Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный университет промышленных технологий и дизайна»
Высшая школа технологии и энергетики
Кафедра прикладной математики и информатики

КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Текст лекций для студентов всех форм обучения
по направлению подготовки
01.03.02 — Прикладная математика и информатика

Составители:
С. В. Тихов
А. И. Кушнеров

Санкт-Петербург
2025

Утверждено
на заседании кафедры ПМИ
17.10.2024 г., протокол № 2

Рецензенты:
И. В. Ремизова, А. Н. Вершинин

Текст лекций соответствует программам и учебным планам дисциплины «Компьютерные системы и сети. Вычислительные системы» для студентов, обучающихся по направлению подготовки 01.03.02 «Прикладная математика и информатика». Текст лекций охватывает весь объем материалов по дисциплине. Издание предназначено для самостоятельной работы студентов.

Текст лекций предназначен для бакалавров очной и заочной форм обучения.

Утверждено Редакционно-издательским советом ВШТЭ СПбГУПТД в качестве текстов лекций

Режим доступа: http://publish.sutd.ru/tp_get_file.php?id=202016, по паролю.
- Загл. с экрана.
Дата подписания к использованию 23.10.2025 г. Рег. № 5089/24

Высшая школа технологии и энергетики СПбГУПТД
198095, СПб., ул. Ивана Черных, 4.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данный курс лекций посвящен изложению вопросов организации структуры и функционированию компьютерных систем, при этом большое внимание уделяется вопросам эффективности традиционных и перспективных решений в области компьютерной техники. Рассмотрены вопросы классификации и архитектуры компьютеров, принципы организации памяти и устройств управления, системы ввода-вывода. Курс лекций предназначен для студентов направления подготовки 01.03.02 «Прикладная математика и информатика», обучающихся по дисциплине «Компьютерные системы и сети».

1. КЛАССИФИКАЦИЯ СОВРЕМЕННЫХ КОМПЬЮТЕРОВ

1.1. Общие тенденции развития компьютеров

По темпам развития компьютерная промышленность опережает все остальные отрасли. Главная движущая сила – способность производителей помещать с каждым годом все больше и больше транзисторов на микросхему. Чем больше транзисторов (крошечных электронных переключателей), тем больше объем памяти и мощнее процессоры. Гордон Мур, один из основателей и бывший председатель совета директоров Intel, однажды сострил по поводу того, что, если бы авиационные технологии развивались с такой же скоростью, как компьютерные, самолеты стоили бы 500 долларов и облетали землю за 20 минут на 20 литрах топлива. Правда, для этого они должны стать размером с обувную коробку.

Он же сформулировал закон технологического прогресса, известный теперь под именем закона Мура. Когда Гордон готовил доклад для одной из промышленных групп, он заметил, что каждое новое поколение микросхем появляется через три года после предыдущего. Поскольку у каждого нового поколения компьютеров было в 4 раза больше памяти, чем у предыдущего, стало понятно, что число транзисторов на микросхеме возрастает в постоянной пропорции, и таким образом, этот рост можно предсказать на годы вперед. Закон Мура часто представляется в формулировке, которая гласит, что число транзисторов па одной микросхеме удваивается каждые 18 месяцев, то есть увеличивается на 60 % каждый год. Размеры микросхем и даты их производства подтверждают, что закон Мура действует до сих пор (рис. 1).

По большому счету, закон Мура – это никакой не закон, а простое эмпирическое наблюдение о том, с какой скоростью физики и инженеры-технологи развивают компьютерные технологии, и предсказание, что с такой скоростью они будут работать и в будущем.

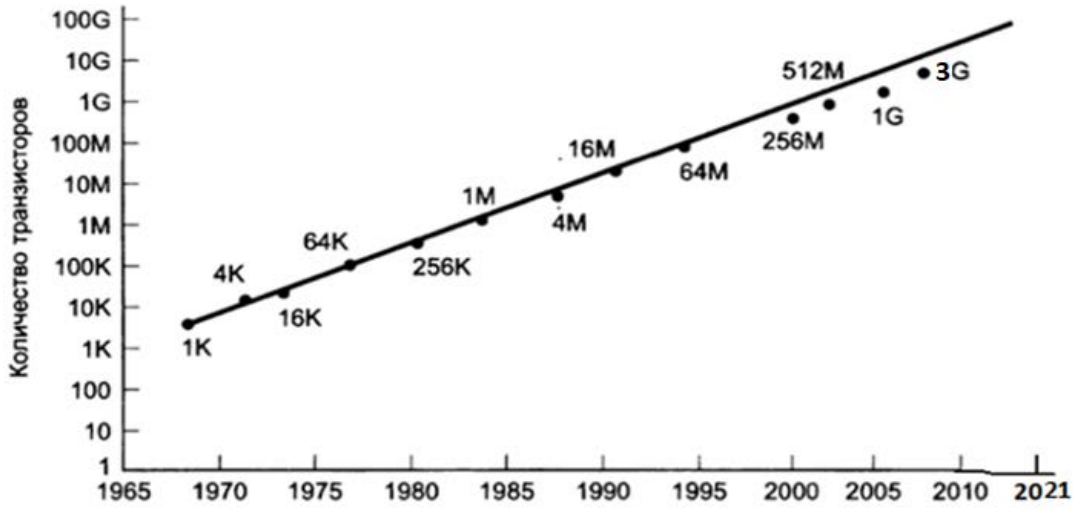

Рисунок 1 – График увеличения транзисторов на одной микросхеме (точки на графике – объем памяти в битах)

Еще один фактор развития компьютерных технологий – первый закон программного обеспечения, названный в честь Натана Мирвольда, одного из руководителей компании Microsoft. Этот закон гласит: «Программное обеспечение – это газ. Он распространяется и полностью заполняет резервуар, в котором находится». Программное обеспечение продолжает развиваться и порождает постоянный спрос на процессоры, работающие с более высокой скоростью, на память большего объема, на устройства ввода-вывода более высокой производительности.

Развивать компьютерные технологии, исходя из закона Мура, можно двумя путями: создавать компьютеры все большей и большей мощности при постоянной цене или выпускать одну и ту же модель с каждым годом за меньшие деньги. Компьютерная промышленность идет по обоим путям, создавая широкий спектр разнообразных компьютеров.

1.2. Классификация современных компьютеров

Приблизительная классификация современных компьютеров представлена в таблице 1.

Таблица 1 – Классификация компьютеров

Тип	Цена, $	Сфера применения
Одноразовые компьютеры	0,5	Поздравительные открытки
Встроенные компьютеры (микроконтроллеры)	5	Часы, машины, различные приборы
Мобильные и игровые компьютеры	**50**	**Домашние компьютерные игры, смартфоны**
Персональные компьютеры	**500**	**Настольные и портативные компьютеры**
Серверы	5000	Сетевые серверы
Мэйнфреймы	5 000 000	Пакетная обработка данных в банке

Одноразовые компьютеры

В самой верхней строчке таблицы 1 находятся микросхемы, которые приклеиваются на внутреннюю сторону поздравительных открыток для проигрывания мелодий типа «Happy Birthday», свадебного марша или чего-нибудь в этом роде. Авторам пока не доводилось видеть на прилавках открытки с соболезнованиями, играющие похоронный марш, но, поскольку идея сформулирована, можно ожидать появления и таких открыток. Те, кто

воспитывался на компьютерах стоимостью в миллионы долларов, воспринимают такие одноразовые компьютеры примерно так же, как одноразовый самолет.

Как бы то ни было, одноразовые компьютеры окружают нас. Вероятно, наиболее значимым достижением в этой области стало появление микросхем RFID (Radio Frequency Identification – радиочастотная идентификация). Теперь на безбатарейных микросхемах этого типа, толщиной меньше 0,5 мм и себестоимостью и несколько центов, устанавливаются крошечные приемопередатчики сигналов; кроме того, им присваивается уникальный 128-разрядный идентификатор. При получении импульса с внешней антенны они получают питание на время, достаточное для отправки ответного импульса со своим номером. Несмотря на крошечные размеры, спектр практического применения таких микросхем весьма значителен.

Микроконтроллеры

Вторую строку в таблице 1 занимают компьютеры, которыми оснащаются разного рода бытовые устройства. Такого рода встроенные компьютеры, называемые также микроконтроллерами, выполняют функцию управления устройствами и организации их пользовательских интерфейсов. Диапазон устройств, работающих с помощью микрокомпьютеров, крайне широк:

- бытовые приборы (будильники, стиральные машины, микроволновые печи, охранные сигнализации);
- коммуникаторы (беспроводные и сотовые телефоны, факсимильные аппараты, пейджеры);
- периферийные устройства (принтеры, сканеры, модемы, приводы CD-ROM);
- развлекательные устройства (видеомагнитофоны, DVD-плееры, музыкальные центры, MP3-плееры, телеприставки);
- формирователи изображений (телевизоры, цифровые фотокамеры, видеокамеры, объективы, фотокопировальные устройства);
- медицинское оборудование (рентгеноскопические аппараты, томографы, кардиомониторы, цифровые термометры);
- военные комплексы вооружений (крылатые ракеты, межконтинентальные баллистические ракеты, торпеды);
- торговое оборудование (торговые автоматы, кассовые аппараты);
- игрушки (говорящие куклы, приставки для видеоигр, радиоуправляемые машинки и лодки).

Мобильные и игровые компьютеры

В сущности, это обычные компьютеры, в которых расширенные возможности графических и звуковых контроллеров сочетаются с ограничениями по объему ОП и пониженной расширяемостью. Первоначально в эту категорию входили компьютеры с процессорами низших моделей для простых телефонов и игр типа пинг-понга, которые предусматривали вывод изображения на экран монитора. С годами они превратились в достаточно мощные системы, которые по некоторым параметрам производительности

ничем не хуже, а иногда даже лучше персональных компьютеров. У мобильных компьютеров появляется дополнительное ограничение: они должны потреблять как можно меньше энергии для решения своих задач.

Персональные компьютеры

Персональные компьютеры делятся на две основных категории: настольные и портативные (ноутбуки). Как правило, те и другие комплектуются модулями памяти общей емкостью в несколько гигабайт, жестким диском с данными на несколько терабайтов, звуковой картой, сетевым интерфейсом, монитором с высоким разрешением и другими периферийными устройствами. На них устанавливаются сложные операционные системы, они расширяемы, при работе с ними используется широкий спектр программного обеспечения.

Серверы

Мощные персональные компьютеры и рабочие станции часто используются и качестве сетевых серверов – как в локальных сетях, так и в Интернете. Серверы, как правило, поставляются в однопроцессорной и мультипроцессорной конфигурациях. В системах из этой категории обычно устанавливаются модули памяти общим объемом в несколько гигабайтов, жесткие диски емкостью в терабайты и высокоскоростные сетевые интерфейсы. Некоторые серверы способны обрабатывать тысячи транзакций в секунду. С точки зрения архитектуры, однопроцессорный сервер не слишком отличается и от персонального компьютера. Он просто работает быстрее, занимает больше места, содержит больше дискового пространства и устанавливает более скоростные сетевые соединения.

Мэйнфреймы

Большие компьютеры размером с комнату, напоминающие компьютеры 60-х годов и традиционно, называемые мэйнфреймами. Кроме выполнения программ, накопленных за последние 40 лет, в последние годы мэйнфреймы начали возрождаться под влиянием Интернета. Они заняли нишу мощных серверов Интернета, способных обрабатывать огромное количество транзакций в секунду, что крайне актуально для электронной коммерции в целом, и компаний, вынужденных обслуживать громадные базы данных в частности.

До последнего времени существовала еще одна крупная категория вычислительных машин – суперкомпьютеры. Их процессоры работали с очень высокой скоростью, в них устанавливались модули памяти общей емкостью в несколько десятков гигабайтов, высокоскоростные диски и сетевые интерфейсы. Суперкомпьютеры используются для решения различных научных и технических задач, которые требуют сложных вычислений, например таких, как моделирование сталкивающихся галактик, синтез новых лекарственных препаратов, моделирование потока воздуха вокруг крыла самолета. Сейчас, когда вычислительные возможности, аналогичные тем, что предлагают суперкомпьютеры, реализуются в виде кластеров, эта категория компьютеров постепенно отмирает.

Кластеры

В связи с тем, что по соотношению «цена/производительность» позиции

рабочих станций и персональных компьютеров постоянно улучшаются, в последние годы появилась практика их объединения в кластеры. Кластер состоит из нескольких стандартных серверных систем, подключенных друг к другу по высокоскоростной сети и снабженных специальным программным обеспечением, которое позволяет направлять их ресурсы на решение единых задач (как правило, научных и инженерных). В большинстве случаев компоненты кластера – это совершенно обычные компьютеры. Однако основным дополнением становятся высокоскоростные сетевые соединения, которые, как правило, тоже можно организовать при помощи стандартных сетевых плат. Нередко кластеры используются для создания веб-серверов.

2. КОДИРОВАНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ ФОРМАТЫ ДАННЫХ

2.1. Кодирование информации в компьютере

Каждому символу в компьютере (букве, цифре, знаку препинания, математическому знаку) ставится определенная двоичная комбинация. Совокупность возможных символов и назначенных им двоичных кодов образует *таблицу кодировки.* В настоящее время применяется множество различных таблиц кодировки. Объединяет их весовой принцип, при котором веса кодов цифр возрастают по мере увеличения цифры, а веса символов увеличиваются в алфавитном порядке. Так вес буквы «Б» на единицу больше веса буквы «А». Это способствует упрощению обработки в ВМ.

До недавнего времени наиболее распространенными были кодовые таблицы, в которых символы кодируются с помощью восьмиразрядных двоичных комбинаций (байтов), позволяющих представить 256 различных символов:

- расширенный двоично-кодированный код EBCBIC (Extended Binary Decimal Interchange Code);
- американский стандартный код для обмена информацией ASC II (American Standard Code for Information Interchange).

Код EBCBIC используется в качестве внутреннего кода в универсальных ВМ фирмы IBM. Он же известен под названием ДКОИ (двоичный код для обработки информации).

Стандартный код ASCII 7-разрядный, восьмая позиция отводится для записи бита четности. Это обеспечивает представление 128 символов, включая все латинские буквы, цифры, знаки основных математических операций и знаки пунктуации. Позже появилась европейская модификация ASC II, называемая Latin 1(стандарт ISO 8859-1). В ней «полезно» используются все 8 разрядов. Дополнительные комбинации (коды 128-255) в новом варианте отводятся для представления специфических букв алфавитов западноевропейских языков, символов псевдографики, некоторых букв греческого алфавита, а также ряда математических и финансовых символов. Именно эта кодовая таблица считается мировым стандартом де-факто, который применяется с различными модификациями во всех странах. В зависимости от использования кодов 128-255 различают несколько вариантов стандарта ISO 8859 (от первого до шестнадцатого). Так, например, стандарт ISO 8859-2 характеризует языки стран центральной и восточной Европы.

Хотя код ASCII достаточно удобен, он все же слишком тесен и не вмещает множества необходимых символов. По этой причине в 1993 году консорциумом нескольких компаний был разработан 16-битовый стандарт ISO 10646, определяющий универсальный набор символов UCS (Universal CharacterSet). Новый код, известный под названием Unicode, позволяет задать до 65 536 символов, то есть дает возможность одновременно представить символы всех основных «живых» и «мертвых» языков. Для букв русского языка выделены коды 1040-1093.

В «естественном» варианте кодировки Unicode, известном как UCS-2, каждый символ описывается двумя последовательными байтами m и n, так что номеру символа соответствует численное значение 256 x m + n. Таким образом, кодовый номер представлен 16-разрядным двоичным числом.

2.2. Форматы данных. Формы представления чисел в компьютере

Машинные команды оперируют данными, которые принято называть *операндами.* К наиболее общим (базовым) типам операндов можно отнести: адреса, числа, символы и логические данные. Помимо них компьютер обеспечивает обработку и более сложных информационных единиц: графических изображений, аудио-, видео- и анимационной информации. Такая информация является производной от базовых типов данных и хранится в виде файлов на внешних запоминающих устройствах. Для каждого типа данных в ВМ предусмотрены определенные форматы.

Среди цифровых данных можно выделить две группы:

- целые типы, используемые для представления целых чисел;
- вещественные типы для представления рациональных чисел.

В рамках первой группы имеется несколько форматов представления численной информации, зависящих от ее характера. Для представления вещественных чисел используется форма с плавающей запятой.

Числа в форме с фиксированной запятой

Представление числа X в форме с фиксированной запятой (ФЗ), которую иногда называют также естественной формой, включает в себя знак числа и его модуль в q-ичном коде. Здесь q – основание системы счисления или база. Для современных ВМ характерна двоичная система (q = 2). Знак положительного числа кодируется двоичной цифрой 0, а знак отрицательного числа – цифрой 1.

