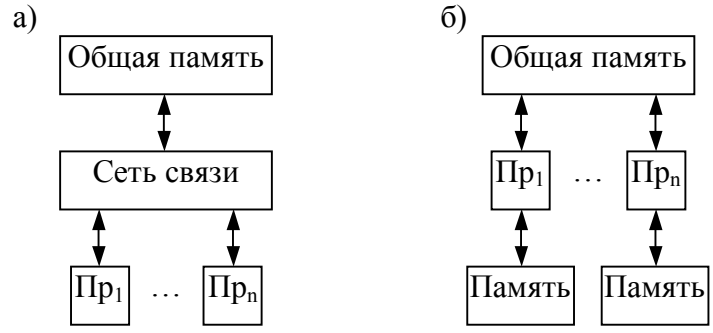


Рис.5.10. Структуры мультипроцессорных систем:
а) с общей памятью б) с разделенной памятью

к общей разделяемой памяти (рис.5.10,а) или к памяти всех остальных процессоров (рис.5.10,б). Устройство управления рассылает на каждом такте различные команды всем процессорам, каждый из которых производит обработку своих данных. Поэтому имеется множественность, как данных, так и команд.

При увеличении количества взаимосвязей между потоками команд характер алгоритмической структуры совершенно меняется, она превращается в параллельную структуру общего вида (рис.5.9,б). Эта структура таит в себе наибольшие возможности для организации параллельных вычислений и увеличения скорости обработки. Так как у этого класса алгоритмов практически отсутствуют линейные участки, то возникает необходимость явно (с помощью указателей) задавать отношения предшествования на множество конструкций, что значительно усложняет представление программ. Это обусловлено тем, что для инициализации вычислительной конструкции (команды) необходимо завершение предшествующих команд. Управление порядком исполнения команд в этом случае становится более сложным.

Существует три способа перехода к выполнению следующей команды по завершению предыдущей, которые определяют программную организацию ЭВМ.

ЭВМ с логически–программным управлением используют принудительный способ перехода к следующей команде. Этот способ характерен для традиционных фон-неймановских машин. Каждая команда помимо операции и операндов содержит явное (указывается адрес перехода) или неявное (к адресу текущей команды добавляется константа) указание к выполнению следующей команды. Форматы таких команд описаны в главе 1. Такой способ управления также получил название командного (или управление потоком команд). Процессоры этого класса машин имеют запросный механизм вызова операндов (вызов по ссылке). При использовании механизма вызова операнда по ссылке для передачи операндов между командами используется общая область памяти (возможно, отдельная ячейка). Результат R, порождаемая командой, помещается в область памяти по адресу (А). Каждая из команд, которая использует этот операнд, должна сделать запрос по адресу (А) к общей памяти и получить из нее операнд R. Таким образом, передача операндов между командами осуществляется путем взаимных ссылок (А) к одной общей памяти. Механизм ссылок прост в реализации, но является постоянным источником ошибок при программировании и ограничивает возможность параллельных вычислений из-за конфликтов при одновременных запросах от разных команд к одной общей памяти.

В ЭВМ с управлением потоками данных выполнение команд осуществляется по мере готовности участвующих в них данных. Алгоритм для машин потоков данных представляется в виде графа, вершины которого соответствуют вычислительным конструкциям (командам), а дуги – потокам данных. Причем каждая дуга направлена от вершины, в которой данные вырабатываются как результаты, к вершине, в которой эти данные используются как операнды. На рис.5.11 приведен пример графа потоков данных (ГПД), вычисляющий выражение U=(x*y)+(y*z).. Результат выполнения этой задачи будет получен после срабатывания последней вершины ГПД.

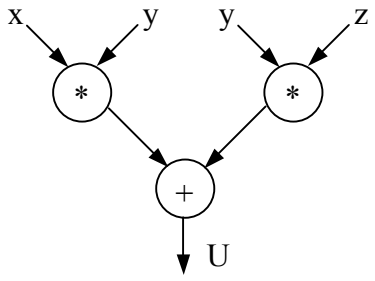


Рис.5.11. Граф потоков данных

В общем случае этот ГПД можно представить в виде списочной структуры. Элемент такого списка определяется шаблоном:

$$T=(O_p, C, R_1, \ldots, R_n, D_1, \ldots, D_m),$$

где O_p – обозначают примитивную операцию (команду); C – условие срабатывания вершины ГПД; $R_1, \ldots, R_n$ – поля операндов, заполняемые по мере выполнения предшествующих операций (предшествующих вершин в ГПД); эти операнды могут использоваться для определения условий срабатывания спусковой функции C; $D_1, \ldots, D_m$ – поля значений, указывающие какому шаблону должен быть передан результат выполнения операции O_p.

Программу для потоковых машин можно рассматривать как совокупность командных шаблонов, последовательность реализации которых определяется в момент получения результатов выполнения операций на предыдущем шаге. При наличии достаточного количества процессорных элементов одновременно могут обрабатываться произвольное число готовых к исполнению операций. Параллелизм в машинах потоков данных не задается явно, а выявляется в процессе исполнения. Другой особенностью потоковых машин является то, что в них реализуется механизм передачи операндов по значению. Механизм вызова операнда по значению предполагает непосредственную передачу отдельных копий полученного значения во все команды, в которых он будет использован. Рассылка операндов при этом осуществляется в виде сообщений, каждое из которых имеет начало, конец, адрес получателя и, собственно, результат операции. При обмене сообщениями отсутствует совместно используемое общее поле памяти, что приводит к упрощению синхронизации и уменьшению числа ошибок при программировании.

В ЭВМ с редукционно-программным управлением стимулом начала операции является потребность в результате ее выполнения, т.е. выполнение команд в последовательности, определяемых запросами операндов.

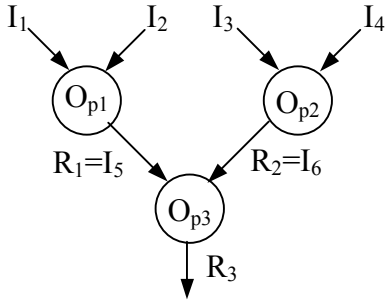


Рис.5.12. Информационный граф

Рассмотрим пример. Имеется алгоритм, который представлен в виде графа на рис.5.12. Здесь {I} – множество входных операндов, {R} – множество выходных результатов, множеству вершин $\{O_p\}$ представлены в соответствие простые операторы. Предположим, что возникла необходимость в получении результата R_3, т.е. сформирован "запрос R_3". Этот запрос используется для вызова оператора O_{p3}, но он не может быть выполнен из-за отсутствия операндов $I_5=R_1$ и $I_6=R_2$. В этом случае автоматически порождаются два новых запроса: "запрос R_1" и "запрос R_2". Процесс порождения запросов будет продолжаться до тех пор, пока не будет встречен оператор, готовый к выполнению. По выполнению этого оператора происходит замена его своим результатом, который передается по обратному пути, обеспечивая возможность реализации всех предыдущих запросов более высокого уровня. Такой способ управления вычислениями получил название управление потоком запросов, он хорошо отражает особенность параллельной обработки и позволяет контролировать число порождаемых параллельных процессов.

5.2.2. Последовательная организация и конвейеризация

Успехи в развитии технологии СБИС создали предпосылки для появления в начале 70-х годов нового класса вычислительных систем – микропроцессоров. Проектирование микропроцессоров первых и последующих поколений базировалось на традиционной архитектуре фон-Неймана, которая по классификации Флинна относится к классу ОКОД машин. В состав таких машин входят единственная память для хранения данных и программ, одно арифметическое устройство, исполняющее текущую команду, одно устройство управления, осуществляющее контроль над исполнением команд, счетчик команд, хранящий адрес текущей команды, и простой механизм модификации счетчика команд. Взаимные связи между перечисленными компонентами системы довольно просты. Так как алгоритмическая структура для этих машин имеет последовательный характер (рис.5.2.), то быстродействие такой машины ограничено ее последовательной сущностью и определяется временем исполнения каждой команды.

Важное достоинство такой последовательной организации вычислительного процесса состоит в том, что для машин данного класса разработано огромное количество алгоритмов и программ, создана обширная база данных, используемая существующими программами, накоплен богатый опыт программирования, разработаны фундаментальные языки и технологии программирования. Поэтому дальнейшее совершенствование организации шло по пути сохранения последовательной структуры, но с добавлением механизмов параллельной обработки.

Методы параллельной обработки, примененные в 70-х годах в суперкомпьютерах, в настоящее время проникли в архитектуру перспективных микропроцессоров нового поколения. Начиная с универсальных 16-разрядных микропроцессоров, низкоуровневый параллелизм имеет место в конвейерной форме, а в матричных системах, содержащих множество МП, могут одновременно выполняться несколько вычислительных процессов.

Фон-неймановская архитектура имеет четыре основные характеристики:

- наличие единого вычислительного устройства, включающего процессор, средства передачи информации и память;
- линейная структура адресации памяти, состоящей из слов фиксированной длины;
- низкий уровень машинного языка, команды которого осуществляют простые операции над элементарными операндами;
- централизованное последовательное управление.

Повышение производительности таких систем шло по пути увеличения скорости выполнения простых команд и повышения быстродействия полупроводниковой памяти. По мере развития полупроводниковой технологии, с одной стороны, росло быстродействие микропроцессоров за счет совершенствования методов проектирования, охватывающих первые три уровня создания ЭВМ. С другой стороны, по мере увеличения емкости памяти, которые размещались на кристаллах, внешних по отношению к МП, значимыми становились задержки, вносимыми паразитными емкостями, протяженными линиями связи, буферными декодирующими схемами, что ограничивало быстродействие системной памяти. Передача команд и данных между МП и памятью осуществляется по единственному коммуникационному тракту, при этом темп пересылки данных по интерфейсу памяти накладывает жесткие ограничения на скорость обработки данных.

Эта проблема обостряется в мультипроцессорных системах. Работа системной магистрали сопряжена с необходимостью разрешения конфликтов между различными процессорами, запрашивающими разрешения на управление шиной. Общая шина и общая память в любой момент времени могут обслуживать только один процессор, и по мере добавления новых процессоров с целью повышения скорости обработки общая шина все в большей степени становится узким местом систем.

Для снижения влияния узкого места, каким является в фон-неймановской архитектуре общая шина памяти, разработчики перспективных МП вводят в их состав конвейерный механизм и кэш – память.

5.2.2.1. Принцип организации конвейера

В основе повышения производительности вычислительных систем, как отмечалось ранее, лежат методы проектирования известные под общим названием совмещение операции (принцип параллельной обработки), при котором аппаратные средства системы в любой момент времени выполняют одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризация. Эти два термина отражают два различных подхода. При параллелизме совмещение достигают путем воспроизведения нескольких или многих копий аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структуры, осуществляющих решение различных частей задачи.

Конвейеризация основана на расщепление функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Команды и данные перемещаются по ступеням конвейера со скоростью, которая зависит не от его длины (числа ступеней), а только от скорости, с которой новые объекты могут подаваться на вход конвейера. Эта скорость в свою очередь, определяется временем прохождения самой медленной ступени конвейера. В конвейере в каждый момент времени одновременно работают все ступени, производящие обработку различных объектов, образующих один поток данных. Такое совмещенное во времени использование различных ступеней многими объектами часто называют перекрытием.

В общем, оценка производительности конвейера определяется следующим образом. Если некоторая функция при реализации ее на не конвейерном процессоре может быть выполнена за 1 такт и если эта функция может быть разбита на некоторые ступени, то конвейер, спроектированный для повторяющегося выполнения той же функции, может выполнять ее за $1/N$ тактов, что дает N-кратное увеличение производительности. В реальности ограничения, накладываемые на достижение максимальной скорости, связаны как с аппаратными средствами, так и с самой функцией. Однако, несмотря на эти ограничения, резкое повышение производительности конвейера все же возможно, и оно регулярно достигается в реальных системах.

Современные перспективные МП широко используют принцип совмещения операций, в которых базовые операции (система команд) охватывают исполнение команд на машинном уровне и достигают значительного перекрытия при функционировании процессора и процессора ввода-вывода. Они проектируются как иерархические, где каждая ступень одного уровня конвейеризации проектируется как конвейер.

Конвейеризация часто применяется в качестве поддержки в машинах класса ОКОД и ОКМД, так как современные МП строятся на базе архитектуры ОКОД, то в них заложен механизм конвейера. Конвейерный механизм для этого класса машин реализуется на основе анализа процесса исполнения машинных команд. Несмотря на разнообразие типов команд, процесс исполнения типовой команды (сложение, загрузка и т.п.) можно разбить на следующие основные этапы:

1) выборка команды (ВЫБК);
2) декодирование команды (ДЕКОД);
3) определение адреса операнда (ВЫЧА);
4) выборка операнда (ВЫБО);
5) собственно выполнение (ВЫП);
6) запоминание результата (СОХР);
7) завершение операции (КОНОП).

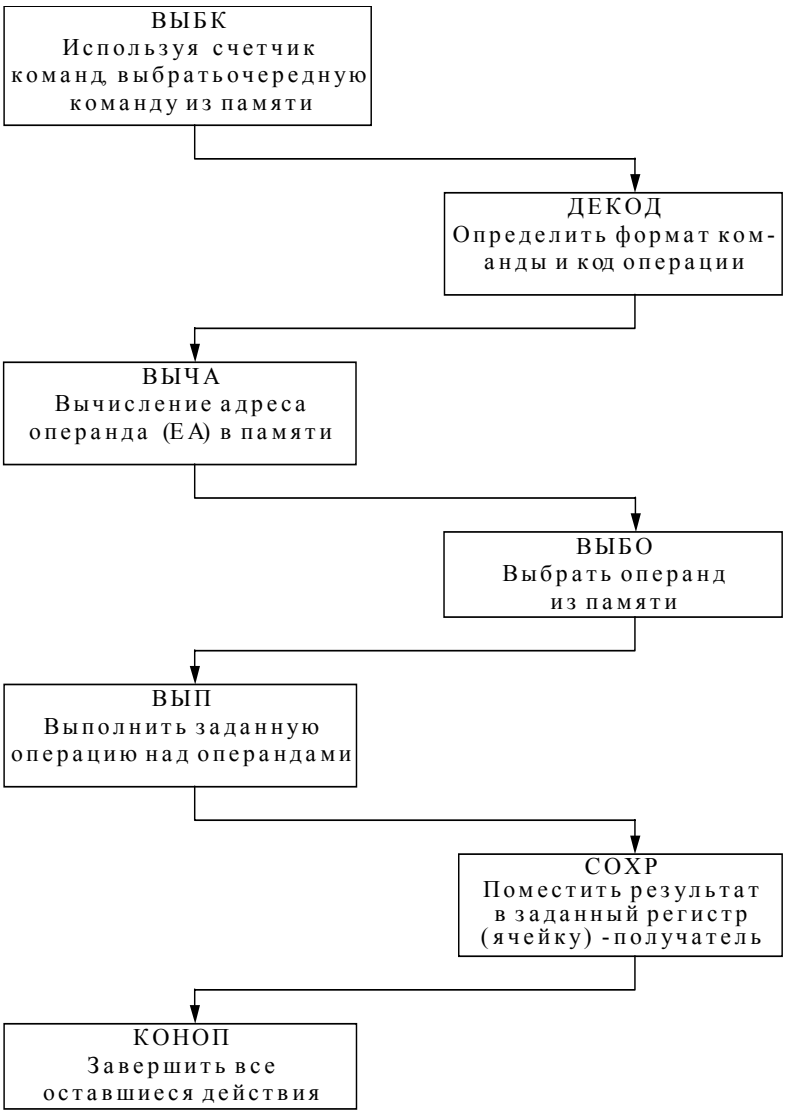


Рис.5.13. Разбиение исполнения типовой команды

Все эти функции, выполняемые в процессе исполнения команды, определяют порядок протекания этого процесса. Первые шесть функций являются основными и представлены на рис.5.13. Функция КОНОП (конец операции) осуществляет следующие действия процессора: установка флажков ошибок (переполнение, деление на нуль и т.д.), установка кода состояния (при условных командах управления), увеличение счетчика команд и проверка на наличие запретов на прерывание.

Разбиение на ступени процесса исполнения команды можно осуществить различными способами. Ступени можно объединить (например, фазу ВЫП с фазой СОХР) или разделить (например, разбить ВЫЧА – на меньшие ступени), фазу КОНОП можно совместить с другими фазами.

Всякая реальная программа, исполняемая на ЭВМ, является комбинацией или смесью команд каждого класса. Хотя смесь зависит от структуры программы, но исследования процессов выполнения большого количества программ из разных прикладных областей показывают, что существует явное смещение частот использования в направлении небольшого набора простых команд.

В таблице 5.1 приведена одна из таких смесей.

Таблица 5.1

Тип команды	Доля, %
ADD (загрузка, сложение, умножение и т.п.)	60
STORE – запоминание	15
JUMP - безусловный переход	5
BRANCH – условный переход (выполняемый)	12
BRANCH - условный переход (невыполняемый)	8
	100

Большинство команд в этой таблице относится к классу команд, показанному на рис.5.13. На рис.5.14 показано разбиение исполнения операции для трех типов команд представленных в таблице 5.1.

Команда STORE перемещает копию содержимого регистра центрального процессора в указанную ячейку памяти. В этой команде отсутствуют фаза выборки операнда из памяти и фаза выполнения, не надо записывать ни флажков ошибок, ни результатов проверок.

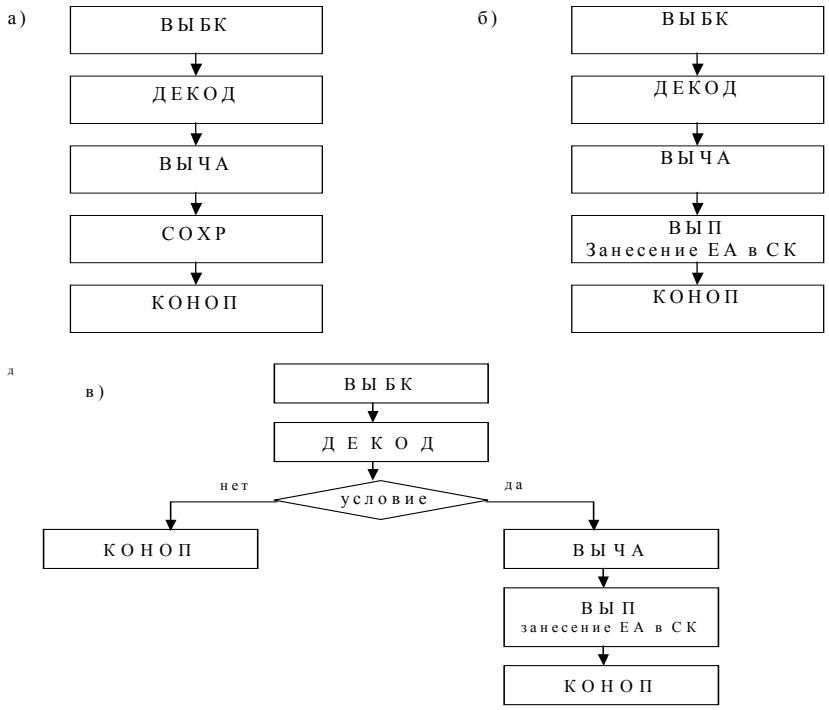


Рис.5.14. Разбиение исполнения команд различных типов:
а) запоминание; б) безусловный переход; в) условный переход

Безусловный переход JUMP имеет еще более простое разбиение, команда заканчивается, как только исполнительный адрес ЕА вводится в счетчик команд (СК). Действия КОНОП, связанные с завершением, сокращаются, поскольку не надо обновлять СК.

Команда условного перехода BRANCH имеет два варианта разбиения, причем проверка условия является одной из главных фаз исполнения команды. Если проверка дает отрицательный результат, то команда завершается действиями КОНОП. Если проверка дает положительный результат, то выбирается разбиение, подобное разбиению для команды безусловного перехода JUMP.

Как видно из приведенных разбиений количество фаз для разных типов команд различаются между собой. Когда в конвейере заканчивается выполнение определенного

этапа очередной команды, высвобождается соответствующий исполнительный блок и может быть начато выполнение аналогичного этапа следующей команды. При перекрывании процесса исполнения команд возникают проблемы зависимостей между перекрываемыми командами, которые получили название межкомандных зависимостей, а сами зависимости называются помехами. Аппаратный метод, который реализует обнаружение и разрешение помех, называется блокировкой.

5.2.2.2. Межкомандные зависимости

Межкомандные зависимости в конвейере бывают двух типов: зависимости по управлению и зависимости по данным. Зависимость по управлению между двумя командами возникает в ситуации, когда при выполнении предыдущей команды остается неизвестным, какая из двух окажется следующей. Такая зависимость обычно порождается условными переходами и является отрицательным фактором, существенно влияющим на производительность конвейера. Анализ команды условного перехода (команды i) показывает (рис. 5.14, в), что при положительном переходе выполнения команды происходит приостановка выполнения команды i+1 и загрузка нового исполнительного адреса EA в счетчик команд. Возобновление работы конвейера будет начинаться с цикла выборки команды с нового исполнительного адреса EA. В потери времени конвейера входит полный цикл обработки команды. Для уменьшения этих потерь существуют различные аппаратные средства: предварительный вызов начальных команд обеих ветвей; использования нескольких командных буферов для хранения различных участков программ; хранение задержанных ветвей в виде позволяющем загрузить конвейер сразу после завершения выбора ветви.

В случае зависимостей по данным помеха возникает, когда к объекту данных (например, к регистру, ячейке памяти или флажку) обращаются, или его модифицируют две различные соседние команды, и конвейер перекрывает их исполнение. Имеются три класса таких помех: чтение после записи (RAW), запись после чтения (WAR) и запись после записи (WAW).

Определим эти помехи формальным образом. Имеется две команды i и j, причем команда j логически следует за командой i. Область определения команды i, обозначаемая D(i), это множество всех объектов (регистров, ячеек памяти и флагов), содержимое которых влияет на исполнение команды i. Множество значений команды i, обозначаемое R(i), это множество всех объектов (регистров, ячеек памяти и флажков), содержимое которых может быть изменено за счет исполнения команды i.

Таким образом, D(i) –это множество всех объектов, читаемых командой i, а R(i) – множество всех объектов, модифицируемых ею. Тогда эти три класса помех можно определить следующим образом:

1) RAW $R(i) \cap D(j) \neq 0$;
2) WAR $D(i) \cap R(j) \neq 0$;
3) WAW $R(i) \cap R(j) \neq 0$.

Эти межкомандные зависимости показаны на рис.5.15.

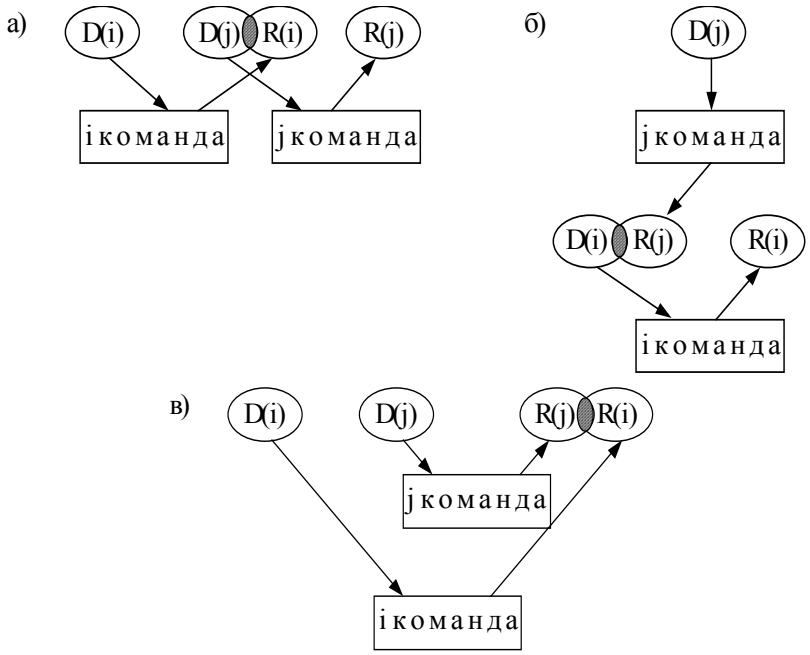


Рис.5.15. Межкомандные зависимости:
а) помеха "чтение после записи"; б) помеха "запись после чтения"; в) помеха "запись после записи"

Для примера рассмотрим следующий фрагмент программы:

STORE X;
ADD X;
STORE X;
STORE X.

Помеха RAW. Если команда ADD читает из ячейки памяти с адресом X раньше, чем первая команда STORE успела обновить ее содержимое, то к соответствующему регистру МП будет прибавлено неправильное значение. Команда ADD (команда i) получила значение, которое является "слишком старым".

Помеха WAR. Если вторая команда STORE перекрывается с командой ADD, то может случиться, что модификация содержимого ячейки X произойдет раньше, чем к ней обратиться команда ADD. В этом случае команда ADD получит неправильное значение, и это значение является теперь “слишком новым”.

Помеха WAW. Это может случиться между двумя последними командами STORE. Если вторая STORE завершается после третьей, то содержимое ячейки X не будет таким, какое ожидается по логике выполнения программы.

Существуют два подхода к проблеме обнаружения этих помех.

Первый состоит в том, чтобы централизовать обнаружение помех на одной ступени, обычно связываемой с фазой выборки команды, и сравнивать на этой ступени области определения и значений команды с соответствующими областями определений и значений для команд находящихся в конвейере.

Второй подход состоит в том, чтобы позволить команде продвигаться по конвейеру до тех пор, пока она не достигнет ступени, в которой требуется тот или иной элемент, либо из области определения, либо из множества значений. Тогда осуществляется проверка на потенциальную помеху от другой команды, находящейся в конвейере. Второй подход более гибкий, чем первый, однако, объем аппаратуры, необходимой для выполнения сравнения, может расти как квадрат числа ступеней.

После обнаружения помехи существуют две возможности по устранению их. В первой случае приостанавливают конвейер. Так , если обнаружена команда j, находящаяся в состоянии помехи с ранее инициализированной командой i, то все последующие действия

команд j, j+1, j+2, … останавливаются до тех пор, пока команда i не пройдет через точку конфликта.

Во втором случае, останавливается команда j, но командам j+1, j+2, … разрешается двигаться по конвейеру. Это дает командам j+1, j+2, … возможность опередить команду j в конвейере, т.е. нарушается очередность выполнения (внеочередное исполнение). При этом остается в силе проверки на наличие помех команды j с командами j+1, j+2, …

Для устранения помехи типа RAW используется метод, называемый коротким замыканием (или запоминанием промежуточного результата). В этом случае, копия данных, подлежащих запоминанию i команды, передается непосредственно ожидающей команде j, что позволяет избежать фазы чтения из памяти и необходимости ждать завершения i команды. В общем случае для ускорения устранения всех помех типа RAW использует метод, называемый опережающей засылкой. В каждой ступени конвейера, в которой должно происходить обращение к данным (выборка операнда) помещается множество фиксаторов ступени (регистров) для приема этих данных. Если данные команды j (проходящей через некоторую ступень) находятся в состоянии помехи типа RAW с ранней командой i, один из фиксаторов этой ступени загружается не данными, которых все еще нет, а индексом или тегом, указывающим на ту ступень, которая их вырабатывает. Затем команда j, ожидающая эти данные, задерживается на этой ступени до их поступления. Остальные фиксаторы на входе в ступень позволяют проходить другим командам через нее, пока команда j, зависящая от RAW, задержана. Когда ступень завершает выполнение i команды, она проверяет все соответствующие фиксаторы ступеней на предмет совпадения его идентификатора с тегом. Если такой тег имеется, то соответствующая ступень получает копию данных. Это позволяет возобновить выполнение задержанных команд.

Для устранения помехи типа WAR и WAW используют метод дублирования или переименования регистров. Он предполагает, например, использование двух внутренних регистров Reg0 и Reg1. После такого дублирования можно одновременно и читать,и записывать или одновременно записывать в них данные от команд находящихся в конвейере.

5.2.2.3. Структурная организация конвейера

Процесс исполнения команды в общем случае можно разделить на две основные стадии: подготовку и исполнение команды. Реализация этих стадий в процессоре осуществляется двумя блоками, которые получили стандартные названия (рис.5.16).

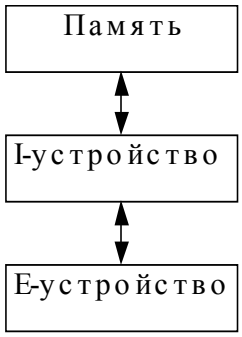


Рис.5.16. Стандартное разбиение процессора в архитектуре ОКОД

Ступени, осуществляющие выборку команды, часть декодирования и некоторые действия функции КОНОП, образуют устройство обработки команд или I-устройство. Ступени, осуществляющие функции исполнения и остальные действия функции КОНОП, образуют устройство исполнения (обработки данных) или E-устройство.

При этом вопрос о доступе к данным (выборка операнда) является ключевым моментом при таком разбиении. Помещение этой функции в I-устройство позволяет создать единый интерфейс к памяти, но несколько ограничивает диапазон разбиений. В этом случае, основные функции выборки команды, большая часть декодирования, вычисление

исполнительного адреса EA, и доступ к операндам будут производиться в I-устройстве и все команды (табл.5.1), кроме команд класса ADD, будут целиком исполняться в нем. Это приводит к усложнению процесса разбиений этих функций в I-устройстве.

При помещении функции доступа к данным в E-устройство придает большую гибкость при разбиении других функций, но требует системы памяти с двумя интерфейсами. Усложняются схемы обнаружения и устранения помех, поскольку I- и E-устройства для предотвращения помех должны точно знать, к чему обращается, или обращалось другое устройство.

Решением этой проблемы является введение устройства управления памятью (MCU) между I- и E-устройствами и памятью, как показано на рис.5.17.

Это устройство обрабатывает запросы на доступ к памяти и позволяет централизовать обнаружение помех, возникающих из-за обращений к одним и тем же ячейкам памяти. Устройство MCU может быть конвейеризовано, и выступать как отдельная ступень в процессе исполнения команд.

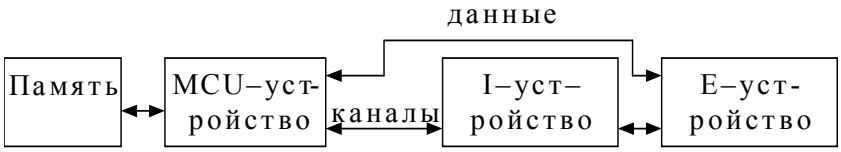


Рис.5.17. Использование устройств управления памятью в конвейере ОКОД

Другим важным элементом при структурной организации процессора в виде конвейера является блок регистров, называемыми регистрами общего назначения (РОН). На рис.5.18 приведена схема размещения регистров между I- и E-устройствами.

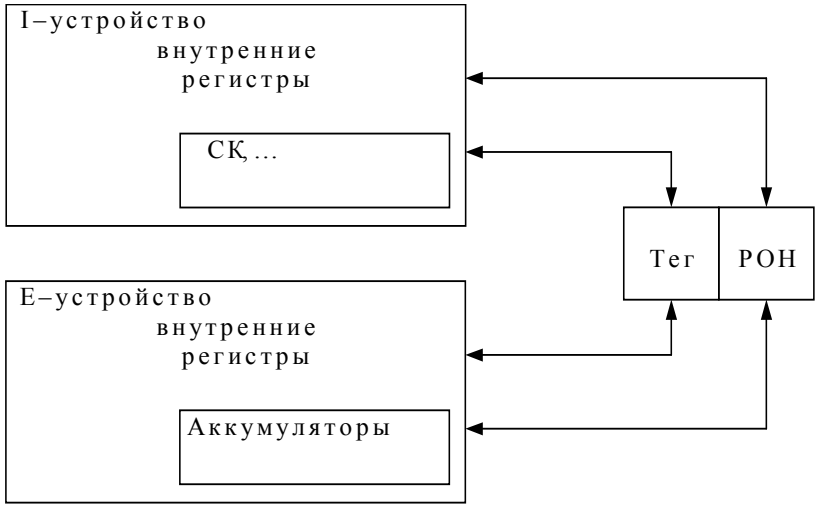


Рис.5.18. Размещение регистров между I- и E-устройствами

Регистры, принадлежащие I- и E-устройствам, встраиваются в них, а РОН (блок накопителя или файл регистров) пакетируются отдельно. Каждый регистр в общем случае имеет логику тега (блок диспетчирования у современных микропроцессоров), с помощью которого осуществляется обнаружение и устранение помех. I-устройство декодирует новую команду до момента определения, содержимое каких из этих регистров должно ею читаться, или модифицироваться. Тегу каждого регистра, который должен модифицироваться, присваивается значение, указывающее ее будущим командам, что содержимое этого регистра подвергается изменению. Аналогично, для каждого регистра, из которого должно читаться ее содержимое с помощью новой команды, читается его тег. Если этот тег указывает, что содержимое регистра подвергается модификации предыдущей командой,

новая команда должна быть задержана. Это соответствует обнаружению и устранению помехи типа RAW. Подобные использования тега предотвращает помехи типов WAR и WAW.