Числам с ФЗ соответствует запись вида $X = \pm a_{n-1}\ldots a_1 a_0 a_{-1} a_{-2}\ldots a_{-r}$. Отрицательные числа обычно представляются в дополнительном коде. Разряд кода числа, в котором размещается знак, называется знаковым разрядом кода. Разряды, где располагаются значащие цифры числа, называются цифровыми разрядами кода. Знаковый разряд размещается левее старшего цифрового разряда. Положение запятой одинаково для всех чисел и в процессе решения задач не меняется. Хотя запятая и фиксируется, в коде числа она никак не выделяется, а только подразумевается. В общем случае разрядная сетка ВМ для размещения чисел в форме с ФЗ имеет вид, представленный на рис. 2, где n разрядов используются для записи целой части числа и r разрядов – для дробной части.

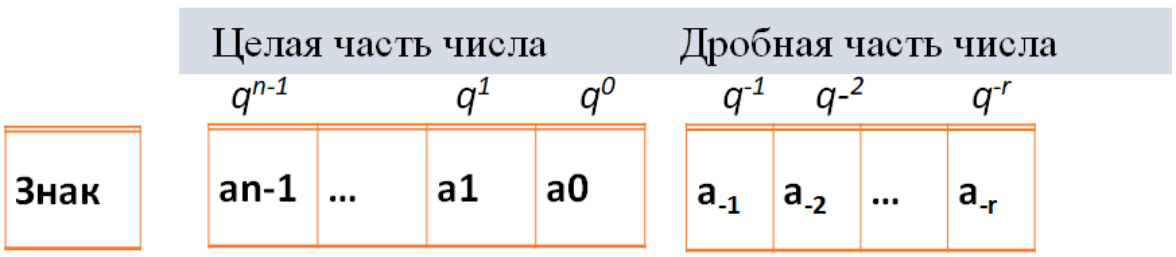

Рисунок 2 – Формат представления чисел с фиксированной запятой

При фиксации запятой перед старшим цифровым разрядом могут быть представлены только правильные дроби. Для ненулевых чисел возможны два варианта представления (нулевому значению соответствуют нули во всех разрядах): знаковое и беззнаковое. Фиксация запятой перед старшим разрядом встречалась в ряде машин второго поколения, но в настоящее время практически отжила свое.

При фиксации запятой после младшего разряда представимы лишь целые числа. Это наиболее распространенный способ, поэтому в дальнейшем понятие ФЗ будет связываться исключительно с целыми числами (рис. 3).

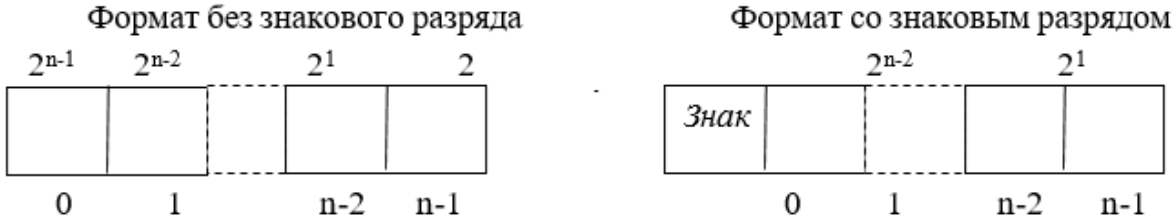

Рисунок 3 – Представление целых чисел в формате ФЗ

Представление чисел в формате ФЗ упрощает аппаратурную реализацию процессора сокращает время выполнения машинных операций, однако при решении задач необходимо постоянно следить за тем, чтобы все исходные данные, промежуточные и окончательные результаты не выходили за допустимый диапазон формата, иначе возможно переполнение разрядной сетки и результат вычислений будет неверным.

Числа с плавающей запятой

От недостатков ФЗ в значительной степени свободна форма представления чисел с плавающей запятой (ПЗ), известная также под названиями нормальной или полулогарифмической формы. В данном варианте каждое число разбивается на две группы цифр. Первая группа цифр называется мантиссой, вторая – порядком. Число представляется в виде произведения $X = \pm mq^{\pm p}$, где m – мантисса числа X, p – порядок числа, q – основание системы счисления.

Для представления числа в форме с ПЗ требуется задать знаки мантиссы и порядка, их модули в q-ичном коде, а также основание системы счисления. Нормальная форма неоднозначна, так как взаимное изменение m и p приводит к «плаванию» запятой, чем и обусловлено название этой формы (рис. 4).

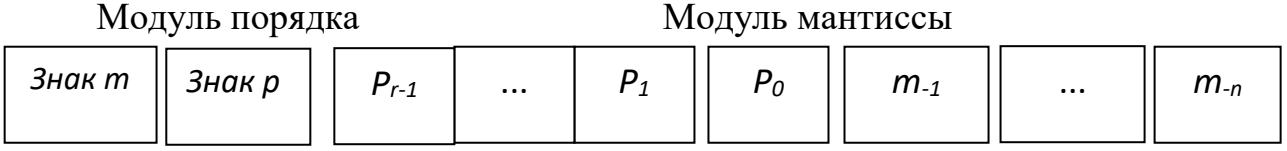

Рисунок 4 – Форма представления чисел с плавающей запятой

2.3. Форматы команд

Типовая команда, в общем случае, должна указывать:

- подлежащую выполнению операцию;
- адреса исходных данных (операндов), над которыми выполняется операция;
- адрес, по которому должен быть помещен результат операции.

В соответствии с этим команда состоит из двух частей: операционной и адресной (рис. 5).

Операционная часть, КОП	Адресная часть, А

Рисунок 5 – Формат команды

Формат команды определяет ее структуру, то есть количество двоичных разрядов, отводимых под всю команду, а также количество и расположение отдельных полей команды. Полем называется совокупность двоичных разрядов, кодирующих составную часть команды. При создании ВМ выбор формата команды влияет на многие характеристики будущей машины. Оценивая возможные форматы, нужно учитывать следующие факторы:

- общее число различных команд;
- общую длину команды;
- тип полей команды (фиксированной или переменной длины) и их длина;
- простоту декодирования;
- адресуемость и способы адресации;
- стоимость оборудования для декодирования и исполнения команд.

Способы адресации операндов в команде. Вопрос о том, каким образом в адресном поле команды может быть указано местоположение операндов, считается одним из центральных при разработке архитектуры компьютера. С точки зрения сокращения аппаратурных затрат очевидно стремление разработчиков уменьшить длину адресного поля при сохранении возможностей доступа ко всему адресному пространству. С другой стороны, способ задания адресов должен способствовать максимальному сближению конструктов языков программирования высокого уровня и машинных команд. Все это привело к тому, что в архитектуре системы команд любой ВМ предусмотрены различные способы адресации операндов.

Приступая к рассмотрению способов адресации, вначале определим понятия «исполнительный» и «адресный код».

Исполнительным адресом операнда ($А_{исп}$) называется двоичный код номера ячейки памяти, служащей источником или приемником операнда. Этот код подается на адресные входы запоминающего устройства (ЗУ), и по нему происходит фактическое обращение к указанной ячейке. Если операнд хранится не в основной памяти, а в регистре процессора, его исполнительным адресом будет номер регистра.

Адресный код команды ($А_к$) – это двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

В современных компьютерах исполнительный адрес и адресный код, как правило, не совпадают, и для доступа к данным требуется соответствующее

преобразование. Способ адресации – это способ формирования исполнительного адреса операнда по адресному коду команды. Способ адресации существенно влияет на параметры процесса обработки информации. Одни способы позволяют увеличить емкость адресуемой памяти без удлинения команды, но снижают скорость выполнения операции, другие – ускоряют операции над массивами данных, третьи – упрощают работу с подпрограммами и т. д. В сегодняшних ВМ обычно имеется возможность приложения нескольких различных способов адресации операндов к одной и той же операции.

Чтобы устройство управления вычислительной машины могло определить, какой именно способ адресации принят в данной команде, в разных ВМ используются различные приемы. Часто разным способам адресации соответствуют и разные коды операции. Другой подход – это добавление в состав команды специального поля способа адресации, содержимое которого определяет, какой из способов адресации должен быть применен. Иногда в команде имеется нескольких полей – по одному на каждый адрес. Отметим, что возможен также вариант, когда в команде вообще отсутствует адресная информация, то есть имеет место неявная адресация. При неявной адресации адресного поля либо просто нет, либо оно содержит не все необходимые адреса – отсутствующий адрес подразумевается кодом операции. Так, при исключении из команды адреса результата подразумевается, что результат помещается на место второго операнда. Неявная адресация применяется достаточно широко, поскольку позволяет сократить длину команды. Рассмотрим более подробно некоторые из способов адресации операндов.

Непосредственная адресация

При непосредственной адресации (НА) в адресном поле команды вместо адреса содержится непосредственно сам операнд. Этот способ может применяться при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры.

Когда операндом является число, оно обычно представляется в дополнительном коде. При записи в регистр, имеющий разрядность, превышающую длину непосредственного операнда, операнд размещается в младшей части регистра, а оставшиеся свободными позиции заполняются значением знакового бита операнда.

Помимо того, что в адресном поле могут быть указаны только константы, еще одним недостатком данного способа адресации является то, что размер непосредственного операнда ограничен длиной адресного поля команды, которое в большинстве случаев меньше длины машинного слова.

Регистровая адресация

Регистровая адресация (РА). Адресное поле инструкции указывает не на ячейку памяти, а на регистр процессора. Идентификатор регистра в дальнейшем будем обозначать буквой R. Обычно размер адресного поля в данном случае составляет три или четыре бита, что позволяет указать соответственно на один из 8 или 16 регистров общего назначения (РОН).

Двумя основными преимуществами регистровой адресации (рис. 6) являются: короткое адресное поле в команде и исключение обращений к памяти. Малое число РОН позволяет сократить длину адресного поля команды.

Рисунок 6 – Регистровая адресация

Прямая адресация

При прямой адресации адресный код прямо указывает номер ячейки памяти, к которой производится обращение (рис. 7), т. е. адресный код совпадает с исполнительным адресом.

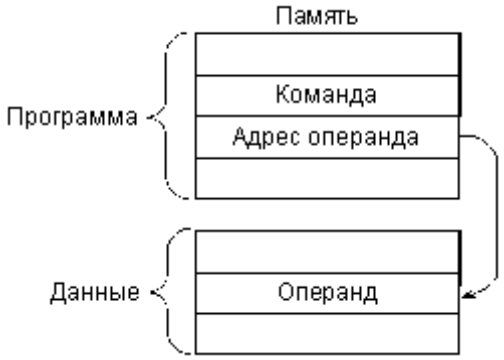

Рисунок 7 – Прямая адресация

При всей простоте использования этот способ имеет существенный недостаток – ограниченный размер адресного пространства.

Косвенная адресация

Одним из путей преодоления проблем свойственных прямой адресации является косвенная адресация. При этом способе содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять.

При косвенной адресации в адресном поле команды указывается адрес ячейки (или номер РОН), в свою очередь, содержащей полноразрядный адрес операнда (рис. 8). В качестве недостатка можно отметить необходимость в двукратном к памяти.

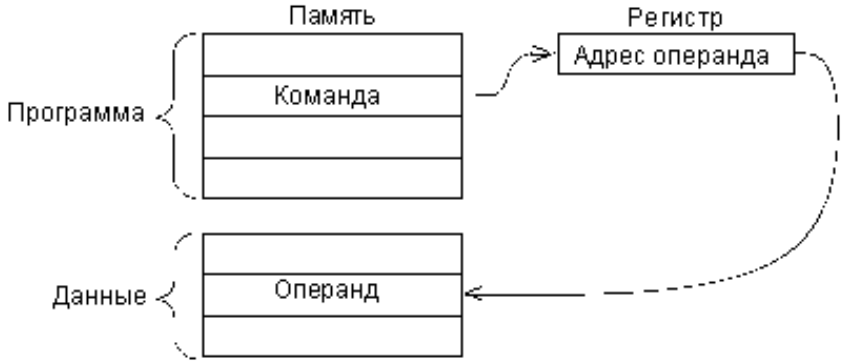

Рисунок 8 – Косвенная адресация

Кроме вышеперечисленных способов адресации существуют также индексная адресация, относительная индексная адресация, блочная адресация, стековая адресация и др.

3. ОБЩАЯ ОРГАНИЗАЦИЯ КОМПЬЮТЕРНЫХ СИСТЕМ

3.1. Структура процессора

На рис. 9 показана структура обычного компьютера с шинной организацией. Процессор основным узлом. Его задача – выполнять программы, находящиеся в основной памяти. Для этого он вызывает команды из памяти, определяет их тип, а затем выполняет одну за другой. Компоненты соединены шиной, представляющей собой набор параллельно связанных проводов для передачи адресов, данных и управляющих сигналов. Шины могут быть внешними (связывающими процессор с памятью и устройствами ввода-вывода) и внутренними. Современный компьютер использует несколько шин. Процессор состоит им нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. Арифметико-логическое устройство выполняет арифметические операции (например, сложение) и логические операции (например, логическое И).

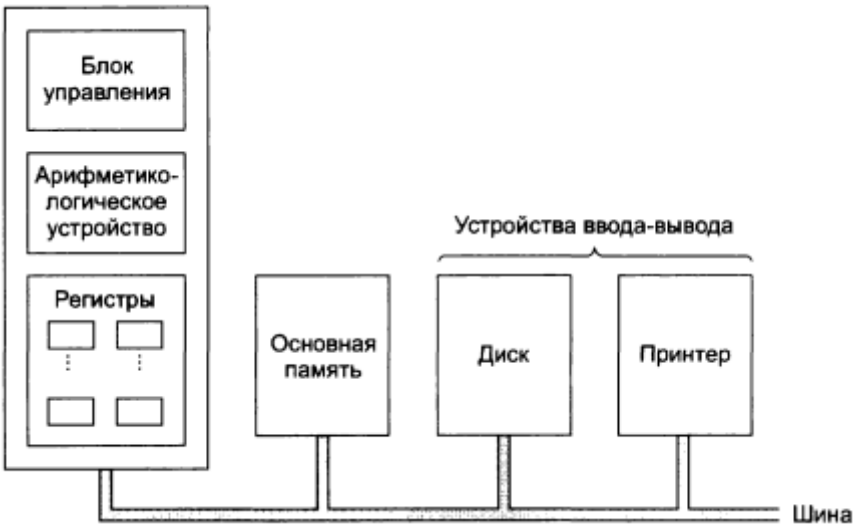

Рисунок 9 – Схема простейшего компьютера

Процессор состоит им нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. Арифметико-логическое устройство выполняет арифметические операции (например, сложение) и логические операции (например, логическое И).

Внутри центрального процессора находится быстрая память небольшого объема для хранения промежуточных результатов и некоторых команд управления. Она состоит из нескольких регистров, каждый из которых выполняет определенную функцию. Обычно размер всех регистров одинаков. Каждый регистр содержит одно число в диапазоне, верхняя граница которого зависит от размера регистра. Операции чтения и записи с регистрами выполняются очень быстро, так как они находятся внутри центрального процессора.

В состав процессора входят также счетчик команд, который указывает, какую команду нужно выполнять следующей (название «счетчик команд» выбрано неудачно, поскольку он ничего не считает, но этот термин употребляется повсеместно), и регистр команд, в котором находится выполняемая в данный момент испила. У большинства компьютеров имеются и другие регистры, одни из них являются многофункциональными, другие служат лишь какие-либо конкретным целям. Третьи используются операционной системой для управления компьютером.

3.2. Тракт данных процессора

Тракт данных состоит из регистров (обычно от 1 до 32), арифметико-логического устройства (АЛУ) и нескольких соединительных шип.

Содержимое регистров поступает во входные регистры АЛУ, которые на рис. 10. обозначены буквами А и В. В них находятся входные данные АЛУ, пока АЛУ производит вычисления. Тракт данных – важная составная часть всех компьютеров.

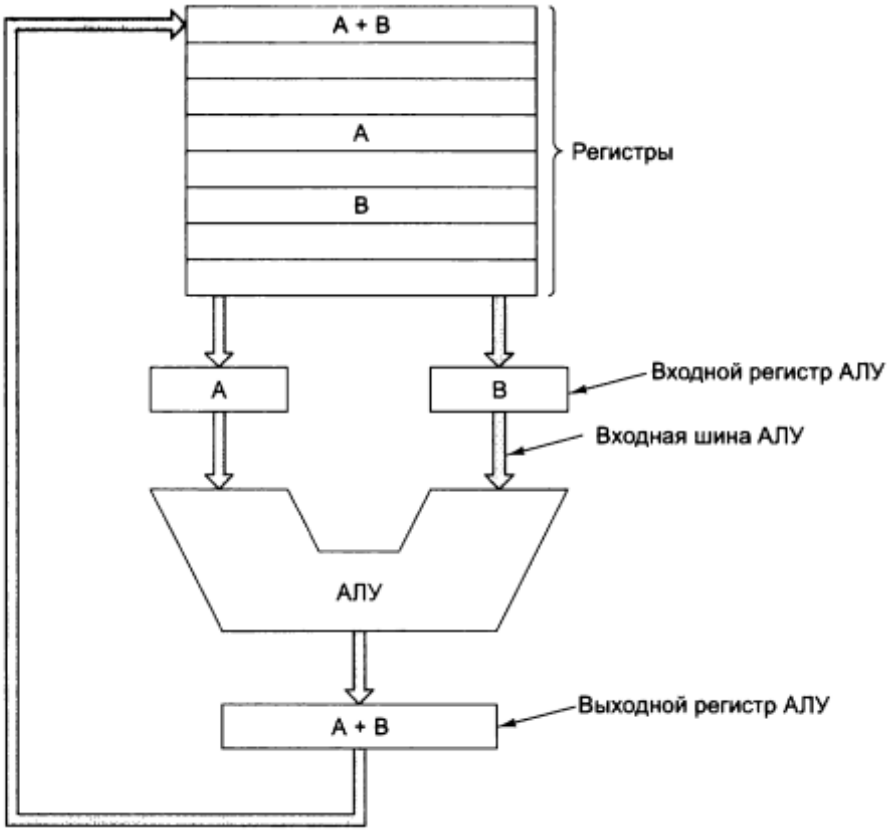

Рисунок 10 – Тракт данных

АЛУ выполняет сложение, вычитание и другие простые операции над данными и помещает результат в выходной регистр. Содержимое этого регистра может записываться обратно в один из регистров или храниться в

памяти. На рис. 10 представлена операция сложения, но АЛУ может выполнять и другие операции.

В самом простейшем случае большинство команд можно разделить на две группы: команды типа регистр-память и типа регистр-регистр. Команды первого типа вызывают слова из памяти, помещают их в регистры, где они используются в качестве входных данных АЛУ. Словом, может быть целое число. Другие команды этого типа помещают регистры обратно в память.

Команды второго типа вызывают два операнда из регистров, помещают их во входные регистры АЛУ, выполняют над ними какую-нибудь арифметическую или логическую операцию и переносят результат обратно в один из регистров.

Этот процесс называется циклом тракта данных. В какой-то степени он определяет, что может делать компьютер. Современные компьютеры оснащаются несколькими АЛУ, работающими параллельно и специализирующимися на разных функциях. Чем быстрее происходит цикл тракта данных, тем быстрее компьютер работает.

3.3. Системы RISC и CISC

3.3.1. История разработки RISC архитектур

В конце 1970-х годов проводилось много экспериментов с очень сложными командами, появление которых стало возможным благодаря интерпретации.

Примечание. Исполнение программы, написанной на языке Я1, подразумевает замену каждой команды эквивалентным набором на языке машинных команд. В этом случае компьютер исполняет новую программу, написанную на языке машинных команд, вместо старой программы, написанной на Я1. Эта технология называется трансляцией.

Интерпретация заключается в создании на языке машинных команд программы, использующей в качестве входных данных программы, написанные на языке Я1. При этом каждая команда языка Я1 обрабатывается поочередно, после чего сразу исполняется эквивалентный ей набор машинных команд. Эта технология не требует составления новой программы на языке машинных команд. Она называется интерпретацией, а программа, которая осуществляет интерпретацию, называется интерпретатором.

Между трансляцией и интерпретацией много общего. В обоих случаях компьютер в конечном итоге исполняет набор команд на языке машинных, эквивалентных командам Я1. Отличие лишь в том, что при трансляции вся программа Я1 переделывается в программу понятную для компьютера, а программа Я1 отбрасывается, а новая программа на языке машинных команд загружается в память компьютера и затем исполняется. Во время выполнения сгенерированная программа управляет работой компьютера.

При интерпретации каждая команда программы на Я1 перекодируется в язык машинных команд и сразу же исполняется. Транслированная программа при этом не создается. Работой компьютера управляет интерпретатор, для которого программа на Я1 есть не что иное, как входные данные.

Разработчики пытались уменьшить «семантический разрыв» между тем, что компьютеры способны делать, и тем, что требуют языки высокого уровня. Едва ли кто-нибудь тогда думал о разработке более простых машин, так же как

сейчас мало кто (к сожалению) занимается разработкой менее мощных электронных таблиц, сетей, веб-серверов и т. д.

В 1980 году группа разработчиков в университете Беркли во главе с Дэвидом Паттерсоном и Карло Секвином начала разработку ориентированных на интерпретацию процессоров VLSI. Для обозначения этого понятия они придумали термин RISC, а новый процессор назвали RISC I. Немного позже, в 1981 году, Джон Хеннеси разработал и выпустил другую микросхему, которую он назвал MIPS. Эти две микросхемы развились в коммерчески важные продукты SPARC и MIPS соответственно.

Новые процессоры существенно отличались от коммерческих процессоров того времени. Поскольку они были несовместимы с существующей продукцией, разработчики вправе были включать туда новые наборы команд, которые могли повысить общую производительность системы. Первоначально основное внимание уделялось простым командам, которые могли быстро выполняться.

Однако вскоре разработчики осознали, что ключом к высокой производительности компьютера является разработка команд, которые можно быстро запускать, то есть не так важно, как долго выполняется та или иная команда, важнее то, сколько команд в секунду может быть запущено.

В то время, когда разрабатывались эти простые процессоры, всеобщее внимание привлекало относительно небольшое количество команд (обычно около 50). В компьютерах производства DEC и IBM число команд составляло от 200 до 300.

По мнению создателей RISC, лучший способ разработки компьютеров – это включение туда небольшого количества простых команд, каждая из которых выполняется за один цикл, то есть производит над парой регистров какую-либо арифметическую или логическую операцию (например, сложение или операцию логического **И**) и помещает результат обратно в регистр. В качестве аргумента они утверждали, что даже если системе RISC приходится выполнять 4 или 5 программ вместо одной, которую выполняет CISC, RISC все равно выигрывает в скорости, так как RISC-команды выполняются в 10 раз быстрее (поскольку они не интерпретируются). Следует также отметить, что к этому времени быстродействие основной памяти приблизилась к быстродействию специальных командных ПЗУ, потому недостатки интерпретации были на лицо, что еще больше поднимало популярность компьютеров RISC.

Учитывая преимущества RISC в плане производительности, можно было предположить, что на рынке такие компьютеры должны доминировать над компьютерами CISC. Однако, во-первых, компьютеры RISC несовместимы с другими моделями, а многие компании вложили миллиарды долларов в программное обеспечение для продукции Intel. Во-вторых, как ни странно, компания Intel сумела воплотить те же идеи в архитектуре CISC. Процессоры Intel, начиная с процессора 486, содержат RISC-ядро, которое выполняет самые простые (и обычно самые распространенные) команды за один цикл, а по обычной технологии CISC интерпретируются более сложные команды. В

результате обычные команды выполняются быстро, а более сложные и редкие – медленно. Хотя при таком «гибридном» подходе производительность ниже, чем в архитектуре RISC, новая архитектура CISC имеет ряд преимуществ, поскольку позволяет использовать старое программное обеспечение без изменений.

3.3.2. Принципы построения RISC архитектур

Существует ряд принципов разработки, иногда называемых принципами RISC, которым по возможности стараются следовать производители универсальных процессоров.

Все команды должны выполняться непосредственно аппаратным обеспечением. То есть обычные команды выполняются напрямую, без интерпретации микрокомандами. Устранение уровня интерпретации повышает скорость выполнения большинства команд. В компьютерах типа CISC более сложные команды могут разбиваться на несколько шагов, которые затем выполняются как последовательность микрокоманд. Эта дополнительная операция снижает быстродействие машины, но может использоваться для редко применяемых команд.

Компьютер должен запускать как можно больше команд в секунду. В современных компьютерах используется много различных способов повышения производительности, главный из которых – запуск как можно большего количества команд в секунду. В конце концов, если процессор сможет запустить 500 млн команд в секунду, то его производительность составляет 500 MIPS, сколько бы времени ни занимало выполнение этих команд (MIPS – сокращение от Millions of Instructions Per Second миллионов команд в секунду). Этот принцип предполагает, что параллелизм должен стать важным фактором повышения производительности, поскольку запустить на выполнение большое количество команд за короткий промежуток времени можно только в том случае, если есть возможность одновременного выполнения нескольких команд.

Хотя команды любой программы всегда располагаются в памяти в определенном порядке, компьютер изменит порядок их запуска (так как необходимые ресурсы памяти могут быть заняты) и (или) завершения. Конечно, если команда 1 устанавливает значение в регистре, а команда 2 использует этот регистр, нужно синхронизировать операции так, чтобы команда 2 не считала значение из регистра раньше, чем оно там окажется. Чтобы не допускать подобных ошибок, необходимо хранить в памяти большое количество дополнительной информации, но благодаря возможности выполнять несколько команд одновременно производительность все равно оказывается выше.

Команды должны легко декодироваться. Предел количества запускаемых в секунду команд зависит от темпа декодирования отдельных команд. Декодирование команд позволяет определить, какие ресурсы им необходимы и какие действия нужно выполнить. Все, что способствует упрощению того процесса, полезно. Например, можно использовать единообразные команды с фиксированной длиной и с небольшим количеством

полей. Чем меньше разных форматов команд, тем лучше.

Регистров должно быть много. Поскольку доступ к памяти происходит относительно медленно, в компьютере должно быть много регистров (по крайней мере 32). Если слово было однажды загружено из памяти, при наличии большого числа регистров оно может содержаться в регистре до тех пор, пока не потребуется. Возвращение слова из регистра в память и новая загрузка этого же слова в регистр нежелательны. Лучший способ избежать излишних перемещений – наличие достаточного количества регистров.

3.4. Параллелизм на уровне команд

Один из способов заставить повысить производительность процессоров – увеличение тактовой частоты, однако при этом существуют некоторые технические ограничения. Поэтому большинство проектировщиков для повышения производительности при данной тактовой частоте процессора используют параллелизм (выполнение двух или более операций одновременно).

Существует две основные формы параллелизма: параллелизм на уровне команд и параллелизм на уровне процессоров. В первом случае параллелизм реализуется зачет запуска большого количества команд каждую секунду. Во втором случае над одним заданием работают одновременно несколько процессоров. Каждый подход имеет свои преимущества.

3.4.1. Конвейеризация команд

Известно, что главным препятствием высокой скорости выполнения команд является необходимость их загрузки из памяти. Для разрешения этой проблемы можно вызывать команды из памяти заранее и хранить в специальном наборе регистров. Эта идея использовалась еще в 1959 году при разработке компьютера Stretch компании IBM, а набор регистров был назван буфером выборки с упреждением. Таким образом, когда требовалась определенная команда, она вызывалась прямо из буфера, а обращения к памяти не происходило.

В действительности при выборке с упреждением команда обрабатывается за два шага: сначала происходит выборка команды, а затем ее выполнение.

Дальнейшим развитием этой стратегии стала концепция конвейера. При использовании конвейера команда обрабатывается уже не за два, а за большее количество шагов, каждый из которых реализуется определенным аппаратным компонентом, причем все эти компоненты могут работать параллельно.

На рис. 11*а* изображен конвейер из 5 блоков, которые называются ступенями. Первая ступень (блок С1) вызывает команду из памяти и помещает ее в буфер, где она хранится до тех пор, пока не потребуется. Вторая ступень (блок С2) декодирует эту команду, определяя ее тип и тип ее операндов.

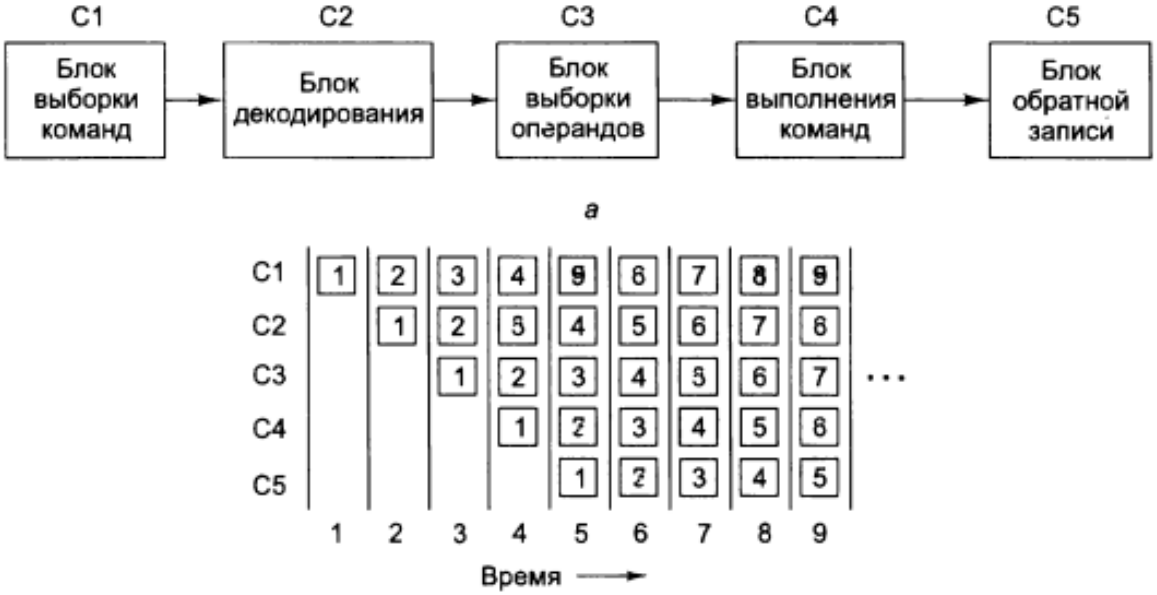

Рисунок 11 – Пятиступенчатый конвейер

Третья ступень (блок С3) определяет местонахождение операндов и вызывает их из регистров или из памяти. Четвертая ступень (блок С4) выполняет команду, обычно проводя операнды через тракт данных (см. рис. 11). И наконец, блок С5 записывает результат обратно в нужный регистр.

На рис.11 мы видим, как действует конвейер во времени. Во время цикла 1 блок С1 обрабатывает команду 1, вызывая ее из памяти. Во время цикла 2 блок С2 декодирует команду 1, в то время как блок С1 вызывает из памяти команду 2. Во время цикла 3 блок С3 вызывает операнды для команды 1, блок С2 декодирует команду 2, а блок С1 вызывает команду 3. Во время цикла 4 блок С4 команду 1, С3 вызывает операнды для команды 2, С2 декодирует команду 3, а С1 вызывает команду 4. Наконец, во время цикла 5 блок С5 записывает результат выполнения команды 1 обратно в регистр, тогда как другие ступени конвейера обрабатывают следующие команды.

Предположим, что время цикла у этого компьютера – 2 нс. Тогда, для того чтобы одна команда прошла через конвейер, требуется 10 нс. На первый взгляд может показаться, что такой компьютер будет выполнять 100 млн команд в секунду, в действительности же скорость его работы гораздо выше. В течение каждого цикла (2 нс) завершается выполнение одной новой команды, поэтому машина выполняет не 100, а 500 млн команд в секунду!

Конвейеры позволяют добиться компромисса между временем запаздывания (время выполнения одной команды) и пропускной способностью процессора (количество команд, выполняемых процессором в секунду).

3.4.2. Суперскалярные архитектуры

Одна из возможных схем процессора с двумя конвейерами показана на рис. 12.

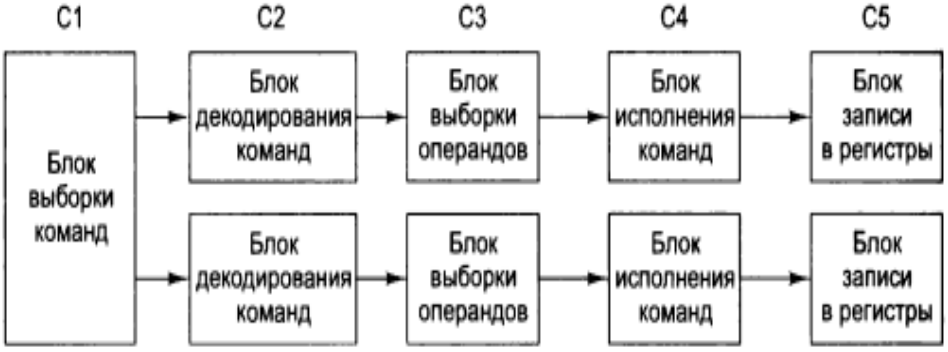

Рисунок 12 – Процессор с двумя конвейерами

В ее основе лежит конвейер. Здесь общий блок выборки команд вызывает из памяти сразу по две команды и помещает каждую из них в один из конвейеров. Каждый конвейер содержит АЛУ для параллельных операций. Чтобы выполняться параллельно, две команды не должны конфликтовать из-за ресурсов (например, регистров) и ни одна из них не должна зависеть от результата выполнения другой. Как и в случае с одним конвейером, либо компилятор должен гарантировать отсутствие нештатных ситуаций (когда, например, аппаратура не обеспечивает проверку команд на несовместимость и при обработке таких команд выдает некорректный результат), либо конфликты должны выявляться и устраняться дополнительным оборудованием непосредственно в ходе выполнения команд.