Когда команда завершает модификацию содержимого одного из регистров, модифицируемое значение посылается в соответствующий регистр, а его тег гасится. После погашения тега следующая команда может продолжить свое исполнение.

В предыдущем подразделе было рассмотрено два варианта приостановки команд при возникновении помех. В первом случае, команда может быть приостановлена сразу после вхождения в I-устройство, а ее исполнение возобновлено только после устранения всех помех (когда будут погашены все теги регистров и данные в них станут доступными).

Во втором варианте, который нашел широкое распространение в современных МП, исполнение i команды задерживается, но в конвейере продолжается выполнение команд i+1, i+2, …, это приведет к внеочередному исполнению команд i+1, i+2, …, операнды которых уже имеются в наличии.

При такой организации конвейера регистры общего назначения могут быть узким местом, поскольку I- и E-устройства и отдельные ступени в них могут одновременно потребовать доступа к этим регистрам (характерно для помехи типа RAW). В современных перспективных микропроцессорах вся регистровая логика дублируется с целью предоставления их содержимого каждой ступени нуждающейся в них. Это позволяет без конфликтов одновременно читать из многих различных регистров. Однако, если регистр или его тег подлежит модификации, то эта модификация должна быть выполнена на всех дублирующих регистрах. Это требует дополнительного цикла для каждого устройства.

Для полной загрузки I- и E-устройств вводится специальный фиксатор, называемый очередью команд (I-очередь), который организован по принципу FIFO (первое записанное слово считывается первым), как показано на рис.5.19.

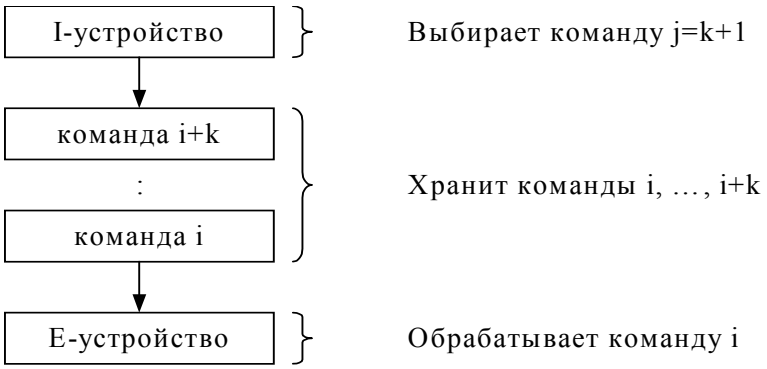


Рис.5.19. Очередь команд между I- и E-устройствами

I-устройство работает независимо от E-устройство по принципу опережающей или предварительной выборки команд, функционирование которой показано на рис.5.20.

Работа I-устройства будет происходить до тех пор, пока в очереди команд имеется свободное место, I-устройство будет увеличивать содержимое счетчика команд СК и производить выборку следующей команды. Эти команды размещаются в очереди для последующего исполнения E-устройством.

При реализации команды перехода (рис.5.14, в) E-устройство сигнализирует об его окончании I-устройству, которое должно очистить I-очередь и перезагрузить счетчик команд СК значением, вычисленным E-устройством. Работа E-устройства при этом останавливается до тех пор, пока эта команда не будет выбрана и помещена в I-очередь, после чего E-устройство возобновляет свою работу.

Обнаружение помех и их устранение при таком разбиении I-устройства являются сравнительно простыми.

Зависимости по управлению обрабатываются в явной форме, а зависимости по данным определяются на этапе выборки команды.

Эффективность функционирования конвейера, показанного на рис.5.20, начинает резко падать при наличии в программе значительного количества команд условного перехода типа BRANCH.

При условном переходе BRANCH происходит полная остановка I-устройства, до окончания выполнения этой команды E-устройством (рис.5.14,в), и перезапуск конвейера с вычисленного адреса EA.

Производительность конвейера падает до производительности не конвейерного процессора.

Для того, чтобы минимизировать этот эффект используют предварительную выборку команд по одному из возможных направлений перехода или обоим

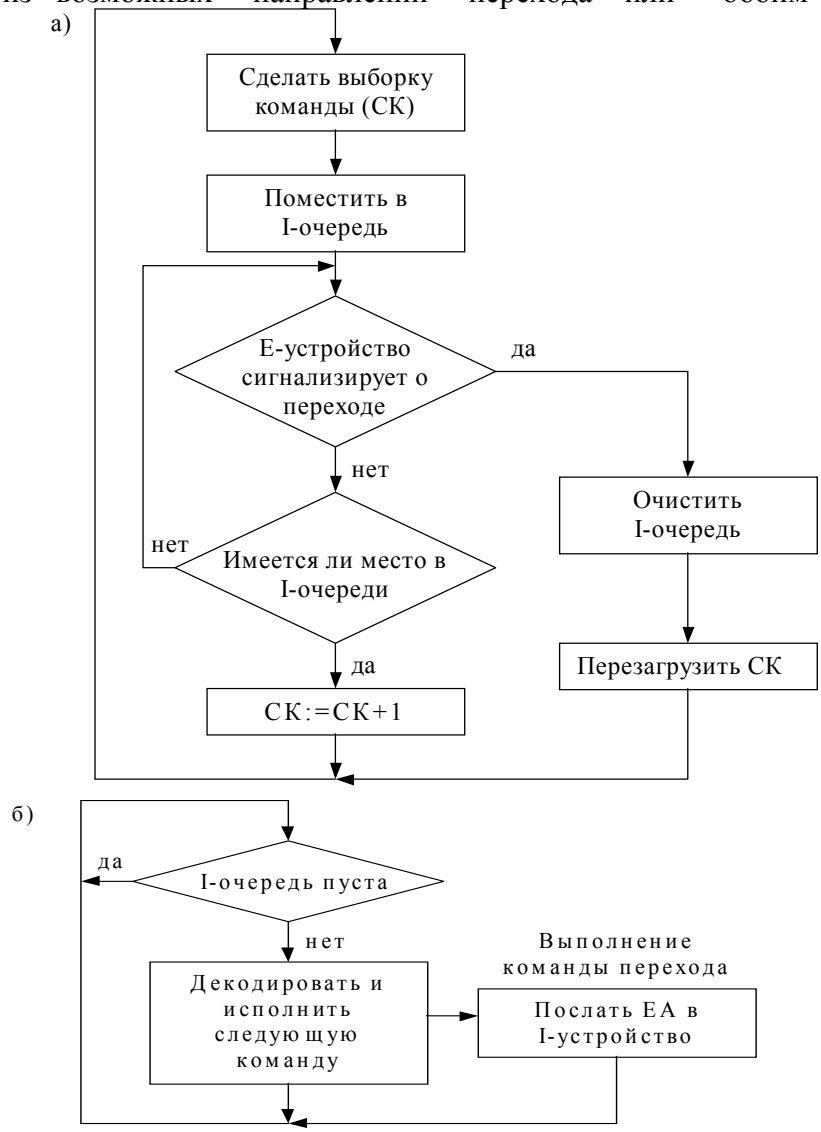


Рис.5.20. I-устройство как устройство предварительной выборки: а) I-устройство; б) E-устройство

направлениям, когда встречается переход BRANCH. Затем, E-устройство окончательно разрешает проверяемое условие и переходит к выполнению команды ветви, определенной данным условием. Пока E-устройство обрабатывает эту команду, I-устройство может осуществлять продвижение вперед, выбирая дальнейшие команды. На рис.5.21 показана работа такой структуры.

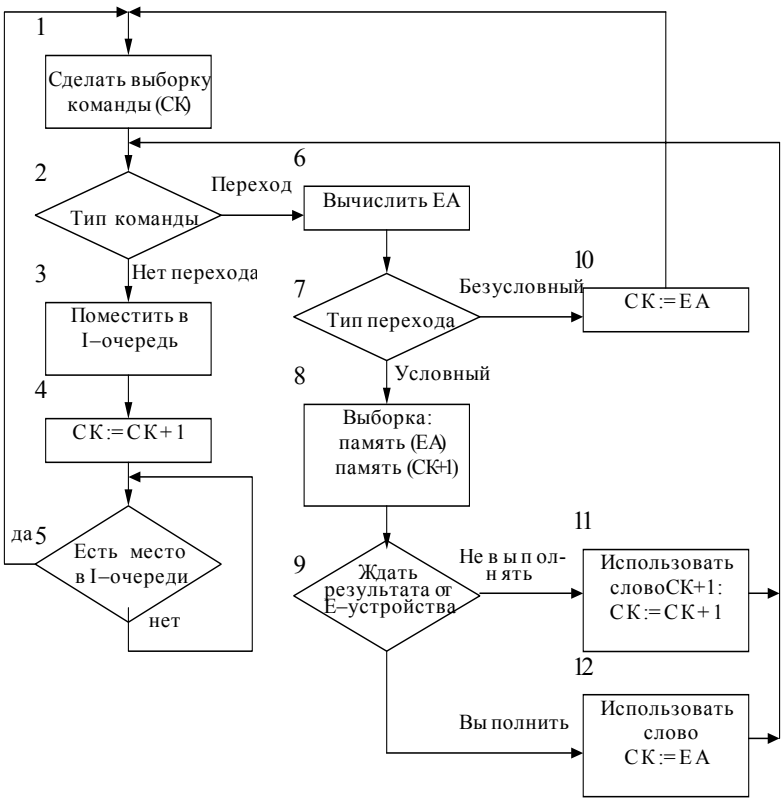


Рис.5.21. I-устройство, совмещающее функции предсказания направления переходов

При такой организации I-устройства выборка направлений возможных переходов возлагается на специальный блок, называемый блоком предсказания направлений переходов. Этот блок содержит буфер адресов переходов, в который заносятся команды, из соответствующих ячеек памяти, находящихся по адресу ЕА.

Это позволяет значительно сократить время простоя конвейера. С другой стороны, если адрес перехода выходит за рамки буфера адресов перехода (блок 6, рис.5.21), то все операции на конвейере прекращаются, он очищается и начинается исполнение с правильного адреса. Поэтому весьма важно, чтобы вероятность правильного прогноза была наиболее высокой. В современных микропроцессорах они лежит в пределах 80-97%.

Рассмотрим пример. Микропроцессор 8086, выполненный в виде однокристальной БИС, был спроектирован как расширение вверх семейства 8080 и включает в набор команд много 8- и 16-разрядных операций. Внешние 20 выводов предназначены как для адресов, так и для данных (16 разрядов на доступ).

В этом МП реализована предварительная выборка и отдельное Е-устройство для повышения быстродействия. Для управления этим устройством используется микропрограммирование. В этом МП впервые был заложен механизм конвейеризации.

Из всех разработанных структур этого МП с различной степенью конвейеризации самым оптимальным по степени интеграции и быстродействия была признана структура, показанная на рис.5.22.

МП представляет собой двухступенчатую конвейерную структуру с предварительной выборкой (I-устройство) и Е-устройством, с I-очередью между I- и Е-устройствами.

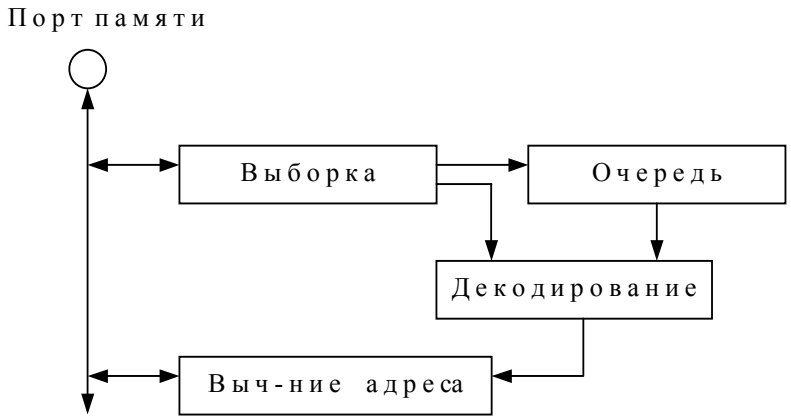


Рис.5.22 МП 8086 – двухступенчатый конвейер с I-очередью

Е-устройство перекрывает также фазу КОНОП данной команды с фазой декодирования следующей. Глубина очереди команд составляло 6 байтов, что в среднем соответствует двум командам.

5.3. Организация памяти

Повышение производительности вычислительных систем на структурном уровне, как отмечалось выше, можно достигнуть за счет сокращения временных затрат при обращениях к памяти. Сокращение временных затрат предполагает совершенствование таких основных характеристик памяти, как быстродействие и емкость. За последние десятилетия за счет совершенствования полупроводниковой технологии, емкость памяти наиболее крупных вычислительных систем возросла от 10^3 до 10^8 слов, а время цикла уменьшилось с 20 мкс до 50 нс. Однако даже с учетом прогресса в технологии быстродействующие запоминающие устройства (ЗУ) остаются более дорогими, чем медленные. Следовательно, с точки зрения стоимости эффективнее иметь иерархию памяти, которая позволяет согласовать характеристики памяти и процессора. Иерархия памяти существует благодаря разнице в стоимости устройств хранения информации. Если бы ЗУ с минимальным временем доступа и большой емкостью были достаточно дешевыми, то системы имели бы только один уровень памяти. Из-за отсутствия такой памяти требования высокой производительности и низкой стоимости наилучшим образом удовлетворяются использованием различных технологий, позволяющих сочетать дорогие быстродействующие устройства с дешевыми и медленными блоками памяти. Память, разработанная на основе этой концепции, называется иерархической или многоуровневой системой памяти.

Таким образом, память в современных вычислительных системах строится по иерархической структуре. Главной целью разработки эффективной иерархической системы памяти является обеспечение обмена необходимым количеством информации с процессором со скоростью наиболее быстродействующего ЗУ, имеющегося в иерархии. Многоуровневая система памяти показана на рис.5.23.

Такая многоуровневая система памяти включает регистровый, сверхоперативный, оперативный, внешний и архивный уровни. Как и в любой иерархии памяти, эти структуры упорядочены по убыванию емкости и по возрастанию скорости.

Наибольший по емкости является архивная память, а наименьший - регистровые файлы (РФ). Наибольшей по возрастанию скорости является регистровые файлы и наименьшей - архивная память на базе накопителей на магнитных лентах (НМД) и накопителей на оптических дисках (НОД).

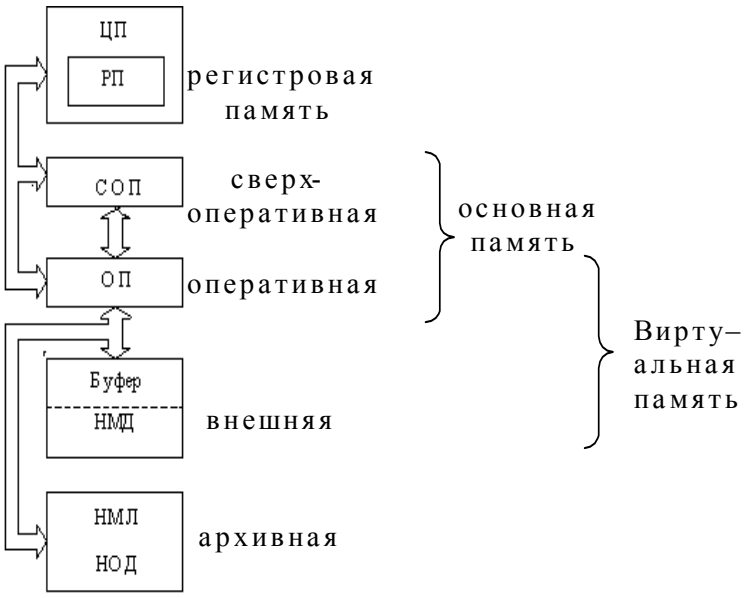


Рис.5.23. Многоуровневая система памяти в вычислительных системах

Внешняя память строится на базе магнитных дисков с подвижными головками (НМД), которые подразделяются на два вида: жесткие (винчестеры) и гибкие диски.

5.3.1. Виртуальная память

В первых микропроцессорах из-за небольшого объема основной памяти большая часть времени работы программ отводилось проблеме перекрытия. Перекрытие определяет организацию передачи информации между основной и внешней памятью за счет включения соответствующих команд и процедур в основную программу. Эти команды и процедуры, производящие разбиение программы на последовательность сегментов и которые перекрывали бы друг друга в памяти, были относительно просты. Однако разработка эффективных методов перекрытия требовала от программистов детального знания особенностей работы машины, на изучение которых они тратили больше времени, чем на решение собственных задач.

Совершенствование инструментальных средств программирования дало возможность строить большие программы и больше внимания уделять решению собственных задач, а не изучению особенностей работы машины. Однако с возрастанием сложности программ повышалось и роль их перекрытия. Фактически на решение этой проблемы затрачивалось от 25 до 40% стоимости программирования.

С увеличением емкости ЗУ эффективные методы перекрытия и принцип иерархии памяти стали играть еще большую роль при их построении.

Необходимость размещения больших программ в малых пространствах памяти стимулировало развитие аппаратных и программных средств ЭВМ с целью обеспечения автоматического размещения информации между основной и внешней памятью. Решение этой проблемы привело к разработке двух подходов распределения памяти: статического и динамического.

Статический подход предусматривал наличие сведений о доступных ресурсах памяти программе до начала ее работы, тогда как динамический подход не предполагал наличие этих предварительных сведений программе и предусматривал сокращение или расширение выделяемой основной памяти в соответствии с текущими потребностями программы.

Создание более совершенных, машинно-независимых, модульных, перемещаемых программ с эффективной структурой, использующей каталоги, делало статическую концепцию все менее и менее привлекательной.

Динамическое распределение памяти основывается на концепции виртуальной памяти. Виртуальная память создает у пользователя иллюзию одного большого, одноуровневого пространства хранения информации. При этом, все функции эффективного управления этой памяти по передаче информации между основной и внешней запоминающими устройствами возлагаются на систему.

Впервые для класса микропроцессоров эта концепция была реализована у 32-разрядных МП. Виртуальное управление памятью было введено с целью реализации мощности 32-разрядных МП в многопользовательских мультизадачных применений, за счет освобождения пользователей от необходимости выполнять сложные оверлейные процедуры при работе с дисковой памятью. Оно позволяет полностью использовать физический адресный диапазон процессора путем рассмотрения полупроводниковой основной памяти в качестве одной из частей этого адресного пространства; остальная часть памяти размещается в виртуальном адресном пространстве на диске.

В 32-разрядных МП заложены два вида организации виртуальной памяти: страничная и сегментная. При страничной организации память разделяется на страницы фиксированного объема (например, 512 байт), и программы загружаются в несвязанные между собой области памяти, называемые блоками, которыми управляет операционная система. На рис.5.24. показана процедура формирования физического адреса при страничной организации памяти. Процессор вырабатывает логический адрес, состоящий из номера страниц и смещения.

Номер страницы используется в качестве указателя для обращения к таблице страниц, из которого извлекается адрес блока, в котором находится эта страница. Адрес блока в совокупности со смещением определяет физический адрес нужного слова памяти. С помощью таблицы страниц логический адрес памяти преобразуется в физический адрес.

Это преобразование осуществляется в два этапа: вначале происходит обращение к таблице страниц, а затем вычисляется физический адрес отыскиваемого слова с использованием адреса блока и смещения.

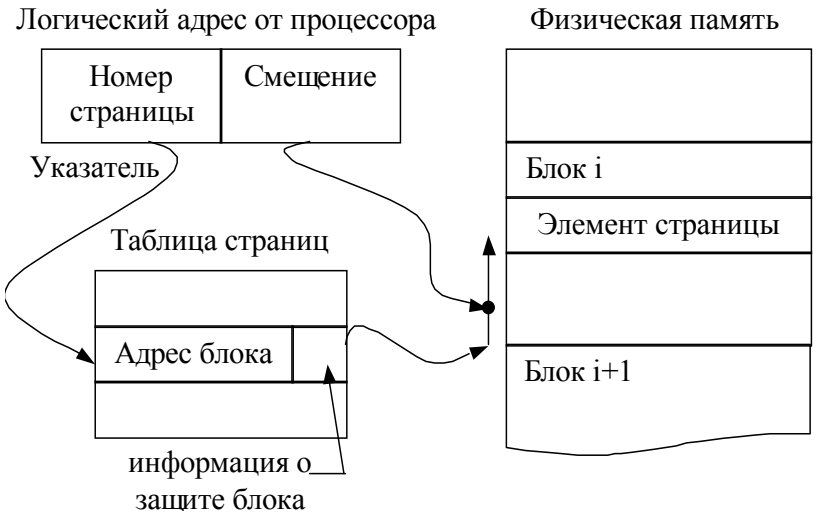


Рис.5.24. Преобразование адреса при страничной организации

В таблице страниц может быть задан вид защиты каждого блока. Например, право на монопольное использование блока определенным пользователем или разрешение на выполнение операций чтения и записи. Преобразование адресов требует дополнительных затрат времени, которые могут значительно снизить скорость обращения к памяти.

Сегментная организация находит широкое применение в модульном программировании, при котором с целью упрощения понимания, написания и контроля программы и отдельных ее частей для реализации каждой функции используется отдельный программный модуль. Модуль определяется функцией, которую он выполняет, а не размерами, которые могут быть разными для различных модулей. Сегментная организация виртуальной памяти позволяет каждому модулю занимать свою сплошную область памяти, тогда как при страничной организации модуль разбивается на страницы. На рис.5.25 представлена процедура обращения к памяти при сегментной организации виртуальной памяти.

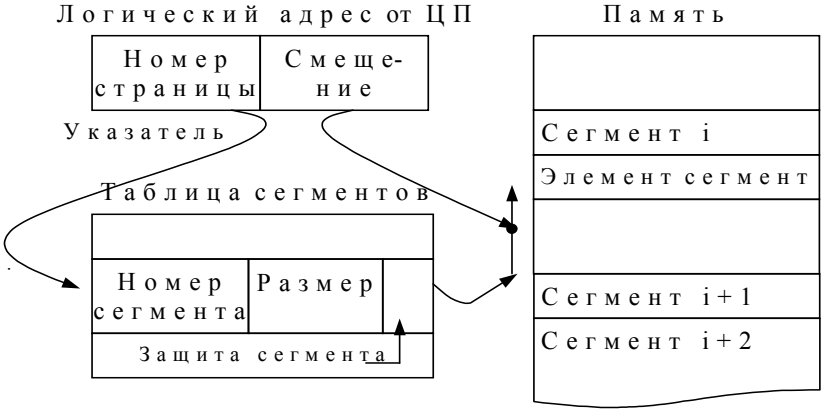


Рис.5.25. Преобразование адреса при сегментной организации

Логический адрес, вырабатываемый процессором для системы с сегментной организации, состоит их номера сегмента и смещения. В таблице сегментов содержатся базовые адреса, которые являются начальными адресами блоков физической памяти, и границы, определяющие размеры сегментов. Величина смещения не должна превышать значение границы. Защита сегмента, содержащего законченный программный модуль, обеспечивается путем однократного задания прав на использование и доступ для записи и чтения.

В процессе работы сегментной виртуальной памяти в ней могут появляться зоны, представляющие собой неиспользуемые области памяти, которые время от времени необходимо объединять. Поэтому сегментная виртуальная память должна быть снабжена алгоритмом размещения, с помощью которого осуществляется поиск неиспользуемых зон памяти для размещения каждого сегмента. В системах страничной организации виртуальной памяти отсутствует опасность появления неиспользуемых зон между программами (внешняя фрагментация), а величина неиспользуемых зон внутри программы (внутренняя фрагментация) не может превышать объема одного блока для каждой программы.

В страничной виртуальной памяти с перемещением по запросам страница пересылается из дисковой в основную память при возникновении необходимости в этом (т.е. когда процессор обращается к области принадлежащей этой странице). Такая организация работы страничной виртуальной памяти реализована в большинстве 32-разрядных МП, и аналогична методам, применяемым в многопользовательских системах, построенных на базе мини и больших ЭВМ.

5.3.2. Основная память

Основная память в микропроцессорных системах (МПС) строится на базе полупроводниковой памяти, реализованной по интегральной технологии в виде модулей БИС. Обычно с МП используются три типа БИС полупроводниковой памяти: постоянное запоминающее устройство (ПЗУ), перепрограммируемое постоянное запоминающее устройство (ППЗУ) и запоминающее устройство с произвольным доступом (ЗУПВ).

Два первых типа памяти (ПЗУ, ППЗУ) являются энергонезависимыми, т.е. их содержимое не меняется при отключении питающих напряжений и не может быть изменено с помощью программы. Они используются для хранения постоянных величин, таблиц и для окончательных вариантов отлаженных программ в многосерийных изделиях.

Базу основной памяти МПС составляют ЗУПВ или, что-то же самое, оперативное запоминающее устройство (ОЗУ), которое служит для хранения программ и данных, участвующих в вычислениях. БИС ОЗУ являются наиболее сложными и дорогими видами памяти, но их содержимое можно модифицировать в процессе вычислений, и они энергозависимы, т.е. теряют содержимое при отключении напряжения питания.

Логически ЗУПВ представляют собой линейную последовательность ячеек с последовательными адресами и с произвольным доступом к любой ячейке. Обычно накопитель такого ЗУПВ организован в виде матричной формы. В состав ЗУПВ с такой организацией входят: матрица запоминающих элементов $m \times n$ (где m –количество строк, n – количество столбцов), служащая для хранения информации. Дешифратор адреса, преобразующий двоичный код адреса в сигналы выборки строки и столбца матрицы. Схемы управления и ввода-вывода данных, определяющие режимы записи или считывания и активизацию буферных схем по линиям данных. Усилители записи и считывания, предназначенные для усиления сигналов записи и считывания.

По принципу действия ЗУПВ подразделяются на два вида: статические и динамические.

Статические ЗУПВ (СЗУПВ) строятся на базе простейших триггеров с непосредственной связью и цепями установки в то или иное состояние, где типовая ячейка содержит шесть транзисторов, и вследствие этого имеют меньшую емкость, чем динамические ЗУПВ.

Триггеры могут выполняться по любой технологии (ТТЛ, ЭСЛ, $И^2Л$, p-МОП, n-МОП, КМОП и др.). Они обладают высоким быстродействием и хорошей помехоустойчивостью, не требуют сложных схем управления и имеют простой интерфейс с процессором, также, по сравнению с динамическими ЗУПВ, в них отсутствуют схемы регенерации хранящихся в них данных. Основными недостатками СЗУПВ являются малая степень интеграции из-за сложности запоминающего элемента и сравнительно большая потребляемая мощность.

Динамические ЗУПВ реализуются в виде матрицы запоминающих элементов, построенных на базе МОП-транзисторов. Хранение информации в таких элементах обеспечивается в виде заряда на емкости между затвором МОП-транзистора и общей точкой схемы – землей. Таким образом, типовая ячейка ДЗУПВ содержит конденсатор, служащий для хранения бита данных, и транзистор, соединяющий этот конденсатор с линией данных для записи и чтения. Из-за постепенного перезаряда паразитных емкостей требуется периодическая регенерация, в ходе которой корректируется значение хранящегося заряда.

В БИС памяти регенерация производится не реже чем через 1…2 мс и выполняется путем считывания и повторной записи хранимой информации.

Использование динамической памяти позволяет значительно уменьшить потребляемую мощность, особенно, в режиме пассивного хранения информации. Кроме того, за счет простоты запоминающего элемента значительно возрастает степень интеграции БИС памяти этого типа. Вследствие этого ДЗУПВ являются наиболее экономичными компонентами для реализации относительно больших по объему матриц памяти с произвольной выборкой.

Основным недостатком динамической памяти является необходимость организации периодической регенерации, которая требует дополнительных действий и средств:

1) управление емкостными нагрузками, содержащимися в схемах ДЗУПВ;

2) внешний счетчик регенерации или регенерационные устройства в составе МП, служащие для восстановления зарядов на запоминающих конденсаторах;

3) управляющая логика для мультиплексирования адресов строк и столбцов при выполнении циклов регенерации и внешних обращений.

Адресация динамического ЗУПВ осуществляется путем подачи стробирующих сигналов на строку и столбец, на пересечении которых находится необходимая ячейка. Работа динамического ЗУПВ может быть организована с использованием нескольких режимов адресации, к числу которых относятся: страничный, сквозной, полубайтовый и с статической дешифрацией столбцов.

В этих методах адресации для обращения к данным служит либо строб-сигнал адреса строки (RAS), либо строб-сигнал адреса столбца (CAS). Сначала вводится и фиксируется адрес строки, а затем вводят адреса столбцов. В страничном режиме для каждого бита подается новый адрес столбца, как показано на рис.5.26. Обычно при работе в этом режиме до того, как памяти потребуется регенерация, удается осуществить чтение лишь части страницы

В сквозном режиме ввод новых адресов столбцов происходит с гораздо более высокой скоростью, что обеспечивает возможность чтения 256-битовой страницы без необходимости регенерации памяти.

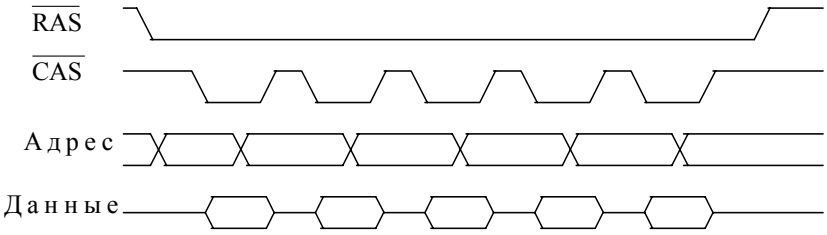


Рис.5.26. Временные отношения при страничной адресации

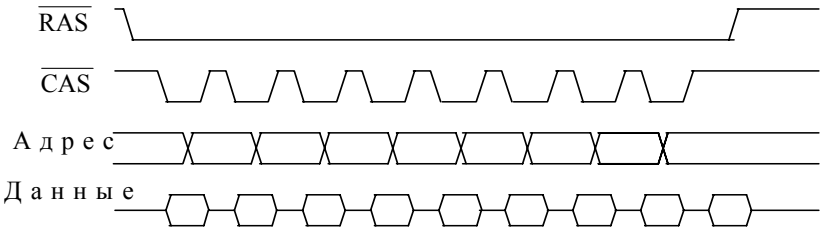


Рис.5.27. Временные соотношения при сквозной адресации

Как видно на рис.5.27 использование сквозного режима позволяет уменьшить время обращения примерно до одной трети типовой длительности времени чтения/записи. Полубайтовый режим дает возможность быстро считать из памяти последовательность из четырех бит; пока на линии $\overline{\text{RAS}}$ имеет место низкий уровень, на линию $\overline{\text{CAS}}$ для считывания каждого бита подается импульсный сигнал низкого уровня. Для последовательной адресации четырех битов, образующих полубайт, младшие части адресов строки и столбца подвергаются инкременту внутри запоминающего устройства. В расширенном варианте реализации полубайтового режима может быть быстро осуществлено последовательное считывание восьми бит. Длительность цикла памяти может снижаться примерно до одной трети типового времени чтения/записи, но, как правило, средняя величина этой длительности составляет около половины типового времени чтения/записи (рис.5.28).

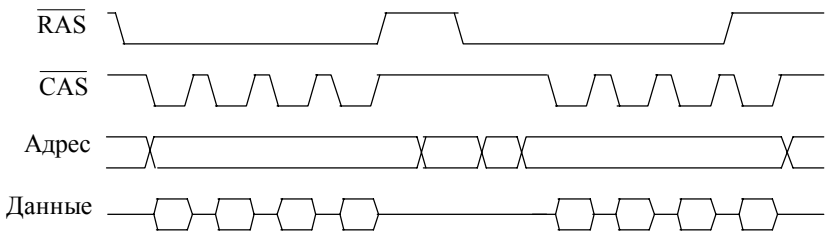


Рис.5.28. Временные соотношения при полубайтовой адресации

При работе в режиме статического дешифрирования столбцов с помощью сигнала $\overline{RAS}$ фиксируется адрес строки, после чего прямо с адресной шины, как показано на рис.5.29 считывается адрес столбца. За адресами столбцов следуют выходные данные. Сигнал $\overline{CAS}$ здесь уже не нужен и может быть заменен управляющим сигналом разрешения кристалла.

Регенерация осуществляется обычно одним из самых распространенных способов "только $\overline{RAS}$". При использовании этого способа для каждой строки памяти подается адрес, фиксируемый сигналом $\overline{RAS}$.