3.5. Параллелизм на уровне процессоров

Спрос на компьютеры, работающие все с более и более высокой скоростью, не прекращается. Быстродействие процессоров растет, но у них постоянно возникают проблемы со скоростью передачи информации, поскольку скорость распространения электромагнитных волн в медных проводах и света в оптико-волоконных кабелях по-прежнему недостаточно высока. Кроме того, чем быстрее работает процессор, тем сильнее он нагревается, поэтому возникает задача защиты его от перегрева.

Параллелизм на уровне команд в определенной степени помогает, но конвейеры и суперскалярная архитектура обычно повышают скорость работы всего лишь в 5-10 раз. Чтобы увеличить производительность в 50, 100 и более раз, нужно создавать компьютеры с несколькими процессорами.

3.5.1. Матричные компьютеры

SIMD – процессор (Single Instruction – stream Multiple Data-stream – один поток команд с несколькими потоками данных) состоит из большого числа сходных процессоров, которые выполняют одну и ту же последовательность команд применительно к разным наборам данных.

Современные графические процессоры (GPU) широко используют SIMD обработку для обеспечения высокой вычислительной мощности при относительно небольшом количестве транзисторов. Обработка графики отлично подходит для SIMD-процессоров, потому что большинство алгоритмов имеет четкую структуру с повторением операций для пикселов, вершин, текстур и ребер. На рис. 13 изображен SIMD-процессор ядра графического процессора.

Он содержит до 16 потоковых мультипроцессоров (SM, Stream Multiprocessor) каждый из которых содержит 32 процессора SIMD. За каждый цикл планировщик выбирает два потока для выполнения на процессоре SIMD. Затем следующая команда каждого потока выполняется на процессорах SIMD (до 16, они при отсутствии достаточного параллелизма данных используется меньшее количество процессоров). Если каждый поток способен выполнить 16 операций за цикл, полностью загруженное ядро графического процессора с 32 SM будет выполнять целых 512 операций за цикл. Это весьма впечатляющее достижение, если учесть, что четырехъядерный процессор общего назначения того же размера с трудом достигнет 1/32 такой вычислительной мощи.

С точки зрения программиста векторный процессор очень похож на SIMD-процессор. Он также чрезвычайно эффективен при выполнении последовательности операций над парами элементов данных. Однако в отличие от SIMD-процессора, все операции сложения выполняются в одном блоке суммирования, который имеет конвейерную структуру.

Оба типа процессоров работают с массивами данных. Оба они выполняют одни и те же команды, которые, например, попарно складывают элементы двух векторов. Однако если у SIMD-процессора столько же суммирующих устройств, сколько элементов в массиве, векторный процессор содержит векторный регистр, состоящий из набора традиционных регистров. Эти регистры загружаются из памяти единственной командой, которая фактически делает это последовательно.

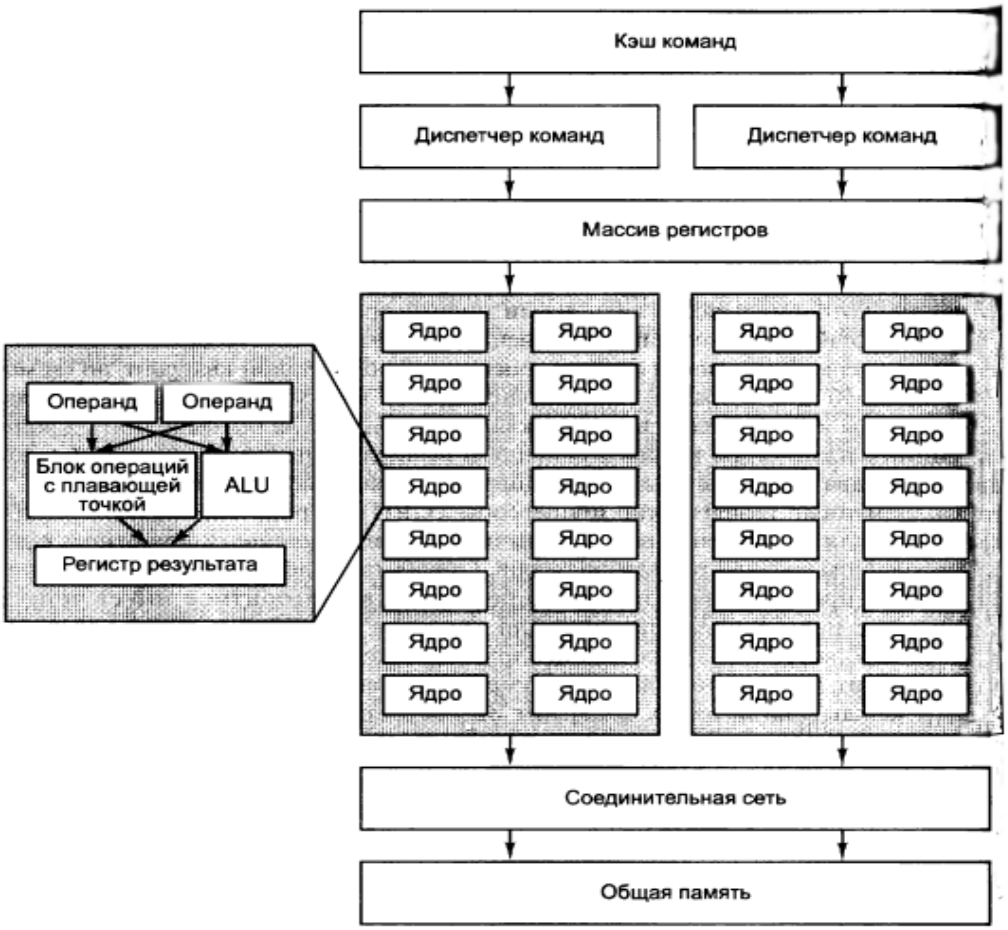

Рисунок 13 – SIMD-процессор ядра графического процессора

Команда сложения попарно складывает элементы двух таких векторов, загружая их из двух векторных регистров в суммирующее устройство с конвейерной структурой. В результате из суммирующего устройства выходит другой вектор, который либо помещается в векторный регистр, либо сразу используется в качестве операнда при выполнении другой операции с векторами. Команды SSE (Streaming SIMD Extension) в архитектуре Intel Core используют эту модель расширения для ускорения вычислений с высокой степенью упорядоченности, например, обработки мультимедийных и научных данных.

3.5.2. Мультипроцессоры

Элементы процессора, распараллеленного по данным, связаны между собой, поскольку их работу контролирует единый блок управления. Система из нескольких параллельных процессоров, имеющих общую память, называется мультипроцессором. Поскольку каждый процессор может записывать информацию в любую часть памяти и считывать информацию из любой части памяти, чтобы не допустить каких-либо нестыковок, их работа должна согласовываться программным обеспечением. В ситуации, когда два или несколько процессоров имеют возможность тесного взаимодействия, а именно так происходит в случае с мультипроцессорами, эти процессоры называют

сильно связанными.

Возможны разные способы этой реализации. Самый простой соединение единственной шиной нескольких процессоров и общей памяти. Схема такого мультипроцессора показана на рис. 14*а*.

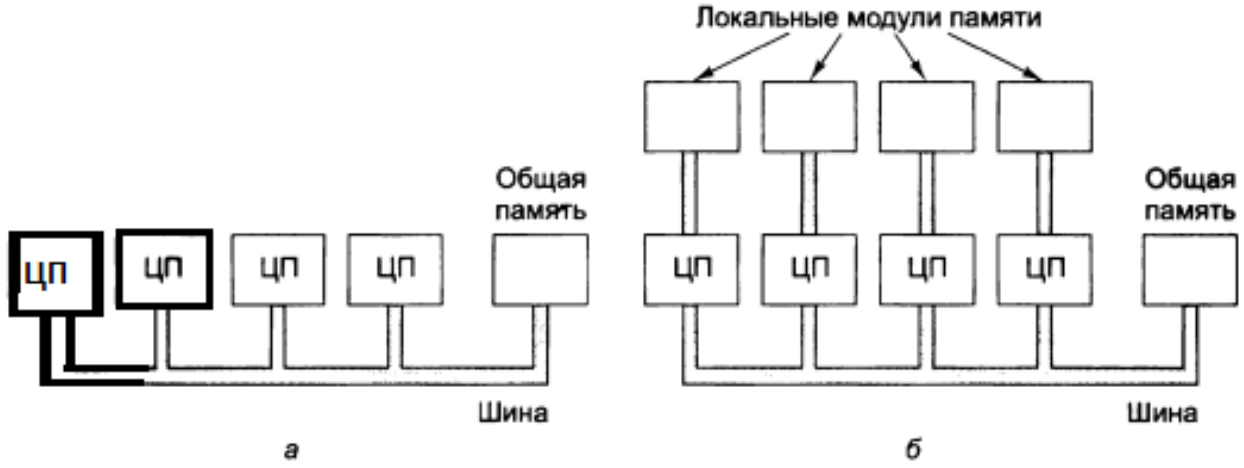

Рисунок 14 – Мультипроцессор с общей памятью (*а*); мультипроцессор с собственной локальной памятью для каждого процессора (*б*)

Естественно, при наличии большого числа быстродействующих процессоров, которые постоянно пытаются получить доступ к памяти через одну и ту же шину, могут возникать конфликты. Чтобы разрешить эту проблему и повысить производительность компьютера, разработаны различные схемы. Одна из них изображена на рис. 14*б*. В таком компьютере каждый процессор имеет собственную локальную память, недоступную для других процессоров. Эта память используется для тех программ и данных, которые не нужно разделять между несколькими процессора. При обращении к локальной памяти основная шина не используется, и, таким образом, объем передаваемой по ней информации становится меньше. Возможны и другие варианты решения проблемы (например, кэширование).

Мультипроцессоры имеют преимущество перед другими видами параллельных компьютеров, поскольку с единой общей памятью очень легко работать.

4. ФУНКЦИИ, СТРУКТУРА И ТИПЫ УСТРОЙСТВ УПРАВЛЕНИЯ

4.1. Функции устройства управления

Устройство управления (УУ) компьютера реализует функции управления ходом вычислительного процесса, обеспечивая автоматическое выполнение команд программы. Процесс выполнения программы в компьютере представляет собой последовательность машинных циклов. Для простоты примем, что компьютер обеспечивает одноадресную систему команд. При этом, в частности, полагается, что до начала выполнения двухоперандной арифметической команды второй операнд уже находится в процессоре.

- Первым этапом в машинном цикле является выборка команды из памяти.
- За выборкой команды следует этап декодирования ее операционной части (кода операции). Формирование адреса следующей команды.
- Третий этап – этап формирования исполнительного адреса операнда или адреса перехода. Функция имеет столько модификаций, сколько способов адресации предусмотрено в системе команд компьютера.
- На четвертом этапе реализуется функция выборки операнда из памяти по исполнительному адресу, сформированному на предыдущем этапе.
- На последнем этапе машинного цикла действия задаются функцией исполнения операции. Очевидно, что количество модификаций равно количеству операций, имеющихся в системе команд компьютера.

4.2. Структура устройства управления

Как уже отмечалось ранее, процесс функционирования компьютера состоит из последовательности элементарных действий в ее узлах. Такие элементарные преобразования информации, выполняемые в течение одного такта сигналов синхронизации, называются микрооперациями (МО). Совокупность сигналов управления, вызывающих одновременно выполняемые микрооперации, образует микрокоманду (МК). В свою очередь, последовательность микрокоманд, определяющую содержание и порядок реализации машинного цикла, принято называть микропрограммой. Сигналы управления вырабатываются устройством управления, а точнее одним из его узлов – микропрограммным автоматом (МПА). Название отражает то, что МПА определяет микропрограмму как последовательность выполнения микроопераций. Микропрограммы реализации перечисленных ранее целевых функций инициируются задающим оборудованием, которое вырабатывает требуемую последовательность сигналов управления и входит в состав управляющей части УУ. Выполняются микропрограммы исполнительным оборудованием, входящим в состав основной памяти и операционного устройства. В обобщенной структуре УУ (рис. 15) можно выделить две части: управляющую и адресную. Управляющая часть УУ предназначена для координирования работы операционного блока компьютера, адресной части

устройства управления, основной памяти и других узлов компьютера. Адресная часть УУ обеспечивает формирование адресов команд и исполнительных адресов операндов в основной памяти.

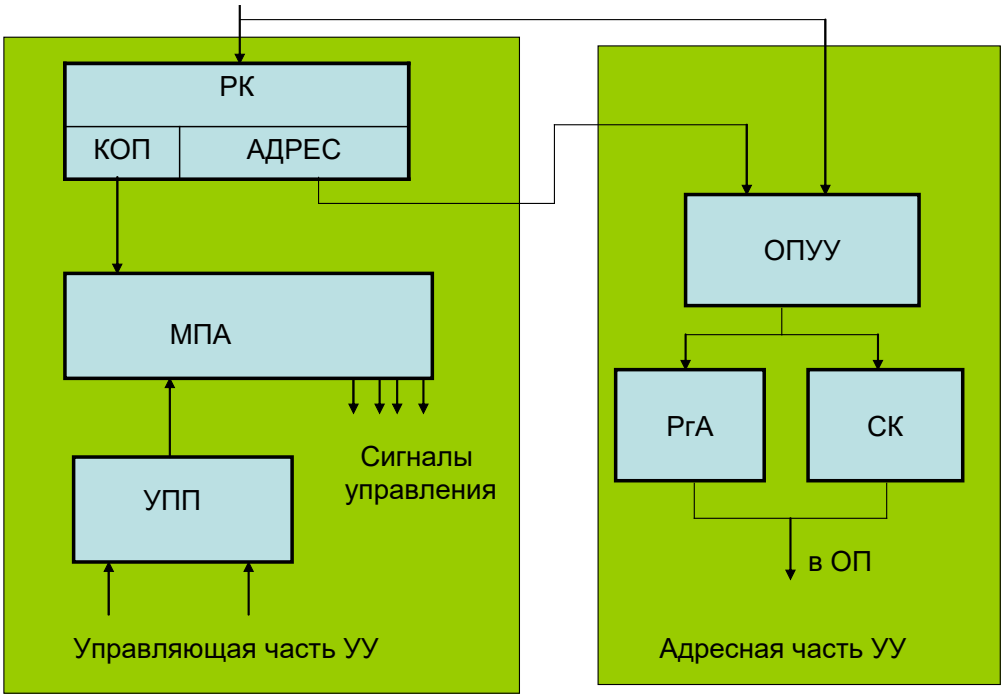

Рисунок 15 – Обобщенная структура устройства управления

В состав управляющей части УУ входят:

- регистр команды (РК), состоящий из адресной (Адрес) и операционной (КОп, СА) частей;
- микропрограммный автомат (МПА);
- узел прерываний и приоритетов УПП).

Регистр команды. РК предназначен для приема очередной команды из запоминающего устройства. Микропрограммный автомат на основании результатов расшифровки операционной части команды (Коп) вырабатывает определенную последовательность микрокоманд, вызывающих выполнение всех целевых функций УУ.

В зависимости от способа формирования микрокоманд различают *микропрограммные автоматы:*

- с жесткой или аппаратной логикой;
- с программируемой логикой.

Узел прерываний и приоритетов позволяет реагировать на различные ситуации, связанные как с выполнением рабочих программ, так и с состоянием компьютера.

Адресная часть УУ включает в себя:

- операционный узел устройства управления (ОПУУ);
- регистр адреса (РгА);
- счетчик команд (СК).

Регистр адреса используется для хранения исполнительных адресов

операндов, а *счетчик команд* – для выработки и хранения адресов команд. Содержимое РгА и СК посылается в регистр адреса основной памяти (ОП) для выборки операндов и команд соответственно.

ОПУУ, называемый иначе узлом индексной арифметики или узлом адресной арифметики, обрабатывает адресные части команд, формируя исполнительные адреса операндов, а также подготавливает адрес следующей команды при выполнении команд перехода. Состав ОПУУ может быть аналогичен составу основного операционного устройства компьютера (иногда в простейших вычислительных устройствах с целью экономии затрат на оборудование ОПУУ совмещается с основным операционным устройством).

Все множество технологий, используемых при реализации микропрограммных автоматов устройств управления, можно свести к двум категориям:

- МПА с «жесткой» логикой или аппаратурной реализацией;
- МПА с программируемой логикой.

4.3. Микропрограммный автомат с жесткой логикой

Обычно тип микропрограммного автомата (МПА), формирующего сигналы управления, определяет название всего УУ. Так, УУ с жесткой логикой управления имеет в своем составе МПА с жесткой (аппаратной) логикой. При создании такого МПА выходные сигналы управления реализуются за счет однажды соединенных между собой логических схем.

Типичная структура микропрограммного автомата с жесткой логикой управления показана на рис. 16.

Рисунок 16 – Микропрограммный автомат с жесткой логикой

Сигналы управления, по которым выполняется каждая микрооперация, должны вырабатываться в строго определенные моменты времени, поэтому все

СУ «привязаны» к импульсам синхронизации (СИ), формируемым узлом синхроимпульсов. Период СИ должен быть достаточным для того, чтобы сигналы успели распространиться по трактам данных и другим цепям. Каждый СУ ассоциируется с одним из тактовых периодов в рамках машинного цикла. Формирование сигналов, отмечающих начало очередного тактового периода, возлагается на синхронизатор. Синхронизатор содержит счетчик тактов, осуществляющий подсчет СИ. Узел синхроимпульсов после завершения очередного такта работы добавляет к содержимому счетчика тактов единицу. К выходам счетчика подключен дешифратор тактов, с которого и снимаются сигналы тактовых периодов: T_1,,.., T_n,,. В i-м состоянии счетчика тактов, то есть вовремя i-го такта, дешифратор тактов вырабатывает единичный сигнал на своем i-м выходе. При такой организации в УУ должна быть предусмотрена обратная связь, с помощью которой по окончании цикла команды счетчик тактов опять устанавливается в состояние T_1.

Дополнительным фактором, влияющим на последовательность формирования СУ, являются состояние осведомительных сигналов (флагов), отражающих ход вычислений, и сигналы с шины управления. Эта информация также поступает на вход УУ, причем каждая линия здесь рассматривается независимо от остальных.

4.4. Микропрограммный автомат с программируемой логикой

Отличительной особенностью микропрограммного автомата с программируемой логикой является хранение микрокоманд в виде кодов в специализированном запоминающем устройстве – памяти микропрограмм. Каждой команде компьютера в этом ЗУ в явной форме соответствует микропрограмма, поэтому часто устройства управления, в состав которых входит микропрограммный автомат с программируемой логикой, называют микропрограммными.

Типичная структура микропрограммного автомата представлена на рис. 17. В составе узла присутствуют: память микропрограмм (ПМП), регистр адреса микрокоманды (РАМ), регистр микрокоманды (РМК), дешифратор микрокоманд (ДшМК), преобразователь кода операции, формирователь адреса следующей микрокоманды (ФАСМ), формирователь синхроимпульсов (ФСИ). Запуск микропрограммы выполнения операции осуществляется путем передачи кода операции из РК на вход преобразователя, в котором код операции преобразуется в начальный (первый) адрес микропрограммы $А_н$. Этот адрес поступает через ФАСМ в регистр адреса микрокоманды. Выбранная по адресу $А_н$ из ПМП микрокоманда заносится в РМК.

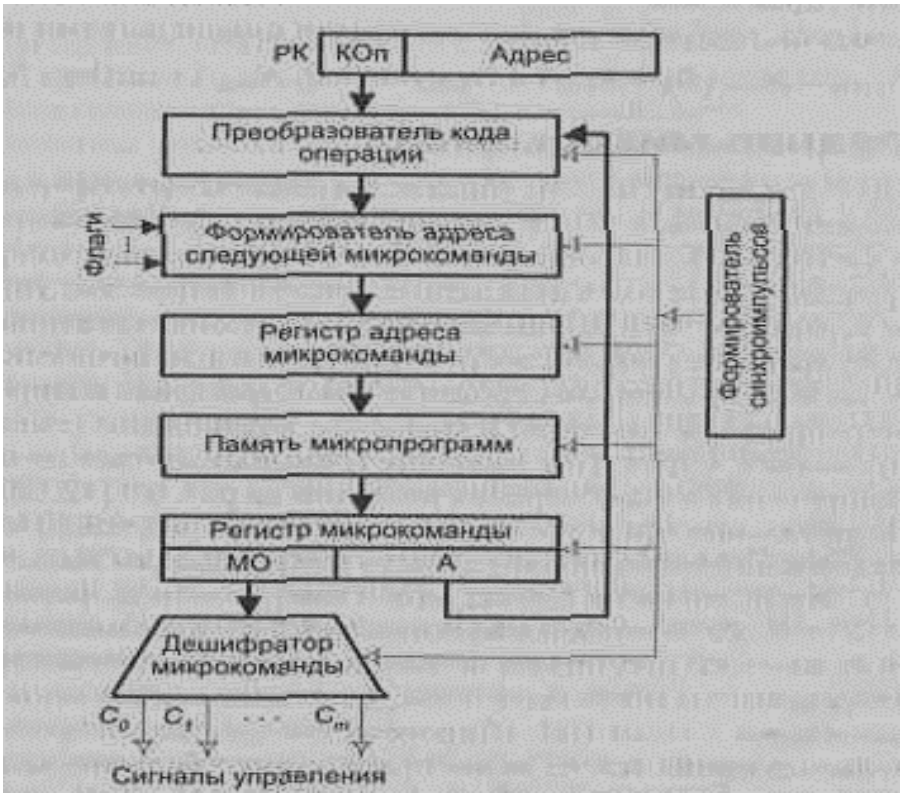

Рисунок 17 – Микропрограммный автомат с программируемой логикой

Каждая микрокоманда в общем случае содержит микрооперационную (МО) и адресную (А) части. Микрооперационная часть микрокоманды поступает на дешифратор микрокоманды, на выходе которого образуются управляющие сигналы C_i, инициирующие выполнение микроопераций в исполнительных устройствах и узлах компьютера. Адресная часть микрокоманды подается в ФАСМ, где формируется адрес следующей микрокоманды $A_{мк}$. Этот адрес может зависеть от адреса на выходе преобразователя кода операции $A_н$, адресной части текущей микрокоманды А и значений осведомительных сигналов (флагов) Х, поступающих от исполнительных устройств. Сформированный адрес микрокоманды снова записывается в РАМ, и процесс повторяется до окончания микропрограммы.

5. СИСТЕМА ПАМЯТИ КОМПЬЮТЕРА

5.1. Назначение и характеристики систем памяти

В любом компьютере, вне зависимости от его архитектуры, данные хранятся в запоминающих устройствах (ЗУ), которые, учитывая их характеристики, место расположения и способ взаимодействия с процессором, относят к внутренней или внешней памяти. Внутренняя память располагается частично на общем кристалле с процессором (регистры и кэш-память), а частично – на системной плате (основная память и, возможно, кэш-память 3-го и более высоких уровней). Медленные ЗУ большой емкости (твердотельные, магнитные и оптические диски, магнитные ленты) называют внешней памятью, поскольку к компьютеру они подключаются аналогично устройствам ввода/вывода. Основными функциями ЗУ являются прием, хранение и выдача данных в процессе работы ВМ. Процесс приема данных, в ходе которого осуществляется их занесение в ЗУ, называется записью, процесс выдачи данных – чтением или считыванием, а совместно их определяют как процессы обращения к ЗУ.

От характеристик оперативной памяти в значительной степени зависит быстродействие и надежность работы компьютера. К оперативной памяти предъявляется множество требований.

Во-первых, она должна обладать высоким быстродействием и производительностью, так как устройство должно иметь достаточную скорость, чтобы успевать обслуживать запросы от центрального процессора. Во-вторых, ОЗУ компьютера должно иметь достаточную емкость, позволяющую хранить большое количество данных. Если для выполнения некоторых задач, поставленных перед системой, оперативной памяти требуется меньше, чем имеется в системе, то неиспользованная оперативная память не принесет прироста производительности при выполнении данной работы.

Однако нехватка оперативной памяти при выполнении задач приводит к существенному снижению производительности компьютера.

В-третьих, ОЗУ компьютера ввиду специфики своей работы должно обладать повышенной надежностью, а возможности потери даже 1 бита данных должны быть исключены.

Рассмотрим характеристики оперативной памяти поподробнее:

– Быстродействие оперативной памяти определяется количеством времени, необходимым устройству для выполнения операции записи или считывания данных. Характеристика быстродействия памяти связана с двумя другими характеристиками – длительностью цикла обращения памяти и минимальным временем доступа.

– Длительность цикла определяется минимальным количеством времени, необходимым оперативной памяти на выполнение одной полной операции чтения или записи. Различают длительность цикла чтения и длительность цикла записи – эти характеристики могут в значительной степени различаться.

– Время доступа позволяет оценить временной промежуток между моментом начала цикла чтения и моментом, когда на выходе памяти появляются действительные данные, которые могут быть использованы другими устройствами.

– Характеристики длительности цикла и времени доступа не совпадают по причине того, что в полный цикл обращения к памяти помимо активной фазы самого доступа также входит фаза восстановления.

– Фаза восстановления необходима для возврата памяти к исходному состоянию. Без фазы восстановления следующий цикл обращения к памяти был бы не возможен по причине неготовности устройства.

– Производительность оперативной памяти определяется двумя важнейшими характеристиками: частотой системной шины и разрядностью шины памяти. Вместе частота системной шины и разрядность шины памяти позволяют судить о пропускной способности памяти. Разрядность шины памяти определяет количество бит информации, которые передаются по шине памяти за один такт операции чтения или записи. Современные модули оперативной памяти обычно имеют разрядность 64 бит и тактовую частоту порядка 1066-1866 МГц.

– Пропускная способность памяти позволяет судить о скорости записи и чтения данных. Для большинства пользователей эта характеристика значительно удобнее и понятнее, чем характеристики быстродействия памяти. Пропускная способность памяти измеряется в мегабайтах в секунду и вычисляется произведением частоты системной шины устройства и разрядности шины памяти. Типичная пропускная способность современных модулей оперативной памяти – 12-16 Гбайт/с.

– Емкость ЗУ характеризуют числом битов, либо байтов, которое одновременно может храниться в запоминающем устройстве. На практике применяются более крупные единицы, а для их обозначения к словам «бит» или «байт» добавляют приставки: кило, мега, гига, тера, пета, экса, зетта, йотта (kilo, mega, giga, tera, peta, exa, zetta, yotta).

5.2. Иерархическая структура памяти

Память часто называют «узким местом» фон-неймановских ВМ из-за ее серьезного отставания по быстродействию от процессоров, причем разрыв этот неуклонно увеличивается. Так, если производительность процессоров возрастает вдвое примерно каждые 1,5 года, то для микросхем памяти прирост быстродействия не превышает 9 % в год (удвоение за 10 лет), что выражается в увеличении разрыва в быстродействии между процессором и памятью приблизительно на 50 % в год.

Если проанализировать используемые в настоящее время типы ЗУ, выявляется следующая закономерность:

- чем меньше время выборки, тем выше стоимость хранения бита;

- чем больше емкость, тем ниже стоимость хранения бита, но больше время выборки.

При создании системы памяти постоянно приходится решать задачу обеспечения требуемой емкости и высокого быстродействия за приемлемую цену. Наиболее распространенным подходом здесь является построение системы памяти ВМ по иерархическому принципу. Иерархическая память состоит из ЗУ различных типов (рис. 18), которые, в зависимости от характеристик, относят к определенному уровню иерархии. Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость в пересчете на бит, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем нижележащем уровне и т. д.

Рисунок 18 – Иерархия запоминающих устройств

Три верхних уровня иерархии образуют внутреннюю память ВМ, а все нижние уровни – это внешняя или вторичная память (самый нижний уровень иерархии, представленный ЗУ со сменными носителями, благодаря чему обеспечивается практически неограниченная емкость, часто называют третичной памятью). По мере движения вниз по иерархической структуре:

1) уменьшается соотношение «стоимость/бит»;
2) возрастает емкость;
3) растет время выборки;

4) уменьшается частота обращения к памяти со стороны центрального процессора.

Если память организована в соответствии с пунктами 1–3, а характер размещения в ней данных удовлетворяет пункту 4, иерархическая организация ведет к уменьшению общей стоимости при заданном уровне производительности. Справедливость этого утверждения вытекает из принципа локальности по обращению. Если рассмотреть процесс выполнения большинства программ, то можно заметить, что с очень высокой вероятностью адрес очередной команды программы либо следует непосредственно за адресом, по которому была считана текущая команда, либо расположен вблизи него. Такое расположение адресов называется пространственной локальностью программы. Обрабатываемые данные, как правило, структурированы, и такие структуры обычно хранятся в последовательности смежных ячеек памяти. Данная особенность программ называется пространственной локальностью данных. Кроме того, программы содержат множество небольших циклов и подпрограмм. Это означает, что небольшие наборы команд могут многократно повторяться в течение некоторого интервала времени, то есть имеет место временная локальность. Все три вида локальности объединяет понятие локальность по обращению. Принцип локальности часто облекают в численную форму и представляют в виде так называемого правила «90/10»: 90 % времени работы программы связано с доступом к 10 % адресного пространства этой программы.

Из свойства локальности вытекает, что программу разумно представить в виде последовательно обрабатываемых фрагментов – компактных групп команд и обрабатываемых ими данных. Помещая такие фрагменты в более быструю память, можно существенно снизить общие задержки на обращение, поскольку команды и исходные данные, будучи один раз переданы из медленного ЗУ в быстрое, затем могут использоваться многократно, и среднее время доступа к ним в этом случае определяется уже более быстрым ЗУ. Это позволяет хранить большие программы и массивы данных на медленных, емких, но дешевых ЗУ, а в процессе обработки активно использовать сравнительно небольшую быструю память, увеличение емкости которой сопряжено с высокими затратами.

На каждом уровне иерархии информация (данные) разбивается на блоки, выступающие в качестве наименьшей информационной единицы, пересылаемой между двумя соседними уровнями иерархии. Размер блоков может быть фиксированным либо переменным. При фиксированном размере блока емкость памяти обычно кратна его размеру. Размер блоков на каждом уровне иерархии чаще всего различен и увеличивается от верхних уровней к нижним.

При доступе к командам и исходным данным, например, для их считывания, сначала производится поиск в памяти верхнего уровня. Факт обнаружения нужной информации называют попаданием (hit), в противном случае говорят о промахе (miss). При промахе производится поиск в ЗУ следующего более низкого уровня, где также возможны попадание или промах.

После обнаружения необходимой информации выполняется пересылка блока, содержащего искомую информацию, с нижних уровней на верхние.

При оценке эффективности подобной организации памяти обычно используют следующие характеристики:

- коэффициент попаданий (hit rate) – отношение числа обращений к памяти, при которых произошло попадание, к общему числу обращений к ЗУ данного уровня иерархии;
- коэффициент промахов (miss rate) – отношение числа обращений к памяти, при которых имел место промах, к общему числу обращений к ЗУ данного уровня иерархии;
- время обращения при попадании:
- (hit time) – время, необходимое для поиска нужной информации в памяти верхнего уровня (включая выяснение, является ли обращение попаданием), плюс время на фактическое считывание данных;
- потери на промах (miss penalty) – время, требуемое для замены блока в памяти более высокого уровня на блок с нужными данными, расположенный в ЗУ следующего (более низкого) уровня.

Потери на промах включают в себя: время доступа (access time) – время обращения к первому слову блока при промахе и время пересылки (transfer time) – дополнительное время для пересылки оставшихся слов блока. Время доступа обусловлено задержкой памяти более низкого уровня, в то время как время пересылки связано с полосой пропускания канала между ЗУ двух смежных уровней.

Самый быстрый, но и минимальный по емкости тип памяти – это внутренние регистры ЦП, которые иногда объединяют понятием сверхоперативное запоминающее устройство – СОЗУ или регистровый файл. Как правило, количество регистров невелико, хотя в архитектурах с сокращенным набором команд их число может доходить до нескольких сотен. Основная память (ОП), значительно большей емкости, располагается ниже. Между регистрами ЦП и основной памятью часто размещают кэш-память, которая по емкости ощутимо проигрывает ОП, но существенно превосходит последнюю по быстродействию, уступая в то же время СОЗУ. Все виды внутренней памяти реализуются на основе полупроводниковых технологий и в основном являются энергозависимыми.

Долговременное хранение больших объемов данных обеспечивается внешними ЗУ, среди которых наиболее распространены запоминающие устройства на основе магнитных и оптических дисков, а также магнитоленточные ЗУ. В последнее время все большую популярность получают твердотельные диски на базе флэш-памяти.