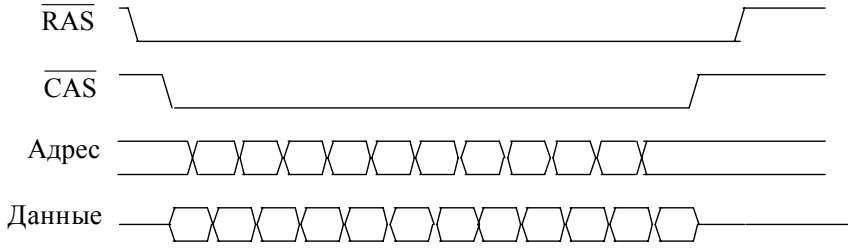


Рис.5.29. Временные отношения при статической дешифрации столбцов

Для формирования этого регенерационного адреса используется счетчик регенерации, который последовательно производит перебор адресов строк и, тем самым, восстанавливает хранимую информацию в матрице запоминающих элементов.

Высокая скорость обработки в МПС требует, чтобы в оперативной памяти хранились большие объемы перерабатываемой информации, однако увеличение емкости ЗУПВ связано со снижением их быстродействия. Применение механизма конвейеризации при построении современных микропроцессоров позволило значительно увеличить их тактовую частоту. Вследствие этого процесс выборки или записи элемента информации в ОЗУ по длительности превышает цикл МП, при этом на каждую выборку элемента информации из памяти обычно затрачивается несколько процессорных циклов ожидания, что приводит к значительному снижению быстродействия системы. Чтобы увеличить пропускную способность ОЗУ необходимо уменьшить число обращений МП к памяти.

Прямой способ сокращения числа обращений состоит в увеличении числа битов, передаваемых по шине параллельно. При выборке широким словом за одно обращение к ОЗУ производится одновременная запись или считывание нескольких команд и слов данных по широкой шине. Широкое слово заносится в буферную память или регистр, где расформировывается на отдельные команды или слова данных, которые могут последовательно использоваться МП без дополнительных обращений к ОЗУ. Разрядность шин микропроцессоров была сначала увеличена с 8 до 16, а затем до 32 бит.

Среди микропроцессоров нового поколения разрядность шин составляет 64 бит, что позволяет строить системы с высокой пропускной способностью шин памяти.

Другим более распространенным способом повышения пропускной способности ОЗУ является методика, называемая расслоением памяти. При этом способе на физическом уровне память строится в виде нескольких модулей с автономными схемами адресации, записи и считывания (один из модулей показан на рис.5.30).

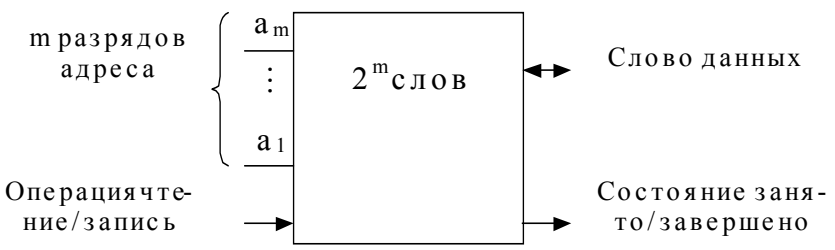


Рис.5.30. Модуль памяти

Схема управления памятью организует одновременный доступ ко всем модулям и последовательной коммутацией элементов информации на общую шину (или наоборот). Обращения к различным модулям могут перекрываться, образуя своеобразный конвейер. Известно несколько вариантов организации расслоения. Наиболее часто используется расслоение в соответствии с младшими разрядами, а также разбиение памяти на память программ и память данных.

На рис.5.31,а показана структура памяти с расслоением в соответствии с младшими разрядами, а на рис.5.31,б временная диаграмма для чтения группы слов (диаграмма для записи группы слов выглядит аналогично).

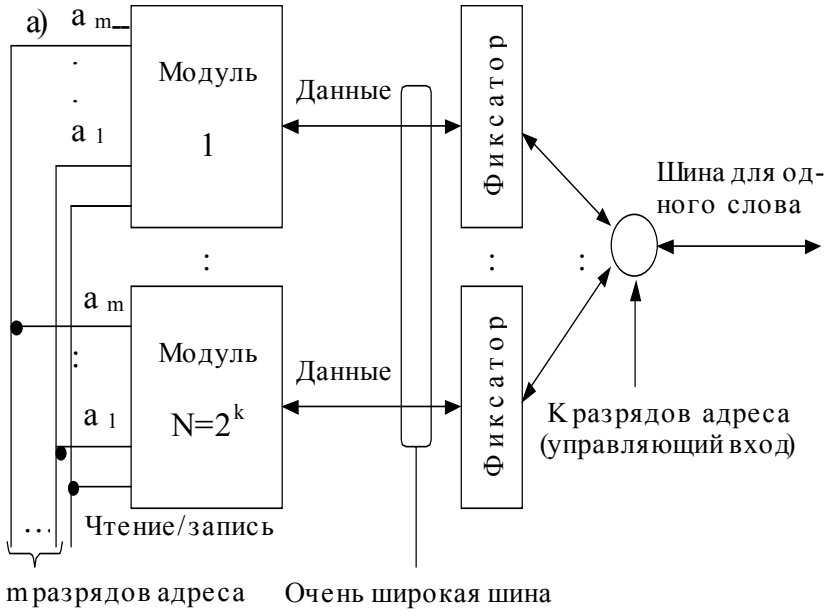
Структура памяти приведенная на рис.5.31,а обеспечивает доступ к N словам параллельно при каждом запросе. Если $N = 2^k$ и $M = 2^m$, то m разрядов из $m + k$ разрядов адреса посылаются всем модулям памяти, а младшие k разрядов адреса используются для выбора, какое из N слов читать первым

При каждом обращении к системе памяти прочитываются одновременно N последовательных слов. Эти слова помещаются в фиксаторы и при следующем обращении последовательно выбираются на шину шириной в одно слово, как показано на рис.5.31,б. Из диаграммы видно, что время выборки слов при такой организации будет в N быстрее, чем время доступа к отдельному модулю.

Этот способ расслоения памяти в соответствии с младшими разрядами наиболее эффективен, когда обращения к памяти осуществляется последовательно. Это характерно для архитектуры ОКОД конвейерного типа, а также для выборки векторов в векторных процессорах.

Для большинства машин класса ОКОД и задач научно-технического характера доля команд в программе, требующих ссылок к находящимся в ОЗУ данным, превышает 50%. Это приводить к поочередному обращению к командам и данным, расположенных в памяти. В этом случае, для повышения пропускной способности ОЗУ используется двунаправленное расслоение памяти, при котором программы и данные размещают в разных модулях.

Разделение памяти на память команд и память данных широко используются в системах управления и



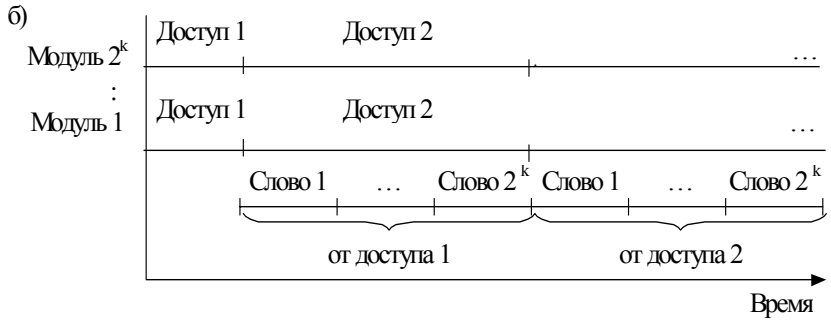


Рис.5.31. Расслоение памяти: а) структура памяти;
б) временная диаграмма его работы

обработки сигналов. В таких системах в качестве памяти команд нередко используются постоянные запоминающие устройства, которые значительно уменьшают стоимость основной памяти, и делает такое разделение памяти весьма эффективным.

5.3.3. Кэш-память

Концепция кэш памяти является частью общей концепции иерархии памяти, в которой более быстрая, но менее емкая память размещается вблизи процессора, а более медленная, но более емкая память поддерживает ее. Она базируется на локальности ссылок.

Локальность ссылок наблюдается в большинстве программ, где основная масса ссылок на коротком промежутке времени относится к некоторым окрестностям ячеек памяти, называемым рабочими множествами. Такое рабочее множество медленно перемещается в памяти по мере исполнения программ. Быстрая и менее емкая память в иерархии отслеживает эти рабочие множества, тогда как более медленная память содержит большие блоки программы, которые вводится на быстрый уровень по мере необходимости. При использовании этой концепции общая скорость системы памяти приближается к скорости ее самого быстрого уровня, а общая емкость – к наиболее медленному и обычно наиболее дешевому уровню. В микропроцессорных системах кэш-память представляет собой быстродействующее запоминающее устройство, размещенное в одном кристалле с процессором или же внешнее по отношению к кристаллу процессора, но находящееся на той же плате вычислителя, и служит высокоскоростным буфером между процессором и относительно медленной памятью. Этот буфер совершенно незаметен и недоступен пользователям и не может быть непосредственно адресован.

Таким образом кэш-память обеспечивает обмен информацией между процессором и памятью с более высокой скоростью и меньшими затратами по сравнению с отдельной большой быстродействующей памятью, а также создает впечатление, что большая память работает со скоростью кэша. До тех пор пока для современной памяти будут сохраняться различия в быстродействии и скорости, подход с использованием кэш-памяти будет оставаться экономически эффективным решением проблемы согласования скоростей процессора и основной памяти.

Для микропроцессорных систем, снабженных страничной виртуальной памятью с перемещением по запросам используется ассоциативной кэш. В таком кэше базовой единицей информации является строка, которая перемещается между основной памятью и кэш-памятью. Когда процессор желает получить доступ к элементу данных, он обращается к кэш-памяти для поиска в нем этих данных. Этот поиск по своей природе является ассоциативным, каждая строка (или набор строк, образующий блок) в кэш-памяти имеет связанный с ней адресный тег, управляющий адрес элемента данных (метка или указатель) размещенного в основной памяти. Эти теги одновременно сравниваются с выработанным процессором адресом и при совпадении указывают на попадание. При несовпадении возникает промах, требующий выбора строки в кэш-памяти для приема новых данных из

основной памяти. Этот выбор управляется специальной аппаратурой, содержащей информацию о замене строк. Это может быть организовано с помощью тегов состояния, являющихся элементом строки, в виде очередей использования строк кэш-памяти или в виде счетчиков.

Логический адрес, вырабатываемый процессором, вводится в кэш-память и пропускается через блок отображения адресов для выделения из него адреса, содержащего в поле метки, а также адреса строк или набора строк кэш-памяти, где должен вестись поиск. После идентификации адреса строки кэш-памяти, его адресный тег сравнивается с полем метки логического адреса вырабатываемого процессором. При совпадении оставшаяся часть логического адреса поступает на селектор данных, где происходит выбор байта или слова для передачи его в процессор. Если же возникает промах, то блок обработки промахов передает содержимое поля метки логического адреса основной памяти и осуществляет выбор строки для приема новой информации из нее. Затем, как и прежде, желаемая информация выделяется и передается в процессор. На рис. 5.32 показана базовая организация любой кэш-памяти.

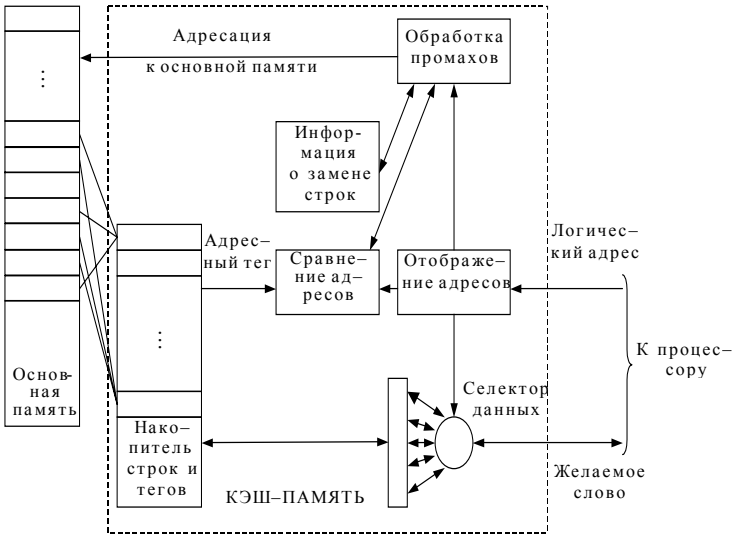


Рис.5.32. Базовая организация кэш-памяти

Существуют два основных способа отображения кэша на основную память. Первый способ называется полностью ассоциативным, при котором любая строка памяти может находиться в любой строке кэш-памяти и входить в любые комбинации с ней. При этом если имеется нехватка информации в кэш-памяти, то любая из ее строк может быть загружена новой информацией. Недостатком этого способа является то, что каждая строка в этом случае имеет свой компаратор и при увеличении количества строк увеличивается объем аппаратных затрат и стоимость такой памяти.

Второй способ представляет структуру кэш-памяти с прямым отображением. Каждая строка кэш-памяти может содержать слово только из определенного подмножества слов основной памяти, а подмножества, соответствующие различным строкам кэш-памяти не пересекаются. Реализация такой кэш-памяти является простой, так как прямое отображение использует поразрядное отображение для выбора подмножества памяти. Например, если в логическом адресе, вырабатываемым процессором содержится N разрядов, то младшие n из них выбирают, в какую строку кэш-памяти она может копироваться.

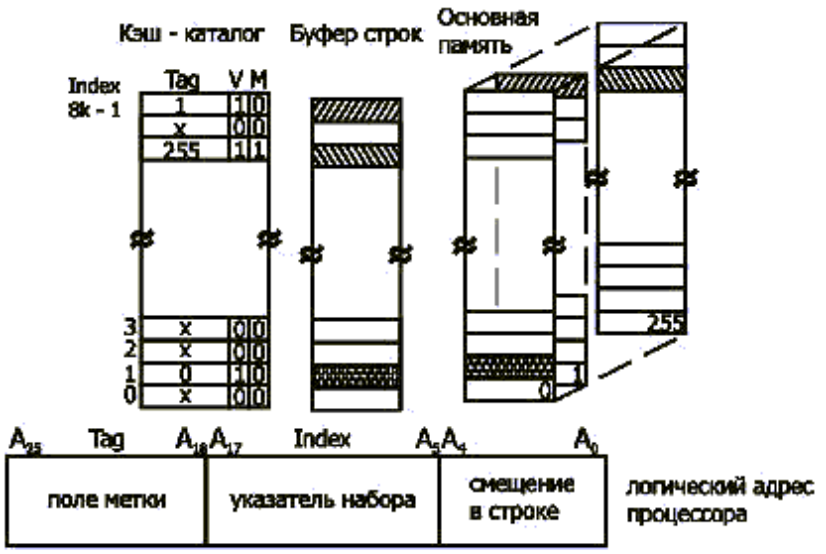


Рис.5.33. КЭШ прямого отображения

Таким образом, все слова основной памяти, имеющие одинаковые младшие n разрядов в своем адресе подпадают в одно и то же подмножество и могут появляться только в одной строке кэш-памяти.

На рис.5.33 представлена организация кэш-памяти с прямым отображением. Такая кэш-память имеет самую простую аппаратную организацию и применяется во вторичном кэше большинства микропроцессорных систем.

Накопитель этой кэш-памяти состоит из признаковой памяти и буфера строк. Признаковая память, представленная кэш-каталогом, который содержит теги кэш-строки и битов состояния V, M. Бит V (Valid) определяет доступность данной строки, и что ее содержимое совпадает с содержимым соответствующего ей блока основной памяти (на рис.5.33 это показано соответствующей штриховой). Когда этот бит, находится в состоянии «0» (Invalid), строка становится недоступной в кэш-памяти. Чтение этой строки приводит к считыванию соответствующего блока из основной памяти в кэш-память, а при записи - генерируется цикл обращения к основной памяти. Бит M (modified) показывает, что ее содержимое отлично от содержимого соответствующего блока основной памяти. Доступ к M-строке (чтение и запись) осуществляется без обращения к основной памяти через, внешнюю шину. Сброс этого бита будет произведен только после выгрузки содержимого этой строки в основную память.

Буфер строк непосредственно служит для хранения данных и имеет емкость 256 Кбайт с размером строки 32 байта, определяемой младшими битами (A0-A4) логического адреса. Кэш накопитель делится на строки, количество которых равно 8К строк (256К/32). Указатель кэш-строки задается полем Index логического адреса, формируемого процессором. Кэшируемая основная память условно разбита на страницы, размер которых совпадает с размером кэш-памяти (256Кбайт). Количество страниц, кэшируемой основной памяти соответствует длине поля тега логического адреса, что соответствует 8 битам, и состоит из 256 страниц.

Таким образом, объем кэшируемой основной памяти равен, (256*256) 64Мбайт. На основе такой организации построена кэш (второго уровня) системной платы для Pentium.

В начале каждого обращения к кэшируемой памяти контроллер считывает ячейку каталога с заданным адресом, сравнивает биты адреса тега со старшими битами полем метки логического адреса и анализирует признак действительности V. Если в результате анализа выявляется, что требуемый блок не находится в кэше, то генерируется цикл обращения к основной памяти (случай кэш-промаха), в противном случае запрос обслуживается кэш-памятью.

При промахе после считывания основной памяти приемником информации новое данное помещается в кэш-строку (если бит M=0), а в ее тег помещается, старшие биты логического адреса и устанавливается признак действительности данных (V бит).

Этот способ отображения наиболее часто используется в микропроцессорных системах вследствие своей простоты и низкой стоимости. Так для его реализации требуется один компаратор и, что еще более важно, сам накопитель в кэш-памяти может иметь форму стандартной памяти, которая может реализовываться в виде статических ЗУ с высоким быстродействием. Он имеет ряд недостатков. Один из них является ограниченное число комбинаций блоков основной памяти, которые могут находиться в кэш-строке. Другой недостаток заключается в следующем. Если в процессе выполнения программы, процессору поочередно будут требоваться блоки памяти из разных страниц, основной памяти (на рис. 5.33 они расположены на одной горизонтали в различных страницах), то кэш будет работать интенсивно, но не эффективно. Очередное обращение будет замещать данные, считанные в предыдущем или требующиеся в последующем обращениях. В этом случае будет наблюдаться череда сложных кэш-промахов. Переключение страниц в многозадачных операционных системах снижает кэш-попадания, что тоже приводит к снижению производительности системы.

Наборно-ассоциативная структура кэш является комбинацией двух методов: прямого и ассоциативного. Он позволяет каждому блоку кэшируемой основной памяти претендовать на одну из нескольких строк кэша, объединенных в набор. При такой организации имеется несколько параллельно и согласованно работающих каналов прямого отображения, где контроллеру кэш-памяти приходится принимать решение о том, в какую из строк набора помещать очередной блок данных.

На рис.5.34 показан двухканальный наборно-ассоциативный кэш, который содержит два банка памяти и тегов. Номер набора, в котором может отображаться блок основной памяти, определяется полем Index логического адреса процессора. Строка набора, в который загружается требуемый блок основной памяти, определяется сравнением адресных тегов кэш-каталогов с полем метки логического адреса процессора. Сравнение происходит параллельно для обоих каналов. Из рис.5.34 видно, что блок расположенный в 1023 странице основной памяти, отображается в банк А с номером набора равным 1. На рисунке заштрихованные блоки основной памяти также отображаются в кэш-строки банка А или В согласно адресным тегам, содержащимся в кэш-каталогах каждого канала.

Наборно-ассоциативная структура широко применяется в первичных кэшах современных процессорах. Объем кэшируемой памяти определяется так же, как кэш с прямым отображением, но здесь будет фигурировать объем одного банка и разрядность относящихся к нему ячеек тега. В нашем случае объем кэш памяти составляет 16 Кбайт (два банка по 8Кбайт), а объем кэшируемой памяти – 8Мбайт.

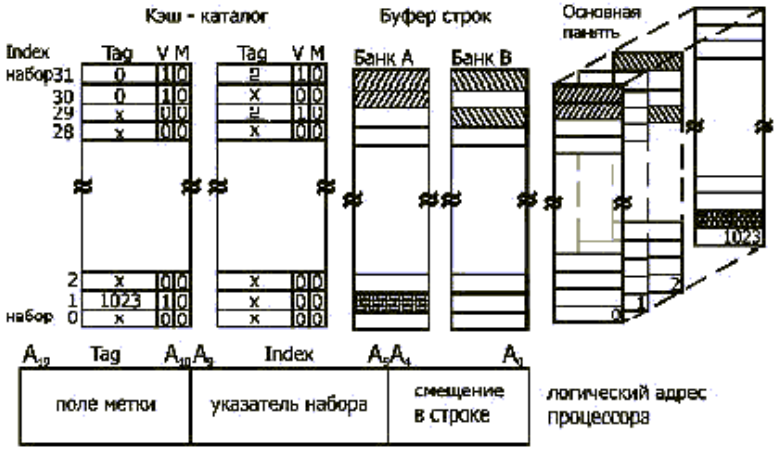


Рис.5.34.Двухканальный наборно-ассоциативный КЭШ

У кэша, использующий ассоциативный метод отображения, любая его строка может отображать любой его блок памяти, что существенно повышает эффективность использование его ограниченного объема. При этом все биты логического адреса кэшированного блока, за вычетом бит, определяющих положение (смещение) данных в строке, хранятся в памяти тегов. В этом случае требуется сравнение содержимого поля метки логического адреса процессора с адресами тегов всех строк кэш-памяти. Последовательный перебор ячеек памяти тегов отпадает – на это уходит слишком много времени. Для сохранения высокого быстродействия необходимо параллельное сравнение всех ячеек памяти тегов, что является сложной задачей. При этом увеличивается объем аппаратных затрат и стоимость памяти. Реализация этой памяти возможна только для небольших объемов кэша первого уровня. Применение полностью ассоциативной кэш-памяти больших объемов пока не предвидится.

При возникновении промаха во время обращения к кэш-памяти возникает необходимость перемещения новой информации из основной памяти. Решение этой задачи возлагается на блок обработки промахов (рис.5.32), который определяет способ обмена новой информации, называемый свопингом, и метод замены строк, который определяет порядок удаления строки или набора строк из кэш-памяти.

В реальных структурах существует два широко используемых подхода для решения первой проблемы. Первый подход использует алгоритм сквозной записи. При этом, каждый раз при появлении запроса на запись по некоторому адресу, обновляется содержимое области по этому адресу, как в кэш, так и в основной памяти, даже если копия содержимого по этому адресу находится в кэш-памяти. При возникновении запроса на запись по адресу, относящегося к области, содержимое которой не находится в кэш-памяти (наличие промаха), новая информация записывается просто на место строки в кэш-памяти, которое предполагается переслать в основную память. Такое постоянное обновление содержимого основной памяти и кэша, при каждом запросе на запись, поддерживает информацию, находящуюся в основной памяти, в обновленном состоянии.

Данный алгоритм отличается простотой реализации и его преимущество проявляется особенно отчетливо в мультипроцессорных системах с разделенными кэшами и общей памятью. Так как содержимое основной памяти все время поддерживается в обновленном состоянии, то для отдельных процессоров отпадает необходимость в справочном поиске для нахождения последней копии и замены конкретного слова. Недостатком этого алгоритма является отсутствие минимизации доли обращений к основной памяти. При увеличении числа записей в кэш-память, увеличивается число обращений к основной памяти, что приводит к снижению эффективного быстродействия памяти.

Второй подход базируется на алгоритме обратной записи. Он минимизирует число обращений в основную память приходящее на каждое обращение к кэш-памяти. В случае запроса на запись, вырабатываемого процессором, и наличии попадания изменяется содержимое только кэш-строки и блокируется запрос к основной памяти. Для предотвращения утери этой обновленной информации производят обратную запись в основную память только в момент удаления этой строки из кэш-памяти. Для реализации этого алгоритма в кэш-строку вводится дополнительный разряд, называемый разрядом модификации М и входящего в строку тега состояния. Этот разряд сбрасывается в нуль при исходной загрузке и устанавливается в единицу при изменении содержимого кэш-строки. При замене кэш-строки осуществляется опрос разряда модификации и, если он установлен в единицу, содержимое этой строки копируется обратно в основную память. При этом алгоритме требуется более сложная организация управления и нарушается целостность информации на период нахождения обновленной строки в кэш-памяти. При этом уменьшается число обращений к основной памяти, что в целом приводит к увеличению быстродействия памяти.

Этот метод эффективно используется в однопроцессорных вычислительных системах. Использование его в мультипроцессорных системах с разделенными кэшами и общей

памятью связано с большими трудностями. В такой системе возможно наличие такой ситуации, где одновременно могут присутствовать несколько одноименных "копий" кэш-строк с различным содержанием. Для разрешения этой конфликтной ситуации используются специальные меры, требующие дополнительных затрат.

При решении второй проблемы замены строк в кэш-памяти, пользуются двумя распространенными методами.

Первый метод базируется на стратегии FIFO ("первый пришел – первым заменяется"). При этом методе новая строка вводится на место ранее других внесенных в кэш. Реализация этого метода основывается на введение специального счетчика, который при каждом промахе увеличивается на единицу и его содержимое указывает, какая строка должна быть заменена. При этом из стека удаляется строка, находящаяся в начале стека; список смещается на единицу вперед, а новая строка вводится в конец списка. Его достоинством является простота реализации.

Второй метод является модификацией стратегии FIFO, использующий алгоритм LRU (Least Recently Used) замена наименее используемой строки. При этом методе модификация списка происходит при каждом попадании, а не при каждом промахе. Адрес строки, к которой произведен доступ, перемещается из того места, где он находится, в конец списка, остальная часть поднимается в начало для заполнения пробела. При промахе удаляется строка, находящаяся в начале списка, а новая, как и в предыдущем случае помещается в конец списка. Таким образом, удаляется та строка, на которую дольше всего не было никаких ссылок. Этот алгоритм является одним из самых сложных способов, но он обеспечивает большой процент попадания.

Контрольные вопросы

1. Укажите основные направления ускорения вычислений.
2. Какие существуют типы отношений между элементами вычислительных конструкций, образующих алгоритм? Определите свойство вложенности алгоритма.
3. Каким образом влияют на архитектуру вычислительных систем структура алгоритма?
4. Определите и дайте характеристики основным классам архитектур вычислительных систем по Флинну.
5. Какие механизмы синхронизации используются при построении мультипроцессорных систем? Укажите основные их структуры.
6. Укажите основные способы программной организации вычислительных машин и дайте им краткую характеристику.
7. Дайте краткую характеристику фон-неймановским машинам и укажите их достоинства и недостатки.
8. В чем заключается суть конвейерной обработки и как определяется оценка их по производительности?
9. Определите основные этапы исполнения типовой команды и укажите их функции, выполняемые на этих этапах.
10. Выберите небольшой набор команд и проведите их разбиение. Дайте сравнительный анализ этого разбиения.
11. Укажите основные факторы, влияющие на производительность конвейера.
12. Определите основные виды зависимостей по данным и дайте им краткую характеристику.
13. Укажите основные аппаратные методы борьбы против зависимостей по управлению.
14. Какие существуют методы борьбы, устраняющие зависимости по данным?
15. Укажите основные элементы при структурной организации микропроцессора в виде конвейера.

16. С какой целью вводится очередь команд в конвейере, и какие функции он выполняет?

17. Чем объясняется необходимость введения многоуровневой памяти в вычислительных системах? Перечислите ее уровни.

18. В чем заключается сущность концепции виртуальной памяти? Определите основные виды организации этой памяти.

19. Укажите основные отличия статической памяти от динамической памяти (ЗУПВ).

20. Какие методы адресации используются при организации интерфейса динамической памяти?

21. Укажите способы увеличения пропускной способности основной памяти и дайте им краткую характеристику.

22. В чем заключается сущность концепции кэш-памяти?

23. Укажите основные способы отображения кэш на основную память.

24. Какие алгоритмы свопинга используются для построения кэш-памяти? Дайте им краткую характеристику.

25. Какими способами осуществляется замена строк при возникновении промаха в кэш-памяти?

Глава 6

ОРГАНИЗАЦИЯ СОВРЕМЕННЫХ МИКРОПРОЦЕССОРОВ

Микропроцессоры первых поколений строились на фон-неймановских архитектурных принципах и оставались неизменными до конца 70-х годов. В начале 80-х годов появились первые отклонения от этих традиционных принципов, которые были использованы при построении 16-разрядных микропроцессоров. Впервые в таких микропроцессорах начали применять конвейеризацию при структурной организации процессора (рис.5.22) и расширенные наборы обрабатывающих устройств в форме сопроцессоров.

Эволюция развития новых архитектурных решений наглядно прослеживается на базе семейства микропроцессоров 80×86 фирмы Intel, одной из ведущих производителей в данной области. Все процессоры, совместимые с набором команд семейства 80x86, являются CISC-процессорами, хотя некоторые (в последних моделях) могут иметь элементы RISC-архитектуры.

6.1. 32-разрядный микропроцессор

Микропроцессор 80386 – первый 32-разрядный микропроцессор в семействе 80×86. Ряд усовершенствований, введенные в архитектуру МП, позволили улучшить характеристики и расширить функциональные возможности микропроцессора по сравнению с 16-разрядными его моделями. В таблице 6.1 приведены основные характеристики микропроцессоров семейства 80×86.

Таблица 6.1

Характеристика	**Тип микропроцессора**		
	I 8086	80286	I80386
Технология	nMOS	HMOS	HCMOS
Количество транзисторов в кристалле, тыс.	30	130	275
Количество контактов корпуса	40	68	132

Тактовая рабочая частота, МГц	5; 8; 10	6; 8; 10; 12	16; 20; 25; 33; 40
Максимальная производительность млн. опер./сек.	1,66	3,4	4
Потребляемая мощность, Вт	1,7	3,0	2,3
Очередь команд, байт	6	6	16
Количество регистров общего назначения	8	8	8
Принцип организации виртуальной памяти	-	Сегментная	сегментная, страничная
Физическое адресное пространство, байт	1М	16М	4Г
Виртуальное адресное пространство, байт	-	1Г	64Т
Размер операндов, бит	8; 16	8;16	8; 16; 32
Шина адреса/данных, бит	20/16, мульти-плексир.	24/16, раздельная	32/32, раздельная
Тип математического сопроцессора	I8087	I80287	I80287, I80387

Основные отличительные характеристики МП 80386 следующие:

− встроенный диспетчер памяти позволяет использовать как сегментную, так и страничную организацию памяти;

− виртуальное адресное пространство увеличено до 64 Терабайт;

− 32-разрядное адресная шина расширяет физическое адресное пространство памяти до 4 Гбайт;

− режим виртуального 8086 обеспечивает возможность использования прикладных программ и операционных систем МП 8086 в защищенном режиме, как часть виртуальной задачи;

− возможность динамического изменения ширины шины данных позволяет эффективно взаимодействовать как с 32-разрядными, так и с 16-разрядными устройствами памяти и ввода-вывода;

− обеспечена возможность взаимодействия с математическими сопроцессорами 80287, 80387;

− средства самоконтроля обеспечивают проверку до 50% оборудования микропроцессора;

− расширена возможность отладки за счет введения шести специальных регистров;

− длина очереди команд в буфере предвыборки увеличена до 16 байт; с учетом средней длины команды 3,2 байта в буфере может находиться до 5 команд.

Структурная организация МП 80386 показана на рис.6.1.

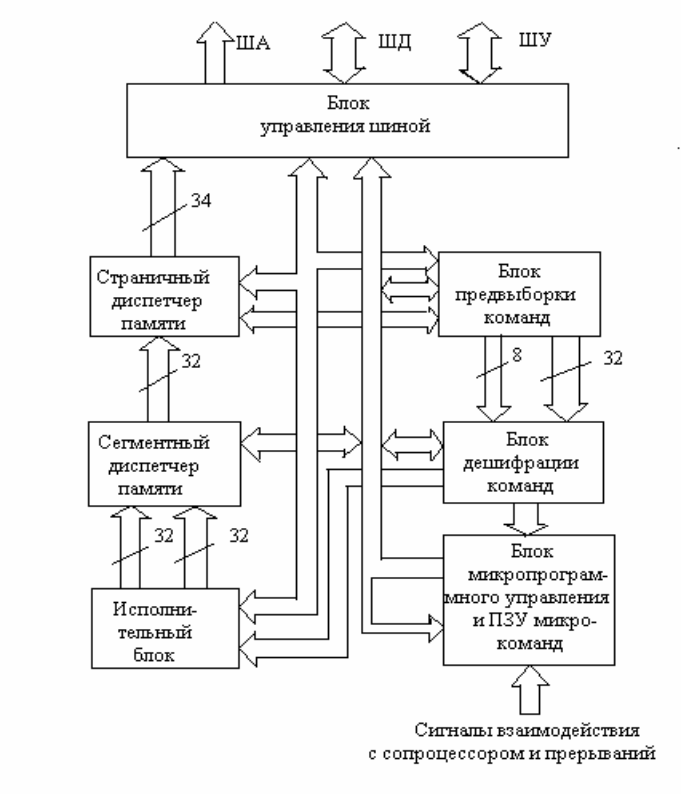


Рис. 6.1. Структурная организация МП 80386

В состав микропроцессора входят следующие функциональные блоки:

- блок управления шиной;
- блок предвыборки команд;
- блок декодирования команд;
- исполнительный блок;
- сегментный диспетчер памяти;
- страничный диспетчер памяти;
- блок микропрограммного управления, включая ПЗУ микрокоманд.

Функциональные блоки (ФБ) объединены 32-разрядными внутренними шинами данных и адреса. Их взаимодействие синхронизируется сигналами, передаваемыми по шине управления.

Блок управления шиной обеспечивает взаимодействие микропроцессора с памятью, сопроцессором и устройствами ввода-вывода по сигналам запросов на передачу (чтение или запись) данных, поступающим от блока предвыборки команд. Адрес и данные передаются по раздельным внешним 32-разрядным шинам.

Блок предвыборки команд принимает команды от блока управления шиной двойными словами, выстраивая их в 16-байтовую очередь, выдача байтов, из которой осуществляется в соответствии с дисциплиной типа FIFO (First In – First Out, первым пришел – первым обслужен). Каждый байт из очереди, являющийся частью команды, поступает в блок декодирования команд, который преобразует каждую команду в код, включающий в себя адрес микрокоманды, номера используемых регистров, непосредственный операнд, адресное смещение и другие атрибуты. Коды запоминаются в буфере в виде очереди декодированных команд, которая может содержать до 5 команд.

Исполнительный блок выполняет операции над данными в соответствии с кодами команд и содержит все узлы, необходимые для выполнения операций:

- набор регистров;
- 32-разрядное арифметико-логическое устройство;
- 64-разрядный сдвигатель;

– специальную логику для умножения и деления.

Исполнительный блок имеет два входа: первый связан с внутренней 32-разрядной шиной данных; второй, по которому поступает непосредственный операнд и смещение - с декодера команд.

32-разрядные компоненты по двум шинам поступают одновременно на диспетчер памяти сегментов, осуществляющий формирование исполнительного адреса операнда. В сегментном диспетчере памяти производится проверка защиты сегментов, прав доступа, а также формируется линейный адрес операнда в зависимости от режима работы микропроцессора.

В реальном режиме (см. параграф 2.1) страничная адресация не используется, поэтому полученный из сегментного диспетчера памяти линейный адрес является физическим. При страничной организации памяти преобразование линейного адреса осуществляется с помощью кэш-памяти страниц. Каждая страница величиной в 4 Кбайт имеет собственную защиту. Результатом страничной адресации является физический адрес, который поступает в блок управления шиной.

В микропроцессоре 80386 используется принцип двухуровневого управления, при котором, помимо центрального блока управления (микропрограммного), осуществляющего общую координацию работы всех ФБ, каждый ФБ имеет также и собственный (локальный) блок управления.

Рост производительности МП 80386 достигнуто за счет следующих факторов: повышение тактовой частоты; повышения пропускной способности внешней шины; совершенствования внутренней структуры. Применение в МП 80386 технологии КМОП с повышенной плотностью с проектными нормами 1,5 мкМ позволило поднять тактовую частоту до несколько десятков МГц. Увеличение ширины шины данных в два раза при тактовой частоте 16 МГц привело к повышению пропускной способности шины до 32 Кбайт/сек.

Основным классическим методом повышения быстродействия, используемых во всех микропроцессорах семейства 80×86, является конвейеризация. При этом последовательность выполнения команд разделяется на следующие фазы:

– выборка команды;
– декодирование команды;
– формирование адреса операнда;
– выполнение команды;
– сохранение результата.

Конвейер микропроцессора 80386 позволяет совмещать выполнение различных фаз нескольких команд; при этом блоки микропроцессора работают параллельно. Каждая фаза выполняется соответствующим операционным блоком. Различные фазы выполнения команд могут разделять во времени один и тот же функциональный блок. Например, блок управления шиной может быть использован как для выполнения фазы выборки команды, так и для выборки операнда из памяти. Время выполнения команды зависит от типа команды, занятости блоков, времени доступа к запоминающему устройству (если команда обращается за операндом в память) и других факторов. Конвейер МП 80386 состоит из пяти ступеней: блока управления шиной, блока предвыборки команд, блока декодирования команд, блока управления памятью и исполнительного блока.

В идеальном случае все ступени конвейера должны быть сбалансированы по времени выполнения операций. Однако в микропроцессоре 80386, например, время дешифрации команды может изменяться в зависимости от длины команды. Поэтому для достижения максимальной эффективности конвейеризации блок предварительной выборки и блок декодирования команд содержат буферы типа FIFO.

Существенное влияние на работу конвейера 80386 оказывает исполнительный блок, который используется при выполнении большинства команд. Поэтому для ускорения

выполнения наиболее сложных операций (например, сдвига, умножения и деления) используется 64-разрядный сдвигатель и специальная логика умножения и деления.

Для команд, в которых осуществляется обращение к памяти, введен специальный режим – "Ранний старт команды", в котором исполнительный блок приступает к выполнению следующей команды до завершения текущей команды. Этот режим обеспечивает увеличение производительности микропроцессора примерно на 9%.

Снижение производительности микропроцессора происходит в результате нарушения последовательности выполнения команд программы, которое может быть вызвано выполнением команд безусловного и условного переходов, вызова подпрограммы, появлением аппаратных и программных прерываний и исключительных ситуаций. Доля таких команд обычно составляет 15-25% от их общего количества.

Стремление достигнуть максимальной производительности микропроцессора в значительной степени сдерживается реальными ограничениями, связанными с площадью, занимаемой логикой на кристалле, и выходом годных кристаллов. Поэтому разработчикам микропроцессоров приходится принимать компромиссные решения, выбирая каждый раз между производительностью и стоимостью. Примером такого компромиссного решения является размещение в микропроцессоре 80386 диспетчера сегментной и страничной памяти. Поскольку преобразование адресов осуществляется при выполнении примерно 50% команд, поэтому целесообразно использовать имеющуюся площадь кристалла для реализации адресации через кэш страниц, а не кэш команд и данных. Структурная организация 32-разрядного МП 80486 идентична организации МП 80386, а отличия их описаны в параграфе 2.1.

6.2. Транспьютеры

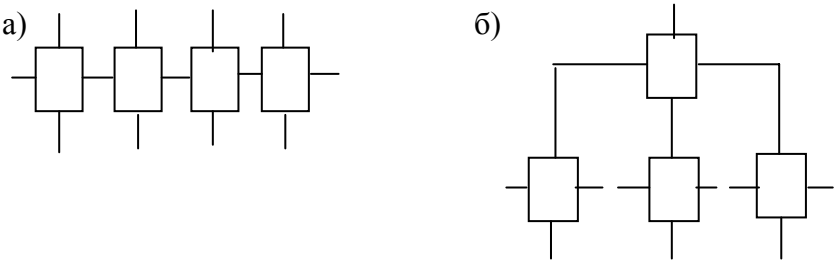
Транспьютеры представляют собой микропроцессоры, рассчитаны на работу в мультипроцессорных системах. На базе их стало возможным построение машин класса МКМД, позволяющие реализовывать алгоритмические структуры со сложными связями (рис.5.9,б). Внутренняя структура транспьютера и специально разработанный для него язык программирования Оккам позволяют явно управлять параллелизмом. С другой стороны, архитектура и средства связи транспьютера эффективно реализуются в технологии изготовления интегральных схем очень высокой степени интеграции (СБИС). Благодаря четырем линиям связи транспьютеры могут соединяться друг с другом различными способами, образуя сети транспьютеров. Некоторые разновидности этих сетей показаны на рис. 6.2.

При реализации N – шагового конвейера имеется 2*N свободных линий связи, что приводит к неоптимальному использованию средств связи транспьютера (рис. 6.2, а). При этом конвейеру присущи все недостатки машин класса МКОД

Древовидная топология подходит для иерархических процессов, таких как данных с передачей информации вверх по дереву и сортировка данных с передачей информации вниз по дереву (рис. 6.2, б).

Двухмерный массив используется для данных, имеющих структуру массива, как, например, образы и мат-
рицы, и применяется для обработки информации на низком уровне (рис. 6.2, в).

а) б)



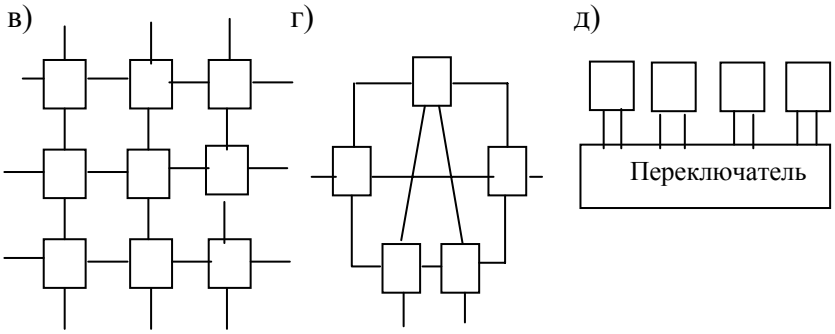


Рис. 6.2. Транспьютерные сети
а) конвейер; б) дерево; в) двухмерный массив; г) 5-узловая; д) переключаемая сеть

Топология двухмерного массива эквивалентна топологии матричных процессоров в машинах класса ОКМД, и в них полностью используются четыре линии связи транспьютера. Для такой топологии применима методика отображения вычислительного процесса, разработанная для машин класса ОКМД. При увеличении размерности двухмерного массива возникает проблемы передачи данных в массив и нерегулярные передачи информации. Например, если верхний левый процессор должен взаимодействовать с правым нижним процессором и одновременно правый процессор – с левым нижним процессором, то данные или сообщения маршрутизируются через процессоры в центре массива, который могут стать «узким местом». Для коммуникаций более высокого уровня разработаны управляющие программы по обеспечению связи, основанные на маршрутизации сообщений или методах переключений пакетов, но при этом меняется пропускная способность и увеличивается задержка. Эти проблемы отсутствуют в сетях, топология которых показана на рис. 6.2, г, где в узел может входить максимум пять транспьютеров.

Для квадратной матрицы процессоров размерности N число процессоров равно N^2, где 4*N линий связи являются внешними, а 2N(N-1) – внутренними. Ввод-вывод данных в таких матрицах ограничен отношением линий связи с внешним миром к числу внутренних связей. В этом случае при увеличении размерности матрицы N накладные расходы при передаче данных в центр увеличивается с коэффициентом:

$$N : (N^2 - N)^2.$$

Для уменьшения накладных расходов при передаче данных в сетях и для обеспечения максимальной гибкости можно использовать переключаемые сети (рис. 6.2, г). Использование переключаемых сетей обеспечивают уменьшение времени связи путем уменьшения числа промежуточных процессоров, через которые должны проходить сообщения или данные.

Достоинствами переключаемых сетей является простота программирования, при котором разработчик имеет больше возможностей при выборе оптимальных методов отображения алгоритмов на аппаратуру, а также высокая степень надежности, т.к. вышедшие из строя процессоры могут быть обойдены, или исключены из сети.

6.2.1. Транспьютерное семейство

Семейство транспьютеров фирмы INMOS состоит из трех моделей: T212, T414, T800. Первым элементом этого семейства, который стал широкодоступен для пользователей в 1985 г., стал транспьютер T414. Транспьютер T414 представляет собой 32-разрядный МП, структурная схема, которой показана на рис. 6.3.

В состав транспьютера входит 32-разрядное центральное процессорное устройство (ЦПУ) с архитектурой RISC – процессора, внутри кристальное ЗУПВ емкостью 2 Кбайт, четыре последовательных быстродействующих канала связи, интерфейс памяти, которая

осуществляет доступ к внешней памяти с диапазоном физических адресов 4 Гбайт, и таймер с разрешающей способностью 1 мкс. Внутренняя архитектура транспьютера соответствует фон-неймановским принципам, т.е. включает единую шину адресов и данных, связывающую ЦПУ со встроенной внешней по отношению к кристаллу памятью, а также с другими системными компонентами. Транспьютер обладает многими особенностями, определяющими возможность его использования для параллельной обработки и включения его в состав двухмерных процессорных матриц с небольшими добавлениями других компонентов.

При использовании транспьютера Т414 в качестве одиночного МП достигается производительность, большая, чем при работе со многими 32-разрядными МП; при этом, как правило, примерно вдвое меньший объем имеет программа.

Однако в Т414 не заложена возможность работы с виртуальной памятью и отсутствует тонко организованная структура прерываний, не позволяющая обслуживать периферийные устройства, отображенные в пространство
памяти. С использованием специфического для программирования языка Оккам транспьютер успешно работает в составе параллельных мультипроцессоров, но для обеспечения возможности его эффективного

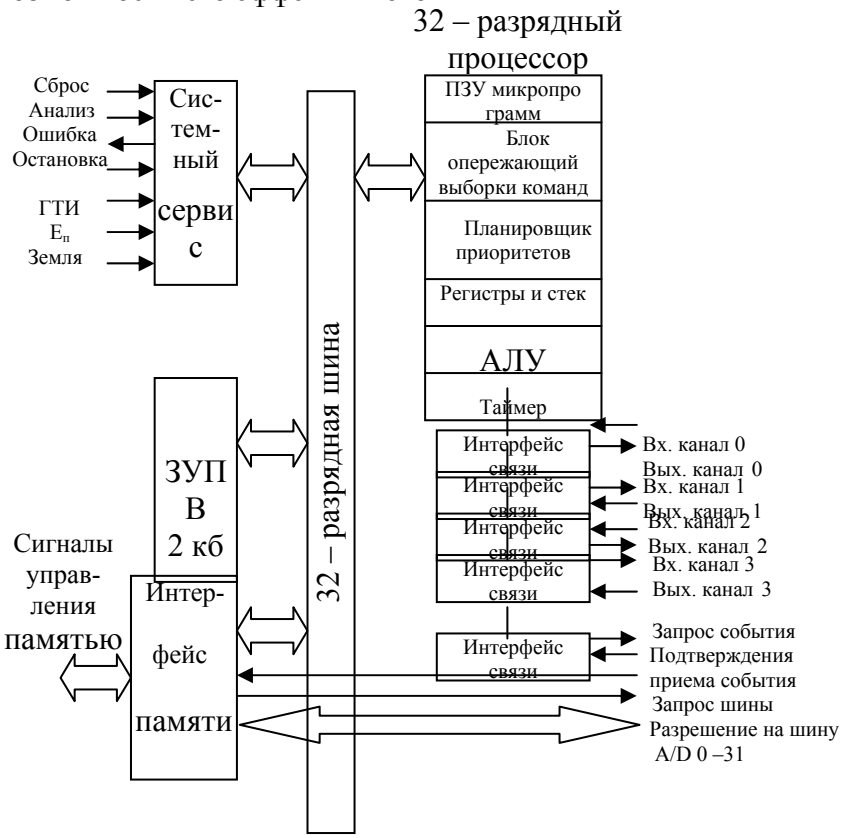


Рис. 6.3. Структурная схема транспьютера Т414

применения в более традиционных целях разработаны компиляторы языков СИ, Паскаль, Ада и Лисп.

Основные характеристики транспьютеров семейства INMOS приведены в таблице 6.2.

Взаимодействие каждого транспьютера с другими транспьютерами и периферийными устройствами осуществляется посредством четырех коммуникационных

Таблица 6.2

Характеристика	Т212	Т414	Т800
Разрядность, бит	16	32	32
Адресное пространство, Кбайт	64	4000	4000
Оперативная память:			

Емкость, Кбайт	2	2	4
Пропускная способность, Кбайт/с	40	80	80
Скорость обработки данных, млн.оп./сек:			
целых чисел	10	20	20
с плавающей точкой	-	0,45	2,25
Каналы связи:			
организация (число каналов * число портов)	4*2	4*2	4*2
скорость обмена, Мбит/с	10	10/20	10/20
Процессор с плавающей точкой (ППТ)	-	-	есть

каналов связи. Механизм блочных ПДП-пересылок применяется во время передачи внутренней или вне кристальной локальной памяти к внешним устройствам по последовательным каналам. Интерфейсы связи и процессор работают одновременно. Использование прямых последовательных коммуникационных каналов делает ненужным арбитраж приоритетов, необходимый для представления шины в мультипроцессорных системах, и исключает проблемы связанные с пропускной способностью шин и их перегрузкой при введении в систему новых процессоров.

Каждый последовательный канал состоит из двух частей, служащих для передачи информации в противоположных направлениях и образующих два Оккам канала. Сообщения пересылаются в виде последовательностей байтов, формат которого представлен на рис. 6.4.

Подтверждение

Рис. 6.4.Формат сообщения при последовательной передаче

Каждый 8-битовый байт предваряется двумя единичными битами и сопровождается одним нулевым битом. В передающий транспьютер подается сигнал подтверждения приема, состоящий из одного единичного и одного нулевого битов и указывающий на то, что принимающий транспьютер готов к получению очередного байта. Чтобы пересылка не прерывалась, принимающий транспьютер должен посылать сигнал подтверждения приема, одновременно, с вводом в него передаваемого байта, как показано на рис.6.4. Для увеличения длины линий связи могут быть применены формирователи и приемники, отвечающие стандарту RS-422, или аналогичные сбалансированные устройства.

Быстродействие каналов связи транспьютеров равно 20 Мбит/сек, но возможно переключение скорости передачи на 10 Мбит/сек, что необходимо для обеспечения совместимости с относительно медленными устройствами системы. Для сопряжения последовательных каналов транспьютера с не транспьютерными устройствами предусмотрен ряд интегральных адаптерных схем. Контроллеры периферийных устройств тоже могут быть подключены к шине памяти, посредством которой их обращение ко всему пространству памяти для осуществления пересылок в режиме прямого доступа к памяти (ПДП).

Дополнительная память может иметь различную конфигурацию, причем возможно включение в ее состав одновременно и быстродействующих, и медленных устройств. Предусмотрены сигналы подключения динамических ЗУПВ. Скорость пересылки данных по шине внешней памяти соответствует 25 Мбайт/сек.

К числу вспомогательных ИС, входящих в транспьютерное семейство, относится адаптер последовательного канала СО11, реализующий функции байтового ввода-вывода. Адаптер последовательного канала СО12 обеспечивает сопряжение последовательного канала с 8-битовой двунаправленной системной шиной. ИС С004 представляет собой групповой переключатель для соединения 32-х входных линий последовательной передачи данных с 32-выходными линиями. Управление этим соединением осуществляет 33-линия последовательной передачи. Такие ИС могут быть использованы для реконфигурации транспьютерных сетей. Их можно соединять каскадно для коммутации сетей большой размерности.

6.2.2. Базовая архитектура

Существует ряд факторов обеспечивающих высокую производительность транспьютеров. Один из них обусловлен тем, что ЦПУ реализовано по RISC – архитектуре. Увеличение производительности таких процессоров происходит за счет простоты и регулярности структур команд, позволяющих выполнять комбинацию из несложных команд быстрее, чем одну эквивалентную этой комбинации сложную команду. За счет организации команд RISC – процессоры работают быстрее, так как они выполняются за один машинный цикл, тогда как в CISC – процессорах на выполнение команды затрачивается несколько машинных циклов. Большинство операций в RISC – процессорах имеет характер регистр-регистр, а обращение к основной памяти происходит только для выполнения простых операций загрузки в регистры и занесения в память, вследствие чего и повышается скорость простых операций. Сложные команды преобразуются в последовательность простых, которые выполняются быстрее, чем при использовании больших наборов команд. Уменьшение количества команд приводит к уменьшению количества вентилей и объема микропрограммного ПЗУ (64 Кбит), необходимых для декодирования и реализации сложных команд. Это, в свою очередь, позволяет существенно снизить размеры микропроцессора с RISC – архитектурой, а также его стоимость. Процессор, память и система связи транспьютера Т414 занимает 25% всей площади кремния, а остальная площадь используется для отвода тепла, таймеров и внешних соединений. Транспьютер Т800, содержащий процессор с плавающей точкой (ППТ) занимает на 25% больше площади, чем транспьютер Т414.

Наличие скоростной оперативной памяти на одном кристалле транспьютера увеличило пропускную способность и обеспечило быстрое выполнение команд. С другой стороны, за один машинный цикл из памяти извлекается одно слово, содержащее несколько инструкций, и помещается в буфер предвыборки команд, где имеется дополнительный регистр, предназначенный для записи коротких инструкций. Так как этот буфер небольшой емкости, задержки при выполнении инструкций переходов, приводящих к тому, что содержимое буфера становится ненужным, являются незначительными. Исследование системы команд транспьютера показали, что 70% выполняемых инструкций занимает один байт и затрачивается только один цикл процессора. Таким образом, короткие инструкции также повышают эффективность предварительной выборки команд, что в свою очередь повышает производительность транспьютера.

Центральное процессорное устройство транспьютера содержит шесть регистров, которые используются при выполнении последовательного процесса. Эти регистры показаны на рис. 6.5.

Рис. 6.5. Регистры транспьютера

Указатель на рабочее поле (Регистр W) указывает на область памяти, где хранятся локальные переменные и адреса каналов процесса. Счетчик команд определяет команду, которая должна выполняться вслед за текущей командой. Регистр операнда используется для формирования операндов команды.

Регистры A, B, C образуют вычислительный стек и является источником и приемником для большинства арифметических и логических операций. При записи в стек, содержимое регистра B записывается в регистр C, содержимое регистра A – в регистр B, а затем новые значения записывается в регистр A. При чтении из регистра A содержимое регистра B записывается в A, а содержимое регистра C – в регистр B.

Для вычисления выражения используется вычислительный стек, к которому команды обращаются неявным образом. Например, команда add (сложение со знаком и с переполнением) складывает содержимое регистра A с содержимым регистра B и помещает результат в регистр A. При выполнении операции сдвига операнд, находящийся в регистре B, сдвигается на количество позиций, задаваемых в регистре A. Использование стека исключает необходимость переопределять положение операндов команд. Таким образом, три регистра обеспечивают эффективный баланс между компактностью кода и сложностью реализации. Нет аппаратного механизма, определяющего загрузку в стек более чем трех элементов. Такие ошибки легко определяются во время компиляции.

Система команд

В систему команд транспьютера входит относительно небольшое число команд одинакового формата, обеспечивающих компактное представление операций и наиболее часто встречающихся в программах. Набор команд не зависит от разрядности процессора, что позволяет один и тот же микрокод использовать для транспьютеров с различной разрядностью. Каждая базовая команда занимает один байт, который разделен на два 4-х битовых поля (рис. 6.6).

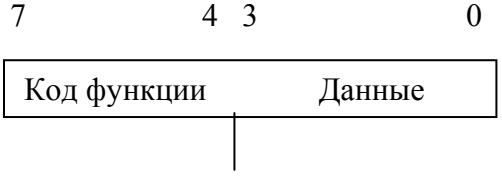


Рис. 6.6. Формат команды

Четыре старших бита этого байта являются кодом функции, а младшие четыре бита содержат операнд. Посредством четырех бит в команде транспьютера задаются 16 функций, значения данных для каждой из которых может достигать 16. В число 16 прямых базовых команд входят следующие инструкции:

ldc – загрузка константы

ldl – загрузка локальная

ldnl – загрузка нелокальная

ldlp – загрузка указателя локальная

eqc – сравнение с константой

j – переход

call – вызов

Adc – прибавление константы

stl – запись в память локальная

stnl – запись в память нелокальная

ldnlp – загрузка указателя нелокальная

ajw – установка рабочего пространства

cj – условный переход

Наиболее широко используемыми операциями являются загрузка в регистры и запись в память значений переменных и констант. Команда ldc осуществляет загрузку значений от 0 до 15 в стек. При выполнении команд ldl, stl, ldlp идентификация ячеек памяти производится относительно указателя рабочего пространства, а для команд ldnl, stnl, ldnlp обращение к памяти осуществляется относительно регистра A.

Две базовые (префиксные) команды: префиксация (pfix) и отрицательная префиксация (nfix) – позволяют увеличить длину операнда любой команды. При выполнении команды pfix четыре содержащихся в ней бита данных загружаются в регистр операнда, после чего содержимое регистра сдвигается на четыре разряда влево. Команда nfix выполняется аналогичным образом, но перед сдвигом формируется дополнительный код. При последовательном выполнении нескольких операций префиксации длина операндов может быть доведена до 32 разрядов. С помощью одной операции префиксации можно сформировать операнд в диапазоне от -256 до +256.

Использование префиксации имеет и другие преимущества. Во-первых, эти операции декодируются и выполняются таким же образом, как и любая другая команда, что упрощает и ускоряет декодирование команд. Во-вторых, упрощается процесс компилирования с языка программирования, так как любая команда может работать с операндом любой размерности единым образом. В-третьих, эти команды позволяют формировать операнды, независящие от разрядности процессора.

Наконец 16 базовая команда, имеющая код функции Operate (opr), интерпретирует свой операнд как код операции над значениями, находящимися в вычислительном стеке. Это позволяет в однобайтовой команде закодировать до 16 таких команд. С помощью операций префиксации можно также расширить поле операнда команды Operate, что позволяет

закодировать неограниченное число операций. Однако команда с использованием одной префиксной операции встречаются редко и в основном используются в транспьютере Т800.

Транспьютер Т800 имеет дополнительные команды для работы процессора с плавающей точкой, а также имеет новые команды для работы с цветной графикой, распознавания образов и использование кодов, корректирующих ошибки. Это стало возможным благодаря тому, что в транспьютере применяется расширяемое кодирование команд.

Программная обработка

Аппаратные средства транспьютера прямым образом ориентированны на реализацию параллельной обработки и организации информационных пересылок. Планировщик приоритетов является одним из основных устройств обеспечивающей эффективную поддержку модели параллельности и связи, принятых в языке Оккам. Этот микропрограммный планировщик обеспечивает параллельное выполнение любого количества процессов в режиме разделения времени процессора, что позволяет обходится без ядра операционной системы. Так как процессор транспьютера не имеет средств для динамического выделения памяти, то распределение памяти для параллельных процессов осуществляет компилятор языка Оккам. Время переключения процессов составляет менее 1 мкс, а обмен информацией между процессами осуществляется посредством блочных передач ввода-вывода.

Транспьютер обеспечивает реализацию двухуровневый системы приоритетов. Параллельный процесс может находиться в следующих состояниях.

Активное: - выполняется;
- находится в очереди в ожидании выполнения;

Пассивное: - готов к вводу;
- готов к выводу;
- ожидает, пока не наступит указанное время.

Процессам, находящимся в активном состоянии, присваиваются высокий приоритет, а процессам, находящимся в пассивном состоянии – низкий.

Активные процессы, ожидающие выполнения, содержатся в связанной очереди рабочей памяти, которая реализована с помощью двух регистров, один из которых указывает на первый процесс в очереди, а другой – на последний процесс.

На рис. 6.7 показаны регистры и рабочая память, необходимые для операций с параллельными процессами.

Активный процесс выполняется до тех пор, пока не перейдет в режим ввода, вывода или таймера.

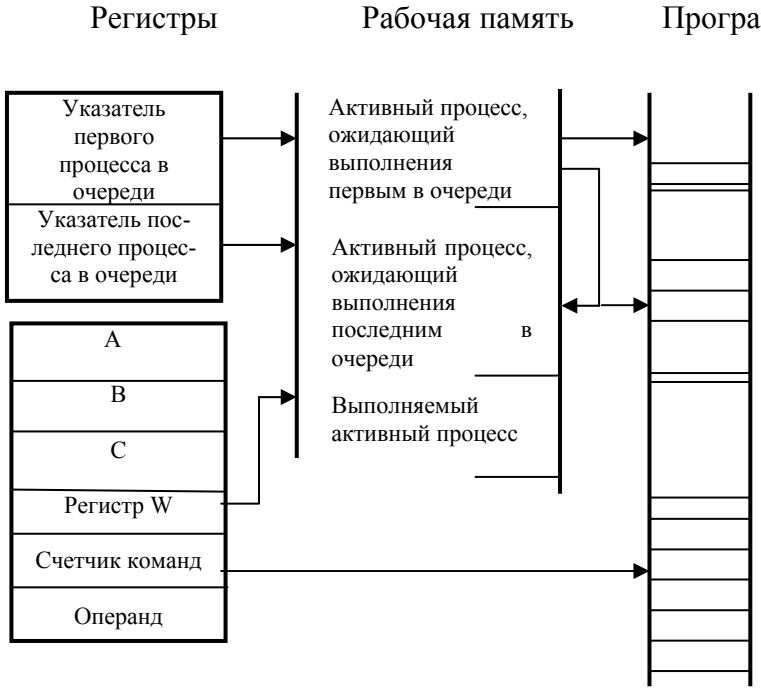


Рис. 6.7. Связный список процессов

При этом ему присваивается низкий приоритет, указатель запоминается в его рабочей памяти и из очереди для выполнения берется следующий процесс.

Пассивные процессы не занимают процессное время, так как исключены из списка очереди, и ожидают выполнения до тех пор, пока в активном состоянии не останется ни одного процесса высокого приоритета, или не выйдут из режима ввода, вывода, таймера.

Выполнение процесса низкого уровня разделяется с определенной периодичностью на интервалы времени. Продолжительность периода квантования составляет 4096 периодов входного тактового сигнала, имеющих частоту 5 мГц (около 800 мкс). Это необходимо для равномерного распределения процессорного времени между задачами, требующих больших временных затрат.

В транспьютере имеется таймер, работающий на частоте 1 мГц. Значение процессорного таймера можно прочитать с помощью команды read timer, а команда timer input с указанием времени приостанавливает выполнение процесса до тех пор, пока не наступит указанное время. Если указанное время наступило до выполнения этих команд, процесс продолжается выполняться. Если указанное время не наступило, процесс исключается из списка очереди. При наступлении указанного времени процесс снова включается в список очереди.

Как указывалось ранее, распределение процессов в рабочей памяти осуществляется компилятором. В процессоре реализованы специальные операции, позволяющие создавать новые процессы и исключить старые процессы. Операция start process создает новый процесс и добавляет новую рабочую область в конец стека очереди, что позволяет новому процессу выполняться вместе с уже существующими и выполняющимися процессами. Параллельная конструкция при корректном завершении выполняет операцию end process.

В процессоре существует специальный счетчик, определяющий число компонентов параллельной конструкции. При выполнении операции start process содержимое этого счетчика увеличивается, а при операции end process -уменьшается на единицу, и этот компонент выводится из списка очереди.

Оккам – каналы обеспечивают связи между процессами. Связи (внутренние каналы) между процессами, находящимися на одном транспьютере, реализуются посредством

одиночных слов памяти, а связи (каналы) между процессами, выполняемыми на разных транспьютерах, строятся на базе последовательных каналов связи. Адрес канала указывает, является ли канал внутренним или внешним.

При этом написание и компиляция процесса не налагает ограничений на соединение его каналов.

Подготовка процесса к вводу или выводу заключается в загрузке в вычислительный стек:

а) идентификатора канала (регистр B);

б) количества передаваемых байтов (регистр A);

в) указателя сообщений (регистр C).

При выполнении операций input message или output message передача сообщения по внутреннему каналу осуществляется путем занесения идентификатора первого процесса, оказавшегося готовым, в слово канала (ячейку памяти) и указателя этого процесса в рабочую память. При этом адрес передаваемого сообщения и количество байтов в сообщении, также заносится в рабочую память этого процесса. Процесс выводится из списка очереди и переходит в режим ожидания, а процессор выполняет очередной процесс из списка очереди. Когда оказывается готовым второй процесс, использующий этот канал, происходит перепись сообщения, а ожидающий процесс включается в конец списка очереди активных процессов и производится начальная установка канала в пустое состояние (empty).

Когда сообщение пересылается по внешнему каналу, процессор возлагает эту передачу на автономный интерфейс канала и исключает процесс из списка очереди. Если интерфейс канала реализует передачу сообщения посредством прямого доступа к памяти, процессор вновь вводит ожидающий процесс в список очереди. При внешней пересылке сообщений процессор параллельно может выполнять другие процессы.

Интерфейс канала включает три регистра, которые содержат:

а) указатель рабочей памяти процесса;

б) указатель сообщения;

в) количество байтов, подлежащих пересылке.