Еще один уровень иерархии может быть добавлен между основной памятью и магнитными дисками. Этот уровень носит название дисковой кэш-памяти и реализуется в виде самостоятельного ЗУ, включаемого в состав магнитного диска. Дисковая кэш-память существенно улучшает производительность при обмене информацией между дисками и основной памятью.

5.3. Основная оперативная память. Типы кэш-памяти

5.3.1. Назначение ОП. Основные понятия и определения

Основная память (ОП) представляет собой единственный вид памяти, к которой ЦП может обращаться непосредственно, (исключение составляют лишь регистры центрального процессора).

Информация, хранящаяся на внешних ЗУ, становится доступной процессору только после того, как будет переписана в основную память. Основную память образуют запоминающие устройства с произвольным доступом. Такие ЗУ образованы как массив ячеек, а «произвольный доступ» означает, что обращение к любой ячейке занимает одно и то же время и может производиться в произвольной последовательности. Каждая ячейка содержит фиксированное число запоминающих элементов и имеет уникальный адрес, позволяющий различать ячейки при обращении к ним для выполнения операций записи и считывания.

Основная память может включать в себя два типа устройств: оперативные запоминающие устройства (ОЗУ) и постоянные запоминающие устройства (ПЗУ).

Преимущественную долю основной памяти образует ОЗУ, называемое оперативным, потому что оно допускает как запись, так и считывание информации, причем обе операции выполняются однотипно, практически с одной и той же скоростью, и производятся с помощью электрических сигналов. В англоязычной литературе ОЗУ соответствует аббревиатура RAM – Random Access Memory, то есть «память с произвольным доступом», что не совсем корректно, поскольку памятью с произвольным доступом являются также ПЗУ и регистры процессора. Для большинства типов полупроводниковых ОЗУ характерна энергозависимость – даже при кратковременном прерывании питания хранимая информация теряется. Микросхема ОЗУ должна быть постоянно подключена к источнику питания и поэтому может использоваться только как временная память.

Вторую группу полупроводниковых ЗУ основной памяти образуют энергонезависимые микросхемы ПЗУ (ROM – Read-Only Memory). ПЗУ обеспечивает считывание информации, но не допускает ее изменения (в ряде случаев информация в ПЗУ может быть изменена, но этот процесс сильно отличается от считывания и требует значительно большего времени).

Емкость основной памяти современных ВМ слишком велика, чтобы ее можно было реализовать на базе единственной интегральной микросхемы (ИМС). Необходимость объединения нескольких ИМС ЗУ возникает также, когда разрядность ячеек в микросхеме ЗУ меньше разрядности слов ВМ.

Увеличение разрядности ЗУ реализуется за счет объединения адресных входов, объединяемых ИМС ЗУ. Информационные входы и выходы микросхем являются входами и выходами модуля ЗУ увеличенной разрядности (рис. 19). Полученную совокупность микросхем называют модулем памяти. Модулем можно считать и единственную микросхему, если она уже имеет нужную разрядность. Один или несколько модулей образуют банк памяти.

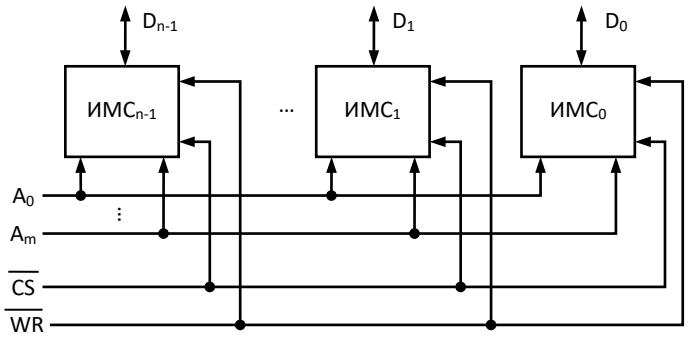

Рисунок 19 – Увеличение разрядности памяти

Для получения требуемой емкости ЗУ нужно определенным образом объединить несколько банков памяти меньшей емкости. В общем случае основная память ВМ практически всегда имеет блочную структуру, то есть содержит несколько банков.

5.3.2. Блочная организация основной памяти

При использовании блочной памяти, состоящей из B банков, адрес ячейки A преобразуется в пару (b, w), где – b номер банка, w – адрес ячейки внутри банка. Известны три схемы распределения разрядов адреса A между b и w:

- блочная (номер банка b определяет старшие разряды адреса);
- циклическая (b = A mod B; w = A div B);
- блочно-циклическая (комбинация двух предыдущих схем).

Рассмотрение основных структур блочной ОП будем проводить на примере памяти емкостью 512 слов (29), построенной из четырех банков по 128 слов в каждом. Типовая структура памяти, организованная в соответствии с блочной структурой, показана на рис. 20.

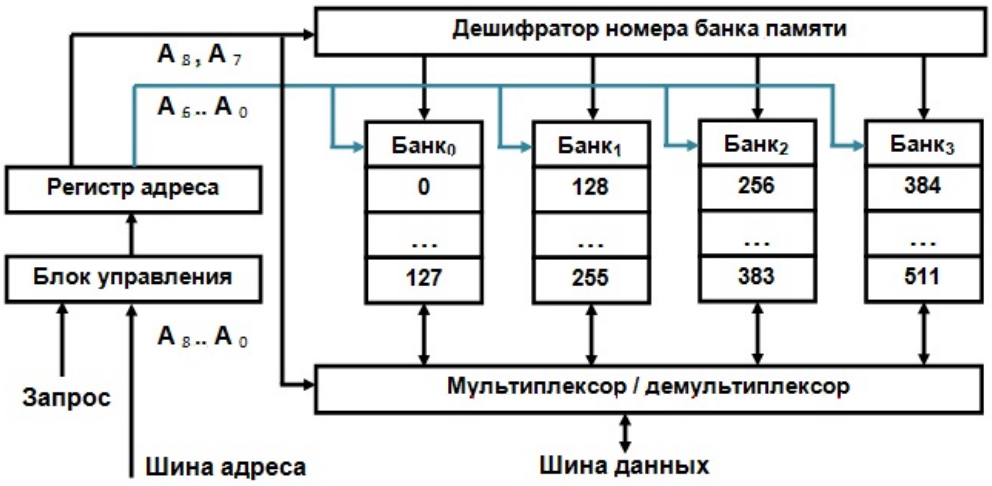

Рисунок 20 – Структура основной памяти на основе блочной схемы

Адресное пространство памяти разбито на группы последовательных

адресов, и каждая такая группа обеспечивается отдельным банком памяти. Для обращения к ОП используется 9-разрядный адрес, семь младших разрядов которого поступают параллельно на все банки памяти и выбирают в каждом из них одну ячейку. Два старших разряда адреса (А8, А7) содержат номер банка. Выбор банка обеспечивается либо с помощью дешифратора номера банка памяти, либо путем мультиплексирования информации (на рис. 19 показаны оба варианта). В функциональном отношении такая ОП может рассматриваться как единое ЗУ, емкость которого равна суммарной емкости составляющих, а быстродействие – быстродействию отдельного банка.

В блочно-циклической схеме расслоения памяти каждый банк состоит из нескольких модулей, адресуемых по круговой схеме. Адреса между банками распределены по блочной схеме. Таким образом, адрес ячейки разбивается на три части. Старшие биты определяют номер банка, следующая группа разрядов адреса указывает на ячейку в модуле, а младшие биты адреса выбирают модуль в банке. Схему иллюстрирует рис. 21.

Рисунок 21 – Блочно-циклическая схема расслоения ОП

Традиционные способы расслоения памяти хорошо работают в рамках одной задачи, для которой характерно свойство локальности. В многопроцессорных системах с общей памятью, где запросы на доступ к памяти достаточно независимы, не исключен иной подход, который можно рассматривать как развитие идеи расслоения памяти. Для этого в систему включают несколько контроллеров памяти, что позволяет отдельным банкам работать совершенно автономно. Эффективность данного приема зависит от частоты независимых обращений к разным банкам. Лучшего результата можно ожидать при большом числе банков, что уменьшает вероятность последовательных обращений к одному и тому же банку памяти.

5.3.3. Назначение кэш-памяти

Кэш-память представляет собой быстродействующее ЗУ, размещенное на одном кристалле с ЦП или внешнее по отношению к ЦП. Кэш служит высокоскоростным буфером между ЦП и относительно медленной основной памятью. Идея кэш-памяти основана на прогнозировании наиболее вероятных обращений ЦП к *оперативной памяти*. В основу такого подхода положен *принцип временной и пространственной локальности программы*.

Если ЦП обратился к какому-либо объекту *оперативной памяти*, с высокой долей вероятности ЦП вскоре снова обратится к этому объекту.

Примером этой ситуации может быть код или данные в циклах. Эта концепция описывается ***принципом временной локальности***, в соответствии с которым часто используемые объекты *оперативной памяти* должны быть «ближе» к ЦП (в кэше).

Почему возникла необходимость использования кэш-памяти?

Одним из самых важных вопросов при разработке компьютеров было и остается построение такой системы памяти, которая могла бы передавать операнды процессору с той же скоростью, с которой он их обрабатывает. Быстрый рост скорости работы процессора, к сожалению, не сопровождается столь же высоким ростом скорости работы памяти. Относительно процессора память работает все медленнее и медленнее с каждым десятилетием. С учетом огромной важности основной памяти эта ситуация сильно ограничивает развитие систем с высокой производительностью и направляет исследование таким путем, чтобы обойти проблему очень низкой по сравнению с процессором скорости работы памяти.

Современные процессоры предъявляют определенные требования к системе памяти и относительно времени ожидания (задержки в доставке операнда), и относительно пропускной способности (количества данных, передаваемых в единицу времени). К сожалению, эти два аспекта системы памяти сильно расходятся. Обычно с увеличением пропускной способности увеличивается время ожидания.

Один из способов решения этой проблемы – добавление кэш-памяти. Кэш-память содержит наиболее часто используемые слова, что повышает скорость доступа к ним. Если достаточно большой процент нужных слов находится в кэш-памяти, время ожидания может сильно сократиться.

Одной из самых эффективных технологий одновременного увеличения пропускной способности и уменьшения времени ожидания является применение нескольких блоков кэш-памяти. Между разделенной кэш-памятью (отдельной кэш-памяти для команд и отдельной для данных) и основной памятью часто помещается кэш-память второго уровня. На рис. 22 изображена система с тремя уровнями кэш-памяти. Прямо на микросхеме центрального процессора находится небольшая кэш-память для команд и небольшая кэш-память для данных, обычно от 16 до 64 Кбайт. Есть еще кэш-память второго уровня, которая расположена не на самой микросхеме процессора, а рядом с ним в том же блоке. Кэш-память второго уровня соединяется с процессором через высокоскоростной тракт данных. Эта кэш-память обычно не является разделенной и содержит смесь данных и команд. Ее размер – от 512 Кбайт до 1 Мбайт. Кэш-память третьего уровня находится на той же плате, что и процессор, и обычно состоит из статического ОЗУ в несколько мегабайтов, которое функционирует гораздо быстрее, чем динамическое ОЗУ основной памяти. Обычно все содержимое кэш-памяти первого уровня находится в кэш-памяти второго уровня, а все содержимое кэш-памяти второго уровня находится в кэш-памяти третьего уровня (см. рис. 22).

Рисунок 22 – Система с тремя уровнями кэш-памяти

Во всех типах кэш-памяти используется следующая модель. Основная память разделяется на блоки фиксированного размера, которые называются строками кэш-памяти. Строка кэш-памяти состоит из нескольких последовательных байтов (обычно от 4 до 64). Строки нумеруются, начиная с 0, то есть если размер строки составляет 32 байта, то строка 0 – это байты с 0 по 31, строка 1 – байты с 32 по 63 и т. д. В любой момент несколько строк находится в кэш-памяти. Когда происходит обращение к памяти, контроллер кэш-памяти проверяет, есть ли нужное слово в данный момент в кэш-памяти. Если есть, то можно сэкономить время, требуемое на доступ к основной памяти. Если данного слова в кэш-памяти нет, то какая-либо строка из нее удаляется, а вместо нее помещается нужная строка из основной памяти или из кэш-памяти более низкого уровня. Существует множество вариаций данной схемы, но в их основе всегда лежит идея держать в кэш-памяти как можно больше часто используемых строк, чтобы число успешных обращений к кэш-памяти было максимальным.

5.3.4. Выбор емкости кэш-памяти

Емкость кэш-памяти

Выбор емкости кэш-памяти – это всегда определенный компромисс. С одной стороны, кэш-память должна быть достаточно мала, чтобы ее стоимостные показатели были близки к величине, характерной для ОП. С другой – она должна быть достаточно большой, чтобы среднее время доступа в системе, состоящей из основной и кэш-памяти, определялось временем доступа к кэш-памяти.

Кроме того, чем вместительнее кэш-память, тем больше логических схем должно участвовать в ее адресации. Как следствие, микросхемы повышенной емкости работают медленнее по сравнению с микросхемами меньшей емкости,

даже если они выполнены по одной и той же технологии. Реальная эффективность использования кэш-памяти зависит от характера решаемых задач, и невозможно заранее определить, какая ее емкость будет действительно оптимальной. Рис. 23*а* иллюстрирует зависимость вероятности промахов от емкости кэш-памяти для трех программ A, B и C.

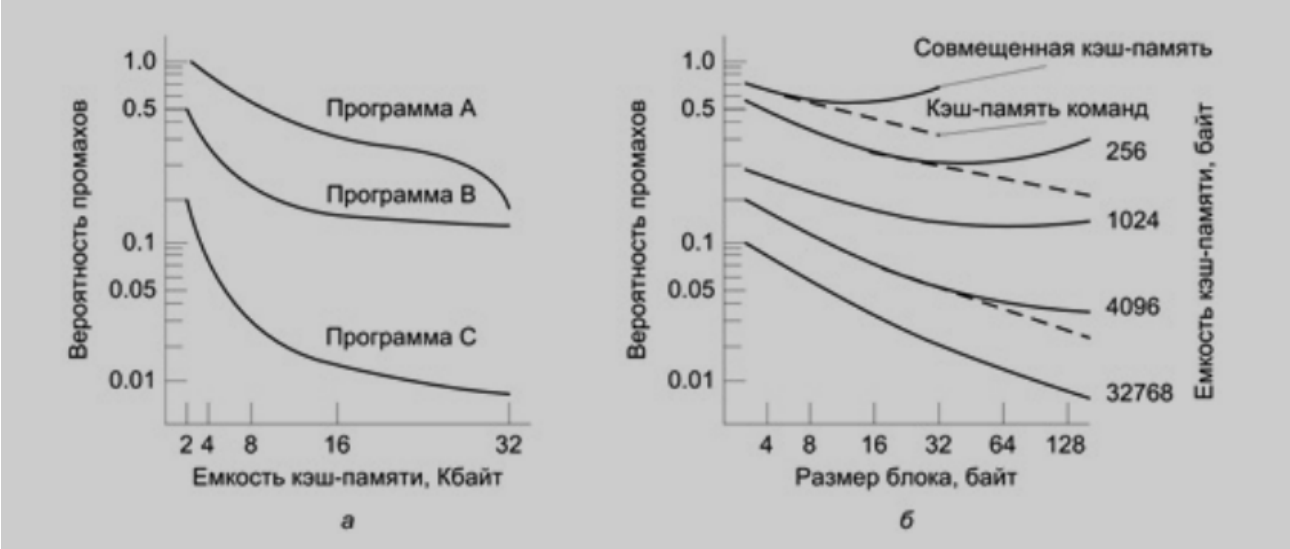

Рисунок 23 – Зависимость вероятности промахов от:
а – емкости кэш-памяти; *б* – размера блока

Несмотря на очевидные различия, просматривается и общая тенденция: по мере увеличения емкости кэш-памяти вероятность промахов сначала существенно снижается, но при достижении определенного значения эффект сглаживается и становится несущественным. В настоящее время считается, что для большинства задач близкой к оптимальной является емкость кэш-памяти от 1 до 512 Кбайт.