При вводе или выводе сообщения производится инициализация этих трех регистров и занесение указателя команды в рабочую память процесса. После того как оба процесса инициализируют свои канальные интерфейсы, происходит перепись сообщения, и оба интерфейса ставят процесс в конец списка очереди активных процессов.

6.2.3. Оккам – язык параллельной обработки

Оккам представляет собой новый язык программирования, разработанный для реализации параллельной обработки, в ходе которой множество частей системы функционируют по отдельности и взаимодействуют друг с другом. Оккам позволяет использовать единую нотацию для проектирования и программирования всей системы, начиная от описания сети и кончая операциями ввода-вывода низкого уровня.

Оккам основан на модели взаимодействующих последовательных процессов, позволяющих специфицировать параллельность алгоритма и его внутренние взаимосвязи. Программа на языке Оккам представляется как набор параллельных процессов, взаимодействующих друг с другом через каналы. Канал используется для передачи данных между двумя процессами, один из которых вводит из канала, другой – выводит в канал. Процессы не могут взаимодействовать посредством общих данных. Единственным способом взаимодействия процессов является посылка сообщений по каналам. Внутри процесса переменные и структуры данных определяют состояние процесса. Для программирования процесса можно использовать как язык Оккам, так и любые традиционные языки программирования, например, СИ, Паскаль.

Модель параллельности языка Оккам имеет распределенную природу. Если программа выполняется на нескольких транспьютерах, процесс, выполняющийся на одном транспьютере, может воздействовать на поведение процесса, выполняющегося на другом транспьютере, только посылкой ему сообщений. С другой стороны, если два процесса выполняются в одном транспьютере, то они взаимодействуют с помощью общих переменных. Однако, такое взаимодействие не допустимо в языке Оккам. Единственной формой взаимодействия двух параллельных процессов является посылка сообщений. Таким образом, одна и та же модель параллелизма используется как в одном процессоре, так и между процессорами. Такой подход исключает класс ошибок программирования, связанный с общими переменными, и облегчается перераспределение программы между процессорами.

Процесс в языке Оккам начинается, выполняет обработку и завершается. При выполнении обработки могут осуществляться следующие действия: присвоение (изменение значений переменной), ввод (получение значения из канала) и вывод (засылка значения в канал).

Ввод и вывод синхронизированы: когда процессы, как ввода, так и вывода в одном и том же канале готовы к коммутации, выходное значение переписывается из процесса вывода в процесс ввода без явного использования буфера, после чего оба процесса продолжают независимо выполняться. Каждый канал обеспечивает обмен информацией лишь в одном направлении. Для двухсторонней коммуникации необходимо наличие двух каналов.

Для объединения отдельных процессов в укрупненные процессы, в языке Оккам используются следующие конструкторы: последовательный, параллельный и альтернативный. Последовательный конструктор (SEQ) задает поочередное выполнение своих компонентов; он завершается с окончанием последней компоненты. Параллельный конструктор (PAR) определяет параллельное выполнение своих компонентов; его выполнение завершается лишь после окончания выполнения всех компонентов. Альтернативный конструктор (ALT) выбирает для выполнения один из компонентных процессов из списка возможных; он заканчивается завершением выборного процесса.

Возможно, также построение условных конструкторов типа IF и WHILE. Конструктор IF проверяет условие всех процессов и тот процесс, которому соответствующее условие оказывается истинным, поступает на выполнение. Если все условия ложны, то выполнение конструктора IF завершается. При использовании конструктора WHILE компонентный процесс повторяется до тех пор, пока результат оценки выполнения, реализуемого конструктором, не будет ложным.

Конструкторам могут предшествовать связанные с ними описания. Посредством описаний вводятся идентификаторы переменных, каналов и констант, используемых в процессе. Эти идентификаторы являются локальными и действуют в пределах конструктора, которому предшествует описание.

С помощью конструкторов можно объединить любое число процессов, причем сочетания конструкторов и элементарных процессов могут иметь любое количество уровней. Поскольку процесс сам по себе может состоять из других параллельных процессов, в механизме работы с процессами предусмотрены средства иерархической декомпозиции задач. Оккам может применяться для иерархической декомпозиции задач с использованием только параллельных конструкторов и каналов до тех пор, пока отдельные процессы не станут максимально простыми. Каждый из полученных в результате этой процедуры простых процессов может в последствии выполняться отдельным процессором. Синхронизация в этом случае будет осуществляться только при обмене информации.

6.3. Pentium процессоры

Пятое поколение микропроцессоров ознаменовалось появлением в 1993 году первых процессоров Pentium, работающих на частотах 60 и 66 МГц. Они были выполнены по

технологии 0,8 мкм с 3,1 млн. транзисторами на кристалле, имеющие 32-разрядную внутреннюю архитектуру с 64 разрядной шиной данных. В Pentium впервые был применен двойной конвейер, позволяющий параллельно исполнять команды, до двух за один такт. Эта архитектура получила название суперскалярной и этим она принципиально отличается от процессора 80486.

Процессоры Pentium с частотой 75, 90 и 100 МГц, появившиеся в 1994 году, представляли второе поколение процессоров Pentium. Они выполнены по технологии 0,6 мкм при том же числе транзисторов на кристалле, это позволило снизить потребляемую мощность. От первого поколения отличались внутренним умножением частоты, поддержкой мультипроцессорных конфигураций и другим типом корпуса. Эти процессоры стали популярными в персональных компьютерах. В 1995 году были выпущены процессоры на 120 и 133 МГц, где второй процессор был реализован по технологии 0,35 мкм, а в 1996 появились процессоры на 150, 166 и 200 МГц.

Pentium Pro является следующим процессором после Pentium и представляет шестое поколение микропроцессоров, получившие общее название P6. Дальнейшее распараллеливание вычислительного процесса позволило увеличить число параллельно исполняемых инструкций. В его корпусе поместили вторичный кэш, работающий на частоте ядра. Процессор состоит из 5,5 млн. транзисторов ядра и 15,5 млн. транзисторов вторичного кэша объемом 256 Кбайт. Первый процессор с частотой 150 МГц появился в начале 1995 года (технология 0,6 мкм), а в конце года были достигнуты частоты 166, 180 и 200 МГц (технология 0,35 мкм), а кэш увеличен до 512 Кбайт.

В начале 1997 года Intel выпустила процессоры Pentium MMX (MultiMedia eXtentions). Эта линия процессоров базировалась на ядре Pentium и обладала рядом преимуществ. Были введены 57 новых инструкций, предназначенных для оптимизации обработки мультимедийных данных. Технология MMX предполагает параллельную обработку группы операндов одной инструкцией (одиночный поток команд, множественный поток данных). Этот подход позволил ускорить выполнение мультимедийных приложений, в частности операции с изображениями и обработку сигналов. Кроме MMX эти процессоры имеют удвоенный объем первичного кэша, добавлена дополнительная ступень у обоих конвейеров. Процессор работает на частотах 166, 200 и 233 МГц, выполнен по технологии 0,35 мкм и имеет 4,5 млн. транзисторов.

Дальнейшее повышение производительности процессоров Pentium было достигнуто за счет совершенствования его микроархитектуры. В 1997 году появился процессор Pentium II, сочетающий в себе комбинацию различных архитектурных методов. Он представляет собой слегка урезанный вариант ядра Pentium Pro с более высокой внутренней тактовой частотой, в которое ввели блок MMX. Вдвое увеличен размер кэша памяти первого уровня (16 Кбайт данных, 16 Кбайт команд). Кэш память второго уровня выполнена в виде отдельной микросхемы, работающей на половине внутренней частоты и располагающейся вместе с кристаллом на небольшой плате, которая заключена в корпус, называемый картриджем. Первые процессоры были реализованы по 0,35 мкм технологии (7,5 млн. транзисторов) и имели тактовые частоты 233, 266 и 300 МГц. В 1998 году Intel освоила технологию 0,25 мкм, что позволило поднять тактовую частоту процессора. Процессор на 333 МГц имеет частоту шины 66 МГц, а процессоры на 350 и 400 МГц уже имеют частоту системной шины 100 МГц. Семейство Pentium II динамично развивается и имеет широкий спектр вариаций от относительно дешевых до очень дорогих моделей. Для систем начального уровня появилось семейство процессоров Celeron, младшие модели которого вообще не имеют кэша второго уровня. Новые модели имеют вторичный кэш на 128 Кбайт, который теперь размещен прямо на кристалле ядра. Достигнута частота 333 МГц, при этом кэш работает на частоте ядра. В перспективе частота ядра будет поднята до 366, 400 МГц, а частота шины – до 100 МГц. Для мощных систем разработан процессор Pentium II Xeon с частотой ядра 400 и 450 МГц (частота шины 100 МГц), у которого вторичный кэш объемом до 2 Мбайт работает на частоте ядра. Предполагается развитие линии Xeon, где процессоры

будут иметь расширенный набор инструкций MMX2 и частоту ядра, начиная с 500 МГц и выше.

Процессоры Pentium III (1999 г.) являются дальнейшим развитием Pentium II. Их главным отличием является расширение набора SIMD-инструкций – SSE (Streaming SIMD Extensions), основанное на новом блоке 128- разрядных регистров XMM. Этот блок позволяет одной инструкцией выполнять операции сразу над четырьмя комплектами 32-рарядных операндов в формате с плавающей точкой (одинарная точность). Инструкции с регистрами XMM могут работать и в скалярном режиме (с одним комплектом операндов). По возможностям мультипроцессрных конфигураций эти процессоры аналогичны своим предшественникам Pentium II и Pentium II Xeon. Возможность избыточных конфигураций (FRC) есть только у Pentium III Xeon. Частота ядра начинается с 500 МГц, частота системной шины – 100 и 133 МГц. Процессоры Pentium III (Coppermine) изготовлены по технологии 0,18 мкм содержит 28 миллионов транзисторов, площадь кристалла – 106 мм^2. выпускаются в картридже SECC-2 для слота 1(SC-242) и в корпусе FC-PGA для сокета-370. Первичный кэш составляет 32 Кбайт (16+16), на кристалле ядра расположен улучшенный вторичный кэш размером 256 Кбайт с ECC- контролем, который работает на частоте ядра. Пропускная способность этого кэша соответствует 17 Мбайт/сек.

В 2001 году появился первый микропроцессор с архитектурой IA-64, получивший известность под названием Itanium. Архитектура этого процессора базируется на принципах EPIC (явный параллелизм на уровне команд) и тесно связана с идеями VLIW (сверхбольшое командное слово). Intel предлагает Itanium с тактовыми частотами 733 и 800 МГц и кэшем третьего уровня емкостью 2 или 4 Мбайт, использующая технологию SRAM. При производстве Itanium используется 0.18-микронная КМОП-технология. Конструктивно оформлен в виде картриджа, который содержит процессор и кэш L3. Степень интеграции составляет 320 млн. транзисторов, но процессор содержит только 25 млн. транзисторов. Следующей более эффективной реализацией IA-64 предполагается микропроцессор McKinley. В нем будет использоваться новая микроархитектура по технологии 0.18 или 0.15 микрон.

6.3.1 Базовая архитектура

В архитектуре процессоров Pentium впервые использованы различные способы конвейеризации и распараллеливания вычислительного процесса. Это было достигнуто за счет введения дополнительных аппаратных средств и новых алгоритмов обработки команд. Процессоры Pentium имеют ряд принципиальных отличий от предыдущих поколений:

1) суперскалярная архитектура: процессор имеет два параллельно работающих конвейера, позволяющих выполнить за один такт две команды. Это преимущество будет достигаться только при выполнении хорошо скомпилированных программ;

2) возможность динамического предсказания направления переходов в программе позволяет обеспечить максимальную загрузку конвейеров;

3) новые алгоритмы обработки команд сокращают число тактов процессора, необходимых для их выполнения;

4) блок с плавающей точкой реализован в виде конвейера, что почти в 10 раз позволило увеличить его производительность по сравнению с 486-процессором;

5) конвейерная обработка циклов шины данных;

6) внутренний разделенный кэш данных и команд, который работает с отложенной записью (до освобождения внешней шины) настраивается на режим сквозной или обратной записи;

7) наличие 64-разрядной внешней шины данных, которая может перестраиваться на работу с 8- , 16- и 32-разрядными данными;

8) дополнительный контроль за целостностью данных: проверка на четность адресов и внутренних массивов данных. Проверка на четность обеспечивает обнаружение ошибок у 53% компонент в кристалле Pentium, не уменьшая скорость выполнения команд;

9) контроль с помощью функциональной избыточности. Осуществляется дублирование вычислений вторым процессором – «контроллером», который выдает сигнал ошибки при расхождении вычислений на выходах процессоров. Функциональная избыточность обеспечивает практически 100 % гарантию обнаружения ошибок.

На рис. 6.8 изображено прохождение команд через конвейер 486 процессора.

Рис. 6.8.Поток команд через конвейер 486

Целочисленные команды процессора Pentium, также, как и у 486-процессора, проходит пять стадий конвейера:

1) PF –предвыборка команд (Prefetch);
2) D1 - декодирование команды (Instruction Decode);
3) D2 - генерация адреса (Address Generate);
4) EX – выполнение (Execute);
5) WB - сохранение результата (Write Back).

Поток команд в процессоре Pentium изображен на рис 6.9.

Рис. 6.9. Поток команд через конвейер Pentium

Целочисленный конвейер процессора Pentium представлен на рис. 6.10.

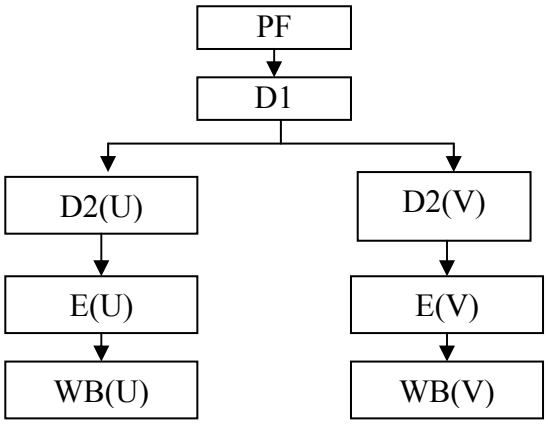


Рис. 6.10. Конвейер процессора Pentium

Основной конвейер U может выполнять все команды над целыми числами и команды с плавающей точкой, а конвейер V простые целые команды и несколько команд с плавающей точкой. Каждый конвейер может выполнять наиболее часто встречающиеся команды за один такт, а вместе оба конвейера могут исполнять до двух команд за такт (рис. 6.9) или одну операцию с плавающей точкой (в некоторых случаях – до двух). При параллельном исполнении команд V-конвейер всегда выполняет команду кода, следующую за командой, выполняемой U-конвейером.

С введением второго конвейера возникает ряд проблем. Одна из них проблема «зависимости данных», когда выполнение очередной команды зависит от результата предыдущей. В этом случае выполнение такой команды останавливается до тех пор, пока не будет получен требуемый результат, что снижает общую производительность.

Различают два типа зависимостей:

- запись после чтения (зависимость типа WAR). Первая команда читает данные из регистра или памяти, вторая пишет в этот регистр или ячейку памяти. В этом случае, прежде чем производить запись, необходимо дождаться, пока будет выполнено чтение;
- чтение после записи (зависимость типа RAW). Обратная ситуация, когда до операции чтения необходимо дождаться выполнения записи.

Существует несколько методов уменьшения влияния зависимости данных на производительность конвейеров. Это переименование регистров (register renaming), одновременное использование одних данных в обоих конвейерах (data forwarding).

Переименование регистров позволяет обойти архитектурное ограничение (предусмотрено всего 16 регистров общего назначения) на возможность параллельного исполнения команд. Процессоры с переименованием регистров фактически имеют более 16 РОН, и при записи промежуточных результатов устанавливается соответствие логических имен и физических регистров. Такой подход позволяет снять зависимость первого типа. Например:

MOV AX,BX ; запись содержимого регистра BX в AX;

MOV BX,CX ; запись содержимого регистра CX в BX.

В данном примере запись в регистр BX должна быть выполнена после завершения первой команды. Переименование регистров позволяет для регистра BX использовать два внутренних регистра –Reg0 и Reg1. После такого дублирования можно одновременно и читать из BX и записывать в него, так как физически работают два разных регистра. Но этот подход не снимает зависимости второго типа. Полное снятие зависимости второго типа невозможно, можно только уменьшить время ожидания работы второго конвейера. Метод, позволяющий уменьшить влияние зависимости второго типа, состоит в одновременном использовании данных. При этом выполняются все возможные действия, и декодированная

команда с одним операндом помещается в исполнительное устройство второго конвейера, где дожидается готовности второго операнда, выходящего с первого конвейера. В этом случае второй операнд сразу передается на второй конвейер, что позволяет избежать операции чтения из регистра или ячейки памяти, и вынужденное простаивание второго конвейера сводится к минимуму.

При параллельном выполнении двух команд на двух конвейерах возникает ситуация, где одна команда выполняется быстрее другой. Эта команда заканчивается раньше и нарушается очередность выполнения команд. Исполнение с изменением последовательности команд, свойственное RISC-архитектуре, теперь реализовано и в процессорах Pentium. При этом изменяется порядок внутренних манипуляций данных, а внешние (шинные) операции ввода-вывода и записи в память выполняются в порядке предписанным программным кодом. Это достигается за счет введения специального буфера; куда результаты выполнения операций заносятся в произвольном порядке, а данные выходят в порядке очередности.

Другой проблемой, связанной с введением второго конвейера, является проблема сдваивания команд. Параллельное выполнение двух команд называется сдваиванием. При этом одновременное исполнение команд на U и V конвейерах определяется специальными правилами сдваивания, которые удовлетворяют следующим условиям:

- обе команды должны быть простыми;
- между ними не должно быть регистровых зависимостей на WAR и RAW;
- ни одна из команд не может содержать одновременно смещение и непосредственный операнд;
- команды с префиксами могут появляться только в U – конвейере (кроме OF в командах JCC).

Простыми являются следующие целочисленные команды:

1	MOV reg, reg/mem/imm	Команды
2	MOV mem, reg/imm	пересылки
3	ALU reg, reg/mem/imm	Арифметические и
4	ALU mem, reg/imm	логические операции
5	INC reg/mem	Увеличение и
6	DEC reg/mem	уменьшение на 1
7	PUSH reg/mem	Операции
8	POP reg	над стеком
9	LEA reg, mem	Загрузка
10	JMP/CALL/JCC near	команды передачи управления
11	NOP	Пустая команда

Простые команды выполняются аппаратным способом и исполняются за один такт, за исключением команд ALU mem, reg и ALU reg, mem, которые требуют для выполнения два и три такта соответственно, и их выполнение обеспечивается специальными компонентами процессора.

Условные и безусловные переходы могут выступать только как вторая команда в паре. SHIFT на 1, ROT на 1, SHIFT imm могут быть лишь первой командой в паре.

Регистровые зависимости WAR, RAW, которые запрещают сдваивание, включают неявные зависимости через регистры и флаги. Например, ALU команда в U – конвейере (устанавливает флаги) не может быть в паре с использующимися флагами команд в V – конвейере. Два исключения из этого правила – это последовательность «сравнение – переход» и пары из команд PUSH и POP. Во втором случае для исключения неявной зависимости от указателя стека добавлены специальные аппаратные компоненты, чтобы эти частые операции могли параллельно выполняться. Все команды подчиняющиеся правилам сдваивания выполняются параллельно и независимо, кроме команд имеющих зависимости

по данным и пары команд типа «прочитать – изменить – записать», т.е. ALU команд с операндом в памяти. Если две такие команды сдвоены, то к необходимым для их выполнения трем тактам добавится задержка в два такта.

Чаще одним фактором, отрицательно влияющим на производительность конвейера, является зависимость по управлению. Она возникает в программах, где встречаются условный или безусловный переходы. После декодирования команды перехода и получения адреса процессор начинает считывать данные с нового адреса. До получения этого адреса конвейер простаивает. Для снижения этого фактора в процессорах используются ряд архитектурных методов.

Предсказание переходов (branch prediction) позволяет продолжать выборку и декодирование потока команд после выборки команды перехода (условного), не дожидаясь проверки самого условия. В процессорах прежних поколений команда перехода приостанавливала конвейер (выборку команд) до исполнения собственно перехода при этом терялась производительность.

Существуют два подхода предсказания переходов. Статистический метод предсказания работает по схеме, заложенной в процессор, направляя поток выборки и декодирования по одной наиболее вероятной ветви. Динамическое предсказание опирается на предысторию вычислительного процесса: для каждого конкретного случая перехода накапливается статистика поведения, и переход предсказывается, основываясь именно на ней.

Опережающее исполнение (speculative execution) позволяет до поверки условия перехода не только выбирать и декодировать команды, но и выполнять их. В процессоре Pentium реализован только первый из этих методов – метод предсказания переходов. Для реализации этого метода в процессор Pentium введено устройство, называемое буфером адресов переходов BTB (Branch Target Buffer), и предназначенное для предвидения возможных «нелинейных» переходов. В случае неудачного прогноза выполнение прекращается, конвейер очищается и начинается исполнение с правильного адреса. Эффективность этого блока оценивается вероятностью правильности прогноза. В современных процессорах она лежит в пределах 80-97 %. Таким образом, введение новых архитектурных методов по обработке команд позволило существенно повысить производительность Pentium процессоров.

6.3.2. Организация процессора Pentium

Структурная схема процессора Pentium показана на рис. 6.11. Он построен на основе двух конвейеров общего назначения для целочисленных операций и конвейерного FPU блока (Floating Point Unit). Оба конвейера функционально схожи, но второй V-конвейер по сравнению с главным U-конвейером имеют некоторые ограничения. Процессор выполняет до двух команд за один такт. Во время исполнения проверяются следующие две команды, и, если возможно, они запускаются в V- и U- конвейеры. Если две команды сразу запустить невозможно, то первая запускается в U-конвейер, а V-конвейер простаивает. При выполнении команд на обоих конвейерах их функциональное поведение в точности совпадает с поведением при последовательном выполнении.

На первой стадии (PF) осуществляется предвыборка команд из кэш или оперативной памяти. Поскольку кэш для команд и данных раздельный, то ликвидирован конфликт при обращениях к кэш-памяти между командами и данными. Если требуемой команды в кэш нет, происходит обращение к оперативной памяти.

В предвыборке участвуют два независимых 32-байтовых буфера предвыборки и буфер переходов. В каждый момент времени предвыборку осуществляет только один из буферов предвыборки. Предвыборка выполняется до тех пор, пока не будет выбрана команда перехода. В этот момент буфер переходов (BTB) предсказывает, будет, или не будет

осуществляться переход. Процессор Pentium реализует схему динамического предсказания переходов с

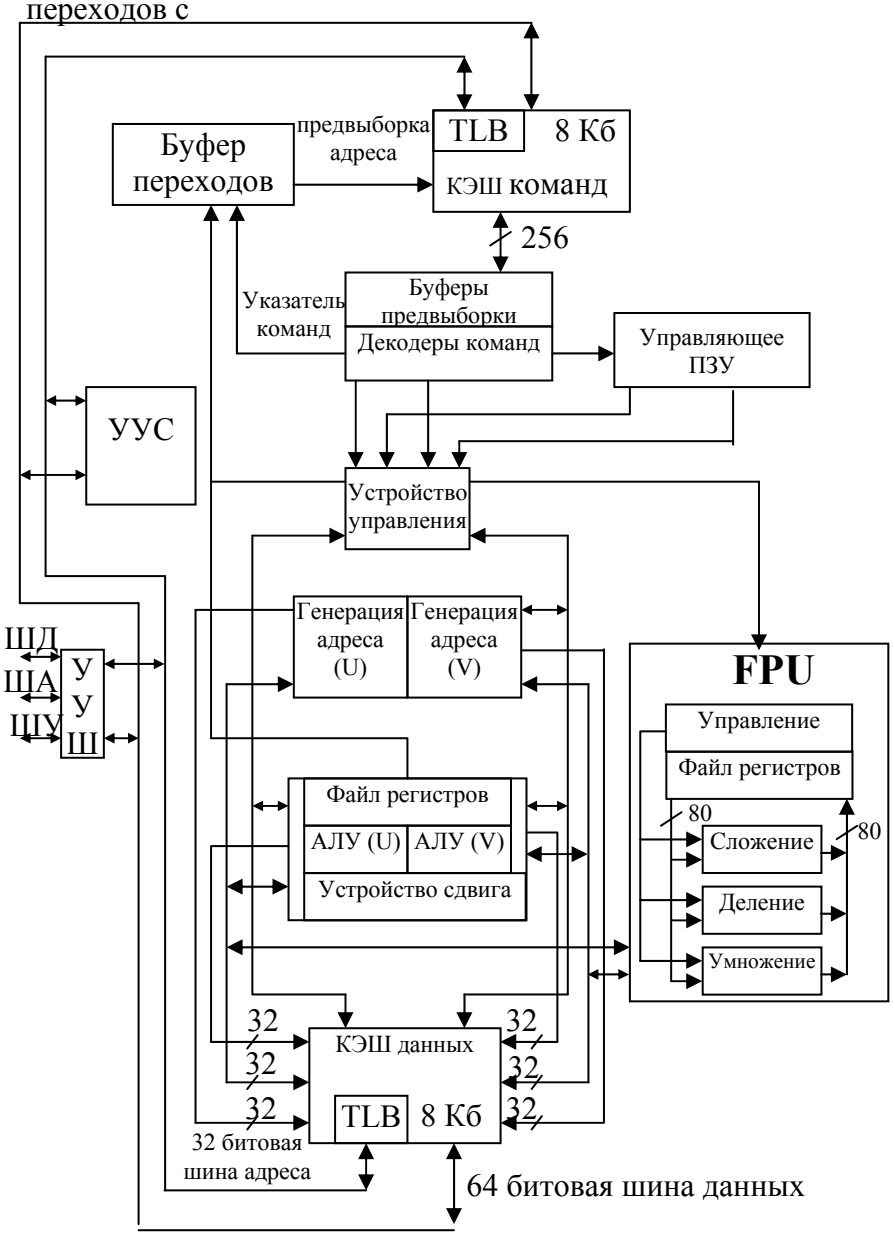


Рис. 6.11. Структурная схема процессора Pentium

буфером BTB на 256 вхождений. Если предсказано, что перехода не будет, то продолжается последовательная предвыборка. Если переход предсказан, то включается другой буфер предвыборки, который выбирает команды, начиная с той, на которую должен произойти переход. В обоих случаях команды ветвления выполняются без штрафных циклов. Если переход предсказан ошибочно, условный переход, выполняемый на U-конвейере, требует три штрафных цикла, а на V-конвейере – четыре. Ошибочно предсказанный вызов подпрограммы или безусловный переход требует три штрафных цикла на любом конвейере.

На второй стадии (D1) два параллельных дешифратора декодируют и отправляют на выполнения две последовательные команды. Дешифраторы определяют, одна или две команды будут выполняться в соответствии с правилами сдваивания. На третьей стадии (D2) определяются адреса операндов в памяти. На этой стадии команды, содержащие смещение и непосредственный операнд, а также базовый и индексный режимы адресации, выполняются за один такт.

На четвертой стадии (EX) Pentium осуществляет операции в АЛУ и обращения к КЭШ данным. Команды, требующие и того и другого, задерживаются на этой стадии более чем на один такт. На этой стадии все команды в U-конвейере и все команды, за исключением условных переходов, в V-конвейере проверяются на правильность предсказания переходов.

На последней стадии (WB) команды могут изменить состояние процессора и завершают выполнения. На этой стадии на правильность предсказания переходов проверяются условные переходы в V-конвейере. Каждый конвейер имеет свой буфер записи WB для повышения производительности при последовательных записях в память. Буферы имеют разрядность 64 бит, и могут оба заполняться за один такт, например, при одновременных кэш-промахах записи на обоих конвейерах. На внешнюю шину операции записи из этих буферов направляются в том порядке, в каком они генерируются процессорным ядром. Pentium процессор поддерживает строгий порядок записи .

Высокопроизводительный математический сопроцессор представляет собой 8 стадийный конвейер и может выполнять одну команду, с плавающей точкой за один такт.

Команды сопроцессора проходят по общему конвейеру до ступени EX, после чего выполняются в блоке FPU (рис.6.11). Как правило, команды с плавающей точкой не могут выполняться параллельно с целочисленными командами. Однако, существует несколько исключений, позволяющих сдваивать команды. Многие команды с плавающей точкой могут быть выполнены только над операндом, находящегося в вершине стека

Для обмена данными с вершиной стека предназначена команда FXCH. Это часто встречающаяся при вычислениях с плавающей точкой команда может выступать в качестве второй команды, если ей предшествует одна из длинных команд: FLD single/double, FLD ST[i], FADD, FSUB, FMUL, FDIV, FCOM, FUCOM, FTST, FABS, FCHS. Сдвоенная таким образом команда FXCH не требует времени на выполнение (0 тактов).

Другим исключением является то, что с этими длинными командами могут быть выполнены и простые команды в целочисленном конвейере. Другим новшеством при построении математического сопроцессора является использование новых алгоритмов обработки операций с плавающей точкой, что позволило увеличить скорость выполнения обычных операций (ADD, MUL, LOAD). Конвейерное исполнение и более совершенные алгоритмы позволили почти в 10 раз увеличить скорость обработки по сравнению с 486-ым процессором.

Встроенная подсистема кэширования включает два двухканальных наборно-ассоциативных кэша размером по 8 Кбайт: кэш данных и кэш программ. Длина строки кэша составляет 32 байт, ширина внешней шины данных – 8 байт. Оба кэша придерживаются одного из двух алгоритмов обновления данных: алгоритма связной или обратной записи. Задать алгоритм для конкретного блока памяти можно как программным, так и аппаратным способами. Замещение строк в кэш–памяти основывается на алгоритме LRU. Если в кэш–памяти есть недействительные строки, то новая строка будет помещена вместо одной из них. Если недействительных строк нет (кэш–память переполнена), то новая строка замещает строку, на которую дольше всего не было никаких ссылок. Память каждого кэша разбивается на восемь банков с чередованием по границам в 4 байта. Кэш данных доступен обоим конвейерам, и если им требуются данные из разных банков, запросы от обоих конвейеров могут обслуживаться одновременно.

Размерность внешней шины данных в Pentium увеличена вдвое и равна 64 байтам. Введено конвейерное выполнение циклов передачи данных, при этом два цикла шины могут одновременно находиться на выполнении. Максимальная скорость передачи данных по шине данных при тактовой частоте 66 МГц составляет:

$$8 \text{ байтов} \times 66 \text{ МГц} = 528 \text{ Кбайт/сек},$$

что больше чем в три раза превосходит максимальную скорость передачи 50 МГц 486 процессора (160 Кбайт/сек).

Таким образом, в основе построения процессора Pentium лежит суперскалярная архитектура. Эта архитектура базируется на технологии параллельного выполнения команд программы на двух конвейерах, определяемые специальными правилами сдваивания, предсказании будущих переходов, на развитой системе кэширования и конвейерного исполнения операций с плавающей точкой. Эти архитектурные методы позволили резко повысить производительность процессоров Pentium.

6.4. Процессоры P6

Возможности суперскалярной архитектуры процессора Pentium, с ее способностью к выполнению двух команд за один такт, можно было превзойти только за счет всестороннего улучшения микроархитектуры процессора. Принципиальные решения, позволившие улучшить микроархитектуру процессорного ядра, были введены в следующие поколения процессоров Pentium Pro, Pentium II и Pentium III. Микроархитектура этих процессоров логически близка друг к другу и, в этом смысле, их называют P6. При разработке микроархитектуры P6 использовалась тщательно продуманная и настроенная комбинация различных архитектурных методов. Этот новый подход устраняет жесткую зависимость между традиционными фазами выполнения команды: фазами выборки и выполнения. Он связан с использованием специального накопителя, называемого пулом команд, и с эффективными методами предсказания будущего поведения программы. При этом традиционная фаза выполнения команды распадается на диспетчирование/выполнение и завершение (откат). В результате команды могут выполняться в произвольном порядке, но завершают свое выполнение всегда в соответствии с их исходным порядком в программе. Ядро процессора P6 представляется как три независимых устройства, взаимодействующих через пул команд (рис. 6.12.).

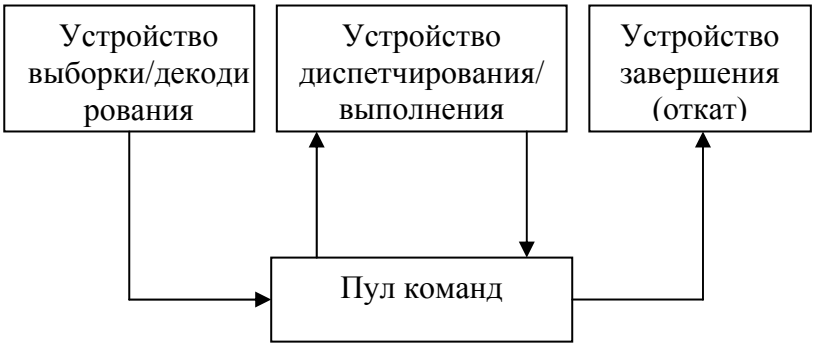


Рис. 6.12. Ядро процессора P6

С другой стороны, ядро процессора P6 реализовано в виде 12 ступенчатого конвейера. Увеличение числа стадий приводит к уменьшению выполняемой на каждой стадии работы и, как следствие, к уменьшению времени нахождения команды на каждой стадии на 33 % по сравнению с Pentium. Это означает, что использование при производстве P6 той же технологии, что и при производстве 100 МГц Pentium, приведет к получению P6 с тактовой частотой 133 МГц.

Основной проблемой процессоров Pentium и многих других является не полная загрузка их конвейеров. Особенно снижается их производительность при наличии между соседними командами помехи типа RAW. Рассмотрим в качестве примера следующий фрагмент программы, записанной на некотором условном языке:

```
r1 ← mem (r0)      * команда 1*
r2 ← r1 + r2       * команда 2*
r3 ← r3 + 1        * команда 3*
```

r4 ← r4 – r5 * команда 4*.

Первая команда этого фрагмента осуществляет загрузку из ячейки памяти в регистр r1. Если при выполнении этой команды содержимое ячейки не окажется в кэш-памяти, то данные из ячейки будут прочитаны через интерфейс шины. При традиционном подходе переход ко второй команде будет произведен, только после завершения выполнения первой команды. Во время цикла шины конвейер при этом будут простаивать.

В то время как скорость процессоров за последние 10 лет выросла в 10 раз, время доступа к основной памяти уменьшилось на 60 %. Это увеличивающееся отставание скорости работы с памятью по отношению к скорости процессора явилось причиной недозагрузки конвейеров.

Одним из возможных подходов к решению этой проблемы является разработка высокоскоростных компонентов, окружающих процессор. Однако массовый выпуск высокоскоростных специализированных микросхем окружения (особенно памяти) является слишком дорогим мероприятием. Другой подход заключается в увеличении объема кэша второго уровня, чтобы уменьшить процент кэш-промахов. Это решение эффективное, но также чрезвычайно дорогостоящее. Требование высокой производительности системы с использованием недорогой подсистемы памяти можно удовлетворить введением новых архитектурных решений.

Одно из этих решений называется внеочередным исполнением команд. Он заключается в том, что до завершения команды – тормоза произвести максимум полезной работы. В нашем примере процессор не может выполнить команду 2 до завершения команды 1. В то же время процессор может выполнить команды 3 и 4, не зависящие от результата выполнения команды 1. Результаты опережающего выполнения команд 3 и 4 не будут сразу записаны в регистры, поскольку изменение состояния вычислительной системы должно происходить в соответствии с логикой выполнения программы. Эти результаты также, как и все команды, сохраняются в пуле команд и извлекаются оттуда позднее.

Таким образом, процессор выполняет команды в соответствии с их готовностью к выполнению, вне зависимости от первоначального порядка в программе. В этом случае, с точки зрения реального порядка выполнения команд процессор P6 является машиной, управляемой потоком данных. В то же время изменение состояния вычислительной системы, например запись в регистры, производиться в строгом соответствии с истинным порядком команд в программе.

Для исключения зависимости типа WAR в процессоре P6 используется метод переименования регистров. При этом устройство диспетчирования/выполнения работает с дублями регистров, находящихся в пуле команд (одному регистру может соответствовать несколько дублей). Реальный набор регистров контролируется устройством завершения, результаты выполнения команд отражаются на состояние вычислительной системы только после того, как выполненная команда удаляется из пула команд в соответствии с истинным порядком команд в программе.

В блоке предсказания переходов реализован новый алгоритм предсказания переходов, называемый опережающим исполнением команд, обеспечивающий высокую точность предсказания переходов.

Таким образом, принятая в P6 технология динамического выполнения программы характеризуется как оптимальное выполнение программы, основанное на предсказании переходов, анализе графа потоков данных с целью наилучшего порядка исполнения команд (спекулятивное выполнение) и на внеочередном исполнении команд в выбранном оптимальном порядке.

6.4.1. Организация процессора P6

Отличительной особенностью процессоров P6 является то, что команды, проходящие

через конвейер в порядке поступления, разбивается на простейшие микрооперации, которые выполняются суперскалярным процессорным ядром в порядке, удобным процессору. Ядро процессора содержит пул команд, к которому подключаются исполнительные устройства целочисленных вычислений, обращение к памяти, предсказания переходов,

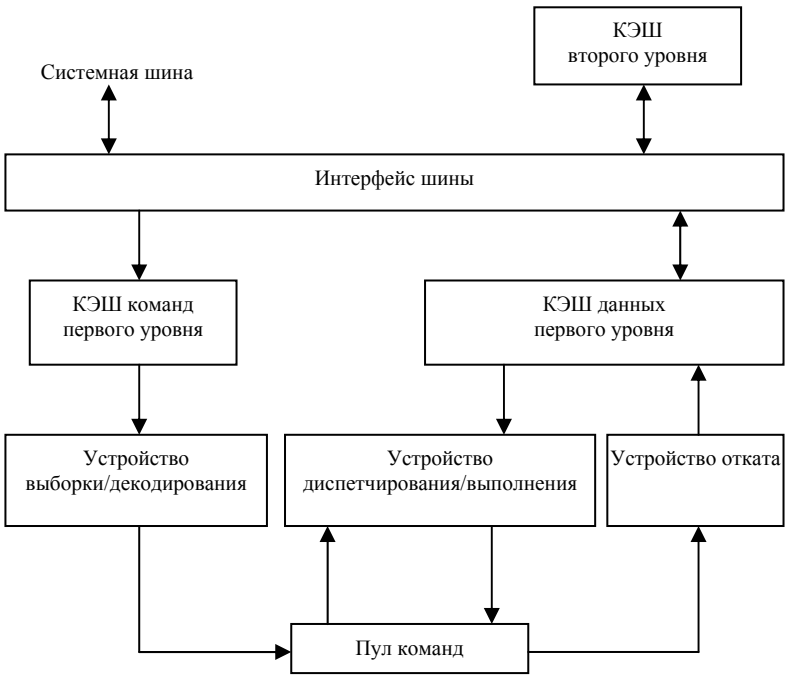


Рис. 6.13. Структурная организация P6

вычислений с плавающей точкой и завершения выполнения операций. Несколько исполнительных устройств могут объединяться в одном конвейере. На рис. 6.13 представлена организация процессора P6.

Под упорядоченным устройством понимается устройство, которое работает в соответствии с исходным порядком команд в программе, соответственно, под неупорядоченным - устройством, которое не обращает внимания на исходный порядок команд в программе.

Пул команд состоит из переупорядочивающего буфера чтения, способного выполнять два чтения регистров за такт (ReOder Buffer Read – ROB R) и переупорядочивающего буфера записи (ROB W).

Буфер переупорядочивания ROB R предназначен для хранения микрокоманд после стадии переименования регистров и реализован в виде массива контекстно адресуемой памяти. Буфер содержит сорок элементов размером 254 байт каждый, и может хранить микрокоманду, два связанных с ней операнда, результат и несколько бит состояния. В ROB R могут находиться микрокоманды как вещественного, так и целочисленного типа. Все микрокоманды в ROB R имеют определенный статус: «может быть выполнено», «ожидает данные», «выполнено», «переход» и т.д.

Переупорядочивающий буфер записи ROB W представляет собой файл регистров, предназначенный для хранения операндов и результатов выполнения микроопераций. Он поддерживает механизм переименования регистров. При отображении регистров происходит преобразование программных ссылок на архитектурные регистры в ссылки на большой набор физических регистров. В действительности же P6 содержит сорок физических регистров, реализованных в виде буфера ROB W.

Устройство выборки/декодирования является упорядоченным устройством, который воспринимает на входе поток команд из программы и декодирует их в последовательность микрокоманд соответствующих потоку данных в программе. Устройство

выборки/декодирования включает в себя два буфера переходов (Branch Target Buffer – BTB0,BTB1), блока выборки команд (Instruction Fetch Unit – IFU), который состоит из трех устройств IFU0, IFU1, IFU2, двух декодеров (Instruction Decode – ID0, ID1), блока переименования регистров и выделения ресурсов (Register Allocation – RAT) и буфера микрокоманд.

Устройство диспетчирования/выполнения является неупорядоченным устройством, который воспринимает поток данных и планирует выполнение микрокоманд с учетом зависимостей по данным и доступности ресурсов, а также временно сохраняет результаты внеочередного выполнения команды в пуле команд. Он содержит пятипортовую распределительную станцию (Reservation Station – RS) и функциональные исполнительные устройства (два целочисленных АЛУ0 и АЛУ1, блока выполнения операций с плавающей запятой, блока выполнения переходов и устройств генерации адресов – рис.6.15).

Устройство отката - упорядоченное устройство, преобразующее временные результаты внеочередного выполнения в постоянное состояние вычислительной системы и выполнено в виде файла регистров (Register Retirement File – RRF).

Интерфейс шины является частично упорядоченным устройством, отвечающим за связь с внешним миром. Интерфейс шины взаимодействует непосредственно с кэшом второго уровня и поддерживает до 4 параллельных обращений к кэшу. Он также управляет обменом данными с основной памятью.

Устройство выборки/декодирования

Структура этого устройства представлена на рис. 6.14. Устройство выборки/декодирования осуществляет быструю предварительную выборку кодов линейной программы из кэша команд. Указатель на следующую команду – это индекс кэша команд, содержимое которого определяется буфером переходов, состоянием процессора и сообщениями о неправильном предсказании перехода, поступающими из устройства выполнения целочисленных команд.

На начальной стадии устройство IFU0 выбирает строку кэша, соответствующую индексу в указатели на следующую команду, и следующую за ней строку. Выборка команд происходит 16-байтными блоками и считывается две строки подряд. Две строки кэша считываются из-за того, что команды в архитектуре Intel выравниваются по границам байтов, и поэтому может происходить передача управления на середину или конец строки кэша.

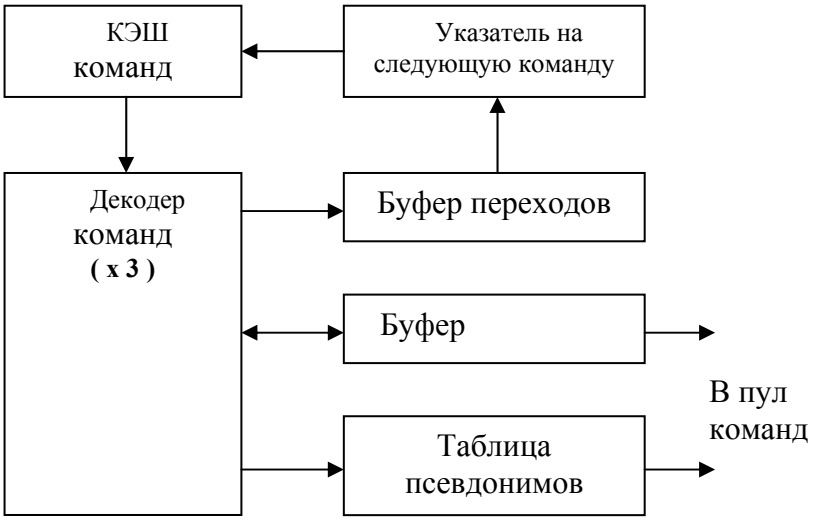


Рис. 6.14. Устройство выборки/декодирования

На следующей ступени устройствами IFU0 и IFU1 производятся вращение предвыбранных байтов (выравнивание по 16 байтным границам), пометка начала и конца команд и подачи их в декодеры команд. Эти две стадии занимают три такта процессора.

После окончания фазы выборки команд осуществляется их декодирование. Декодер

ID0 принимает поток отмеченных байтов и обрабатывает их, определяя форматы поступивших команд и распределяя их в три параллельно работающих декодеров блока ID1 или в планировщик последовательности микрокоманд (MIS – Microcode Instruction Sequencer).

Декодеры преобразуют команды x86 в микрокоманды. В P6 имеется три параллельных декодеров – два из них "простые", а третий "сложный". Простые декодеры обрабатывают команды x86, транслируемые в единственную микрокоманду. Сложный же декодер работает с командами, которые соответствуют от одной до четырех микрокоманд. Некоторые особенно сложные микрокоманды невозможно непосредственно декодировать даже сложным декодером, они передаются планировщику последовательности микрокоманд (MIS), генерирующего необходимое число микрокоманд. Если простой декодер встречает микрокоманду, которая не поддается трансляции, то в конечном счете она оказывается в сложном декодере, либо в планировщике последовательности микрокоманд. Такая пересылка слегка замедляет дешифрацию, но, возможно, не сказывается на исполнение благодаря буферизации с помощью станции RS. Декодеры способны генерировать в общей сложности шесть микрокоманд за такт, если сложные и простые команды безупречно выровнены декодером ID0, но в более типичном случае из всех трех декодеров за один такт выдаются три микрокоманды. Именно поэтому P6 и имеет трех потоковый суперскалярный механизм обработки данных. Как правило, в среднем этим трем микрокомандам соответствует чуть меньше трех команд x86.

Декодеры ID1 преобразуют команды x86 в микрокоманды – триады (операция, указатель на операнд, указатель на место помещения результата). Первый декодер за один такт способен декодировать одну команду, содержащую до четырех микрокоманд. Два других декодера могут за такт декодировать только команду, состоящую из одной микрокоманды.

Одну микрокоманду имеют команды типа регистр - регистр и команды загрузки. Из двух микрокоманд состоят команды записи и команды чтения - модификации. Команды регистр - память состоят из двух - трех микрокоманд. Команды чтение - модификация - запись состоят из четырех микрокоманд. Сложные команды содержат набор заранее составленных последовательностей микрокоманд, количество которых в наборе превышает четырех. Декодирование таких команд происходит за несколько тактов. Если при программировании на ассемблере планировать последовательность команд так, чтобы они содержали по 4-1-1 микрокоманд, число команд декодированных за один такт будет максимальной (три команды).

При обнаружении декодером команд инструкции условного или безусловного переходов выполняется стадия определения адреса перехода с помощью блока BTB. Для определения адреса ветвления обычно требуется выполнить целочисленное сложение, прибавляющее к указателю на следующую команду смещение, заданное в поле команды ветвления. Это не требует дополнительных тактов для обращения к регистрам и осуществляется благодаря блоку BTB, содержащие ранее использованные адреса переходов.

Буфер переходов состоит из двух буферов BTB0 и BTB1. Буфер BTB0 отслеживает все целевые адреса потока команд следующих за командой ветвления при невыполнении условия перехода, а буфер BTB1 – второй поток команд при выполнении условия перехода (см. рис. 5.14,в).

Буфер переходов (BTB) хранит историю происходивших переходов и их целевых адресов. Каждый из 512 элементов BTB хранит целевой адрес и четыре бита предыстории, которые несут информацию о том, случился ли переход за последних четыре прохода через команду ветвления, ссылающийся на данный адрес. Если на основание анализа предыстории предсказывается переход, целевой адрес посылается в блок выборки, не дожидаясь выполнения команды ветвления. Тем самым реализуется динамический метод предсказания переходов. Кроме BTB, процессор P6 имеет буфер стековых возвратов RSB (Return Stack Buffer), позволяющий корректно предсказывать адрес возврата из процедур, которые

вызываются из разных точек. Таким образом, вызов процедуры в линейной последовательности кодов не приводит к потере производительности предвыборки.

По числу штрафных циклов переходы различаются следующим образом:

1) переходы, не приводящие к штрафным циклам. К ним относятся корректно предсказанные отсутствия переходов к адресам, которых нет в BTB (по умолчанию принимается отсутствие переходов);

2) переходы, приводящие к минимальному числу штрафных циклов (приблизительно к одному). К ним относятся переходы, корректно предсказанные с помощью BTB;

3) ошибочно предсказанные переходы приводят к потере 10 – 15 циклов, а иногда и до 26.

Для переходов с целевыми адресами (безусловные переходы), которых в BTB еще нет, применяется статическое предсказание.

После декодирования микрокоманды сохраняются в буфере микрокоманд (рис. 6.14) и реализуется процедура переименования регистров с помощью блока RAT, после чего каждая из микрокоманд вместе с дополнительной информацией о ее состоянии (статусе) посылается в пул команд (блок ROB R).

На стадии переименовании регистров определяются существенные зависимости по данным (RAW) между командами и преодолеваются несущественные (WAW, WAR). Для того чтобы преодолеть несущественные зависимости, возникающих в результате ограниченности логических ресурсов (ячеек памяти, регистров), в процессоре P6 используется механизм динамического отображения определяемых текстом программы логических ресурсов на физические ресурсы микропроцессора.

Таким образом, когда команда создает новое значение для логического регистра, физический ресурс, в который помещается это значение, получает имя. Последующие команды, использующие это значение, снабжаются именем физического ресурса. Данная процедура называется переименованием регистров.

Устройство диспетчирования/выполнения

Организация устройства диспетчирования/ выполнения показана на рис. 6.15.

В основе его построения лежит устройство диспетчирования, представляющую собой пятипортовую распределительную станцию (RS) и очередь микрокоманд. Взаимодействие пула команд с исполнительными устройствами осуществляется через эту распределительную станцию.

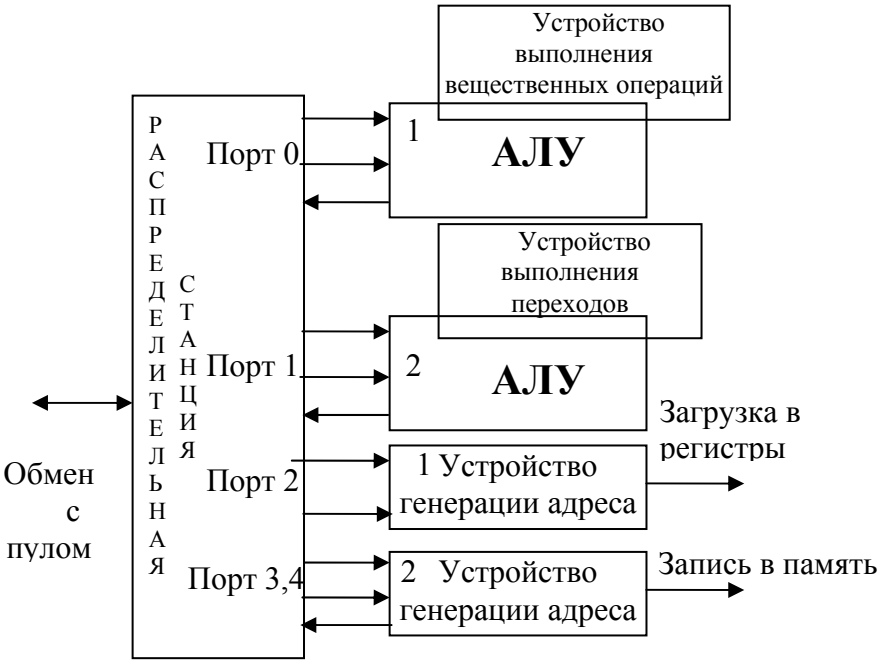


Рис. 6.15. Устройство диспетчирования/выполнения

Порт 0, … , порт 4 – порты вычислительного ядра с неупорядоченным исполнением, на которых группируются исполнительные устройства:

1. порт 0 содержит целочисленное АЛУ и устройство выполнения вещественных операций;
2. порт 1 – целочисленное АЛУ и устройство выполнения переходов;
3. порт 2 – блок загрузки данных;
4. порт 3 – блок записи адреса;
5. порт 4 – блок записи данных.

Устройство диспетчирования выбирает микрокоманды из пула команд в очередь микрокоманд RS в зависимости от их статуса. В нем может храниться до двадцати микрокоманд в единственной централизованной станции RS (центральное окно команд), обслуживающей все его исполнительные блоки. Станция RS напрямую соединяется со всеми исполнительными блоками P6. Он может посылать максимум пять микрокоманд за такт, но при работе с типичными командными последовательностями x86 более вероятен непрерывный поток пересылок с интенсивностью в три микрокоманды за такт.

Распределительная станция RS состоит из пяти портов, каждая из которых содержит позицию для размещения кода операции, наименование первого операнда, самого операнда, признака доступности первого операнда, наименование второго операнда, самого операнда, признака доступности второго операнда и наименование регистра результата. При выполнении всех этих условий микрокоманды получают статус «готовы к выполнению». Если статус микрокоманды показывает, что ее операнды вычислены и доступны, а необходимое для ее выполнения вычислительное устройство (ресурс) также доступно, то устройство диспетчирования направляет ее на выполнение.

Когда микрокоманда завершает исполнение и вырабатывает результат, то наименование результата сравнивается с наименованиями операндов в распределительной станции. Если устройство диспетчирования обнаружит микрокоманду, ждущую этого результата, то данные записываются в соответствующие позиции и устанавливается признак их доступности. Таким образом, результаты выполнения предыдущих микрокоманд можно использовать в качестве исходных операндов для последующих, поэтому все результаты из каждого устройства исполнения посылаются назад в станцию RS.

По сути дела, процессор P6 применяет сложную схему межсоединений, подобную матричному переключателю, которая связывает все выходные порты исполнительных блоков со станцией RS. Это позволяет продвигать (data forwarding) результаты в другой исполнительный блок, которому данный результат требуется в качестве входного операнда, без задержки, вызванной обновлением и повторным чтением регистров.

Распределительная станция следит за доступностью операндов. Если микрокоманда попадает в распределительную станцию, все готовые операнды из регистрового файла переписываются в поля этой микрокоманды. Для P6 распределительная станция содержит не сами операнды, а указатели на них в переупорядочивающем буфере (ROB W). Когда все операнды микрокоманды будут доступны, тогда станция RS инициирует исполнение микрокоманды, и его результаты выполнения будут возвращены в пул (ROB W).

Таким образом, планирование процесса выполнения микрокоманд осуществляет устройство диспетчирования. При этом момент направления микрокоманд на исполнительное устройство определяется только потоками данных (у P6 три потока) и доступностью ресурсов, без какой бы то ни было связи с первоначальным порядком команд в программе.

Алгоритм, осуществляющий планирование выполнения микрокоманд, является крайне важным для производительности процессора в целом. Если в каждом такте для каждого ресурса готова к выполнению одна микрокоманда, то проблема выбора не возникает. Она возникает в случае, если имеется несколько микрокоманд готовых к выполнению на данном ресурсе. Идеальным был бы выбор микрокоманды, выполнение

которой привело бы к максимальному сокращению графа потоков данных выполняемой программы. Из-за невозможности определения такой микрокоманды, используется алгоритм планирования имитирующая модель «первый пришел – первый обслужен» (FIFO).

Команды, которые исполняются не в той последовательности, которая предписана программой, следует в конечном итоге расположить в должной последовательности – иначе процессор не всегда сможет получить правильные результаты. Пул команд ROB сохраняет статус исполнения и результаты каждой микрокоманды. Микрокоманда выводится, и результаты заносятся в архитектурные регистры и память только после того, как станет известно, что предыдущие микрокоманды завершились. Это выглядит так: после того как команды были декодированы и регистры переименованы, микрокоманды помещаются в ROB R (циклическая очередь FIFO) последовательно, в определяемом программой порядке. Это происходит тогда, когда эти же самые микрокоманды пересылаются в станцию RS.

Последовательный процесс упорядочения очень важен для восстановления программного порядка микрокоманд после исполнения с изменением последовательности. Это обеспечивает упорядоченное обновление архитектурных регистров и ячеек памяти после завершения исполнения.

Тот факт, что и ROB R, и RS получают одни и те же микрокоманды в одно и то же время из декодеров – важный элемент архитектуры P6. Пока ROB R следит, за программной последовательностью микрокоманд, RS сохраняет их в своем буфере и определяет момент, когда конкретная микрокоманда будет готова к пересылке в соответствующее устройство исполнения.

Выступая в роли резервуара, буфер микрокоманд хранит группу декодированных команд с тем, чтобы исполнительные блоки продолжали работать, даже если декодеры "зависали". И наоборот, если устройства исполнения заняты, станция RS предоставляет декодерам возможность продолжить работу. В редких случаях, когда буфер микрокоманд станции RS заполняется, блок декодирования может приостановить свою работу.

Станция RS, работающий согласованно с буфером ROB, – вот механизм, позволяющий процессору исполнять программы с измененной последовательностью команд. По существу, процессор освобождается от необходимости исполнять каждую команду в той последовательности, которая предписывается программой, зато он может рассмотреть несколько ожидающих своей очереди микрокоманд и определить, какая из них наилучшим образом подходит для исполнения в данный момент времени. Принятое решение основывается на таких факторах: доступность операндов, не занятость нужных исполнительных блоков и устранения взаимозависимостей.

Поскольку система команд x86 содержит множество команд перехода, многие микрокоманды также являются переходами. В типичной программе до 10% команд могут быть безусловными переходами, и еще 10-20 % представлять собой условные переходы. Безусловные переходы проблем не вызывают; процессор "уверен", что они будут выполнены, и поэтому просто начинает выборку команд по указанному адресу.

Алгоритм, реализованный в буфере переходов для условных команд, обеспечивает более чем 93% точность предсказания. При неправильном прогнозе применен следующий подход. Микрокомандам, следующие за микрокомандой перехода, еще в упорядоченной части конвейера ставятся в соответствии адрес следующей команды и предполагаемый адрес перехода для спекулятивного выполнения. После вычисления перехода устройством выполнения переходов реальная ситуация сравнивается с предсказанным адресом. Если они совпадают, то проделанная работа оказывается полезной, спекулятивным микрокомандам, следующим за микрокомандой перехода, присваивается статус «выполнено», а сама микрокоманда перехода удаляется из пула команд. При несовпадении адресов устройство выполнения переходов изменяет статус всех микрокоманд, засланных в пул после микрокоманды перехода, с целью их удаления. При этом правильный адрес перехода направляется в буфер переходов, который перезапускает весь конвейер с нового адреса.

Устройство отката

Структура устройства отката RRF представлена на рис. 6.16. Он состоит из устройства управления и файла регистров, способного за один такт принимать до трех завершенных микрокоманд.

Устройство отката проверяет статус микрокоманд в пуле команд (ROB R). Оно ищет микрокоманды, которые уже выполнены и могут быть удалены из пула. Удаление выполненных микрокоманд устройством отката производится с учетом первоначального порядка команд в программе.

Результаты из исполнительных блоков направляются назад в ROB W, после чего RRF определяет, готова ли микрокоманда к удалению. В процессе удаления происходит запись результатов – обновление архитектурных регистров и сохранение данных в памяти.

Операции записи в память также откладываются до тех пор, пока вызывавшая их микрокоманда не будет выведена из ROB R. Для этого в P6 предусмотрен буфер упорядочения обращений к памяти (файл регистров), в котором по командам, выдаваемым устройствами в память, сохраняется информация и данных, и адресов

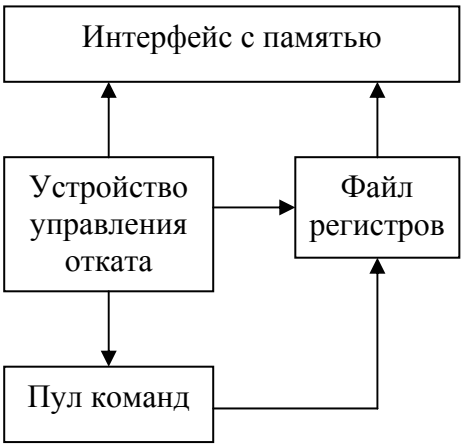


Рис 6.16.

Процесс отката занимает два такта. В первом такте устройство управления отката отыскивает готовые к удалению микрокоманды, затем оно определяет, какие из этих микрокоманд могут быть удалены из пула в соответствии с исходным порядком команд в программе. В это же время результаты из буфера ROB W переписываются в упорядоченный файл регистров блока RRF, который представляют собой архитектурные регистры процессора P6. Во втором такте устройство управления отката осуществляет запись в память в соответствии программному коду.

Блок RRF обеспечивает вывод всех микрокоманд в указанном программой порядке; максимальная скорость вывода – три микрокоманды за такт, что примерно соответствует средней производительности декодеров.

Кэш-память и интерфейс шины

На рис. 6.17 изображена структура кэш-памяти и интерфейса шины.

Процессор P6 имеет два уровня встроенной кэш-памяти. Первый уровень состоит из четырехканального наборно-ассоциативного кэша команд и двухканального наборно-ассоциативного кэша данных. Кэш второго уровня представляет собой единый четырехканальный наборно-ассоциативный кэш емкостью от 128 до 1024 Кбайт. Длина строки обоих кэша – 32 байта. Строки процессор заполняет всегда целиком пакетными циклами чтениями (4 передачи на строку) из основной памяти, выровненными по 32-байтным границам. Любой внутренний запрос процессора на обращение к памяти

направляется во внутренний кэш. Если затребованная область памяти присутствует в строке внутреннего кэша, то он обслужит этот запрос. Вторичный кэш устраняет многие промахи первого кэша. В случае промаха на обоих уровнях, минимальная задержка передачи

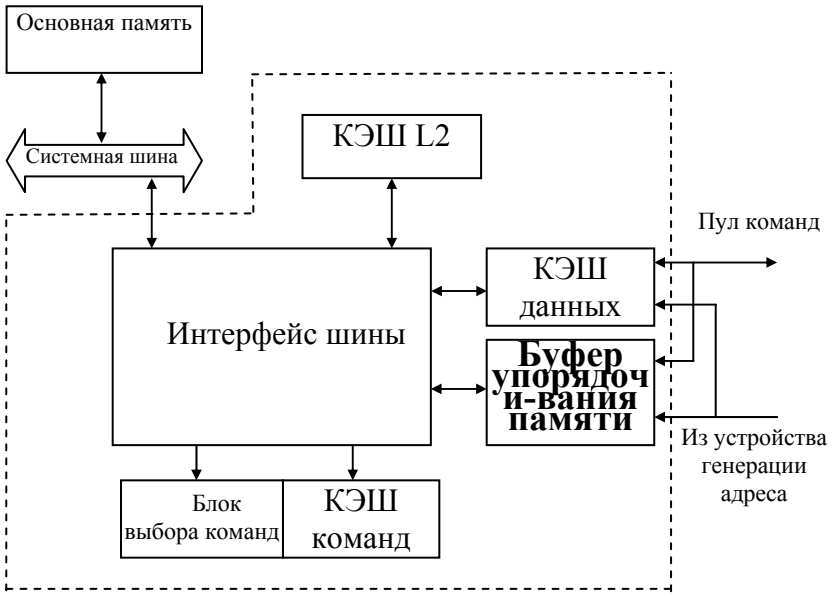


Рис. 6.17. Кэш – память и интерфейс шины

данных из основной памяти составляет 10-15 тактов (в зависимости от попадания в страницу). К кэшу данных возможен одновременный доступ по записи и чтению, если запросы относятся к разным банкам кэш-памяти.

Использование кэш-памяти в мультипроцессорных системах требуют специальных мер предосторожности. Дело в том, что когда один из процессоров обращается к фрагменту памяти, с которым другой процессор работает через свою кэш-память, первый может получить неправильные данные. Процессор P6 (а также Pentium) обеспечивает специальные механизмы, которые гарантируют соблюдение кэш-согласования. Кэш-согласование означает, что если один из процессоров, входящий в систему из нескольких процессоров, и, возможно, нескольких внешних устройств кэш-памяти, изменяет какие-либо данные, то все процессоры при обращении к этим данным получает эти изменения.

Кэш-согласование достигается с помощью использования протокола MESI (modified/exclusive/ shared/invalid). Согласно этому протоколу, каждой строке в кэш-памяти данных присваивается одно из четырех состояний. Это означает, что в тег адреса строки вводятся два дополнительных бита тега состояния. Эти теги состояния строки могут быть изменены как процессором, содержащим эту кэш-память, так и другими устройствами, подключенными к внешней шине.

Состояние строк для каждого процессора определяется следующим образом:

M – состояние (изменённая). Строка доступна только в одном устройстве кэш-памяти и была изменена (его содержимое отлично от содержимого соответствующего ей фрагмента основной памяти). Доступ к M-строке (и чтение и запись) может быть осуществлен без обращения к основной памяти через внешнюю шину;

E – состояние (исключительная). Строка так же доступна в одном устройстве кэш-памяти, но она не изменялась (ее копия в основной памяти действительна). Доступ к E-строке (чтение или запись) также может быть осуществлен без обращения к внешней шине. Запись в E-строку переводит её в M-состояние;

S – состояние (разделённая). Возможно, эта строка присутствует более чем в одном устройстве кэш-памяти. Чтение S-строки не приводит к действиям на шине, но запись в S-строку генерирует цикл записи через внешнюю шину в основную память. При этом та же строка в других устройствах кэш-памяти становится недействительной.