Размер блока кэш-памяти

Еще одним важным фактором, влияющим на эффективность использования кэш-памяти, служит размер блока. Когда в кэш-память помещается блок, вместе с требуемым словом туда попадают и соседние слова. По мере увеличения размера блока вероятность промахов сначала падает, так как в кэш, согласно принципу локальности, попадает все больше данных, которые понадобятся в ближайшее время. Однако, когда размер блока становится излишне большим, вероятность промахов начинает расти (рис. 23*б*). Объясняется это тем, что:

- большие размеры блока уменьшают общее количество блоков, которые можно загрузить в кэш-память, а малое число блоков приводит к необходимости частой их смены;
- по мере увеличения размера блока каждое дополнительное слово оказывается дальше от запрошенного, поэтому такое дополнительное слово менее вероятно понадобится в ближайшем будущем. Зависимость между размером блока и вероятностью промахов во многом

определяется характеристиками конкретной программы, из-за чего трудно рекомендовать определенное значение величины блока. Исследования показывают, что наиболее близким к оптимальному можно признать размер блока, равный 4–8 адресуемым единицам (словам или байтам). На практике его обычно выбирают сравнительно небольшим, равным ширине шины данных, связывающей кэш-память с основной памятью.

Первый уровень (L1) иерархии образует наиболее скоростная кэш-память сравнительно небольшой емкости (не более 128 Кбайт). На кристалле ее располагают по возможности ближе к процессору, чтобы минимизировать длину соединяющей их шины и тем самым способствовать ускорению обмена информацией. Этот уровень в перспективных микропроцессорах обычно строится по разделенной схеме.

Для новейших многоядерных микропроцессоров типично размещение на том же кристалле еще и кэш-памяти третьего уровня (L3). Обычно этот уровень является общим для всех ядер (или группы ядер).

5.4. Обнаружение и исправление ошибок при обращении к ОП

5.4.1. Математический аппарат кода Хэмминга

Память компьютера из-за всплесков напряжения и по другим причинам время от времени может ошибаться. Чтобы бороться с ошибками, используются специальные коды, умеющие обнаруживать и исправлять ошибки. В этом случае к каждому слову в памяти особым образом добавляются дополнительные биты. Когда слово считывается из памяти, эти дополнительные биты проверяются, что и позволяет обнаруживать ошибки.

Чтобы понять, как обращаться с ошибками, необходимо внимательно изучить, что представляют собой эти ошибки. Предположим, что слово состоит из **m** бит данных, к которым мы дополнительно прибавляем **k** бит (контрольных разрядов). Пусть общая длина слова составит **n** бит (то есть n = m + k). **Единицу из n бит, содержащую m бит данных и k контрольных разрядов, часто называют кодовым словом.**

Для любых двух кодовых слов, например 10001001 и 10110001, можно определить, сколько соответствующих битов в них различаются. В данном примере таких бита три. Чтобы определить количество различающихся битов, нужно над двумя кодовыми словами произвести логическую операцию ИСКЛЮЧАЮЩЕЕ ИЛИ и сосчитать количество битов со значением 1 в полученном результате. Число битовых позиций, по которым различаются два слова, называется интервалом Хэмминга. Если интервал Хэмминга для двух слов равен **d**, то значит достаточно **d** битовых ошибок, чтобы превратить одно слово в другое. Например, интервал Хэмминга для кодовых слов 11110001 и 00110000 равен 3, поскольку для превращения первого слова во второе достаточно 3 битовые ошибки.

Память состоит из **m**-разрядных слов, следовательно, существуют $\mathbf{2^m}$

вариантов сочетания битов. Кодовые слова состоят из **n** бит, но из-за способа подсчета контрольных разрядов допустимы только 2^m **из** 2^n кодовых слов. Если в памяти обнаруживается недопустимое кодовое слово, компьютер знает, что произошла ошибка. При наличии алгоритма подсчета контрольных разрядов можно составить полный список допустимых кодовых слов, и из этого списка найти два слова, для которых интервал Хэмминга будет минимальным. Это интервал Хэмминга для полного кода.

Возможности проверки и исправления ошибок определенного кода зависят от его интервала Хэмминга. Чтобы обнаружить **d** ошибок в битах, необходим код с интервалом $d + 1$, поскольку d ошибок не могут превратить одно допустимое кодовое слово в другое допустимое кодовое слово. Соответственно, чтобы исправить d ошибок в битах, необходим код с интервалом $2d + 1$, поскольку в этом случае допустимые кодовые слова настолько сильно отличаются друг от друга, что, даже если произойдет d изменений, изначальное кодовое слово окажется ближе к ошибочному, чем любое другое кодовое слово, поэтому его без труда можно будет выявить.

5.4.2. Правила построения и использования кода Хэмминга

Правила построения и использования кода Хэмминга рассмотрим на примере, для случая:

1. Общее количество разрядов кодового слова **n = 15**.
2. Количество контрольных разрядов $\mathbf{k = \log_2(15 + 1) = 4.}$
3. Количество информационных разрядов **m = n-k = 11.**

Первым этапом построения кода Хэмминга является разбиение разрядов кодового слова на взаимно пересекающиеся подмножества. За каждым подмножеством закрепляется контрольный разряд четности, дополняющий до четного количество единиц в разрядах кодового слова, включенных в данное подмножество.

Контрольные разряды имеют номера 2^{S_i}, где S_i **= 0, 1, 2,..., [(n-m)-1]**. В рассматриваемом примере контрольные разряды имеют номера: 1, 2, 4, 8, т. е. располагаются в 1, 2, 4 и 8 разрядах кодового слова при нумерации его разрядов слева направо: 1, 2, 3, ..., 15 (рис. 24).

Формирование контролируемых подмножеств проведем на основании анализа номеров разрядов кодового слова при записи их в двоичной системе счисления (рис. 24).

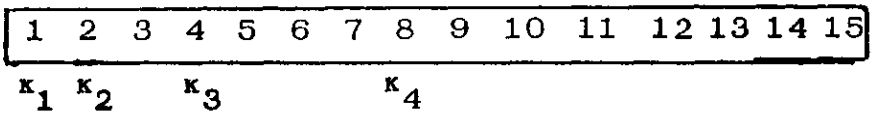

Рисунок 24 – Формирование контролируемых подмножеств

Все разряды кодового слова, имеющие единицу в первом разряде своего номера, включаются в подмножество E_1, во втором – в подмножество E_2 и т. д.

В таблице 2 показано закрепление разрядов кодового слова за подмножествами.

Таблица 2 – Закрепление разрядов кодового слова за подмножествами

Номера разрядов кодового слова		Подмножества по Хэммингу			
Десятичный номер	Двоичный номер	E_4	E_3	E_2	E_1
1 ($к_1$)	0001				+
2 ($к_2$)	0010			+	
3	0011			+	+
4 ($к_3$)	0100		+		
5	0101		+		+
6	0110		+	+	
7	0111		+	+	+
8 ($к_4$)	1000	+			
9	1001	+			+
10	1010	+		+	
11	1011	+		+	+
12	1100	+	+		
13	1101	+	+		+
14	1110	+	+	+	
15	1111	+	+	+	+

Для первого подмножества контрольным разрядом является первый разряд кодового слова – $к_1$, для второго подмножества – второй разряд кодового слова – $к_2$, для третьего – четвертый разряд $к_3$ и для E_4 – восьмой разряд $к_4$.

Таким образом, в подмножества по Хэммингу входят следующие разряды кодового слова:

E_1 = {1, 3, 5, 7, 9, 11, 13, 15}
E_2 = {2, 3, 6, 7, 10, 11, 14, 15}
E_3 = {4, 5, 6, 7, 12, 13, 14, 15}
E_4 = {8, 9, 10, 11, 12, 13, 14, 15}

Из приведенной схемы видно, что контрольные разряды (подчеркнуты) входят **только в контролируемые ими подмножества и начинают эти подмножества**. Другие разряды могут входить в несколько подмножеств, (например 15 – во все четыре).

При записи числа в коде Хэмминга контрольные разряды заполняются таким образом, чтобы сумма единиц по **mod2** информационных разрядов каждого подмножества E_S и его контрольного разряда равнялась нулю.

При схемной реализации контроля по коду Хэмминга устройство контроля должно содержать следующие основные узлы:

приемный регистр на **n = m + k** разрядов;

- схемы сверток по **mod2** для каждого из подмножеств с целью

образования контрольных кодов (кодирующие устройства);
- схема свертки по **mod2,** охватывающая все **n** разрядов, для обнаружения двойной ошибки;
- схемы сравнения контрольных разрядов, получаемых при записи и после считывания числа из ОП, для образования синдромов ошибок;
- схемы обнаружения места одиночной ошибки и ее инвертирования.

Перед записью информации в ОП кодирующее устройство определяет значения контрольных разрядов и записывает их в соответствующие биты слова.

Пример 1. Пусть значения информационных разрядов передаваемого слова А равны: **А = 11101111010.**

1. Запишем число, пропуская позиции контрольных разрядов (таб. 3).

Таблица 3 – Информационные разряды числа А

Номера разрядов	1 $к_1$	2 $к_2$	3	4 $к_3$	5	6	7	8 $к_4$	9	10	11	12	13	14	15
Код числа А			1		1	1	0		1	1	1	1	0	1	0

2. Вычислим первый контрольный разряд. В первое подмножество входят разряды – 3, 5, 7, 9, 11, 13, 15. 3, 5, 9 и 11 разряды равны единицы, а остальные равны нулю. Следовательно, к 1 = 0. Аналогично получаем значения остальных контрольных разрядов. С учетом контрольных разрядов код числа А будет иметь вид, приведенный в таблице 4.

Таблица 4 – Кодовое число

Номера разрядов	1 $к_1$	2 $к_2$	3	4 $к_3$	5	6	7	8 $к_4$	9	10	11	12	13	14	15
Код числа А	0	1	1	0	1	1	0	1	1	1	1	1	0	1	0

Контрольные разряды $к_1$, $к_2$, $к_3$, $к_4$ записаны в 1, 2, 4, 8 разряды кодового слова А. Цифры в контрольных разрядах по каждому подмножеству $Е_1$, $Е_2$, $Е_3$, $Е_4$ дополняют до нечетного значения число единиц в соответствующем подмножестве. Число А вместе с контрольными разрядами записывается в оперативную память.

Пример 2. Пусть значения информационных разрядов передаваемого числа А = 10101010101.

С учетом контрольных разрядов код числа А будет иметь вид,

приведенный в таблице 4.

В процессе хранения или считывания этого числа из ОП может произойти искажение информации. Теория и практика показывают, что наиболее вероятны одиночные ошибки.

После считывания числа А из ОП вновь выделяются контрольные разряды $к'_1$, $к'_2$, $к'_3$, $к'_4$ и поразрядно сравниваются с первичными контрольными разрядами $к_1$, $к_2$, $к_3$, $к_4$, выделенными перед записью числа А в ОП.

Таблица 5 – Кодовое число

Номера разрядов	1 $к_1$	2 $к_2$	3	4 $к_3$	5	6	7	8 $к_4$	9	10	11	12	13	14	15
Код числа А	0	1	1	0	1	1	0	1	1	1	1	1	0	1	0

Если сравниваемые контрольные разряды равны, ошибка отсутствует. В противном случае – в считанной информации имеется ошибка. Одиночные ошибки могут быть исправлены путем определения разряда числа, содержащего ошибку и инверсии его содержимого.

Пример 3. Введем ошибку в 13 разряд числа А путем замены 0 на 1. Код считанного числа А* с ошибкой приведен в таблице 6.

Таблица 6 – Ввод ошибки в 13 разряд кодового числа.

Номера разрядов	1 $к_1$	2 $к_2$	3	4 $к_3$	5	6	7	8 $к_4$	9	10	11	12	13	14	15
Код числа А’	0	1	1	0	1	1	0	1	1	1	1	1	1	1	0

Вычислим значения контрольных разрядов для считанного числа А с использованием подмножеств по Хэммингу.

Отметим, что 13-й разряд входит только в подмножества E_1, E_3, E_4, следовательно, произойдет инверсия контрольных разрядов (определенных для числа А) только в данных подмножествах. Получим: $к'_1 = 1$, $к'_2 = 1$, $к'_3 = 1$, $к'_4 = 0$.

Сложим по **mod2** одноименные контрольные разряды кодового слова (числа А) и числа А’. В результате получим номер разряда кодового слова (числа А'), содержащего ошибку в 13 разряде ($D_{16} = 13_{10}$) (рис. 25).

$к_4 = 1, к_3 = 0, к_2 = 1, к_1 = 0;$

контрольные разряды числа A':

$к_4 = 0, к_3 = 1, к_2 = 1, к_1 = 1$

Номер разряда числа A', содержащего ошибку:

$1 \quad 1 \quad 0 \quad 1 = 13_{10}$

Пример 2. Ошибка в 5 разряде, инверсия $к_1$ и $к_3$.

A: $к_4 = 1, к_3 = 0, к_2 = 1, к_1 = 0,$

A': $к_4 = 1, к_3 = 1, к_2 = 1, к_1 = 1$

Номер разряда числа A', содержащего ошибку

$0 \quad 1 \quad 0 \quad 1 = 5_{10}$

Рисунок 25 – Поиск одиночной ошибки кодового слова

А теперь посмотрим, как может использоваться алгоритм Хэмминга при создании кодов исправления ошибок для слов любого размера. В коде Хэмминга, к слову, состоящему из m бит, добавляются k бит четности, при этом образуется слово длиной m + k бит. Биты нумеруются с единицы (а не с нуля), причем первым считается крайний левый. Все биты, номера которых – степени двойки, являются битами четности (контрольными); остальные используются для данных. Например, к 16-разрядному слову нужно добавить 5 бит четности. Биты с номерами 1, 2, 4, 8 и 16 – биты четности, все остальные – биты данных. Всего слово содержит 21 бит (16 бит данных и 5 бит четности). В рассматриваемом примере мы будем использовать проверку на четность.

Каждый бит четности позволяет проверять определенные битовые позиции. Общее число битов со значением 1 в проверяемых позициях должно быть четным. Ниже указаны позиции проверки для каждого бита четности:

бит 1 проверяет биты 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21;

бит 2 проверяет биты 2, 3, 6, 7, 10, 11, 14, 15, 18, 19;

бит 4 проверяет биты 4, 5, 6, 7, 12, 13, 14, 15, 20, 21;

бит 8 проверяет биты 8, 9, 10, 11, 12, 13, 14, 15;

бит 16 проверяет биты 16, 17, 18, 19, 20, 21.

В общем случае бит b проверяется битами b_1 b_2 b_3.., b_j такими, что $b_1 + b_2 + ... + b_j = b$. Например, бит 5 проверяется битами 1 и 4, поскольку 1 + 4 = 5. Бит 6 проверяется битами 2 и 4, поскольку 2 + 4 = 6 и т. д.

Рис. 26 иллюстрирует построение кода Хэмминга для 16-разрядного слова 1111000010101110.

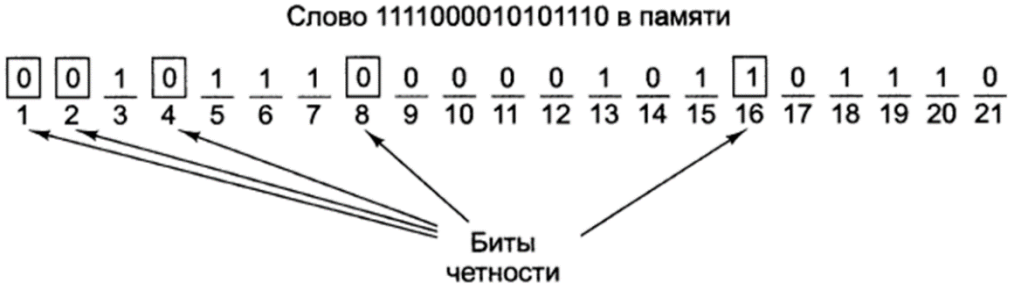

Рисунок 26 – Построение кода Хэмминга для слова 11110000010101110 добавлением к битам данных пяти контрольных разрядов

Соответствующим 21-разрядным кодовым словом является 001011100000101101110. Чтобы понять, как происходит исправление ошибок, рассмотрим, что произойдет, если бит 5 изменит значение (например, из-за резкого скачка напряжения). В результате вместо кодового слова 001011100000101101110 получится 001001100000101101110. Будут проверены 5 бит четности. Вот результаты:

- неправильный бит четности 1 (биты 1, 3, 5, 7, 9, И, 13, 15, 17, 19, 21 содержат пять единиц);
- правильный бит четности 2 (биты 2, 3, 6, 7. 10, 11, 14, 15, 18, 19 содержат шесть единиц);
- неправильный бит четности 4 (биты 4, 5, 6, 7, 12, 13, 14, 15, 20, 21 содержат пять единиц);
- правильный бит четности 8 (биты 8, 9, 10, 11, 12, 13, 14, 15 содержат две единицы);
- правильный бит четности 16 (биты 16, 17, 18, 19, 20, 21 содержат четыре единицы).