I – состояние (недействительная). Строка недоступна в кэш-памяти. Чтение I-строки приводит к считыванию фрагмента в кэш из основной памяти. Запись в I-строку генерирует цикл записи через внешнюю шину в основную память.

Кэш-согласование для кэш-памяти команд первого уровня поддерживается с помощью подмножества MESI-протокола. Строки в кэш-памяти команд могут находиться только в двух состояниях: S- и I- состояниях. Это связано с тем, что запись в кэш-память команд запрещена во избежание некорректного изменения кода.

Интерфейс шины генерирует два типа обращения к основной памяти: чтение из памяти в регистр и запись из регистра в память. При чтении из памяти задаются адрес памяти, размер блока считываемых данных и регистр - назначение. Команда чтения кодируется одной микрокомандой. При записи задаются адрес памяти, размер блока записываемых данных и сами данные. Поэтому команда записи кодируется двумя микрокомандами: первая генерирует адрес, вторая готовит данные. Микрокоманды чтения и записи планируются независимо и могут выполняться параллельно. Они могут переупорядочиваться в буфере записи.

В процессоре P6 реализована архитектура подсистемы памяти, позволяющая командам чтения опережать команды записи и другие команды чтения. Записи из буферов производятся в память всегда в порядке, предписанном программным кодом. Процессор поддерживает логическое соответствие порядка физических операций чтения и записи памяти их порядку в программном коде. При этом операции чтения могут иметь любой порядок, они могут пропускать буферизованные записи, а записи в память идут всегда в порядке, предписанном программой. Буфер упорядочивания памяти служит в качестве диспетчера и буфера переупорядочивания. В нем хранятся отложенные команды чтения и записи, и он осуществляет их повторное диспетчирование, когда блокирующее условие (зависимость по данным, недоступность ресурсов) исчезает.

Динамическое выполнение команд в процессоре P6 резко повышает частоту запросов процессорного ядра к шине за данными и командами, поскольку ядро одновременно обрабатывает несколько команд. Для обхода узкого места - внешней шины - процессорное ядро использует архитектуру независимой двойной шины (рис.5.50.). Одна из этих шин используется только для связи с вторичной кэш-памятью, расположенной в том же корпусе микросхемы. Эта шина является локальной и в геометрическом смысле - проводники имеют длину порядка единиц сантиметров, что позволяет использовать ее на частоте ядра процессора. Значительный объем вторичного кэша позволяет удовлетворять большинство запросов к памяти, при этом коэффициент загрузки локальной шины достигает 90 %. Вторая шина процессорного кристалла выходит на внешние выводы микросхемы и является системной шиной процессора P6. Эта шина работает на внешней частоте независимо от внутренней шины.

По статистическим данным загрузка процессором внешней шины для обычных применений составляет 10 % от ее пропускной способности (528 Кбайт/сек. при 66 МГц и 800 Кбайт/сек. при 100 МГц), а для серверных применений может достигать 60 % при четырехпроцессорной конфигурации.

Таким образом, снижение нагрузки на внешнюю шину, за счет введения локальной шины, позволяет эффективно использовать процессор P6 в многопроцессорных системах.

Как видно из организации процессора P6, его основное вычислительное ядро выполнено по технологии RISC–процессоров с применением архитектурных методов, используемых при их построении. Следующим фактором повышения производительности процессора является применение принципа многоуровневой конвейеризации, позволяющего увеличить пропускную способность конвейера. Операции, выполняемые на каждой ступени процессора Pentium, все еще остаются достаточно сложными, и требуется их дальнейшее разбиение. Так как пропускная способность конвейера определяется самой медленной стадии, то при разработке конструкции конвейера необходимо обеспечить равенство скоростей обработки на всех стадиях. Для системы команд x86, имеющие в своем составе

простые (от 1 до 7 байт), сложные (7-11 байт) и очень сложные команды, обеспечить данное условие невозможно. В этом случае для повышения пропускной способности конвейера применяют метод межстадийной буферизации. Межстадийная буферизация в процессоре P6 используется во многих местах конвейера, но, особенно, полезна в интерфейсах памяти, поскольку ее быстродействие относительно невелико. Буферизация также используется между двумя основными функциями исполнения команды: выборки и выполнения. Роль буфера на этой стадии выполняет пул команд, который позволяет реализовывать принцип динамического исполнения команд.

Разбиение сложных команд x86 на более простые операции (микрокоманды) привело к увеличению числа ступеней конвейера. При этом конвейер процессора P6 выглядит следующим образом:

1. Выборка команд из кэша (IF0);
2. Выравнивание по границам параграфа (IF1,IF0);
3. Декодирование, определение формата команд (ID0);
4. Декодирование команд (ID1 – три декодера);
5. Определение адресов перехода (BTB0, BTB1);
6. Переименование регистров (RAT);
7. Загрузка микрокоманд в пул (ROB R);
8. Загрузка микрокоманд в буфер станции RS;
9. Диспетчирование (распределение по портам);
10. Выполнение (работа ФИУ);
11. Запись результатов в пул (ROB W);
12. Переупорядочивание (RRF).

Здесь показаны основные стадии выполнения команд на процессоре P6. Этап выборки, в свою очередь, разбиваются на две стадии, где на первой стадии происходит выборка двух строк кэша команд, а на второй стадии из выбранных байтов выделяются команды. Эти команды помечаются и направляются в устройство декодирования.

Н а третьей стадии происходит определение формата команд блоком ID0 и подготовка их передачи в блок ID1, содержащий три параллельно работающих декодера, либо планировщику последовательности микрокоманд (MIS).

На четвертой стадии три параллельных декодера позволяют в лучшем случае декодировать 3 команды за 1 такт с соотношением 4-1-1. Для декодирования сложных (7-11 байт) и сверхсложных команд (11-16 байт) семейства x86 требуются от 1 до 2 дополнительных машинных тактов соответственно, и используется планировщик последовательности микрокоманд (MIS). Следовательно, эта ступень конвейера исполняется за три такта.

На пятой стадии (после декодирования инструкций ветвлений) процессор осуществляет вычисление адресов перехода с помощью блоков BTB0 и BTB1 и формирует два потока микрокоманд (один поток помечается - при отрицательном исходе условия, другой – при положительном).

На шестой стадии осуществляется процедура переименования регистров.

На седьмой стадии микрокоманды с выхода устройства декодирования записываются в пул команд (ROB R), и в этой точке заканчивается упорядоченная часть конвейера.

На восьмой и девятой стадиях происходит загрузка микрокоманд в буфер станции RS и диспетчирование их по исполнительным устройствам.

На десятой стадии производится выполнение микрокоманд в исполнительных устройствах. Алгоритм планирования выполнения микрокоманд, реализованный в процессоре P6, может запустить до 5 микрокоманд за такт.

На 11 стадии после выполнения микрокоманд на исполнительных устройствах их результаты возвращаются в пул команд (ROB W). На операции диспетчирования, выполнения, сохранения результатов выполненных микрокоманд, процессор затрачивает по одному машинному такту.

На заключительной стадии происходит упорядочивание потоков команд и данных, реализуемое устройством отката и выполняется за два машинных такта. На этой стадии конвейера происходит сохранение результатов выполнения команд в памяти машины. При отсутствии промаха эти результаты сохраняются в кэш-памяти данных, а при промахе - они записываются в память системы в режиме отложенной записи. Этот режим обеспечивает буфер упорядочивания памяти (RRF).

Большинство основных операций (целочисленная арифметика и логика, умножение с плавающей точкой) могут конвейеризироваться с производительностью исполнения в одну, две операции за такт. Делитель FPU неконвейеризирован. Длительные операции могут выполняться параллельно с короткими командами.

Основные, функциональные устройства P6 выполнены в виде конвейеров и в совокупности образуют 12-ти ступенчатый конвейер, реализующий трех потоковую модель обработки данных.

Таким образом, наряду с новыми введенными архитектурными методами использование многоуровневой конвейеризации с параллельной обработкой на всех стадиях позволяет дополнительно увеличить производительность на 30% относительно процессора Pentium.

6.4.2. Сравнительные характеристики Pentium и P6

В таблице 6.3 приведены архитектурные особенности микропроцессоров пятого и шестого поколений.

Процессор ММХ базируется на ядре Pentium и представляет собой расширение набора команд с использованием технологий ОКМД (SIMD – Singes Instruction Multiple Data), предназначенное для ускорения мультимедийных и коммуникационных программ за счет параллельной обработки. Набор команд ММХ содержит 57 новых операций и новые типы 64-битных данных. Эти новые типы данных хранят упакованные целочисленные

Таблица 6.3

Процессор	Технология, мкм	Частота системной шины, МГц	Частота ядра, МГц	Кэш команд / данных, Кбайт L1	Кэш L2, Кбайт	Предсказание переходов	Опережающее исполнение (спекулятивное)	Переименование регистров	Внеочередное исполнение	Поддержка ММХ	Многопроцессорные системы	Сокет (слот)
Pentium 1 поколение	0.8	60 66	60 66	16 8/8	-	+		-	-	-	FRC**	Сокет4
Pentium 2 поколение	0.6 0,35	50 60 66	75 90 100 120 133 150 166 180 200	16 8/8	-	+	-	-	-	-	FRC 2SMP*	Сокет 5,7

Pentium MMX	0.35	66	166 200 233	32 16/16	-	+	-	-	+	+	2SMP	Сокет7
Pentium Pro	0.6 0.35	66	150 166 180	16 8/8	256 512	+	+	+	+	-	FRC, 4SMP	Сокет 8
Pentium II OverDrive	0.35	66	333	32 16/ 16	512	+	+	+	+	+	FRC, 2SMP	Сокет 8
Pentium II	0.35 0.25	66 100	233 266 300 350 400	32 16/16	512	+	+	+	+	+	FRC, 2SMP	Слот 1
Pentium II Xeon	0.25	100	400	32 16/16	512 1024 2048	+	+	+	+	+	FRC, 8SMP	Слот 2
Celeron	0.25	66	266 300 300A 333	32 16/ 16	128	+	+	+	+	+	–	Слот 1
Pentium III	0.18	133	< 1 ГГц	16/16	256	+	+	+	+	+	2SMP	Слот 1
Pentium 4	0.18	400	>1.4 ГГц	8/8	256	+	+	+	+	+	-	Сокет 423

**FRC – контроль с помощью функциональной избыточности
*SMP – симметричные мультипроцессорные системы

значения во время выполнения операций ММХ. Дополнительно введены восемь новых 64-битных регистров. Одновременно обрабатываемое 64-битное слово может содержать как одну единицу обработки, так и восемь
однобайтных, четыре двухбайтных или два четырехбайтных операндов.

Кроме ММХ-расширения в архитектуре Pentium ММХ имеется ряд усовершенствований, повышающих его производительность.

В целочисленный конвейер после ступени PF введена дополнительная ступень F, на которой производится синтаксический разбор команд. Для увеличения точности предсказания переходов увеличены в два раза размер кэш-памяти команд и данных (16*16 Кбайт) и размер буфера адресов переходов (512 вхождений). Улучшена возможность параллельных вычислений (процессор способен выполнять две ОКМД команды с 16-битными данными за 1 такт).

Суперскалярная архитектура Pentium ММХ позволяет выполнять команды парами, с

учетом следующих ограничений:

1) АЛУ исполняет арифметические и логические операции. Наличие двух АЛУ позволяет выполнять эти команды парами на обоих конвейерах;

2) Устройство умножения исполняет все операции умножения за 3 такта, оно конвейеризовано, что позволяет получать результат от очередного запроса в каждом такте. Процессор имеет одно устройство умножения, так что операции умножения не могут исполняться парами. Однако они могут исполняться в паре с любыми другими командами. Умножения могут исполняться как на U, так и на V конвейерах.

3) Сдвиговое устройство выполняет все операции сдвигов, упаковки и распаковки. Так как это устройство одно, то эти команды могут выполняться в паре только с другими командами.

4) Команды MMX-расширения, требующие доступа к памяти или к регистрам, могут выполняться только U конвейером и не могут выполняться в паре не с MMX-командами.

Процессор Pentium2 основан на архитектуре Pentium Pro, в которую добавлено несколько исполнительных устройств для операций MMX. Теперь порт0 дополнительно содержит АЛУ MMX и устройство умножения MMX, а порт1– АЛУ MMX и устройство сдвигов MMX.

6.5. Процессор Pentium 4

В настоящее время микропроцессоры Intel Pentium 4 имеют наивысшие тактовые частоты среди всех серийно выпускаемых процессоров. Сейчас в процессоре Pentium 4 частота достигла рубежа в 1,5 ГГц, в не столь отдаленном будущем можно ожидать рост частоты до 15 ГГц.

Первый Pentium 4 изготавливался по 0,18 – микронной технологии и содержал 42 млн. транзисторов. Однако площадь его кристалла вдвое больше, чем у Pentium 3. Он поставляется в защищенном корпусе FC – PGA и устанавливается в разъем Socket 423 с 423 контактами. Применение Pentium 4 требуют более дорогую память Rambus и интерфейсные БИС серии i850, которые используют 6-слойные материнские платы. Кроме этого, он имеет повышенное электропитание 52-55 Вт.

В 2001 году был выпущен Pentium 4 с тактовой частотой 2 ГГц (2,26., 2,4., 2,53 ГГц). Он был реализован по 0,13 – микронной технологии, поддерживает системную шину на частоте 533 МГц. Объем кэш-памяти L2 доведен до 512 Кбайт. Чуть больше года потребовалось Intel, чтобы увеличить тактовую частоту процессора для ПК на 50% и преодолеть рубеж в 3 ГГц.

Главным достоинством Pentium 4 с частотой 3,06 ГГц заключается в том, что новый процессор – первый из семейства изделий, предназначенных для настольных ПК – поддерживает технологию Hyper Threading (многопоточная), которая дает заметный выигрыш в производительности (до 25%).

Суть технологии Hyper Threading состоит в том, что один физический процессор воспринимается операционной системой (ОС) как два логических процессора, благодаря чему эффективнее используются ресурсы компьютера. Эта технология специально разработана для применения в многозадачных средах многопоточных приложений. В перспективе тактовая частота микропроцессора для настольных систем к 2010 году достигнет 15 ГГц. Это в пять раз будет превышать частоту Pentium 4 и будет содержать около 1 млрд. транзисторов.

Для достижения высшей производительности Pentium 4 использовали два подхода: совершенствование микроархитектуры процессора и увеличение его тактовой частоты. Микроархитектура процессора Pentium 4 получила название NetBurst. Основной задачей микроархитектуры NetBurst состоит в том, чтобы обеспечить возможность дальнейшего роста тактовой частоты.

Одним из основных факторов увеличения тактовой частоты является совершенствование технологии изготовления. Здесь, несмотря на прогресс в технологии (прежде всего за счет уменьшения размеров элементов), приходится учитывать конечность скорости распространения сигнала (задержки, связанные с его распространением), которая накладывает определенные ограничения на рост тактовой частоты. Другим фактором увеличения тактовой частоты является использование гиперконвейера, как основного элемента микроархитектуры NetBurst.

Гиперконвейерный подход основан на уменьшение показателя, равного произведению числа стадий конвейера на время такта. Увеличение числа стадий конвейера приводит к тому, что каждая из них становится более простой и может быть выполнена за более короткое время такта.

Недостатком гиперконвейерного подхода является, как было рассмотрено ранее, проблема заполнения конвейера (перезагрузки при неверном предсказании перехода и наличие взаимозависимостей между командами).

Усовершенствования микроархитектуры NetBurst по сравнению с процессором P6 являются не столь кардинальными, как это было при переходе от Pentium к P6. Основные архитектурные идеи процессора P6, связанные с декодированием x86 – команд во внутренние RISC- подобные микрооперации, их постановкой в очередь и внеочередным опережающим (спекулятивным) выполнением с последующим упорядочиванием завершившихся микроопераций, сохранились и в NetBurst.

Основные отличительные особенности архитектуры NetBurst следующие:

1) гиперконвейерная технология;
2) кэш трассировки исполнения (TC - Execution Trace Cache);
3) 400-мегагерцовая системная шина;
4) механизм ускоренного выполнения (целочисленных АЛУ);
5) расширенное динамическое выполнение (одновременно до 126 команд);
6) потоковые расширения SSE2 (Streaming SIMD Extensions2);
7) усовершенствования функционального исполнительного устройства с плавающей запятой (мультимедийной обработки);
8) усовершенствованная кэш-память.

6.5.1. Организация Pentium 4

Основные элементы микроархитектуры Pentium 4 представлены на рис. 6.18.

В общем случае логическая схема Pentium 4 соответствует структуре процессоров P6. В нем можно выделить следующие логические блоки:

- устройство выборки/декодирования, называемое фронтальной частью Pentium 4;
- устройство планирования/выполнения, образующее исполнительное ядро процессора;
- устройство отката, соответствующее завершающей части процессора.

Фронтальная часть является упорядоченным устройством процессора Pentium 4 и состоит из следующих блоков: кэш-памяти первого и второго уровней (L1,L2); кэша трассировки TC; ПЗУ микроопераций (постоянное запоминающее устройство - ROM); устройства предсказания переходов совместно с буфером адресов переходов (BTB) и декодера команд.

Фронтальная часть процессора обеспечивает предварительную выборку (Prefetch), декодирование команд (Instruction Decode), кэширование микроопераций в TC и предсказание переходов.

Исполнительное ядро является неупорядоченной частью процессора и представляет собой базовый конвейер, который отвечает за выполнение микроопераций с элементами механизмов внеочередного и спекулятивного их исполнения. Он состоит из блока

распределения и переименования регистров, очереди микроопераций, планировщика, исполнительных устройств и файла регистров.

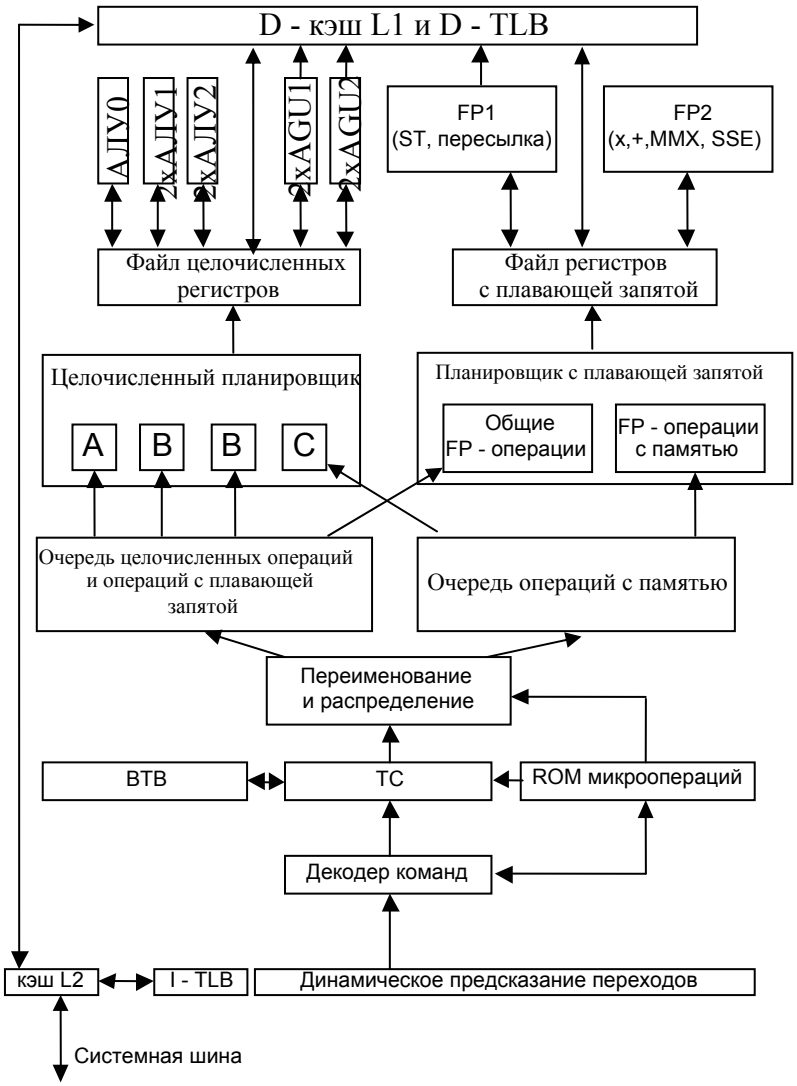


Рис. 6.18. Основные элементы архитектуры Pentium 4

Блок распределения регистров в зависимости от типа микроопераций служит для выделения необходимых регистров для загрузки или записи в память. В Pentium 4 имеются 48 буферов для команд загрузки (L) и 24 буфера для команд записи (ST).

Блок переименования регистров служит для преобразования ссылок логических регистров процессора типа EAX в ссылки на физические регистры, количество которых в Pentium 4 равно 128. Работа с большим числом физических регистров позволяет снять взаимозависимости между микрооперациями типа WAW и WAR. Процессор Pentium 4 позволяет разрешать и зависимость типа RAW, используя метод одновременного использования данных (data forwarding). В этом случае, данные, которые команда ST записывает в память, обнаруживаются и направляются в команду L напрямую.

Очередь микроопераций Pentium 4 разбивается на две части: очереди операций с памятью и очереди целочисленных операций и операций с плавающей запятой. Существует 2 типа очередей микроопераций – один для операций с памятью (загрузка и хранение) и один для остальных операций. Каждые из этих двух типов очередей хранят микрооперации в структуре FIFO (First-In, First-Out). Микрооперации из каждой очереди могут быть прочитаны неупорядочено. Это позволяет планировщикам производить динамическое планирование (переупорядочивание) микроопераций.

Устройство планирования или планировщик служит для управления работой функциональных исполнительных устройств. Планировщик осуществляет прием потока

микроопераций, определяет зависимость между микрооперациями с последующим их разрешением и распределяет исполнительные устройства для выполнения микроопераций. Планировщик состоит из двух частей в зависимости от типа обрабатываемых данных: целочисленного планировщика и планировщика с плавающей запятой.

Целочисленный планировщик обрабатывает четыре потока данных поступающих из очереди микроопераций (рис. 6.18) среди них: один поток «медленных» целочисленных операций блока A (сложные, число микроопераций для одной инструкции превышает четырех); два потока «быстрых» целочисленных операций блока B (число микроопераций меньше 4 – это характерно для простых инструкций подчиняющих правилу сдваивания) и поток операций с памятью блока C для целочисленных данных.

Планировщик с плавающей запятой работает с двумя потоками данных: операциями с плавающей запятой и операциями с памятью для данных с плавающей запятой. Планировщик содержит четыре выходных порта, с помощью которых происходит взаимодействие с функциональными исполнительными устройствами.

Основу базового конвейера составляют функциональные исполнительные устройства конвейерного типа, количество которых равно семи. Устройства АЛУ1, АЛУ2 являются высокоскоростными и используются для обработки простых целочисленных инструкций, которые подключаются к портам 0 и 1 соответственно.

Устройства выполнения АЛУ1, АЛУ2 процессора Pentium 4 разрабатывались для оптимизации выполнения простых операций. Процессор Pentium 4 может выполнять простые, часто встречающиеся целочисленные операции АЛУ, на двойной тактовой частоте процессора. Устройства, позволяющие осуществлять такие вычисления, называются высокоскоростными АЛУ (Fast ALU). Процессор выполняет операции АЛУ на двойной частоте процессора за три быстрых цикла (быстрый цикл равен двойной частоте цикла процессора). Выполнение операций на двойной частоте процессора позволяет увеличить скорость выполнения для большинства программ почти в 2 раза.

Сложные целочисленные операции поступают на отдельный аппаратный узел, называемый медленное целочисленное АЛУ0 (Slow Integer ALU). Устройство АЛУ0 предназначено для обработки сложных целочисленных инструкций и подключается к порту 1. К сложным операциям относится большинство целочисленных сдвиговых операций, такие инструкции как shift и rotate. Эти операции выполняются за четыре такта процессора. Целочисленное умножение и деление также имеет большое время выполнения. Типичное умножение и деление выполняются за 14 и 60 тактов соответственно.

Для операций с памятью Pentium 4 располагает двумя исполнительными устройствами AGU1 и AGU2 (Address Generation Unit), которые подключаются к портам 2 и3 соответственно. Блок генерации адреса AGU1 служит для загрузки операндов из памяти, а блок AGU2 – для записи операндов в память.

Для обработки инструкций с плавающей запятой имеются два независимых конвейера FP1 и FP2. Устройство FP1 предназначено для обработки простых инструкций с плавающей точкой (FXCH, SSE Move, Store и др.) и подключено к порту 0. Для обработки сложных инструкций (сложение/умножение/деление, инструкции MMX и т.п.) служит устройство FP2, которое связано с портом 1. Кроме того, устройство FP2 дополнено специальными аппаратными средствами для реализации мультимедийного расширения SSE2.

Инструкции, включенные в SSE2 – расширение, позволяют ускорить работу таких приложений, как трехмерная графика, распознавание речи, обработка изображений и другие мультимедийные программы. В SSE2 включены 144 новых инструкций (дополнительно к 70 инструкциям SSE-расширения), которые работают с операндами, расположенными в памяти или XMM – регистрах.

Инструкции SSE позволяли оперировать с восемью 128-битными регистрами XMM0..XMM7, в которых хранились по четыре вещественных числа одинарной точности. При этом все SSE операции проводились одновременно над четверками чисел, в результате чего специально оптимизированные программы, в которых производилось большое

количество однотипных вычислений (а к ним, помимо обработки потоков данных в какой-то мере относятся и 3D-игры), получали существенный прирост производительности.

SSE2 же оперирует с теми же самыми регистрами XMM и обратно совместим с SSE процессора Pentium III. А расширение набора команд Pentium 4 вызвано тем, что теперь операции со 128-битными регистрами могут выполняться не только с четверками вещественных чисел одинарной точности, но и с парами вещественных чисел двойной точности. Процессор может работать также с шестнадцатью однобайтовыми целыми, с восемью короткими двухбайтовыми целыми, с четырьмя четырехбайтовыми целыми, с двумя восьми байтовыми целыми. То есть, теперь SSE2-расширение представляет собой симбиоз MMX и SSE и позволяет работать с любыми типами данных, вмещающимися в 128-битные регистры.

Блок SSE2 инструкций реализован по архитектуре OKMD (SIMD) и относится к классу векторных конвейеров. Он поддерживает операции над двумерными векторами, элементами которых являются 64-разрядные числа с плавающей запятой двойной точностью. Использование этого блока позволяет Pentium 4 выполнять две операции с плавающей запятой двойной точности за 1 такт.

К недостаткам реализации SSE2 Pentium 4 можно отнести следующее: 80-разрядное представление чисел с плавающей точкой, используемые в x87-инструкциях, заменяется на 64-разрядное. Такая замена диктует необходимость перекомпиляции программного кода x87 инструкций. Неполная конвейеризация арифметических операций (64-разрядное умножение, сложение) приводит к пропуску одного такта между соседними инструкциями.

Файл регистров Pentium 4 играет роль накопителя, где хранятся операнды, участвующие при выполнении микроопераций и их результаты. Он содержит 128 регистров и разделяется на два файла: файла целочисленных регистров и файла регистров с плавающей запятой. Из них 48 регистров служат для загрузки операндов, 24 регистра для записи в память, а остальные для хранения результатов и переименования регистров. Завершающая часть процесса обеспечивает изымание результатов закончивших выполнение микроопераций из файла регистров и «сборку» из них результатов в порядке, задаваемом исходными программными кодами в соответствии с правилами архитектуры IA-32.

Фронтальная часть

Увеличение эффективной частоты системной шины до 400 МГц повысило максимальную, допустимую пропускную способность оперативной памяти до 3,2 Гбайт/сек. Это связано с увеличением тактовой частоты процессора и предполагает использование более дорогой быстродействующей памяти Rambus.

От системной шины данные и команды попадают в интегрированный в Pentium 4 кэш второго уровня емкостью 256 Кбайт, который по своим техническим характеристикам близок к используемому в Pentium III. Оба кэша работают на частоте ядра и являются 8-канальными наборно-ассоциативными, реализующий алгоритм обратной записи. Однако в Pentium 4 увеличена длина строки кэш L2-до 128 байт (2 сектора по 64 байт).

Задержка по обращению в кэш второго уровня составляет 7 тактов. Пропускная способность кэш L2 при частоте 1,5 ГГц составляет 48 Гбайт/сек. Для замещения информации в кэш L2 используется алгоритм псевдо-LRU. При непопадании в кэш L2 требуется 12 тактов процессора на организацию взаимодействия с системной шиной, и еще не менее 6-12 тактов шины (если она свободна) на доступ в подсистему оперативной памяти.

Кэш первого уровня является 4-канальным наборно-ассоциативным, использующий алгоритм сквозной записи, и имеет длину строки 64 байт. Емкость его уменьшилась вдвое по сравнению с Pentium III и составляет 8 Кбайт. Это уменьшение вызвано тем, чтобы обеспечить задержку выборки из кэша L2, равную 2 тактам (в предыдущих процессорах Intel она равнялась 3 тактам).

Кэш второго уровня кроме данных содержит также команды, которые попадают из него в декодер. Процессор Pentium 4 содержит один декодер, и процесс декодирования вообще исключен из базового конвейера. Непрерывность подачи микроопераций в исполнительную часть ядра Pentium 4 обеспечивается потоком микроопераций из TC, который фактически служит кэшем команд первого уровня. В TC команды сохраняются в декодированном виде, как микрооперации. В идеальном случае TC может выдавать 3 микрооперации за такт, как и три декодера P6. Емкость TC составляет 92-96 Кбайт и содержит до 12 тысяч микроопераций со средней длиной 64 разряда (хотя максимальная длина микрооперации в Pentium III достигает 118 бит).

Применение схемы один декодер и TC по сравнению со схемой трех декодеров в P6 имеет ряд преимуществ. Во-первых, если будут декодироваться сразу три сложные x86-команды, темп поступления микроопераций может упасть практически до нуля в отличие от варианта с одним декодером. Во-вторых, когда имеется короткий цикл в P6 его команды приходиться декодировать при каждом проходе, а в Pentium 4 он окажется один раз декодированным в кэше TC.

Имеется еще одна особенность при декодировании команд в Pentium 4. когда процессору требуется декодировать сложную x86-команду, которая преобразуется в длинную последовательность микроопераций (сложные инструкции, больше 4 микроопераций), взводится специальный флаг, и эти микрооперации извлекаются не из TC, а из памяти ROM-типа (рис. 6.18). Поскольку такие команды в программах встречаются редко, это не оказывает отрицательного влияния на производительность.

Фронтальная часть процессора Pentium 4 отвечает за реализацию механизма предварительной выборки и предсказания переходов.

Процессор осуществляет автоматическую предварительную выборку команд в буфера командных потоков из пути, предсказанного в BTB. Осуществляется это порциями по 32 байта, начиная с адреса перехода. Процессор также поддерживает автоматическую выборку данных и команд в кэш второго уровня. Кроме аппаратной реализации предварительной выборки имеются 4 команды предварительной выборки данных в кэш-память различных уровней. Они дают процессору подсказку о необходимости переноса в кэш соответствующей строки. Использование этих команд способствует в ряде случаев скрыть задержки по обращению в память за счет извлечения оттуда данных до того, как они реально будут использованы, что приводит к существенному росту производительности. Эти команды требуют много процессорных ресурсов, и их применение следует ограничивать (в многопроцессорных SMP-системах их использовать не рекомендуется).

Предсказание переходов является одним из компонентов микроархитектуры, определяющих производительность Pentium 4. Если предсказание верное, то в базовом конвейере не возникает дополнительных задержек, а при неверном - равно глубине конвейера, т.е. порядка 20 стадий. Использование новых алгоритмов предсказания позволило уменьшить число неправильных предсказаний на одну треть по сравнению с Pentium III. В процессоре Pentium 4 могут предсказываться все типы ближних (near) переходов, включая условные, безусловные, непрямые и переходы типа вызов/возврат из подпрограммы (CALL/RET). Предсказания для дальних (far) переходов не делаются. При этом используются несколько механизмов предсказаний.

При статическом предсказании переход назад (к младшим адресам, это характерно для циклов) предсказывается, а вперед - нет. При предсказании переходов CALL/RET, адреса возврата берутся из стека адресов возврата, имеющего емкость 16 строк.

Динамическое предсказание переходов (т.е. направление и адреса переходов) основываются на традиционной таблице предсказания переходов в буфере BTB емкостью 4096 строк. Если в буфере BTB не имеется необходимой информации, применяется статическое предсказание.

В Pentium 4 предусмотрен еще один вид предсказания переходов – «подсказка» со стороны программного обеспечения. Эта подсказка реализована в форме префикса команд

условного перехода. Подсказки порождаются компилятором и используются в процессе декодирования команд в последовательность микроопераций в ТС. Если информации о переходе нет в BTB, то подсказка имеет приоритет над статическим предсказанием. Недостатком применения подсказок является небольшое увеличение размера кода программы.

Исполнительное ядро процессора

Исполнительная часть ядра процессора представляет собой базовый конвейер, как основного элемента микроархитектуры NetBurst.

Число стадий этого конвейера увеличено почти вдвое по сравнению с P6 и глубина его достигает 20. Приведем основные операции, выполняемые базовым конвейером по стадиям.

На стадиях 1 и 2 с использованием значения счетчика команд у буфера переходов (BTB-branch Target Buffer) в кэш трассировки ТС помещаются микрооперации.

На стадиях 3 и 4 происходит выборка микроопераций у ТС, и на стадии 5 микрооперации передаются в блок распределения ресурсов (блок переименования регистров или регистров псевдонимов). Эта стадия введена в связи с задержкой передачи данных по проводникам.

На 6 стадии происходит распределение ресурсов, требуемых для выполнения микрооперации. На этой стадии в зависимости от микрооперации выделяются буферы для загрузки регистров или буферы для записи в память.

На стадиях 7 и 8 производится переименование регистров. На стадии 9 происходит помещение микроопераций в очередь, откуда они выбираются в блок планирования, отвечающий за работу с разными типами функциональных исполнительных устройств.

На стадиях 10-12 осуществляется собственно планирование: запись в блок планирования, определение зависимостей между микрооперациями и ожидание разрешения этих зависимостей. На стадиях 13-14 микрооперации посылаются в конвейер с плавающей запятой или в один из целочисленных конвейеров.

На стадиях 15-16 происходит чтение из файлов регистров целочисленных операндов или операндов с плавающей запятой. Стадия 17-это стадия выполнения. Эта стадия выполняется за 1 такт, хотя сложные целочисленные инструкции и операции с плавающей запятой требуют больше тактов, а некоторые целочисленные операции запускаются каждые полтакта.

На стадии 18 осуществляется вычисление значения флагов («равно нулю», «перенос» и т.д.), которые используются в командах условных переходов.

На стадии 19 происходит сравнение вычисленного результата команды перехода со сделанным ранее предсказанием. Наконец, последняя стадия осуществляет запись результата проверки перехода, выполненного на предыдущей стадии, в буфер переходов BTB.

В каждом такте ядро может посылать микрооперации в один из четырех портов, причем 2 из них могут выдавать на исполнение по 2 микрооперации за такт (точнее, по одной микрооперации каждые полтакта).

Таким образом, в каждом такте ядро может запускать на выполнение до 6 микроопераций, т.е. обрабатывать одновременно до шести потоков данных.

Порт 0 за первые полтакта может отправить на выполнение инструкции с плавающей запятой (FXCH и др.) в блок FP1 и простые целочисленные инструкции в АЛУ 1, а на втором полутакте еще одну целочисленную инструкцию на АЛУ1.

Первый порт в первом полутакте может запустить все SIMD-команды и арифметические инструкции с плавающей запятой (сложение/умножение/деление и т.п.) в блок FP2, либо сложные целочисленные команды с обычной скоростью в блок АЛУ0, а также одну простую инструкцию в блок АЛУ2. Во втором полутакте первый порт может выдать еще одну простую целочисленную инструкцию на АЛУ2. Второй порт запускает

одну операцию загрузки регистров за такт (L), а третий порт – одну операцию записи в память (ST) за такт.

Одновременная обработка шести потоков данных достигается за счет механизма ускоренного выполнения. Арифметический логический модуль, производящий вычисления с целыми числами, работает в два раза быстрее основного ядра, что вкупе с большой тактовой частотой процессора позволяет увеличить скорость выполнения программ.

Увеличение числа стадий конвейера и использование механизма расширенного динамического выполнения позволяет обрабатывать одновременно в разных стадиях конвейера до 126 микроопераций, что втрое больше, чем у Pentium III, и предсказывать оптимальную последовательность их исполнения.

Таким образом, анализ отличительных особенностей микроархитектуры Net Burst позволяют Pentium 4 достигать наивысшей производительности при работе с длинными векторами, что характерно для многих приложений научного характера. Такие приложения позволяют получать хорошую локализацию в кэш-памяти, которые не будут вызывать потери производительности из-за неверного предсказания переходов.

При использовании SSE2 расширения будут также ускоряться такие приложения, как трехмерная графика, распознавание речи, обработка изображений и другие мультимедийные приложения. Для популярных офисных приложений и некоторых программ делового назначения со случайным обращением в память выигрыш в производительности может оказаться несущественным.

6.6.Перспективные архитектуры микропроцессоров

Хотя достигнутый уровень конвейеризации и распараллеливания процесса в P6 и Pentium 4 позволил резко повысить их производительность, дальнейшее повышение скорости обработки связаны с решением ряда проблем, стоящими перед этими процессорами.

Среди этих проблем можно отметить сложность логики, обеспечивающих загрузку функциональных исполнительных устройств (ФУ); проблемы пропускной способности и задержек при обращении к разным уровням иерархии памяти - от кэша до оперативной памяти; проблемы предсказания переходов и т.д. Нерешенность этих проблем грозит простоями ФУ современных суперскалярных микропроцессоров и невозможность дальнейшего повышения их производительности.

Один из подходов увеличения степени загрузки основных ФУ в настоящее время базируется на известной концепции VLIW (Very Large Instruction World – сверхбольшое командное слово). Этот подход использован при создании фирмами HP и Intel микропроцессора IA-64 (Itanium). В основе его лежит два наиболее принципиальных нововведения по сравнению с процессорами RISC-архитектуры: применение технологии явного параллелизма на уровне команд (EPIC – Explicitly Parallel Instruction Computing) и использование предикатных вычислений.

Между тем близкий к этому подход уже был реализован в нашей стране - в произведенном в единственном экземпляре суперкомпьютере Эльбрус-3, выпущенным в 1991 году. Далее появившееся сообщение весной 1999 года о разработке группой российских компаний “Эльбрус” микропроцессора E2K представляет большой интерес и говорит о сходности архитектуры IA-64 с E2K.

Основной идеей в архитектуре VLIW является введение в командное слово компонент управляющие отдельными блоками микропроцессора, которые вводят явный параллелизм на уровне команд. При этом задача распараллеливания работы отдельных блоков возлагается на компилятор, который должен сгенерировать машинные команды, содержащие явные указания на одновременное исполнение операций в разных блоках. В современных суперскалярных микропроцессорах такое распараллеливание вычислительного процесса осуществляется в ходе выполнения команд аппаратным способом.

6.6.1. Архитектура E2K

Так как архитектура E2K основана на концепции сверхдлинных командных слов, то используемый формат команды является ключевым моментом в его построении. В классическом варианте VLIW процессора длина команды фиксирована. Так в IA-64 3 команды объединятся в связки длиной в 128 разрядов (рис. 6.19).

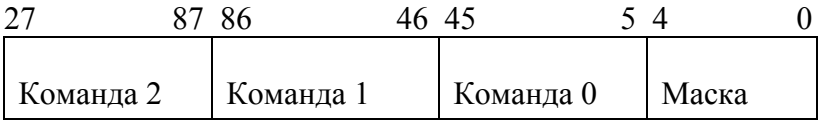


Рис. 6.19. Формат команд IA-64

В формат команды вводятся специальные разряды маски, которые указывают на зависимость между командами. Разряды маски указывают как на зависимость внутри одной связки, так и на зависимость между связками команд. Наличие такой зависимости подавляет возможность параллельного выполнения соответствующих операций. С другой стороны, фиксированная длина команды приводит к значительному дополнительному расходу памяти для команд (т.е. увеличению кэша команд) и является ограничением на “масштабируемость” микропроцессора (т.е. возможность наращивания числа ФУ, не требующих изменения формата команды).

В E2K формат команды имеет переменную длину и представлен на рис. 6.20.

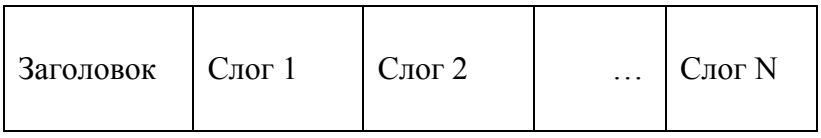


Рис.6.20. Формат команды E2K

Команда E2K состоит из слогов длиной в 32 разряда каждый. Число слогов может изменяться от 2 до 16, с возможностью расширения до 32 слогов. Любая команда всегда включает заголовок и от 1 до 15 слогов, указывающих на операции, которые могут выполняться параллельно. Слог заголовка содержит информацию о структуре команды и ее длине, что упрощает дешифрацию команды переменной длины.

В таблице 6.4 показаны основные типы слогов применяемых в командах E2K.Слоги в команде располагаются слева на право в определенном порядке в том, в котором они указаны в таблице 6.4 сверху вниз.

Для распараллеливания процесса выполнения команд в E2K имеется сверхбольшой файл регистров. Все регистры процесса являются универсальными и могут содержать как целочисленные данные, так и числа с плавающей точкой. Всего имеются 256 регистров по 64 разряда каждый. Для сравнения, в IA-64 имеется 128 целочисленных и 128 регистров с плавающей точкой. Для реализации циклов с постоянным шагом или для контекстного переключения программы в E2K используются оконный механизм переключения регистров. . Для реализации циклов с постоянным шагом или для контекстного переключения программы в E2K используются оконный механизм переключения регистров. В суперскалярных процессорах этот механизм называется переименованием регистров.

Таблица 6.4

N°	Типы слогов	Число слогов

1.	Заголовок	1
2.	Операции АЛУ	6
3.	Управление подготовкой перехода	3
4.	Дополнительные операции	2
5.	АЛУ при зацеплении	
	Загрузка из буфера предварительной выборки массивов в регистр	4
6.	Литеральные константы для ФУ	4
7.	Логические операции с предикатами	3
8.	Предикаты и маски для управления ФУ	3

Набор регистров, используемых в цикле, представляется в виде окна, накладываемого на регистровый файл. На начала окна указывает специальный регистр базы. При переходе на следующую итерацию цикла регистр базы увеличивается на размер окна, что эквивалентно продвижению окна вперед по файлу регистров, настолько регистров, сколько их имеется в окне. При контекстном переключении адресация регистра внутри контекста происходит относительно базы, а при вызове другой процедуры достаточно сменить значение базы.

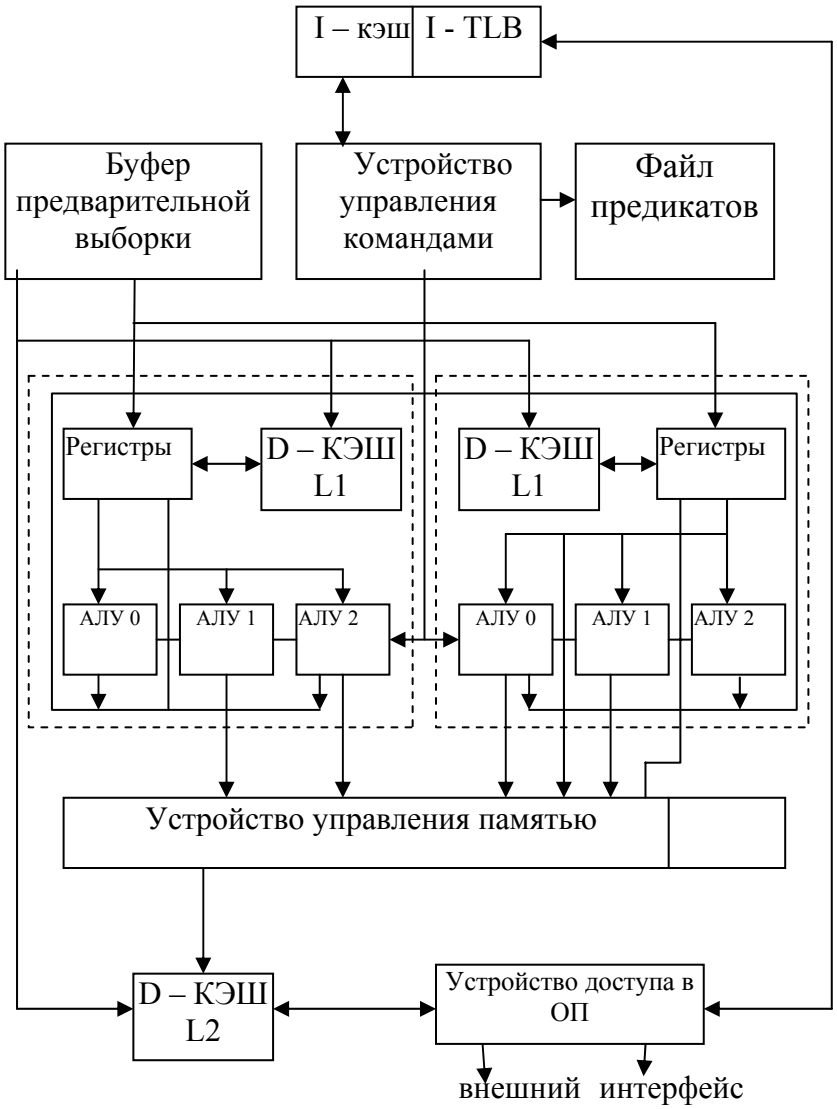


Рис. 6.21. Структурная схема E2K

Недостатком этого подхода является накладные расходы - необходимость суммирования относительного регистрового адреса с базой, что требует дополнительной стадии в конвейере.

На рис.6.21 представлена упрощенная структурная схема E2K. Процессор содержит 2 кластера. Внутри кластеров логические блоки расположены достаточно близко друг к другу, что позволяет уменьшить расстояние между блоками и увеличить тактовую частоту до 2 ГГц и выше. Кластеры содержат по одной копии файла регистров и ФУ. Каждый кластер содержит по 256 регистров. Всего в процессоре имеется 30 регистровых портов: 20 портов чтения (по 10 на кластер) и 10 портов записи.

Кластеры E2K являются почти симметричными и содержат по три одинаковых целочисленных АЛУ, за исключением того, что один кластер имеет два ФУ деления - целочисленного и с плавающей точкой. Такая симметрия позволяет направлять практически любую команду в любой кластер.

Система команд E2K более расширена, чем у традиционных RISC-процессоров: в нем представлены четырехадресные команды, например, d=a+b+c. Такого нет в IA-64. Команды с плавающей точкой, помимо полной поддержки стандарта IEEE754, осуществляют поддержку работы с 80-разрядным представлением Intel x86. Кроме того, для приближения системы команд к x86 в E2K реализованы также команды расширения MMX.

Все ФУ в E2K конвейеризированы, так целочисленный конвейер имеет длину 8 тактов, длина конвейера с плавающей точкой на этапе выполнения составляет 4 такта, а конвейер загрузки регистров/записи в оперативную память имеет длину 9 тактов. E2K обеспечивает высокий уровень параллельно выполняемых операций, в команде кодируется до 23 таких операций (сюда, кроме арифметико-логических операций входят также доступ к оперативной память, приращение индекса массива и т.п.).

В E2K используется трехуровневая схема кэш-памяти. Кэш первого уровня имеет емкость 8Кбайт и продублирован в каждом кластере. Этот кэш является кэшом с прямым отображением, не блокирующий и использует алгоритм сквозной записи данных. Кэш данных в обоих кластерах содержит одинаковые данные и имеет по два порта в командном кластере. Время доступа составляет два такта, а пропускная способность при попадании составляет 38 Гбайт/с (при частоте 1,2 ГГц), что можно сопоставить с пропускной способностью файла регистров-288 Гбайт/с. При непопадании в кэш данных первого уровня E2K никогда не останавливает работу. Этот кэш имеет буфер адресов отсутствующих строк кэша. Обычно кэш второго уровня успевает заполнить кэш первого уровня до того, как этот буфер переполнится. Если же это все-таки происходит, новые запросы к памяти будут направлены прямо в кэш второго уровня.

Кэш данных второго уровня в E2K имеет емкость 256 Кбайт при времени доступа в 8 тактов. Он является двухканальным наборно-ассоциативным и имеет 4 банка, то есть обеспечивает 4-кратное расслоение памяти. Этот кэш реализует алгоритм обратной записи и также является не блокирующим. Учитывая большую ширину команды в E2K, кэш команды первого уровня имеет емкость 64 Кбайт при длине строки 256 байт. Этот кэш является 4-канальным наборно-ассоциативным. Ширина магистрали, по которой команды из кэша поступают в устройство управления командами, составляет 256 байт, однако минимальная скорость заполнения кэша составляет 32 байта за такт.

В E2K предусмотрены два варианта подключения кэша третьего уровня. Непосредственно через локальную шину, что позволяет разгрузить системную шину, и через коммутатор. В процессоре E2K предусмотрены также два независимых порта по 32 байта, они работают на частотах 400-600 МГц.

В архитектуре E2K, как и в IA-64, применяется техника предикатных вычислений, основная задача которой является уменьшение количества условных переходов и, соответственно, ошибочных предсказаний направления переходов. Для этого в E2K имеется 32 одноразрядных регистра-предиката, причем команда способна сформировать до 7 предикатов: 4 в операциях сравнения в АЛУ и еще 3 - в операциях логики с предикатами (см.

табл. 6.4). В IA-64 предикатных регистров в два раза больше, так как в них хранятся пары – предикат и его отрицание. В IA-64 поля предикатов всегда представлены в команде, а в E2K - могут отсутствовать.

Предикаты могут использоваться в канале АЛУ или в канале доступа к массивам, для этого используется условные слоги, содержащие маски предикатов и ФУ. Всего в этих слогах может кодироваться до 6 предикатов, указывающих на то, нужно ли выполнять соответствующие операции из «широкой» команды.

Если условие перехода удается вычислить до выполнения этого перехода, компилятор стремится применить предикатные вычисления, чтобы обойтись без переходов вообще. Если это не удается, компилятор порождает явные спекулятивные коды.

Компилятор E2K порождает коды для обеих ветвей программы, возникающих при условном переходе, и, пользуясь большим числом ФУ и регистров, заставляет процессор выполнять обе ветви программы. Эта же процедура применяется и в IA-64. До тех пор, пока условие остается неизвестным, обе ветви выполняются спекулятивно. Как только условие выполнится, выбираются нужные результаты. Признак спекулятивного выполнения взводится при этом в спекулятивном бите кода операций соответствующего слога. При возникновении ситуации исключения результат снабжается тегом недействительного значения. В этом случае ненужная ветвь программы должна быть удалена.

В суперскалярных процессорах останавливается конвейер, очищается буфер предварительной выборки, после чего начинается загрузка новых команд. Накладные расходы в подобных случаях велики, например у P6 составляет 10-15 тактов.

Для сохранения накладных расходов в E2K применяется так называемая операция подготовки переходов. Она считывает команды из кэша по адресу перехода и может быть выполнена заранее. Когда переход происходит, команды по новому адресу частично будут выполнены. E2K может выполнять одновременно до трех операций подготовки перехода.

В настоящее время разработка E2K еще не завершена, она находится на уровни, так называемого Verilog-описания, что позволяет проводить ”эмуляционное” исполнение кодов со скоростью порядка 10 команд в секунду. При этом заявленная тактовая частота составляет 1,2 ГГц при 0,18-микронной технологии изготовления.

Разработчики E2K привели полученные ими оценки производительности: SPECintT95/FP95=135/350, что можно сопоставить с оценками SPECint95/fp95=45/70 для Merced (IA-64) с частотой 800 мГц. Площадь E2K оценивается в 126 кв.мм при тепловыделении 35 Вт соответственно.

Таким образом, повышение производительности современных микропроцессоров возможно за счет совершенствования полупроводниковой технологии и применение новых архитектурных решений.

Совершенствование полупроводниковой технологии осуществляется в направлении увеличения степени интеграции сверхбольших интегральных схем (СБИС). Степень интеграции СБИС зависит от размера кристалла и количества помещенных на нем транзисторов. Основным фактором, определяющим возможность увеличения числа транзисторов в СБИС, являются минимальные топологические размеры элементов, называемые проектными нормами. По мере уменьшения проектных норм может быть увеличена и тактовая частота работы микропроцессора. Однако потребляемая мощность при этом является одним из основных факторов, ограничивающих сложность кристалла.

В свое время высокие уровни быстродействия и степени интеграции связывались главным образом с арсенид-галлиевыми полупроводниковыми приборами. Однако высокая стоимость и большая рассеиваемая мощность ограничивает области их применения. В настоящее время большие надежды связываются с так называемой кремниево-германиевой технологией, которая является улучшенной разновидностью биполярной КМОП-технологии (BICMOS), но обеспечивает низкий уровень энергопотребления, присущий обычному производственному процессу CMOS.

Другим перспективным направлением повышения быстродействия при низкой потребляемой мощности считается объединения технологий “кремний на изоляторе” – КНИ (SOI, Silicon On Insulator) с медными межсоединениями на полупроводниковых кристаллах. По сравнению с технологией, при котором межсоединения выполнены на основе алюминия, в результате применения меди кристалл становится не только быстрее, но и меньше. Медная металлизация приводит к уменьшению общего сопротивления и соответственно увеличению скорости работы кристалла на 15 – 20%. Малое сопротивление позволяет уменьшать и геометрические размеры проводников. При этом прогнозируемый рост степени интеграции выглядит следующим образом. Стандартными проектными нормами в 2002 г. должны стать 0,13 мкм, 2005 г. - 0,1 мкм, 2008 г. - 0,07 мкм и в 2014 г. - 0,035мкм.

Последние цифры, в частности означают, что при производстве терабитных микросхем на 1 кв. см будет расположено до 390 млн. транзисторов. В свою очередь технологии КНИ снижают паразитные емкости, возникающие между элементами микросхемы и подложкой. Благодаря этому тактовая частота работы транзисторов возрастает. Увеличение скорости от использования КНИ может составить 20 – 30%. Таким образом, в идеальном случае, возможно достижение 50% общего роста производительности.

Дальнейшее увеличения производительности современных микропроцессоров можно достичь за счет увеличения скорости выполнения программ, которого можно добиться в первую очередь благодаря реализации определенного вида параллелизма. Параллелизм на уровне команд (ILP – Instruction Lively Parallelism) лежит в основе построения современных микропроцессоров: процессоров с суперскалярной архитектурой (RISC-процессоры) и процессоров со сверхдлинным командным словом (процессоры VLIW).

Суперскалярные процессоры - это реализация ILP - процессора для последовательных архитектур - архитектур, программа для которых фактически не может передавать точную информацию о параллелизме. Поскольку программа не содержит точной информации о наличии ILP, то задача обнаружения параллелизма возлагается на аппаратуру, которая в процессе выполнения программы создает план действия для обнаружения “скрытого параллелизма”.

Процессоры VLIW представляют собой пример архитектуры, для которой программа представляет точную информацию о параллелизме. Компилятор выявляет параллелизм в программе и сообщает процессору, какие операции не зависят друг от друга и что их можно выполнять одновременно в одном и том же такте. Создание плана выполнения операций во время компиляции является существенным фактором для обеспечения высокой степени распараллеливания на уровне команд.

Таким образом, применение технологии явного параллелизма и предикатных вычислений говорит о наступлении нового этапа развитии микропроцессорной технологии, связанного с появлением VLIW-процессоров.

Контрольные вопросы

1. Укажите основные факторы повышения производительности 32-разрядного микропроцессора 80386.
2. Чем вызвана необходимость разработки транспьютера? Укажите основные типы сетей на базе транспьютеров.
3. Укажите основные архитектурные особенности транспьютера Т414.
4. Определите формат команды транспьютера и средства увеличения длины операнда.
5. Какие аппаратные средства поддержки параллельных алгоритмов заложены в транспьютер?
6. Дайте краткую характеристику языку программирования ОККАМ.
7. В чем заключаются основные особенности суперскалярной архитектуры Pentium?
8. Укажите основные способы повышения производительности процессора Pentium.

9. Перечислите основные проблемы возникающие с введением второго конвейера в процессоре Pentium?

10. Каким образом происходит сдваивание команд процессора Pentium?

11. Какие методы используются в Pentium для снятия зависимости по данным?

12. Как решается проблема снятия зависимости по управлению в Pentium и поясните это на примере?

13. Опишите механизм реализации предсказаний переходов в Pentium и какие средства используются для этого?

14. Дайте краткую характеристику базовому конвейеру процессора Pentium?

15. Укажите отличительные особенности организации математического сопроцессора Pentium.

16. Дайте краткую характеристику организации кэш-памяти процессора Pentium?

17. Какие новые архитектурные решения были заложены в процессор P6?

18. Какие существуют подходы к увеличению загрузки конвейеров в архитектуре P6?

19. На какие основные стадии разбит конвейер P6 и какие функции выполняются на каждой стадии?

20. В чем заключается особенность спекулятивного выполнения команд процессора P6?

21. Как реализуется внеочередное выполнение команд и каким способом осуществляется планирование выполнения команд в процессоре P6?

22. Для чего служит буфер переходов и как он реализован в процессоре P6?

23. В чем заключается основное назначение устройства отката в процессоре P6?

24. Дайте краткую характеристику кэш-памяти процессора P6.

25. В чем состоят различия в организации конвейеров Pentium и P6?

26. Укажите основные отличительные особенности микроархитектуры Net Burst?

27. Определите состав и функции основных элементов фронтальной части микроархитектуры процессора Pentium 4?

28. Определите основное назначение кэш трассировки микроопераций процессора Pentium 4 и какую он роль выполняет?

29. Определите состав и роль буфера адресов перехода?

30. Какие способы предсказания переходов заложены в процессоре Pentium 4 и дайте краткую им характеристику?

31. Для чего служит ROM микроопераций?

32. Определите основные элементы базового конвейера Pentium 4?

33. Укажите число стадий и кратко опишите работу базового конвейера по стадиям?

34. Определите основное назначение блока назначения и переименования регистров?

35. Дайте краткую характеристику устройства планирования Pentium 4?

36. Дайте характеристику основным исполнительным устройствам Pentium 4?

37. Определите назначение SSE2 расширения Pentium 4 и укажите его основные отличительные особенности?

38. В чем заключаются основные недостатки суперскалярных микропроцессоров?

39. На каких принципах базируется концепция сверхдлинного командного слова (VLIW)?

40. Опишите формат команд процессора E2K?

41. Укажите основные элементы архитектуры процессора E2K?

42. Дайте краткую характеристику организации кэш-памяти процессора E2K?

43. Укажите перспективные направления увеличения производительности современных микропроцессоров.

Список использованной литературы

1. Аллан А. и др. Направления развития полупроводниковых технологий//Открытые системы. – 2002. - №4.
2. Балашов Е.П., Григорьев В.А., Петров Г.А. Микро- и мини- ЭВМ: Учебное пособие для вузов. – Л.: Энергоатомиздат, 1984.
3. Брамм П., Брамм Д. Микропроцессор 80386 и его программирование. – М.:Мир,1990.
4. Бройтман Д. Микропроцессор Pentium. Часть 1 // Монитор – 1993. - №3. -С.76.
5. Бройтман Д. Микроархитектура процессора P6 // Монитор. – 1995. - №3. - С.6.
6. Водяхо А.И., Горнец Н.Н., Пузанков Д.Н. Высокопроизводительные системы обработки данных: Учебное пособие. – М.: Высшая школа, 1997.
7. Григорьев В.Л. Программирование однокристальных микропроцессоров.– М.:Энергоатомиздат,1987.
8. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 1. Программная архитектура. – М.:ГРАНАЛ, 1993.
9. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 2,3,4. – М.:ГРАНАЛ, 1993.
10. Гук М.Ю. Аппаратные средства IBM PC: Энциклопедия. - СПб : Питер,1998. – 815 с.
11. Гук М. Процессоры Pentium II, Pentium Pro и просто Pentium: Архитектура. Интерфейс. Программирование. – СПб.: Питер, 1999.
12. Гук М., Юров В. Процессоры Pentium III, Athlon и другие. - СПб.: Питер, 2000.- 478 с.
13. Гук М., Юров В. Процессоры Pentium 4, Athlon и Duron. – СПб.: Питер, 2001. – 512 с.
14. Каган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств автоматики. – М.:Энергоатомиздат, 1987.
15. Кондратьев И. Intel преодалел рубеж в 3 ГГц // Computerword – 2002. - № 11.
16. Корнеев В.В., Киселев А.В. Современные микропроцессоры. – М.:Нолидж, 1998.
17. Корнеев В.В. Параллельные вычислительные системы. - М: Нолидж,1999.- 320 с.
18. Корнеев В. Эволюция микропроцессорных архитектур// Открытые системы. – 2000. - № 4.
19. Коуги П.М. Архитектура конвейерных ЭВМ: Пер. с англ. – М.:Радио и связь, 1985.
20. Кузьминский М. Отечественные микропроцессоры: Elbrus E2K//Открытые системы – 1999. - № 5-6.
21. Кузьминский М. Краткий обзор IA-64//Открытые системы. – 1999. - № 11-12.
22. Кузьминский М. Многонитевая архитектура микропроцессоров//Открытые системы. – 2001. - №1.
23. Кузьминский М. Дорога к высоким частотам//Открытые системы. – 2001. - №2.
24. Кузьминский М. Микроархитектура Itanium//Открытые системы. – 2001. - №9.
25. Куприянов М.С., Петров Г.А., Пузанков Д.В. Процессор Pentium. Архитектура и программирование. – Спб: Изд-во ЭТУ, 1996.
26. Левенталь Л. Введение в микропроцессоры: Программное обеспечение, аппаратные средства, программирование: Пер. с англ. – М.:Энергоатомиздат, 1983.
27. Майоров С.А., Новиков Г.И. Структура электронных вычислительных машин. - Л.:Машиностроение, 1979.
28. Маклафин Л. Pentium 4 наращивает мощь // Мир ПК – 2002. - №3.
29. Маклафин Л. Pentium 4 и его память // Мир ПК – 2002. - №12.
30. Микропроцессоры и микро-ЭВМ в системах автоматического управления: Справочник / С. Т. Хвощ, Н.Н. Варлинский, Е. А. Попов; под ред. С. Т. Хвоща. – Л.:Машиностроение. Ленинградское отделение, 1987.
31. Микропроцессоры 80x86, Pentium: Архитектура, функционирование, программирование, оптимизация кода / В.М. Михальчук, А.А. Ровдо, С.В. Рыжиков. – Минск: Битрикс, 1994. – 400 с.
32. Однокристальные микроЭВМ. М.:МИКАП,1994.-400с.

33. Пом А., Агравал О. Быстродействующие системы памяти: Пер. с англ.–М.:Мир, 1987.
34. Процессоры Intel 80486 Pentium / Под ред. А.Г. Алексенко. - М: Радио и связь, 1994.
35. Супер-ЭВМ. Аппаратная и программная организация: Пер. с англ. / Под. ред. С.Фернбаха. – М.:Радио и связь, 1991.
36. Таненбаум Э. Архитектура компьютера – СПб : Питер, 2002. – 704 с.
37. Транспьютеры. Архитектура и программное обеспечение: Пер. с англ./Под ред. Г.Харпа. - М.: Радио и связь, 1993.
38. Уильямс М. Горизонты процессоров Intel // Computerword – 2002. - №11.
39. Фрир Дж. Построение вычислительных систем на базе перспективных микропроцессоров: Пер. с англ. – М.: Мир, 1990.
40. Хокни Р., Джессхоуп К. Параллельные ЭВМ. Архитектура, программирование, алгоритмы. – М.: Радио и связь, 1986.
41. Шланскер М.С, Рау Б.Р. Явный параллелизм на уровне команд//Открытые системы – 1999. - №11-12.
42. Эссекс Д. Pentium 4: прорыв или провал? //Мир ПК. – 2001. - №2.
43. Carmean D. Inside the Pentium 4 Processor Microprocessor Microarchitecture // Intel Development Forum, 2000.
44. Intel Pentium 4 Processor Optimization Reference Manual – Intel, 1999-2000.
45. Pentium Processor User's Manual: v.1-3 – Intel Corporation, 1993.
46. Pentium 4 Processor Architecture and Desktop Performance Evaluation Guide – Intel, 2000.

ОГЛАВЛЕНИЕ

Петр Борисович Могнонов

ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Учебное пособие

Редактор **Е.В. Белоплотова**

Подписано в печать 08.10.2003 г. Формат 60x84 1/16.
Усл.п.л. 18,69,уч.-изд.л.18,0. Печать операт., бум. писч.
Тираж 100 экз. Заказ № 137.

Издательство ВСГТУ. г. Улан-Удэ, ул. Ключевская, 40, а.