Общее число единиц в битах 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 и 21 должно быть четным, поскольку в данном случае используется проверка на четность. Неправильным должен быть один из битов, проверяемых битом четности 1 (а именно 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 и 21). Бит четности 4 тоже неправильный. Это значит, что изменил значение один из следующих битов: 4, 5, 6, 7, 12, 13, 14, 15, 20, 21. Ошибка должна быть в бите, который содержится в обоих списках. В данном случае общими являются биты 5, 7, 13, 15 и 21. Поскольку бит четности 2 правильный, биты 7 и 15 исключаются. Правильность бита четности 8 исключает наличие ошибки в бите 13. Наконец, бит 21 также исключается, поскольку бит четности 16 правильный. В итоге остается бит 5, в котором и кроется ошибка. Поскольку этот бит имеет значение 1, он должен принять значение 0. Именно таким образом исправляются ошибки.

Чтобы найти неправильный бит, сначала нужно определить все биты четности. Если они правильные, ошибки нет (или есть, но ошибка не однократная). Если обнаружились неправильные биты четности, нужно сложить их номера. Сумма, полученная в результате, даст номер позиции неправильного бита. Например, если биты четности 1 и 4 неправильные, а 2, 8 и 16 правильные, то ошибка произошла в бите 5 (1 + 4).

6. СИСТЕМЫ ВВОДА-ВЫВОДА И ОРГАНИЗАЦИЯ ПРЕРЫВАНИЙ

6.1. Организация ввода-вывода в компьютерной системе

Принципы функционирования рассмотрим на примере микропроцессорной системы.

В МПС применяются три режима ввода/вывода:

- программно-управляемый ввод/вывод;
- ввод/вывод в режиме прерываний;
- ввод/вывод в режиме прямого доступа к памяти.

Первый из них характеризуется тем, что инициирование и управление осуществляется программой, выполняемой процессором, а внешние устройства играют сравнительно пассивную роль и только сигнализируют о своем состоянии, в частности, о готовности к операциям ввода/вывода.

Во втором режиме ввод-вывод инициируется не процессором, а внешним устройством, генерирующим специальный сигнал прерывания. Реагируя на этот сигнал готовности устройства к передаче данных, процессор передает управление подпрограмме обслуживания устройства, вызвавшего прерывание. Действия, выполняемые этой подпрограммой, определяются пользователем, а непосредственными операциями ввода/вывода управляет процессор.

Наконец, в режиме прямого доступа к памяти, который используется, когда пропускной способности процессора недостаточно, процессор приостанавливается, он отключается от системной шины и не участвует в передаче данных между основной памятью и быстродействующим ВУ.

6.2. Программно-управляемый ввод/вывод

Данный режим характеризуется тем, что все действия по вводу/выводу реализуются командами прикладной программы через порт (рис. 27).

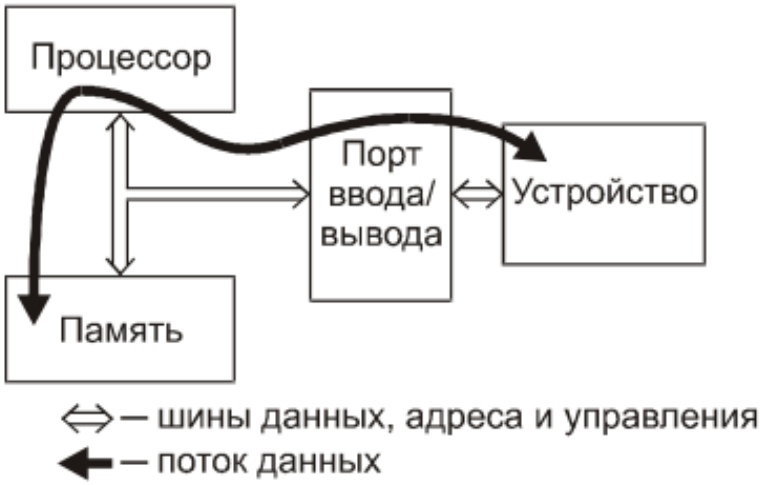

Рисунок 27 – Аппаратная реализация программно-управляемого ввода-вывода

Наиболее простыми эти действия оказываются для «всегда готовых» внешних устройств, например, индикатора на светодиодах. При необходимости ввода/вывода в соответствующем месте программы используются команды MOV (A, P_0) или MOV (P_0, A). Такая передача данных называется синхронным вводом/выводом (рис. 28*а*).

Однако для большинства ВУ до выполнения операций ввода/вывода необходимо убедиться в их готовности к обмену (рис. 28*б*). Такой режим ввода/вывода называют асинхронным.

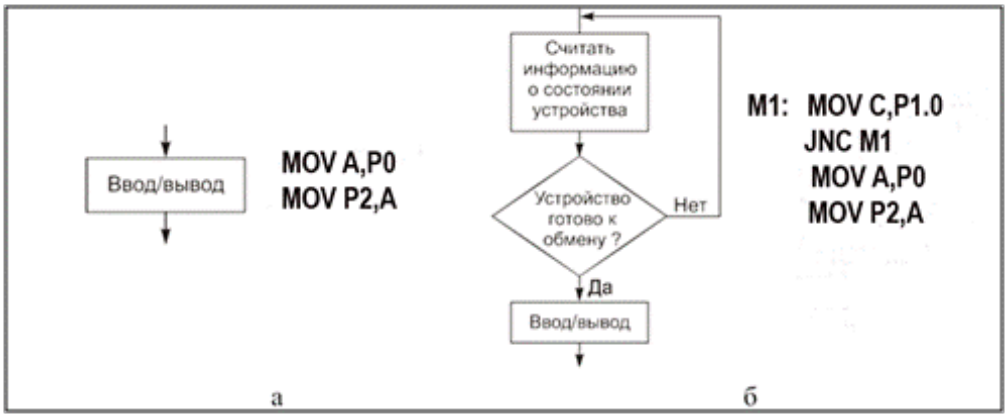

Рисунок 28 – Выполнение операций ввода/вывода:
а – синхронный; *б* – асинхронный

Общее состояние устройства характеризуется флагом статуса С, называемым также флагом готовности/занятости. Процессор проверяет готовность устройства с помощью команды JNC, выставляет флаги командой MOV (C, P0.1) и осуществляет проверку флагов командой JNC. Если флаг С установлен, то инициируются собственно ввод или вывод данных из порта P_0.

Когда же флаг сброшен, процессор выполняет цикл из 2–3 команд с повторной проверкой флага С до тех пор, пока устройство не будет готово к операциям ВВ. Данный цикл называется циклом ожидания готовности ВУ.

Основной недостаток программного ввода-вывода связан с непроизводительными потерями времени процессора в циклах ожидания. К достоинствам следует отнести простоту его реализации, не требующей дополнительных аппаратных средств.

6.3. Ввод/вывод в режиме прерываний

При организации в МПС системы прерываний непроизводительные потери времени процессора в циклах ожидания готовности резко сокращаются. Режим с прерыванием программы позволяет организовать обмен данными с внешними устройствами в произвольные моменты времени, не зависящие от программы (определяемые внешними устройствами).

Главное отличие ввода/вывода в режиме с прерыванием от программного

ввода/вывода состоит в том, что:

- инициатором является внешний сигнал;
- заранее не известно, в какой момент будет прервана фоновая программа.

При этом каждое периферийное устройство может посылать в процессор сигнал запроса прерывания, когда оно готово к операциям ввода/вывода. Сигнал появляется в произвольные моменты времени, асинхронно по отношению к действиям процессора, и управлять его появлением программа не может. Следовательно, заранее не известно, в какой точке программы и какие периферийные устройства инициируют прерывания, поэтому непосредственно в программе команды ввода/вывода использовать нельзя. Остается одно: реагируя на сигнал INT, процессор должен прервать, т. е. временно приостановить текущую программу, идентифицировать прерывающее устройство, перейти к подпрограмме обслуживания прерываний работы этого устройства, а после ее завершения возобновить выполнение прерванной программы. Подпрограмме обслуживания потребуются внутренние регистры процессора: аккумулятор, программный счетчик, некоторые РОН, и их текущее содержимое будет модифицировано. Но прерванная программа должна возобновиться так, как будто прерывания вообще не было (рис. 29).

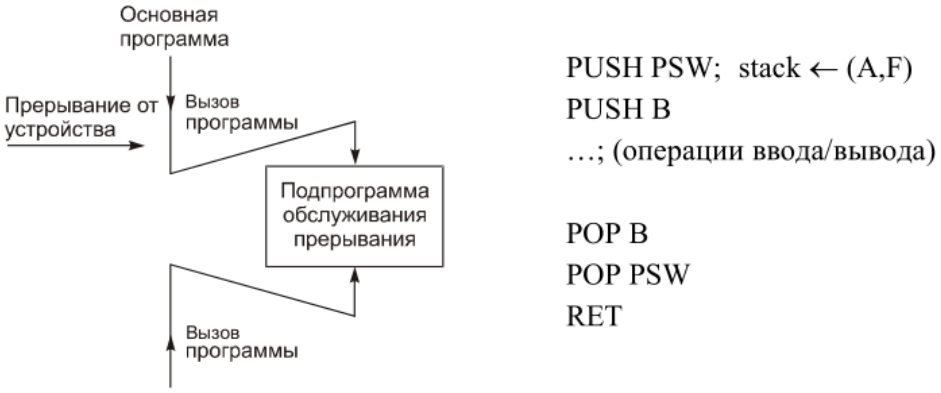

Рисунок 29 – Обслуживание прерывания

Факт обслуживания прерывания влияет на прерванную программу только увеличением времени ее выполнения. Таким образом, содержимое всех регистров, необходимых подпрограмме обслуживания прерывания, следует временно запоминать. В качестве такого временного «хранилища» удобно использовать стек (однако, чем больше информации запоминается, тем больше время реакции на прерывание).

Последовательность обработки прерывания примерно одинакова для всех микропроцессоров и содержит следующие шаги:

1. Периферийное устройство генерирует сигнал прерывания, который подается на вход INT процессора; на этой линии по схеме ИЛИ объединяются запросы всех устройств, работающих в режиме прерываний.

2. Процессор завершает текущую команду и, если прерывания разрешены (не замаскированы), формирует сигнал подтверждения прерывания

INTA.

3. Запоминается содержимое счетчика команд PC и некоторых других внутренних регистров в стеке.

4. Процессор идентифицирует прерывающее устройство для перехода к соответствующей подпрограмме обслуживания.

5. Выполняется короткая (30–50 байт) подпрограмма обслуживания прерывания, в которой запрограммированы действия по передаче данных, модификации указателей, проверке окончания операций ввода/вывода.

6. Восстанавливается состояние прерванной программы, для чего сохраненное содержимое регистров извлекается из стека.

7. Возобновляется выполнение прерванной программы; это действие инициируется командой возврата из прерывания RET, являющейся последней командой подпрограммы обслуживания прерывания.

6.4. Ввод/вывод в режиме прямого доступа к памяти

Этот метод используется для скоростных внешних запоминающих устройств, например, в накопителях на жестких магнитных дисках. В них обмен производится блоками фиксированного размера от 128 байт и более, причем этот обмен должен осуществляться в строгой последовательности без пропусков, так как пропуск хотя бы одного байта вызовет необходимость в повторном обмене. При этом время, отводимое на обмен одним байтом, строго ограничено скоростью перемещения магнитного носителя относительно магнитных головок и не превышает, как правило, нескольких микросекунд.

Обеспечить обмен большими блоками данных с ВЗУ как при помощи программно-управляемого обмена, так и в режиме прерываний процессора невозможно, так как на обмен каждым байтом затрачивается несколько команд, суммарное время выполнения которых превышает допустимое время на обмен одним байтом с ВЗУ.

Поэтому высокоскоростной обмен производится в режиме прямого доступа к памяти (DMA) без участия процессора. При этом ряд функций реализуется аппаратно, а время обмена одним байтом данных равно одному циклу обращения к памяти. Причем обменом управляет не программа, а специальный контроллер прямого доступа к памяти.

Существует две разновидности DMA. В режиме «прозрачного DMA» передача данных выполняется без информирования процессора, для чего используются те интервалы машинных циклов, когда процессор не обращается к памяти, а выполняет внутренние преобразования данных.

Процессор идентифицирует эти интервалы специальным сигналом, означающим доступность системной шины. Производительность процессора в таком режиме не уменьшается, но сами передачи носят нерегулярный характер, что ведет к уменьшению скорости передачи данных.

Другим способом является «DMA с приостановкой процессора», при котором выполнение команд задерживается на несколько тактов. Как показано

на рис. 30, контроллер непосредственно связан с памятью МПС через шины данных и адреса. При этом возникает проблема совместного использования шин процессором и контроллером DMA. Для исключения конкуренции двух устройств за право владения шиной контроллер DMA снабжен двумя управляющими линиями: «Захват шин» (HOLD) и «Подтверждение захвата» (HLDA).

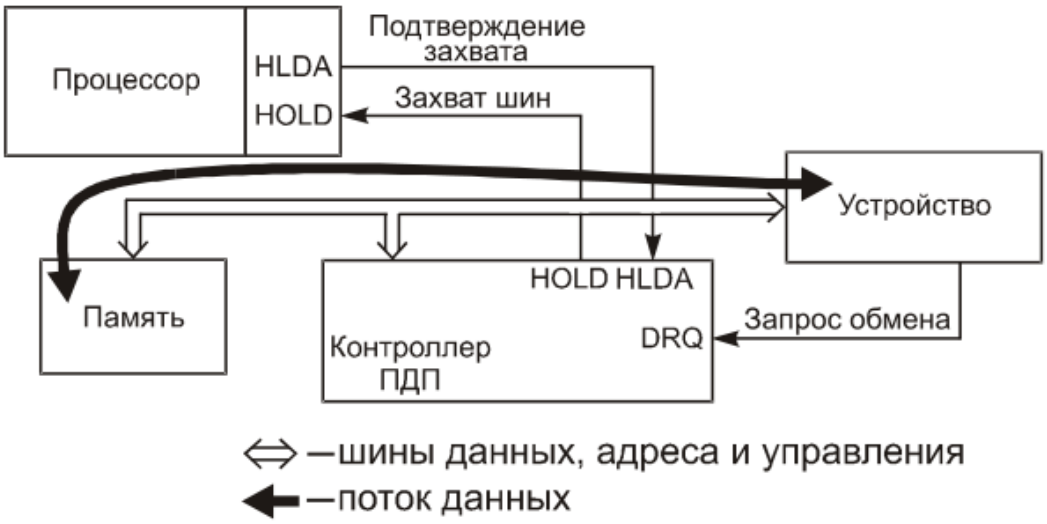

Рисунок 30 – Организация ввода/вывода в режиме DMA

Если требуется обмен данными, то периферийное устройство адресует запрос обмена (сигнал DRQ) контроллеру DMA. После получения от устройства такого запроса контроллер посылает на вход HOLD микропроцессора сигнал «Захват шин». Процессор, получив этот сигнал, приостанавливает выполнение очередной команды (закончив очередной машинный цикл), отключается от шин системного интерфейса (путем перехода шин в состояние высокого сопротивления) и выдает на системный интерфейс управляющий сигнал «Подтверждение захвата». С этого момента все шины управляются контроллером DMA, который осуществляет обмен побайтно или блоками данных с памятью МПС и затем, сняв сигнал захвата, контроллер возвращает управление системным интерфейсом процессору. Как только контроллер DMA будет готов к обмену следующим блоком данных, он вновь «захватывает» цикл процессора и т. д.

В промежутках между сигналами подтверждения процессор может продолжать выполнение команд программы. Тем самым выполнение программы замедляется, но в меньшей степени, чем при обмене в режиме программного ввода/вывода.

Применение МПС в режиме DMA всегда требует предварительной подготовки, а именно: для каждого ВУ необходимо выделить область памяти, используемую при обмене, и указать ее размер, т. е. число данных, подлежащих пересылке. Следовательно, контроллер DMA должен обязательно иметь в своем составе регистр адреса и счетчик байт. Перед началом обмена в режиме DMA необходимо выполнить подготовительные операции по записи в указанные регистры контроллера DMA начального адреса выделенной внешнему устройству памяти и ее объема в байтах.

ЗАКЛЮЧЕНИЕ

В данном курсе лекций были рассмотрены общие вопросы организации компьютерных систем, особое внимание было уделено системам памяти. Были также рассмотрены вопросы общей организации КС и системы ввода-вывода.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Брайант, Р. Э. Компьютерные системы. Архитектура и программирование / Р. Э. Брайант, Д. Р. О'Халларон ; перевод с английского А. Н. Киселева. – 3-е изд. – Москва : ДМК Пресс, 2022. – 994 с. – Текст: непосредственный.

2. Архитектура вычислительных систем: учебное пособие / С. В. Грейбо, Т. Е. Новосёлова, Н. Н. Пронькин, И. Ф. Семёнычева. – М., 2019. – URL: http://scipro.ru/conf/computerarchitecture.pdf. – Текст: электронный.